Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Застосування Solana Token Extensions для розробки NFT з унікальними властивостями

Текстова частина до курсової роботи
за спеціальністю «Комп’ютерні науки» - 122

Керівник курсової роботи
Гороховський К.С.
_____ (Підпис)
“___” _____ 2025 року
Виконав студент
КН-3 Рубан А.О
“___” _____ 2025 року

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
Жежерун Олександр Петрович
“___” _____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Студента Рубана Антона Олеговича факультету інформатики 3 курсу
ТЕМА: Застосування Solana Token Extensions для розробки NFT
з унікальними властивостями

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Вступ
3. Розділ 1. Блокчейн-платформа Solana
4. Розділ 2. Технологія Solana Token Extensions
5. Розділ 2. Розробка додатку
6. Висновки
7. Список використаних джерел
8. Додатки

Дата видачі „___” _____ 2025 р.

Керівник_____ (підпис)

Завдання отримав _____ (підпис)

Календарний план виконання роботи

№ з/п	Назва Етапу	Термін Виконання	Примітка
1	Вибір теми	07.10.2024	
2	Затвердження теми з керівником	16.10.2024	
3	Дослідження обраної технології. Вивчення інструментів розробки	31.01.2025	
4	Розробка застосунку	30.03.2025	
5	Написання текстової частини	20.04.2025	
6	Захист курсової роботи	15.05.2025	

Студент Рубан А.О.

Керівник Гороховський К.С. “_____” _____ 2025 р.

Зміст

Перелік прийнятних скорочень

NFT — невзаємозамінний токен;

SPL — Solana Program Library;

URI — уніфікований ідентифікатор ресурсу;

PoH — Proof of History (доказ історії);

PoS — Proof of Stake (доказ володіння);

SDK — Software Development Kit (набір інструментів для розробки);

UI — User Interface (користувацький інтерфейс);

XP — досвід (Experience Points);

WNS — Wen New Standard;

SPL Token — стандарт токенів у мережі Solana;

IPFS — InterPlanetary File System (міжпланетна файлова система);

JSON — JavaScript Object Notation (формат обміну даними);

Анотація

Метою курсової роботи є розробка застосунку для створення та керування невзаємозамінними токенами (NFT) з унікальними властивостями на базі блокчейну Solana. У роботі було реалізовано проєкт NFT-персонажів із використанням Solana Token Extensions, що забезпечує можливість додавання динамічних атрибутів і налаштувань метаданих без зміни основної структури токена. Для реалізації рішення використовувались можливості Solana Token-2022 та технології взаємодії з блокчейном Solana. У межах проєкту досліджено основи технології блокчейн Solana, архітектуру Token Extensions та практичні аспекти розробки та взаємодії з NFT у середовищі Solana.

Вступ

Станом на 2025 рік ринок NFT продовжує активно розвиватися, а технології блокчейн стають основою для створення нових цифрових активів із унікальними властивостями. Платформа Solana виділяється серед інших завдяки своїй високій пропускній здатності, низьким комісіям і підтримці інноваційних рішень для розробників.

Одним із таких рішень є Solana Token Extensions — набір розширень для стандарту токенів, що дозволяє реалізовувати функціональні можливості, які раніше вимагали створення окремих смарт-контрактів. З їх допомогою можна створювати NFT із динамічними атрибутами, правами доступу, що відкриває нові горизонти у світі децентралізованих застосунків.

Метою курсової роботи є розробка застосунку для створення NFT з унікальними властивостями на базі Solana Token Extensions. У межах роботи було поставлено завдання:

- вивчити технологічні основи платформи Solana та Token Extensions;
- реалізувати створення NFT із розширеними метаданими та динамічними атрибутами;
- забезпечити можливість безпечної взаємодії з NFT через клієнтський застосунок;
- оцінити переваги та обмеження використання Token Extensions для практичної розробки.

Під час виконання курсової роботи було проведено дослідження можливостей Solana Token-2022, особливостей роботи з NFT у блокчейні Solana та реалізовано прикладний проєкт, який демонструє переваги використання розширених функціональностей для цифрових активів.

РОЗДІЛ 1. БЛОКЧЕЙН-ПЛАТФОРМА SOLANA

1.1. Технологічні основи платформи Solana

Solana є високопродуктивною блокчейн-платформою, яка створена для забезпечення масштабованої роботи децентралізованих застосунків і обігу криптоактивів. Її розробка розпочалася у 2017 році з ініціативи Анатолія Яковенко, а основна мережа була офіційно запущена у 2020 році. Унікальною особливістю Solana є консенсусний механізм Proof of History (PoH), який дозволяє упорядковувати транзакції без необхідності їхнього традиційного підтвердження, що забезпечує надзвичайно високу пропускну здатність мережі до 65 000 транзакцій на секунду при збереженні мінімальних комісій. [1]

Поєднання механізмів Proof of History та Proof of Stake дозволяє Solana підтримувати безпечну, масштабовану та енергоефективну інфраструктуру, що є основою для розвитку фінансових сервісів, ігрових платформ та NFT-проєктів. Архітектура мережі орієнтована на розпаралелювання обробки транзакцій і мінімізацію затримок, що робить Solana однією з найшвидших децентралізованих систем у світі.

1.2 Переваги Solana для NFT-проєктів

Однією з ключових причин популярності Solana серед розробників NFT є швидкість обробки транзакцій. Завдяки оптимізованій архітектурі, підтвердження угоди відбувається за частки секунди. Комісії за створення і передавання NFT залишаються мінімальними, що робить мережу доступною навіть для невеликих проєктів. Висока масштабованість дає змогу обслуговувати велику кількість користувачів без зниження продуктивності. [2]

Особливої уваги заслуговує відкритість Solana до інновацій. Введення стандарту Token Extensions значно розширює можливості NFT на базі Solana, дозволяючи створювати активи з динамічними властивостями, змінними атрибутами та додатковими функціональними можливостями без потреби перевипуску токена.

Екосистема NFT на Solana активно розвивається, демонструючи приклади успішних колекцій, таких як Degenerate Ape Academy, Aurory та Solana Monkey Business.

Рис. 1.1 – Приклад популярної колекції NFT на Solana – "Solana Monkey Business" [3]

1.3. Архітектурні особливості

Інноваційна архітектура Solana складається з восьми ключових компонентів, які разом забезпечують високу продуктивність та масштабованість мережі. Proof of History формує криптографічну часову шкалу для упорядкування транзакцій. Консенсусний алгоритм Tower BFT базується на цій шкалі й дозволяє мінімізувати затримки під час досягнення згоди між учасниками мережі. Протокол Turbine оптимізує розповсюдження даних, забезпечуючи стабільну передачу інформації навіть при великих обсягах трафіку.

Gulf Stream реалізує механізм переадресації транзакцій без використання мемпула, що пришвидшує обробку заявок у мережі. Sealevel забезпечує паралельне виконання тисяч смарт-контрактів, дозволяючи максимально ефективно використовувати ресурси вузлів. Компонент Pipelining оптимізує обробку транзакцій за рахунок поділу процесу валідації на окремі етапи. [4]

Cloudbreak є горизонтально масштабованою базою даних для облікових записів користувачів, що дозволяє обробляти великий обсяг одночасних транзакцій. Система Archivers відповідає за розподілене зберігання історичних даних у мережі, забезпечуючи їхню доступність без надмірного навантаження на основні валідатори.

Ці технічні інновації забезпечують виняткову продуктивність Solana та її конкурентоспроможність серед інших блокчейнів, таких як Ethereum та Cardano.

1.4. Напрямки застосування

Solana вже сьогодні демонструє високий потенціал для застосування в різних сферах. Одним із ключових напрямків є децентралізовані фінанси (DeFi), де висока пропускна здатність і низькі транзакційні комісії дозволяють реалізовувати безпечні й ефективні операції, зокрема кредитування, децентралізовані біржі та токенізовані активи.

Іншою важливою сферою є ігрова індустрія. Завдяки можливості швидкої обробки тисяч транзакцій у режимі реального часу Solana ідеально підходить для великих геймінгових платформ, де критично важлива швидкість мережі.

Також Solana відкриває перспективи для соціальних мереж нового покоління, де взаємодія користувачів може базуватися на принципах токенізації та децентралізованого управління даними. Окремо варто відзначити можливості для розвитку кросчейн застосунків завдяки сумісності Solana із Ethereum через відповідні мостові рішення, що розширює межі інтеграції різних блокчейн-екосистем.

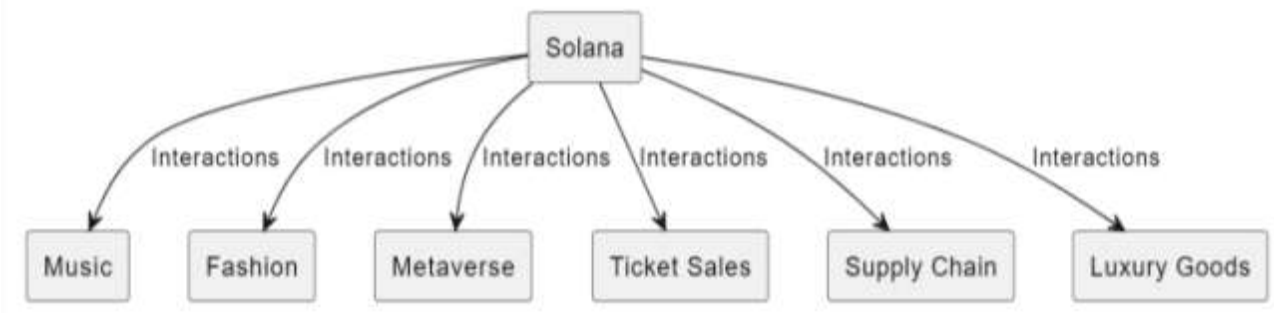

Рис. 1.2 – Напрямки застосування Solana [1]

1.5. Потенційні виклики

Попри значні переваги, Solana стикається із певними викликами. Високі апаратні вимоги для роботи валідаторів обмежують децентралізацію мережі. Інколи спостерігаються перебої у функціонуванні мережі під час пікових навантажень, що викликає занепокоєння щодо її надійності. Крім того, вартість участі у консенсусі є досить високою, що обмежує доступ нових операторів до процесу валідації.

Водночас команда Solana Foundation активно працює над оптимізацією мережі шляхом вдосконалення протоколів, покращення процесів масштабування і розвитку інфраструктури підтримки для розробників.

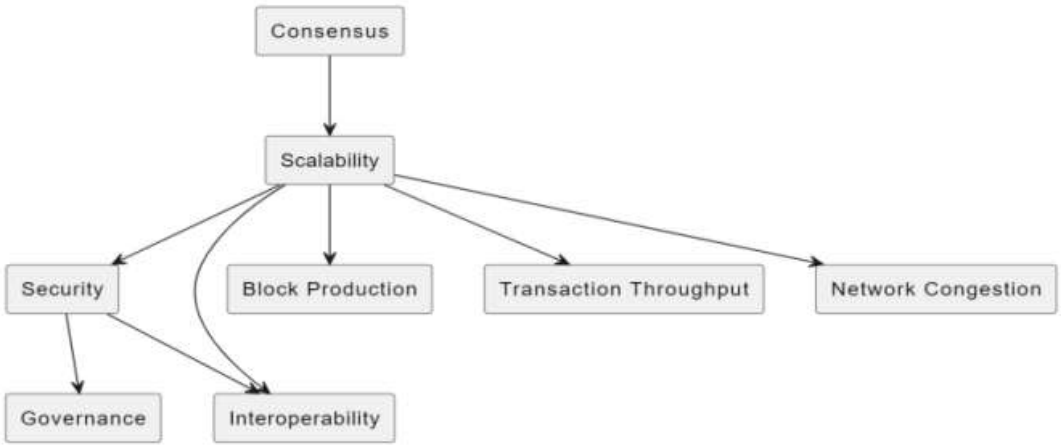

Рис. 1.3 – Основні виклики Solana Blockchain [1]

РОЗДІЛ 2. ТЕХНОЛОГІЯ SOLANA TOKEN EXTENSIONS

2.1. Передумови створення Token Extensions

На початку розвитку платформи Solana для взаємодії з токенами використовували стандарт SPL Token Program, створений у 2017–2018 роках. Він добре підходив для простих сценаріїв — таких як створення, передача або зберігання звичайних токенів. Але з часом цього стало недостатньо. З'явилися нові виклики: NFT-проєкти, токенізовані активи, системи доступу, фінансові інструменти — усе це вимагало більшої гнучкості та додаткових можливостей. Саме тому виникла потреба у вдосконаленому стандарті токенів, який зміг би покрити ці розширені кейси використання. [5]

Основні проблеми класичного SPL стандарту полягали в тому, що:

- Він не підтримував гнучке зберігання додаткових метаданих (наприклад, для NFT).
- Він не дозволяв створювати токени з обмеженням на передачу (наприклад, невзаємозамінні або закріплені за користувачем активи).
- Кожна нова необхідність вимагала створення окремої програми або нестандартних рішень, що ускладнювало екосистему. [5]

У відповідь на ці виклики команда розробників Solana вирішила створити більш гнучкий підхід. Їхньою метою було зробити так, щоб нові можливості можна було додавати до токенів без змін у базовому стандарті — і при цьому зберігалася сумісність з уже існуючими гаманцями та сервісами. Так з'явився новий підхід — Token Extensions, який і став частиною оновленого стандарту Token-2022.

2.2. Загальна характеристика Solana Token Extensions

Архітектура Solana Token Extensions побудована на принципі модульності: кожен токен може мати нуль або кілька розширень, які додають до нього нові властивості або змінюють його поведінку. Це дає змогу розробникам адаптувати

функціональність токена під конкретні потреби застосунку — і робити це без створення окремих смарт-контрактів чи нових токен-програм.

Технічно, під час створення токена вказується, які саме розширення потрібні, і для них одразу резервується пам’ять у токен-акаунті. Після ініціалізації цей набір розширень уже не змінюється — і це важливо, бо забезпечується сталість структури даних.

Серед основних переваг використання Token Extensions можна виділити такі:

- Гнучкість конфігурації: при створенні токена розробник самостійно визначає набір необхідних розширень відповідно до потреб застосунку.
- Сумісність із існуючими рішеннями: токени з розширеннями зберігають базову функціональність SPL Token, тому навіть сервіси з частковою підтримкою можуть із ними працювати.
- Масштабованість: підтримка широкого спектра сценаріїв використання без необхідності створення нових токен-програм.
- Оптимізація розробки: скорочення часу і ресурсів на розробку нових токен-функцій за рахунок повторного використання стандартних розширень. [5]

2.3. Основні типи розширень і їх функціональність

2.3.1. Metadata Pointer та Metadata Extension

Одним із ключових розширень у Token Extensions є Metadata Pointer Extension. Воно дозволяє токену містити посилання (URI) на зовнішнє джерело метаданих. Це особливо важливо у випадку з NFT, де метадані — це не просто опис, а повноцінна частина цифрового активу. Замість того щоб зберігати всю інформацію в самому токені, можна просто вказати URI — наприклад, на файл у мережах Arweave або IPFS. [6]

Доповненням до цього є Metadata Extension, яке дає змогу прямо вказати базову інформацію про токен: його назву, символ, URI і тощо. Така структура спрощує роботу з токенами у гаманцях та застосунках, оскільки стандартні поля

стають доступними без додаткових запитів. Для NFT це критично, адже саме ці дані використовуються для коректного відображення зображень, назв і описів.

2.3.2. Non-Transferable Tokens

Non-Transferable Tokens — це розширення, яке забороняє передачу токенів після їхнього створення. Така властивість є корисною для токенів, що репрезентують сертифікати, бейджі, досягнення, або інші унікальні ознаки, які повинні залишатися у володінні однієї адреси.

Технічно розширення блокує будь-які операції передачі. Токен можна лише спалити (burn) або виконати інші дії, які прямо дозволені адміністратором.

2.3.3. Permanent Delegate Extension

Permanent Delegate Extension дозволяє власнику токена призначити "делегата" — іншого користувача або програму, яка отримає постійні права на певні дії з токеном. Наприклад, делегат може мати дозвіл на передачу токена або його спалення, навіть якщо основний власник цього робити не може.

Це розширення особливо корисне для створення токенів з адміністративними правами або побудови внутрішніх економік у застосунках, де діють додаткові правила доступу чи передачі активів.

2.3.4. Інші розширення

Крім вищезгаданих, Solana Token Extensions також включають інші корисні розширення:

- Mint Close Authority — дає можливість закрити "мінтинг", тобто заборонити подальший випуск токенів після певного моменту.
- Default Account State — дозволяє встановити початковий стан нових токен-акаунтів. Наприклад, їх можна створювати одразу замороженими.
- Interest-Bearing Tokens — підтримують автоматичне нарахування відсотків для власників токена, що може бути корисно для DeFi-застосунків чи програм лояльності. [6]

2.4. Використання Token Extensions в реальних проєктах

Завдяки своїй гнучкості й модульній природі, Token Extensions вже знаходять застосування у різноманітних галузях: від ігрових проєктів до фінансових платформ.

Ігрові проекти

- Star Atlas: У грі Star Atlas NFT-контракти активно використовують розширення метаданих для зберігання атрибутів космічних кораблів, екіпажу та спорядження. Завдяки Metadata Pointer Extension вони можуть оновлювати атрибути без необхідності перевипуску токенів.
- Blessed Burgers: Ця гра використовує «NFT-бургери», що мають унікальні характеристики. Проєкт інтегрує Metadata Pointer Extension для гнучкого управління атрибутами бургерів, а також Non-Transferable Extension для закріплення унікальності активів за власником.

Фінансові платформи

- UXD Protocol: Використання Mint Close Authority Extension дозволяє контролювати емісію стейблкоїнів після забезпечення повної капіталізації. Це допомагає гарантувати стабільність пропозиції активу.
- Parrot Protocol: Для автоматизації фінансових продуктів застосовується Default Account State Extension, що дозволяє створювати акаунти із початковим замороженим станом до проходження верифікації.

Токени

- BERN Token: Перший токен, створений на базі Token-2022, використовує Transfer Fee Extension. При кожному трансфері 6.9% суми стягується як комісія: частина йде на спалювання токенів BONK і BERN, а частина розподіляється між тримачами. [7]

- Wen New Standard (WNS): Новий стандарт NFT побудований на базі Token Extensions. Використовує Metadata, Metadata Pointer, Transfer Hook, Immutable Owner та Group Pointer розширення для забезпечення роялті та структуризації колекцій. [7]
- Paxos USDP Stablecoin: Використовує Mint Close Authority, Permanent Delegate, Confidential Transfer та Metadata Extensions для дотримання регуляторних вимог і забезпечення безпеки та конфіденційності транзакцій. [7]

2.5. Поточний стан розвитку

На момент 2025 року Solana активно інтегрує Token Extensions у свою інфраструктуру. Стандарт Token 2022, який включає підтримку розширень, є доступним для використання всіма розробниками через оновлені версії Solana Program Library (SPL) та відповідні SDK.

Підтримка на рівні гаманців і сервісів

- Найпопулярніші гаманці, такі як Phantom та Solflare, активно працюють над повною підтримкою Token 2022 і розширень, хоча деякі функції (наприклад, робота з Non-Transferable Token або Custom Metadata) все ще впроваджуються поступово.
- Платформи для створення NFT, такі як Metaplex і Wen New Standard (WNS), впровадили підтримку Metadata Pointer Extension та Immutable Owner Extension. [7]
- NFT-маркетплейси, такі як Magic Eden і Tensor, оголосили про плани додати підтримку токенів із Token Extensions у майбутніх оновленнях.

Інфраструктура інтеграція

- Бібліотеки Solana SPL, включаючи spl-token-2022 та spl-token-metadata-interface, активно підтримуються і оновлюються для роботи з розширеннями.

- Впроваджено сервіси сканування блокчейну (наприклад, Solana FM і Solscan), які дозволяють переглядати структуру розширень у токенах.

Розділ 3. РОЗРОБКА ДОДАТКУ

3.1. Визначення вимог

Функціональні вимоги:

- Можливість авторизуватися через підключення криптогаманця (Phantom або Solflare).
- Створення NFT-персонажа (mint_character) із прив'язкою до токена наступної інформації:
    - Базова інформація (ім’я, зображення) зберігається в URI JSON.
    - Динамічні атрибути (клас персонажа, рівень, зброя) зберігаються через metadata pointer
- Можливість виконання «місій» (complete_mission) для прокачки персонажа (оновлення атрибутів NFT через Token Extensions).
- Підтримка оновлення метаданих без необхідності перевипуску токену.
- Можливість переглянути основну інформацію про NFT-персонажа.
- Усі операції над NFT мають підтверджуватися підписом користувача через гаманець.

Нефункціональні вимоги:

- Використання програмних засобів:
    - Смарт-контракт на Rust із використанням Solana Program Library (SPL) та Token-2022 Extensions.
    - Фронтенд на Next.js (React) із інтеграцією через бібліотеки Solana Wallet Adapter та Anchor.
- Розміщення базових метаданих NFT на децентралізованому сховищі Arweave.
- Застосунок має забезпечувати простий, інтуїтивний та адаптивний користувацький інтерфейс.
- Приватні дані користувачів не повинні зберігатися у застосунку ні в якій формі.

- Всі оновлення NFT повинні бути відображені в інтерфейсі в реальному часі (або після оновлення сторінки).

3.2. Вибір технологій

Solana

Основною платформою для розробки проєкту була обрана блокчейн-мережа Solana. Вона добре зарекомендувала себе завдяки своїй високій швидкості транзакцій, низьким комісіям і здатності масштабуватися. Окрім цього, Solana підтримує сучасний стандарт Token-2022, який дозволяє значно розширити функціональність токенів без створення окремих контрактів чи нестандартних рішень.

Мова програмування Rust

Для створення смарт-контрактів було використано мову програмування Rust. Вона забезпечує високу продуктивність, безпечну роботу з пам’яттю і є основною мовою для розробки програм на Solana. Використання Rust дозволяє створювати оптимізовані програми з мінімальними витратами ресурсів і високим рівнем захисту від помилок. Завдяки потужній підтримці інструментів для розробки на Solana, зокрема бібліотеки Solana SDK та фреймворку Anchor, розробка смарт-контрактів відбувається швидко та структуровано.

Next.js + React

Фронтенд було реалізовано з використанням Next.js, побудованого на основі React. Next.js дозволяє реалізувати серверний рендеринг, що забезпечує високу швидкість завантаження сторінок і покращує взаємодію користувача із застосунком. Завдяки можливостям Next.js легко створювати адаптивні інтерфейси, що ефективно працюють як на комп’ютерах, так і на мобільних пристроях.

Вибір саме Next.js був також зумовлений технічними обмеженнями браузерного середовища: для роботи з Solana та її токен-розширеннями потрібні

певні Node.js-залежності (crypto, buffer, stream), які не підтримуються у віртуальній машині браузера. Завдяки тому, що Next.js виконується на сервері, він дозволяє обробляти ці обчислення на серверній стороні. Це зробило Next.js не просто зручним, а фактично необхідним вибором для коректної інтеграції з Solana SDK.

Solana Wallet Adapter

Інтеграція криптовалютних гаманців у застосунок здійснюється за допомогою бібліотеки Solana Wallet Adapter. Вона дозволяє безпечно підключати популярні гаманці, такі як Phantom, Solflare, забезпечуючи простий процес автентифікації користувачів. Бібліотека також надає зручний інтерфейс для підписання транзакцій та взаємодії з блокчейном без потреби у прямому доступі до приватних ключів користувача.

Arweave

Для зберігання зображень NFT і метаданих було обрано децентралізовану платформу збереження даних Arweave. Вона гарантує постійний доступ до контенту та захищає його від втрати через збої централізованих серверів. Це рішення добре підходить для NFT-проєктів, оскільки користувачі можуть бути впевнені в тому, що їхні токени не втратять пов’язаний із ними контент.

Chakra UI

Інтерфейс користувача створено за допомогою бібліотеки Chakra UI. Вона надає готові, зручні у використанні компоненти та дозволяє швидко будувати сучасний дизайн, що виглядає добре як у світлій, так і в темній темі. Завдяки цій бібліотеці вдалося не лише пришвидшити розробку, а й забезпечити привабливий вигляд та хорошу адаптивність інтерфейсу.

3.3. Смарт-контракти

У рамках реалізації проєкту були розроблені два основних смарт-контракти: mint_character та complete_mission. Вони відповідають за створення

NFT-персонажів із динамічними характеристиками та за оновлення атрибутів персонажів після виконання місій відповідно.

1) Смарт-контракт mint_character

Контракт mint_character реалізує процес створення нового NFT-персонажа на базі Solana Token-2022 із використанням Metadata Pointer Extension. Спочатку створюється обліковий запис для майбутнього токена, з урахуванням того, що потрібно зарезервувати пам’ять для метаданих. Потім ініціалізується сам токен, встановлюється авторитет керування, і додається посилання на метадані (через URI), де зберігається інформація про персонажа.

Далі встановлюється Metadata Pointer, що прив'язує токен до метаданих, розміщених за окремим URI. Після цього створюється базова інформація про персонажа через інструкцію ініціалізації метаданих, включаючи ім'я персонажа, символ колекції та посилання на зображення. Окрім базових полів, одразу встановлюються динамічні атрибути, такі як клас персонажа, тип зброї, рівень та кількість досвіду. Ці атрибути записуються безпосередньо у метадані токена за допомогою розширення Token Metadata Interface.

Також в межах інструкції створюється токен-акаунт користувача через Associated Token Program, і на нього мінтиться один екземпляр щойно створеного персонажа. Щоб запобігти створенню дублікатів, авторитет на мінтинг одразу відкликається — таким чином, токен стає унікальним і незмінним у плані кількості.

2) Смарт-контракт complete_mission

Контракт complete_mission реалізує функціонал оновлення характеристик персонажа після проходження місії. Користувач передає кількість досвіду, яку персонаж отримав за місію. Спочатку контракт перевіряє, чи є підписувач транзакції дійсним власником персонажа, щоб запобігти несанкціонованим змінам.

Після підтвердження авторизації кількість досвіду персонажа збільшується. Якщо нова кількість досвіду перевищує встановлений поріг для поточного рівня (визначений як добуток рівня на фіксовану величину), рівень персонажа підвищується, а кількість досвіду скидається до нуля.

Далі контракт оновлює відповідні поля у токені NFT через інструкції оновлення полів у Token Metadata Interface. Зокрема, значення рівня та досвіду записуються безпосередньо в метадані токена на блокчейні. Оновлення відбувається через підписання транзакцій авторитетом NFT, що гарантує автентичність змін.

Таким чином, обидва смарт-контракти забезпечують повний цикл роботи з NFT-персонажами: від їхнього створення із встановленням початкових атрибутів до поступового розвитку та прокачування характеристик через взаємодію в межах гри.

3.4. Веб-застосунок

У межах реалізації проєкту також був створений веб-застосунок, який забезпечує зручну взаємодію користувачів із системою NFT-персонажів на базі Solana Token Extensions. Веб-застосунок є інтерфейсом між користувачем і блокчейном, надаючи можливість створювати нових персонажів, переглядати наявні NFT та виконувати дії для розвитку персонажів.

Створення NFT-персонажу

Створення нового персонажа починається з натискання кнопки "Mint New Character" на головній сторінці застосунку.

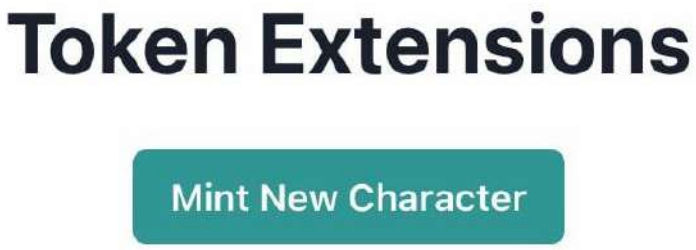

Рис. 3.1 – Кнопка мінту нового персонажу

Після цього відкривається форма для введення параметрів персонажа: ім'я, клас і тип зброї. Користувач може обрати клас героя серед варіантів, таких як «Warrior», «Archer» або «Mage». Залежно від вибраного класу змінюється опис персонажа та його зовнішній вигляд.

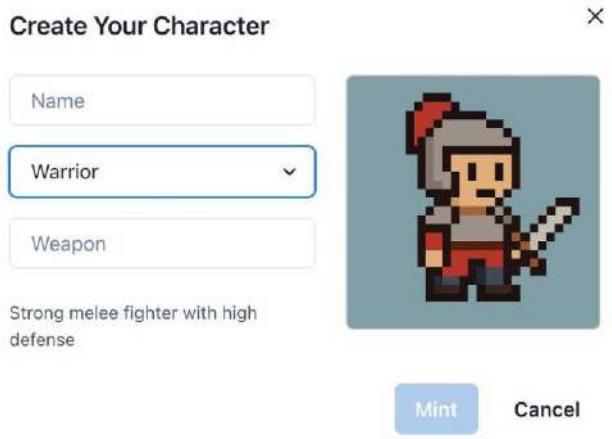

Рис. 3.2 – Інтерфейс створення персонажа типу “Warrior”

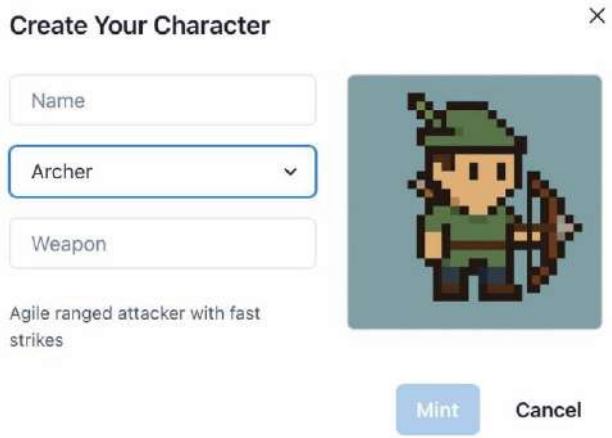

Рис. 3.3 – Інтерфейс створення персонажа типу “Archer”

Рис. 3.4 – Інтерфейс створення персонажа типу “Mage”

Після заповнення усіх обов'язкових полів користувач натискає кнопку "Mint", яка викликає функцію handleMint на фронтенді. Ця функція звертається до бекенд-логіки через виклик mint_character у смарт-контракті, передаючи введені користувачем дані (ім'я, клас, тип зброї).

Після створення персонажа токен автоматично відображається у підключеному криптовалютному гаманці користувача. Однак варто зауважити, що наразі більшість гаманців підтримують лише базові метадані (ім'я, зображення і опис), і не відображають додаткові атрибути, які були записані через розширення Metadata Pointer Extension у стандарті Token-2022.

Рис. 3.5 – Перегляд створеного NFT-персонажа у гаманці Phantom

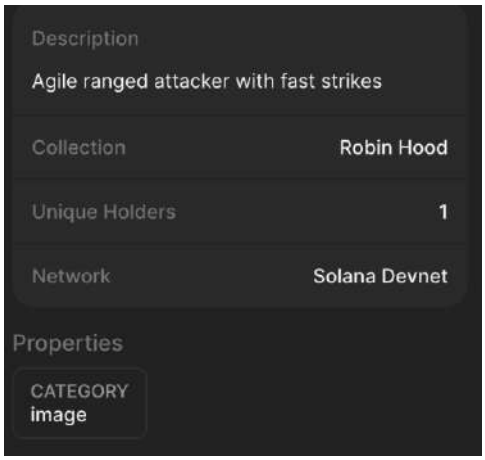

Рис. 3.6 – Інформація про NFT-персонажа у гаманці Phantom

Щоб переглянути повну інформацію про створений токен, включно із динамічними атрибутами персонажа (наприклад, клас, зброя, рівень, досвід), користувач може скористатися сервісом SolanaFM. У SolanaFM можна переглянути повні деталі транзакції створення NFT та всі розширення, прикріплені до токена.

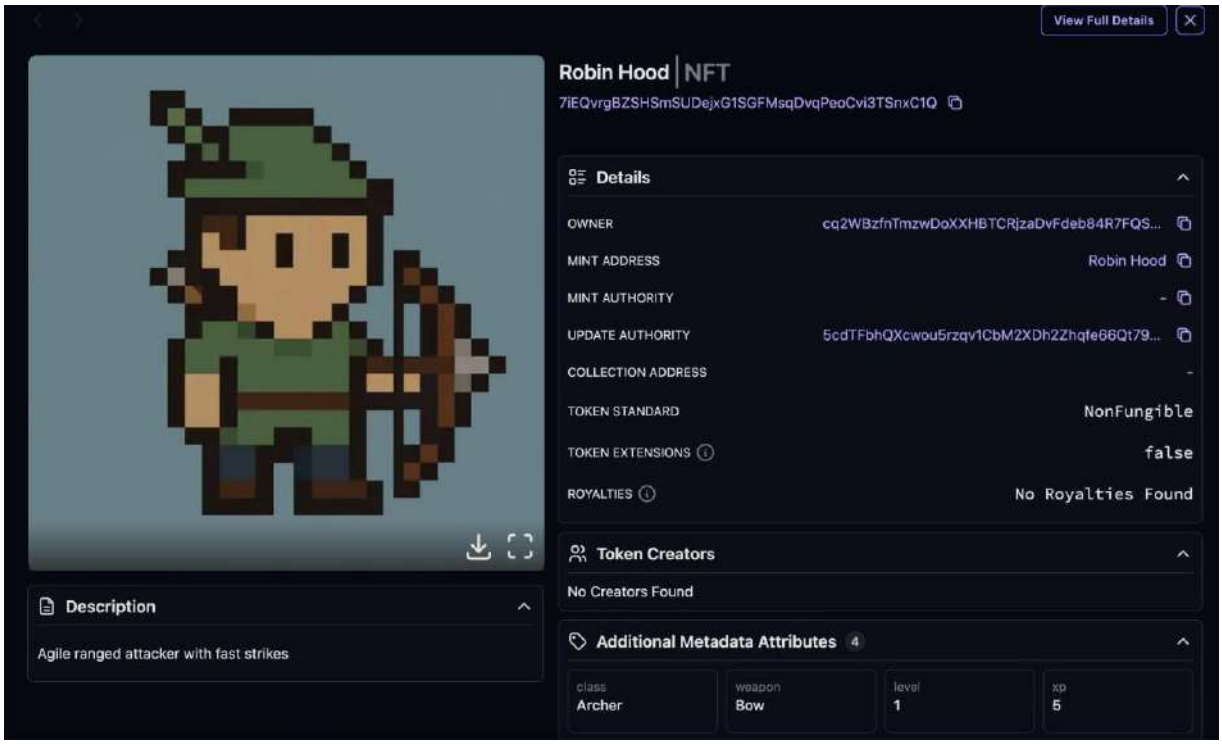

Рис. 3.7 – Перегляд детальної інформації про NFT-персонажа через SolanaFM

Рис. 3.8 – Перегляд усіх метаданих NFT через SolanaFM

Стандарт метаданих NFT та використання Arweave

У рамках створення NFT-персонажів у цьому проєкті було дотримано стандарту метаданих Metaplex Metadata Standard, який широко використовується в екосистемі Solana для токенів [8]. Структура метаданих включає базову інформацію про NFT, таку як ім'я токену ("name"), символ колекції ("symbol"), опис ("description"), посилання на зображення ("image"), атрибути ("attributes") та додаткові властивості ("properties"). Кожен NFT містить посилання на JSON-файл із цими даними.

```
{
  "name": "Archer",
  "symbol": "RPG",
  "description": "Agile ranged attacker with fast strikes",
  "image": "https://arweave.net/6PwvG0IMwcjT46rTAjK__6awEVeLQpDRGl86uWcf0vk",
  "attributes": [],
  "properties": {
    "files": [
      {
        "uri": "https://arweave.net/6PwvG0IMwcjT46rTAjK__6awEVeLQpDRGl86uWcf0vk",
        "type": "image/png"
      }
    ],
    "category": "image"
  }
}
```

Рис. 3.9 – Приклад метаданих для одного з персонажів:

Для зберігання метаданих було обрано використання децентралізованої мережі Arweave. Вона дозволяє розміщувати файли таким чином, щоб вони були доступними без обмеження у часі та не залежали від роботи окремого централізованого сервера. Завдяки цьому забезпечується довговічність і

незмінність інформації про NFT, що критично важливо для підтримки цілісності цифрових активів.

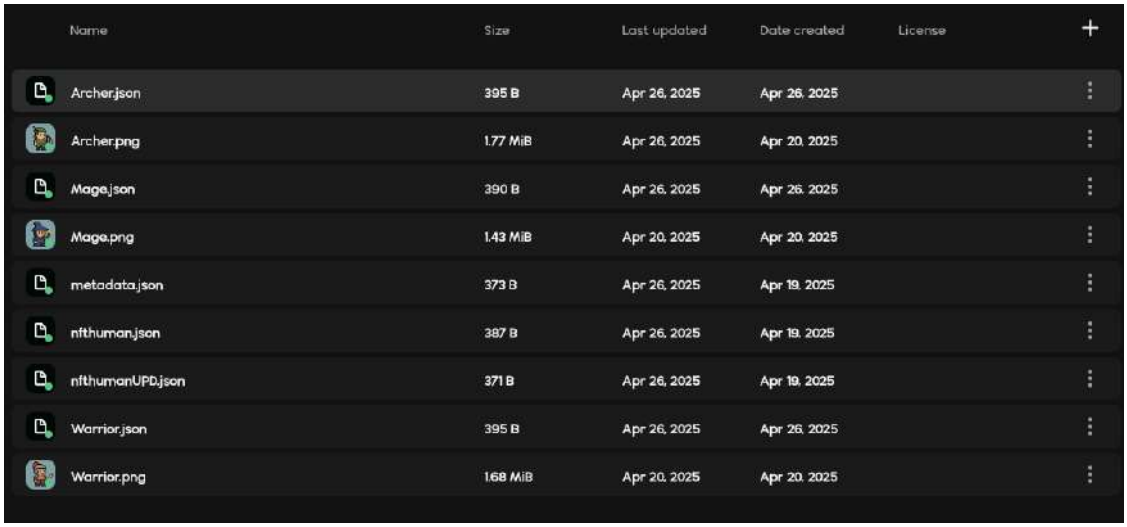

Рис. 3.10 – Збереження метаданих і зображень NFT у децентралізованій мережі Arweave

Динамічна зміна атрибутів NFT

Після створення персонажа користувач має можливість розвивати свого героя, виконуючи місії. Веб-застосунок реалізує зручний інтерфейс для взаємодії з функцією complete_mission, яка відповідає за оновлення характеристик NFT-персонажа без необхідності перевипуску токену.

Користувач бачить свого створеного NFT-персонажа, для якого може ініціювати виконання місії через кнопку "Complete Mission". При натисканні кнопки на фронтенді викликається функція handleCompleteMission, яка підключена до гаманця користувача через Solana Wallet Adapter.

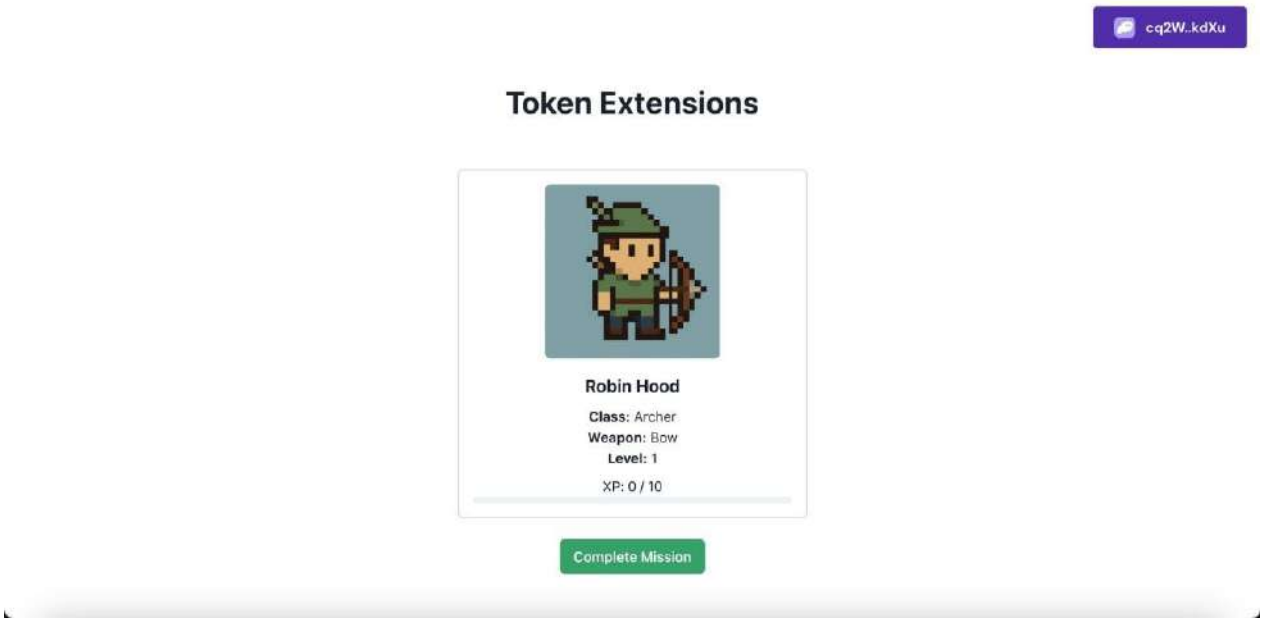

Рис. 3.11 – Інтерфейс користувача для взаємодії з NFT-персонажем

Після підтвердження транзакції у криптогаманці дані персонажа оновлюються безпосередньо на блокчейні. Кількість досвіду (XP) збільшується на певну величину. При досягненні встановленого порогу досвіду рівень персонажа автоматично підвищується на одиницю, а кількість досвіду скидається до нуля. Ці зміни миттєво відображаються у веб-застосунку завдяки оновленню стану через контексти CharacterStateProvider та NftProvider.

Оскільки використовується стандарт Token-2022 і розширення Metadata Pointer Extension, нові атрибути (рівень, досвід) записуються безпосередньо у розширення токену, зберігаючи незмінними базові метадані NFT, такі як зображення чи опис.

Крім того, усі оновлення можна переглянути через SolanaFM або аналогічні сервіси, що дозволяють побачити зміну атрибутів безпосередньо у метаданих токена.

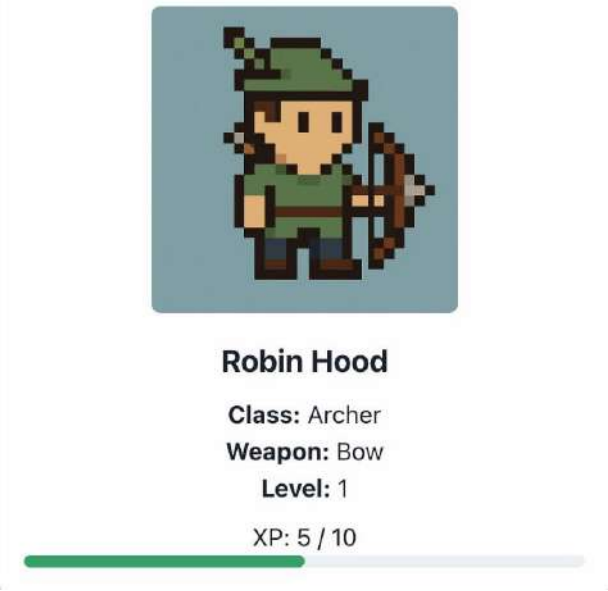

Рис. 3.12 – Відображення атрибутів NFT-персонажа до переходу на новий рівень

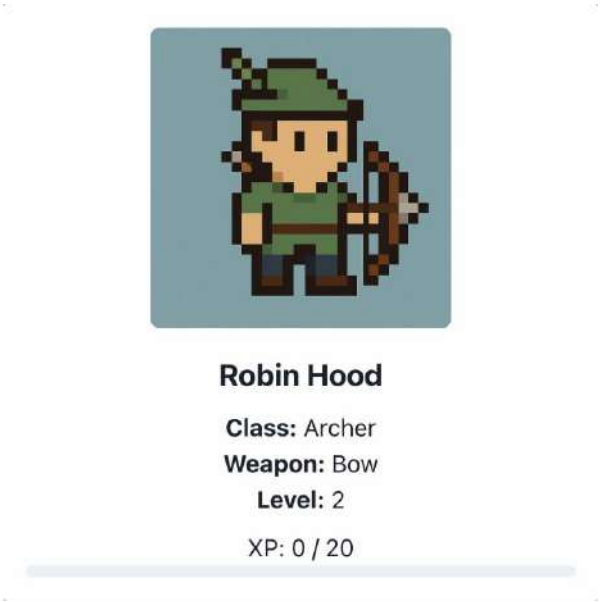

Рис. 3.13 – Відображення атрибутів NFT-персонажа після досягнення нового рівня з оновленими метаданими

Висновки

У результаті виконання курсової роботи було розроблено застосунок для створення та керування NFT з унікальними властивостями на базі блокчейну Solana. Проведено ґрунтовне дослідження технологічних особливостей блокчейн-платформи Solana та її архітектури, зокрема нового стандарту Token-2022 і підтримуваних розширень.

Було реалізовано смарт-контракти mint_character та complete_mission, які забезпечують повний цикл життя NFT-персонажів — від створення до оновлення їх характеристик без перевипуску токена.

Окрім того, розроблено веб-застосунок на основі Next.js і React, що забезпечує взаємодію з блокчейном через гаманець користувача та надає інтерфейс для створення і прокачування персонажів. Децентралізоване зберігання метаданих реалізовано через Arweave, що гарантує незмінність та доступність даних NFT у довгостроковій перспективі.

Таким чином, результати роботи демонструють високий потенціал використання Solana Token Extensions у сфері NFT-розробки. Запропонований підхід дозволяє створювати адаптивні цифрові активи з розширеною поведінкою без необхідності складних смарт-контрактів, що відкриває нові можливості для розвитку Web3-застосунків, ігрових платформ і токенізованих екосистем.

Список використаних джерел

1. Mishra D. P., Behera S. R., Behera S. S., Patro A. R., Salkuti S. R. *Solana blockchain technology: a review* [Електронний ресурс] / International Journal of Informatics and Communication Technology. – Режим доступу: https://ijict.iaescore.com/index.php/IJICT/article/view/20829/13042
2. Kong D., Li X., Li W. *Characterizing the Solana NFT Ecosystem* [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/2403.10879
3. *Solana Monkey Business* [Електронний ресурс]. – Режим доступу: https://solanamonkey.business/
4. Solana. Gulf Stream – *Solana's mempool-less transaction forwarding protocol* [Електронний ресурс]. – Режим доступу: https://solana.com/uk/news/gulf-stream--solana-s-mempool-less-transaction-forwarding-protocol
5. *Token Extensions on Solana* [Електронний ресурс] / Solana Foundation. – Режим доступу: https://cdn.builder.io/o/assets%2Fce0c7323a97a4d91bd0baa7490ec9139%2F83c26b9268f64400b8eb67442579a31a?alt=media&token=525415b0-d3ea-48d7-83d6-0fb0d9e522c6
6. Solana Foundation. *Solana Program Library: Token Extensions* [Електронний ресурс]. – Режим доступу: https://spl.solana.com/token-2022/extensions
7. *Solana Token Extensions – Crypto 101* [Електронний ресурс] / Phantom. – Режим доступу: https://phantom.com/learn/crypto-101/solana-token-extensions
8. Metaplex Foundation. *Token Metadata: JSON Standard* [Електронний ресурс]. – Режим доступу: https://developers.metaplex.com/token-metadata#a-json-standard

Додаток А

mint_character.rs

```
pub use crate::errors::ProgramErrorCode;
use anchor_lang::prelude::*;
use anchor_spl::{
    associated_token::{ self, AssociatedToken },
    token_2022,
    token_interface::{ spl_token_2022::instruction::AuthorityType, Token2022,
TokenAccount },
};
use anchor_lang::solana_program::{
    program::{invoke, invoke_signed},
    system_instruction,
    sysvar::rent::Rent,
};
use spl_token_2022::{ extension::ExtensionType, state::Mint };


pub use crate::state::character_data::CharacterMetadata;

pub fn mint_character(
    ctx: Context<MintCharacter>,
    name: String,
    class: String,
    weapon: String,
    uri: String,
) -> Result<()> {
    let mint_space = match

ExtensionType::try_calculate_account_len::<Mint>(&[ExtensionType::MetadataPointe
r])
    {
        Ok(space) => space,
        Err(_) => {
            return err!(ProgramErrorCode::InvalidMintAccountSpace);
        }
    };
    let meta_data_space = 250;
    let mint_rent = Rent::get()?.minimum_balance(mint_space + meta_data_space);

    anchor_lang::system_program::create_account(
        CpiContext::new(
            ctx.accounts.system_program.to_account_info(),
            anchor_lang::system_program::CreateAccount {
                from: ctx.accounts.payer.to_account_info(),
                to:   ctx.accounts.mint.to_account_info(),
            },
        ),
        mint_rent,
        mint_space as u64,
        &ctx.accounts.token_program.key()
    )?;

    anchor_lang::system_program::assign(
        CpiContext::new(ctx.accounts.token_program.to_account_info(),
anchor_lang::system_program::Assign {
            account_to_assign: ctx.accounts.mint.to_account_info(),
        }),
        &token_2022::ID
    )?;
```

```
    let init_meta_data_pointer_ix =
        spl_token_2022::extension::metadata_pointer::instruction::initialize(
            &Token2022::id(),
            &ctx.accounts.mint.key(),
            Some(ctx.accounts.nft_authority.key()),
            Some(ctx.accounts.mint.key()),
        )?;

    invoke(
        &init_meta_data_pointer_ix,
        &[
            ctx.accounts.mint.to_account_info(),
            ctx.accounts.nft_authority.to_account_info()
        ],
    )?;

    let mint_cpi_ix = CpiContext::new(
        ctx.accounts.token_program.to_account_info(),
        token_2022::InitializeMint2 {
            mint: ctx.accounts.mint.to_account_info(),
        },
    );

    token_2022::initialize_mint2(
        mint_cpi_ix,
        0,
        &ctx.accounts.nft_authority.key(),
        None)?;

    let seeds = b"nft_authority";
    let bump = ctx.bumps.nft_authority;
    let signer: &[&[&[u8]]] = &[&[seeds, &[bump]]];

    msg!("Init metadata {0}", ctx.accounts.nft_authority.to_account_info().key);

    let init_token_meta_data_ix =
&spl_token_metadata_interface::instruction::initialize(
        &spl_token_2022::id(),
        ctx.accounts.mint.key,
        ctx.accounts.nft_authority.to_account_info().key,
        ctx.accounts.mint.key,
        ctx.accounts.nft_authority.to_account_info().key,
        name.clone(),
        "RPG".to_string(),
        uri.to_string(),
    );

    invoke_signed(
        init_token_meta_data_ix,
        &[
            ctx.accounts.mint.to_account_info().clone(),
            ctx.accounts.nft_authority.to_account_info().clone(),
        ],
        signer
    )?;

    for (key, val) in [
        ("class", class.as_str()),
        ("weapon", weapon.as_str()),
        ("level", "1"),
        ("xp", "0"),
    ] {
```

```
        let ix = &spl_token_metadata_interface::instruction::update_field(
            &spl_token_2022::id(),
            ctx.accounts.mint.key,
            ctx.accounts.nft_authority.to_account_info().key,
            spl_token_metadata_interface::state::Field::Key(key.to_string()),
            val.to_string(),
        );

        invoke_signed(
            &ix,
            &[
                ctx.accounts.mint.to_account_info().clone(),
                ctx.accounts.nft_authority.to_account_info().clone(),
            ],
            signer,
        )?;
    }

    let metadata = &mut ctx.accounts.metadata_account;
    metadata.authority = ctx.accounts.payer.key();
    metadata.name = name;
    metadata.class = class;
    metadata.weapon = weapon;
    metadata.level = 1;
    metadata.xp = 0;

    associated_token::create(CpiContext::new(
        ctx.accounts.associated_token_program.to_account_info(),
        associated_token::Create {
            payer: ctx.accounts.payer.to_account_info(),
            associated_token: ctx.accounts.token_account.to_account_info(),
            authority: ctx.accounts.payer.to_account_info(),
            mint: ctx.accounts.mint.to_account_info(),
            system_program: ctx.accounts.system_program.to_account_info(),
            token_program: ctx.accounts.token_program.to_account_info(),
        },
    ))?;

    token_2022::mint_to(
        CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            token_2022::MintTo {
                mint: ctx.accounts.mint.to_account_info(),
                to: ctx.accounts.token_account.to_account_info(),
                authority: ctx.accounts.nft_authority.to_account_info(),
            },
            signer
        ),
        1,
    )?;

    token_2022::set_authority(
        CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            token_2022::SetAuthority {
                current_authority: ctx.accounts.nft_authority.to_account_info(),
                account_or_mint: ctx.accounts.mint.to_account_info(),
            },
            signer
        ),
        AuthorityType::MintTokens,
        None,
    )?;
```

```
    Ok(())
}

#[derive(Accounts)]
pub struct MintCharacter<'info> {
    #[account(mut)]
    pub payer: Signer<'info>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token2022>,
    /// CHECK: Will be created later
    #[account(mut)]
    pub token_account: AccountInfo<'info>,
    #[account(mut)]
    pub mint: Signer<'info>,
    pub rent: Sysvar<'info, Rent>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    #[account(init_if_needed, seeds = [b"nft_authority".as_ref()], bump, space =
8, payer = payer)]
    pub nft_authority: Account<'info, NftAuthority>,
    #[account(
        init,
        payer = payer,
        space = 8 + CharacterMetadata::LEN,
        seeds = [b"character", payer.key().as_ref()],
        bump,
    )]
    pub metadata_account: Account<'info, CharacterMetadata>,
}

#[account]
pub struct NftAuthority {}
```

complete_mission.rs

```
pub use crate::errors::GameErrorCode;
use crate::{state::character_data::CharacterMetadata, NftAuthority};
use anchor_lang::prelude::*;
use anchor_spl::token_interface::{Token2022};
use solana_program::program::invoke_signed;

pub fn complete_mission(ctx: Context<CompleteMission>, xp_gain: u32) ->
Result<()> {
    let character = &mut ctx.accounts.character;

    if ctx.accounts.signer.key() != character.authority {
        return err!(GameErrorCode::WrongAuthority);
    }

    character.xp += xp_gain;

    let xp_threshold = (character.level as u32) * 10;
    if character.xp >= xp_threshold {
        character.level += 1;
        character.xp = 0;
    }

    let seeds = b"nft_authority";
    let bump = ctx.bumps.nft_authority;
```

```
    let signer: &[&[&[u8]]] = &[&[seeds, &[bump]]];

    let ix_level = spl_token_metadata_interface::instruction::update_field(
        &spl_token_2022::id(),
        ctx.accounts.mint.to_account_info().key,
        ctx.accounts.nft_authority.to_account_info().key,
        spl_token_metadata_interface::state::Field::Key("level".to_string()),
        character.level.to_string(),
    );
    invoke_signed(
        &ix_level,
        &[
            ctx.accounts.mint.to_account_info().clone(),
            ctx.accounts.nft_authority.to_account_info().clone(),
        ],
        signer,
    )?;

    let ix_xp = spl_token_metadata_interface::instruction::update_field(
        &spl_token_2022::id(),
        ctx.accounts.mint.to_account_info().key,
        ctx.accounts.nft_authority.to_account_info().key,
        spl_token_metadata_interface::state::Field::Key("xp".to_string()),
        character.xp.to_string(),
    );
    invoke_signed(
        &ix_xp,
        &[
            ctx.accounts.mint.to_account_info().clone(),
            ctx.accounts.nft_authority.to_account_info().clone(),
        ],
        signer,
    )?;

    msg!(
        "Mission complete! XP: {}, Level: {}",
        character.xp,
        character.level
    );

    Ok(())
}

#[derive(Accounts)]
pub struct CompleteMission<'info> {
    #[account(
        mut,
        seeds = [b"character".as_ref(), character.authority.key().as_ref()],
        bump,
    )]
    pub character: Account<'info, CharacterMetadata>,

    #[account(mut)]
    pub signer: Signer<'info>,

    pub system_program: Program<'info, System>,

    /// CHECK: ensure mint belongs to signer externally
    #[account(mut)]
    pub mint: AccountInfo<'info>,

    #[account(
        init_if_needed,
```

```
        seeds = [b"nft_authority"],
        bump,
        space = 8,
        payer = signer,
    )]
    pub nft_authority: Account<'info, NftAuthority>,

    pub token_program: Program<'info, Token2022>,
}
```

MintCharacterModal.tsx

```
"use client"

import {
    Modal,
    ModalOverlay,
    ModalContent,
    ModalHeader,
    ModalCloseButton,
    ModalBody,
    ModalFooter,
    Button,
    Input,
    Select,
    useDisclosure,
    Image,
    Box,
    Flex,
    VStack,
} from "@chakra-ui/react"
import { useState } from "react"
import { useWallet, useConnection } from "@solana/wallet-adapter-react"
import {
    PublicKey,
    SystemProgram,
    Keypair,
} from "@solana/web3.js"
import {
    TOKEN_2022_PROGRAM_ID,
    getAssociatedTokenAddressSync,
    ASSOCIATED_TOKEN_PROGRAM_ID,
} from "@solana/spl-token"
import { web3 } from "@coral-xyz/anchor"
import { program } from "@/utils/anchor"

const MintCharacterModal = () => {
    const { publicKey, sendTransaction } = useWallet()
    const { connection } = useConnection()
    const { isOpen, onOpen, onClose } = useDisclosure()

    const [name, setName] = useState("")
    const [classType, setClassType] = useState("Warrior")
    const [weapon, setWeapon] = useState("")
    const [isMinting, setIsMinting] = useState(false)

    const classImage = {
        Warrior: "/Warrior.png",
        Archer: "/Archer.png",
```

```
        Mage: "/Mage.png",
    }

    const classDescription = {
        Warrior: "Strong melee fighter with high defense",
        Archer: "Agile ranged attacker with fast strikes",
        Mage: "Master of magic with powerful spells",
    }

    const handleMint = async () => {
        if (!publicKey || !name || !weapon) return
        setIsMinting(true)

        const uriMapping = {
            Warrior:
"https://arweave.net/SM3UTFwlDHG_X5_VXqm5ALwUairPpxy_PfuoA6sI9pc",
            Archer: "https://arweave.net/M4OWZK-ZkTBD460iXdIKMnM4F-
xzViHHBsX3GCYSNbc",
            Mage: "https://arweave.net/G2-
CzvZg9eFd2UwKFA6PtKba4NgU3h8TRul6aSZE2-o",
        }

        try {
            const mint = new Keypair()
            const tokenAccount = getAssociatedTokenAddressSync(
                mint.publicKey,
                publicKey,
                false,
                TOKEN_2022_PROGRAM_ID,
                ASSOCIATED_TOKEN_PROGRAM_ID
            )

            const [metadataPDA] = PublicKey.findProgramAddressSync(
                [Buffer.from("character"), publicKey.toBuffer()],
                program.programId
            )

            const [nftAuthority] = PublicKey.findProgramAddressSync(
                [Buffer.from("nft_authority")],
                program.programId
            )

            const selectedUri = uriMapping[classType as keyof typeof uriMapping]

            const tx = await program.methods
                .mintCharacter(name, classType, weapon, selectedUri)
                .accounts({
                    payer: publicKey,
                    mint: mint.publicKey,
                    tokenAccount,
                    metadataAccount: metadataPDA,
                    nftAuthority,
                    tokenProgram: TOKEN_2022_PROGRAM_ID,
                    associatedTokenProgram: ASSOCIATED_TOKEN_PROGRAM_ID,
                    systemProgram: SystemProgram.programId,
                    rent: web3.SYSVAR_RENT_PUBKEY,
                })
                .signers([mint])
                .transaction()

            const txSig = await sendTransaction(tx, connection, {
                signers: [mint],
                skipPreflight: true,
```

```
            })

            console.log("Minted! Tx:",
`https://explorer.solana.com/tx/${txSig}?cluster=devnet`)
            onClose()
        } catch (e) {
            console.error("Mint failed", e)
        } finally {
            setIsMinting(false)
        }
    }

    return (
        <>
            <Button colorScheme="teal" onClick={onOpen}>
                Mint New Character
            </Button>

            <Modal isOpen={isOpen} onClose={onClose} size="lg">
                <ModalOverlay />
                <ModalContent>
                    <ModalHeader>Create Your Character</ModalHeader>
                    <ModalCloseButton />
                    <ModalBody>
                        <Flex gap={6}>
                            {/* LEFT SIDE: Form + description */}
                            <VStack flex="1" spacing={4} align="stretch">
                                <Input
                                    placeholder="Name"
                                    value={name}
                                    onChange={(e) => setName(e.target.value)}
                                />
                                <Select
                                    value={classType}
                                    onChange={(e) =>
setClassType(e.target.value)}
                                >
                                    <option value="Warrior">Warrior</option>
                                    <option value="Archer">Archer</option>
                                    <option value="Mage">Mage</option>
                                </Select>
                                <Input
                                    placeholder="Weapon"
                                    value={weapon}
                                    onChange={(e) => setWeapon(e.target.value)}
                                />
                                <Box fontSize="sm" color="gray.600" mt={2}>
                                    {classDescription[classType as keyof typeof
classDescription]}
                                </Box>
                            </VStack>

                            {/* RIGHT SIDE: Image */}
                            <Box flexShrink={0}>
                                <Image
                                    src={classImage[classType as keyof typeof
classImage]}
                                    alt={`${classType} preview`}
                                    boxSize="200px"
                                    objectFit="contain"
                                    borderRadius="md"
                                    border="1px solid #ccc"
                                />
```

```
                            </Box>
                        </Flex>
                    </ModalBody>


                    <ModalFooter>
                        <Button
                            colorScheme="blue"
                            mr={3}
                            onClick={handleMint}
                            isLoading={isMinting}
                            isDisabled={!name || !weapon}
                        >
                            Mint
                        </Button>
                        <Button variant="ghost" onClick={onClose}>
                            Cancel
                        </Button>
                    </ModalFooter>
                </ModalContent>
            </Modal>
        </>
    )
}

export default MintCharacterModal
```

CompleteMissionButton.tsx

```
"use client"
import { useCallback, useState } from "react"
import { Button, HStack, VStack } from "@chakra-ui/react"
import { useConnection, useWallet } from "@solana/wallet-adapter-react"
import { useSessionWallet } from "@magicblock-labs/gum-react-sdk"
import { useCharacterState } from "@/contexts/CharacterStateProvider";
import { useNftState } from "@/contexts/NftProvider"
import { program } from "@/utils/anchor"
import { PublicKey } from "@solana/web3.js"
import { TOKEN_2022_PROGRAM_ID } from "@solana/spl-token"

const CompleteMissionButton = () => {
    const { publicKey, sendTransaction } = useWallet()
    const { connection } = useConnection()
    const sessionWallet = useSessionWallet()
    const { nftState } = useNftState()
    const { characterDataPDA } = useCharacterState();
    const [isLoadingSession, setIsLoadingSession] = useState(false)
    const [isLoadingMainWallet, setIsLoadingMainWallet] = useState(false)
    const XP_GAIN = 5

    const getNftCharacter = async () => {
        const nftAuthority = PublicKey.findProgramAddressSync(
            [Buffer.from("nft_authority")],
            program.programId
        );
        console.log("NFT items:", nftState.items)
        return nftState.items.find(
            (nft) => nft.authorities[0]?.address === nftAuthority[0].toBase58()
        )
```

```
    }

    const handleCompleteMission = useCallback(
        async (isSession: boolean) => {
            if (!characterDataPDA) return

            const nft = await getNftCharacter()
            if (!nft) {
                window.alert("Mint your NFT character first")
                return
            }

            const nftAuthority = PublicKey.findProgramAddressSync(
                [Buffer.from("nft_authority")],
                program.programId
            );

            try {
                if (isSession && sessionWallet) {
                    setIsLoadingSession(true)

                    const tx = await program.methods
                        .completeMission(XP_GAIN)
                        .accounts({
                            character: characterDataPDA,
                            signer: sessionWallet.publicKey!,
                            mint: nft.id,
                            nftAuthority,
                            systemProgram: PublicKey.default,
                            tokenProgram: TOKEN_2022_PROGRAM_ID,
                        })
                        .transaction()

                    const txids = await
sessionWallet.signAndSendTransaction!(tx)
                    console.log("Mission transaction (session):", txids)
                } else if (publicKey) {
                    setIsLoadingMainWallet(true)

                    const tx = await program.methods
                        .completeMission(XP_GAIN)
                        .accounts({
                            character: characterDataPDA,
                            signer: publicKey,
                            mint: nft.id,
                            nftAuthority,
                            systemProgram: PublicKey.default,
                            tokenProgram: TOKEN_2022_PROGRAM_ID,
                        })
                        .transaction()

                    const txid = await sendTransaction(tx, connection, {
skipPreflight: true })
                    console.log(`Mission TX:
https://explorer.solana.com/tx/${txid}?cluster=devnet`)
                }
            } catch (err) {
                console.error("Mission failed:", err)
            } finally {
                setIsLoadingSession(false)
                setIsLoadingMainWallet(false)
            }
        },
```

```
        [publicKey, sessionWallet, connection, characterDataPDA, nftState]
    )

    return (
        <>
            {publicKey && (
                <VStack>
                    {/*<Image src="/Mission.png" alt="Mission Icon" width={64}
height={64} />*/}
                    <HStack>
                        <Button
                            colorScheme="green"
                            isLoading={isLoadingMainWallet}
                            onClick={() => handleCompleteMission(false)}
                        >
                            Complete Mission
                        </Button>
                    </HStack>
                </VStack>
            )}
        </>
    )
}

export default CompleteMissionButton
```

WalletMultiButton.tsx

```
import dynamic from "next/dynamic"

export const WalletMultiButtonDynamic = dynamic(
    async () =>
        (await import("@solana/wallet-adapter-react-ui")).WalletMultiButton,
    { ssr: false }
)

const WalletMultiButton = () => {
    return <WalletMultiButtonDynamic />
}

export default WalletMultiButton
```