Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мультимедійних систем

КУРСОВА РОБОТА

за спеціальністю «Інженерія програмного забезпечення»

121

на тему:

Маркетплейс для картинних галерей, які зберігають у вигляді NFT токенів

Науковий керівник:

Гороховський К. С.

Виконала студентка 4-го курсу

Синельник М.С.

Київ – 2022

Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

Жежерун Олександр Петрович

“_”_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентки Синельник Марії 4-го курсу факультету інформатики ТЕМА:

Маркетплейс для картинних галерей, які зберігають у вигляді NFT токенів

Зміст ТЧ до курсової роботи:

Індивідуальнезавдання

Календарний план

Вступ

1. Аналіз предметної області
2. Теоретичні відомості
3. Опис реалізації програмного продукту

Висновки

Перелік використаних джерел

Додатки

Дата видачі “____”____2022 р.

Керівник Гороховський К.С. Завдання отримано ___

Календарний план виконання курсової роботи

Тема: Маркетплейс для картинних галерей, які зберігають у вигляді NFT токенів

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу		
2.	Знайомство з написаннм смарт контрактів за допомогою Solidity		
3.	Знайомство з блокчейном Polygon		
4.	Знайомство з NFT		
5.	Огляд аналогів розробки		
6.	Розробка смарт контрактів		
7.	Тестування смарт контрактів у блокчейні		
8.	Написання фронтенду		
9.	Початок написання текстової частини		
10.	Надання проміжної версії практичної частини		
11.	Аналіз практичної частини, її корегування		
12.	Остаточне завершення розробки dapp		
13.	Остаточне завершення написання текстової частини		
14.	Створення презентації		
15.	Захист курсової роботи		

Студентка Синельник М.С.

Керівник Гороховський К.С.

“___” ___ 2022р.

Зміст

Анотація

Робота присвячена створенню маркетплейсу для розміщення картин молодих художників та продажу їх на аукціонах за криптовалюту. Проєкт реалізований за допомогою Dapp (децентралізованих застосунків), які працюють у мережі блокчейн. Рівень клієнта реалізований за допомогою React Native та відображений у веб-системі у HTML форматі, а серверний рівень побудований за допомогою технології блокчейн Avalanche, де функціональність розроблена за допомогою смарт контрактів, написаних на Solidity.

Ключові слова

- *Blockchain – тип спільної бази даних, яка відрізняється від типової бази даних способом зберігання інформації; блокчейни зберігають дані в блоках, які потім з'єднуються разом за допомогою криптографії [1]*
- *Dapp (decentralized application) – цифрові додатки або програми, які існують і працюють на блокчейні або одноранговіq (P2P) мережі комп'ютерів замість одного комп'ютера [2]*
- *P2P (Peer-to-Peer) – платформа, яка безпосередньо підключає сторони до транзакції без стороннього посередника [3]*
- *SC (smart contract) – програмний код, що виконується самостійно, з умовами угоди між покупцем і продавцем, які автоматично перевіряються та виконуються через комп'ютерну мережу [4]*
- *NFT (non-fungible token) – унікальний криптографічний токени, який існує в блокчейні і не може бути відтвореним [5]*
- *Marketplace (маркетплейс) – онлайн платформа, яка надає послугу або продукти*
- *Avalanche - це платформа для криптовалюти та блокчейну, яка являється одним з конкурентівEthereum. [6]*
- *Web3 (Web 3.) – новітня ідею Інтернету, яка буде базуватися повністю на технології блокчейн*
- *KYC (знати свого клієнта) – термін банківського та біржового регулювання для фінансових інститутів та букмекерських контор*

Вступ

У динамічному сучасному світі все більше й більше стає популярним слово *блокчейн* і розробка застосунків, які використовують технологію блокчейн. Децентралізовані застосунки(Dapp) в найближчому майбутньому повністю замінять централізовані, через низький рівень вразливості та прозорість доступної інформації розміщеної на сайтах, що з одним з ключових проривів у *Web3* – новій формі World Wide Web. Ще одним популярним словом минулого року стало *NFT*. Невзаємозамінні токени дозволяють зберігати будь-яку інформацію доступну в інтернеті у вигляді токенів використовуючи технологію блокчейн, що дає змогу перегляну усю інформацію про неї та дії, які її стосувалися.

Будучи молодим художником, або виконавцем, або просто людиною, яка хоче поділитися своїми знаннями з усім світом, буде важливо бути впевненим, що твої досягнення будуть оцінені достойно, а не вкрадені, чи використані без твого дозволу. Саме маркетплейси, які продають NFT (тут може бути що завгодно: від картин невідомих художників, до речей самої королеви Англії), є чудовим рішенням для молодих підприємців.

Метою курсової роботи є розробка такого веб-застосунку, який буде повністю децентралізованим, що вбереже користувача і його інформацію від інтернет-злочинців, а сам сервер застосунку від непередбачуваних 400х помилок.

Основним завданням проєкту є дослідження найбільш відомих додатків NFT маркетплейсів на різних блокчейнах, вивчення та аналіз їх функцій та можливостей, виявлення переваг та недоліків наявних застосунків, що допоможе покращити розробку та бути більш корисним для кінцевого користувача.

Об’єктом дослідження є створення децентралізованого веб-сайту, що реалізує покупку та продаж робіт сучасного мистецтва. Функціонал сайту

базується на основі аналізу різноманітних веб-застосунків та мобільних додатків, що дозволяють купувати та продавати NFT з меншою комісією за транзакції.

Предметом дослідження стали відомі маркетплейси, такі як OpenSea, Binance NFT Marketplace та інші, дослідження ціни за газ на різних блокчейн платформах для виявлення найменшої комісії для транзакцій, аналіз потреб у функціоналі та отриманні прибутку за продаж зі сторони митців, користувачів та власників маркетплейсів для створення найбільш ефективного рішення.

Основними *методами* дослідження є спостереження, аналіз, обробка та систематизація досліджуваних даних.

Практичне значення полягає в отриманні децентралізованого застосунку, який допоможе сучасним поки що невідомим митцям продавати свої творіння захищаючи право інтелектуальної власності у мережі інтернет та запобіганню власникам онлайн застосунків вберегти приватну інформацію про їхніх користувачівю

Для створення застосунку “KRYPTOPAINTZ MARKETPLACE” було використано сучасні засоби розробки для покращення роботи з децентралізованими програмами. Увесь застосунок було реалізовано в інтегрованому середовищі розробки програмного забезпечення Visual Studio Code версії 1.66.2 від компанії Microsoft, що підтримує роботу зі створенням смарт контрактів на мові програмування Solidity, деплой та верифікацію контрактів за допомогою додаткових плагінів від Hardhat, та систему керування версіями Git. Під час розробки використовувався веб-сервіс для збереження проміжних версій проекту GitHub. Для збереження статичних даних, таких як файли з зображеннями, паролями користувачів та статистичними веб-сторінками було використано гіпермедійний протокол IPFS, що частково замінює бази даних для централізованих застосунків. Для зберігання всієї іншої інформації використовувався блокчейн Avalanche. Для взаємодії користувача зі смарт контрактами було використано бібліотеку

Web3.js. Частина UI реалізована за допомогою фреймворку React Native та шаблонізатора CSS - Tailwind. Результати роботи та працездатність веб-застосунку перевірялися у браузерах Google Chrome та Opera з використанням вбудованих інструментів розробника.

Робота складається з анотації, вступу, трьох розділів, списку використаних джерел та додатків, які місять ілюстрації та програмний код. Перший розділ містить аналіз сучасного стану та обґрунтування предметної області, огляду існуючих аналогів розробки, та постановки задачі. Другий розділ складається з теоретичних відомостей про задачу, методи і розв’язки оптимальних рішень. У третьому розділі наведений опис аналізу технічного завдання, обґрунтування алгоритму, структури, вибору засобів розробки, тестування програми та результати її виконання.

РОЗДІЛ 1

Аналіз предметної області. Постановка завдання курсової роботи

1.1. Аналіз сучасного стану питання та обґрунтування теми

“NFT” став словом року за версією Collins Dictionary у 2021 році, обігнавши "crypto" та "cheugy". Невзаємозамінні токени доволі нове значення у світі технології блокчейн, але дуже швидко набуло шаленого буму і стало частиною нового Web 3.0. Разом з дослідженням ERC-721 токенів, почали з’являтися і маркетплейси, де крієйтори мали змогу мінтити, продавати та купувати свої творіння. Більшість інвесторів та бізнесменів почали вбачати в цьому хороші інвестиції, адже, на відміну від криптовалют, попит на які може впасти, а відповідно і впаде ціна, такі токени лише будуть набувати більшої вартості. Це пояснюється тим, що NFT – унікальний невзаємозамінний токен, що існує лише в одному екземплярі, та не може прирівнюватись по вартості до іншого, а також має властивості збереження права власності за допомогою технології блокчейн*(Рисунк 1)*. У той самий час звичайні криптовалюти, які представляють собою ERC-20 токени, можуть бути легко заміщені або обмінені на інші.

Рисунок 1 (Приклади NFT токенів)

Лише пройшов рік, як стало зрозуміло, що ця нова технологія — інструмент на основі блокчейну, який дає змогу будь-кому монетизувати цифровий контент — переростає у велику галузь. Тому найбільші криптовалютні біржі почали впроваджувати можливість торгувати NFT. Для роздрібної торгівлі створення власних маркетплейсів не є вигідним рішенням, оскільки створення смарт-

контрактів, які імплементують дане рішення коштує дорого. Тому більшість надало перевагу партнерству зі сторонньою платформою для зменшення початкових витрат, отримати доступ до більшої наявної клієнтської бази та надати цінне доповнення такими послугами як маркетинг, юридична та технічна підтримка.

З іншого боку популярні марктплейси забирають більший відсоток комісії за транзакції, що не вигідно для покупця, а також не дає можливість молодим невідомим митцям конкурувати з вже відомими.

Ще одним недоліком великих бірж, які просто впроваджують можливість взаємодіяти з ERC-721 токенами є неповна децентралізація застосунків, що робить їх вразливими.

1.2. Огляд існуючих аналогів розробки

У зв’язку з тим, що NFT набирає все більше популярності у світі, більшість передових блокчейн платформ стараються впровадити NFT маркетплейси на своїх біржах. Але це не завжди вигідно через комісії у криптовалюті, до якої прив’язана біржа.

1.2.1. Crypto.com

Вважається найкращим NFT маркетплейсом з більш ніж десяти мільйонною аудиторією. Створений у 2021 році біржою маркетплейс дозволяє торгувати NFT таких категорій як арт, селебретіз, ігри, спорт, музика та інші. Як показано на Рисунку 2, платформа дозволяє створити, купити, виставити на аунціон токени. Покупка здійснюється у декілька кліків, здійснивши оплату за допомогою банківської картки або за допомогою криптовалюти на біржі. Перевагою є те, що комісія за транзакції відсутня, а комісія за продаж складає лише 1.99%.

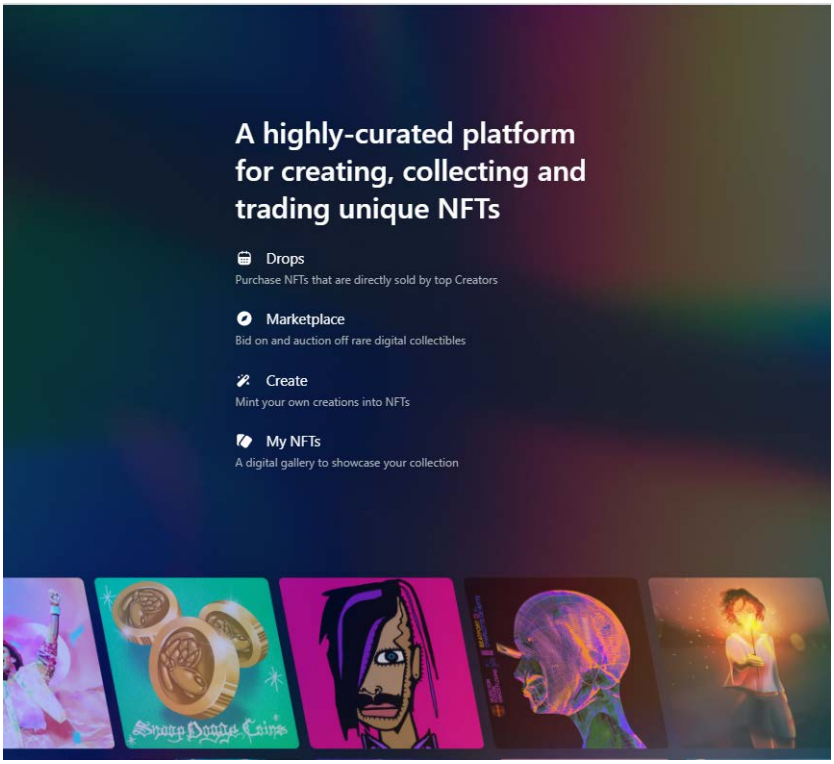

Рисунок 2 (Головна сторінка Crypto.com)

1.2.2. NFT Launchpad

Дана платформа являється найбільш захоплюючим цифровим ринком для інвесторів, так як була анонсована лише у лютому 2022 року. Ринок пропонує NFT, випущені на блокчейнах Binance Smart Chain (BSC) і Polygon, які являються найбільш популярними біржами для продажу. Перейшовши на головну сторінку маркетплейсу *(Рисунок 3)*, одразу пропонується підключити свій MetaMask гаманець, що спрощую авторизацію користувачів.

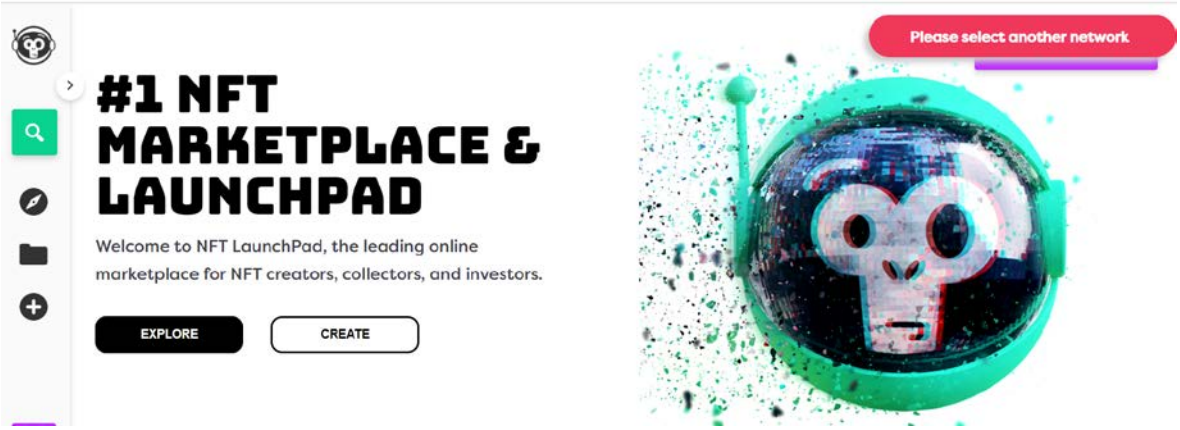

Рисунок 3 (Головна сторінка NFT Launchpad)

Платформа підтримує різні формати NFT, включно з MP4. Користувачі можуть також встановлювати роялті(гонорар) при мінтінгу і вирішувати, яким рівнем рідкості буде виражена їх колекція*(Рисунок 4)*. Розробники додали можливість створювати окрему сторінку з ERC-721 токенами, для людей які зацікавлені стежити за ними, за їхніми власниками, аукціонами.

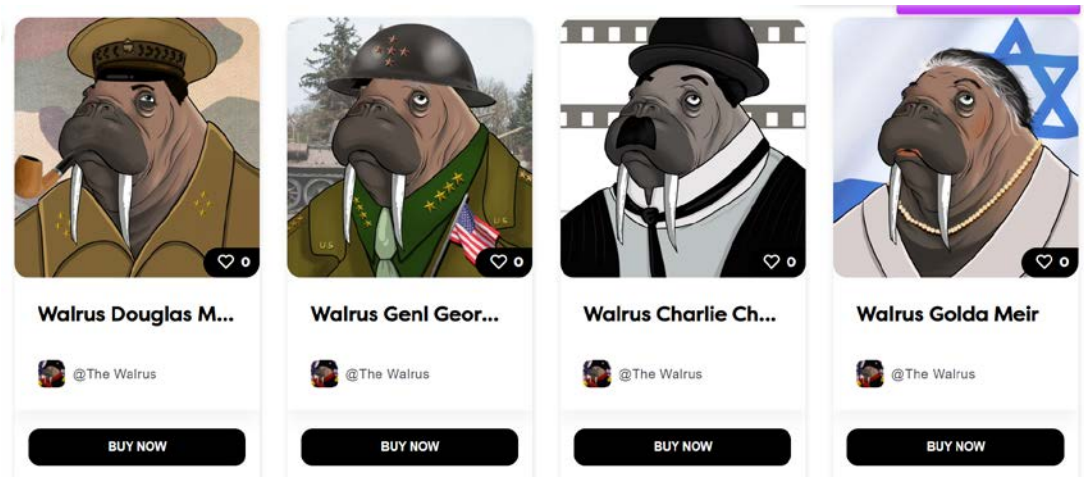

Рисунок 4 (Приклади NFT токенів)

Перевагою застосунку є легкість створення нових токенів, великій вибір колекцій для покупців, підтримка BSC and Polygon NFTs.

Головним недоліком є велика комісія за транзакція через попит на криптовалюти BNB та Matic, до який прив’язана оплата.

1.2.3. Binance

Платформа Binance вважається найбільшою криптовалютною біржою за версією CoinMarketCap. І такоє має свій власний маркетплейс з низькими комісіями за транзакції, всього лиш 1%. Крім того, Binance NFT Marketplace *(Рисунок 5)* пропонує NFT на Binance Smart Chain і блокчейні Ethereum, забезпечуючи широкий вибір оборотних активів. Дана платформа підтримує подвійну верифікацію, але якщо у користувача вже є акаунт на біржі, е потрібно створювати новий. Найцікавішою функцією застосунку є так звана “Таємнича скринька”, де користувач може придбати токен за встановлену ціну не знаючи завчасно що там буде. Такий ERC-721 токен може бути набагато ціннішим, що є гарним маркетинговим рішенням компанії.

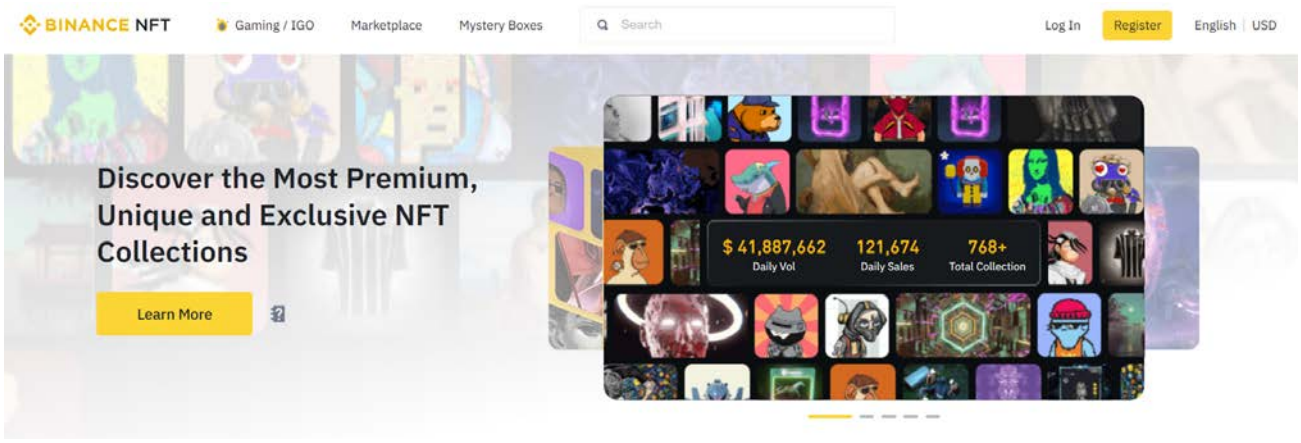

Рисунок 5 (Головна сторінка Binance marketplace)

На даний момент оплату можна здійснити лише у криптовалюті ETH або BNB, а для користувачів, які її не мають можна придбати ERC-20 токени на біржі Binance *(Рисунок 6)*.

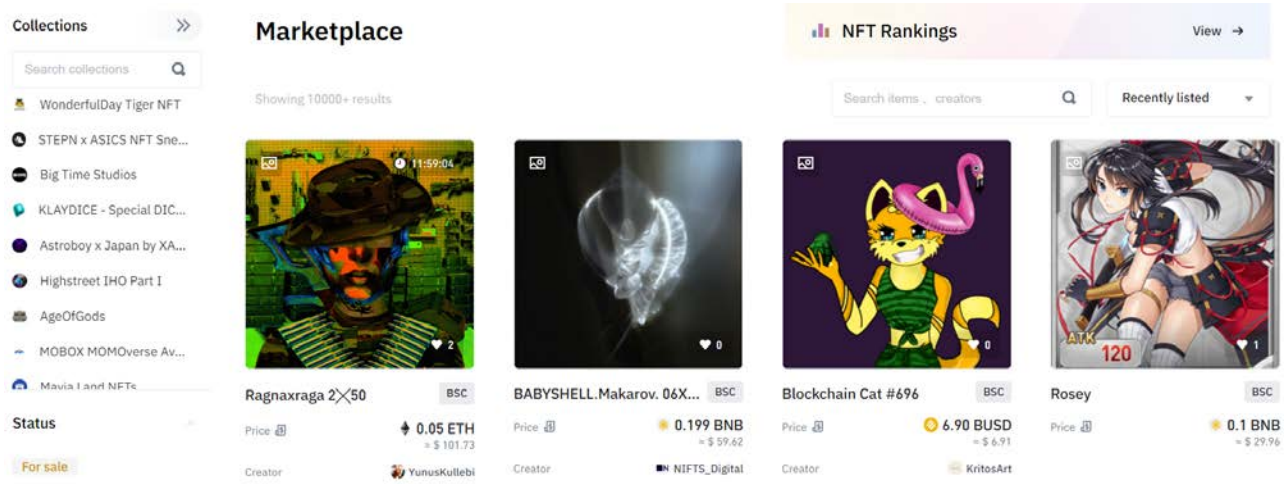

Рисунок 6 (NFT collection in Bainance marketplace)

1.2.4. OpenSea

OpenSea була першою платформою, яка ввела торгівлю NFT *(Рисунок 7-8)*. На ній можна знайти багато різних невзаємозамінних токенів, таких як предмети мистецтва, речі для онлайн ігор, землі для метавсесвіту, доменні імена, і багато чого іншого. Маркетплейс побудований на блокчейні Ethereum. Однією з переваг платформи є те, що вона підтримує більше 150 криптовалют як способів оплати, тож користувачам не потрібно додаткових бірж для обміну валюти. Також OpenSea дозволяє здійснювати транзакції з 14 різних гаманців, чим приваблює велику кількість інвесторів. Ще однією з переваг є низька комісія для продавця,

що складає 2.5%. Але, не беручи до уваги такі плюси, найбільшим мінусом є те, що через велику популярність митці-початківці не можуть отримати великий прибуток за своїх робіт.

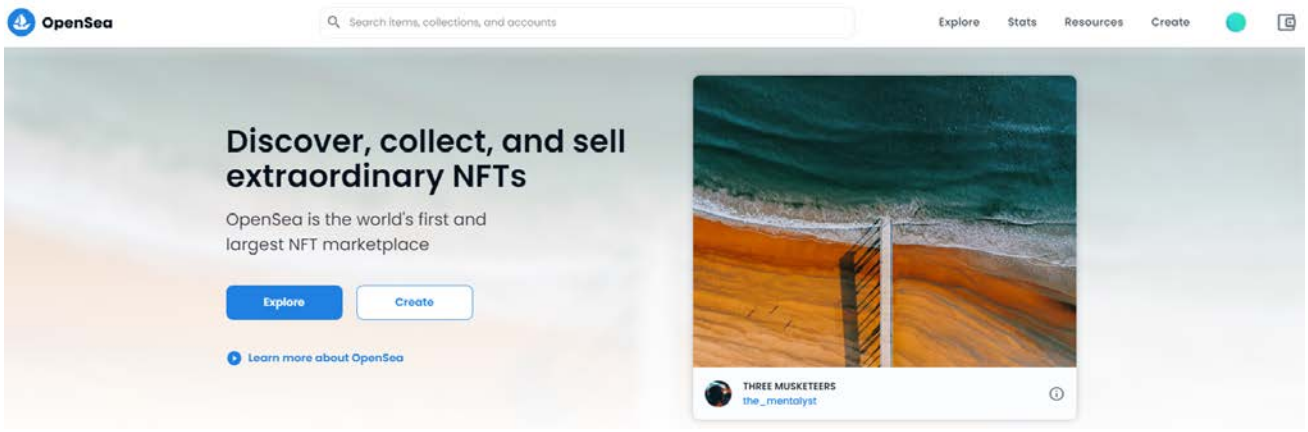

Рисунок 7 (Головна сторінка OpenSea)

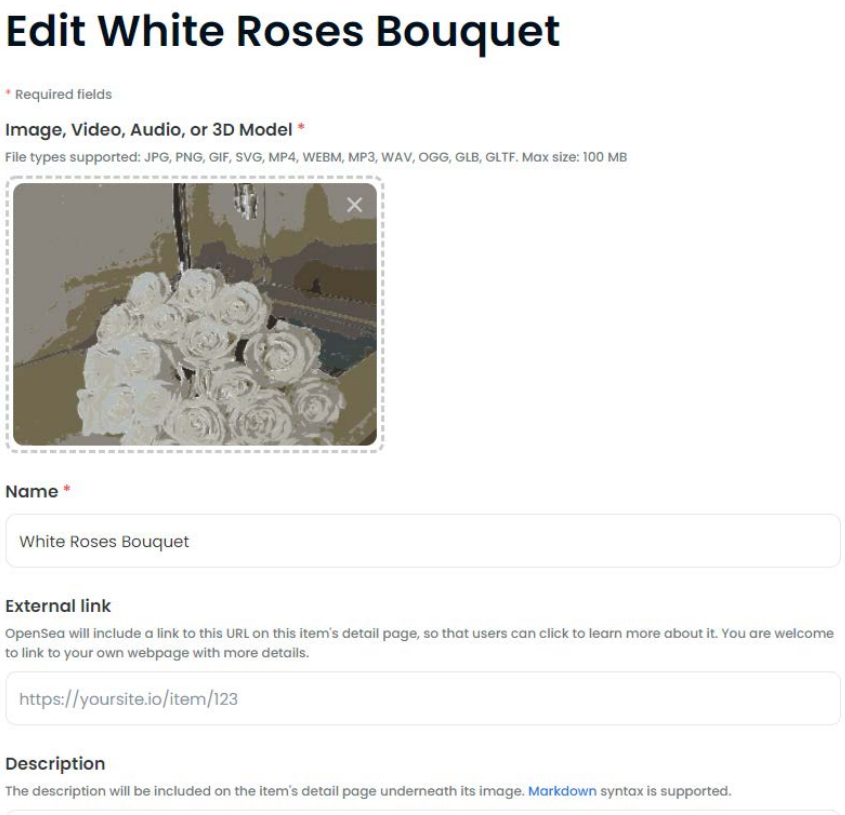

Рисунок 8 (Створення NFT на OpenSea)

1.2.5. Nifty Gateway

Як зазначено в рейтингу найбільш популярних NFT маркетплейсів [8] – дана платформа спеціалізується на «преміальних» випусках ERC-721 токени, які є

випусками NFT з обмеженою кількістю активів. На платформі представлено широкий спектр відомих художників і творців, а в минулому виступали такі, як Стів Аокі та Граймс. Оскільки ці NFT користуються великим попитом, вибір Nifty Gateway більше пристосований до високоякісних інвесторів *(Рисунок 9)*. Одним з мінусів є те, що Nifty Gateway є частиною Gemini, криптовалютної біржі розташованої у Сполучених Штатах Америки, де транзакції можуть здійснюватися за допомогою кредитної або дебетової картки. Для усіх інших країн потрібно мати на гаманці криптовалюту для проведення транзакцій. Ще одним мінусом платформи є те, що вони стягують з продавця кожного NFT комісію в розмірі 5%, а також плату за транзакцію в розмірі 0,30 доларів США, що є достатньо високою в порівнянні з іншими розкрученими маркетплейсами.

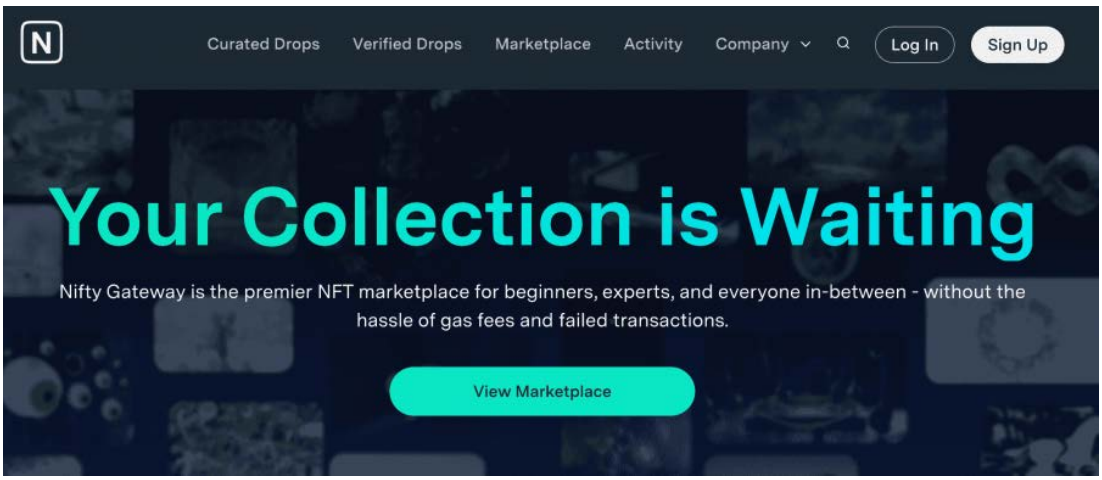

Рисунок 9 (Головна сторінка Nifty Gateway)

1.2.6. Rarible

Ще однією не менш захоплюючою платформою для торгування ERC-721 токенами, як зазначено у статті [8], є Rarible *(Рисунок 10)* —маркетплейс NFT, що належить спільноті, де його власники мають токен ERC-20 RARI. Платформа надає токен RARI активним користувачам, які купують або продають на ринку NFT. Він розподіляє 75 000 RARI щотижня. Платформа приділяє особливу увагу арт-активам. Творці можуть використовувати Rarible для «карбування» (мінтингу) нових NFT для продажу своїх творінь, будь то книги, музичні альбоми, цифрове мистецтво чи фільми. Rarible купує та продає NFT у таких категоріях, як мистецтво, фотографія, ігри, метавсесвіт, музика, домени, меми тощо.

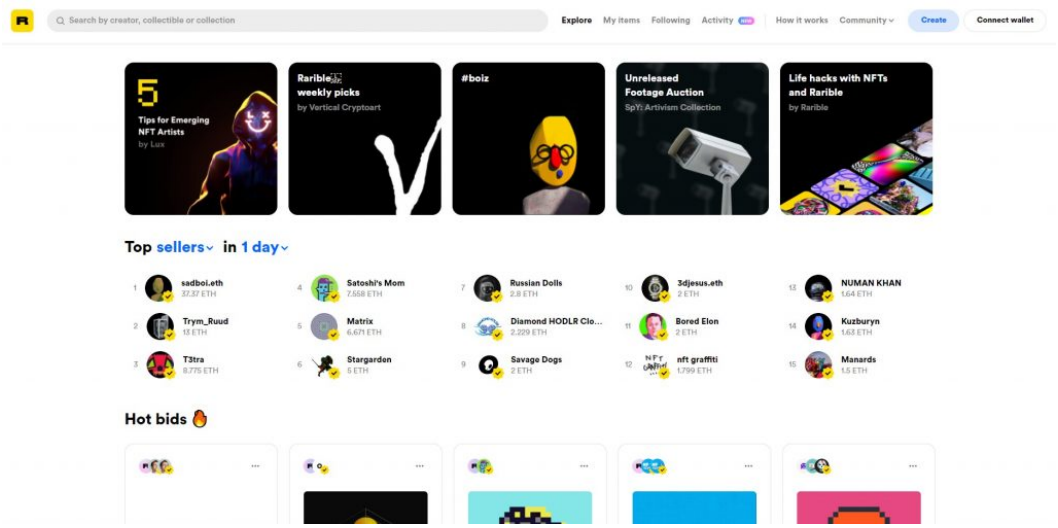

Рисунок 10(Головна сторінка Rarible)

1.2.7. DraftKings

Як зазначено у статті рейтингу найкращих маркетплейсів [8], DraftKings — це величезна американська букмекерська компанія, яка в першу чергу зосереджується на спортивних ставках. Компанія запустила власний ринок NFT наприкінці 2021 року і може похвалитися обмеженим тиражем активів багатьох провідних спортсменів. Відомі NFT, які були перераховані на ринку DraftKings *(Рисунок11)*, включають предмети від Тома Бреді, Тоні Гока, Вейна Гретцкі та інших. На даний момент ринок пропонує понад 300 предметів колекціонування, починаючи від дешевших варіантів і закінчуючи преміальними цінами. Користувачі можуть поповнити свій гаманець за допомогою кредитної або дебетової картки, банківського переказу та навіть PayPal. Покупки дозволені лише в доларах США, підтримка покупки у криптовалюті відсутня. Нарешті, транзакційна комісія невідома, хоча продажі на вторинному ринку будуть нараховувати 10% роялті початковому творцю та 5% комісії за транзакцію, що є значним недоліком для нового копупця під час покупки.

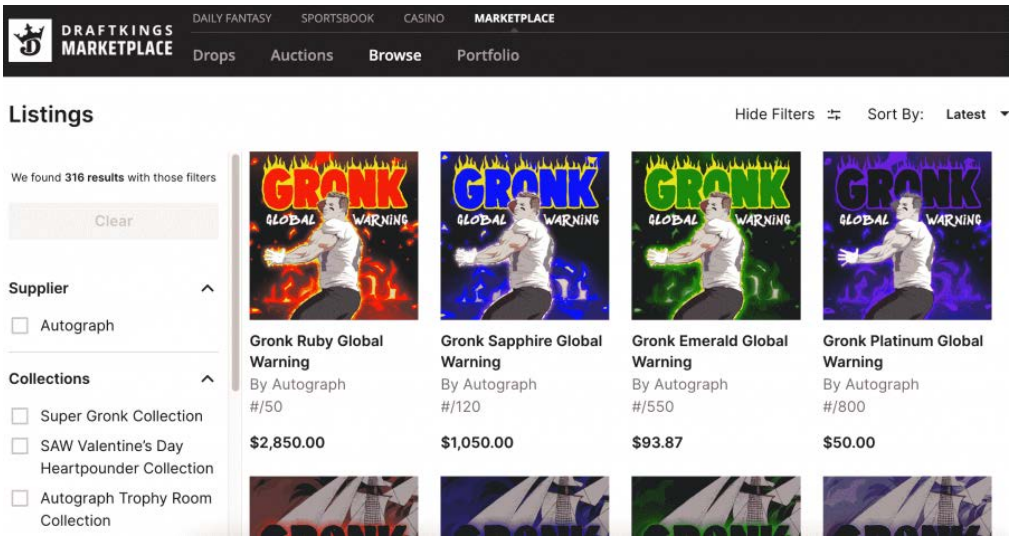

Рисунок 11 (Головна сторінка GraftKings)

1.3. Постановка задачі

У результаті дослідження предметної області та огляду існуючих аналогів NFT маркетплейсів можна сформувати повне представлення децентралізованого сервісу, який потрібно розробити та його основних функціональних можливостей.

Таким чином головною задачею курсової роботи є розробка децентралізованої платформи (веб-застосунку) для ERC-721 токенів. Головним призначенням є випуск(мінтінг), продаж/покупка невзаємозамінних токенів, проведення аукціонів для них. Для цього можна сформувати наступні можливості:

- реєстрація користувача для KYC
- збереження статичної інформації за допомогою IPFS
- підключення гаманця MetaMask
- покупка криптовалюти маркетплейсу(ERC-20 токени)
- випуск власних ERC-721 токенів
- виставлення NFT на продаж
- покупка NFT

- перепродаж NFT для отримання роялтіз
- вивід коштів та токенів з маркетплуйсу на гаманці користувачів
- створення/видалення аукціонів власниками токенів
- ставлення ставок учасниками аукціону для покупки токенів
- роздача додаткових токенів найбільш активним користувачам платформи

РОЗДІЛ 2

Теоретичні відомості

2.1. Предметна область

Після того як слово NFT стало мейнстрімом, попит на деякі ERC-721 токени стрімко зріс. На *Рисунку 12* можна побачити картину *"Everydays - The First 5000 Days"*, що стала одним з найдорожчих невзаємозамінних токенів у світі. Як зазначено у статті про опис цього творіння, вона являється продуктом Майка Вінклеманна, художника, також відомого під псевдонімом «Біпл». Цей файл JPG являє собою колаж із зображень, створених художником один день за раз, починаючи з 2007 року, і його часто приписують початку буму зацікавленності щодо NFT. NFT "Everydays" був придбаний 11 березня 2021 року в аукціонному домі Christie's інвестором криптовалюти з Сінгапуру, який заплатив за ілюстрацію в ефірі (ETH-USD). Покупець отримав лише цифровий артефакт, нічого фізичного для його супроводу. Ця покупка зробила художника одним із найбільш продаваних художників у світі.

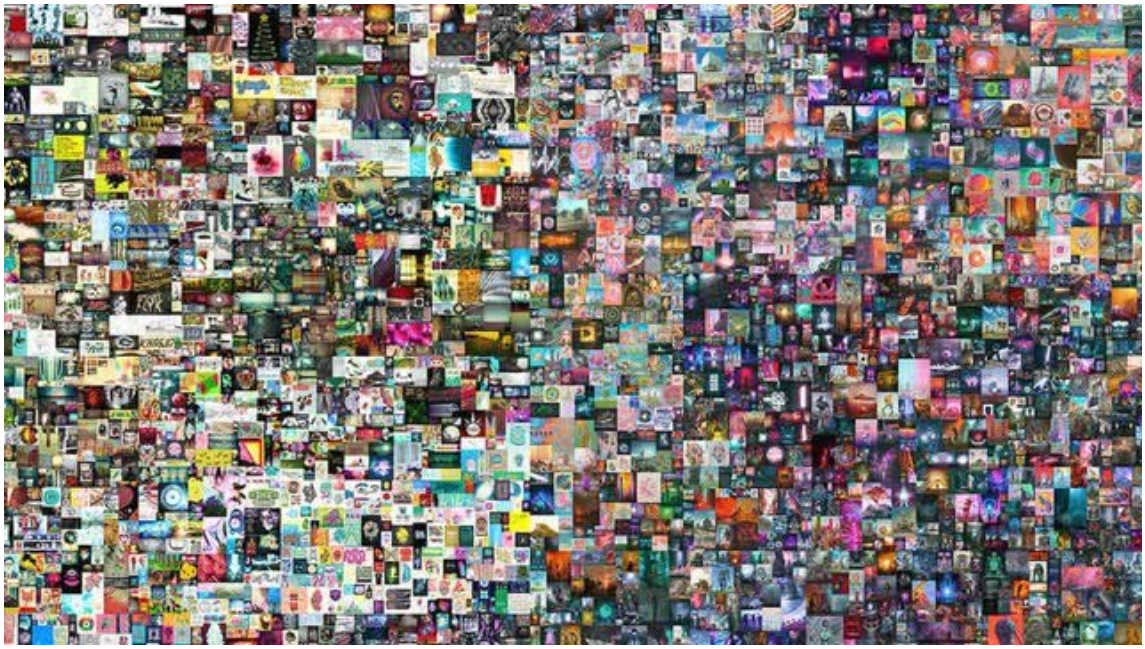

Рисунок 12 (NFT Everydays)

Друге та третє місце за популярністю та впізнаваністю посідають наступні колекції: CryptoPunks та CryptoKitties.

Перша – це колекція ERC-721 токенів, випущена у 2017 році канадськими розробниками з Larva Labs*(Рисунок 13)*. Дана колекція складається з 10 000 унікальних зображень 24×24 пікселів, на яких зображені переважно люди (чоловіки та жінки). Однак є кілька інших унікальних видів, які вважаються більш цінними через свою рідкість. Серед них зомбі (88), мавпи (24) та інопланетяни (9). Кожен CryptoPunk також може демонструвати комбінацію з 87 унікальних атрибутів. Вони відомі як «риси» і включають такі речі, як капелюхи, люльки, намиста, сережки, пов'язки тощо. Максимальна кількість рис, яку може мати один CryptoPunk, становить сім. Проте існує лише один CryptoPunk із сімома ознаками, #Punk 8348. У нього є сигарета, сережка, родимка, глиняні зуби, класичні відтінки, циліндр і велика борода. [9]

Attribute Counts

Attribute	#	Avail	Avg Sale	Cheapest	More Examples
0 Attributes	8	1	0	4KΞ	
1 Attributes	333	33	131.61Ξ	169Ξ	
2 Attributes	3560	446	62.89Ξ	92Ξ	
3 Attributes	4501	563	62.19Ξ	94Ξ	
4 Attributes	1420	212	59.90Ξ	100Ξ	
5 Attributes	166	42	108.67Ξ	122Ξ	
6 Attributes	11	3	0	1.5KΞ	
7 Attributes	1	0	0		

Рисунок 13 (CryptoPunks)

Друга – представляє собою повністю децентралізовану онлайн гру, яка дозволяє гравцям купувати, продавати, збирати та розводити віртуальних котиків, кожен з яких буде унікальним *(Рисунок 14)*. Так само як і CryptoPunks, стала популярною у 2017 році, чим підняла високо ціни на газ на Ethereum за транзакції. Саме завдяки всьому галасу, який оточував гру, з'явився термін «NFT». Для гри було випущено лише 100 початкових котів стандарту ERC-721 і

50000 додаткових, яких можна було створити змішавши риси предків, що і зробило їх унікальними.

Рисунок 14 (CryptoKitties)

2.2. Класифікація Блокчейнів за відкритістю

За останні роки блокчейн набув значної популярності, тож і розвинулися різні варіанти за доступністю зберігання даних та бізнес цілей, які вони мають наслідувати. Загалом популярними є два види: приватні та публічні.

2.2.1. Public Blockchain

Публічний блокчейні відрізняється тим, що будь-хто може вільно приєднатися та брати участь у основній діяльності даної мережі блокчейнів. Як зазначено в описі на одному з найвідоміших сайтів про технологію блокчейн якщо хтось бажає створити повністю відкритий блокчейн, який дає змогу кожному приєднатися та зробити внесок у мережу, він може вибрати публічний блокчейн. Будь-хто може читати, записувати й проводити аудити поточнщї діяльністі у публічній мережі блокчейнів, що допомагає публічному блокчейну підтримувати його самокерований характер. Такий мережа працює за схемою стимулювання, яка заохочує нових учасників приєднуватися та підтримувати мережу динамічною. У публічного блокчейну є кілька недоліків. Основним з них є велике споживання енергії, необхідне для підтримки розподіленої загальнодоступної інформації. Інші

проблеми включають відсутність повної конфіденційності та анонімності. Це може призвести до зниження безпеки мережі та ідентифікації учасників. [10]

2.2.2. Private Blockchain

Приватний блокчейн дає змогу лише вибраним учасникам за автентичним та перевіреним запрошенням приєднатися до приватного блокчейну. Перевірка відбувається операторомами мережі, або за допомогою чітко визначеного встановленого протоколу (алгоритку), реалізованого мережею. Основна відмінність між публічним та приватним блокчейнами полягає в тому, що приватні блокчейни контролюють, кому дозволено брати участь у мережі, виконувати протокол консенсусу, який визначає дозволені права та винагороди учасників. За потреби власник або оператор має право змінювати, редагувати або видаляти необхідні записи в блокчейни. У справжньому сенсі приватний блокчейн не є децентралізованим і виступає як розподілений реєстр, який функціонує як закрита безпечна база даних, заснована на концепціях криптографії. Відповідно не кожен може запустити ноду у приватному блокчейні, здійснити транзакції або перевірити/автентифікувати зміни в блокчейні.

2.2.3. Permissioned Blockchain

Як зазначено в описі на сайті Investopedia permissioned blockchain дозволяють поєднувати публічні та приватні блокчейни та підтримують багато варіантів налаштуваннь. Вони включають в себе дозвіл будь-кому приєднатися до мережі після відповідної перевірки своєї особи, а також виділення вибраних і призначених дозволів для виконання лише певних дій у мережі. Наприклад, Ripple, одна з найбільших криптовалют, підтримує приєднання нових учасників на основі встановлених для них ролей. Це дає учасникам можливість виконувати певні функції, такі як читання, доступ та запис інформації в блокчейн. Компанії все частіше вибирають permissioned мережі блокчейн, оскільки це дозволяє їм вибірково встановлювати обмеження під час налаштування мереж і контролювати діяльність різних учасників відповідно до їх бажаних ролей. [10]

2.3. Класифікація Блокчейнів за комісією для транзакцій

Ще однією важливою складовою у блокчейнах є комісія за газ, яка встановлюєтьс кожної мережею самостійно і може виплачуватися у різних криптовалютах, в залежності від її ціни на біржі. Так окрім популярних блокчейнів Etherscan та BSC, які приймають транзакції відповідно в криптовалюті ETH та BNB, що може досягати десятки доларів, через їх попит на біржах, розробники та замовники все частіше звертають увагу на такі блокчейни як Polygon, Avalanche та Solana.

2.3.1. Ethereum

Хоча платформа отримала певну критику за обмежену масштабованість і високу плату за транзакції, вона, безумовно, є найбільш використовуваною платформою для розробки блокчейну, особливо коли йдеться про розробку децентралізованих додатків. Не дивлячись та конкуретні блокчейни, які пропонують кращу маштабованість та нижчу плату за транхакції, Ethereum розвивається, і наполегливо працює над низкою рішень щодо масштабування, спрямованих на вирішення багатьох поточних проблем Ethereum. [11]

2.3.2. BSC

Одна з платформ, яка найбільше виграла від попиту на більш масштабовану альтернативу Ethereum, зарекомендувала себе як одна з найкращих платформ для розробки смарт-контрактів. Рішення прийняти більш централізовану модель консенсусу дозволило Binance Smart Chain (BSC) отримати значні переваги щодо пропускної здатності транзакцій та економічної ефективності. У той же час BSC сумісна з EVM, що робить її крос-платформеною. [11]

2.3.3. Polygon

Започаткована як Matic Network в 2017 році, платформа почала набирати популярністі після успішного запуску основної мережі в 2020 році, який стався в той час, коли зростання плати за газ в Ethereum викликав серйозний попит на пошук ефективного рішення для масштабування. На початку 2021 року платформа змінила бренд на Polygon, що відображало її зростаючу амбіцію створити протокол і структуру для створення рішень масштабування для Ethereum

рівня L-2. На сьогоднішній день Polygon пропонує популярне рішення для сайдчейнів та рішення на основі Plasma. [11]

2.3.4. Avalanche

Створена після успішного первинного розміщення монет у 2020 році, Avalanche наразі має значну активність у розробці. Платформа підтримує EVM, яка забезпечує cross-chain роботу для активів Ethereum. Однією з найцікавіших характеристик Avalanche є те, що він має три різних ланцюга, які працюють на двох різних алгоритмах консенсусу – Avalanche, дає підтримку швидким транзакціям, і Snowman – пропонує безпечні смарт-контракти. [11]

2.3.5. Solana

Solana впровадила низку великих нововведень, таких як новий механізм консенсусу *Proof of History,* який дозволяє йому отримати значний приріст продуктивності з точки зору пропускної здатності та витрат за трансакції. Розробники стверджують, що протокол є масштабованим, децентралізованим і безпечним, що по суті вирішує проблему «трилеми масштабованості». Завдячуючи своїй потужній продуктивності Solana змогла залучити кілька дуже перспективних проектів і стала одним із головних гравців у найбільших тенденціях технології блокчейн. Прикладом цього є нещодавній бум NFT-маркетплейсів на базі Solana на чолі з Solanart. [11]

2.3.6. Polkadot

Його часто називають «блокчейном для блокчейнів» - є одним з найбільш амбітних блокчейн-проектів. Завдяки своїй унікальній архітектурі Polkadot зміг вивести шардінг на абсолютно новий рівень і потужність. Ця багатоланцюжкова архітектура дозволяє розробникам створювати спеціалізовані ланцюги, оптимізовані для конкретних цілей. Розробники Polkadot також мають доступ до Substrate, потужного фреймворку для створення блокчейну, модульний дизайн якої дозволяє легко додавати функції та фічі, які відповідають дизайну та призначенню їхніх ланцюжків. [11]

2.4. Класифікація стандартів токенів

Стандарт ERC — це абревіатура від Ethereum Request for Comments. Вони визначають методи, поведінку, інновації та дослідження, застосовні до групи розробників і користувачів, які хочуть використовувати екосистему Ethereum. Їх значення полягає у тому, що набір правил, яких повинен дотримуватися кожен токен на основі Ethereum, імплементується у смарт-контрактах. ERC-20, ERC-721 і ERC-1155 – три найбільш популярні стандарти або протоколи маркерів ERC*(Рисунок 15)*.

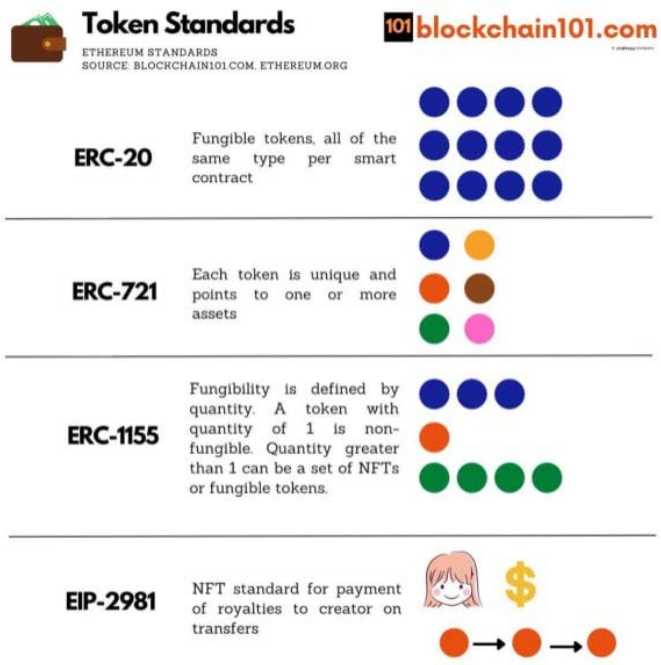

Рисунок 15 (Difference between tokens standarts)

2.4.1. ERC-20

Токени ERC-20 — це інша назва «замінних токенів», де змінюваність визначає здатність активу або токена обмінюватися на активи такої ж вартості, наприклад, на дві банкноти по 1 долар. Кожен токен ERC-20 суворо еквівалентний одному і тому ж значення незалежно від його функції та структури.

2.4.2. ERC-721

ERC-721 або NFT є неоціненним для цифрових активів, які представляють чиєсь право власності на ці активи. ERC-721 може представляти наступне: унікальний цифровий твір, твіти та пости в соціальних мережах, внутрішньоігрові предмети

колекціонування, ігрові персонажі. Кожен NFT має змінну uint256, відому як tokenId, де tokenId має бути унікальним.

2.4.3. ERC-1155

Даний стандарт поєднує у собі можливості ERC-20 і ERC-720, його ще називають стандартом «все включено» для смарт-контрактів Ethereum. Це стандартний інтерфейс, який підтримує розробку взаємозамінних, напіввзаємозаммінних, невзаємозамінних токенів та інших конфігурацій із смарт-контактом, що покращує загальну функціональність попередніх стандартів токенів ERC.

2.4. Опис технічних відомостей

Розуміння роботи програми прямо пропорційно залежить від розуміння всіх її компонентів та взаємозв'язків. Децентралізований маркетплейс частково схожий за своїм архітектурним рішенням на клієнт-серверну архітектуру. Проте, як зазначено у статті Dapp - це програма, яка працює в розподіленій мережі. Він розміщений не на централізованому сервері, а в одноранговій децентралізованій мережі. Відрізняє децентралізований додаток від традиційного те, що він побудований на децентралізованій мережі, як-от Ethereum, що означає що усуваєтьс потреба в третій стороні для обробки транзакцій між одноранговими партнерами (peer-to-peer). [12] Оскільки посередник замінюється кодом, зменшуються всі види витрат, включаючи гроші і час *(Рисунок 16)*.

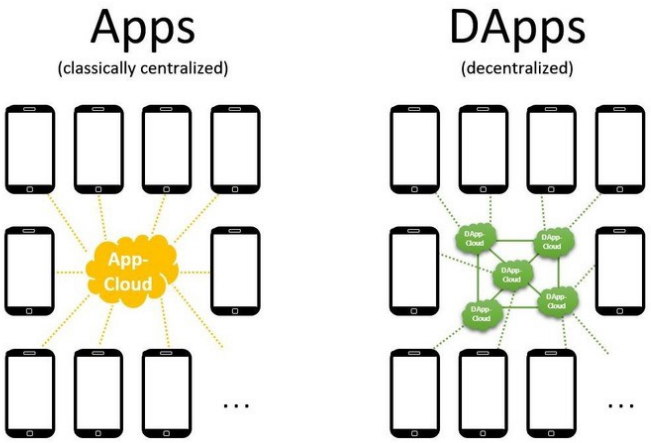

Рисунок 16 (Difference betweek app and dApp)

Відповідно коли створюються власні смарт-контракти на Ethereum, насправді пишеться фрагмент бекенд-коду для Dapp. І хоча він буде мати

користувальницький інтерфейс, як традиційний централізований додаток, або весь, або частина бекенду буде побудована на основі Ethereum. Одже для dapp сожна вивести наступну формулу:

Dapp = frontend + smart contract backend

Щоб досягнути повної децентралізації застосунку можна розмістити код користувацького інтерфейсу на децентралізованих вузлах зберігання, таких як IPFS, щоб зробити як фронтенд, так і бекенд повністю децентралізованими *(Рисунок 17)*.

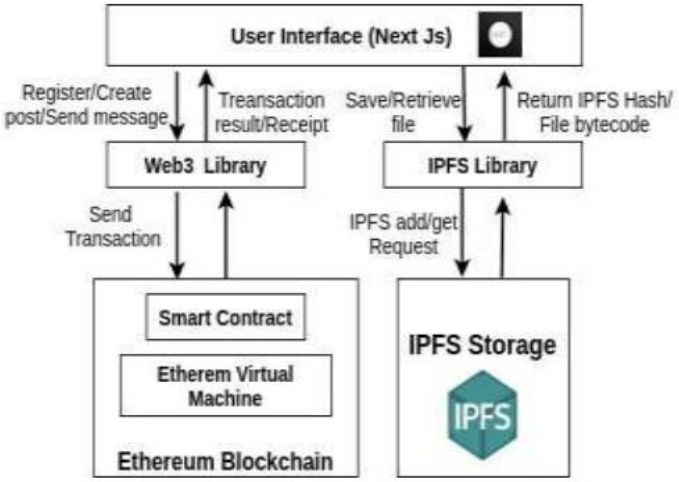

Рисунок 17 (Conncetion between sc, web3 and IPFS)

РОЗДІЛ 3

Опис реалізації програмного продукту

3.1. Аналіз технічного завдання

Як було зазначено у розділі 1.3 Постановка задачі NFT marketplace “KRYPTOPAINTZ MARKETPLACE” реалізований як децентралізований застосунок на блокчейні Avalanche, через її низькі комісії. Дослідивши найвідоміші платформи для продажу NFT, реалізація даної буде мати набагато більше переваг.

Основною аудиторією будуть молоді художники, які прагнуть легально продавати свої творіння, а покупцями – інвестори, які зацікавлені вкластися в цифрові токени за зниженою ціною на транзакції, що можливо завдяки блокчейну Avalanche. Власники маркетплейсу також будуть отримувати певний відсоток

прибутку від продажу NFT, та вхожу на маркетплейс. Хоча цей відсоток є незначним і може регулюватися власниками платформи, на підвищення ціни токенів це не впливає.

Не дивлячись на те, що верифікація відбувається за допомогою гаманця *MetaMask*, користувач має зареєструватися (ввівши *username*, *password* and *bio*, щоб в подальшому розробники маркетплейсу могли проаналізувати дані своїх клієнтів для подальшого розвитку платформи). Після реєстрації та логіну, юзеру будуть доступні всі можливості додатку. Тоді як незареєстровані користувачі можуть лише купити токени, якими вони зможуть розрахуватися при покупці NFT. Мета цих токенів підняти зацікавленість користувачів здійснювати покупки саме на даному маркетплейсі, зацікавлюючи їх та крієйторів роздачою щотижневих токенів (Airdrop).

Основним функціоналом, що доступний після логіну є випуск власних токенів, виставлення їх на продаж, покупка за ефір (ETH) або за ERC-20 токени ***MRSNLK***, де **1ETH = 50 MARIIA COIN**. Якщо власник не хоче виставляти токени на продаж, він має можливість виставити їх на аукціон, де початкова ціна, та мінімальні ставка можуть регулюватися.

3.2. Обґрунтування алгоритму й структури програми

Відповідно до поставленого завдання децентралізований додаток має складатися з фонтенду (для взаємодії з користувачем) та бекенду, яким виступають смарт контракти (для взаємодії з блокчейном).

На *Рисунку 18* зображена діаграма класів (смарт-контрактів), які наявні у програмі, та як користувач взаємодіє з ними. Окремо винесено різних учасників, які можуть виконувати наступні ролі: user (початковий користувач, який може лише переглянути головну сторінку та купити токени маркетплейсу), registered user (зареєстровний користувач, який має доступ до більшості функціоналу), owner (власник контрактів, той, з чиєї адреси було їх задеплоїно), NFT owner (власник NFT, набуває статусу після покупки токенів), NFT creator (мінтер NFT,

цей статус залишається за ним назавжди), bidder(зареєстрований користувач, який ставить ставки на аукціоні щоб придбати токени).

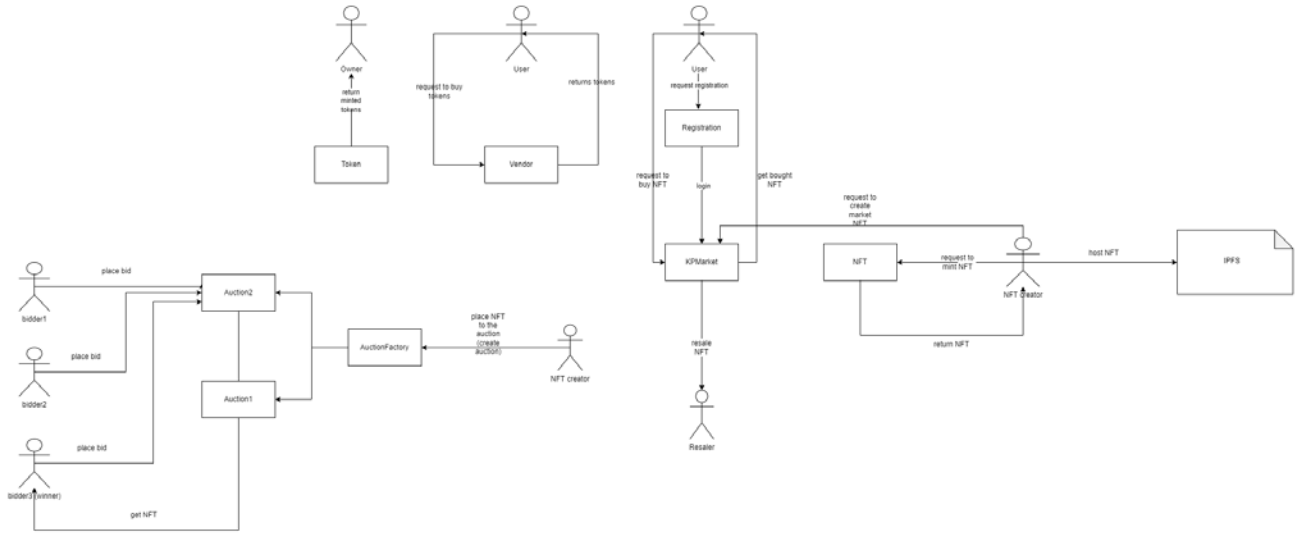

Рисунок 18 (High-level diagram)

Усі контракти написані мовою *Solidity*. Для збереження статичних файлів, таких як NFT після їх створення мінтерами, використовується хостинг *IPFS*. Для реалізації фронтенду використовується *React Native* та бібліотека-шаблонізатор *Tailwind*. Для взаємодії фронтенду та смарт-контрактів використовується бібліотека *Web3.js*. Структура проекту поділена на дві частини: *contracts* і *fronfend*, та допоміжних файлів і пакетів *Рисунок 19*

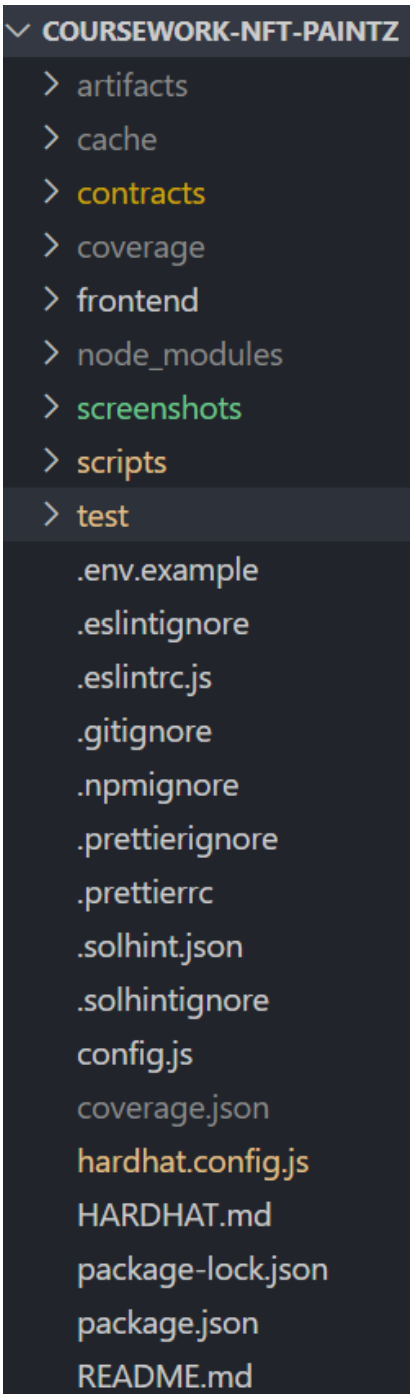

Рисунок 19 (coursework structure)

3.3. Обґрунтування вибору засобів розробки

Оскільки розробка здійснювалася за допомогою блокчейну, основні засоби розробки стосуються написання, тестування, деплою та верифікації смарт-контрактів.

- *Solidity* - це об'єктно-орієнтована мова високого рівня для реалізації смарт-контрактів. Призначена для націлювання на віртуальну машину Ethereum (EVM). Solidity є статично типізованою мовою програмування, підтримує

успадкування, бібліотеки та складні визначені користувачем типи серед інших функцій. В проекті використовується версія *0.8.9*, яка є частково покращеною та безпечною для роботи з математичними операціями, *low-level* викликами функцій.

- *VisualStudio Code* – середа розробки висхідного ходу, розроблена компанією Microsoft, та підтримується всіма операційними системами. Підтримує плагіни для написання смарт-контрактів на Solidity, тестування та взаємодію через Hardhat та Web3.js.
- *Hardhat* – це середовище розробки для компіляції, розгортання, тестування та налагодження програмного забезпечення Ethereum. Допомагає розробникам керувати та автоматизувати повторювані завдання, які є невід'ємними для процесу створення смарт-контрактів і dApps, а також легко впроваджувати більше функціональні можливості навколо цього робочого процесу: компіляцію, запуск та тестування, деплой смарт-контрактів. Початкове встановлення створює костяк майбутнього проекту з відповідними папками.
    - *@nomiclabs/hardhat-web3* – плагін від Hardhat для впровадження web3 в Hardhat Runtime Environment
    - *solidity-coverage* - плагін від Hardhat для перевірки покритості тестами смарт-контрактів
    - *hardhat-gas-reporter* - плагін від Hardhat для перевірки скільки газу та яка ціна буде витрачатися на виклик тої чи іншої функції
- *Web3.js* - набір бібліотек JS, який дозволяє взаємодіяти з блокчейном Ethereum віддалено або локально. Надає API для легкого використання з блокчейном. Web3 працює як обгортка для JSON RPC для підключення до віддаленого або локального вузла Ethereum за допомогою з'єднання HTTP або IPC.

- *OpenZeppelin* - надає продукти безпеки для створення, автоматизації та експлуатації децентралізованих програм, також захищаємо провідні організації, проводячи аудит безпеки їхніх систем і продуктів.
    - *contracts* - бібліотека для безпечної розробки смарт-контрактів. Реалізує такі стандарти, як ERC-20 і ERC-721 та інші.
    - *test-helpers* - бібліотека для тестування смарт-контрактів Ethereum. Дозволяє перевірити, чи повертається очікуваний результат транзакції, обробляються дуже веоикі числа, частіше для ERC-20 і т.д.
- *ERC-20* – запроваджений стандарт для взаємозамінних токенів, іншими словами, вони мають властивість, яка робить кожен токен точно таким же (за типом і значенням) як інші. Стандарт дозволяє полегшити розробку імплементуючи обов'язкові функції, що дозволяє обмінювати токени на різних біржах.
- *ERC-721* – запроваджений стандарт для NFT, іншими словами, робить токен унікальним, за своєю характеристикою. Так само як і ERC-20, дозволяє полегшити розробку імплементуючи обов'язкові функції.
- *Metamask* – програмний гаманець для криптовалюти, який використовується для взаємодії з блокчейном Ethereum. Доступ до нього можна отримати через браузер або встановивши додаток. Усі транзакції підписуються приватним ключем, яким володіє адреса, створена у Metamask.
- *IPFS* – (InterPlanetary File System) – це нова децентралізована мережа обміну файлами. Відрізняється від інших децентралізованих мереж тим, що самостійною одиницею, що передається в мережі, є блок. Блок може містити частину файлу і посилання інші блоки. З блоків вибудовується спрямований ациклічний граф із якого надалі збирається файл чи каталог.
- *Avalanche* - це децентралізований блокчейн із відкритим вихідним кодом і функціональністю смарт-контрактів, являється самостійною

криптовалютною платформою з власною монетою AVAX, в якій і платиться комісія за транзакції.

- *React* – кросплатформений фреймворк з відкритим висхідним кодом для розробки мобільних або веб додатків на JavaScript і TypeScript.
- *Tailwind* - перший утилітарний фреймворк CSS із такими класами, як flex, pt-4, text-center і rotate-90, які можна використавти для створення будь-якого дизайну безпосередньо у розмітці, є аналогом Bootstrap та використовується разом з React.

3.4. Опис розробки програми

Децентралізований веб-застосунок (dApp) являє собою набір файлів, що забезпечують роботу з блокчейном та файли, що використовуються на рівні представлення даних. Умовно розробку програми можна поділити на такі етапи:

- Розробка архітектури взаємозв’язків між смарт-контрактами
- Реалізація смарт-контрактів
- Написання тестів для функціоналі смарт-контракітв
- Написання скрипту для деплою sc
- Тестування sc у тестовій мережі Ropsten
- Розробка макету сайту
- Написання UI
- Створення стилю веб-застосування: підключення файлів .css та фреймворку Tailwind
- Підключення фронтенду до контрактів і взаємодія з ними
- Фінальне тестування dApp

Варто зазначити, що написання смарт-контрактів поділено на папки відповідно до функціоналу, який вони реалізовують. Кожен клас має свій власний інтерфейс,

що відповідає одним із принципів ООП. Також додавання зовнішніх бібліотек допомагає

3.5. Створення об'єктів і розробка головної програми

Для кращого розуміння як створюються об’єкти головного функціоналу наведено діаграми.

Користувач має зареєструватися у системі, щоб отримати доступ до сайту. Підключення гаманця MetaMask недостатньо *(Рисунок 20)*.

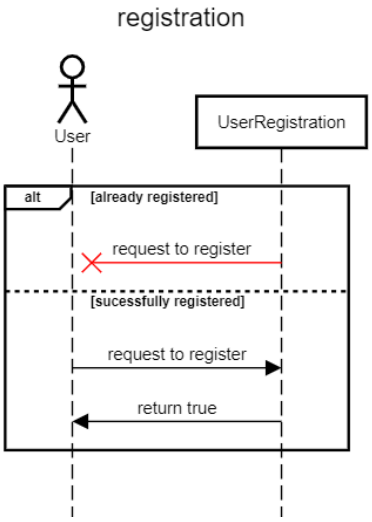

Рисунок 20 (registration functionality)

Купити токени може не зареєстрований користувач викликавши функцію buyTokens() з відповідними параметрами у контракті Vendor *(Рисунок 21)*.

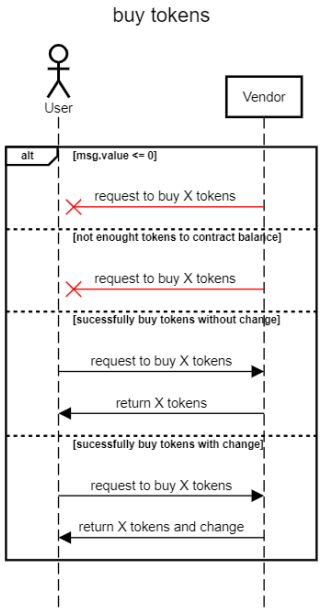

Рисунок 21 (buy tokens functionality)

Продати токени може не зареєстрований користувач лише контракту Vendor назад, викликавши функцію sellTokens() з відповідними параметрами у контракті Vendor *(Рисунок 22)*.

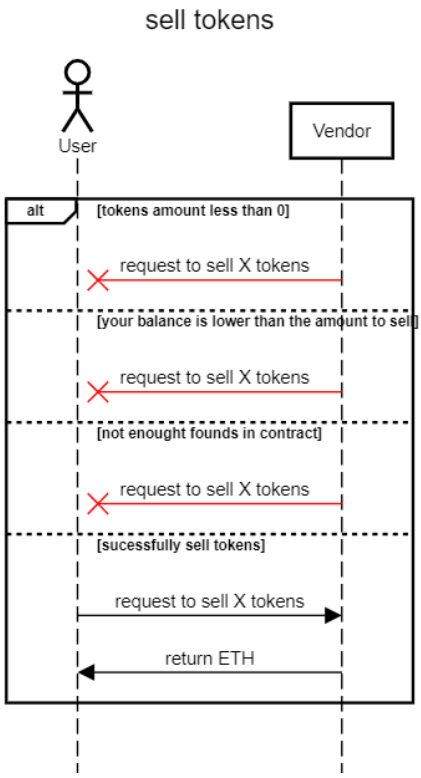

Рисунок 22 (sell tokens functionality)

Власник контракту Vendor може вивести кошти на свій гаманець за продаж токенів *(Рисунок 23)*.

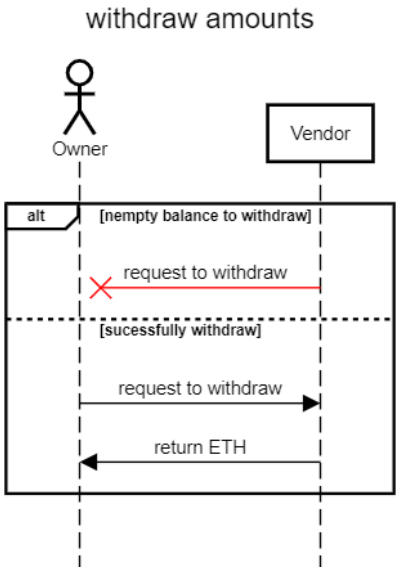

Рисунок 23 (withdraw functionality)

Власник NFT токена може продати його, створивши маркет для продажу. Після чого NFT зберігається на контракті маркета, чекаючи поки його куплять. Адреса власника змінюється, але крієйтора залишається незиінною *(Рисунок 24)*.

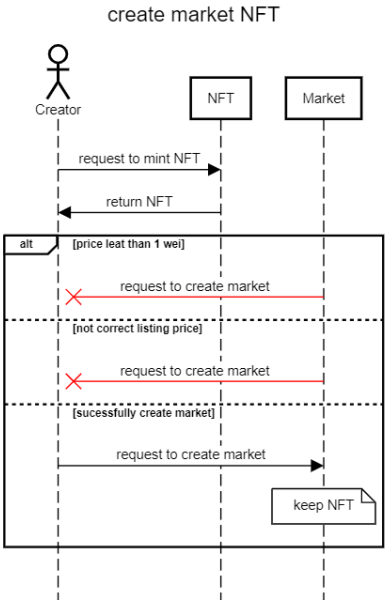

Рисунок 24 (create NFT market functionality)

Перепродаж NFT означає, що токен переходить від маркета до нового власника. Цей крок є частиною продажу токена крієйтором до покупки кінцевим користувачем. Маркетплейс виступає посередником *(Рисунок 25)*.

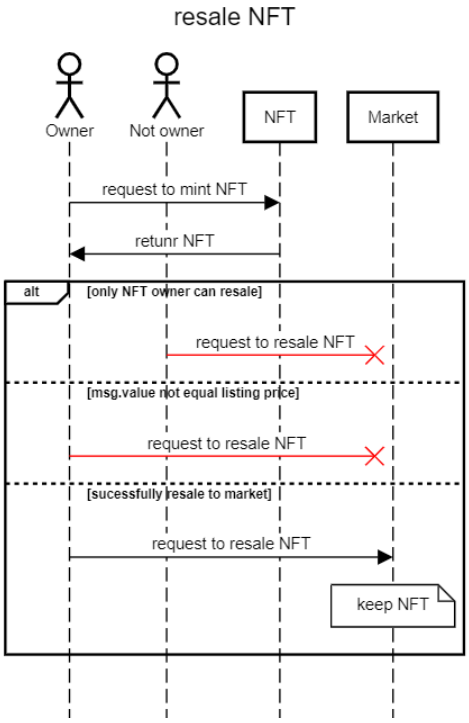

Рисунок 25 (resale NFT functionality)

Перепродаж NFT означає, що власник токена змінюється, але крієйтор завжди залишається тою самою адресою *(Рисунок 26)*.

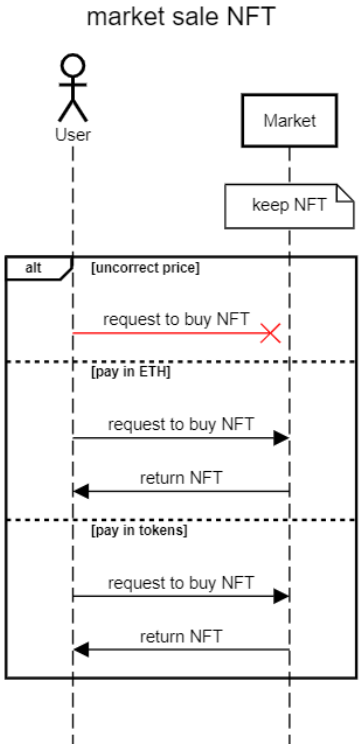

Рисунок 26 (market NFT sale functionality)

Продаж NFT означає, що токен переходить від маркета до нового власника. Цей крок є частиною продажу токена крієйтором до покупки кінцевим користувачем. В даному випадку маркетплейс виступає посередником *(Рисунок 27)*.

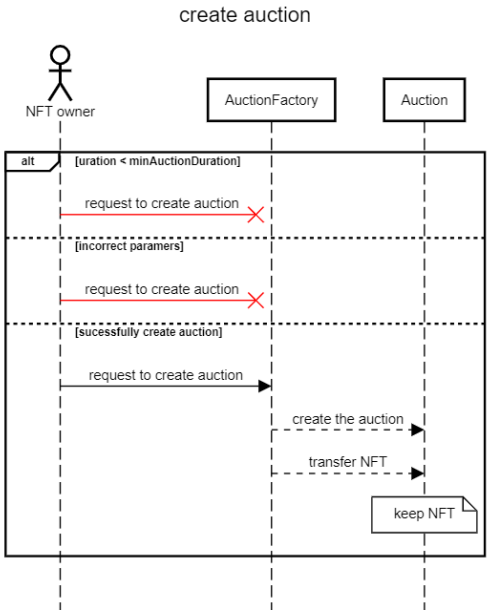

Рисунок 27 (create auction functionality)

Власник NFT токена може створити аукціон для продажу. Сам аукціон створюється через паттерн фабрика, так як аукціонів може бути багато і всі вони будуть мати одні й ті самі властивості, лише значення можуть відрізнятися *(Рисунок 28)*.

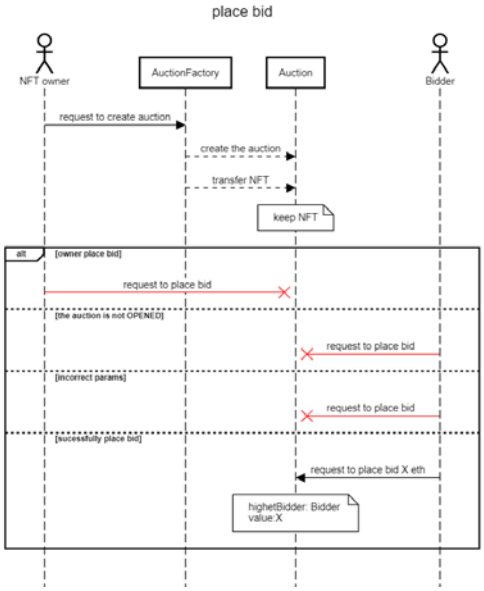

Рисунок 28 (Place bid)

Зареєстрований користувач може приймати участь в аукціоні, щоб купити бажаний токен.Для цього потрібно зробити ставку, яка приймається лище в ефірі. Якщо ця ставка більша ніж попередня, і більше проку ставки, то в блокчейні буде записаний новий претиндент на покупку *(Рисунок 29)*.

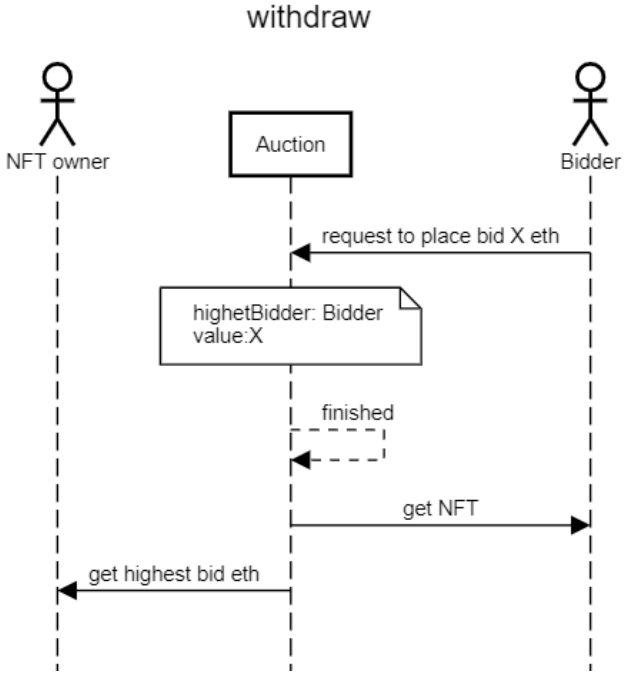

Рисунок 29 (withdraw functionality)

3.6. Опис файлів даних та інтерфейсу програми

Структура файлів і сигнатура смарт-контрактів відрізняється від інших мов програмування.

Об’єкти, такі як аукціон, інформація про NFT зберігаються у структурі, яка називається ***struct*** *(Рисунок 30).*

```
/**
 * @notice mft token added to the market info
 * @param id id
 * @param nftContract address of the nft contract
 * @param tokenId nft token
 * @param creator address of the creater
 * @param owner addreess of the owner
 * @param price amount price to buy
 * @param likes amount of likes
 * @param sold is nft token issold
 **/
struct MarketNFT {
    uint256 id;
    address nftContract;
    uint256 tokenId;
    address creator;
    address owner;
    uint256 price;
    uint256 priceInTokens;
    uint256 likes;
    bool sold;
}
```

Рисунок 30 (NFT structure)

Окрім цього також відрізняються структури даних, у Solidity немає великого вибору. Існують масиви та ***mapping***. Останній схожий на HashMap, представлений у вигляді ключа та значення. Ключем можуть бути всі типи, окрім *struct*, а значенням може виступати будь-який тип чи структура *(Рисунок 31)*

```
/// @notice auction id => auction info
mapping(uint256 => Auction) public auctions;
/// @notice auction id => AuctionInfo (address, bool)
mapping(uint256 => AuctionInfo) public auctionsInfo;
```

Рисунок 31 (mapping example)

Ще одним цікавим типом, що відрізняє Solidity від інших мов програмування, є івенти (***event***). Зазвичай вони викликаються в кінці методу. Параметри івентів записуються в логи транзакції. На ці івенти користувач мож підписатися або відписатися для відслідковування статусу транзакції, використавши web3.js *(Рисунок 32 -33)*

```
/**
 * @notice emitted when an auction is created
 * @param creator address of user who bids
 * @param startTime timestamp when the auction will be started
 * @param duration timestamp how much the auction will be
 * @param minIncrement the minimum increment for the bid
 * @param directBuyPrice the price for a direct buy
 * @param startPrice the starting price for the auction
 * @param nftAddress address of the nft
 * @param tokenId the id of the token
 **/
event CreatedAuction(
    address indexed creator,
    uint256 startTime,
    uint256 duration,
    uint256 minIncrement,
    uint256 directBuyPrice,
    uint256 startPrice,
    address indexed nftAddress,
    uint256 tokenId
);
```

Рисунок 32 (event example)

```
emit CreatedAuction(
    msg.sender,
    block.timestamp,
    _duration,
    _minIncrement,
    _directBuyPrice,
    _startPrice,
    _nftAddress,
    _tokenId
);
```

Рисунок 33 (event example in contract)

Ще однією цікавою перевагою Solidity є ***low-level*** функції. Вони використовуються для виклику функції одного контракту з іншого *(Рисунок 34)*

```
(bool success, bytes memory result) = auctionAddress.delegatecall(
    abi.encodeWithSignature("cancelAuction()")
);
```

Рисунок 34 (low-level function - delegatecall)

Їх виклик дозволений лише тоді, воли виклик через інтерфейс контракту з якихось причин неможливий. Наприклад, немає алреси контракту, або msg.sender повинен виступати початковий юзер. Для тразакції ефіру завжди використовується ***low-level*** функція ***call***. *(Рисунок 35)*

```
(bool success, ) = payable(creator).call{value: msg.value}("");
require(success, "Failed to transfer Ether");
```

Рисунок 35 (low-level function - call)

Для того, щоб розділити доступність функцій для різних користувачів, тобто зробити аунтентифікацію, використано підхід за допомогою ***modifier***. Ці перевірки встановлюються в сигнатурі функції і викникаються перед основним її функціоналом. Якщо умова викнується, модифікатор переходить до виконання тіла функції. *(Рисунок 36)*

```
modifier onlyLogged(address user) {
    require(issUserLogged(user), "ONLY_LOGIN_USER");
    _;
}
```

Рисунок 36 (modifier example)

Модифікатори можуть бути як кастомні, так і вже існуючі в бібліотеках. Найпопулярнішим таким модифікатором є ***onlyOwner*** з контракту ***Owner*** від бібліотеки *OpenZeppelin*. Він перевіряє чи є *msg.sender* власником контрату. *(Рисунок 37)*

```
/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    require(owner() == _msgSender(), "Ownable: caller is not the owner");
    _;
}
```

Рисунок 37 (modifier from Ownable contract)

Висновки

У результаті виконання курсової роботи було проаналізовано питання створення NFT та продажу їх на маркетплейсах та біржах. Окрім того, було розглянуто існуючі аналоги, їх функціонал та зручність використання. З огляду на це, було вибрано, які можливості децентралізованого веб-застосунку будуть найбільш корисними та зручними.

Класифікація існуючих блокчейнів допомогла зорієнтуватися, який краще вибрати з огляду на ціну газу за транзацію. Після цього було визначено який основний функціонал має підтримувати застосунок з огляду на потреби крієйтора, інвестора та власника самого маркетплейсу. Після визначення того, що це буде dApp, тобто децентралізований застосунок, було прийнято рішення, що він має використовувати блокчейн та IPFS для збереження даних. Зрештою, було розроблено архітектуру смарт-контрактів.

Практична частина була реалізована як веб-застосунок, який підтримує децентралізацію, тобто рішення Web3.0 де основним функціоналом є випуск NFT токенів та різні види його продажу.

Розроблений веб-застосунок є унікальним, адже розрахований, в першу чергу, на українців. У майбутньому функціонал системи може розширюватись в залежності від потреб користувачів, а саме можуть бути доданий Airdrop для привернення нових інвесторів, ширший вибір предметів продажу у вигляді цифрових активів, покращення функціоналу для зменшення ціни за транзакцію.

За реалізацією та майбутніми оновленнями можна стежити на сторінці проету на GitHub - https://github.com/Mariesnlk/MarketPlace-NFT-Avalanche-Polygon- .

Список використаної літератури

1. Adam Hayes What is a Binance? [Електронний ресурс] / Adam Hayes – 2022. – Режим доступу до ресурсу: https://www.investopedia.com/terms/b/blockchain.aspra%D1%97ni-zalisha%D1%94-bazhati-krashchogo
2. Jake Frankenfield What is a Decentralized Application(dApp)? [Електронний ресурс] / Jake Frankenfield – 2021. – Режим доступу до ресурсу: https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp
3. Adam Hayes What IS Peer-to-Peer(P2P) [Електронний ресурс] / Adam Hayes – 2021. – Режим доступу до ресурсу: *https://www.investopedia.com/terms/p/peertopeer-p2p-service.asp*
4. Jake Frankenfield What is Smart Contract? [Електронний ресурс] / Jake Frankenfield – 2022. – Режим доступу до ресурсу: *https://www.investopedia.com/terms/s/smart-contracts.asp*
5. Rakesh Sharma What is Non-Fungible Token(NFT)? [Електронний ресурс] / Rakesh Sharma – 2022. – Режим доступу до ресурсу: *https://www.investopedia.com/non-fungible-tokens-nft-5115211*
6. Nathan Reiff What is Avalanche? [Електронний ресурс] / Nathan Reiff – 2022. – Режим доступу до ресурсу: *https://www.investopedia.com/avalanche-avax-definition-5217374*
7. Nathan Reiff What is Avalanche? [Електронний ресурс] / Nathan Reiff – 2022. – Режим доступу до ресурсу: *https://www.investopedia.com/avalanche-avax-definition-5217374*
8. Connor Brooke Best NFT Marketplace [Електронний ресурс] / Connor Brooke – 2022. – Режим доступу до ресурсу: *https://www.business2community.com/nft/best-marketplaces*

9. Langston Thomas A guide to CryptoPunks NFT: What are CryptoPunks?[Електронний ресурс] / Langston Thomas – 2022. – Режим доступу до ресурсу: https://nftnow.com/guides/cryptopunks-guide/

10. Shobhit Seth Public, Private, Permissioned Blockchains Compared[Електронний ресурс] / Shobhit Seth – 2021. – Режим доступу до ресурсу: https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/

11. Dimitar Bogdanov Top Blockchain Platform foe development in 2021[Електронний ресурс] / Dimitar Bogdanov – 2021. – Режим доступу до ресурсу: https://limechain.tech/blog/top-blockchain-platforms-2021/

12. Grant Bartel What is Dapp? A guide to Ethereum Dapps[Електронний ресурс] / Grant Bartel – 2020. – Режим доступу до ресурсу:https://www.freecodecamp.org/news/what-is-a-dapp-a-guide-to-ethereum-dapps/

Додаток А. Додаткові ілюстрації

Додаток Б. Серверний код

Vendor.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "./interfaces/IVendor.sol";

contract Vendor is IVendor, Ownable, ReentrancyGuard {
    using SafeERC20 for IERC20;
    // @noticw token address
    IERC20 private immutable token;
    // @noticw default price for 1 Token
    uint256 public price = 0.008 ether;
```

```
constructor(address token_) {
    require(token_ != address(0), "Vendor: Invalid token address");
    token = IERC20(token_);
}

/**
 * @notice get tokens by caller depending on how much wei was sent to contract
 * @dev return tokens amount that buy msg.sender
 * @dev before buy transfer tokens from owner to vendor contract address
 */
function buyTokens()
    external
    payable
    override
    nonReentrant
    returns (uint256 amounts)
{
    require(msg.value > 0, "Vendor: value cannot be low or equal zero");

    uint256 amountToBuy = msg.value / price;
    uint256 returnAmounts = msg.value % price;
    uint256 amountToPay = msg.value - returnAmounts;

    require(
        token.balanceOf(address(this)) >= amountToBuy,
        "Vendor: contract has not enough tokens in its balance"
    );

    token.safeTransfer(msg.sender, amountToBuy);

    emit BoughtToken(msg.sender, amountToPay, amountToBuy);

    if (returnAmounts != 0) {
        (bool success, ) = msg.sender.call{value: returnAmounts}("");
        require(success, "Vendor: Failed to return unused amounts");
    }

    return amountToBuy;
}

/**
 * @notice transfer amount of tokens back to the contract
 * @param _amount tokens amount that recipient wants to buy
 * @dev approve vendor contract
 */
function sellTokens(uint256 _amount) external override nonReentrant {
    require(
        _amount > 0,
        "Vendor: specify an amount of token greater than zero"
    );
```

```
    uint256 userBalance = token.balanceOf(msg.sender);
    require(
        userBalance >= _amount,
        "Vendor: your balance is lower than the amount of tokens you want to sell"
    );

    uint256 amountToTransfer = _amount * price;
    uint256 ownerBalance = address(this).balance;

    require(
        ownerBalance >= amountToTransfer,
        "Vendor: contract has not enough funds to accept the sell request"
    );

    token.safeTransferFrom(msg.sender, address(this), _amount);

    emit SoldToken(msg.sender, _amount, amountToTransfer);

    (bool success, ) = msg.sender.call{value: amountToTransfer}("");
    require(success, "Vendor: failed to send ETH to the user");
}

/**
 * @notice - set price in wei for buy tokens
 * @param _price - token price
 * @return _newPrice - reurn new setted price
 */
function setPrice(uint256 _price)
    external
    override
    onlyOwner
    returns (uint256 _newPrice)
{
    require(_price > 0, "Vendor: price cannot be low or equal zero");
    price = _price;

    emit SettedPrice(price);

    return price;
}

/**
 * @notice allow the owner of the contract to withdraw eth
 */
function withdraw() external override onlyOwner {
    uint256 contractBalance = address(this).balance;
    require(
        contractBalance > 0,
        "Vendor: contract has not balance to withdraw"
    );
```

```
        (bool success, ) = msg.sender.call{value: contractBalance}("");
        require(
            success,
            "Vendor: failed to send user balance back to the owner"
        );
    }
}
```

Token.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/// @title Token contract with ERC20 standart
contract Token is ERC20 {
    constructor(
        string memory name,
        string memory symbol,
        uint256 supply
    ) ERC20(name, symbol) {
        _mint(msg.sender, supply);
    }
}
```

NFT.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

/// @title NFT contract with ERC721 standart
contract NFT is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    /// @notice market contract address
    address private market;
    /**
     * @notice emitted when NFT is minted
     * @param minter address of creater
     * @param tokenURI nft url
     * @param nftId nft id
     **/
    event MintedNFT(address indexed minter, string tokenURI, uint256 nftId);
```

```
    constructor(address marketplaceAddress) ERC721("KryptoPaintz", "KPAINTZ") {
        require(
            marketplaceAddress != address(0),
            "NFT: invalid market address"
        );
        market = marketplaceAddress;
    }

    function mintToken(string memory nftURI) external returns (uint256) {
        _tokenIds.increment();
        uint256 newNftId = _tokenIds.current();
        //passing id and url
        _mint(msg.sender, newNftId);
        //set the token URI: id and url
        _setTokenURI(newNftId, nftURI);
        //give the marketplace the approval to transact between users
        setApprovalForAll(market, true);

        emit MintedNFT(msg.sender, nftURI, newNftId);

        return newNftId;
    }
}
```

KPMarket.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "./interfaces/IKPMarket.sol";
import "./users/UserRegistration.sol";

/// @title Market contract
contract KPMarket is IKPMarket, UserRegistration, ReentrancyGuard {
    using Counters for Counters.Counter;
    /// @notice count of all nfts
    Counters.Counter private tokenIds;
    /// @notice count of sold nfts
    Counters.Counter private tokenSold;
    /// @notice NFT contract
    IERC721 public nftContract;
    /// @notice Token contract
    IERC20 public token;
```

```
    /// @notice UserRegistration contract to allowed only registered and login users to use some functions
    IUserRegistration public usersRegistration;
    /// @notice price that permit add nft to the market
    uint256 public listingPrice = 0.045 ether;
    /// @notice tokenId return which MarketToken - fetch which one it is
    mapping(uint256 => MarketNFT) public idToMarketToken;
    /// @notice token price to 1 ETH
    uint256 public TOKENS_PRICE = 50;

    //// TODO ERC2981: add implementation of the royalty standard, and the respective extensions for ERC721 and ERC1155

    constructor(address _token) {
        require(_token != address(0), "KPMarket: invalid token address");
        token = IERC20(_token);
    }

    /**
     * @notice set address to NFT contract
     */
    function setNFTContract(address _nftContract) external onlyOwner {
        require(
            _nftContract != address(0),
            "KPMarket: invalid nftContract address"
        );
        nftContract = IERC721(_nftContract);
    }

    /**
     * @notice set new price for ERC20 tokens
     * @dev only contract owner can do
     */
    function setTokensPrice(uint256 _price) external onlyOwner {
        require(_price > 0, "INVALD_VALUE");
        TOKENS_PRICE = _price;
    }

    /**
     * @notice set new value for listing price
     * @dev only owner can update
     */
    function updateListingPrice(uint256 _listingPrice)
        external
        payable
        onlyOwner
    {
        require(
            _listingPrice > 0,
            "KPMarket: listing price value should be more than 0."
        );
        listingPrice = _listingPrice;
```

```
}

/**
 * @notice add NFT to market
 * @param nftId id of the NFT
 * @param price price to sell
 * @dev function to put item up for sale
 * @dev only logged user can create market
 */
function createMarketNFT(uint256 nftId, uint256 price)
    external
    payable
    override
    nonReentrant
    returns (bool)
{
    require(price > 0, "KPMarket: price must be at least one wei");
    require(
        msg.value == listingPrice,
        "KPMarket: price must be equal to listening price"
    );

    tokenIds.increment();
    uint256 itemId = tokenIds.current();
    uint256 priceInTokens = _calculatePriceInTokens(price);

    //putting nft up for sale
    idToMarketToken[itemId] = MarketNFT(
        itemId,
        address(nftContract),
        nftId,
        msg.sender,
        address(0),
        price,
        priceInTokens,
        0,
        false
    );

    // transer nft to market contract
    nftContract.transferFrom(msg.sender, address(this), nftId);

    emit MarketTokenCreated(
        itemId,
        address(nftContract),
        nftId,
        msg.sender,
        address(0),
        price,
        priceInTokens,
        0,
```

```
        false
    );

    return true;
}

/**
 * @notice reselle NFT by ETH or by ERC20 tokens
 * @param nftId id of the NFT
 * @param price to resell
 * @dev only NFT owner can resalle
 */
function reselleNFT(uint256 nftId, uint256 price)
    external
    payable
    override
    returns (bool)
{
    require(
        idToMarketToken[nftId].owner == msg.sender,
        "KPMarket: only item owner can perform this operation"
    );
    require(
        msg.value == listingPrice,
        "KPMarket: price must be equal to listing price"
    );
    idToMarketToken[nftId].sold = false;
    idToMarketToken[nftId].price = price;
    idToMarketToken[nftId].creator = msg.sender;
    // resale to market
    idToMarketToken[nftId].owner = address(this);

    tokenIds.decrement();

    nftContract.transferFrom(msg.sender, address(this), nftId);

    return true;
}

/**
 * @notice create selling of the NFT (user buy it) to market
 * @param nftId id of the NFT
 * @param tokens amount o MRSNLK tokens
 */
function marketSaleNFT(uint256 nftId, uint256 tokens)
    external
    payable
    override
    nonReentrant
    returns (bool)
{
```

```
    uint256 price = idToMarketToken[nftId].price;
    uint256 tokenId = idToMarketToken[nftId].tokenId;
    address creator = idToMarketToken[nftId].creator;

    require(
        msg.value == price,
        "KPMarket: please submit the asking price in order to continue"
    );

    // payable(owner).transfer(listingPrice);
    (bool result, ) = payable(msg.sender).call{value: listingPrice}("");
    require(result, "Failed to transfer Ether");

    if (tokens > 0) {
        token.transfer(creator, tokens);
    } else {
        // idToMarketToken[nftId].creator.transfer(msg.value);
        (bool success, ) = payable(creator).call{value: msg.value}("");
        require(success, "Failed to transfer Ether");
    }

    nftContract.transferFrom(address(this), msg.sender, tokenId);

    idToMarketToken[nftId].owner = msg.sender;
    idToMarketToken[nftId].sold = true;

    tokenSold.increment();

    return true;
}

/**
 * @notice get info list of unsoled NFT
 * @dev fetchMarketTokens
 */
function getUnsoldNFTs()
    external
    view
    override
    returns (MarketNFT[] memory)
{
    uint256 nftCount = tokenIds.current();
    uint256 unsoldItemCount = tokenIds.current() - tokenSold.current();
    uint256 currentIndex = 0;

    MarketNFT[] memory items = new MarketNFT[](unsoldItemCount);

    for (uint256 i = 0; i < nftCount; i++) {
        //if it is unsold item
        if (idToMarketToken[i + 1].owner == address(0)) {
            uint256 currentId = i + 1;
```

```
            MarketNFT storage currentItem = idToMarketToken[currentId];
            items[currentIndex] = currentItem;
            currentIndex++;
        }
    }

    return items;
}

/**
 * @notice get info list of NFT that owns msg.sender
 * @dev fetchMyNFTs
 */
function getMyNFTs()
    external
    view
    override
    returns (MarketNFT[] memory)
{
    uint256 totalNFTCount = tokenIds.current();
    //counter for each individual user
    uint256 itemCount = 0;
    uint256 currentIndex = 0;

    for (uint256 i = 0; i < totalNFTCount; i++) {
        if (idToMarketToken[i + 1].owner == msg.sender) {
            itemCount++;
        }
    }

    MarketNFT[] memory items = new MarketNFT[](itemCount);

    for (uint256 i = 0; i < totalNFTCount; i++) {
        if (idToMarketToken[i + 1].owner == msg.sender) {
            uint256 currentId = idToMarketToken[i + 1].id;
            MarketNFT storage currentItem = idToMarketToken[currentId];
            items[currentIndex] = currentItem;
            currentIndex++;
        }
    }

    return items;
}

/**
 * @notice get info list of NFT that only added to market
 * @dev fetchItemsCreated
 */
function getOnlyCreatedNFTs()
    external
    view
```

```
        override
        returns (MarketNFT[] memory)
    {
        uint256 totalNFTCount = tokenIds.current();
        uint256 itemCount = 0;
        uint256 currentIndex = 0;

        for (uint256 i = 0; i < totalNFTCount; i++) {
            if (idToMarketToken[i + 1].creator == msg.sender) {
                itemCount++;
            }
        }

        MarketNFT[] memory items = new MarketNFT[](itemCount);

        for (uint256 i = 0; i < totalNFTCount; i++) {
            if (idToMarketToken[i + 1].creator == msg.sender) {
                uint256 currentId = idToMarketToken[i + 1].id;
                MarketNFT storage currentItem = idToMarketToken[currentId];
                items[currentIndex] = currentItem;
                currentIndex++;
            }
        }

        return items;
    }

    function _calculatePriceInTokens(uint256 _price)
        internal
        view
        returns (uint256)
    {
        return _price * TOKENS_PRICE;
    }
}
```

AuctionFactory.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "./interfaces/IAuctionFactory.sol";
import "./interfaces/IAuction.sol";
import "./Auction.sol";
import "./users/UserRegistration.sol";
```

```
/// @title AuctionFactory contract
contract AuctionFactory is IAuctionFactory, UserRegistration {
    using Counters for Counters.Counter;
    Counters.Counter private auctionIds;
    /// @notice UserRegistration contract to allowed only registered and login users to use some functions
    IUserRegistration public usersRegistration;
    /// @notice min value auction duration
    uint256 public minAuctionDuration = 5 minutes;
    /// @notice auction id => auction info
    mapping(uint256 => Auction) public auctions;
    /// @notice auction id => AuctionInfo (address, bool)
    mapping(uint256 => AuctionInfo) public auctionsInfo;

    /**
     * @notice set new minumum value of the auction duration
     * @param auctionDuration minumum value of the auction duration
     * @dev only owner can set
     */
    function setMinAuctionDuration(uint256 auctionDuration) external onlyOwner {
        require(
            auctionDuration > 0,
            "AuctionFactory: invalid min auction duration"
        );
        minAuctionDuration = auctionDuration;
    }

    /**
     * @notice creating auction
     * @param _duration timestamp when the auction will be finished
     * @param _minIncrement the minimum increment for the bid
     * @param _directBuyPrice the price for a direct buy
     * @param _startPrice the starting price for the auction
     * @param _nftAddress address of the nft
     * @param _tokenId the id of the token
     **/
    function createAuction(
        uint256 _duration,
        uint256 _minIncrement,
        uint256 _directBuyPrice,
        uint256 _startPrice,
        address _nftAddress,
        uint256 _tokenId
    ) external override returns (bool) {
        require(
            _duration >= minAuctionDuration,
            "Auction: invalid auction duration"
        );
        uint256 auctionId = auctionIds.current();
        auctionIds.increment();
        Auction auction = new Auction(
            msg.sender,
```

```
        _duration,
        _minIncrement,
        _directBuyPrice,
        _startPrice,
        _nftAddress,
        _tokenId
    );

    IERC721 nftToken = IERC721(_nftAddress);
    nftToken.transferFrom(msg.sender, address(auction), _tokenId);
    auctions[auctionId] = auction;
    auctionsInfo[auctionId].auction = address(auction);
    auctionsInfo[auctionId].isExists = true;

    emit CreatedAuction(
        msg.sender,
        block.timestamp,
        _duration,
        _minIncrement,
        _directBuyPrice,
        _startPrice,
        _nftAddress,
        _tokenId
    );

    return true;
}

/**
 * @notice deleting auction
 * @dev only owner of the auction can delete
 * @param auctionId address of the auction that will be deleted
 **/
function deleteAuction(uint256 auctionId) external override returns (bool) {
    require(auctionsInfo[auctionId].isExists, "ALREADY_DELETED");

    address auctionAddress = auctionsInfo[auctionId].auction;

    (bool success, bytes memory result) = auctionAddress.delegatecall(
        abi.encodeWithSignature("cancelAuction()")
    );

    require(!success, "FAILED_DELEDATECALL");

    // IAuction(auctionsInfo[auctionId].auction).cancelAuction();

    delete auctionsInfo[auctionId];
    delete auctions[auctionId];

    return true;
}
```

```
/**
 * @notice get a list of all auctions
 */
function getAuctions()
    external
    view
    override
    returns (address[] memory _auctions)
{
    uint256 getAuctionsIds = auctionIds.current();
    _auctions = new address[](getAuctionsIds);
    for (uint256 i = 0; i < getAuctionsIds; i++) {
        _auctions[i] = address(auctions[i]);
    }
    return _auctions;
}

/**
 * @notice get the information of each auction address
 */
function getAuctionsInfo(address[] calldata _auctionsList)
    external
    view
    override
    returns (
        uint256[] memory directBuy,
        address[] memory holder,
        uint256[] memory highestBid,
        uint256[] memory tokenIds,
        uint256[] memory endTime,
        uint256[] memory startPrice,
        uint256[] memory auctionState
    )
{
    directBuy = new uint256[](_auctionsList.length);
    holder = new address[](_auctionsList.length);
    highestBid = new uint256[](_auctionsList.length);
    tokenIds = new uint256[](_auctionsList.length);
    endTime = new uint256[](_auctionsList.length);
    startPrice = new uint256[](_auctionsList.length);
    auctionState = new uint256[](_auctionsList.length);

    for (uint256 i = 0; i < _auctionsList.length; i++) {
        directBuy[i] = Auction(auctions[i]).directBuyPrice();
        holder[i] = Auction(auctions[i]).creator();
        highestBid[i] = Auction(auctions[i]).maxBid();
        tokenIds[i] = Auction(auctions[i]).tokenId();
        endTime[i] = Auction(auctions[i]).endTime();
        startPrice[i] = Auction(auctions[i]).startPrice();
        auctionState[i] = uint256(Auction(auctions[i]).getAuctionState());
```

```
        }

        return (
            directBuy,
            holder,
            highestBid,
            tokenIds,
            endTime,
            startPrice,
            auctionState
        );
    }
}
```

Auction.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "./interfaces/IAuction.sol";
import "./users/UserRegistration.sol";

/// @title Auction contract
contract Auction is IAuction, UserRegistration, ReentrancyGuard {
    /// @notice the block timestamp which marks the start of the auction
    uint256 public startTime;
    /// @notice the block timestamp which marks the end of the auction (in seconds)
    uint256 public endTime;
    /// @notice the address of the maximum bidder
    address public maxBidder;
    /// @notice the maximum bid
    uint256 public maxBid;
    /// @notice is the the auction is cancelled
    bool public isCancelled;
    /// @notice is the auction ended due to direct buy
    bool public isDirectBuy;
    /// @notice the minimum increment for the bid
    uint256 public minIncrement;
    /// @notice the address of the auction creator
    address public creator;
    /// @notice the id of the token
    uint256 public tokenId;
    /// @notice the price for a direct buy
    uint256 public directBuyPrice;
    /// @notice the starting price for the auction
    uint256 public startPrice;
    /// @notice NFT address
```

```
IERC721 public nft;
/// @notice array of the bids made by the bidders
Bid[] public bids;
/// @notice UserRegistration contract to allowed only registered and login users to use some functions
IUserRegistration public usersRegistration;

constructor(
    address _creator,
    uint256 _duration,
    uint256 _minIncrement,
    uint256 _directBuyPrice,
    uint256 _startPrice,
    address _nftAddress,
    uint256 _tokenId
) {
    require(
        msg.sender != address(0),
        "Auction: Creator of the auction can't be zero address"
    );
    require(
        _minIncrement > 0,
        "Auction: min increment for bid cannot be less than 0"
    );
    require(
        _directBuyPrice > 0,
        "Auction: buy price cannot be less than 0"
    );
    require(
        _startPrice < _directBuyPrice,
        "Auction: start price is smaller than direct buy price"
    );
    require(
        _nftAddress != address(0),
        "Auction: NFT address can't be zero address"
    );
    require(_tokenId > 0, "Auction: token id cannot be less than 0");

    creator = _creator;
    startTime = block.timestamp;
    endTime = startTime + _duration;
    minIncrement = _minIncrement;
    directBuyPrice = _directBuyPrice;
    startPrice = _startPrice;
    nft = IERC721(_nftAddress);
    tokenId = _tokenId;
    maxBidder = _creator;
}

modifier onlyLogin() {
    require(usersRegistration.checkIsUserLogged(), "ONLY_LOGIN_USER");
    _;
```

```
}

function setRegistrationContract(address _usersRegistration) external {
    require(_usersRegistration != address(0), "INVALID_ADDRESS");
    usersRegistration = IUserRegistration(_usersRegistration);
}

/**
 * @notice Place a bid on the auction
 **/
function placeBid() external payable override nonReentrant returns (bool) {
    require(
        msg.sender != creator,
        "Auction: the bidder cannot be the auction creator"
    );
    require(
        getAuctionState() == AuctionState.OPEN,
        "Auction: the auction must be open"
    );
    require(
        msg.value >= startPrice,
        "Auction: the bid must be greater than the started price"
    );
    require(
        msg.value > maxBid + minIncrement,
        "Aucton: the bid must be greater than the highest bid + minimum bid increment"
    );

    address lastHightestBidder = maxBidder;
    uint256 lastHighestBid = maxBid;
    maxBid = msg.value;
    maxBidder = msg.sender;

    if (msg.value >= directBuyPrice) {
        isDirectBuy = true;
    }

    bids.push(Bid(msg.sender, msg.value));

    if (lastHighestBid != 0) {
        // refund the previous bid to the previous highest bidder
        (bool success, ) = payable(lastHightestBidder).call{
            value: lastHighestBid
        }("");
        require(success, "Failed to transfer Ether");
    }

    emit NewBid(msg.sender, msg.value);

    return true;
}
```

```
/**
 * @notice Withdraw the token after the auction is over
 **/
function withdrawToken() external override {
    require(
        getAuctionState() == AuctionState.ENDED ||
            getAuctionState() == AuctionState.DIRECT_BUY,
        "Auction: the auction must be ended by either a direct buy or timeout"
    );
    require(
        msg.sender == maxBidder,
        "Auction: only the highest bidder can withdraw the token"
    );

    nft.transferFrom(address(this), maxBidder, tokenId);

    emit WithdrawToken(maxBidder);
}

/**
 * @notice Withdraw the funds after the auction is over
 **/
function withdrawFunds() external override nonReentrant onlyLogin {
    require(
        getAuctionState() == AuctionState.ENDED ||
            getAuctionState() == AuctionState.DIRECT_BUY,
        "Auction: The auction must be ended by either a direct buy or timeout"
    );
    require(
        msg.sender == creator,
        "Auction: Only the auction creator can withdraw the funds"
    );

    (bool success, ) = payable(msg.sender).call{value: maxBid}("");
    require(success, "Failed to transfer Ether");

    emit WithdrawFunds(msg.sender, maxBid);
}

/**
 * @notice Cancel the auction
 **/
function cancelAuction() external override returns (bool) {
    require(
        msg.sender == creator,
        "Auction: only the auction creator can cancel the auction"
    );
    require(
        getAuctionState() == AuctionState.OPEN,
        "Auction: the auction must be open"
```

```
        );
        require(
            maxBid == 0,
            "Auction: the auction must not be cancelled if there is a bid"
        );

        isCancelled = true;

        // Transfer the NFT token to the auction creator
        nft.transferFrom(address(this), creator, tokenId);

        emit AuctionCancelled();

        return true;
    }

    /**
     * @notice Get a list of all bids and addresses
     **/
    function allBids()
        external
        view
        override
        returns (address[] memory, uint256[] memory)
    {
        address[] memory addressesOfBidders = new address[](bids.length);
        uint256[] memory bidsPrices = new uint256[](bids.length);
        for (uint256 i = 0; i < bids.length; i++) {
            addressesOfBidders[i] = bids[i].sender;
            bidsPrices[i] = bids[i].bid;
        }
        return (addressesOfBidders, bidsPrices);
    }

    /**
     * @notice Get the auction state
     **/
    function getAuctionState() public view override returns (AuctionState) {
        if (isCancelled) {
            return AuctionState.CANCELLED;
        } else if (isDirectBuy) {
            return AuctionState.DIRECT_BUY;
        } else if (block.timestamp >= endTime) {
            return AuctionState.ENDED;
        } else {
            return AuctionState.OPEN;
        }
    }
}
```

UserRegistration.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "./interfaces/IUserRegistration.sol";

contract UserRegistration is IUserRegistration, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private usersIds;
    /// @notice address of the wallet => struct of user`s info
    mapping(address => UserDetail) public users;

    modifier onlyLogged(address user) {
        require(issUserLogged(user), "ONLY_LOGIN_USER");
        _;
    }

    /**
     * @notice allows user to register in dapp to get access to merket
     * @param _username nickname to login
     * @param _password password to login
     * @param _bio some more info about user (optional)
     * @dev valigation of username and password in FE, bio field can be empty
     **/
    function register(
        string memory _username,
        string memory _password,
        string memory _bio
    ) external override returns (bool) {
        require(msg.sender != address(0), "ZERO_ADDRESS");
        require(
            keccak256(abi.encodePacked(users[msg.sender].username)) !=
                keccak256(abi.encodePacked(_username)),
            "ALREADY_REGISTERED"
        );

        usersIds.increment();
        users[msg.sender].id = usersIds.current();
        users[msg.sender].username = _username;
        users[msg.sender].password = _password;
        users[msg.sender].bio = _bio;
        users[msg.sender].isUserLoggedIn = false;

        emit Registered(usersIds.current(), _username, msg.sender);

        return true;
    }

    /**
     * @notice allows registered user log in dapp
```

```
 * @param _username nickname to login
 * @param _password password to login
 **/
function login(string memory _username, string memory _password)
    external
    override
    returns (bool)
{
    if (
        (keccak256(abi.encodePacked(users[msg.sender].username)) ==
            keccak256(abi.encodePacked(_username))) &&
        (keccak256(abi.encodePacked(users[msg.sender].password)) ==
            keccak256(abi.encodePacked(_password)))
    ) {
        users[msg.sender].isUserLoggedIn = true;
    }

    return users[msg.sender].isUserLoggedIn;
}

/**
 * @notice checking if the user is logged in or not
 */
function checkIsUserLogged() external view override returns (bool) {
    return users[msg.sender].isUserLoggedIn;
}

/**
 * @notice logout from dapp, all pages are invalid
 */
function logout() external override {
    users[msg.sender].isUserLoggedIn = false;
}

/**
 * @notice get user username and bio
 * @dev this info will be display in the personal page
 */
function getUser()
    external
    view
    override
    returns (string memory, string memory)
{
    require(users[msg.sender].isUserLoggedIn, "NOT_LOGIN");
    return (users[msg.sender].username, users[msg.sender].bio);
}

/**
 * @notice get a list of all registered users
 */
```

```
    function getAllUsers()
        external
        pure
        override
        returns (address[] memory _users)
    {}

    /**
     * @notice checking if the user is logged in or not
     * @param user user address to check
     */
    function issUserLogged(address user)
        private
        view
        returns (bool)
    {
        return users[user].isUserLoggedIn;
    }
}
```