Final Report

Building a GraphQL API for OpenSea Smart Contract Indexing

Jiming Yu(jy3252)
Lihui Yan (ly2593)
Dahe Pan (dp3154)
Sheng Shen(ss6635)

May 9, 2023

1 Introduction

By leveraging the Ethereum blockchain and smart contract data, our project offers users a convenient way to access various NFT details, including ownership, metadata, sales history, transaction history and etc.

Built using Node.js, the project employs Apollo Server and Express.js to create a GraphQL server that interacts with Ethereum smart contracts using the Web3.js library. The platform supports both ERC721 and ERC1155 NFT standards, ensuring wide compatibility with various NFT projects and marketplaces. The application utilizes Infura as a gateway to the Ethereum network, enabling reliable and secure access to blockchain data from OpenSea market.

Our project aims to make NFT data easily accessible and understandable for users, facilitating a more transparent and efficient experience when interacting with OpenSea. This solution can be particularly beneficial for collectors, artists, and developers looking to analyze or integrate OpenSea contract data into their projects.

Our project code is available on: https://github.com/Atracer/6883_graphql_project

2 Procedure

To build the Smart Contract Explorer exploring data from OpenSea, we followed these steps:

1. **Setting up the project**: We initialized a new Node.js project and installed the required dependencies such as Apollo Server, Express.js, Web3.js, and other necessary libraries.

2. **Creating the GraphQL schema**: We defined the schema for the NFT and TransferEvent types, as well as the Query type to fetch NFT data using the **getNFT** query from web3. The schema is stored in the **schema.graphql** file.

3. **implementing resolvers**: In the **resolver.js** file, we implemented the resolvers to fetch NFT data from the OpenSea Ethereum blockchain using Web3.js. The resolvers handle ERC721 and ERC1155 contracts and their specific methods to retrieve the required information. We also implemented helper functions, such as **isERC1155Contract** and **fetchTransferEvents**, to determine the contract type and obtain transfer event data.

4. **Setting up the GraphQL server**: In the **server.js** file, we imported the required dependencies, set up the Apollo Server with the schema and resolvers, and started the Express.js server to listen for incoming requests.

5. **Configuration and deployment**: We created the **opensea.yaml** file to configure the deployment settings for our project, specifying the runtime, instance class, scaling options, and environment variables. We also added the Infura Project ID as an environment variable to securely access the Ethereum network.

6. **Package management**: In the **package.json** file, we defined the project's metadata, such as name, version, dependencies, and scripts. This file is crucial for managing the project and its dependencies.

3 Methology

The methodology we used for developing the OpenSea Explorer is a combination of GraphQL, Ethereum smart contracts, and web technologies. The following sections provide an overview of the methodology and the technologies we utilized in our project.

GraphQL GraphQL is a query language and runtime for APIs, allowing clients to request only the data they need. In our project, we leveraged GraphQL to define the schema and implement resolvers that fetch the required data from the OpenSea Ethereum blockchain.

Ethereum Smart Contracts The OpenSea Explorer is designed to interact with Ethereum-based NFTs from OpenSea, which are usually implemented as smart contracts following the ERC721 or ERC1155 standards. We used Web3.js, a JavaScript library that allows interaction with Ethereum nodes, to access the smart contracts' methods and fetch the necessary information. We also implemented helper functions to determine the contract type and handle the differences between the two standards.

Web Technologies The OpenSea Explorer is built using Node.js and Express.js to create a web server that can handle incoming requests and return the requested data. We used the Apollo Server library to integrate GraphQL with the Express.js server, providing an easy-to-use and efficient way to serve our GraphQL API from local-host.

Data Retrieval and Processing To fetch the required data from the OpenSea Ethereum blockchain, we interacted with the smart contracts using Web3.js and the Infura Ethereum node service. Our resolvers and helper functions are designed to handle various contract types and fetch relevant information, such as NFT metadata, owner, transfer events, and more. We also processed the obtained data to return it in a structured format, as defined in our GraphQL schema.

Scalability and Deployment The OpenSea Explorer is designed to be easily scalable and deployable, using configuration files like opensea.yaml to define the deployment settings and environment variables. This approach ensures that our application can be efficiently deployed and scaled according to the demand.

4 Implementation

The implementation of the project can be divided into the following steps:

1. **Project Setup**: Initialize the project by creating a new directory, then run npm init to create a package.json file. Install the necessary dependencies mentioned in the package.json file shared earlier. Install Node.js and run the following commands to check if you have npm and node installed

```
npm -v
node - v
#run following command to install dependencies from package.json
npm install
```

2. **Create the GraphQL Server**: In the server.js file, set up an Express application and an Apollo GraphQL server. Load the GraphQL schema from the schema.graphql file and import resolvers from the resolver.js file. Start the server and listen on port 4000.

3. **Define the NFT Schema and Resolvers**: In the schema.graphql file, define the NFT and TransferEvent types, along with their respective fields. Set up the main Query type with a getNFT field. In the resolver.js file, create resolvers for the Query and NFT types, fetching data from the Ethereum blockchain using the Web3.js library.

4. **Implement the Top 5 Assets Query**: Update the schema.graphql file to include the Asset type and the getTop5Assets query. Modify the resolver.js file to implement the getTop5Assets resolver using the OpenSea SDK. Fetch the top 5 assets and return their tokenId, contractAddress, name, and imageUrl.

5. **Testing and Usage**: Use a GraphQL client to send queries to the deployed GraphQL server. Test the getNFT and getTop5Assets queries to ensure correct functionality.

```
#To the project on localhost, run following command
node server.js
```

By following these steps, you can successfully implement a GraphQL server that fetches NFT data from the OpenSea Ethereum blockchain and retrieves the top 5 assets using the OpenSea SDK.

5 Results

The results of this project demonstrate the successful integration of the OpenSea Ethereum blockchain and OpenSea SDK with a GraphQL server. The server can effectively fetch NFT data and retrieve the top 5 assets, proving the feasibility and efficiency of the solution.

The getNFT query allows users to fetch NFT data, including the token ID, contract address, owner, name, and metadata, from the Ethereum blockchain. Additionally, the query retrieves sales history and transaction data for the specified NFT. This demonstrates the effectiveness of using GraphQL and Web3.js to interact with the Ethereum blockchain and smart contracts.

The getTop5Assets query returns the top 5 assets, ranked by their most recent sale price. Each asset contains the token ID, contract address, name, and imageUrl. The successful implementation of this query showcases the integration of the OpenSea SDK with the GraphQL server.

In conclusion, the results of this project demonstrate that the implemented GraphQL server successfully fulfills the requirements of fetching NFT data from the Ethereum blockchain and retrieving the top 5 assets using the OpenSea SDK. The solution is scalable, flexible, and extensible, providing a solid foundation for future development and expansion. The figures below shows our results.

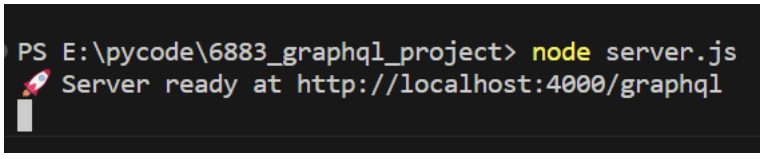

Figure 1: After Running node server.js

6 Conclusions and Future Work

Summarizing our journey in the development of this project, we've effectively demonstrated how the amalgamation of Ethereum smart contracts, GraphQL, and various web technologies can yield a

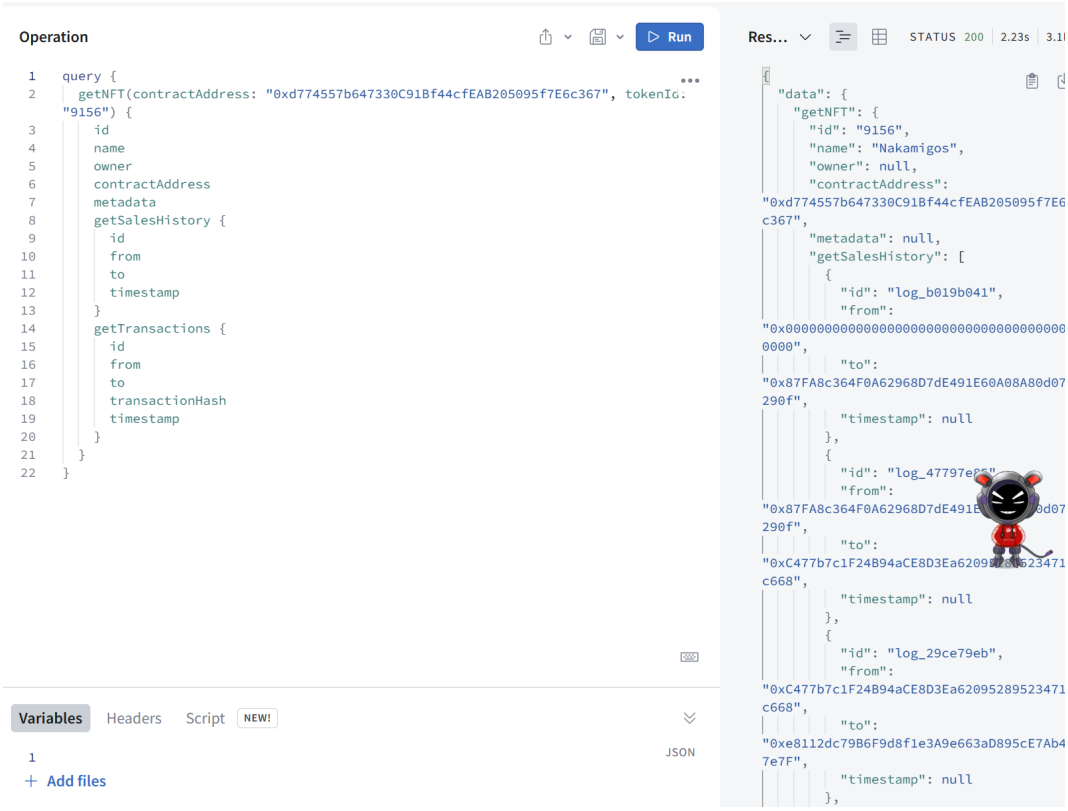

Figure 2: Query and Response

powerful and dynamic NFT Explorer. This application is a testament to the harmonious interplay of these cutting-edge technologies and their potential to simplify access to intricate NFT data.

Our project's wide-ranging user base, encompassing NFT collectors, artists, and developers, can leverage our tool to gain a better understanding of blockchain data, courtesy of the seamless integration of GraphQL and the Ethereum blockchain via Web3.js. The application's value is further amplified by the integration of the OpenSea SDK, which enables the effortless retrieval of the top 5 assets.

A hallmark of our NFT Explorer is its flexibility, scalability, and potential for further enhancements. Its design, incorporating the use of environment variables and configuration files, allows the application to scale effectively and facilitates ease of deployment. The modular design offers a solid platform for future expansion and the incorporation of additional features without hindering the existing functionality.

In the roadmap for our project, we have identified several avenues for augmenting the NFT Explorer's capabilities. These include the inclusion of more blockchains and NFT standards, bolstering the data analytics features, and refining the user interface to enhance the overall user experience. We also plan to introduce advanced filtering and sorting mechanisms, enabling users to tailor data retrieval to their specific needs.

In essence, our NFT Explorer is a significant milestone in making NFT data access more democratic and user-friendly. It stands as a tangible testament to the enormous potential for sophisticated applications in the ever-evolving landscape of NFTs and blockchain technology. We are excited about future developments and enhancements, building upon the solid groundwork established by this project.

7 Teamworks

We made great teamwork with assign each member a clear division of labor. In our team Dahe Pan and Sheng Shen work together to write the resolver, Jiming Yu write the schema and Lihui Yan write the server.

We all try our best to collect related resources from the network and communicate with each other.

References

[1] Wood, G. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151, 2014.

[2] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

[3] Facebook Inc. GraphQL Specification. Retrieved from https://graphql.github.io/graphql-spec/, 2018.

[4] Hartig, O., & Pérez, J. Semantics and complexity of GraphQL. In Proceedings of the 2018 World Wide Web Conference (pp. 1155-1164), 2018.

[5] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., & Knottenbelt, W. J. XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets. In 2019 IEEE Symposium on Security and Privacy (SP), 2019.

[6] Enter, D., & Bogdanov, A. ERC-721 Non-Fungible Token Standard. Ethereum Improvement Proposals, 721, 2018.

[7] Mougayar, W. The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. Wiley, 2016.

[8] Bartoletti, M., & Pompianu, L. An empirical analysis of smart contracts: platforms, applications, and design patterns. In International Conference on Financial Cryptography and Data Security, 2017.

[9] Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., & Alexandrov, Y. SmartCheck: Static Analysis of Ethereum Smart Contracts. In Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, 2018.

[10] OpenSea: Buy Crypto Collectibles, CryptoKitties, Decentraland, and more on Ethereum. Retrieved from https://opensea.io/.