Final Project Instructions

Objective

The goal of this final project is to provide students with hands-on experience in the development of all aspects of Non-Fungible Tokens (NFTs). In this project, students will cover the NFT lifecycles from minting, trading, indexing, querying, and analyzing. To accomplish this, one is expected to complete the following three steps:

1. Develop a secure and efficient NFT marketplace smart contract that supports the creation and trading of NFTs. The smart contract should allow users to create, buy, sell, and transfer NFTs in a secure and efficient manner.
2. Use GraphQL to index the NFT marketplace smart contract and develop query APIs to access the indexed data. The GraphQL API should provide a simple and efficient way to retrieve information about NFTs, such as their ownership, transaction history, and metadata.
3. Collect, analyze, and visualize transaction data, metadata, and associated IPFS images from the NFT marketplace based on various properties, such as categories, design studios, and sales history. The data analysis should provide insights into NFT trends, such as popular categories, top sellers, and average sale prices.

By the end of this project, students will understand a functional NFT marketplace built on a blockchain platform; a functional GraphQL API that interacts with the NFT marketplace smart contract, providing a simple and efficient way to retrieve information about NFTs and their associated transactions; a deeper understanding of NFT trends.

Project Requirements

1. Final report due date: **May 8th, 2022**. Project team members: <= 6 members; For a larger team, more contributions and completeness are expected.

2. Each team should select **one topic** from the following three topics. The development should be based on **Ethereum**. We have created discord channels for the students to consult the TAs and Professors for questions and instructions. Please join us at https://discord.gg/oortech. For topic 1, join channel **#topic-1-marketplace**; for topic 2, join channel **#topic-2-graphql**; for topic 3, join channel **#topic-3-analysis**.

3. You need to submit a final report of **no more than 5 pages**, excluding references and figures. Only one member in your team needs to submit the report. All authors' UNI needs to be included in the report.

4. You need to submit your code to https://github.com/oort-tech/NFT_data_warehouse . Your github link should be included in your final report. Each group please submit your code to a subfolder.

5. Your final report should follow the structure below:
    5.1. Title, Authors
    5.2. Summary of your work: procedure, methodology, results, etc.
    5.3. Showcase of your work (e.g., figures showing the outcome of your program, or one example of user's experience)
    5.4. Conclusion (including contributions of each team member)

Grading Policy

1. Completeness: 40%
2. Report Writing/presentation: 30%
3. Code quality/efficiency: 30%

Topic 1: NFT Marketplace Smart Contract Development

Overview

The NFT marketplace smart contract will allow users to mint, buy, sell, and trade unique digital assets represented as NFTs. The project will involve creating an ERC-721 or ERC-1155 compliant smart contract, designing a user interface for the marketplace, and implementing additional features to ensure the marketplace's security and usability.

Outline

1. Setting Up the Development Environment
    - Installation of necessary tools (Visual studio code, Node.js, Truffle, Ganache, MetaMask)
        i. Visual studio code is the IDE for smart contract development
            1. Configuring Visual Studio code for Ethereum Blockchain Development
        ii. Node.js is commonly used in blockchain development to build decentralized applications (dApps) and to interact with blockchain networks through APIs.
        iii. Truffle is the development framework for building and testing smart contracts on blockchain platforms.
        iv. Ganache is a personal blockchain for Ethereum development that allows developers to simulate an Ethereum network on their local machine.
        v. MetaMask is a browser extension that allows users to interact with decentralized applications (dApps) on the Ethereum blockchain.
        vi. A tutorial on blockchain developer toolkit can be found below
            1. The Complete Blockchain Developer Toolkit
    - Creating a new Truffle project
        i. How to create a project?
2. Designing the NFT Marketplace Smart Contract
    - Understanding ERC-721 and ERC-1155 standards
        i. Articles that explain the difference between ERC-721 and ERC-1155 standards.
            1. ERC 1155 Vs. ERC 721 – Key Differences
            2. ERC-721 VS ERC-1155: WHICH IS BETTER FOR NFT MARKETPLACES?
    - Defining the structure of the smart contract
        i. How to compile a smart contract?
    - Implementing the minting, ownership, and metadata functions
3. Deploying the NFT Marketplace
    - Preparing the smart contract for deployment
    - Deploying the smart contract to a test network (e.g., Rinkeby)

- How to deploy smart contracts?

4. Write test cases for the NFT market place smart contracts. How to test your contracts? Here is a list of test cases:
    - Test that the smart contract can create a new NFT:
        i. Create a new instance of the smart contract
        ii. Call the createNFT function with a unique ID, name, and description
        iii. Assert that the NFT was created and its properties match the input values
    - Test that the smart contract can transfer ownership of an NFT:
        i. Create two user accounts
        ii. Create a new NFT with the first user account
        iii. Transfer ownership of the NFT to the second user account using the transferNFT function
        iv. Assert that the NFT is now owned by the second user account
    - Test that the smart contract can list an NFT for sale:
        i. Create a new user account
        ii. Create a new NFT with the user account
        iii. List the NFT for sale using the listNFTForSale function with a sale price
        iv. Assert that the NFT is now listed for sale and its sale price matches the input value
    - Test that the smart contract can remove an NFT from sale:
        i. Create a new user account
        ii. Create a new NFT with the user account
        iii. List the NFT for sale using the listNFTForSale function with a sale price
        iv. Remove the NFT from sale using the removeNFTFromSale function
        v. Assert that the NFT is no longer listed for sale
    - Test that the smart contract can execute a successful NFT purchase:
        i. Create two user accounts
        ii. Create a new NFT with the first user account
        iii. List the NFT for sale using the listNFTForSale function with a sale price
        iv. Purchase the NFT using the purchaseNFT function with the second user account and the correct amount of Ether
        v. Assert that the NFT is now owned by the second user account and the correct amount of Ether was transferred to the first user account
    - Test that the smart contract can execute an unsuccessful NFT purchase:
        i. Create two user accounts
        ii. Create a new NFT with the first user account
        iii. List the NFT for sale using the listNFTForSale function with a sale price
        iv. Attempt to purchase the NFT using the purchaseNFT function with the second user account but with an incorrect amount of Ether
    - Assert that the NFT ownership remains with the first user account and no Ether was transferred

Deliverables

1. A fully functional NFT marketplace smart contract

Evaluation Criteria

1. Compliance with ERC-721 or ERC-1155 standards
2. Security and efficiency of the smart contract
3. Quality and clarity of project presentation and documentation

Topic 2: Building a GraphQL API for OpenSea Smart Contract Indexing

Overview

The project will involve understanding the OpenSea smart contract, setting up a GraphQL server, and developing schema and resolvers to index and query the contract data. The resulting GraphQL API will provide users with the ability to query NFT data, sales history, and transactions with ease.

Outline

1. Background: Introduction to OpenSea and GraphQL
    - Overview of OpenSea and the NFT market
    - Introduction to smart contracts and the Ethereum blockchain
    - Basics of GraphQL and its advantages over traditional REST APIs
2. Setting Up the Development Environment
    - Install Node.js: Before you can start developing with GraphQL, you need to install Node.js on your machine. You can download and install Node.js from the official website.
    - Create a new project: Once Node.js is installed, you can create a new project folder and navigate to it in your terminal. You can use the following command to create a new project folder:

```
Unset
mkdir my-graphql-project
cd my-graphql-project
```

    - Initialize a new npm package: Next, you need to initialize a new npm package in your project folder. You can use the following command to do this:

```
Unset
npm init -y
```

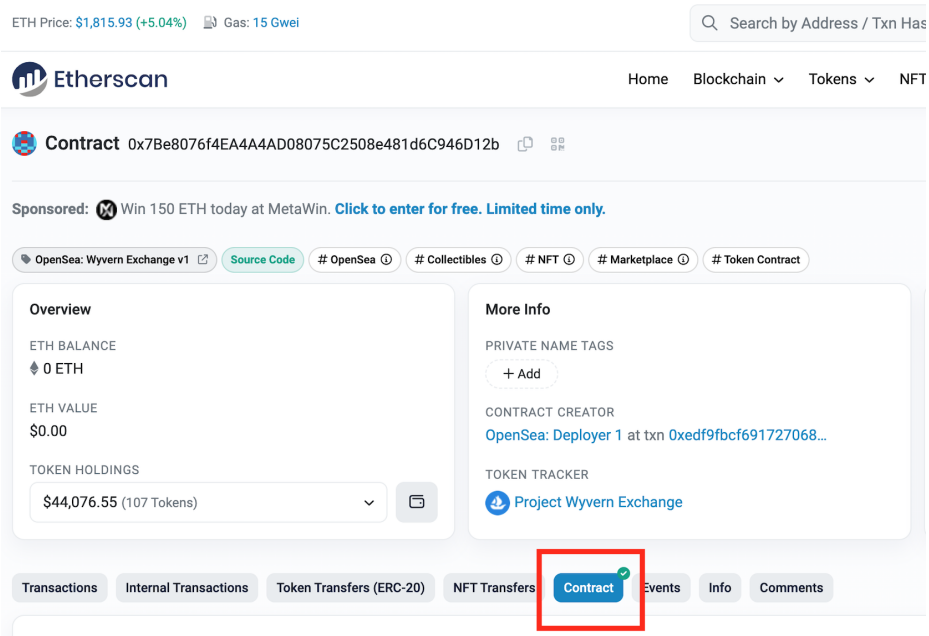
- Install necessary dependencies: You need to install the necessary dependencies for a GraphQL development environment. Some commonly used dependencies include express, graphql, and apollo-server-express. You can install these dependencies by running the following command:

```
Unset
npm install express graphql apollo-server-express
```

- Create a GraphQL schema: A GraphQL schema defines the types and queries that your GraphQL API will support. You can create a new file called schema.graphql in your project folder and define your schema in this file.
- Create a GraphQL resolver: A resolver maps a GraphQL query to a specific function that returns the data. You can create a new file called resolvers.js in your project folder and define your resolvers in this file.
- Create an Express server: Express is a popular Node.js web framework that is often used in conjunction with GraphQL. You can create a new file called server.js in your project folder and define your Express server in this file.
- Start the server: Finally, you can start your GraphQL server by running the following command in your terminal:

```
Unset
node server.js
```

3. Understanding the OpenSea Smart Contract
    - Analyzing the OpenSea smart contract structure and functions
        i. Opensea smart contract Click the “contract” button to read the smart contract source code.



- Identifying relevant data points (NFT details, transactions, sales history)
- Accessing the smart contract using Web3.js
    i. How to interact with smart contract?

4. Designing the GraphQL Schema for opensea smart contracts
    - Define the data types: Start by identifying the core data types that the OpenSea smart contract will expose. These could include types such as "Asset," "Collection," "User," "Order," and "Transaction."
    - Define the fields for each type: For each type, define the fields that represent the data that you want to expose. For example, an "Asset" type might have fields such as "tokenId," "name," "description," "image," "owner," "collection," and "traits."
    - Define the relationships between types: Specify the relationships between the types by defining the fields that link them. For example, an "Asset" might be linked to a "Collection" through a "collection" field, and a "User" might be linked to an "Asset" through an "owner" field.
    - Define the queries: Define the queries that will allow users to retrieve data from the OpenSea smart contract. For example, you might define queries such as "getAssetById," "getAssetsByOwner," and "getOrdersByAsset."
    - Define the mutations: Define the mutations that will allow users to create, update, or delete data in the OpenSea smart contract. For example, you might define mutations such as "createOrder," "cancelOrder," and "transferAsset."
    - Define any custom directives: Use custom directives to modify the behavior of fields or types. For example, you might use a custom directive to specify that a field can only be accessed by authenticated users.
    - Test the schema: Test the schema using a tool such as GraphQL to ensure that it works as expected.
5. Implementing GraphQL Resolvers
    - Creating resolvers to interact with the OpenSea smart contract

- Handling errors and edge cases
- Optimizing resolver performance using batching and caching techniques

Here is an example of using the opensea-js library to interact with the OpenSea API.

JavaScript

```
const { ApolloServer, gql } = require('apollo-server');
const OpenSea = require('opensea-js');
const openSeaApiUrl = 'https://api.opensea.io/graphql/';

// Initialize the OpenSea SDK
const openSea = new OpenSea(openSeaApiUrl);

// Define the GraphQL schema
const typeDefs = gql`
  type Asset {
    tokenId: ID!
    name: String!
    description: String
    imageUrl: String
    owner: User
    collection: Collection
    traits: [Trait!]!
  }

  type User {
    address: ID!
    assets: [Asset!]!
  }

  type Collection {
    slug: String!
    name: String!
    description: String
    imageUrl: String
    assets: [Asset!]!
  }

  type Trait {
```

```
    traitType: String!
    value: String!
  }

  type Query {
    assetById(tokenId: ID!): Asset
    assetsByOwner(ownerAddress: ID!): [Asset!]!
    collectionBySlug(slug: String!): Collection
  }
`;

// Define the resolvers for the schema
const resolvers = {
  Query: {
    // Resolver for the "assetById" query
    assetById: async (_, { tokenId }) => {
      const { asset } = await openSea.api.getAsset({
token_id: tokenId });
      return asset;
    },

    // Resolver for the "assetsByOwner" query
    assetsByOwner: async (_, { ownerAddress }) => {
      const { assets } = await openSea.api.getAssets({
        owner: ownerAddress,
      });
      return assets;
    },

    // Resolver for the "collectionBySlug" query
    collectionBySlug: async (_, { slug }) => {
      const { collection } = await
openSea.api.getCollection({ slug });
      return collection;
    },
  },
```

```
  Asset: {
    // Resolver for the "owner" field of the "Asset" type
    owner: async (asset) => {
      const { owner } = await openSea.api.getAsset({
token_id: asset.tokenId });
      return owner;
    },

    // Resolver for the "collection" field of the "Asset"
type
    collection: async (asset) => {
      const { collection } = await openSea.api.getAsset({
token_id: asset.tokenId });
      return collection;
    },
  },

  User: {
    // Resolver for the "assets" field of the "User" type
    assets: async (user) => {
      const { assets } = await openSea.api.getAssets({
owner: user.address });
      return assets;
    },
  },

  Collection: {
    // Resolver for the "assets" field of the "Collection"
type
    assets: async (collection) => {
      const { assets } = await openSea.api.getAssets({
        collection: collection.slug,
      });
      return assets;
    },
```

```
  },
};

// Create the Apollo Server instance
const server = new ApolloServer({
  typeDefs,
  resolvers,
});

// Start the server
server.listen().then(({ url }) => {
  console.log(`🚀  Server ready at ${url}`);
});
```

6. Testing and Deployment
    - Writing tests for the GraphQL API using Jest
        i. Install dependencies

```
JavaScript
npm install --save-dev jest supertest graphql
```

This installs Jest as the testing framework, Supertest for HTTP requests, and GraphQL for building and executing queries.

ii. Create a new test file:

```
JavaScript
mkdir __tests__
touch __tests__/graphql.test.js
```

This creates a new directory for tests and a new file for the GraphQL tests.

iii. Import necessary modules:

```
JavaScript
const { describe, it, expect } =
require('@jest/globals');
const request = require('supertest');
```

```
const { graphql } = require('graphql');
const schema = require('../schema');
```

Here, we import the necessary modules for testing, including Jest's describe, it, and expect functions, Supertest's request function for making HTTP requests, and the graphql and schema modules for building and executing queries.

iv. Write the test cases:

JavaScript

```
describe('GraphQL API tests', () => {
  it('should return a list of items', async () => {
    const query = `
      query {
        items {
          id
          name
          price
        }
      }
    `;
    const expectedResponse = {
      data: {
        items: [
          {
            id: '1',
            name: 'Item 1',
            price: 10.0
          },
          {
            id: '2',
            name: 'Item 2',
            price: 20.0
          }
        ]
      }
    };
    const result = await graphql(schema, query);
```

```
    expect(result).toEqual(expectedResponse);
  });
});
```

In this example, we write a single test case that queries for a list of items and expects to receive a specific response. We define the GraphQL query, the expected response, and then use graphql(schema, query) to execute the query against the schema defined in schema.js. We then use Jest's expect function to compare the actual result to the expected result.

v. Run the test

```
Unset
jest
```

This runs all the tests in the __tests__ directory using Jest.

- A few Example test cases for the OpenSea smart contract
    i. Verify that a user can retrieve their own NFTs from the OpenSea smart contract using their wallet address.
    ii. Test that a user can retrieve a specific NFT from the OpenSea smart contract using its token ID.
    iii. Ensure that a user cannot retrieve an NFT that they do not own from the OpenSea smart contract.
    iv. Test that a user can list an NFT for sale on the OpenSea smart contract.
    v. Verify that a user can cancel an NFT listing on the OpenSea smart contract.
    vi. Test that a user can place a bid on an NFT that is listed for sale on the OpenSea smart contract.
    vii. Ensure that a user cannot place a bid that is less than the minimum bid amount on an NFT listed for sale on the OpenSea smart contract.
    viii. Test that a user can withdraw a bid that they have placed on an NFT listed for sale on the OpenSea smart contract.
    ix. Verify that a user can purchase an NFT that is listed for sale on the OpenSea smart contract.
    x. Test that a user can transfer an NFT they own to another user on the OpenSea smart contract.

7. Project Presentation and Documentation
    - Preparing a presentation of the project's key features and functionalities
    - Writing comprehensive documentation, including user guides and technical details

Deliverables

1. A fully functional GraphQL API for indexing and querying the OpenSea smart contract

2. Comprehensive tests

Evaluation Criteria

1. Correctness and completeness of the GraphQL schema and resolvers
2. Efficiency and performance of the GraphQL API
3. Quality and clarity of project presentation and documentation

Topic 3: Discovering NFT Trends with OpenSea Data, IPFS, and Property Analysis

Overview

The project will involve extracting NFT transaction data and metadata from OpenSea, retrieving NFT images from IPFS, and using data analysis and visualization techniques to identify trends and patterns based on different properties.

Outline

1. Background: Introduction to NFTs, OpenSea, and IPFS
    - Overview of NFTs, their use cases, and the NFT market
    - Introduction to OpenSea and its role in the NFT ecosystem
    - Basics of IPFS and its advantages for decentralized storage
2. Data Collection and Storage
    - Extracting NFT transaction data and metadata from OpenSea using their API Opensea API documentation
    - Retrieving NFT images or videos from IPFS using their content-addressed hashes
        i. Identify the IPFS hash: The first step is to identify the content-addressed hash that represents the NFT image on IPFS. This hash is typically provided in the NFT metadata or as part of the transaction data.
        ii. Construct the HTTP gateway URL: Once you have the IPFS hash, you can construct the URL for the HTTP gateway using the following format:

```
Unset
https://gateway.ipfs.io/ipfs/<hash>
```

This URL will direct your request to the IPFS HTTP gateway, which will retrieve the data for the specified hash.

iii. Send the HTTP request: You can use a tool like axios or the built-in fetch function to send an HTTP GET request to the constructed URL and retrieve the NFT image data. Here is an example using axios:

JavaScript

```
const axios = require('axios');

const hash = '<content-addressed hash>';
const url = `https://gateway.ipfs.io/ipfs/${hash}`;

axios.get(url, { responseType: 'arraybuffer' })
  .then(response => {
    const imageData = response.data;
    // process the image data here
  })
  .catch(error => {
    console.error(error);
  });
```

In this example, we use axios to send a GET request to the constructed URL and specify the responseType as arraybuffer to ensure that we receive binary image data. We then handle the response and process the image data as needed. If the request fails, we handle the error in a catch block.

iv. Process the image data: Once you have the NFT image data, you can process it as needed. For example, you can save it to disk, display it on a webpage, or manipulate it using an image processing library.

- Storing the collected data in a suitable database (e.g., MongoDB, PostgreSQL)

3. Data Preprocessing and Feature Extraction
   - Cleaning and organizing the collected data for analysis
   - Identifying relevant properties and features for NFT trend analysis (e.g., categories, design studios, intellectual property holders, sales history). There are many approaches to derive the properties from the images. For example,
     i. The NFT metadata
     ii. Any open source image classification software that fits you best. Free image recognition software
   - Extracting and encoding features from the data for further analysis
4. Data Analysis and Trend Discovery
   - Analyzing the data using statistical techniques and machine learning algorithms to identify patterns and trends based on different properties
   - Investigating correlations and relationships between properties and NFT market performance
   - Identifying emerging trends and opportunities in the NFT market
5. Documentation and Presentation

- Writing comprehensive documentation detailing the data collection, analysis, and visualization process
- Preparing a presentation highlighting key findings and insights from the NFT property-based trend analysis
- Sharing the project with the community and gathering feedback for improvement

Deliverables

1. A clean and organized dataset of NFT transaction data, metadata, and associated IPFS image hashes
2. Insightful analysis on NFTs.

Evaluation Criteria

1. Completeness and quality of the data collection and preprocessing
2. Effectiveness of the data analysis techniques in identifying NFT trends and patterns based on properties
3. Quality and comprehensiveness of project documentation and presentation