Repositories for Data Monetization

An Eco-system for Distributed Management of Data and Monitoring of Use

Yumin Zhao
Matr.-Nr.: 368828
E-Mail: yumin.zhao@rwth-aachen.de

Bachelorarbeit in Informatik
vorgelegt der
Fakultät für Mathematik, Informatik, Naturwissenschaften
RWTH Aachen

Erstgutachter: Prof. Dr. Thomas Rose
Zweitgutachter: Prof. Dr. Wolfgang Prinz

Fraunhofer Institute for Applied Information Technology FIT

July 18, 2022

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Zhao, Yumin — Name, Vorname/Last Name, First Name

368828 — Matrikelnummer (freiwillige Angabe) / Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

Repositories for Data Monetization - An Eco-system for Distributed Management of Data and Monitoring of Use

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, den 18.07.2022 — Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, den 18.07.2022 — Ort, Datum/City, Date

Unterschrift/Signature

Abstract

In the present era of digital communication, more and more data is collected through the numerous connected devices in our everyday of life, which leads to a surplus of data. These data are just stored mostly in central data storage and it is difficult to search for the relevant information. The most successful companies are already analyzing data from their industry sector. A structured data repository as a SaaS creates franchises and companies with shared data pools, a platform to integrate, exchange and analyze data sets. In this thesis, a decentralized repository for data management and monitoring data is implemented on an Ethereum testnet. This eco-system is built with ReactJS and Moralis on top of the Ethereum-boilerplate. On the platform, users should be able to buy, access, sell, and provide data sets with their connected crypto wallet. Access control to purchased data sets through the ERC-1155 token is here introduced and implemented. Further, different ERC tokens and smart contracts are proposed and implemented. The usage of the data sets can be logged on the blockchain, which interacts as a digital ledger technology. An incentive system should motivate data users and data providers to work with this platform. In the evaluation chapter, the transaction costs related to gas usage, gas price and Ethereum price are calculated and elaborated in favor to improve the system and analyze the performance. Additionally, the user experience and the user-friendliness toward the platform are discussed and concluded in the last chapter.

Contents

1 Introduction

1.1 Motivation

Big data has fundamentally changed the contemporary corporate landscape over the past few years. As statistics on big data in business show, that digital transformation and technological advances are still significant drivers behind increased Big Data expenditures [1]. During the past 10 years, the volume of data created, captured, copied, and consumed worldwide increased from 2 zettabytes by 2010 to 64.2 by 2020 and is projected to 181 zettabytes in 2025 [2]. Especially, in the Internet of Things (IoT) industry, the data volume reached 13.6 zettabytes by 2019 and will grow to 79.4 zettabytes by 2025 [3]. Altogether with the COVID-19 pandemic in 2019 - 2022, the volume and complexity of generated and collected data are increasing, in particular data sets in cloud computing, artificial intelligence, and machine learning.

Fig. 1.1 shows the immense number of sourced data from a range of online-services and internet-connected devices, for example, smartphones, TVs, speakers with an assistant system, smart fridges, and more. Modern devices generate a large amount of data, which can be analyzed and used for personalization or optimization purposes. These large amounts of various data created, captured, copied, and consumed are defined under the term big data [4]. The authors from [5] describe the term as multiple large data collections which are difficult to process and store under commonly used systems. We must implement complex algorithms and patterns to process and store these large data sets. The objective is to analyze and visualize the data sets to transform them into comparable data. Commercial businesses can use these processed data sets to analyze and optimize various internal and external strategies.

For example, the Siri suggestions function in Apple's iOS software collects a large number of user preference and behavior to predict apps that the user will probably use next [6]. This functionality improves the user experience with the software and leads to better usability of their devices which increases sales. Also, the increased volume, variety, veracity, and velocity are reasons

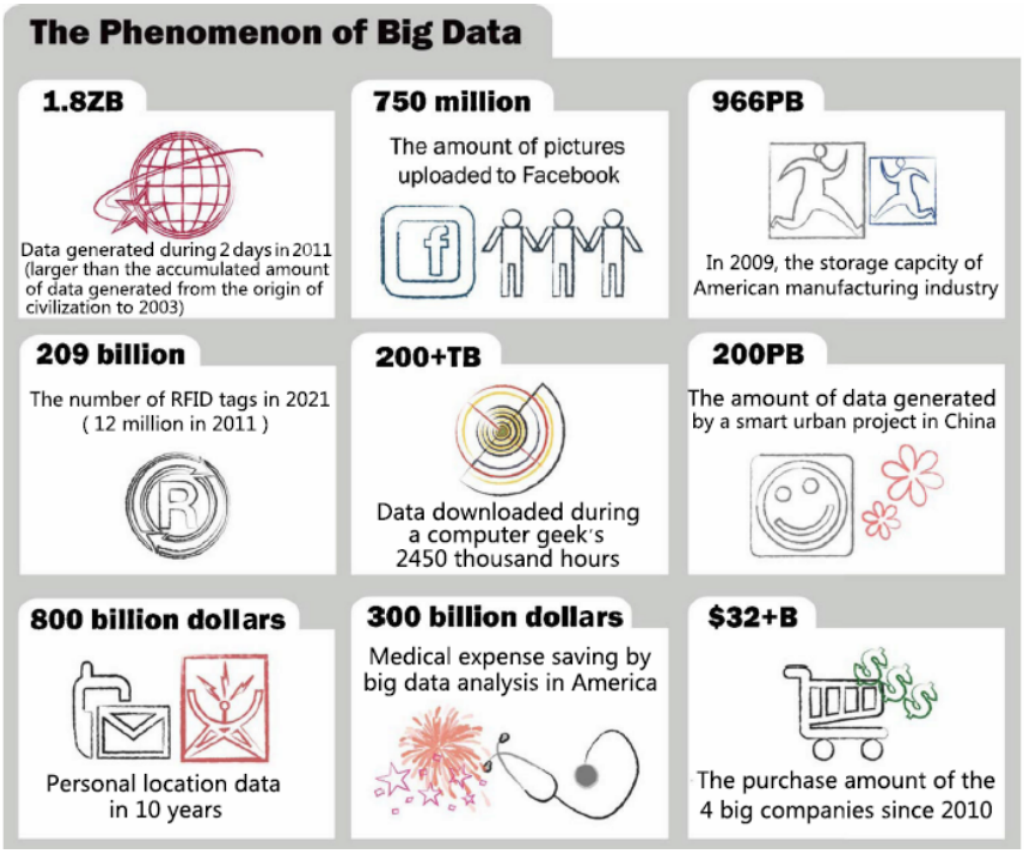

Figure 1.1: The volume of generated data from different devices and industries (taken from [4]).

for enlarged value of collected data sets for business ambitions. The big data market revenue is forecast to reach 61 billion in U.S. dollars by 2021 [7], while the worldwide expenses for big data and business analytics are forecast to be 215.7 billion in U.S. dollars by 2021, 10.1 percent more than in 2020 [8].

In the past 7 years, the structure of decision-making has changed from decisions based on experience to based on data (Fig. 1.2). The Figure shows that companies have placed more emphasis on data related decision-making over the last 7 years due to the increased quantity and quality of accessible data.

Further, it is increasingly popular to store big data on decentral platforms, for example, a decentralized application (dApp) on a blockchain [10]. This due to the increased popularity and development of the blockchain technology over the last 2 years. With the introduction of decentralized finance (DeFi) and advanced decentralized infrastructure and systems, this technol-

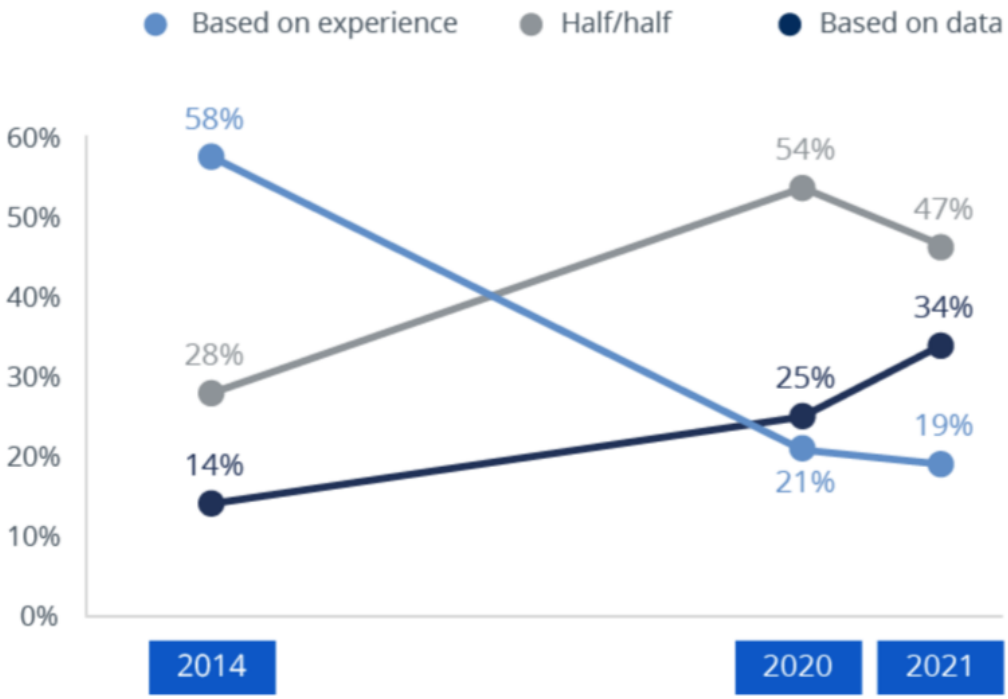

Figure 1.2: The way of decision-making changed through the years 2014 - 2021. More and more companies are seeking to benefit from the big data market (taken from [9]).

ogy gained large numbers of users and acquired appropriate use cases in the financial and online service sectors. Especially, the concepts "Proof of delivery", "Proof of stake" and "Proof of work" and the lack of need for middlemen provide a more efficient framework than centralized systems. Later, differences and advantages will be discussed in chapter 3. Another reason why more companies use blockchain is because of its Peer-to-peer (P2P) network and its distributed ledger technology (DLT). The P2P network is completely secured by a cryptographic algorithm that prevents unauthorized access and control to a node of the ledger. Interactions between users, applications, and blockchain are logged on the blockchain transparency and securely.

1.2 Objectives

The majority of data repositories with integration and exchange service are missing important features, for example, knowledge externalization, incentive model for the platform users, monitoring the use of data sets. Besides, central storage spaces are really expensive [11] in relation to increasing data set sizes. These factors are fundamentals to the success of a marketplace for data monetization [12].

The objectives of this thesis are the development of a conceptual architecture of the data repository envisioned accompanied by a proof of concept (PoC), implementation, and evaluation. This information repository should manage information about:

- the access of data (availability),
- the use of data by services (provenance),
- processing services used for upscaling of data sets (life-cycle).

To understand data sets and the usage of data sets more, it is recommended to monitor the usage of data sets. Forrester Research published in 2019 a report which shows that "between 60% and 73% of all data within an enterprise goes unused for analytics"[13]. Data Monitoring includes real-time monitoring, tracking and reporting of access and usage of data sets that generates audit log of access, usage and polishing. This helps data set owners to find out how their data is being used by other users. When a data set is polished by another user, the original data set owner can monitor it and gain incentives. Besides, security is another important point for monitoring data sets. Company can monitor how, where, when the data set is accessed.

First, we will introduce related works and analyze their repositories. Then, the conception model and solution architecture are described and presented in chapter 3. Furthermore, a list with the implementation requirements can be found in chapter 3.2. The implemented data repository is introduced in chapter 4.1. This platform is then analyzed and evaluated to elaborate enhancement measures. In the evaluations, the transaction costs, the user experience, the eco-system's security and the use case are considered and discussed. Finally, a summary and the further works are determined in chapter 7.

The implementation will be written in ReactJS and uses Moralis's Web3 software development kit (SDK) and application programming interface (API). The InterPlanetary File System (IPFS) is the data storage system (DSS) for

the uploaded data sets and metadata files. Furthermore, the Ethereum token standards Ethereum Request for Comments (ERC)-721, also known as a non-fungible token (NFT), and ERC-1155, a multi-token standard, will provide access control and proof-of-delivery for our eco-system. In the end, the implemented eco-system for distributed management of data and monitoring of use should be evaluated if this kind of electronic marketplace on DLT has any added value to the big data infrastructure and if there are specific use cases that will profit from this type of marketplace.

1.3 Use Cases

Mainly, our eco-system grants data users and data providers the ability to exchange and integrate data sets, view detailed usage tracking, and generate revenue streams with the incentive model. For example, companies in the mining industry are recently starting to collect data from their embedded sensors in heavy mining vehicles and machinery [14]. The operation costs of these machines, as well as the maintenance costs are really expensive [15]. Additionally, the excessive need for resources to run them is another reason for the high operation costs. In particular, companies can benefit from big data analytics and optimization of the outdated processes which lead to high operation and service costs.

Further, optimization and data analysis companies are important parts of this data repository eco-system. An incentive model as a reward for providing, polishing, and reviewing data sets, can motivate members in the mining industry to maximize their potential and generate additional revenue streams. In this use-case scenario, mining companies provide raw data sets collected from their embedded sensors on mining machines. Instead of spending huge amounts of money to process these raw data themselves, they can upload this data set to a marketplace and list them for sale. Data analytic and processing can acquire raw data sets from the marketplace to process them into utilizable data. These processed data sets can be provided as a new data set derived from the raw data set. Not only operation expenses could be minimized, but companies could also benefit from the optimization of internal processes, for example, supply-chain management, real-time data management, remote diagnostic and troubleshooting and event analysis, condition and safety monitoring [16]. Besides, the distributed management and storage of data creates the industry a platform to store and utilize their generated data.

In consideration, a repository for data monetization provides great value

for branches with high maintenance costs and outdated processes. With our data repository platform, companies can utilize their collected industry data, and generate value from the performance and process insights. Monitoring the use of data sets, getting incentives for providing and polishing data, and the distributed management of data, are great features for a data repository eco-system. We will evaluate these features in chapter 6.

2 | Related Work

2.1 Background

2.1.1 Blockchain

A blockchain, also called Distributed Ledger Technology (DLT), can store different kinds of information, data, and transaction records. Mainly it is used to make cryptocurrency transactions, ownership of NFTs, and in general, any digital asset immutable and transparent by using hash functions. These digital ledger functions and the ability to push smart contract code to the blockchain and execute them make a blockchain a "record-keeping and contract-enforcement technology" [17]. A blockchain block is similar to a database, where transactions are recorded within it whenever a new one comes through the blockchain. These blocks need to be validated by the network, and if the validation is complete, the block will close and be chained to the previous block, known as the blockchain. Not only transactions and payments can be stored in the blockchain. Furthermore, it can be used to store data points immutably, such as supply chains, voting governance, personification IDs, and more. In our thesis, the blockchain is used to interact with smart contracts and for an immutable tracking and tracing of interactions with our platform. The immutability of the blockchain network prevents the manipulation of data. Once they are on the blockchain, the data can not be manipulated.

2.1.2 Distributed Data Store

Distributed data stores denote distributed databases in which users store information across many nodes. These nodes are in separate physical locations over a single or different network. The data or information are split among several physical servers. Instead of storing data and maintaining it at one place in centralized data storage, distributed data storage systems refer to storing data in multiple databases in separate physical locations. Distributed

data stores mostly have high scalability and provides high performance for low operation costs. Additionally, data storage providers can be rewarded for their storage provided to the distributed database. With an incentive system, a distributed data store can be built with lower operation and maintenance costs than a central storage.

2.2 Related Systems and Approaches

2.2.1 Chainlink

Chainlink provides a Decentralized blockchain Oracle Network (DON) on the Ethereum blockchain to connect off-chain data feeds with blockchain-based smart contracts. Due to limited storage of arbitrary data in transactions and high costs when storing data on the blockchain, it is recommended to store data on off-chain solutions, e.g., IPFS, International Data Space (IDS), Filecoin, or any decentralized Peer-to-peer (P2P) storage solutions. On the other side, blockchains cannot interact with external systems and run API calls. The Chainlink decentralized oracle network claims to be able to connect "[...] contract directly to real-world data, events, payments and other inputs" [18]. Every Chainlink DON contains at least seven nodes with API credentials, which provides high-quality data, high responses, and a quality guarantee. Besides that, Chainlink also offers on-chain service agreements to define data delivery and quality parameters and connects the quality with the payment. Chainlink uses several data sources, or oracles, to decentralize data fed to smart contracts so malicious users cannot provide incorrect information to manipulate the blockchain. Rather than having to trust one source blindly, smart contracts running on a network can have unfettered access to resources such as data feeds, payments from a traditional bank account, and Web APIs. Intending to create a reliable and secure DON, Chainlink removes counterparty risks and reduces dependencies on third parties. Data providers can sell their real-time data sets through their APIs as feeds to the Chainlink's marketplace, which provides data feeds, nodes, adapters, and jobs. Chainlink is also known for its price feeds which connect smart contracts to the real-world market prices of assets and is used by big companies like crypto.com, Binance, Huobi, and more.

2.2.2 Filecoin

Filecoin is a blockchain-based decentral storage system built on IPFS with "built-in economic incentives to ensure files are stored reliably over time" [19]. It is designed for users with free storage space on their computers to rent out for their prices. As a reward, they receive the Filecoin token from storage users, who pay with this token. By allowing anybody around the world to join the network, it could build an enormous data store. Instead of creating new storage systems, Filecoin is designed to leverage the global untapped storage capacity to form a highly effective storage marketplace where users can pay for storage at low costs. In the Filecoin blockchain network, retrieval miners nodes compete to deliver data to clients as fast as possible. There are storage miners, which are in charge of storing data in the network, and retrieval miners, responsible for restoring the files. In comparison to centralized storage, e.g. Dropbox and Google Drive, there is no single point of failure and no single point of attack.

2.2.3 Ocean Protocol

Ocean Protocol is an open-source, decentralized data exchange protocol to unlock data for Artificial Intelligence (AI) [20]. It provides a multi-chain marketplace, where data sets are tokenized and can be traded across various networks. This data monetization eco-system allows users to monetize their data sets as ERC-20-based data tokens. In their eco-system, unique data assets are represented by ERC-721 tokens and access to data services are permitted by ERC-20 tokens. Recently, Ocean Protocol and the farmer platform Dimitra hosted an agricultural data competition, where participants find ways to utilize the available data sets to derive insights or find correlations between them. Additionally, the participants provides in the second phase of this competition, an algorithm that can be used live on the Ocean Protocol Market with the compute-to-data feature[1].

2.2.4 Other related works

The authors in [21] proposed a decentralized blockchain model to build a data repository system with data encryption, reputation system, access control, incentives, review validation, and dispute handling. There, they divide into four different roles to manage their system. First, the owner is the person or organization that owns the data and provides it to other users. Then the customer is the data set buyer, receives their bought data set from the

[1] https://oceanprotocol.com/technology/compute-to-data

IPFS nodes, and reviews the data on the review system. For encryption and decryption purposes, workers must provide these services to customers and owners. They also authenticate new customers and query smart contracts for requested data. An arbitrator is a trusted role who is involved in dispute handling. For data sharing, they upload the data meta information and the file to IPFS. The owner receives the IPFS and searches for worker nodes, which decrypts the data file later for customers. Their model uses a state-space search algorithm to split the IPFS hash into n shares and store them encrypted on the blockchain. This process can be seen in Fig. 2.1.

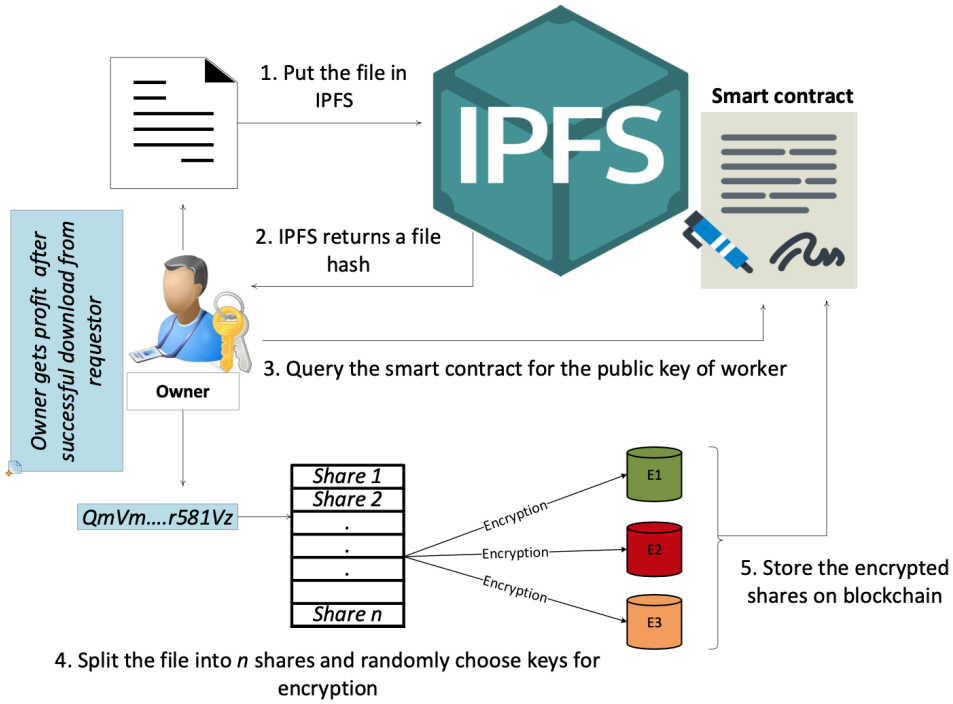

Figure 2.1: The IPFS file hash is split into n shares and encrypted. These encrypted shares are stored on the blockchain with the worker's public key. (Adapted from [21]).

In their model, a customer can retrieve a file by submitting a request to all worker nodes for a specific file and making a deposit for the requested data file. The worker nodes query the smart contract for the encrypted file shares and decrypt them. If all conditions succeed, the customer will receive the decrypted data shares and can subscribe to the owner's services. In [21], the authors provide in chapter 5.2.1 a data review system for uploaded data files to improve reliability, integrity, and quality control provided by customers.

2.3 Data Monetization Solutions

Data Monetization solutions can be very valuable for companies who want to generate additional revenue and use their collected data in useful ways. According to a survey conducted by McKinsey & Co, data monetization has already made its way into the most successful companies [22]. Respondents from the top-performing companies [23] reported that they are more motivated to monetize their data and already offer them in multiple ways. They are also more likely to work with companies in the same industries to build shared data pools. Another important point is that by creating shared data pools in the same industries, these data are used in a greater variety of ways that generate value for customers and their businesses. Based on customers' actual behavior and trends, companies can use this kind of shared data to provide new products and services [24]. These shared data sets could be a product, service, or insights. With data monetization, new areas of revenue or service offerings can be discovered and benefited by using data.

2.4 Summary

The introduced related systems provides a deep insight into distributed systems that operates with the blockchain as DLT. There are a number of approaches to create a eco-system for distributed data management and data monetization. But the monitoring of use of data is still missing in nearly all of these systems. The eco-system from Ocean Protocol offers an open-sourced marketplace that can be forked. In [25], an Ocean Protocol author describes how to fork their market and set up an own data market. Most blockchain apps are open-sourced and it is recommended to fork existing systems, instead starting right from the start. Table 2.1 shows the related systems Chainlink, Filecoin, Ocean Protocol and the eco-system from the authors from [21] in relation to distributed management of data, incentive model and monitoring of use features.

Systems	Chainlink	Filecoin	Ocean Protocol	Muqaddas Naz's [21]
Distributed Data	Yes	Yes	Yes	Yes
Incentive Model	Node Operators	Storage Miners	Yes	Review & Data Owner
Monitoring of Use	No	No	No	No

Table 2.1: Related work in view on distributed management of data, incentive model and monitoring of use.

3 Conception Model & Solution Architecture

First, it is useful to define which objectives our repository for data monetization should provide. In chapter 3.2, we will list the implement requirements for our data repository. We already describe in chapter 1.2 the objectives of this thesis and the approach to reach our goals. The next section defines which components will be in consideration for the implementation.

3.1 Components

3.1.1 React App

To provide a user interface, we create a ReactJS App with Web3 integration to let the user connect their wallet with the web interface. The App should provide an overview of the existing data sets, the payment conditions, a list of bought data sets, a function to upload data sets, and the usage policies. When buying an existing data set, the Data User needs to connect their crypto wallet with the Web3 integration through a "Connect Wallet" button on the App. Then the Data User can browse shared data sets through the marketplace tab. The marketplace tab should provide an overview of various data set groups and the data sets related to these data groups. By clicking on a specific data set group, the user can see which data sets are available for purchase and also explore and monitor the transaction history of each specific data sets on a block explorer and analytics platform like Etherscan. A window with the data set's price will be shown by clicking on the cart button, and if the user clicks on the "Buy" button, a smart contract function will execute from the Data User's crypto wallet. Here the user needs to confirm the transaction through the meta mask popup or the wallet provider of their choice. After confirmation, the Data User will receive the belonging ERC-1155 token. With this specific token in his wallet, the Data User can

access this purchased data set through the "My DataSets" tab.
To sell a data set, the Data Provider uploads the data set on the "Provide DataSets" tab and fills out the belonging metadata for the provided data set, for example, title, description, data type, and price. The Data Provider receives after successfully providing an ERC-721 token marks him as a unique data provider of this data set.

3.1.2 Data Storage System

The next challenge is to find a suitable DSS for our data repository. On the Ethereum blockchain, smart contracts can store data in the input data field in a transaction. At the current gas limit (30 000 000 gwei), we need to subtract 21000 gwei for the transaction fee and divide the result by 16 to calculate the non-zero bytes limit. (30000000-21000)/16 = 1.873.687 bytes. With the current gas price at 23 gwei and Ethereum price at $1000, we must pay 30000000 × 23 = 690.000.000 gwei = 0,69 Ethereum = $726 for every transaction with 1.873.687 non-zero bytes, which is not a relatable solution.

$G_{\text{txdatazero}}$	4	Paid for every zero byte of data or code for a transaction.
$G_{\text{txdatanonzero}}$	16	Paid for every non-zero byte of data or code for a transaction.
$G_{\text{transaction}}$	21000	Paid for every transaction.

Figure 3.1: Transaction costs on the Ethereum blockchain are related to the data size of data or code for a transaction (adapted from Ethereum yellow paper [26]).

Thus, off-chain data storage systems seem to be a better option. Here, we must distinguish between decentralized and centralized data storage systems. Centralized DSSs always have the problem with Single Point of Failure (SPoF), where the whole system will fail if a single part fails. In order to avoid failures that can occur in a decentralized infrastructure, it is recommended to operate with a decentralized DSS. Here we have the IPFS and the International Data Spaces (IDS). The IDS eco-system describes itself as a "[...] de facto market standard for trading and exchanging all kinds of data sets" [27] with more than 100 members in the International Data Space Association, like Google, SAP, and Audi. Here, the Data Providers must establish a usage policy for their data set. The IDS are decentral organized, where no central middlemen manage data and control tasks. Instead, it uses an IDS Connector to combine all endpoints to the data storage system of the International Data Space. IPFS is a P2P network with over 600 nodes around the world. Here we decide to work with IPFS as our data storage system.

3.1.3 Smart Contracts

We need to create smart contracts for logging and on-chain operations, specifically for the ownership token (ERC-721) and the data token (ERC-1155). Separate smart contracts with their token characteristic functions need to be created. Additionally, the decentralized electronic marketplace requires a smart contract with functions to operate.

ERC721 Contract

The ERC-721 token, also called NFT, works in our eco-system as the ownership token for the data set provider. For each provided data set, an ownership token is created with the smart contract and sent to the provider's crypto wallet.

ERC1155 Contract

The ERC-1155 token represents the data sets in our marketplace and operates as access control for data retrieval. These data set tokens can be purchased in the "Marketplace" section on the React App and traded between this token's owner and other users. We also want to track previous owners and the original data set provider for incentive purposes. Another big point is to include metadata from the data set in the tokens.

Marketplace Contract

To make our electronic marketplace work decentralized, we need to implement a smart contract for the marketplace operations. Besides the smart contracts for the ownership and data token already got functions like safe transfer, but we still need to implement a smart contract, where the data token for sale are collected on the marketplace smart contract address and are sent directly from the contract address to the buyer's crypto wallet. Also, the marketplace smart contract should provide the ability to list data set tokens independently on the marketplace. An event should be emitted whenever a market item has been created.

3.1.4 Data Retrieval

An user can only access the bought data set when the dedicated ERC-1155 token is in possession of the connected wallet. With the ERC-1155 token, the smart contract can relocate the IPFS hash and display it correctly. Here, the IPFS hash of the data set content is stored in the ERC-1155 token metadata

and is transmitted through a smart contract read function, which will be introduced in chapter 5.2.

3.1.5 Incentives

Data Provider

We divide here between incentives for Data Providers and Data Users. Data Providers should get an x % provision for every data set sold that they provided. This provision system can be integrated into the buy function, where x % from the purchased price goes directly into the Data Provider's crypto wallet. It also can be implemented in the ERC-1155 token, where the Data Provider's wallet address is saved as "Token Creator" and gets x % every time the token is sold to another Data User. Data Users can help other Data Users to review and examine if the data is useful. With a dedicated review system, Data Users receive incentives for leaving an honest and real review for the bought data set. The system checks the review and transacts an x % provision back to the buyer's wallet. With this review system, the data repository system can scale the usability of each provided data set.

Data Processor

The marketplace in our model contains raw data sets and processed data sets. Every raw data set needs to be analyzed and processed into usable data before profiting from it. Data set users can upload their processed data set as a processed data set y from the raw data set z and earn x % incentives for each purchase. The original data set provider will get a small incentive too. This fair incentive model motivates data users to upload their processed and used data sets.

3.2 Implementation Goals

With the components above, we can set up the implementation goals with for our data repository eco-system:

- **Must**
  - **Electronic Marketplace:** The marketplace should display data sets from different collections.
  - **Buy Data Sets:** Users should be able to buy data sets of their choice from the marketplace.
  - **Provide Data Sets:** Users should be able to provide any kind of data sets to the marketplace and lists them for a custom price.
  - **Access and Retrieve data sets:** Users should be able to access and retrieve their bought data sets.
  - **Incentives:** Data Providers must be compensated for using their data.
  - **Monitoring of Use:** The usage of each provided data set should be logged through smart contracts.
  - **Usage of a blockchain as DLT:** A blockchain should be used as DLT to log usages, handle the data provision and control the access permissions of a storage service, for instance, databases and P2P storage systems, to maintain the consistency of interactions between data providers and users.
  - **Usage of a distributed data store:** A distributed data store, for example, IPFS or IDS, should be used.
- **Should**
  - **User-friendly interface:** The marketplace should be easy to use and should provide a clear overview.
- **Could**
  - **Encryption:** The uploaded IPFS file hash could be stored encrypted in the data set tokens
  - **Secure Data Stream:** The retrieval of a data set's token could be performed through a secured data stream.

4 Implementation Tools

4.1 Platform Selection

To design our data repository eco-system within the constraints given in chapter 3, we must identify a suitable technology setup. The complete ReactJS source and smart contract code are Open Source and provided on RWTH GitLab[1].

4.1.1 Testnet

A publicly accessible Ethereum testnet blockchain is an Ethereum developers-managed blockchain to provide the Ethereum developer community a platform to test their smart contracts [28]. The most popular Ethereum testnets are Ropsten, Rinkeby and Kovan. We will connect our dApp to the Kovan testnet to test our smart contracts. This allows us to monitor and analyze the executed transactions on the Kovan testnet blockchain explorer.

4.1.2 Remix IDE

Remix IDE is a web-based, open-source Integrated Development Environment (IDE) and Code Editor provided by the Ethereum Foundation to write and deploy Solidity smart contracts. Solidity is the programming language for implementing smart contracts on blockchain platforms. The developed code can be compiled with the Solidity compiler and then deployed on JavaScript-based Virtual Machines (VMs), local blockchain networks, and injected web3 providers like MetaMask. By deploying our platform's smart contracts to the Kovan testnet, the Remix IDE suits well for testing and deploying purposes.

[1] https://git.rwth-aachen.de/deryumin/data-repository.git

4.1.3 ReactJS

React is a free open-sourced JavaScript library developed for building user interfaces. It is widely used as the foundation for creating single-page websites and mobile applications. An advantage of React Native, which uses ReactJS as its JavaScript library, is that it encourages creating reusable User Interface (UI) components that represent data that changes dynamically. In React, we can import various numbers of JavaScript libraries as node modules and use these library UI kits, connectors, and helpers. These node modules are imported code collected in a single file or multiple files and folders that reach out to external APIs to fulfill their functionality. Therefore, we must install the specific node modules with a node package manager. In fact that a lot of decentralized applications which operate with the Ethereum blockchain are built with ReactJS. The ReactJS library offers various tools, connectors, and APIs for decentralized applications (dApps).

4.1.4 Node.js

The open-source and cross-platform JavaScript runtime environment Node.js grant developers the capability to run scripts on the server-side instead of in a web browser. It operates on the free and open-source Google Chrome V8 JavaScript engine and builds dynamic web pages before sending them to the web browser. Rather than using different programming languages for each server-side and client-side script, Node.js merge these individual languages into one universal. By executing JavaScript directly on the server, the development of JavaScript-heavy websites improves performance in content loading size and page speed.

4.1.5 IPFS

IPFS is a decentralized hypermedia delivery protocol that is content-aware and identity-aware. It loads uploaded files from thousands of nodes rather than from one centralized server. These nodes relay information or store them directly. If a file is uploaded to IPFS, the data will be shredded into partitions, encrypted and exchanged to multiple nodes in the P2P network. Compared to other DSS (3.1.2), the IPFS identifies files through their content address and not their location. Identification, content linking, and content discovery are realized by unique content addressing, acyclic graphs (DAGs) and distributed hash tables (DHTs). With these features, IPFS supports a resilient internet, anti-censorship and a faster web experience.

4.2 Technology Used

ReactJS and Node.js are used for developing our application. In addition to that, the implementation will profit from other advanced technology. These used technology will be introduced in the following subsection.

4.2.1 Yarn Package Manager

The Node.js package manager Yarn is used to manage the dependencies of JavaScript projects and automate package installation, configuration, updating, and removing packages. Similar to the default Node.js Node Package Manager (npm), yarn was invented in a collaborative effort between Facebook, Google, Ember.js, and Expo.dev to address the issues with consistency, safety, and performance in large codebases. Yarn profits in general from advanced features compared to npm, such as parallel installation, versioning, and zero-install features, allowing offline installing with low latency.

4.2.2 Moralis

Moralis is like firebase, which is an app development platform and provides a software development kit and real-time databases. But rather than using NoSQL databases, it builds a server over a certain blockchain network to manage all blockchain transactions, crypto wallets, tokens, and more [29]. With the Moralis SDK, developers can build high-performance dApps with authorization through Web3 and integrate on-chain data such as token balances, different tokens, transactions, and live events. The easy-to-use SDK grants the dApps communication with the Moralis servers, which are deployed on multiple blockchains. One of the most useful functions is the event listener for smart contract transactions. Whenever a predefined smart contract function is called, which emits an event, the developers can customize the Moralis server to intercept this event and synchronize the data from the transaction to their database.

4.2.3 Ethereum-boilerplate

The Ethereum-boilerplate is a dApp boilerplate developed by Moralis on a thin React-wrapper around Moralis (react-moralis) and Moralis SDK. A boilerplates is in our situation a code construct, which can be reused in a new context. It is an open-sourced project and licensed with MIT License. The boilerplate provides basic dApp functions such as Web3 wallet authentication and WalletConnect support, Decentralized Exchange (DEX), basic wal-

let balance and transactions function [30]. The developers also describe the boilerplate as "light-weight, fast and modular [...]" [31]. Users can connect the boilerplate with the Moralis Server API and benefit from the complete Moralis SDK library. Furthermore, the React UI library antd design[2], provides various number of UI elements to display, for example, data sets, data collections, input fields, buttons and more.

4.2.4 MetaMask

MetaMask is a web browser-based cryptocurrency wallet that allows users to create multiple wallet accounts and addresses on every Ethereum-compatible network like Polygon[3], Fantom[4], and Binance Smart Chain[5]. It is a web browser extension for Brave, Chrome, and Firefox and helps users connect securely with dApps. MetaMask also provides a user-friendly interface for trading digital assets in Ethereum-based networks.

[2]https://ant.design/
[3]https://polygon.technology/
[4]https://fantom.foundation/
[5]https://www.bnbchain.org/en

5 System Overview

The eco-system for distributed data management is implemented on the Ethereum-boilerplate, and the ReactJS and Moralis infrastructure are applied. Because of the good compatibility with the necessary node modules, it is recommended to work with the from Moralis provided Ethereum-boilerplate. Depending on the node module versions, several issues could occur if the versions are incompatible. The system is written in ReactJS with Microsoft Visual Studio Code [32] and the scripts are run with yarn. Moralis SDK and Servers are used for operations on the blockchain and database management.

First, the boilerplate is installed by cloning the original project on the Moralis Github[1] and install the dependencies. Then, the Moralis dApp Server is created and configured on the Moralis website[2]. The environment is set to the Kovan testnet and the server region in Frankfurt. In the ".env" file, the Application ID and Server URL are filled in with the generated Moralis Server. Further, a crypto wallet is needed to interact with smart contracts. Here MetaMask is used as a wallet and gateway to blockchain dApps.

5.1 Marketplace

The marketplace is the landing page where all provided data sets are listed. The user can browse between different data set groups, which are displayed as collections. Fig. 5.1 shows the marketplace landing page with three data set groups.

By clicking on a collection, the associated data sets of this collection are shown with the information if the data set is available for sale. The data sets are displayed as cards containing the data set title and type. The cards provide two buttons to browse the blockchain explorer information of this data set token and buy the data set for a specific price. Users pay with testnet

[1] https://github.com/ethereum-boilerplate/ethereum-boilerplate
[2] https://admin.moralis.io/dapps

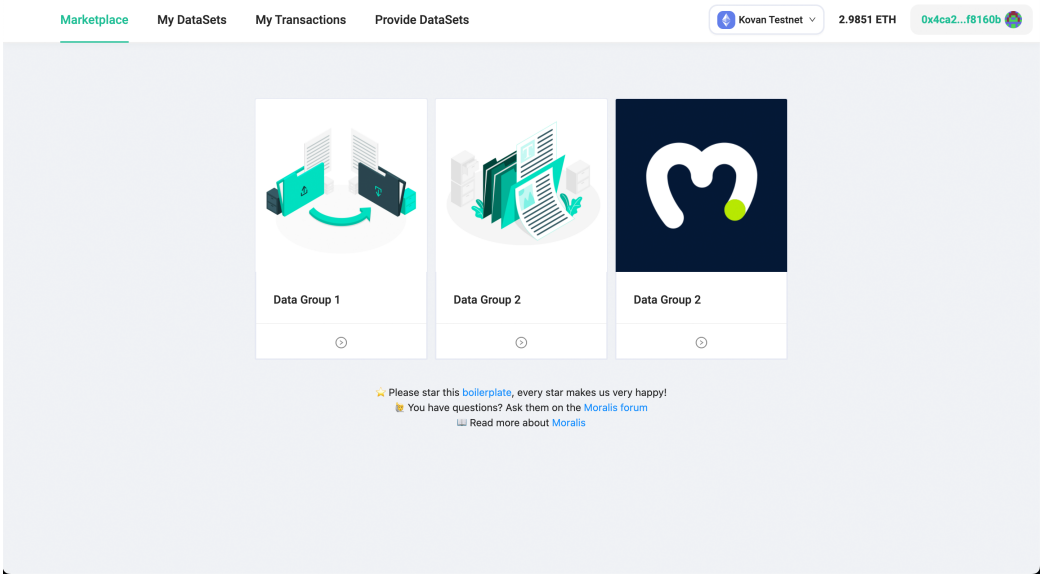

Figure 5.1: The "Marketplace" page shows various displayed data set groups. Users can connect their MetaMask wallet on the top right, and the blockchain can be switched dynamically between operable chains.

Ether from their wallets for a price defined by the data set provider. By clicking on "Buy", the MetaMask wallet opens a confirm transaction window. And if the purchase successes, the bought data set will be transferred as a data set token to the buyer's crypto wallet. The UI is shown in Fig. 5.2.

Besides the user-friendly front end interface, the marketplace should run decentralized and on the blockchain. These preconditions are reached with a marketplace smart contract that collects every on-sale labeled data set token and provides them to the buyer independently. Instead of receiving the data token from the data provider's wallet address, the data token comes directly from the smart contract address when the defined payment is obtained correctly. Additionally, the smart contract should provide a read function and a list of the current on-sale items.

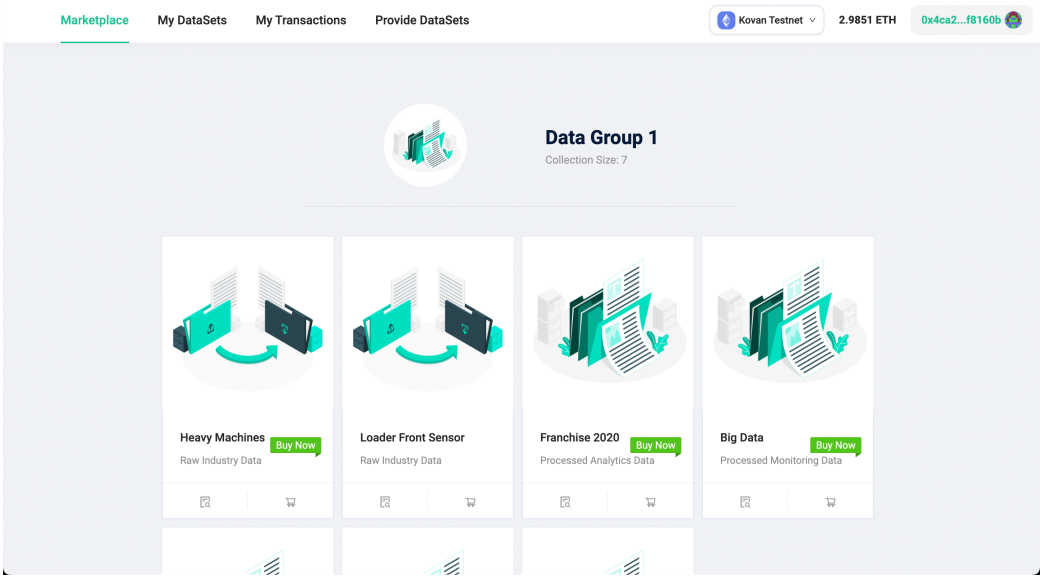

Figure 5.2: On the collection page from a data set group, the data sets are listed with the functions to see on-chain data via blockchain explorer and purchase option.

Smart Contract

The requirements and functionality should be defined to develop the marketplace smart contract:

- **Fetch Market Items:** In fact that the data token from every data set listed for purchase is stored on the marketplace smart contract. A list with all data sets for purchase is useful to synchronize with the UI. `fetchMarketItems` is a read function, where the on-sale labeled data sets are returned as an array. We define here a market item as a struct with the attributes: every market item for sale receives a marketplace itemId, the data token ERC-1155 smart contract address, the unique tokenId within the data token collection, the seller and owner address, the defined retail price in Ethereum, the amounts of the data token for sale and if the data token is sold out. With this struct and a hash table, where the marketplace itemId is mapped to their market item struct element, an array with the actual data tokens for sale can be created and returned. Here, the smart contract verifies in the hash table if a tokenId's owner is equivalent to the smart contract's address. These market item struct elements are collected in an array and returned.

- **Create Market Item:** The `createMarketItem` function is responsible to handle data set listings. When a user lists a data set, this function creates a market item struct element, mapping it to the market item hash table, and transfers the data tokens to the smart contract address. Also, an *MarketItemCreated* event is emitted with the element attributes, which is synchronized with the Moralis Server. Every time a data set is listed, a *MarketItemCreated* event will be emitted that will be intercepted by a configured event listener in our Moralis dApp setting and stored in the Moralis dApp database. Table 5.1 shows a insight into the Moralis database MarketItems table, where the *MarketItemCreated* events are stored.

- **Market Sale:** The listed data sets are sold with the `createMarketSale` function. At first, the function verifies if the data set is for sale and if the Ethereum value sent as *msg.value* by the buyer is equal to the defined price from the seller. If these conditions apply, the amount of Ethereum is transferred securely through the global `transfer` function to the seller's crypto wallet address. Then the data token is transferred with the imported IERC1155 `safeTransferFrom` function from the smart contract address to the buyer's wallet address. The amount attribute from the data token is then decreased, and if the amount is equal to zero, the data token is marked as sold. Additionally, an event is emitted with the itemId and the buyer's crypto wallet address.

block_hash	Reference number of the block processed the transaction.
transaction_hash	Transaction number of the listing.
itemId	Marketplace specific item number.
nftContract	Smart contract address from the data token collection.
tokenId	Unique token number within the data token collection.
seller	Wallet address from the data token seller.
owner	Wallet address from the data token owner.
price	Price for each data token.
amount	Available data tokens for sale.
sold	Indicator when the data token is sold out.
address	Marketplace smart contract address.
confirmed	Indicator when the transaction is confirmed by the blockchain.

Table 5.1: Moralis dApp database entries from the table MarketItems. The Moralis Servers intercept listing transactions and integrate the data above into the dApp database.

UI and Integration

The boilerplate "Marketplace" page shows different ERC-721, ERC-777, and ERC-1155 token collections. The displayed collections can be configured in the `/src/components/helpers/collections.js` file. For each blockchain, the displayed collections can be edited. The collection's smart contract address, name, and image are required. The small button with the circle and arrow to the right leads to the collection site from the data set smart contract address with the collection size and the tokens in the smart contract. The Moralis Server integration and the boilerplate hooks load the token from each specific data group directly from the blockchain. A user can view the blockchain information from the listed data sets and buy them if the listing is confirmed and the attribute *sold* is not true.

The shopping-cart icon opens an antd UI kit Modal with the data set image, price, and the "Buy" button. The "Buy" button is connected with the react-moralis `useWeb3ExecuteFunction` hook and executes the smart con-

tract `createMarketSale` function. Here the JavaScript function `purchase` create a transaction with the `createMarketSale` from the user's wallet. To interact with the smart contract, the `purchase` function needs to be set up with the needed parameters. Fig. 5.3 shows the `purchase` function code to call the smart contract function. In the part before the await function, the parameter for the `createMarketSale` smart contract function are defined. The smart contract function requires the data token contract address, the marketplace itemId of the data set to purchase, the amount, which is here 1, and further data that will be left blank. Also, the marketplace smart contract address, function name, contract Application Binary Interface (ABI) code, and the transaction message value, which is the data set price, is necessary to interact with the `createMarketSale` function. The code after the await function depends on the result of the transaction. A modal will open when the transaction is successful with the message "Success!" In the Moralis database, the amount of this data token will decrease by one. Nevertheless, a modal with the message "Error!" will be shown if the transaction fails.

```
const purchaseItemFunction = "createMarketSale";
async function purchase() {
    setLoading(true);
    const tokenDetails = getMarketItem(nftToBuy);
    const itemID = tokenDetails.itemId;
    const tokenPrice = tokenDetails.price;
    const amountleft = tokenDetails.amount;
    const ops = {
      contractAddress: marketAddress,
      functionName: purchaseItemFunction,
      abi: contractABIJson,
      params: {
        nftContract: nftToBuy.token_address,
        itemId: itemID,
        amount: 1,
        data: []
      },
      msgValue: tokenPrice,
    };
    await contractProcessor.fetch({
      params: ops,
      onSuccess: () => {
        setLoading(false);
        setVisibility(false);
        updateSoldMarketItem();
        succPurchase();
      },
      onError: (error) => {
        setLoading(false);
        failPurchase();
      },
    });
  }
```

Figure 5.3: The purchase function calls the marketplace smart contract function "createMarketSale" with help from the react-moralis "useWeb3ExecuteFunction" hook.

5.2 Balance

The "Balance" page provides an overview of the data sets belonging to the connected crypto wallet. All data sets are shown as cards with their title, data type, and the buttons to view on a blockchain explorer, list the data set token, and access the data set content. The UI can be seen in Fig. 5.4.

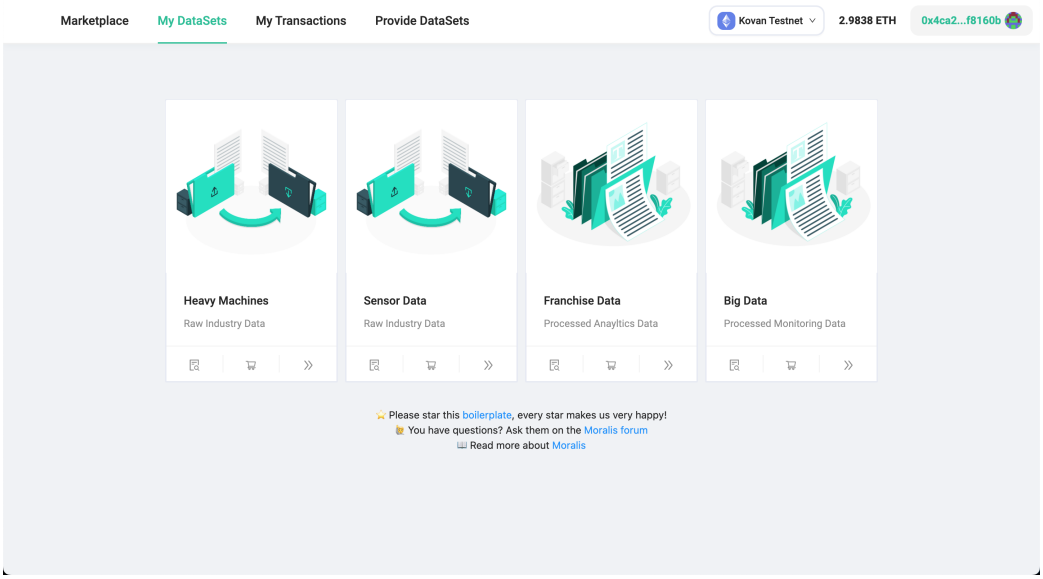

Figure 5.4: The "My DataSets" page lists all data sets that belong to the user's crypto wallet. These data sets can be listed, accessed, and analyzed on the blockchain explorer.

When a user lists a data set token, a price is defined in the Ethereum currency, and the token is sent to the marketplace contract address. The listing smart contract function can be seen in chapter 5.1 under `creatMarketItem`. Before an ERC-1155 can be sent to the marketplace smart contract address, the user has to approve the smart contract and permit them to transfer the token from their crypto wallet. This permission is granted by the ERC-1155 `setApprovalForAll` function. Fig. 5.5 shows the data token listing modal and Fig. 5.6 the data set accessing modal.

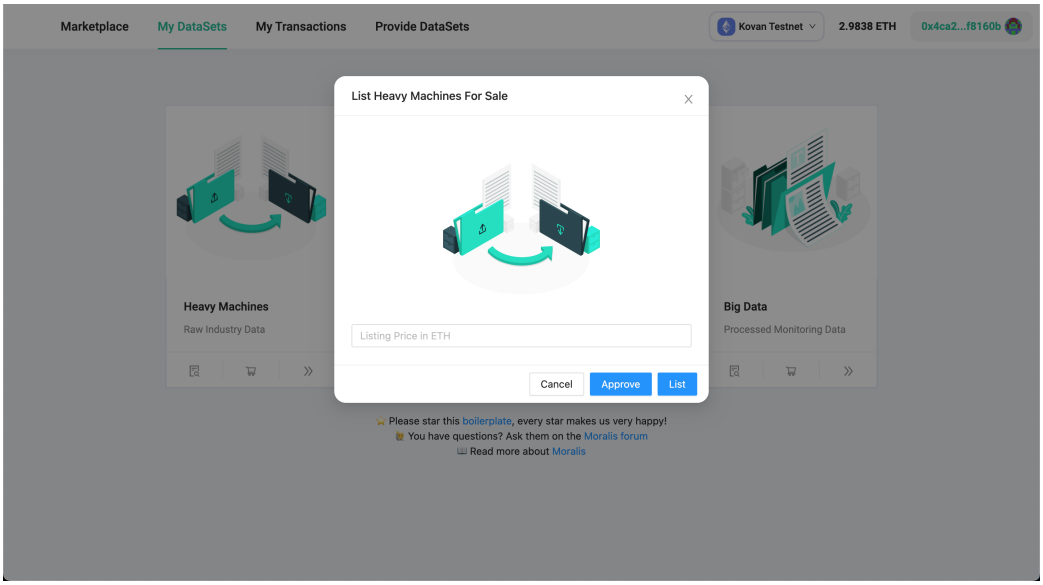

Figure 5.5: When clicking on the shopping-cart icon, a modal will pop up with a text field for the price of this listed data set token. Before listing the data set token, the user must click "Approve" to give the smart contract permission.

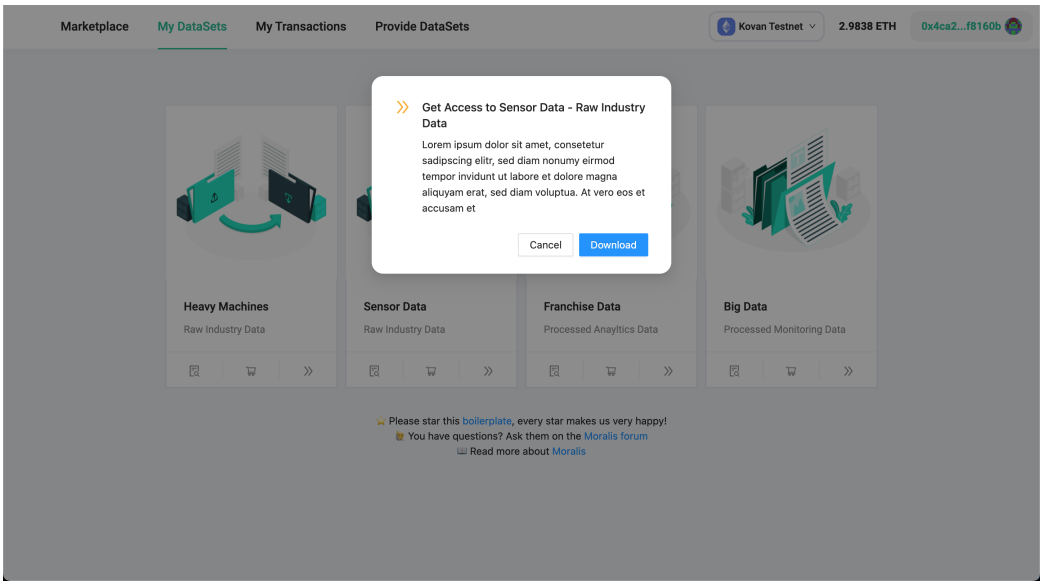

Figure 5.6: The contents of a data set can be accessed through the two right arrows. Here, a usage policy text can be displayed, and the user can access the data set content by clicking on the "Download" button.

Smart Contract

The data set token smart contract is created as an ERC-1155 token standard and extended with three functions to support listing and accessing data sets:

- **URI:** The `uri(uint256 tokenId)` function is a read function which returns the Uniform Resource Identifier (URI) from an ERC-1155 token. There, the data set metadata is stored as a `JSON` file. Fig. 5.7 shows an example of this metadata `JSON` file.
- **Set URI:** With the `setURI(string memory _uri, uint256 tokenId)` function, the URI of a data set token can be configured and updated by executing this function with the to be edited tokenId and the new URI.
- **Mint token:** With the `tokenMint(address _to, uint256 _tokenId, uint256 _amount, string memory _uriInput)` function, data set tokens can be created and configured with a token specific URI. A defined amount of the token with the tokenId _ *tokenId* and URI _ *uriInput* can be minted and sent to the wallet address _ *to*. Further, the previous and current owner, the block timestamp, the number of owners, and the address that minted the token for incentive purposes, are stored as a struct within the token.

```
{
    "title":"Heavy Machines",
    "description":"A short description",
    "datatype":"Raw Industry Data",
    "price":3,
    "filehash":"
       QmUgkz5sCZt39JMW5PWu28mTfwwTyd9WLScg978RMeENLG
       ",
    "fileimage":"
       QmRDBX9rmbcEAi8m5i7Wk3XtmPnUzTVmbycGWNPKbX35BN
       "
}
```

Figure 5.7: In the data set metadata JSON file, the essential information about the data set is stored. Including the data set title, a short description, data set type, price, file hash, and file image hash.

UI and Integration

The UI on the "Balance" page should be simple and sufficiently clear. With help of the Moralis Web3 API, the purchased data set tokens are loaded with the API call `getAllTokenIds` and the parameters *chain* and *address*. The loaded data set tokens are mapped with the Card module from the antd UI kit and displayed on the site. Each Card module represents a purchased data set with image, title, and data type from their metadata. Additionally, there are buttons beneath the metadata to view token information on the blockchain explorer, sell data set tokens and access the content of a data set. The sell data sets button, represented by a shopping cart icon, interacts with the marketplace smart contract function `createMarketItem`. Clicking on the button puts the data set token for the seller's price on the marketplace. Similar to the `createMarketSale` function in Fig. 5.3, the `useWeb3ExecuteFunction` is used to create a transaction with the `createMarketItem` function. Therefore, the data set token contract address, data set token ID, the selling price, amount and a blank data field, are submitted as parameters for the `createMarketItem` function. Before listing a data set token, the seller needs to set approval for the listing procedure. The "Approve" button on the listing modal is connected with the `setApprovalForAll` from the data set token's ERC-1155 smart contract.

The two right-pointing arrows icon gives access to the access data set modal, where the usage conditions can be written and where the data set content can be downloaded. The "Download" button calls the `uri` function from the data set token smart contract, which returns the metadata JSON file. The returned JSON file is parsed, and the file hash is extracted. With the file hash, the data set content can be downloaded directly to the user's computer by clicking on the "Download" button.

5.3 Provide

A user can provide and sell their data set to other users through the "Provide DataSet" page. The provided data set is uploaded to IPFS, and the file hash is stored. Here we introduce the Ownership token, which marks the user as the data set provider. The user's crypto wallet must be connected to provide data sets. In the first step, the data set's title, description, data type, and selling price must be provided. Then the data set content is uploaded to IPFS. An ownership token with the ERC-721 standard is minted and sent to the provider's wallet address. Associated to the ownership token ID, the data set token with the same token ID is then minted with the provided

metadata from the data set. Related to the listing function, the provider has to approve the smart contract, following with the listing of the minted data set token to the selling price. Fig. 5.8 shows the UI where users can upload and sell their collected data sets. To achieve the data-providing process, a function to create JSON files, an IPFS uploader, and an ERC-721 smart contract for the ownership token need to be considered.

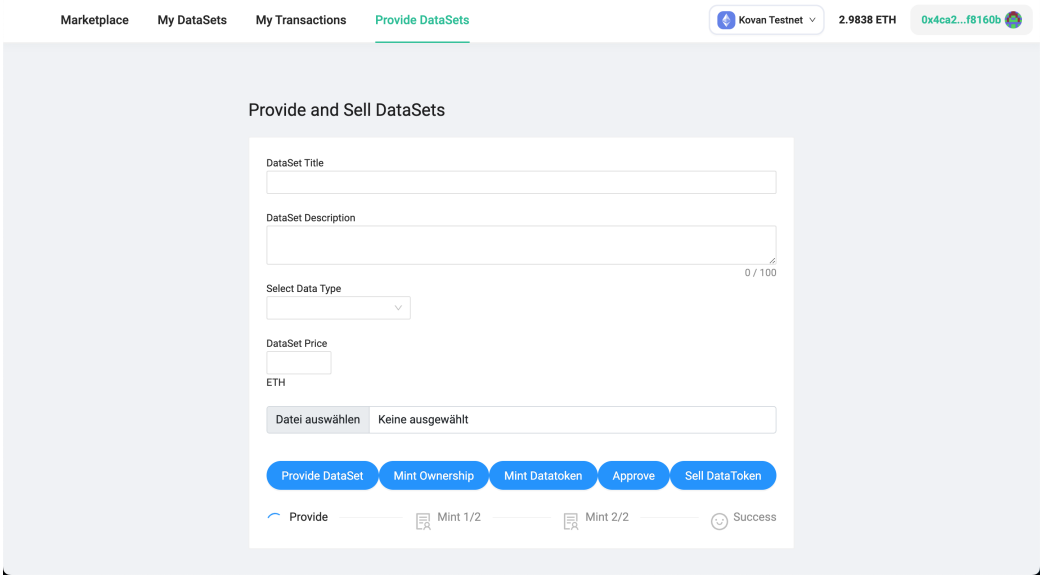

Figure 5.8: To provide a data set, a user needs to fill out the data set's metadata and upload the data set to IPFS. Then he needs to click through the buttons, which create a metadata JSON file, mint the ownership token, mint the data set token, permit approval, and finally, list the data set token on the marketplace.

Smart Contract

An ERC-721 smart contract needs to be created for the ownership token. This smart contract should provide the functionality to mint tokens and the function to read the latest token ID, which is the unique identifier for the tokens. Thereby, these functions are defined as:

- **Mint token:** The `safeMint(address to)` function mints a ERC-721 token with a unique token ID to the wallet address *to*. In the ERC-721 token standard, only one token exists for each token ID. In general, the token ID is ascending. A counter stores the latest minted token ID and the `safeMint` function increases the token ID by one and calls

the predefined ERC-721 `_safeMint(address to, uint256 tokenId)` function.

- **Get token ID:** For minting an associated data set token, the token ID of the ownership token needs to be identified and stored into a local variable. This token ID will be submitted in the mint process of the ERC-1155 data set token. The latest token ID is returned with the `itemId()` read function.

Further, the functions `tokenMint` and `setApprovalForAll` from the ERC-1155 smart contract and `createMarketItem` from the marketplace smart contract, are executed for minting and listing the data set token. Altogether, four transactions are needed to provide and list a data set for sale.

UI and Integration

The "Provide DataSets" page is implemented primarily with a Card module from the antd UI kit with input fields for the title, description, type, and price. The data set content will be uploaded with the *reac-ipfs-uploader* node module. Therefore, this node module must be installed with the yarn package manager. The IPFS URL to the uploaded data set content is stored in the *fileurl* variable. By clicking the "Provide DataSet" button, a `JSON` file from the data set metadata is created with the Moralis `File` function, which can be seen in Fig. 5.7. This `Moralis.File` includes key-value pairs of `JSON` compatible data. The *filehash* is extracted from the *fileurl*. Additionally, a file image is stored in the metadata file, depending on the data set type. Then the metadata `JSON` file is uploaded to IPFS and the file hash is stored in the *erc1155dataurl* variable. That proceeding can be seen in Fig. 5.9. The next step is to mint the belonging data set ownership token, an ERC-721 token with a unique token ID. This means that for each token ID, there is only one unique token. The data set ownership token is minted with its `safeMint` function and sent to the user's crypto wallet. For minting the data set token, which will be listed on the marketplace, the exact token ID from the ownership token needs to be gathered. With this token ID, the data set token is minted. Also, the number of minted tokens and metadata file hash has to be provided as `tokenMint` smart contract function parameters.

Furthermore, the user has to grant the smart contract permission to transfer these tokens by clicking on the "Approve" button, which calls the `setApprovalForAll` function. At last, the minted data set tokens are listed through the "Sell DataToken" button. This button executes the listing function, which we already saw on the "Balance" page.

```
const onSubmit = async(e) =>{
    const filehash="";
    const metadata = {
      title,
      description,
      datatype,
      price,
      filehash,
      fileimage
    };
    metadata.filehash =
    fileUrl.replace("https://ipfs.infura.io/ipfs/"
       , "");

    const file = new Moralis.File(metadata.title +
        "_provide.json", {
      base64: Buffer.from(JSON.stringify(metadata)
        ).toString("base64"),
    }
    );
    await file.saveIPFS();
    const metadataurl =
    file.ipfs().replace(
    "https://ipfs.moralis.io:2053/ipfs/","");
    erc1155dataurl = metadataurl;
    succSubmit();
}
```

Figure 5.9: The "onSubmit" function will be executed when the user clicks on the "Provide DataSet" Button. It creates a metadata JSON file with the value from the input fields and uploads it to IPFS.

When all four transactions are confirmed successfully, the provided data set is fully configured, and the data set tokens are listed on the marketplace. If a user buys this data token, the data set provider receives the revenue transferred directly from the buyer. Fig. 5.10 illustrates the providing process.

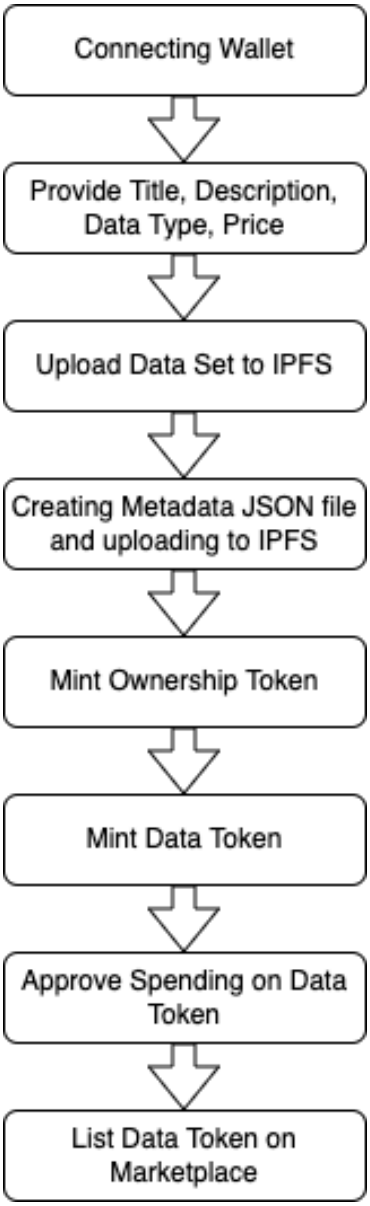

Figure 5.10: Flow diagram of the providing data set process.

5.4 Transactions

The transaction page is already pre-built in the Ethereum-boilerplate and contains the status of sales and recent interactions with the marketplace smart contract. In a table on the transaction page, the date, tokenId, data set group name, status, and price are displayed for each transaction with the marketplace. The JavaScript code pulls the transaction entries from the "MarketItems" table in the Moralis dApp database and lists them as rows. The transaction page can be seen in Fig. 5.11.

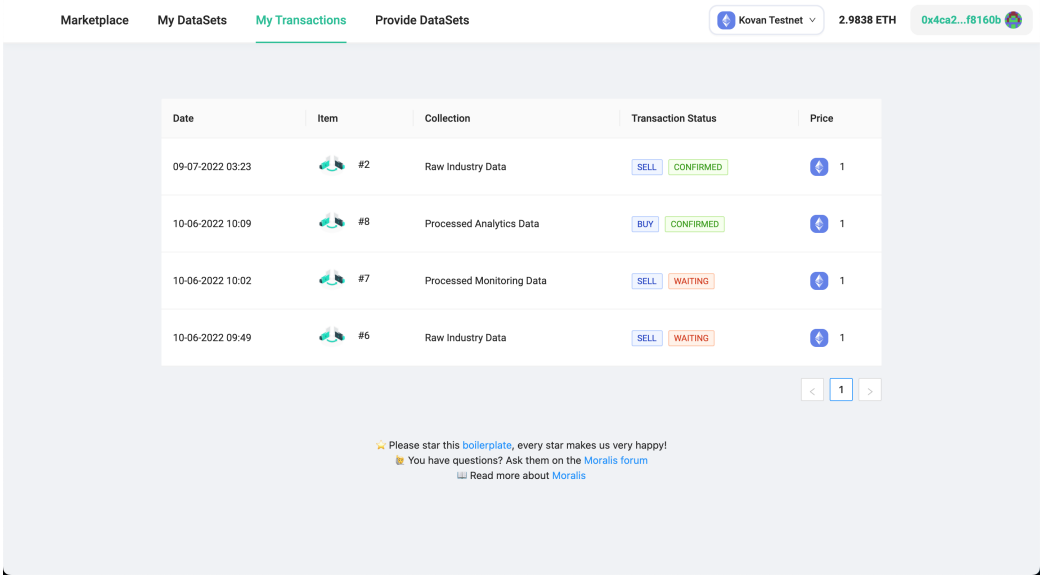

Figure 5.11: On the transaction page, the successful and pending sales are documented. The transaction status shows "Confirmed" when a listing is successfully sold.

5.5 Deployment

Before the react-moralis dApp can be deployed and tested, it is necessary to deploy the three smart contracts with Remix IDE. The smart contracts are first compiled with the correct Solidity compiler version, then deployed and run on the Kovan testnet with injected web3 as the deployment environment, which is the MetaMask wallet extension. The `marketplace`, `ownershiptoken` and `datasettoken` smart contracts are deployed with this procedure. The contract creation transaction is executed here. Each smart

contract address is noted and inserted into the `useWeb3ExecuteFunction` on the marketplace, balance and "Provide DataSet" page. Despite that, the contract ABI code (5.1) also is injected, which can be found after compiling successfully on the Solidity Compiler page in Remix. Only the `datasettoken` has a constructor and has to be configured after deployment. Thus, the admin wallet address, token name, four-digit token symbol, tx cost, token-level metadata, and contract-level metadata needs to be filled out. After deploying all three smart contracts and integrating them successfully in our dApp, the dApp can be started with the "yarn start" command. Therefore, the dApp is now running completely decentral on the Kovan testnet.

6 Evaluation

The data monetization repository is evaluated and analyzed to elaborate points of improvement and evaluate the performance of the application as a distributed management for data and monitoring of use. The gas usage, blockchain performance, and security were evaluated.

6.1 Gas Usage by Transactions

Gas usage is important for keeping transaction costs low. Smart contracts with higher complexity need increased block size on the blockchain, which results in higher gas usage and costs. The full Ethereum transaction cost report can be seen on the Ethereum Yellow Paper [26]. Depending on the data size or the number of "non-zero bytes" sent through a transaction, gas usage can increase. Similar to the bitcoin miner fees [33], a gas limit is set mostly by the user's crypto wallet, and the gas price is declared by the user regarding the execution time of this transaction. Slow transaction completion times are a result of the mining fee that must be paid to the blockchain network. The more gas price a user is willing to pay, the faster the transaction gets confirmed. Each smart contract function in order to operate our data monetization repository was analyzed. In the next section, the gas usage of the different functions is considered with gas fees from the Ethereum Mainnet (23 Gwei). With the Mainnet gas fees, we can compare our data monetization repository to the live environment.

6.1.1 Contract Creation

To operate our data monetization repository, it is necessary to deploy three smart contracts (marketplace, ownership token, data set token), introduced in chapter 3.1.3. By assessing the complexity of the three smart contracts before looking at the gas usage, we claim that the ownership token is less complex and the data set token more complex.

Marketplace Contract:

The marketplace smart contract imports the `/ERC1155/ERC1155.sol` and `/ERC1155/IERC1155Receiver.sol` functions from the ERC-1155 token standard. Furthermore, the util `Counter.sol` for counting the token IDs and the security modifier `ReentrancyGuard.sol` prevents functions to make external smart contract calls. This could be a recursive call to the original function with bad intentions. These libraries are directly imported from the "@openzeppelin" GitHub[1]. This smart contract contains a struct, two events and three complex functions. These functions includes comparisons, loops, creating variables and token transfers Fig. 6.1 shows the average gas usage for deploying the marketplace smart contract.

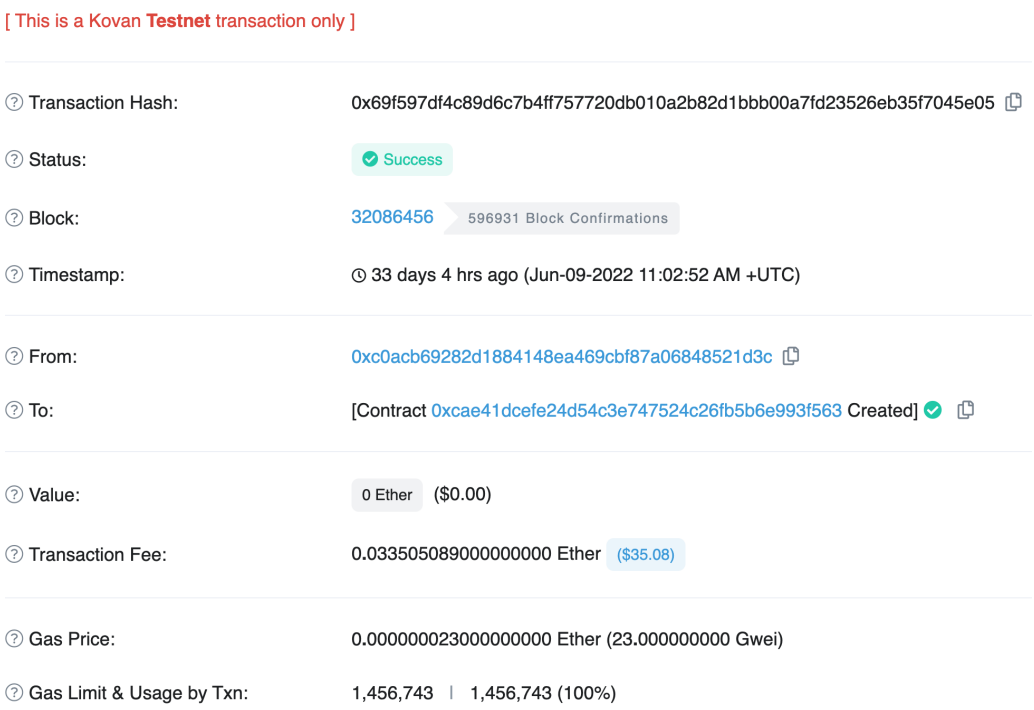

Figure 6.1: The marketplace smart contract uses the efficient Counters. Also the ERC-1155 smart contract imports dont use much gas.

[1]https://github.com/OpenZeppelin/openzeppelin-contracts

Ownership Token:

The ownership token inherits from the ERC-721 token standard. Additionally, `Counter.sol` and `Ownable.sol` were imported. Ownable is a module which provides access control functions. The smart contract also includes five functions to operate its eco-system. The transaction fee and gas usage can be seen on Fig. 6.2.

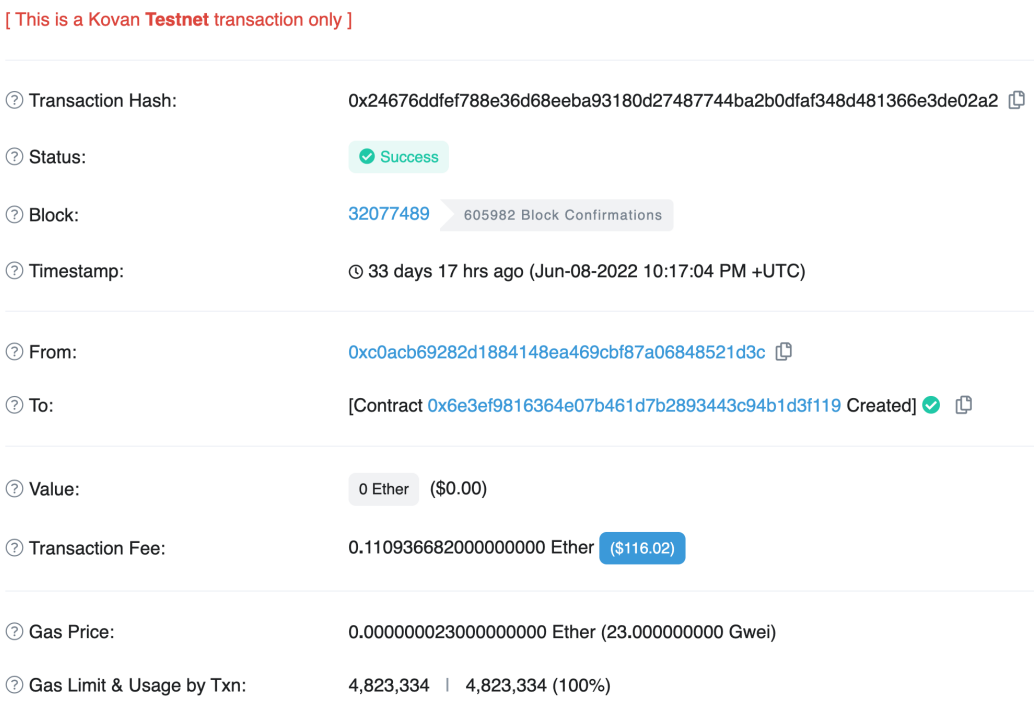

Figure 6.2: The contract creation of the ERC-721 Ownership token requires incredibly much gas. This costs can increase really fast.

Here we can evaluate, that the contract creation of the openzeppelin's ERC-721 token standard demands a large amount of gas and this can scale depending on the current gas price and Ethereum price. This is due to the inefficient imported smart contracts, for example, `ERC721Enumerable.sol` [34]. This issue is already known in the Ethereum developers community. They address that the `totalsupply` function requires very much gas and the `ERC721Enumerable.sol` import should be replaced with `Counters` to track the token ID and quantities of a token [35]. We updated the code and tested the smart contract without the ERC-721 enumerable extension. The results can be seen in Fig. 6.3. In conclusion, gas usage and gas fees dropped by half in comparison to the old smart contract.

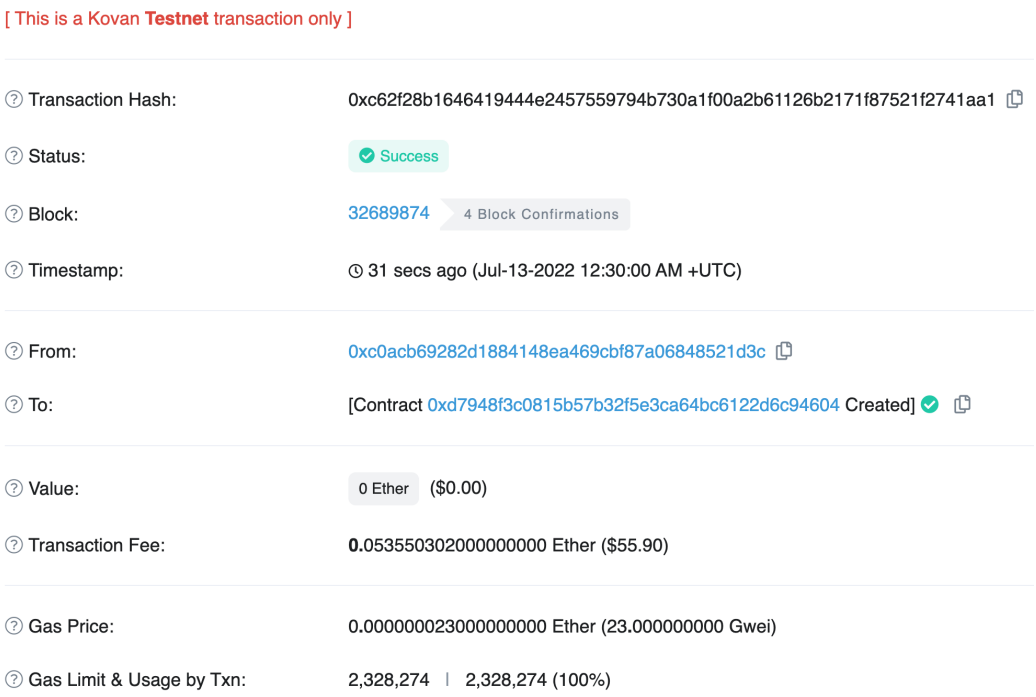

Figure 6.3: By using counters instead of the ERC-721 Enumerable smart-contract functions, the gas usage dropped by half.

Data Set Token:

The Data Set Token is an ERC-1155 multi token that provides multiple tokens for a unique token ID. Thus, we need to import the ERC-1155 contract from `@openzeppelin/contracts/token/ERC1155/ERC1155.sol`, further `Ownable.sol`, and the utils `SafeMath.sol` and `Strings.sol`. The smart contract consists of a struct and six functions. Fig. 6.4 shows the gas usage and transaction costs of the data set token smart contract.

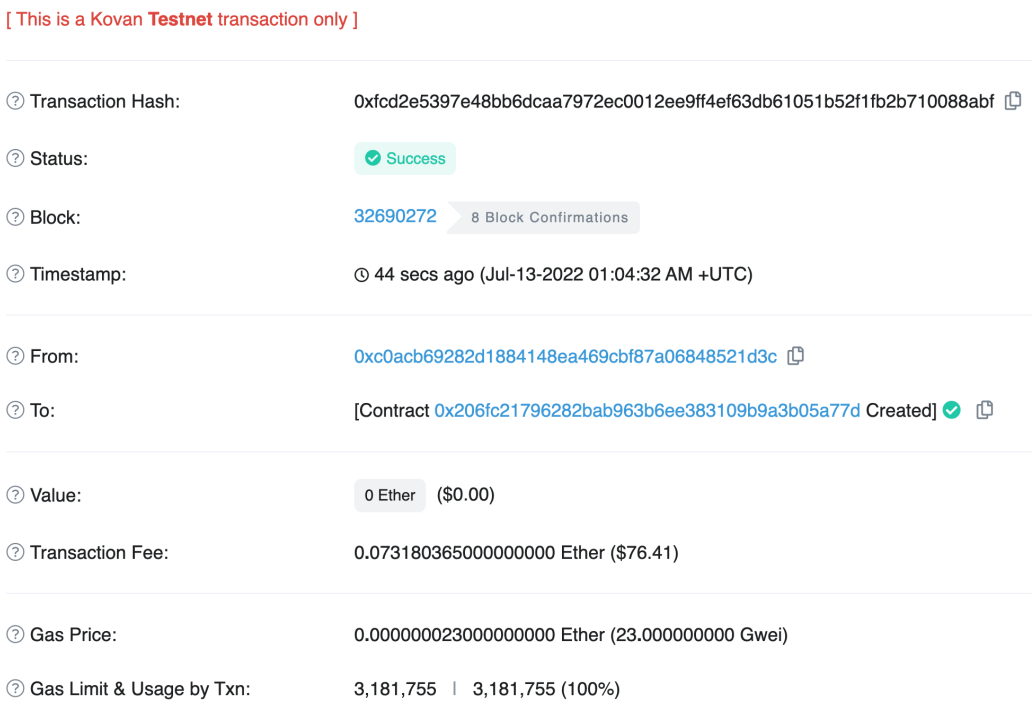

Figure 6.4: The ERC-1155 token takes a lot of gas for the contract creation. This is because of the numerous ERC-1155 token standard functions.

6.1.2 Contract Functions

Next, we analyze the smart contract functions that are integrated with dApp. These are the fees which the users are going to pay when they interact with the dApp. It is important to keep gas usage low, if the gas prices increase, in order to maintain low transaction fees.

In the sections below, we will explore the gas usage for each operator and lists them in the following table:

Operator	Gas Usage	Transaction costs in USD
Buy Data Sets	135.930	$3,27
Sell Data Sets from Balance	248.634	$5,99
Mint Ownership Token	193.916	$4,69
Mint Data Token	235.253	$5,68
Approval	26,796	$0,64
Provide and Sell Data Sets	704.599	$17,03

Table 6.1: Gas Price: 23 Gwei, Ethereum Price: $1000. The Gas Usage is the average number taken from the blockchain explorer (4.1.1).

A data provider needs to complete six steps in order to upload a data set and provide and list it on the marketplace. Within these six steps (5.10), a number of four smart contract functions need to be called. Especially, the minted data token is sent to the data provider's wallet first and then transferred to the marketplace contract address for listing. Instead of these two steps, a function where the data token is minted and listed by the marketplace smart contract, could improve gas usage and lower the transaction fees.

The transaction fees always depend on the market activity and Ethereum value. During the last two years, the Ethereum price was pretty volatile and the number of transactions varies every day. In chapter 7 we will discuss the blockchain choice.

6.2 Security and Potential Attacks

The data retrieval of our data repository is carried out by creating a download URL with the data set content. This URL is generated with the IPFS file hash, that is stored in the metadata JSON file. The already provided data sets possess a unique token ID and the provided metadata, which are not encrypted. Due to the public read function `uri(uint256 tokenID)`, everybody can display the metadata of a token by calling this function with a token ID. Especially, it is not necessary to own a or any data set token to call this function. This results in non-authorized users accessing data sets, which is unacceptable. This issue can be fixed by adding roles to the data set token, so that only the data set token owner can read the data set file

hash from the metadata. Another solutions could be to first encrypt and then store the data set file hash. The file hash can be decrypted only with the data set token.

6.3 User Experience

The implemented eco-system for distributed management of data and monitoring of use provides a clear UI makes operations intuitive and simple. Each page on our dApp represents a functionality, for example, the "Marketplace" page provides a view on the available collections and the items within a collection. The "Balance" page shows the purchased data set token and provide the functions to list data set token, to monitor the use on the blockchain explorer, and to access the data set content. The user can see his sales and purchases on the "Transaction" page.

Because of using dApps require some tools and knowledge, for example, a crypto wallet, basic understanding of transactions and of the economic model behind a token, the state of most dApps are not really great. The minimalist design of our eco-system keeps the UI as simple as possible. Furthermore, instead of searching persistent on the blockchain explorer for analytics. The verified users, token exchanges, and usages of a data set can be seen directly by clicking on the "Explore" button (Fig. 5.4). These features ensure our dApp a great user experience.

6.4 Use Case Elaboration

In comparison to the related work, our eco-system provides a marketplace for data sets with distributed management of data, monitoring of use, access, and data monetization. As in chapter 1.3 described, this eco-system can be suggested as a SaaS to industries with shared data pools, and where data start getting attention, for example, the mining industry. With our three main user roles: data provider, data user, and data polisher, our eco-system creates a data market with lots of opportunities. The so far not used data can be provided, sold or waited to be polished for own benefits.

7 | Conclusion

This bachelor thesis introduces the conceptual design and implementation of a data repository for distributed management of data as an integration and exchange platform. In these years, the most successful companies profit from big data analytics and generate additional revenue streams by selling collected data. A decentralized and self-managed data repository provides users a platform to integrate and trade data sets. Thus, the implemented data repository manages information about the availability, provenance and life-cycle of data. Besides, the usage of a data set can be examined and monitored with the blockchain explorer. Along with the ability to manage data sets on IPFS and access control through tokenization, our eco-system introduces an incentive model to motivate users.

To operate as in chapter 3.2 defined implementation goals, the dApp is built on a blockchain. For developing and testing purposes, we chose the Kovan Testnet as blockchain and DLT for our data repository. Furthermore, the Moralis SDK and Servers are used to provide users a simple and user-friendly interface to connect their crypto wallets to the dApp. Additionally, the data repository is built on the Ethereum-boilerplate (see chapter 4.2.3). ReactJS and react-moralis are used to implement the structure and the content of the pages. The smart contracts for our eco-system are compiled and deployed with the Remix IDE from our MetaMask wallet. The provided data sets are stored on IPFS and distributed via file hash. The provided data sets are uploaded to IPFS and we store the file hash code in the metadata JSON file from each data set token.

In the evaluation chapter, the transaction fees for initial contract creation, buying, selling and providing are analyzed and considered. Furthermore, we introduced the security mechanism and potential attacks on our eco-system. The process to provide a data set is quite complex, due to interacting with four smart contract functions. This inefficient issue can be seen in Fig. 5.10 and fixed in future work. Besides, we elaborated our eco-system, considering the user experience and the introduced use case in chapter 1.3. In conclusion, the transaction fees depend on the chosen blockchain network. On the

Ethereum Mainnet, the transaction fees are slightly high in comparison to other networks. Also, the gas price on the Ethereum Mainnet varies from day to day, due to the inconsistent amount of transactions daily and the high volatile Ethereum price.

The evaluation shows, that a data monetization repository for distributed management of data and monitoring of use provides a clear and user-friendly marketplace for the big data market. Thus, this eco-system can be provided as a Software as a service (SaaS) system for companies with shared data pools in the same industry or franchises.

7.1 Improvements and Future work

Consider to the evaluation results, our eco-system can be improved in sight of monetization model, data retrieval, encryption, monitoring of benefit, chaining data sets, implementing of a incentive system. Alternately, a user could pay for the use or benefits of a specific data, instead of paying only for the access, which is similar to Chainlink's data feeds in chapter 2.2.1. Here, the user is paying depending on the generated value of a data set. In this scenario, our platform is connected to the customer's APIs and the purchased data is streamed directly through the API. This prevents that the stored file hash in the metadata from a data set token could be leaked. In addition to that, the file hash could be encrypted and stored in multiple locations. Also, the monitoring of the data sets could be extended and displayed with more details and information. Especially, the generated value could be monitored and evaluated. The metadata from a polished data sets could be enhanced with original raw data set as an attribute. The polished data sets could be listed on the raw data set card to present the polished results from this raw data set. Further, the incentive system could be extended with rewards for data polishers, rewards for rating the reviewing a purchased data set, and more.

List of Abbreviations

IoT ..... Internet of Things

PoC ..... Proof of Concept

SDK ..... Software Development Kit

API ..... Application Programming Interface

IPFS ..... InterPlanetary File System

DSS ..... Data Storage System

ERC ..... Ethereum Request for Comments

NFT ..... Non-Fungible Token

DON ..... Decentralized Blockchain Oracle Network

IDS ..... International Data Space

P2P ..... Peer-to-peer

DLT ..... Distributed Ledger Technology

AI ..... Artificial Intelligence

SPoF ..... Single Point of Failure

DAG ..... Directed Acyclic Graph

DHT ..... Distributed Hash Table

dApp ..... Decentralized Application

npm ..... Node Package Manager

DEX ..... Decentralized Exchange

UI ..... User Interface

ABI ..... Application Binary Interface

URI ..... Uniform Resource Identifier

IDE ..... Integrated Development Environment

VM ..... Virtual Machine

SaaS ..... Software as a service

DeFi ..... Decentralized Finance

List of Figures

List of Tables

Bibliography

[1] Christo Petrov. *25+ Impressive Big Data Statistics for 2022*. URL: https://techjury.net/blog/big-data-statistics/ (visited on 06/10/2022).

[2] Statista. *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025 (in zettabytes)*. 2021. URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (visited on 12/06/2021).

[3] IDC. *IoT Growth Demands Rethink of Long-Term Storage Strategies, says IDC*. URL: https://www.idc.com/getdoc.jsp?containerId=prAP46737220&utm_medium=rss_feed&utm_source=A (visited on 12/06/2021).

[4] Min Chen, Shiwen Mao, and Yunhao Liu. "Big Data: A Survey". In: *Mob. Netw. Appl.* 19.2 (Apr. 2014), pp. 171–209. ISSN: 1383-469X. DOI: 10.1007/s11036-013-0489-0. URL: https://doi.org/10.1007/s11036-013-0489-0.

[5] Nada Elgendy and Ahmed Elragal. "Big Data Analytics: A Literature Review Paper". In: vol. 8557. Aug. 2014, pp. 214–227. ISBN: 978-3-319-08975-1. DOI: 10.1007/978-3-319-08976-8_16.

[6] Philip Hunter. "Journey to the centre of big data". In: *Engineering & Technology* 8.3 (2013), pp. 56–59.

[7] SiliconANGLE. *Big data market size revenue forecast worldwide from 2011 to 2027 (in billion U.S. dollars)*. 2018. URL: https://www.statista.com/statistics/254266/global-big-data-market-forecast/ (visited on 12/06/2021).

[8] IDC. *Global Spending on Big Data and Analytics Solutions Will Reach $215.7 Billion in 2021, According to a New IDC Spending Guide*. URL: https://www.idc.com/getdoc.jsp?containerId=prUS48165721&utm_medium=rss_feed&utm_source=a (visited on 12/06/2021).

[9] BARC. *BARC Data Culture Survey 22 – How to shape the culture of a data-driven organization.* URL: https://barc.de/docs/barc-data-culture-survey-22-how-to-shape-the-culture-of-a-data-driven-organization (visited on 12/06/2021).

[10] Haiyang Yu et al. "Efficient continuous big data integrity checking for decentralized storage". In: *IEEE Transactions on Network Science and Engineering* 8.2 (2021), pp. 1658–1673.

[11] Christine Ackley. *Comparing the Economics of Centralized and Decentralized Cloud Storage.* URL: https://www.storj.io/blog/comparing-the-economics-of-centralized-and-decentralized-cloud-storage (visited on 06/09/2022).

[12] Thomas Rose et al. "P ig C onomy". In: *Monetarisierung von technischen Daten.* Springer, 2021, pp. 711–733.

[13] Mike Gualtieri. *Hadoop Is Data's Darling For A Reason.* URL: https://www.forrester.com/blogs/hadoop-is-datas-darling-for-a-reason/ (visited on 06/19/2022).

[14] Chong-chong Qi. "Big data management in the mining industry". In: *International Journal of Minerals, Metallurgy and Materials* 27.2 (2020), pp. 131–139.

[15] Arash Shahin, Hadi Shirouyehzad, and Ehsan Pourjavad. "Optimum maintenance strategy: a case study in the mining industry". In: *International Journal of Services and Operations Management* 12.3 (2012), pp. 368–386.

[16] Osden Jokonya. "Towards a Big Data Framework for the prevention and control of HIV/AIDS, TB and Silicosis in the mining industry". In: *Procedia Technology* 16 (2014), pp. 1533–1541.

[17] Windows Azure. *Blockchain.* URL: https://azure.microsoft.com/en-us/solutions/blockchain/#solution-architectures (visited on 06/08/2022).

[18] Chainlink. *Decentralized Oracles for Blockchain Use Cases | Chainlink.* URL: https://chain.link/education/blockchain-oracles (visited on 06/18/2022).

[19] Filecoin. *What is filecoin?* URL: https://docs.filecoin.io/about-filecoin/what-is-filecoin/ (visited on 06/18/2022).

[20] Ocean Protocol. *OceanONDA V4 is now live with data NFTs, solving rug pulls and better community monetization.* URL: https://oceanprotocol.com/press/2022-06-08-ocean-onda-v4-live (visited on 06/08/2022).

[21] Muqaddas Naz et al. "A Secure Data Sharing Platform Using Blockchain and Interplanetary File System". In: *Sustainability* 11.24 (2019). ISSN: 2071-1050. DOI: 10.3390/su11247054. URL: https://www.mdpi.com/2071-1050/11/24/7054.

[22] Sisense. *Data Monetiztion: A Definition.* URL: https://www.sisense.com/data-monetization/ (visited on 06/08/2022).

[23] McKinsey. *Fueling growth through data monetization.* URL: https://www.mckinsey.com/business-functions/quantumblack/our-insights/fueling-growth-through-data-monetization (visited on 06/08/2022).

[24] Elizbeth Mixson. *Measuring and Maximizing Data ROI: A Quick Guide to Data Monetization.* URL: https://www.aidataanalytics.network/data-monetization/articles/a-quick-guide-to-data-monetization (visited on 06/08/2022).

[25] Jamie. *Launch a blockchain-based data marketplace in under 1 hour.* URL: https://blog.oceanprotocol.com/launch-a-blockchain-based-data-marketplace-in-under-1-hour-9baa85a65ece (visited on 06/19/2022).

[26] Ethereum. *Ethereum Yellow Paper.* URL: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 12/14/2021).

[27] Frauenhofer. *International Data Spaces is a global de facto market standard for the sovereign use of data.* URL: https://www.dataspaces.fraunhofer.de/en/InternationalDataSpaces/idsa.html (visited on 12/14/2021).

[28] Yao-Chieh Hu et al. "Hierarchical interactions between ethereum smart contracts across testnets". In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. 2018, pp. 7–12.

[29] shrey vijayvargiya. *Introduction to Moralis.* URL: https://blog.cryptostars.is/introduction-to-moralis-f2b7f69986c9 (visited on 06/09/2022).

[30] Moralis. *Moralis Docs.* URL: https://docs.moralis.io/moralis-dapp/connect-the-sdk/boilerplate-projects (visited on 06/09/2022).

[31] Moralis. *Ethereum React Boilerplate*. URL: https://ethereumboilerplate.com/ (visited on 06/09/2022).

[32] Praveenkumar Bouna. *Visual Studio Code for C# Developers*. 2022.

[33] David Easley, Maureen O'Hara, and Soumya Basu. "From mining to markets: The evolution of bitcoin transaction fees". In: *Journal of Financial Economics* 134.1 (2019), pp. 91–109.

[34] Gavin Zheng et al. "Solidity Advanced Topics". In: *Ethereum Smart Contract Development in Solidity*. Springer, 2021, pp. 85–136.

[35] Tom Hirst and Jonathan Snow. *Cut Minting Gas Costs By Up To 70% With One Smart Contract Tweak*. URL: https://shiny.mirror.xyz/OUampBbIz9ebEicfGnQf5At_ReMHlZyOtB4glb9xQ0E (visited on 06/20/2022).