ITIS 4102/5102: Survey of Programming Languages (Spring 2025)

Term Project Report

Team Name: CodeMiners

Project Title: Crypto Miners – A Decentralized NFT Marketplace on Internet Computer

Team Members:

1. **Nishant Acharekar (801363902)**
2. **Mihir Phatak (801358341)**
3. **Shubham Kakade (801413848)**
4. **Calvin Pinto (801320115)**
5. **Deepiga Sengottuvelu Ravichandran (801370629)**
6. **Sandesh Mahajan (801366489)**

Contents

- **Introduction & Motivation** 3
- **Language Overview – Motoko** 3
- **Real-World Use Cases of Motoko** 4
- **Problem Definition** 5
- **Project Overview** 5
- **Technical Architecture:** 6
- **Key Features Implemented** 11
- **Instructions to Install and Run the Project** 11
- **Development Highlights** 16
- **Challenges Faced** 17
- **Conclusion** 18
- **Acknowledgments & References** 19

Introduction & Motivation

NFTs (Non-Fungible Tokens) have transformed how we think about digital ownership, whether it's art, music, collectibles, or in-game assets. But despite their innovative promise, most existing NFT platforms operate on centralized systems. This means users are still dependent on intermediaries, which can lead to limited transparency, higher transaction fees, and potential security concerns.

Our motivation behind this project was to change that, to create a space where users have full control over their assets without needing to rely on any third-party platform. We wanted to make NFT trading not only decentralized but also intuitive and secure.

By building our marketplace on the Internet Computer (ICP) and writing smart contracts in Motoko, we're able to deliver a truly decentralized experience. Every transaction, from minting to purchasing, happens on-chain with no central authority involved. This ensures greater trust, ownership, authenticity, and long-term scalability for creators and collectors alike.

Language Overview – Motoko

Motoko is a modern, statically typed programming language created specifically for building applications on the Internet Computer. Unlike general-purpose languages, Motoko is tailor-made for blockchain development, offering native support for canisters (smart contracts) and seamless interaction with ICP's decentralized infrastructure.

One of Motoko's standout strengths is its actor-based model, which allows developers to write asynchronous code in a clean and intuitive way. This model is ideal for decentralized apps that rely on multiple smart contracts communicating in parallel.

It also comes with robust type safety, memory management, and tight WebAssembly integration, enabling efficient and secure execution directly in the browser or blockchain node.

a. **Key Features of Motoko:**

i. **Type Safety:** Catches errors early with strict compile-time checks.
ii. **Actor-Based Concurrency:** Perfect for modular and asynchronous smart contract interactions.
iii. **Built for ICP:** Native language for Internet Computer development.
iv. **Smart Contract Ready:** Smooth canister creation and lifecycle management.

Motoko gives developers the tools to build secure, scalable, and future-proof dApps that run entirely on-chain, no external servers or cloud providers needed.

Real-World Use Cases of Motoko

Motoko isn't just a theoretical language, it's actively powering real-world decentralized applications on the Internet Computer. Its ability to handle complex backend logic while maintaining full decentralization makes it a perfect fit for a wide range of use cases.

a. Some prominent areas where Motoko shines include:

i. **NFT Marketplaces:** Platforms like Entrepot use Motoko to allow creators to mint, sell, and trade NFTs with complete on-chain transparency and ownership.
ii. **DAO Governance Platforms:** Motoko enables communities to build decentralized autonomous organizations with smart voting mechanisms and rule enforcement.
iii. **Blockchain-Based Games:** From in-game item ownership to reward mechanisms, Motoko supports truly decentralized gaming economies.
iv. **Identity & Authentication Systems:** Secure user identities can be created and managed natively on the Internet Computer without third-party dependencies.
v. **Web3 Social Networks:** Next-gen social platforms built with Motoko ensure that user data stays private, secure, and fully user-owned.

As Web3 continues to evolve, Motoko is becoming a go-to language for developers looking to build reliable, scalable, and completely serverless applications from the ground up.

Problem Definition

While NFTs have unlocked new possibilities in digital ownership, most existing platforms are still tied to centralized infrastructures. This creates several bottlenecks and concerns that hinder their full potential:

a. **Centralized Control:** Even though NFTs are supposed to represent user-owned assets, centralized marketplaces often control listings, metadata storage, and access, limiting true ownership.
b. **High Gas Fees:** Traditional blockchain networks like Ethereum suffer from expensive and unpredictable transaction fees.
c. **Slow Transactions:** Congested networks can lead to delays, making real-time interaction difficult.
d. **Limited Discoverability:** Discovering new and trending NFTs isn't always intuitive or dynamic on many platforms.
e. **Security Risks**: Centralized asset handling increases the risk of exploits, tampering, or downtime.

Our solution tackles all these issues head-on by building on the Internet Computer, which offers high-speed execution and zero-gas transactions. By writing smart contracts in Motoko, we ensure that all core features, from minting to purchasing, are fully decentralized, secure, and scalable.

Project Overview

Our project focuses on building a fully decentralized NFT marketplace where users have complete control over their digital assets, no intermediaries, no hidden control, and no

reliance on centralized servers. Using the power of the Internet Computer and Motoko smart contracts, we've created a platform that allows users to:

a. **Mint NFTs:** Users can upload their own digital creations, images, collectibles, artwork, and mint them directly into NFTs stored on-chain.
b. **List NFTs for Sale:** Creators and collectors can list their NFTs at any price they choose, giving them full control over how and when to sell.
c. **Buy NFTs:** Buyers can explore available NFTs and securely purchase them using ICP tokens in a peer-to-peer transaction.
d. **Transfer Ownership:** Every purchase triggers a secure, on-chain transfer of ownership through Motoko-powered smart contracts, no manual handling required.
e. **My NFTs:** Users can view, manage, and track all the NFTs they own in a personalized section of the platform.
f. **Discover New NFTs:** A dedicated gallery helps users explore newly minted and trending NFTs, making discovery easy and engaging.

Everything is hosted and executed on the Internet Computer, meaning there are no centralized servers or third-party storage layers. This ensures end-to-end decentralization, greater transparency, and higher security for users at every step of the NFT lifecycle.

Technical Architecture:

Our decentralized NFT platform is designed using a clean separation of frontend, backend, and blockchain layers, all working seamlessly through the Internet Computer ecosystem.

a. **Frontend:** We've used React.js with Bootstrap to build a responsive and intuitive user interface. The UI allows users to easily interact with the NFT marketplace minting, listing, browsing, and purchasing NFTs with minimal effort.
b. **Backend (Smart Contracts):** All core logic and transactions are handled by smart contracts written in Motoko, which run directly on the Internet Computer

Protocol (ICP). These contracts ensure that all actions, like minting or transferring ownership, are securely processed on-chain.

c. **Blockchain & Storage:**
    i. **Blockchain Layer:** Internet Computer Protocol (ICP)
    ii. **Storage Layer:** Canisters are used to persist NFT metadata, ownership records, listing status, and transaction history.

We designed and deployed three Motoko programs as part of our development journey to demonstrate our understanding and application of the language's core features:

- **Program 1 – showcaseDataTypes.mo**
    - **Purpose:** This program demonstrates the usage of Motoko's fundamental data types and built-in methods. It helps establish how data is structured and manipulated in Motoko.

```
// CodeMiners
// 1.   Nishant Acharekar   801363902
// 2.   Mihir Phatak    801358341
// 3.   Shubham Kakade      801413848
// 4.   Calvin Pinto    801320115
// 5.   Deepiga Sengottuvelu Ravichandran   801370629
// 6.   Sandesh Mahajan 801366489
import Int "mo:base/Int";
import Nat "mo:base/Nat";
import Float "mo:base/Float";
import Text "mo:base/Text";
actor Program1 {
    public func showcaseDataTypes() : async Text {
        // 1. Integer (Int)
        let intValue : Int = 42;
        let intSum = Int.add(intValue, 8); // Built-in sum function
        let intAbs = Int.abs(-42); // Get absolute value
        let intToString = Int.toText(intValue); // Convert to string
        // 2. Float
        let floatValue : Float = 3.14;
        let floatProduct = Float.mul(floatValue, 2.0); // Multiplication
        let floatRound = Float.toInt(floatValue); // Convert float to int
        let floatToString = Float.toText(floatValue); // Convert to string
        // 3. Text (String)
        let stringValue : Text = "Hello, Motoko!";
```

```
        let replacedString = Text.replace(stringValue, #text "Motoko", "World");
// Replace substring
        let stringLength = Nat.toText(Text.size(stringValue)); // Get string
length
        // 4. Boolean (Bool)
        let boolValue : Bool = true;
        let boolString = (if boolValue { "true" } else { "false" }); // Convert
boolean to string
        let negatedBoolString = (if boolValue { "true" } else { "false" }); //
Now properly assigned

        // Return results with \n directly in the string
        return Text.concat(
            "Int Sum: " # Int.toText(intSum) # ", Int Absolute: " #
Int.toText(intAbs) # ", Int to String: " # intToString,
            Text.concat(
                "\nFloat Product: " # Float.toText(floatProduct) # ", Float
Rounded: " # Int.toText(floatRound) # ", Float to String: " # floatToString,
                Text.concat(
                    "\nReplaced String: " # replacedString # ", String Length: "
# stringLength,
                    "\nNegated Bool: " # negatedBoolString # ", Bool to String: "
# boolString
                )
            )
        );
```

- **Explanation:** It defines different variables using Motoko's basic types: Int, Float, Bool, and Text, and then returns them in a formatted string using debug_show. This helps in understanding Motoko's type system and printing values for debugging.

- **Program 2 – showcaseDataStructures.mo**
  - **Purpose:** This module dives into control structures and collections such as arrays, loops, and conditional statements. It's designed to model more dynamic logic, useful for implementing NFT logic like filtering or listing.

```
// CodeMiners
// 1.   Nishant Acharekar   801363902
// 2.   Mihir Phatak    801358341
// 3.   Shubham Kakade      801413848
// 4.   Calvin Pinto    801320115
// 5.   Deepiga Sengottuvelu Ravichandran   801370629
```

```
// 6.   Sandesh Mahajan 801366489
import Array "mo:base/Array";
import List "mo:base/List";
import Nat "mo:base/Nat";
import Debug "mo:base/Debug";
actor Program2 {
    // Program 2: Data Structures and Control Structures
    public func showcaseDataStructures() : async Text {
        // 1. Array (Sum elements using a for loop)
        let array : [Nat] = [1, 2, 3, 4, 5];
        var arraySum : Nat = 0;
        for (value in array.vals()) {
            arraySum += value;
        };
        // 2. List (Multiply elements using iteration)
        let list = List.fromArray<Nat>([10, 20, 30]);
        var listProduct : Nat = 1;
        List.iterate<Nat>(list, func (x : Nat) {
            listProduct *= x;
        });
        // 3. Lambda Function (Add two numbers)
        let lambdaExample = func (x : Nat, y : Nat) : Nat { x + y };
        let lambdaResult = lambdaExample(arraySum, listProduct);
        // Return results
        return "Array Sum: " # Nat.toText(arraySum) # ", List Product: " #
Nat.toText(listProduct) # ", Lambda Result: " # Nat.toText(lambdaResult);
    };
};
```

- **Explanation:** Here, we take an array of integers and use Array.map to return a new array of their squares. It showcases Motoko's higher-order function support and array manipulation techniques, which are essential when processing NFT data dynamically.

- **Program 3 – divideNumbers.mo**
  - **Purpose:** This code demonstrates exception handling by performing safe division with checks for divide-by-zero errors. This is critical in smart contract logic to prevent unintended crashes or exploits.

```
// CodeMiners
// 1.   Nishant Acharekar   801363902
// 2.   Mihir Phatak    801358341
// 3.   Shubham Kakade      801413848
```

```
// 4.   Calvin Pinto    801320115
// 5.   Deepiga Sengottuvelu Ravichandran   801370629
// 6.   Sandesh Mahajan 801366489

import Float "mo:base/Float";
import Error "mo:base/Error";

actor Program3 {
    // Program 3: Exception Handling
    public func divideNumbers(numerator: Float, denominator: Float) : async Text
{
        try {
            if (denominator == 0.0) {
                throw Error.reject("Division by zero is not allowed");
            };
            let result = numerator / denominator;
            return "Result: " # Float.toText(result);
        } catch (e) {
            return "Error: " # Error.message(e);
        };
    };
};
```

- **Explanation:** This function takes two float numbers and returns the division result if valid, or an error message if the divisor is zero. It demonstrates the use of conditionals and safe mathematical operations, which are critical in NFT pricing or transaction computations.

- **Deployment Details:**
  - All three programs were deployed and tested locally using DFX CLI (Internet Computer SDK).
  - The commands dfx start, dfx deploy, and dfx canister call were used to run and validate the functionality.
  - This step was essential to simulate a production-like environment for validating smart contract interactions before frontend integration.

Key Features Implemented

Our NFT marketplace brings together essential features that make the experience not just decentralized, but also user-friendly and practical for creators and collectors alike. Here's what we've built so far:

a. **NFT Minting & Asset Uploading:** Users can upload their digital artwork, collectibles, or files and mint them as NFTs directly on-chain. This process ensures that every asset is verifiably unique and securely stored using Internet Computer canisters.
b. **Listing NFTs for Sale with Pricing:** Sellers can list their NFTs for sale by setting a price in ICP tokens. This listing is handled entirely through Motoko smart contracts, with logic that validates and records the sale conditions on the blockchain.
c. **Buying & Ownership Transfers via Smart Contracts:** Buyers can explore listed NFTs and purchase them with one click. Behind the scenes, the Motoko contract securely transfers ownership from the seller to the buyer no need for intermediaries. Ownership records are updated instantly on the blockchain.
d. **Personal NFT Gallery:** Once users own NFTs, they can view them in a dedicated “My NFTs” section. This gallery updates in real-time, allowing users to manage their collection and track ownership at a glance.
e. **NFT Discovery Section with Trending NFTs:** We’ve implemented a dynamic discovery gallery that showcases trending, newly minted, and featured NFTs. This section helps improve visibility for creators and provides collectors with a seamless browsing experience.

Instructions to Install and Run the Project

This guide explains how to set up, run, and interact with our decentralized NFT Marketplace on your local machine. Step-by-Step Setup is as follows:

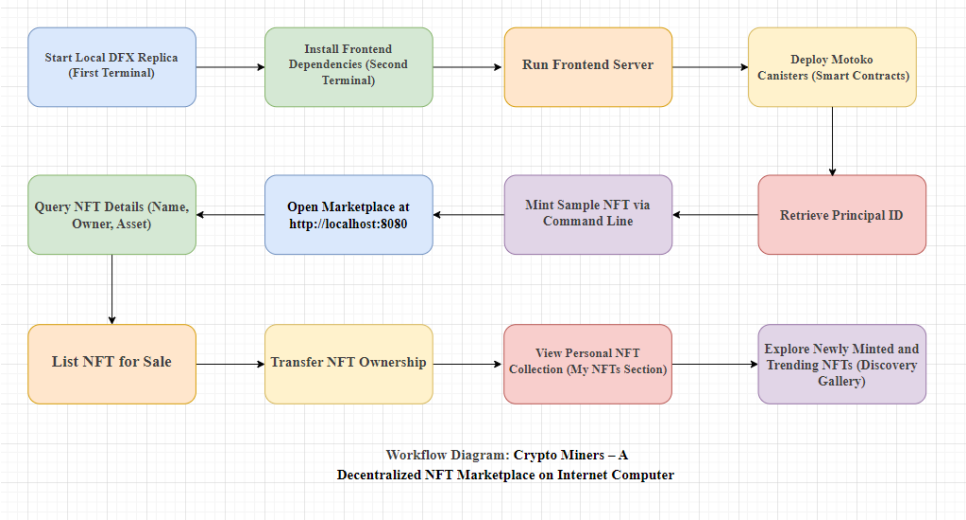


Figure 1: Workflow Diagram showing steps to run the application.

a. Start Local DFX Replica (First Terminal):

i. Open your first terminal window and start a clean replica of the Internet Computer.

ii. Note: Keep this terminal open! This runs the local blockchain environment needed for smart contracts:

A. **dfx start -clean**

b. Install and Run the Frontend (Second Terminal):

i. In a second terminal window, navigate to the project folder and install all frontend dependencies:

A. **npm install**

ii. Then start the frontend development server:

A. **npm start**

iii. **Your React-based UI will now be running locally.**

c. Deploy Canisters:

i. Still in the second terminal, deploy all smart contracts (canisters):

A. **dfx deploy**

ii. This compiles and deploys the Motoko smart contracts to the local Internet Computer environment.

d. Get Your Principal ID:

i. Before minting NFTs, retrieve your principal ID (which identifies your user account) by running:

A. dfx identity get-principal

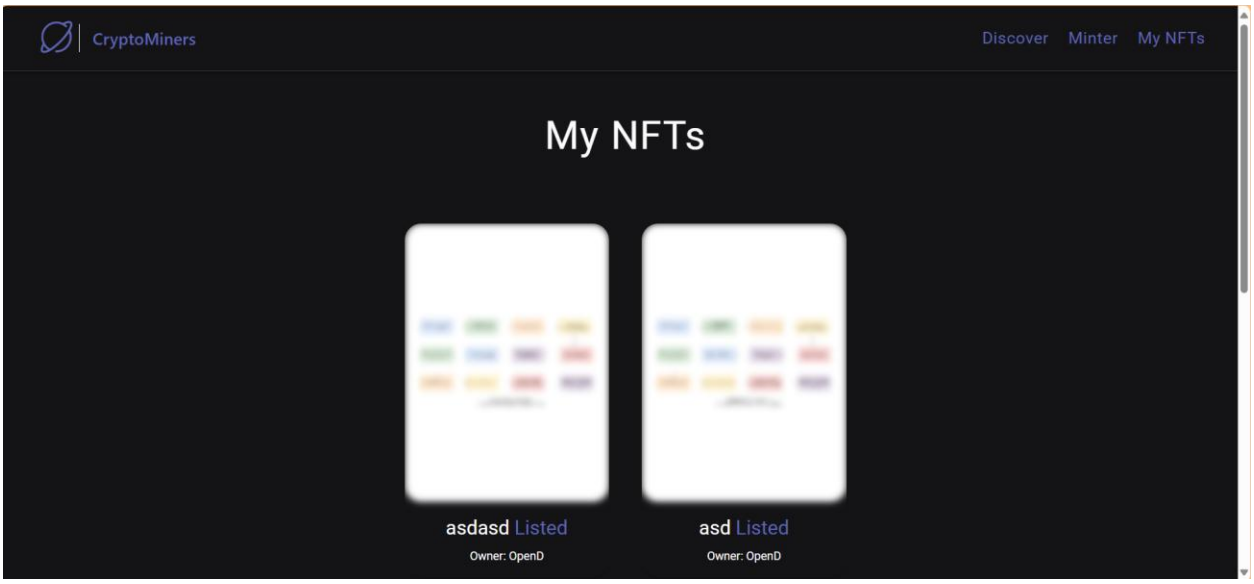
```
nishant@FireEmperor:~/ic-projects/opend-starting/opend$ dfx identity get-principal
hosul-rjfsa-n5acb-pbddr-q3hbc-g7umk-ybwt4-xfyjz-633jk-mm6ar-rqe
```

ii. You'll use this ID when minting NFTs tied to your account.

e. Mint a Sample NFT:

i. After getting your principal ID, mint a sample NFT via the command line.

A. dfx canister call nft mint '("CryptoDunks #123", principal "your-principal-id", (vec {137; 80; 78; 71;}))'



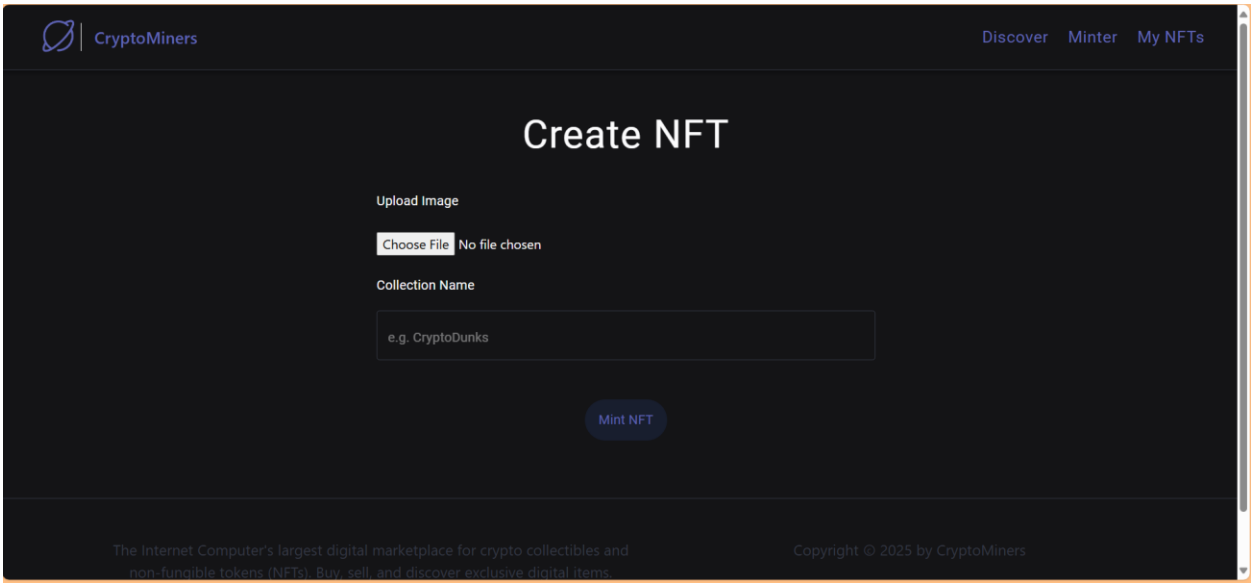
ii. Replace "your-principal-id" with the actual principal you obtained.

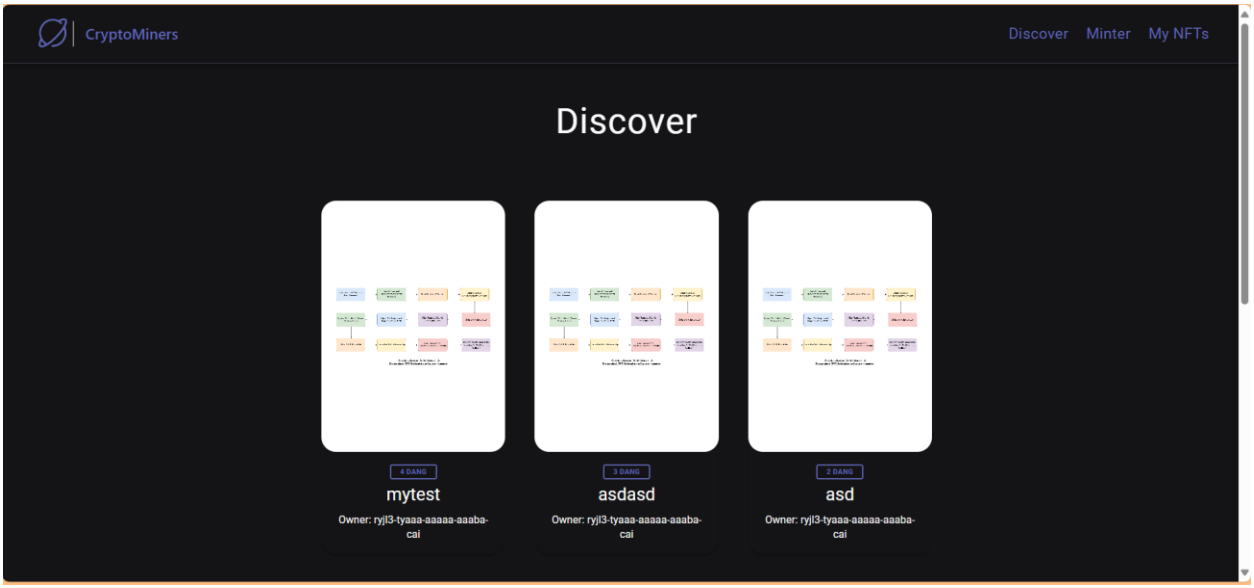
f. Open the Marketplace:

i. Once everything is running, visit:

A. **http://localhost:8080/**

ii. This will open the marketplace UI where you can mint, buy, sell, and view NFTs.





g. **Smart Contract Interaction (Advanced Commands): For deeper interaction and testing:**
    i. Query NFT Details:
        A. **dfx canister call nft getName**
        B. **dfx canister call nft getOwner**
        C. **dfx canister call nft getAsset**
    ii. List an NFT for Sale:
        A. **dfx canister call opend listItem '(principal "<NFT_CANISTER_ID>", 2)'**
    iii. Transfer NFT Ownership:
        A. **dfx canister call <NFT_CANISTER_ID> transferOwnership '(principal "<MARKETPLACE_CANISTER_ID>", true)'**
    iv. Get Canister ID:
        A. **dfx canister id opend**
h. **Replace placeholders (<NFT_CANISTER_ID>, <MARKETPLACE_CANISTER_ID>) with actual canister IDs.**
i. **Creating and Testing NFT Listings:**
    i. To simulate listing and trading NFTs:
        A. **dfx canister call opend mint '(vec {...}, "CryptoDunks #123")'**

ii. List your NFT for sale:
    A. **dfx canister call opend listItem '(principal "NFT_CANISTER_ID", 2)'**
iii. Transfer NFT ownership securely:
    A. **dfx canister call NFT_CANISTER_ID transferOwnership '(principal "MARKETPLACE_CANISTER_ID", true)'**

j. **Important Notes:**
    i. Always run two separate terminals:
        A. **one for dfx start,**
        B. **one for npm install, npm start, and dfx deploy.**
    ii. All deployments and transactions happen on the local Internet Computer replica.
    iii. No cloud services or third-party storage are involved; everything is decentralized by design.

k. **GitHub Repository Link:**
    iv. https://github.com/NishantA9/CryptoMiners

Development Highlights

Throughout the development process, our team focused on building a secure, scalable, and fully decentralized NFT platform. Below are the major technical milestones and highlights from our journey

a. **Smart Contract Modules Deployed & Tested:** We developed and deployed multiple Motoko-based smart contract modules using the DFX CLI on a local replica of the Internet Computer. Each module was carefully tested to ensure reliable execution of core functions like minting, ownership transfers, and secure listing logic.
b. **Frontend-Backend Integration via ICP APIs:** Using Internet Computer APIs, we connected our Motoko smart contracts with a React.js frontend. This allowed

seamless communication between the user interface and on-chain logic, ensuring that every button click triggers an actual blockchain interaction.

c. **Real-Time NFT Rendering with Dynamic UI Updates:** The NFT gallery and listing sections update dynamically as new NFTs are minted, bought, or sold. Users can view changes in real-time without needing to refresh, thanks to efficient state management in the frontend and reactive backend calls.
d. **Version Control and Collaboration via GitHub:** All code, frontend, backend, and deployment scripts-was collaboratively managed using GitHub under the repository CodeMiners. This ensured proper tracking of changes, easy rollbacks, and streamlined collaboration across the team.

Challenges Faced

Like any full-stack decentralized application, building this NFT marketplace came with its own set of technical hurdles. Each challenge helped us better understand the inner workings of the Internet Computer and how to design more resilient smart contracts and interfaces.

a. **Efficient Management of Multiple Canisters:** As each NFT and contract interaction relied on canisters, we faced complexity in managing state, storage, and deployment across multiple modules. Coordinating metadata, listings, and ownership in a scalable manner required careful design to avoid redundancy and ensure reliability.
b. **Smooth Integration Between Frontend and Backend:** Connecting the Motoko backend with the React frontend wasn't plug-and-play. Translating smart contract methods into responsive UI actions required deep knowledge of ICP APIs, candid interfaces, and asynchronous communication models. Debugging these interactions and ensuring consistent behavior across environments was one of the more involved parts of development.
c. **Secure Ownership Transfers & Authentication:** Maintaining user trust meant guaranteeing the security of NFT ownership transfers. Implementing contract

logic that executed atomic transfers, prevented unauthorized access, and protected against bad actors required extensive testing. Additionally, the lack of built-in authentication meant we had to design secure flows without compromising decentralization.

Each challenge shaped the system into a more reliable and user-safe platform and taught us valuable lessons about working with Motoko and the Internet Computer on a scale.

Conclusion

Our decentralized NFT marketplace is a testament to what's possible when powerful technology meets a user-first vision. By combining Motoko's smart contract capabilities with the Internet Computer's scalable, serverless infrastructure, we've created a platform that gives users complete control over their digital assets, without relying on any centralized authority.

This project not only allowed us to explore the full-stack potential of Web3 but also pushed us to think deeply about decentralization, ownership, and user experience. Every component, from minting and purchasing to ownership transfers, runs entirely on-chain, embodying the true ethos of blockchain technology.

2. Future Enhancements

While we've accomplished a lot, this is just the beginning. Our roadmap for future improvements includes:

a. **Live Deployment with Internet Identity:** Integrating secure, passwordless login via Internet Identity to enhance user onboarding and security.
b. **NFT Metadata Standardization:** Ensuring compatibility and interoperability with other dApps by aligning with widely accepted NFT standards.
c. **Advanced Analytics:** Implementing dashboards to help users track performance, trends, and historical data for NFTs.

d. **Cross-Canister & Token Integration:** Expanding marketplace functionality by integrating with other ICP tokens and smart contract ecosystems.

We believe this platform can evolve into a cornerstone of decentralized digital asset management, built on transparency, powered by community, and secured by code.

Acknowledgments & References

We would like to acknowledge the foundational inspiration drawn from the London App Brewery's Web Developer Bootcamp 2024, which provided valuable insights into building NFT marketplaces and smart contracts. Although our project architecture, smart contract logic, and frontend have been extensively customized to fit the decentralized goals of CodeMiners, we are grateful for the educational resources that laid the groundwork.

In addition, this project was developed with reference to several important resources that deepened our understanding of the Internet Computer ecosystem, Motoko programming, and NFT marketplace standards

a. Internet Computer Developer Documentation

b. Motoko Programming Language Guide

c. NFT Marketplace Standards Overview

This project proudly reflects a blend of open-source learnings, our own technical innovations, and a strong commitment to decentralized ownership principles. We sincerely thank all the contributors in the Internet Computer and Web3 communities for their tools, documentation, and support.