TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Igor Mahlinovski 213658
Alexander Khrushkov 213651

DEVELOPING A BLOCK EXPLORER FOR THE NEXT-GENERATION BLOCKCHAIN: A TOOL TO ENHANCE THE NETWORK TRANSPARENCY

Bachelor's Thesis

Supervisor: Pavel Grigorenko
PhD

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Igor Mahlinovski 213658
Alexander Khrushkov 213651

BLOCK EXPLORER'I ARENDAMINE JÄRGMISE PÕLVKONNA PLOKIAHELA JAOKS: TÖÖRIIST VÕRGU LÄBIPAISTVUSE PARANDAMISEKS

Bakalaureusetöö

Juhendaja: Pavel Grigorenko
PhD

Tallinn 2024

Authors' Declaration of Originality

We hereby certify that we are the sole authors of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Authors: Igor Mahlinovski, Alexander Khrushkov

20.05.2024

Abstract

This bachelor's thesis presents the development of an explorer web application for Alphabill blockchain, with the primary focus on enhancing network transparency and user interaction with blockchain data. We describe the development process including the architecture, used technologies, such as React, Vite, Golang and providing implementation examples taken from the code. In the thesis we demonstrate the importance of scalability, reliability, and performance in developing blockchain solutions, highlighting significant milestones such as the successful implementation of a transactions and blocks tracking systems.
In addition, we also highlight the importance of effective planning, collaboration and problem-solving to overcome the faced challenges.
Finally, we identify key areas for future development, including the support for additional Alphabill's transaction systems, improvements to user interface and user experience.

The thesis is written in english and is 60 pages long, including 5 chapters, 23 figures.

Annotatsioon

See bakalaureusetöö tutvustab veebirakenduse arendamise protsessi Alphabill'i exploreri jaoks, keskendudes peamiselt võrgu läbipaistvuse parandamisele ja kasutajate kogemusele blockchaini andmete töötlemisel. Kirjeldame arendusprotsessi, sealhulgas arhitektuur, kasutatud tehnoloogiad nagu React, Vite, Golang ning koodist võetud implementeerimise näited. Bakalaureusetöös me näitame skaleeritavuse, turvalisuse ja jõudluse tähtsust blockchaini lahenduste arendamisel, rõhutades olulisi saavutusi nagu edukas tehingute ja plokkide jälgimissüsteemide rakendamine.
Lisaks sellele rõhutame ka planeerimise, koostöö ja probleemide lahendamise olulisust väljakutsetega hakkama saamisel.
Lõpuks tuvastame tuleviku arenduse valdkondi, sealhulgas täiendavate Alphabilli tehingute süsteemidele toetuse rakendamine, kasutajaliidese ja kasutajakogemuse parendamine.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 60 leheküljel, 5 peatükki, 23 joonist.

List of Abbreviations and Terms

ALPHA	Native currency of Alphabill
API	Application Programming Interface
AVL	Adelson-Velsky and Landis
BNB	Binance coin
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DAO	Decentralized Autonomous Organization
Dapp	Decentralized application
DeFi	Decentralized Finance
DOM	Document Object Model
ETH	Ethereum
EVM	Ethereum Virtual Machine
HMR	Hot Module Replacement
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IOT	Internet Of Things
JSON	JavaScript Object Notation
KAS	Kaspa
LTC	Litecoin
MPA	Multi-Page Application
MVC	Model-View-Controller
NFT	Non-Fungible Token
NoSQL	Not only Structured Query Language
PC	Personal Computer
REST	Representational State Transfer
RPC	Remote Procedure Call
SEO	Search Engine Optimization
SHA256	Secure Hash Algorithm 256-bit
SOL	Solana
SPA	Single Page Application
SQL	Structured Query Language
SSR	Server-Side Rendering

UI	User Interface
URL	Uniform Resource Locator
USDT	United States Dollar Tether
UTXO	Unspent Transaction Output
UX	User Experience
VM	Virtual Machine
VSCode	Visual Studio Code
VVM	Model-View-ViewModel
WSL	Windows Subsystem for Linux

Table of Contents

1. Introduction

1.1 Background and motivation

Blockchain technology has revolutionized the way we perceive and conduct transactions, offering unprecedented transparency, security, and decentralization. At the core of every blockchain network lies a distributed ledger, a tamper-proof record of transactions shared across a network of computers. While this technology has gained widespread adoption across various industries, its true potential can only be realized with effective tools that enable users to navigate and understand the data stored within these ledgers. This is where blockchain explorers come into play, enhancing the readability of blockchain data for users.

1.2 Goals

Our goal is to provide access to information within the Alphabill blockchain, making it transparent and accessible to everyone, regardless of their technical expertise. We envision a platform that serves as a window into the inner workings of the blockchain, providing users with a clear and comprehensive view of its activities and transactions. Rather than teaching people about the intricacies of blockchain technology, our focus is on creating a user-friendly interface that allows individuals to effortlessly explore and understand the data stored on the Alphabill blockchain. Through intuitive design and simplified navigation, our platform will enable users to follow the flow of transactions, monitor blocks, and track the states of units in real-time. By removing barriers to accessing the information and presenting it in a clear and digestible format, we aim to empower users to engage with the Alphabill blockchain confidently. Whether they are seasoned blockchain enthusiasts or newcomers to the technology, our platform will provide everyone with the tools they need to explore and interact with the blockchain ecosystem. Ultimately, our mission is to foster transparency and trust within the Alphabill community, enabling users to verify transactions, validate data, and participate in the blockchain network with ease and confidence. Through our efforts, we aspire to make the Alphabill blockchain a model of transparency and accessibility for blockchain ecosystems worldwide.

1.3 Problem statement

The problem lies in the lack of transparency within Alphabill's blockchain, hindering users' ability to access and interpret crucial information about transactions, blocks, and network activity. This opacity not only undermines user trust but also impedes the blockchain's potential for widespread adoption and utilization.

1.4 Outline of the thesis

Prerequisites
This part describes the concepts and technologies crucial for the project's implementation and evaluation.

System Design and Implementation
This chapter describes the technical aspects of the built application, including the chosen technologies, software architecture, development tools, and UI design.

Evaluation of approach
In this part, we assess the results of the project, the development process, management, and communication.

Conclusion
We highlight the significance of the given work and describe areas for future research and development.

2. Prerequisites

2.1 Blockchain

Most people associate blockchain with Bitcoin [1], but it is actually a much broader technology. Imagine a giant, digital record book — that's essentially what blockchain is. This record book keeps track of transactions, but unlike a regular bank ledger, it is not stored in one place. Instead, copies are spread out across a network of computers around the world. Figure 1 represents the difference between the centralized and decentralized network working principles. This means there's no single person or bank in control. Instead, everyone on the network can see the same information, making it very secure and difficult to tamper with [2]. Transactions are grouped into "blocks" and a block gets linked to the previous one using a cryptographic hash, forming a chain — hence the name "blockchain." Think of it like a chain of building blocks, where each block contains information about transactions. Since everything is linked together, it is almost impossible to change or erase a transaction once it is added to the chain. While Bitcoin uses blockchain for its digital currency, the technology itself has many other potential uses. It can be used as a secure registry for anything from ownership of assets to tracking the movement of goods.

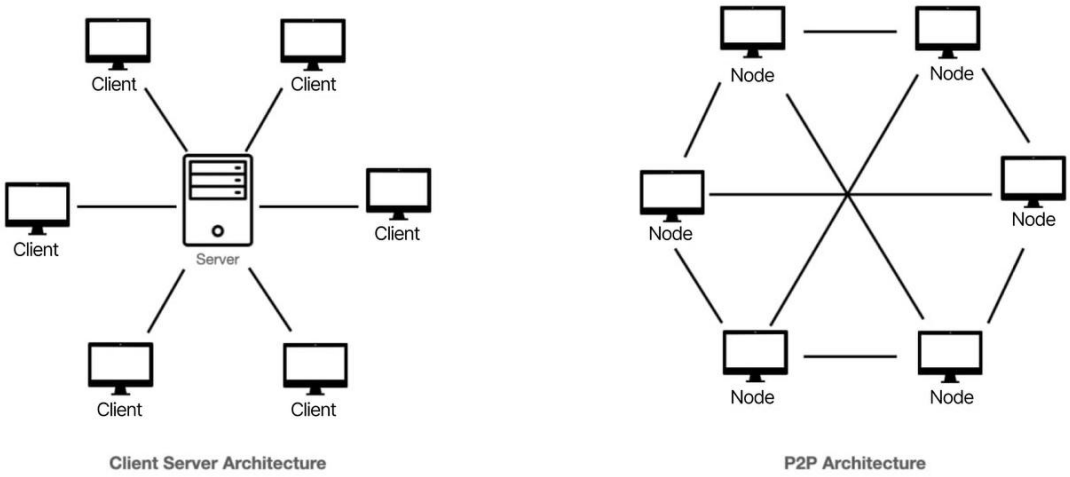

Figure 1. Client-Server vs P2P Network [3]

The year 2008 marked a turning point in technology with the publication of a paper titled

"Bitcoin: A Peer-to-Peer Electronic Cash System." This paper, published by Satoshi Nakamoto, introduced the concept of blockchain. Nakamoto's vision was revolutionary — a system for direct online payments between individuals, bypassing the need for traditional financial institutions. This groundbreaking idea relied on cryptography, a complex form of digital encryption, to ensure security and trust [1, 4, 5].

2.2 Cryptocurrency

Cryptocurrency is a digital currency that uses cryptography to secure transactions and control the issuance. This technology allows users to transact directly between themselves without having to trust a third party, such as a bank or payment services [6, 7].
Nowadays Bitcoin and Ethereum are the most popular cryptocurrencies in the world. Bitcoin was the first cryptocurrency and is still the most valuable by market capitalization. Ethereum is notable for introducing the concept of smart contracts, allowing developers to create decentralized applications on its platform [7, 5].

2.2.1 Altcoins

The term "altcoin" is a blend of "alternative" and "coin," and it encompasses all cryptocurrencies and tokens except the original - Bitcoin. Many altcoins are actually "forks" of existing blockchains, particularly Bitcoin and Ethereum. These forks often arise from disagreements within the developer community, leading them to create altcoins with unique features or functionalities [8].
Examples of altcoins: ETH, LTC, SOL, BNB, KAS.

2.2.2 Stablecoins

Stablecoins are a type of cryptocurrency made to minimize volatility by pegging their market value to the fiat or other assets [9]. This pegging mechanism helps maintain a stable value, which is useful for common transactions, DeFi services and allows the use of stablecoins as a payment method for goods and services.

2.2.3 Non-Fungible Tokens (NFTs)

NFTs represent a unique concept in digital ownership. Unlike traditional currencies where every unit is fungible, each NFT is one-of-a-kind and cannot be directly replaced by another NFT of the same type. NFTs are stored on a secure blockchain ledger, ensuring a transparent and tamper-proof record of ownership. This system guarantees that only one

person can own a specific NFT at a time. This unique ownership model opens doors for buying, selling, and trading unique digital assets within the ever-evolving digital world [10].

2.3 Block Explorers

To enhance the readability of blockchain data for users, numerous block explorers exist. These tools list all blocks and transactions, providing users with the ability to verify transactions and assess the network's status [11, 12].

2.3.1 Transaction Tracking

Explorers allow users to track the details of transactions, including the unique transaction hash, the amount of cryptocurrency transferred, and the time of the transaction. They can also show the fees associated with each transaction and the current state, whether it's pending or confirmed in the blockchain [11].

2.3.2 Address Monitoring

Users can monitor any cryptocurrency address, viewing the balance and all incoming and outgoing transactions. This allows to verify if a transaction has reached its intended destination or analyze the address interactions with other addresses or smart contracts.

2.3.3 Block Verification

Block Explorers enable observation of individual blocks, revealing information such as when the block was proposed, the validator's address, and the block's size. This data is important for those interested in mining.

2.3.4 Network Health and Statistics

Some explorers offer comprehensive charts and data visualizations that reflect the network's overall status. These may include metrics like transaction fees over time, the average value of transactions, mining difficulty, and hashrate.

2.3.5 Multi-Currency and Multi-Chain Support

While some explorers are dedicated to a single cryptocurrency, others support multiple blockchains, offering a unified interface to explore different networks. This multi-chain functionality is particularly beneficial for users dealing with various cryptocurrencies.

2.3.6 Educational Resources

In addition to exploring blockchain data, some platforms provide educational resources about the blockchain and cryptocurrency space. These can include guides on smart contracts, tokenomics, and decentralized finance, helping users expand their knowledge.

2.3.7 Real-Time Updates

Explorers often provide real-time information, such as the latest blocks mined and unconfirmed transactions waiting to be added to blocks. This helps users stay updated with the most recent activities on the blockchain [11].

2.4 Alphabill

Alphabill is a next-generation blockchain platform aiming to overcome limitations in existing technologies. At the heart of AlphaBill's architecture lies the concept of partitions, which represent sub-trees of the global state tree within the network as provided in Figure 2. The partitions are shardable and serve to optimize workload distribution and enable parallel transaction processing, enhancing the platform's scalability and performance. Among the various types of partitions in AlphaBill, the Money partition governs the management of the platform's native currency, ALPHA, and provides an ability to make payments from one wallet to another, manage bills in the wallet, and swap smaller bills into one larger bill. Aplhabill also has an Atomicity Partition which ensures atomicity of transactions across different transaction systems, a User Token Partition provides management for user tokens, and an EVM Partition provides execution of smart contracts based on the EVM.

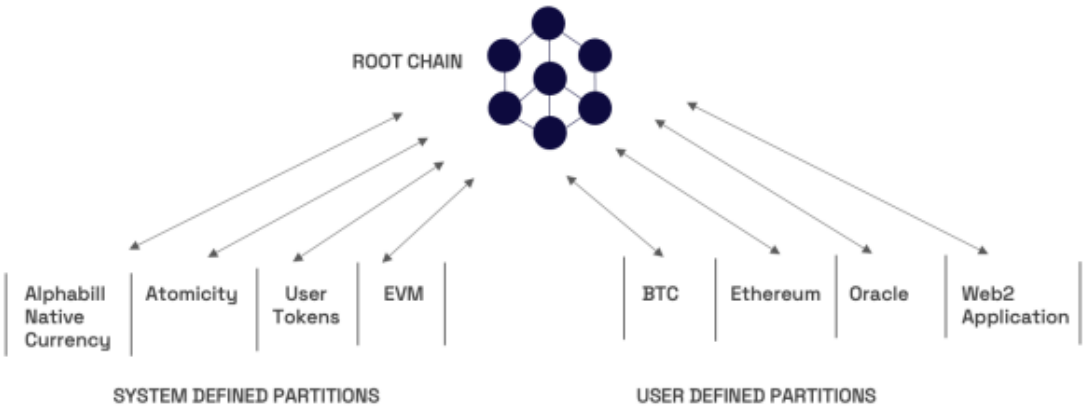

Figure 2. Alphabill Partitions showing system and user defined Partitions [13].

One of the key innovations of AlphaBill is its unique transaction model, similar to physical currency bills. This model allows for concurrent token updates and verification, avoiding the need for coordination between machines during transaction execution. Unlike the UTXO and account-based models used in Bitcoin and Ethereum, respectively, AlphaBill's bill model offers improved scalability and performance, making it suitable for applications requiring high transaction throughput.

The platform's state tree, based on a self-balancing AVL tree, further enhances its efficiency and flexibility. Each node in the tree represents a token, enabling independent token verification and sharding. This decentralized approach to state management ensures the integrity and security of the platform, while also facilitating offline token verification and autonomous operation as provided in Figure 3 [14, 13, 15, 16].

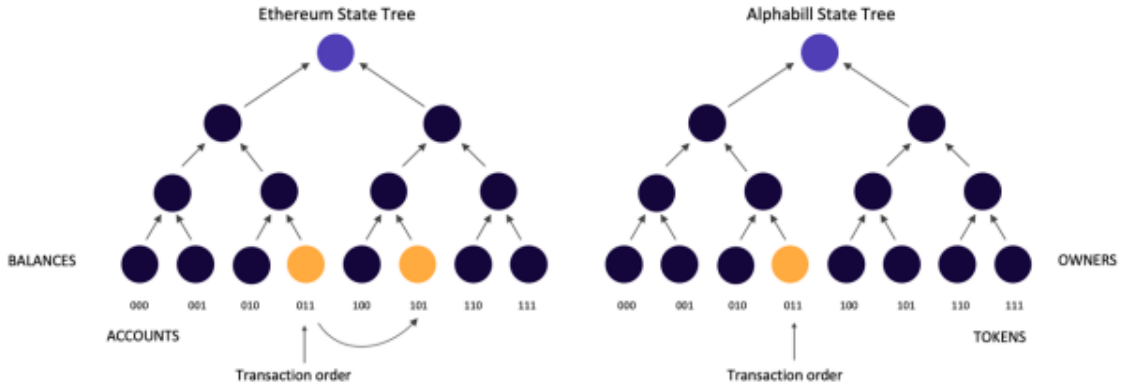

Figure 3. Ethereum uses a state tree of accounts. Alphabill uses a state tree of tokens [13].

3. System Design and Implementation

In this section, we provide a detailed overview of the design and implementation of the Alphabill explorer. This includes user stories that describe the specific features and functionalities from a user perspective, use cases that outline how these features are utilized, backend and frontend development processes.

3.1 User Stories

3.1.1 Get a list of blocks

As a user, I want to get a list of blocks with the general details for each block, so I can get an overview of activity in the blockchain.

Endpoint

GET /blocks?start={startBlock}&limit={limit}

Parameters:

startBlock (optional): The block number from which the list begins. If not specified, the output starts from the most recently added block.

limit: The number of blocks to display.

Output

A list of blocks containing block number, hash, amount of transactions in the block, and total volume of transactions for each block in the list.

Result

The user receives a list of blocks with general details, allowing them to gain an overview of activity in the blockchain.

3.1.2 Block details

As a user, I want to select a block and see the detailed information, so I can analyze activity in the specific block.

Endpoint

GET /blocks/{blockNumber}

Parameters:

blockNumber: the number of a specific block

Output

Detailed information about the block, including block number, hash, creation time, block size, and total number of transactions in the block.

Result

The user can analyze activity in the specific block.

3.1.3 The list of transactions in a block

As a user, I want to get the list of all transactions processed within a specific block, so I can analyze each transaction in the block.

Endpoint

GET /blocks/{blockNumber}/txs

Parameters:

blockNumber: the number of a specific block

Output

A list of all transactions contained in the specified block. For each transaction, its details are returned, including hash, sender, recipient and transfer amount.

Result

The user gains the ability to explore operations within the selected block.

3.1.4 The list of transactions

As a user, I want to get a list of transactions with the general details of each transaction, so I can get a clear understanding of activity in the network and analyze the transfers.

Endpoint

GET /txs?start={startTxId}&limit={limit}

Parameters:

startTxId (optional): The transaction identifier from which the list begins. If the parameter is not specified, the list starts from the latest transaction.

limit: The number of transactions to display.

Output

A list of transactions that meet the query criteria containing hash, sender, recipient, transfer amount and fees for each transaction.

Result

The user gains access to a detailed list of transactions, being able to analyze activity in the blockchain, explore transfers, and identify suspicious transactions.

3.1.5 Transaction details

As a user, I want to see the detailed information of the selected transaction, so I can get a complete understanding of how and where tokens are moving in the blockchain.
Endpoint
GET /txs/{txHash}
Parameters:
txHash: The hash of a specific transaction
Output
Details of the specified transaction, including transaction hash, sender, recipient, transfer amount, fees, date and time.
Result
The user gains a complete understanding of the transaction, including how and where tokens are being transferred.

3.1.6 Search bar

As a user, I want to look for blocks, transactions, addresses, and units through a search bar, so I can easily access the needed information.
Endpoint
GET /search/{query}
Parameters:
query: Searching parameter such as block number, transaction hash, etc.
Output
Search results for blocks, transactions, addresses, and units based on the entered query.
Result
The user quickly finds the necessary information without the need to navigate through complex interfaces.

3.1.7 Units history

As a user, I want to find out the transactions a specific unit has been involved in, so I can track the unit's path and analyze its usage in the network.
Endpoint
GET /units/{unitID}/txs
Parameters:

unitId: The specific unit's identifier

Output

A list of transactions in which the specified Unit participated

Result

The user can track the path and usage of Units in the network.

3.1.8 Activity and balance state

As a user, I want to get the balance state and all the transactions by public key, so I can analyze it's activity.

Endpoint

GET /address/{pubKey}/bills

Parameters:

unitId: The specific unit's identifier

Output

Information about the balance state and a list of all related transactions for the specified public key.

Result

The user receives a complete understanding of the activity by the public key.

3.1.9 Network statistics and status

As a user, I want to see the statistics of the blockchain and status of the network, so I can assess it's state, health, and development dynamics.

Endpoint

GET /status

Parameters:

unitId: The specific unit's identifier

Output

Overall blockchain statistics, including the number of blocks, total number of transactions, current fees per transaction and other important metrics.

Result

The user evaluates the state, health, and development dynamics of the blockchain.

3.2 Use Cases

3.2.1 Blockchain Exploration through a Blockchain Explorer

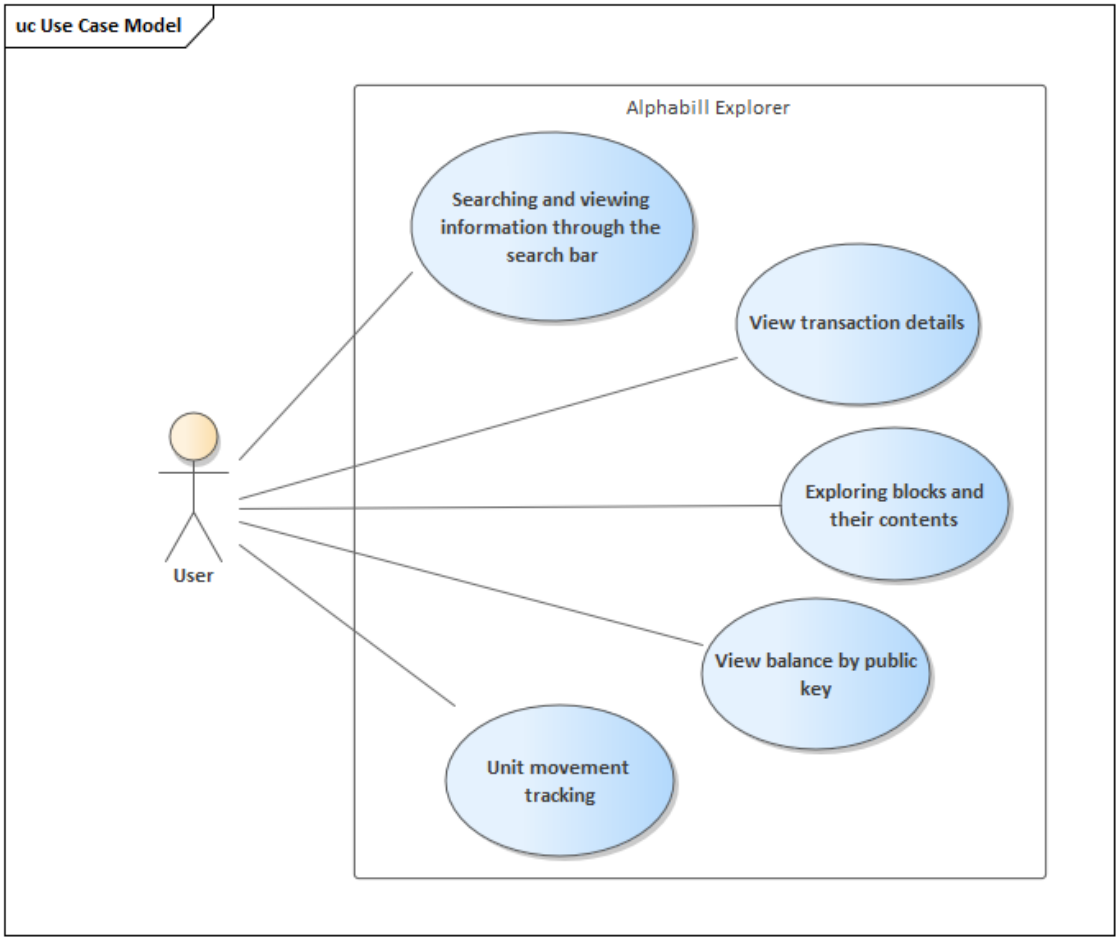

Figure 4. Use Case of Alpabill Explorer

Actor: User (includes cryptocurrency holders, investors, researchers, analysts, developers)
Invariants:
The user has access to the blockchain explorer interface through a web browser or mobile app.
Blockchain explorer is updated and synchronized with the latest blockchain state.
Main flow of events:
Search and Browse Information through the Search Bar: The user enters a relevant query (block number, transaction hash, wallet address, Unit ID) into the search bar. The system processes the request and displays the corresponding information depending on the query type.
View Transaction Details: As a result of the search or through navigation, the user can select a specific transaction to view it's detailed information.The system provides

transaction details, including the hash, fee, date, and time of execution.

Explore Blocks and Their Contents: As a result of the search or through navigation, the user selects a block to view it's details.The system displays information about the block, including its number, hash, previous block hash, creation time, and a list of all transactions within the block.

View Public Key Balance: The user enters a wallet address to track its activity. The system displays the wallet balance and a list of associated transactions, providing a detailed view of the address's activity.

Track Unit Movement: The user searches for a Unit ID to view it's transaction history. The system displays all transactions involving the given Unit, allowing the user to analyze it's path through the blockchain.

Alternative Flows:

If the system cannot find information based on the user's request, it provides an error message.

Result:

The user receives comprehensive information based on their query, enabling them to gain deeper insights and analyze blockchain data.

3.3 Backend Design and Implementation

3.3.1 Overview

The backend of the Alphabill block explorer is engineered using GO, leveraging its performance efficiency and concurrency capabilities, which are crucial for handling the extensive data processing required by blockchain technology. The backend architecture is designed to interface directly with the Alphabill blockchain, facilitating seamless data synchronization and real-time transaction monitoring.

To enhance the development environment, the backend was developed on WSL using Ubuntu [17]. This setup allowed the team to utilize Linux's powerful features and tools while still operating within a Windows environment, providing a flexible and robust development platform. The use of WSL helped in achieving a more consistent development environment, closely mirroring the production servers, and simplifying the process of managing dependencies and testing cross-platform functionalities.

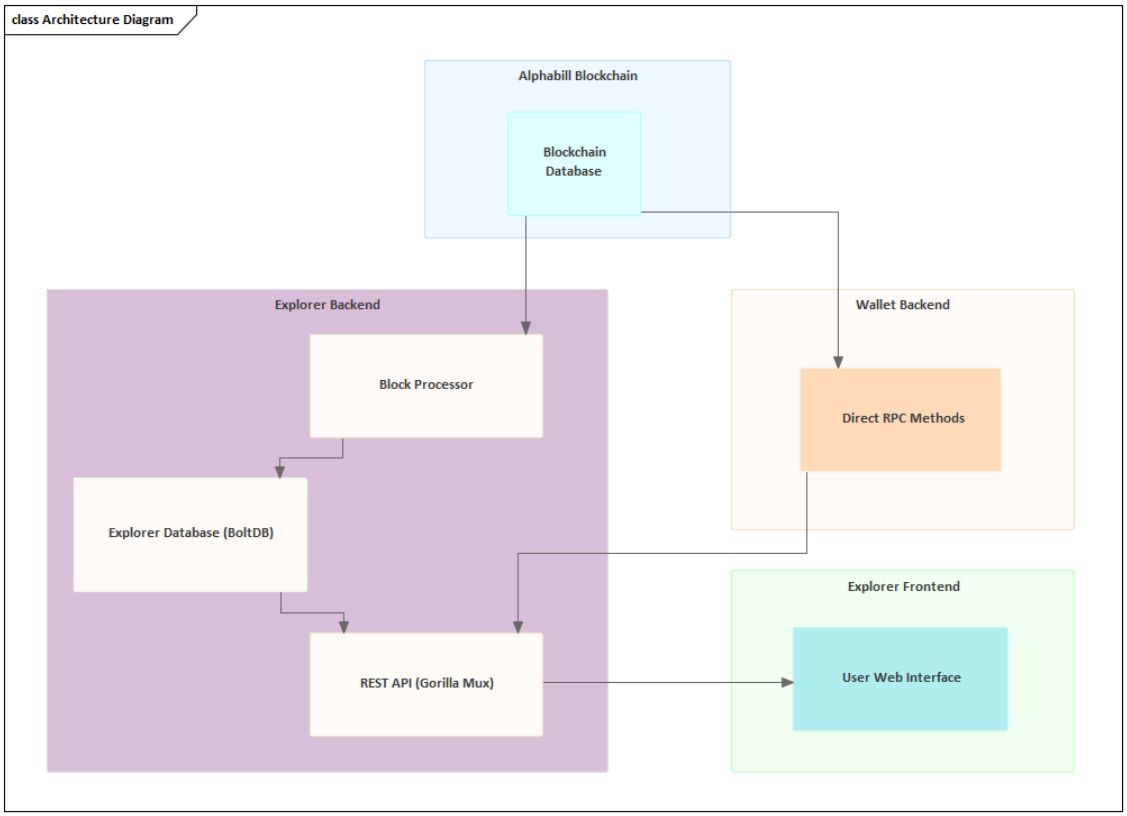

Figure 5. Architecture Diagram of the Alphabill Block Explorer Backend.

The architecture diagram provided in Figure 5 illustrates that Explorer Backend is centrally positioned within the architecture, acting as a bridge between the raw data from the Alphabill Blockchain and the processed information delivered to the UI through the REST API. This setup underscores the backend's role in data processing and distribution, ensuring efficient data flow across the system.

3.3.2 Database and Storage

The backend utilizes BoltDB, a lightweight key-value store, to manage the local storage of blockchain data. The choice of BoltDB is motivated by its simplicity and its efficiency in storing large sets of immutable data, which are common in blockchain environments. Data is organized into buckets—a form of storage that BoltDB handles as a collection of key-value pairs—allowing efficient data retrieval necessary for a block explorer.

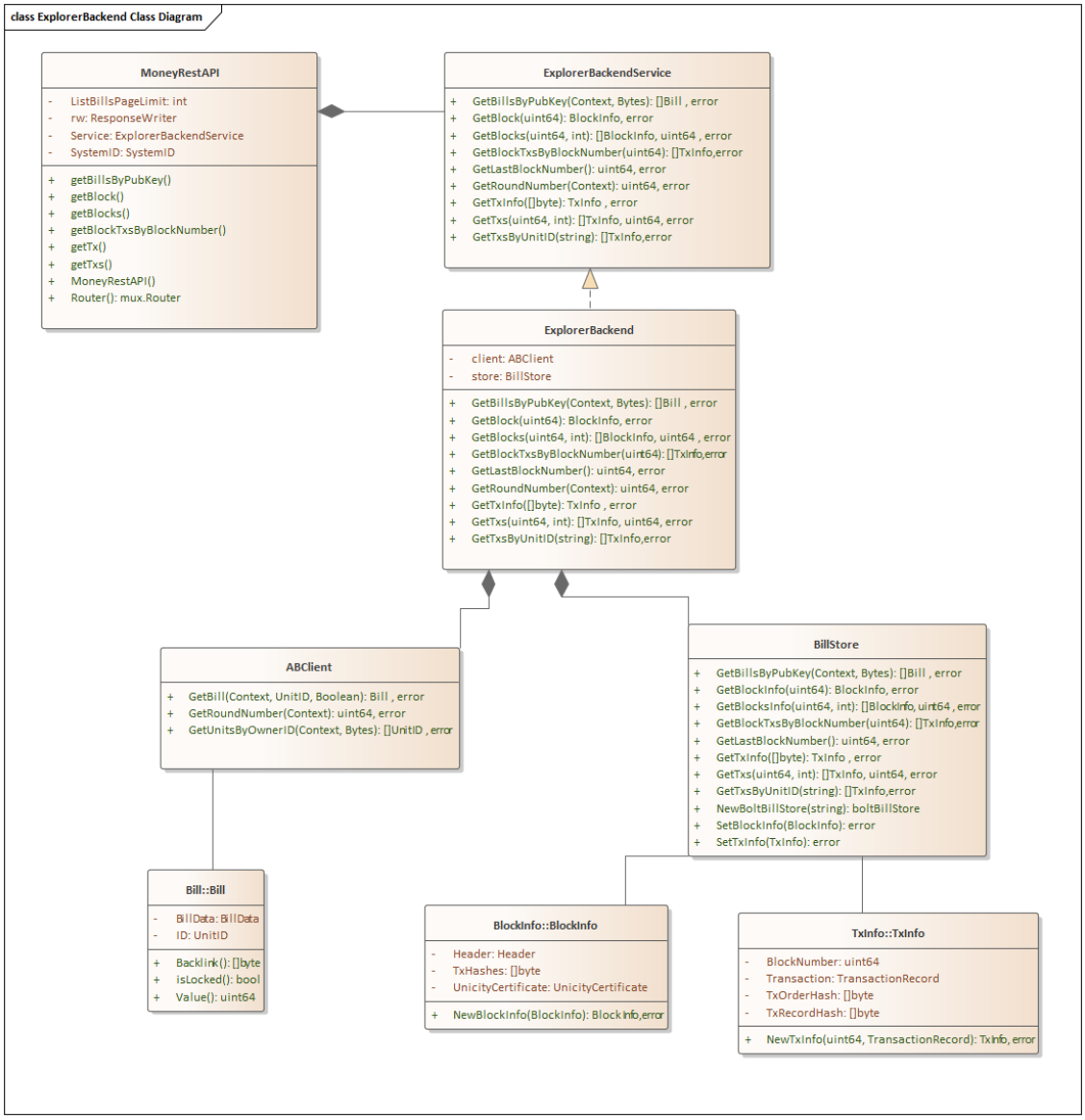

Figure 6. Class Diagram of the ExplorerBackend.

Class diagram provided in Figure 6 illustrates the various components of the ExplorerBackend, describing the interactions and relationships between different classes and highlighting the methods and properties of each class such as MoneyRestAPI, ExplorerBackendService, ExplorerBackend, ABIClient, and BillStore.

3.3.3 Data Processing

A key component of the backend is the block processor, which scans the Alphabill blockchain to fetch block information. This process involves:

- Decoding: Each block's data structure is decoded.
- Transformation: The data is transformed into a suitable format for storage.
- Storage: Transformed data is saved into BoltDB buckets for efficient retrieval.

The block processor uses a mapping system where each block number maps to a BlockInfo object stored in the blockInfoBucket. This object mirrors the block's structure but replaces transaction objects with a list of SHA256 hashes. Transactions are stored separately in the txInfoBucket using transaction hashes as keys, which are related to detailed txInfo objects.

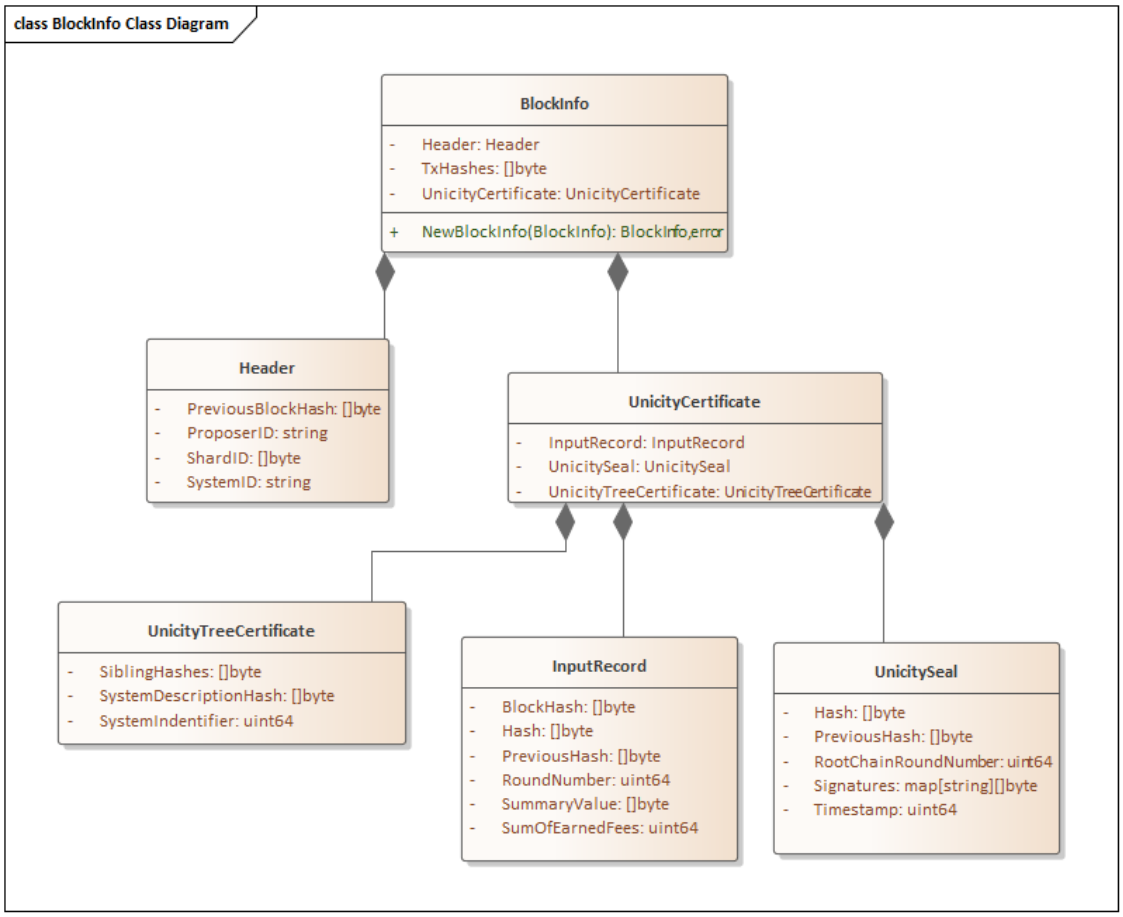

Figure 7. Class Diagram of the BlockInfo.

This class diagram provided in Figure 6 represents the structure and components of the BlockInfo class, visualizing its relations with other classes such as Header and UnicityCertificate. It highlights the methods and attributes used to process and store block data efficiently within the backend.

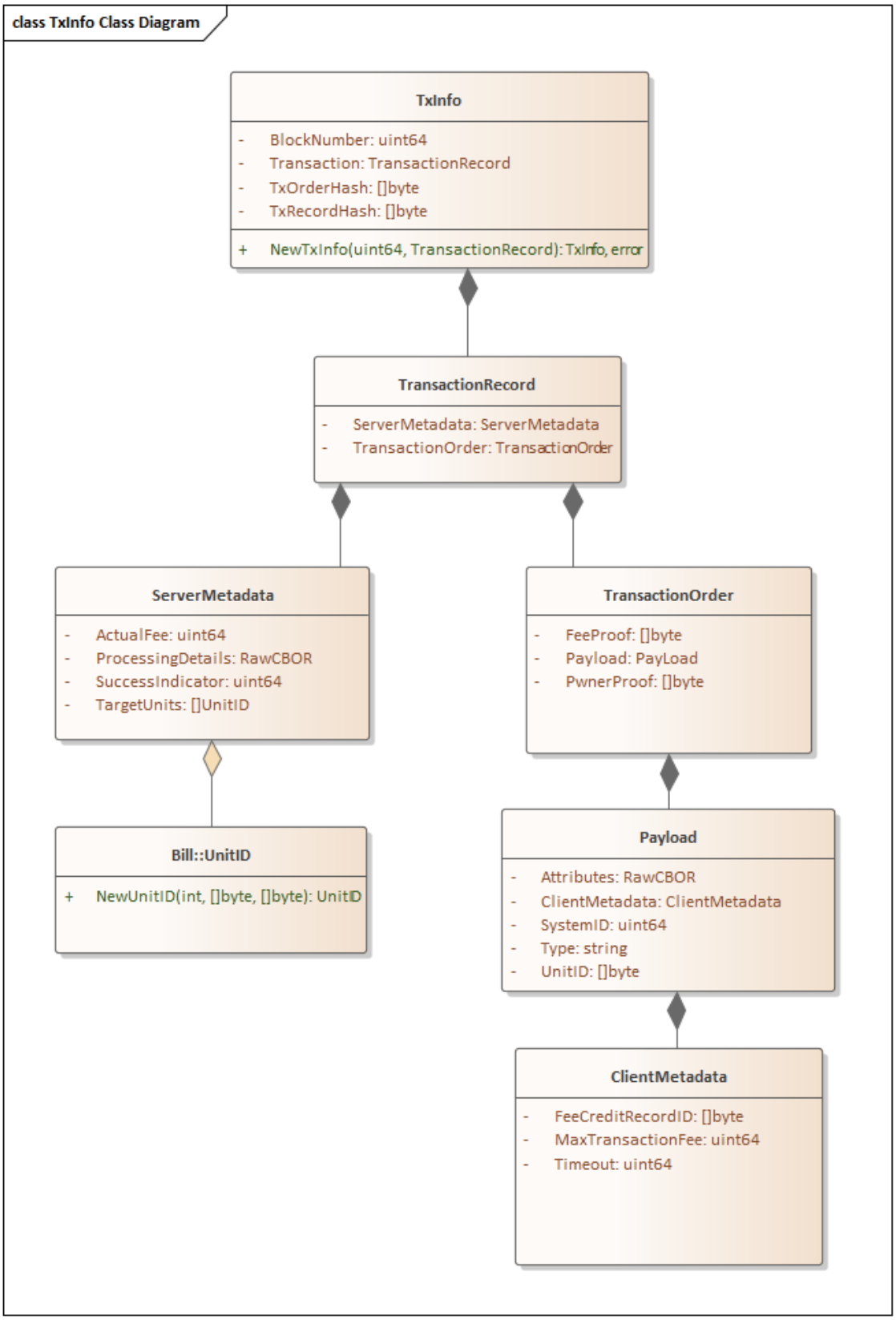

Figure 8. Class Diagram of the TxInfo.

The class diagram provided in Figure 8 represents the structure of the TxInfo class and its

associated classes, such as TransactionRecord, ClientMetadata, and TransactionOrder. It describes the relations and data flow between these classes within the backend.
After data is processed and stored in BoltDB, it is becoming accessible through the REST API, implemented using Gorilla Mux. This API plays a crucial role in retrieving processed data from the database, allowing for efficient data delivery to the Frontend. Through this API frontend fetches the necessary block and transaction data. This end-to-end handling of data not only ensures accuracy and efficiency but also enhances the user's ability to interact with and analyze the blockchain data effectively.

3.3.4 Transaction, Unit Tracking

Transactions within the blocks include units that are also tracked. Each unit's participation in transactions is recorded in the unitIDBucket, mapping unit IDs into lists of hashes. This structure supports queries for transaction history and unit tracking. Additionally, managing unitIDBucket allows the application to track the states of different units across transactions, enhancing the explorer's ability to provide detailed insights into asset flows and relations within the blockchain.

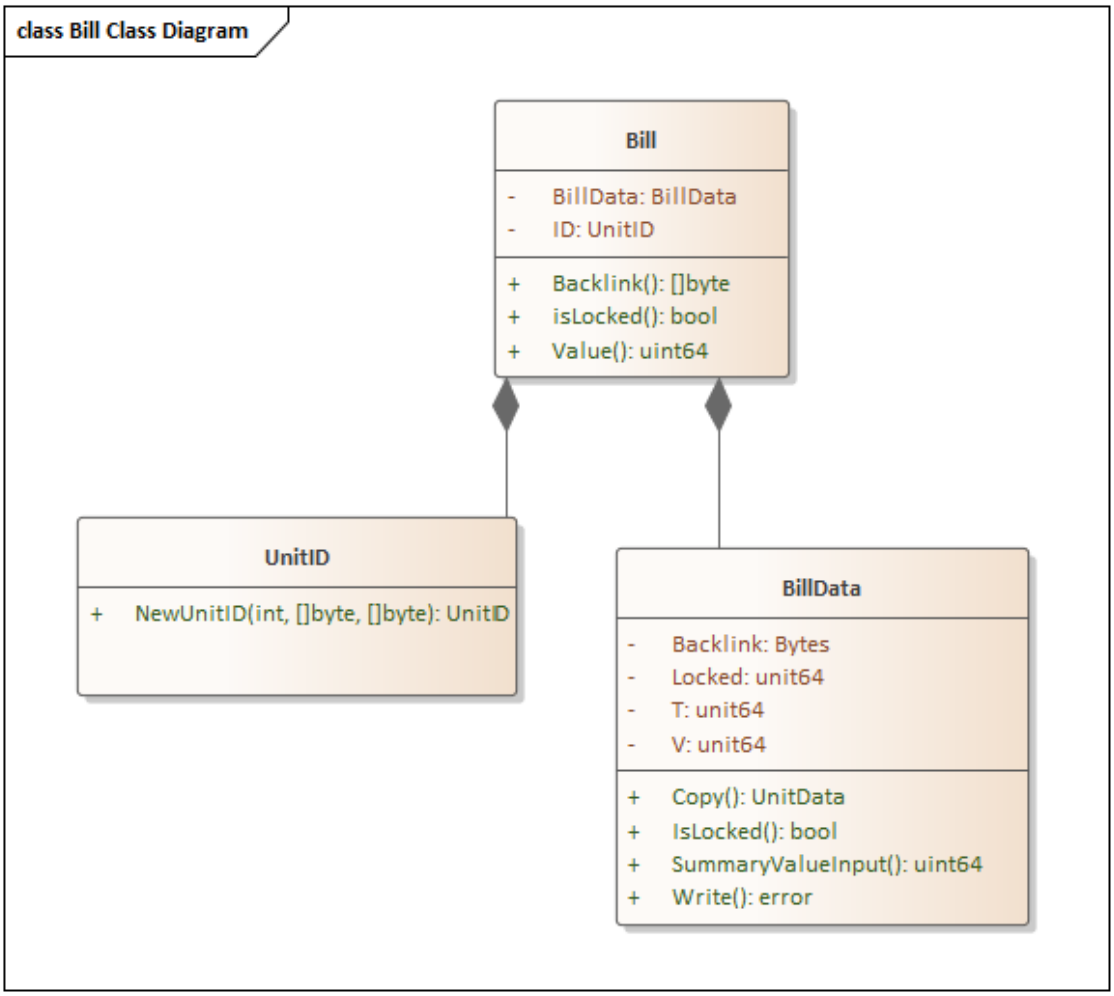

Figure 9. Class Diagram of the Bill.

The class diagram provided in Figure 9 highlights the structure of the Bill class, describing its relations with other classes such as UnitID and BillData. It emphasizes the methods and attributes involved in tracking and managing blockchain units within the backend system.

3.3.5 REST API and Endpoints

The backend provides a RESTl API, implemented using Gorilla Mux, a popular and flexible HTTP routing package for Golang. This technology drives the backend's modularity and scalability, enabling easily extend API functionalities as needed. The API's endpoints are documented using Swagger, making it easy for both internal developers and external users to understand and use the interface.

API endpoints include:

- GET requests for blocks, transactions, and units information: These endpoints provide detailed data about blocks, transactions, and units, helping to ensure transparency and promote user engagement.
- Direct blockchain interactions: Our integration with the blockchain's native RPC client, developed by the Alphabill team, enables the implementation of wallet balance checks and network status queries, providing actual, secure, and reliable data.

Data Handling Across Endpoints:

- Data Format: Information from the endpoints is output in JSON format, enhancing the ease of integration with other systems and services.
- Encryption and Encoding: Text information received from the backend is encrypted and sent as []byte arrays. In the frontend, this encrypted data needs to be converted from text to Base64 and then to Bytes to be usable. This process is crucial for maintaining the security and integrity of data as it moves from the backend to the frontend, ensuring that sensitive information remains protected from unauthorized access.

Endpoint Testing: Pre-Swagger Testing: Before integrating Swagger, endpoints were manually tested under various scenarios through Postman, ensuring their robustness before documenting and publishing them through Swagger Example of API endpoints set up using Gorilla Mux: Appendix B
Detailed Example of an Endpoint Documentation Using Swagger: Appendix C
By implementing these practices, the backend represents a secure, scalable, and user-friendly interface that aligns with modern web standards and security practices. The use of JSON and encrypted byte arrays for data transfer, alongside the robust documentation provided by Swagger, creates a transparent and efficient pathway for accessing blockchain data, supporting a wide range of user requests from simple queries to complex transactions.

3.3.6 Performance and Efficiency

Our backend design prioritizes performance and efficiency, leveraging Go's performance characteristics to effectively handle processes and respond quickly to real-time data processing and API requests. BoltDB's architecture also enables fast read and write operations, essential for handling high-frequency data updates common in blockchain environments.

3.3.7 Testing and Quality Assurance

Testing is a crucial component of the development process, ensuring that all backend functions perform as expected under various conditions. The backend uses unit testing frameworks such as the Go testing package and Testify library for assertions and mock behaviors.

The testing process involves writing test scripts that simulate both successful and error scenarios using mock data after implementing a specific method.

Example of a test case for retrieving block information: Appendix D

This test ensures the API correctly handles requests for block information, validating both the server response and the accuracy of the data returned. It is part of a test suite that verifies functionality across all endpoints, ensuring the backend maintains high reliability.

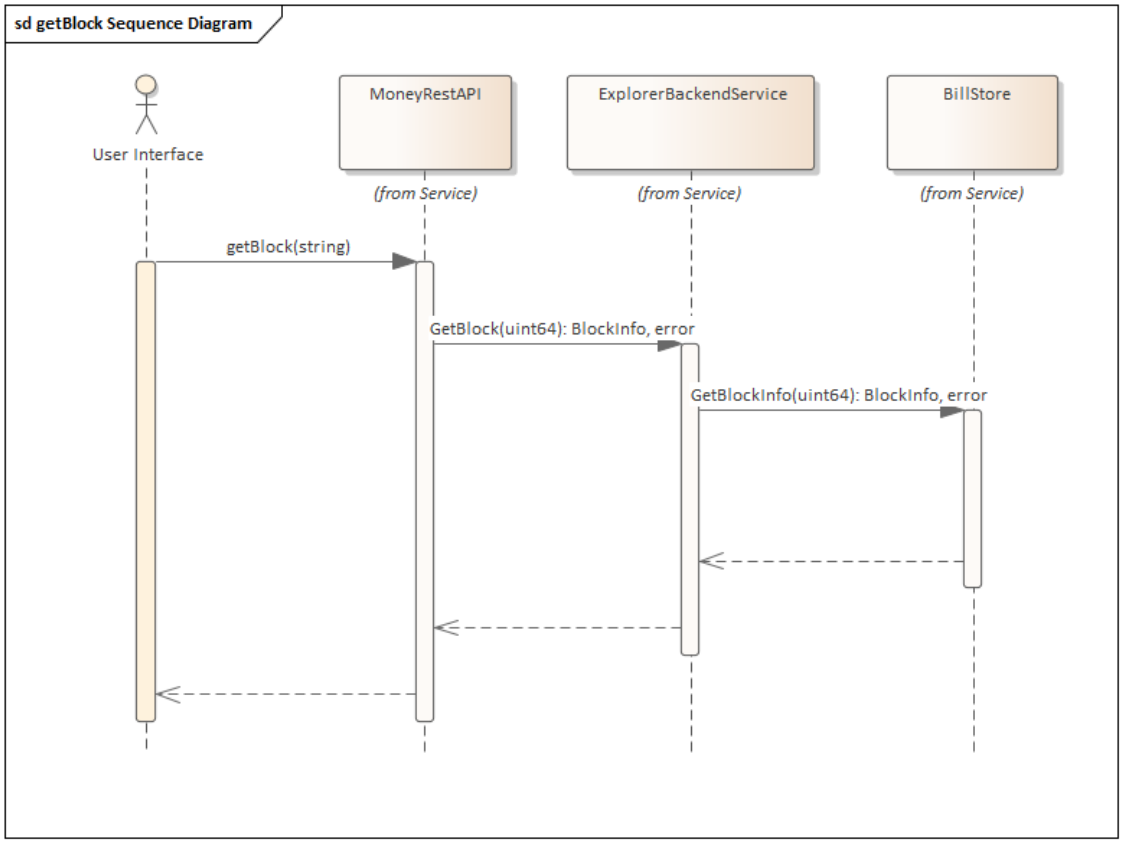

Figure 10. Sequence Diagram for getBlock Method.

The sequence diagram provided in Figure 10 describes the sequence of interactions between the UI, MoneyRestAPI, ExplorerBackend, and BillStore during the execution of the getBlock method. It highlights the flow of data and the responses at each stage, providing a clear picture of the process involved in retrieving block information from the backend.

3.4 Frontend Design and Implementation

3.4.1 Design Process

The frontend development for the Alphabill block explorer began with an extensive review of the existing design elements of the official Alphabill website and wallet interface. This initial review helped to maintain brand consistency and usability standards. The team utilized Figma to sketch the initial design concepts, selecting appropriate fonts, colors, and layout structures to create a cohesive and user-friendly interface.
Figma Mockups: Detailed mockups were created in Figma to outline the visual hierarchy and components of the home page, including the background, navigation bar, footer, search bar, logo, and headings as provided in Figure 11. This step ensured all design elements were aligned with the user's needs and expectations before development began.

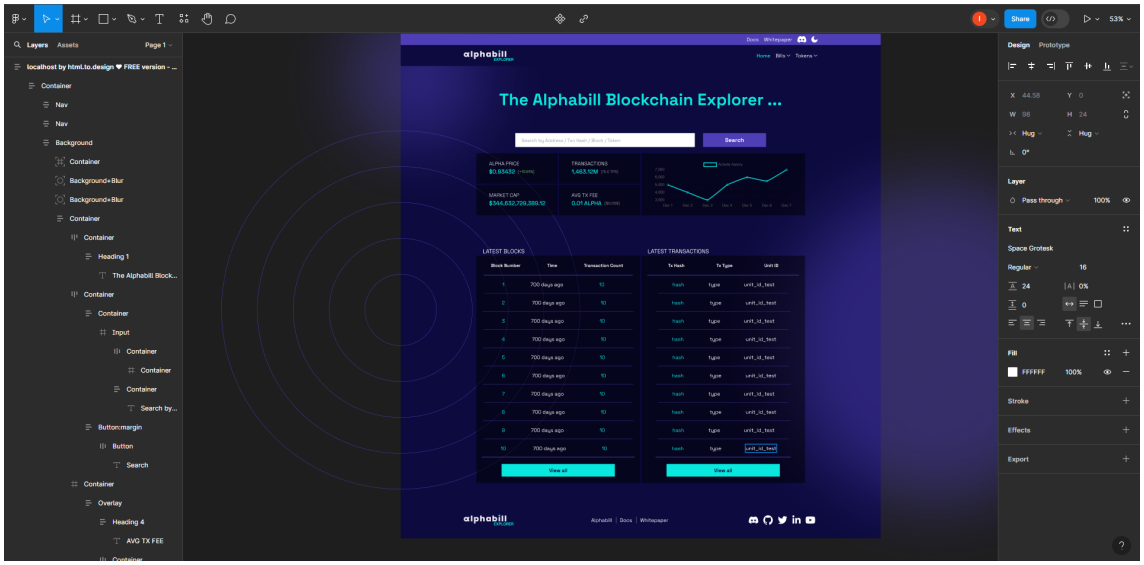

Figure 11. Figma mockup of project's visualization.

3.4.2 Project Initialization

The project was initiated with a modern tech stack specified by Alphabill's team to optimize performance and development efficiency:

- Vite: Chosen for its fast build times and out-of-the-box features for modern web development like HMR [18].
- React: Used for its component-driven architecture, enhancing the maintainability and reusability of code [19].

- TypeScript: Selected for its strong typing features, reducing potential bugs and improving code quality [20].
- ESLint: Ensures code quality and consistency throughout the project, enforcing a uniform coding standard among developers [21].
- TailwindCSS: Chosen for its utility-first approach, facilitating rapid UI development without sacrificing the maintainability of style sheets [22].
- Axios and TanStack Query: Axios was integrated with TanStack Query for fetching backend data, combining Axios's promise-based HTTP capabilities with TanStack Query's powerful data synchronization and caching features [23, 24].
- TanStack Table: TanStack Table was integrated to create a general table component with a unified design, headers, and pagination for displaying tabular data such as blocks and transactions [25].
- React Router: Implemented for handling in-app routing efficiently, ensuring that users can navigate through the application's different views seamlessly [26].
- Chart.js: Used for data visualization helping users more quickly understand the information [27].

These technologies were chosen to ensure robustness, scalability, and ease of development. TailwindCSS was immediately configured to include the necessary fonts and colors defined in the Figma designs to maintain visual consistency.

3.4.3 Directory Structure

A Feature-Sliced Design approach was adopted to organize the codebase effectively. This enhances scalability and maintainability by segregating the codebase into distinct layers based on feature functionality.

3.4.4 Responsive Design

The development followed a desktop-first approach, initially focusing on wide-screen layouts before adapting designs for mobile devices. This strategy ensures an optimal user experience across all device types.

3.4.5 Main User Interface Components

The UI is built with reusable UI components that can be integrated into the different parts of the application. This modular approach aligns with the feature slice architecture,

making the codebase easier to maintain and update. Here's a look at the key UI components:

- Navigation bar - This essential element sits at the top of the screen, providing users with navigation through the different sections of the application.
  Figures 12 and 13 provided below describe the desktop and mobile views of the navigation bar component.

Figure 12. Desktop view of Alphabill Explorer's navigation bar component.

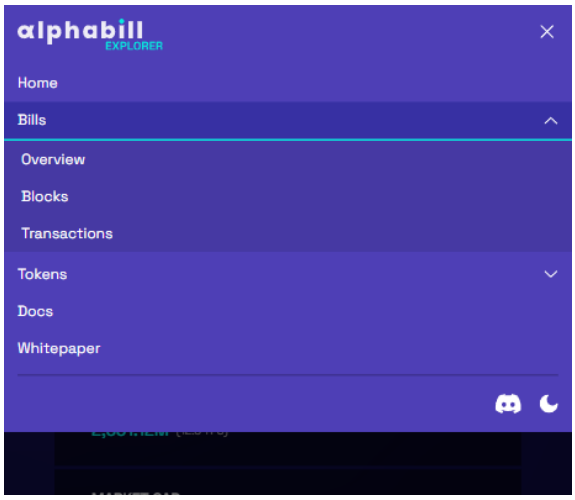

Figure 13. Mobile view of opened Alphabill Explorer's navigation bar component.

- Footer - Located at the bottom of the page, the footer consists of secondary links that complement the overall design. This includes links to the Alphabill's official website, whitepaper, and social media.
  Figures 14 and 15 provided below describe the desktop and mobile views of footer component.

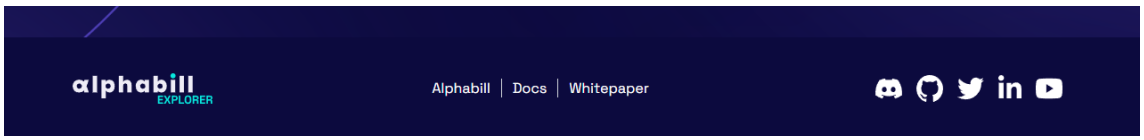

Figure 14. Desktop view of Alphabill Explorer's footer component.

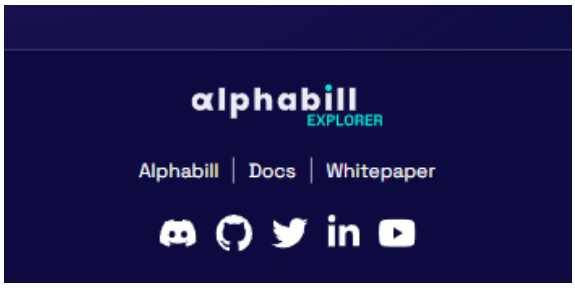

Figure 15. Mobile view of opened Alphabill Explorer's footer component.

- Background with Animations - The home page background features subtle animations to enhance visual appeal without detracting from the user experience.
- Search Bar - A search bar empowers users to find specific blocks, transactions, or addresses quickly and efficiently by providing the block number, transaction hash or public key.
  Figure 16 provided below describes the view of Alphabill Explorer's search bar component.

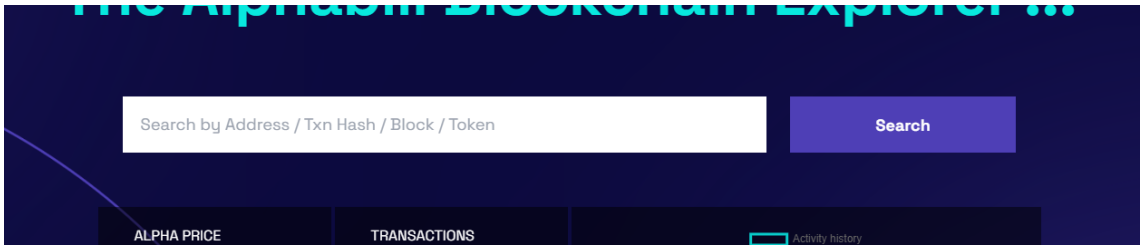

Figure 16. The view of Alphabill Explorer's search bar component.

- Block and Transaction Tables - Specific components for blocks and transactions were developed, including "mini" tables for displaying summarized data and detailed components for individual records.
  Figure 17 provided below describes the view of Alphabill Explorer's table component.

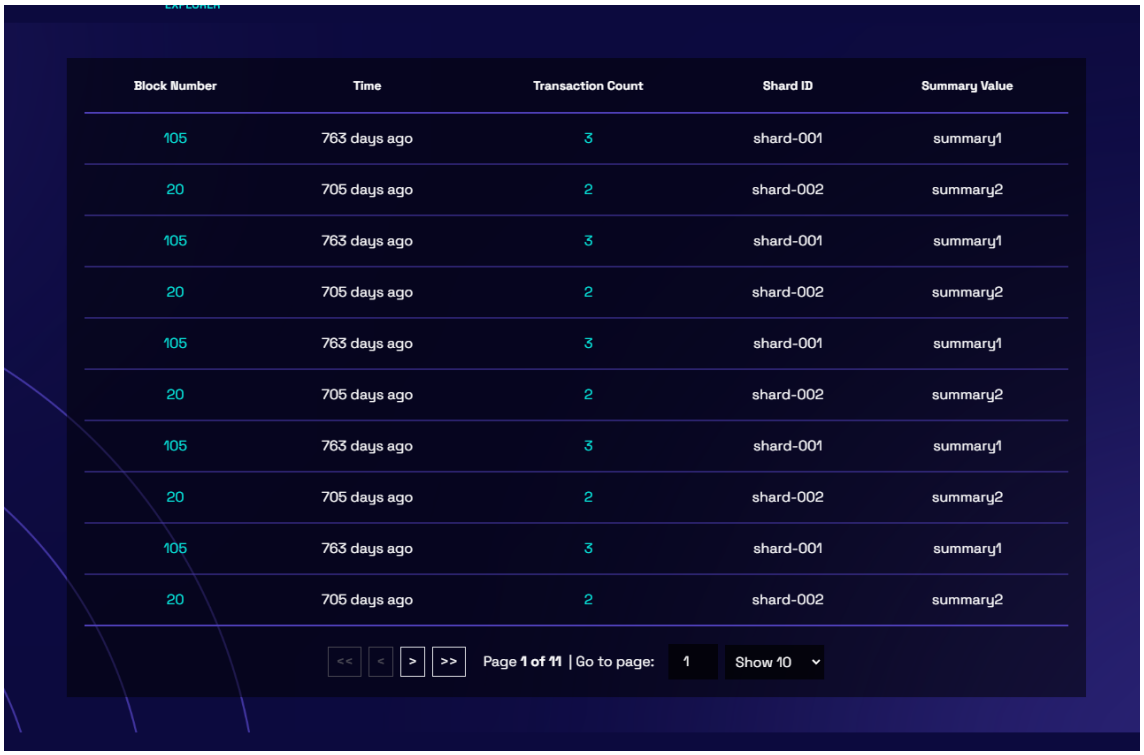

Figure 17. The view of Alphabill Explorer's table component.

- Network state and metrics - A component represents the overall health and key metrics of the network, providing users with an overview of performance and activity.
  Figure 18 provided below describes the view of Alphabill Explorer's state and metrics component.

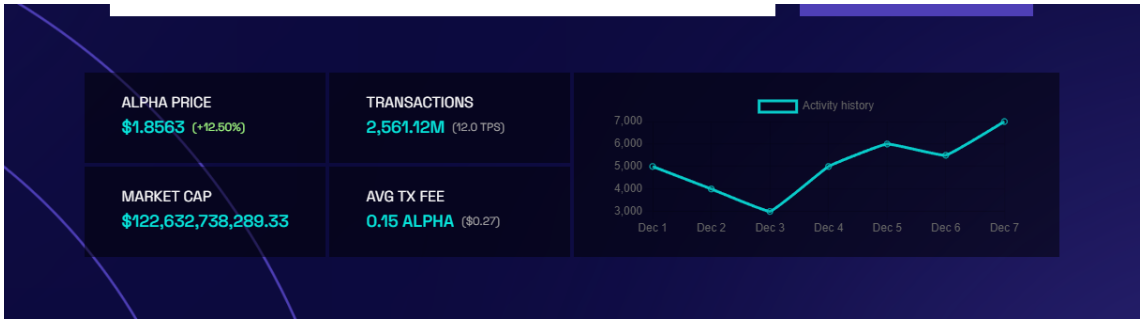

Figure 18. The view of Alphabill Explorer's network state and metrics component.

- Transaction details - Component was designed to display detailed information about the transaction selected by the user.
- Block details - Component was designed to display the detailed information about the block selected by the user. Figure 19 provided below describes the view of Alphabill Explorer's block details component.

Figure 19. The view of Alphabill Explorer's block details component.

3.4.6 Routing and page components

Alphabill explorer leverages a single-page application (SPA) architecture. This means the application updates content dynamically without full page reloads, providing a smoother and more responsive user experience compared to traditional websites.
Main Layout: Component represents the application's base layout, displaying the navigation bar, footer, and animated background throughout the app. Within this layout, the content area dynamically renders the appropriate page component based on the chosen route.
The initial routing setup established core pages with corresponding components:

- Home - Component represents a main landing page featuring a search bar, network health metrics, and summarized block and transaction data.
- Bills - Component represents a page designed to provide an overview of metrics, blocks, and transactions related to a specific "money partition".
- Blocks - Component represents a page showcasing a table of all blocks within the money partition.
- Transaction - Component represents a page displaying a list of transactions related to the money partition, including essential details for each transaction.
- Block details - Component represents a page with the details of the certain block selected by the user. Page contains all necessary information about the chosen block, that allows user to analyze the data.
- Transaction details - Component represents a page with the details of the certain transaction selected by the user. Page contains all necessary information about the

chosen transaction, that allows user to analyze the data.

- 404 not found - Component represents a page displayed when users attempt to access an invalid route.
- Coming soon - Component represents a page displayed when user requests content is not implemented yet and will be available in the future.

3.4.7 Integration with Backend

Once backend endpoints began to be available, the frontend integrated these using environment configurations stored in env.localhost. The fetching mechanisms involve TanStack Query calling Axios, with responses being processed by a transformObject function to handle data encoding and decoding, such as converting Base64 to bytes and handling BigInt types, ensuring accurate data representation in the frontend.

Advanced Features and Final Touches

- Theme Switching - To enhance user experience, theme-switching capabilities were added, allowing users to choose between light and dark modes.
- Charts - Components displaying charts (using Chart.js) were added to provide users with insightful data visualizations.

3.5 Development Tools and Practices

For the development of both the frontend and backend of the Alphabill block explorer, was utilized a streamlined set of tools and practices to ensure consistency, quality, and efficiency throughout the project lifecycle.

3.5.1 Version Control

GitHub was chosen as the primary tool for version control. It facilitated collaborative coding, feature branching, code reviews, and the management of changes throughout the development process. The use of GitHub ensured that all team members could work simultaneously on different features without disrupting the main development flow, while also maintaining a historical record of changes and decisions.

3.5.2 IDE

Visual Studio Code (VSCode) was the preferred IDE for both frontend and backend development. VSCode's rich ecosystem of extensions, including support for Go, TypeScript, and React, provided the developers with powerful tools for code editing, navigation, and debugging, directly within the IDE. It's lightweight nature and customizability made it an ideal choice for rapid development cycles.

3.5.3 Code Style and Formatting

To maintain a consistent code style and ensure that the codebase remained clean and readable, Prettier was integrated into the development workflow. Prettier is a code formatter that supports many languages and integrates well with VSCode. It automatically formats the code on save, aligning with predefined style guidelines, which helped in reducing the time spent on styling discussions during code reviews and increasing overall productivity. By leveraging these tools, the development team was able to focus more on building features and less on the managing project configurations and inconsistencies in coding styles. This chapter underscores the importance of choosing the right tools to support a development environment while ensuring high standards of the code quality and collaboration.

4. Evaluation of approach

4.1 Justification for technical implementation

4.1.1 Requirements

Frontend development

For the frontend, requirements were chosen to ensure high performance, adaptability and ease of use:

- **UI performance**
  Utilizing Vite and React with TypeScript support provides instant module reloading and optimized project bundling, which is crucial for projects with a large volume of interactive elements. React enabled the creation of dynamic user interfaces with high performance, thanks to efficient DOM updates.
- **Responsible UI**
  Utilizing TailwindCSS for managing styles.Rationale simplifies the creation of responsive designs through utility classes, enabling quick adaptation of the interface to any screen sizes and devices.
- **User friendly UI**
  Prototyping in Figma enables the creation of detailed interactive prototypes, facilitating the visualization of the user interface before development begins and gathering feedback from potential users.
- **REST API**
  Utilizing Axios for HTTP requests and TanStack Query for managing data state provides convenient interaction with REST API, while TanStack Query assists in managing, caching, and synchronizing data, enhancing performance and improving user experience by reducing data loading and updating time.

Backend development

For the backend, the requirements were reliability, scalability, and performance.

- **Data reliability**
  Utilizing BoltDB provides high performance and reliability for storing immutable blockchain data, thanks to its simplicity and efficiency in handling large volumes of

data

- **Scalability**
  Implementation of microservices architecture in Go allows scaling individual components of the system independently, providing flexibility in resource management and performance optimization as the load increases.
- **Performance**
  GO for server-side development and Gorilla Mux for routing offers built-in support for concurrency and efficient resource management, which is crucial for high-load systems. Gorilla Mux provides powerful routing capabilities, enhancing performance in handling a large number of requests.

4.1.2 Architecture

Frontend development

For the frontend, the architecture chosen was SPA based on React. The main reasons for choosing this architecture include:

- **Meaningful and relevant UX**
  SPA allows for loading all necessary JavaScript, HTML, and CSS once and then dynamically updating content on the page without full page reloads. This provides instant response to user actions.
- **Convenience of development and maintenance**
  React enables the separation of the interface into independent, reusable components. This facilitates the development, testing, and maintenance of code.
- **Scalability**
  Prototyping in Figma enables the creation of detailed interactive prototypes, facilitating the visualization of the user interface before development begins and gathering feedback from potential users.
- **REST API**
  SPA can easily scale by adding new components and features without the need for significant changes to existing code.

Alternatives

- **MPA**
  MPA provides better SEO optimization and simpler analytics, as each page has its own URL. However it brings complicated development and maintenance due to the necessity of page reloading and resource re-fetching, which increases delays and

reduces performance.

- **SSR**
  SSR provides improved performance for users with slow internet connections and better SEO.However it brings more complicated infrastructure and development, especially when synchronizing client-side and server-side logic. Examples of technologies: Next.js.

Backend development

The backend architecture is based on microservices using Golang and BoltDB. The main reasons for choosing this architecture include:

- **Modularity and manageability**
  Microservices architecture allows breaking down the system into independent services that can be developed, deployed, and scaled independently of each other.
- **Performance**
  Golang offers high performance due to its compilation to machine code and efficient management of concurrency.
- **Ease of storing data**
  BoltDB, as an embedded database, offers a simple interface and high performance for storing immutable blockchain data.

Alternatives

- **Monolithic architecture**
  Simplification of the development and deployment process, as all logic and functionality are contained within a single application. However it increases complexity of scaling and maintaining the system, especially with the growth of features and data volume.
- **SQL**
  SQL allows the execution of complex queries and transactions. However it brings less efficient management of large volumes of immutable data, high performance overhead with frequent write operations.
- **NoSQL**
  NoSQL brings flexibility in managing various data types and high performance scalability. However there is more complicated setup and management, especially in distributed systems requiring data consistency assurance.

4.1.3 Design

Frontend development

For the frontend was chosen design that ensures high performance and usability. The main reasons for choosing such a design were:

- **Component-Based Architecture**
  Pattern: Component-Based Architecture
  React provides a component-based approach, allowing the interface to be divided into independent and reusable blocks. This simplifies the development, testing, and maintenance of the code.
- **Logic Separation**
  Pattern: MVC or MVVM patterns
  The approach of separating application logic and data presentation is often used in React. This improves code readability and maintainability.
- **Responsive design**
  Pattern: Responsive Design
  Utilizing TailwindCSS utility classes to create adaptive interfaces that adapt to different screen sizes.
- **Feature-Sliced Design**
  Pattern: Feature-Sliced Design
  This approach structures the project into functional modules, each of which includes all necessary elements (components, state, styles, API calls) to implement specific functionality. This improves code modularity and manageability, simplifies scaling and implementing new features, and enhances coordination between developers.
- **React Hooks and Context API**
  Pattern: React Hooks and Context API
  Utilizing React hooks and Context API for managing application state and passing data between components.

Backend development

For the backend, a design was chosen to ensure reliability and scalability. The main reasons for choosing such a design were:

- **Microservices Architecture**
  Pattern: Microservices Architecture
  Splitting the application into small independent services, each responsible for its own functionality. This facilitates scalability and system support.

- **Request Processing**
  Pattern: REST API
  Using REST API for communication between frontend and backend, providing a standardized approach to data transmission. This brings ease of integration and scalability, standardization of interactions.
- **Data Storage**
  Pattern: Key-Value Store
  Using BoltDB to store blockchain data in key-value format, providing fast data read and write operations, helps to achieve high performance and reliability in data storage.
- **Data Processing**
  Pattern: Data Processing Pipeline
  Processing blockchain data through multiple stages (decoding, transformation, storage), improving code structure and manageability, brings structured and scalable data processing.

4.1.4 Coding standards

Rules for writing code and code metrics play a key role in maintaining code quality, simplifying its maintenance, and improving collaboration within the team. In the Alphabill Explorer project, the following rules and metrics were chosen:

Frontend development

- **Consistent code style**
  Tool: Prettier
  Prettier provides automatic code formatting, which helps maintain consistency in the codebase, improves code readability and reduces code review time.
- **Correct syntax**
  Tool: ESLint
  ESLint detects syntax errors and code style violations, helping to identify potential issues at an early stage of development, reducing the number of errors and improving code quality.
- **Typed coding language**
  Tool: TypeScript
  TypeScript adds static typing to JavaScript, helping to prevent type-related errors while compiling and making the code more reliable.

Backend development

- **Consistent code style**
  Tool: Go fmt
  Go fmt automatically formats code according to the Go standards, helping to maintain consistency and readability.
- **Correct syntax**
  Tool: Golint
  Golint detects syntax errors and coding style violations in Go, helping to identify potential issues at an early stage of development, reducing the number of errors and improving code quality.
- **Tested code**
  Tool: Go testing package and Testify
  Regular testing helps detect and eliminate errors at early stages of development. Using Testify simplifies writing tests with convenient assertions and functions.

These rules were chosen to ensure high code quality, improve team collaboration, and maintain system performance, which are crucial for the successful operation of the blockchain explorer.

4.2 Comparison with Existing Popular Blockchain Explorers

4.2.1 Etherscan

Etherscan is one of the most popular blockchain explorers for the Ethereum network. It provides users with the ability to view information about transactions, blocks, addresses, smart contracts, and other data related to the Ethereum blockchain [28].
Architecture of Etherscan includes the use of MPA to improve SEO. This allows each page to have a unique URL, making it easier for search engines to index and increasing the visibility of the resource on the internet.
Additionally, Etherscan uses SSR, which enables faster page loading, improving the site's performance and user experience. The data storage is implemented using a combination of various databases and caching technologies, ensuring high performance and scalability of the system [29]. Figure 20 provided below represents UI of the Etherscan's home page.

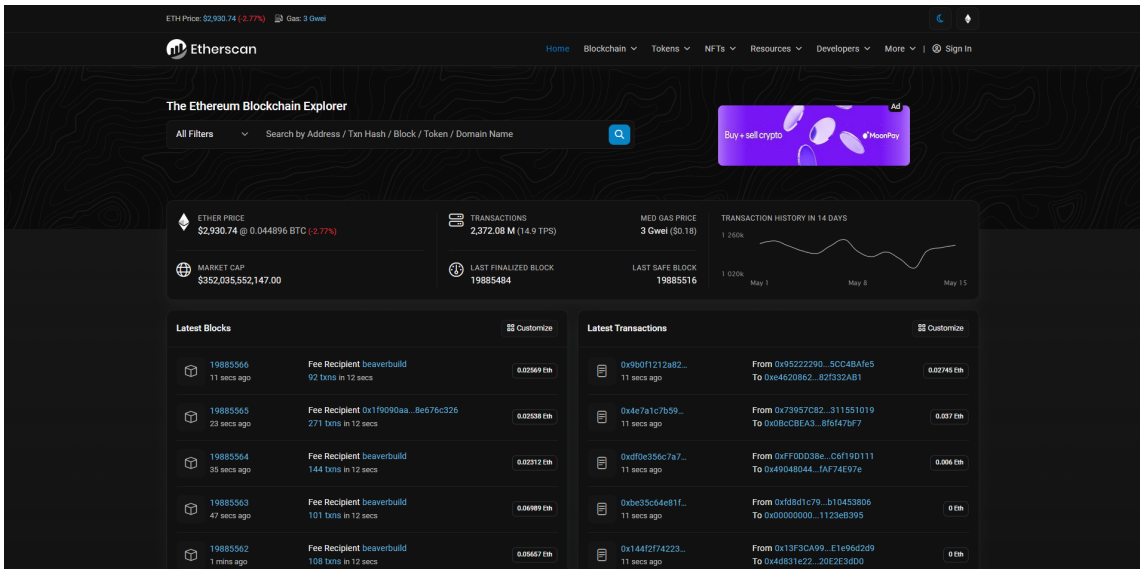

Figure 20. Etherscan's home page.

4.2.2 Blockchain.com

Blockchain.com Explorer is one of the oldest and most popular blockchain explorers. It supports multiple blockchains, including Bitcoin, Ethereum, and Bitcoin Cash [30]. Blockchain.com Explorer uses an MPA architecture to improve SEO and navigation ease. SSR also contributes to fast page loading, which enhances user experience and interaction with the site. For processing and storing data, Blockchain.com Explorer utilizes various databases, including both relational and non-relational databases. This enables the system to handle large volumes of data and ensure high performance. Figure 21 provided below represents UI of the Etherscan's home page.

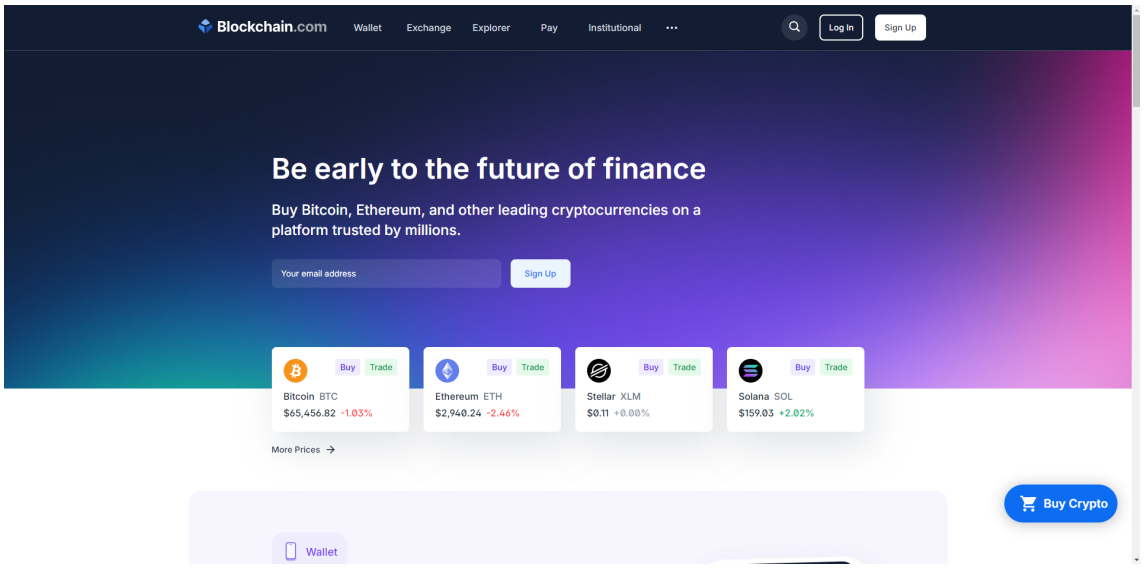

Figure 21. Blockchain.com's home page.

4.2.3 BitInfoCharts

BitInfoCharts provides statistical data and charts for various cryptocurrencies, including Bitcoin, Ethereum, Litecoin, and others. It also utilizes the MPA architecture, which helps improve SEO and provides static URLs for each page, making them easier to access for search engines. SSR in this case enables faster page loading, which has a positive impact on the user experience [31]. For storing and analyzing large volumes of data, BitInfoCharts utilizes relational databases, which ensures reliability and efficiency in working with large datasets and executing complex analytical queries. Figure 22 provided below represents UI of the Etherscan's home page.

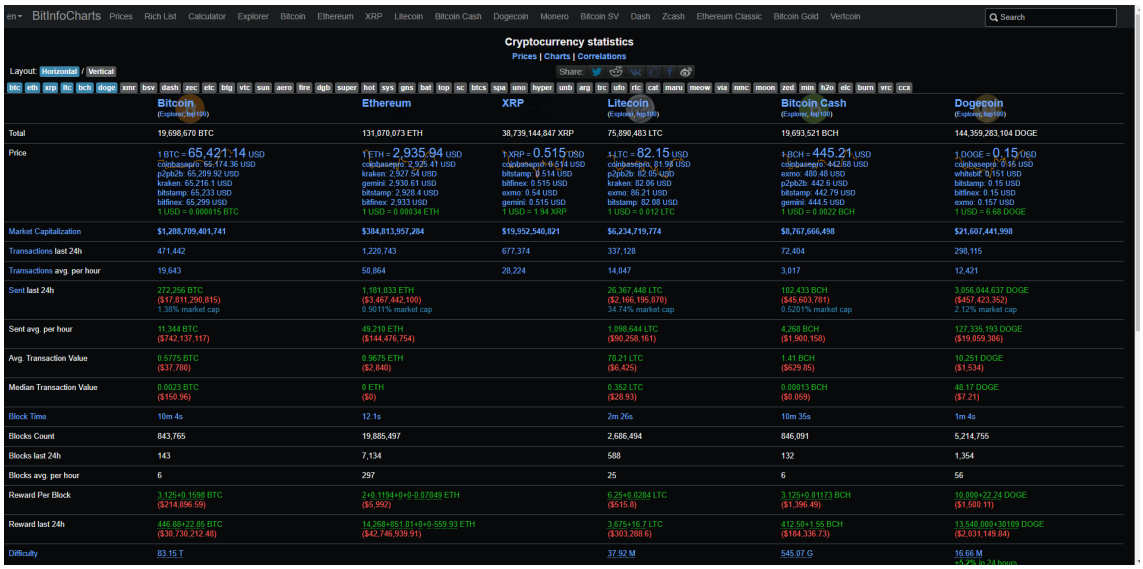

Figure 22. BitInfoCharts's home page.

4.2.4 Alphabill Explorer

Alphabill Explorer differs from the previously mentioned explorers. It is built utilizing the SPA architecture on a React basis, which enables the creation of a more dynamic and interactive interface for users. The microservice architecture ensures high performance and scalability. For storing data, Alphabill Explorer uses BoltDB, a high-performance and lightweight database optimized for fast operations on keys and values. This enables fast data access and processing, which is particularly important for ensuring meaningful and relevant experience [14, 13]. Figure 23 provided below represents UI of the Etherscan's home page.

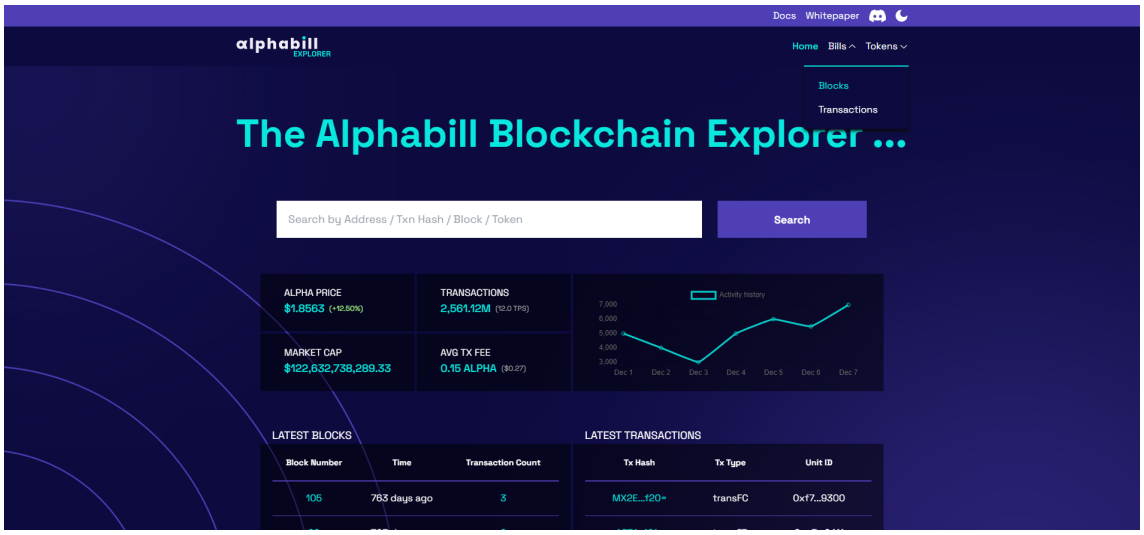

Figure 23. Alphabill Explorer's home page.

4.3 Versions controls

The project management process and completed tasks were recorded using the version control system "GitHub". The project was divided into two separate repositories, one for frontend[1] development and another for backend[2] development. Each team member followed certain rules and procedures to ensure orderly and transparent work.

4.3.1 Git commits and branch management

Before implementing new functionality, developers created separate branches in the repository, allowing them to work independently without affecting the main branch.

- **Consistent code style**
  Each developer created a branch with a name reflecting the essence of the new functionality or fix (for example, "block" or "table").
- **Development and commits**
  In the process of working on the functionality, developers made regular commits, documenting incoming changes or fixed parts. Examples of commit messages: "NavBar PC Dropdown added", "Fix getBillsByPubKey", "Add test TestGetBlock_-latest_Success".
- **Code review and verification**
  After the development process was completed, the branch was sent to a code review

[1]https://github.com/alphabill-org/alphabill-explorer
[2]https://github.com/alphabill-org/alphabill-explorer-backend

done by our mentor. Code review consisted of reviewing the commits, writing comments and making suggestions for the code improvements. If necessary, changes and improvements were made according to the feedback provided during the code review process.

- **Branch merge**
  Once the code review was completed and all changes were approved, the new feature branch was merged into the main "main" branch, ensuring a smooth integration of the new functionality without any conflicts.
- **Local environment and future plans**
  The project is currently running locally, providing a secure environment for testing and debugging. Future plans include creating a testnet branch, which will serve as a testing ground for new features before they are merged into the main branch.

4.4 Evaluation of the project development process

4.4.1 Project management and development

The development of a blockchain explorer for Alphabill began in October 2023 and is ongoing. However, progress was temporarily halted in January 2024 to accommodate changes in the blockchain architecture. The development was resumed by March 2024.

Planning and initialization (October - November 2023)

The team gathered to determine the main goals and requirements of the project. A primary plan was developed, including the roles and tasks of the participants. In October was also discussed and selected the technology stack and architecture of the future application. It was decided to use React and Vite for the frontend and Golang with BoltDB for the backend. November passed, setting up a WSL-based development environment,initializing the project on GitHub,installing ESLint and Prettier tools.

Development and integration (December 2023 - May 2024)

From December to January, the development team actively worked on creating initial components and APIs for the Alphabill blockchain explorer. However, in January, progress was halted to accommodate changes in the blockchain, which required a revision of the architecture and algorithms. After the revision, the team resumed development, adopting a microservice structure to enhance scalability. The project then shifted focus to integrating the updated frontend and backend, optimizing performance, and conducting initial testing, with work commencing in early April.

Testing and optimization (May 2024)

To ensure the quality and reliability of the Alphabill blockchain explorer, the development team employed various testing tools. For the frontend, Jest, React Testing Library, and Cypress were utilized for unit testing and integration testing. Meanwhile, the Go testing package and Verify were used for the backend. To evaluate the system's performance under heavy loads, Apache JMeter and Locust were used for load testing.

Working process

The project management process included regular team meetings, the use of Agile methodologies to flexibly respond to changes and ensure close interaction between the front-end and back-end developers. This made it possible to successfully adapt to changes in requirements and complete the current stages of the project with high quality. In the next stages, it is planned to continue the development of the system, expanding its functionality and providing scalability to meet the needs of users in the future.

4.4.2 Evaluation of successes and faced difficulties during the development

Successes

Using our existing experience in developing web applications, it didn't take much time to initialize the project on the frontend, importing all the necessary tools and libraries.
Effective communication and the enthusiasm of all team members towards achieving the goal accelerated the development process and eliminated misunderstandings within the team. The project leader quickly responded to emerging questions and provided valuable advices on implementing solutions. Additionally, regular meetings were held to discuss the progress made, problems appeared through the development process and set the goals for the next sprint.
It's also worth noting the great teamwork of all team members. Team members always helped each other as needed to resolve issues that were appearing during the development process, and their ability to coordinate tasks and responsibilities effectively ensured a streamlined development process.

Challenges

Resuming the project after being on hold proved to be a challenging task. Changes in the blockchain architecture caused significant changes in the backend, including classes, structures and functional components.

Refactoring the architecture took significantly longer than expected. It required a thorough review and restructuring of many classes and functions, which slowed down the overall development process.
Also changes in endpoint requirements led to the need to review and standardize data formats between the frontend and backend. This included a complete overhaul of all data types and setting up new endpoints. The process of setting up and testing new endpoints was time-consuming and required more time than expected, which slowed down the overall development process.

4.4.3 Overall Assessment of the Project Execution Process

The overall project execution process

The project to develop a blockchain explorer for Alphabill was carried out within a well-organized plan, which included phases of planning, development, testing, and preparation for deployment. Overall, the project execution process can be characterized as successful, despite the challenges and delays that appeared.

Team leader

Team leader, who was an advisor from both the company and university, actively participated in the project, providing us with the feedback and support. This helped to resolve emerging issues promptly and maintain high team motivation among the team members. In addition to this, regular meetings and discussions with the team leader contributed to clear coordination and effective task distribution.

Frontend developer

The frontend developer actively participated in the project, implementing UI components, integrating with APIs, and implementing Feature-Sliced Design. The work was completed with high quality and attention to detail. Regular commits and close collaboration with the teammate contributed to the project's success.

Backend developer

The backend developer was responsible for updating classic structures and setting up new endpoints, as well as refactoring types in the frontend. Despite the complexities related to the transition to a new architecture, the developer successfully completed tasks, ensuring system reliability and performance.

Teamwork

Open and regular communication between team members facilitated the prompt resolution of problems and increased work efficiency. In addition to this team members supported each other, which helped to maintain a high level of motivation and achieve set goals.

Notes and suggestions

- **Planning**
  In the future, it would be beneficial to pay more attention to planning and determining potential risks to minimize the impact of unexpected changes on the project.
- **Testing**
  In the future, it would be beneficial to increase the volume and variety of testing at early stages of development to promptly identify and eliminate errors.
- **Documentation**
  More detailed and structured documentation at each stage of the project will help avoid misunderstandings and simplify the process of integrating new team members.
- **Training and development**
  Regular training sessions and seminars on new technologies and methodologies will help the team stay up-to-date with the latest trends and improve their qualifications.

Overall, the project to develop a blockchain explorer for Alphabill was successful due to clear planning, active support from the project leader, and great teamwork. Despite the challenges that appeared, the project is being completed with high quality.

4.5 Outlining authors' contributions

Both authors demonstrated high levels of professionalism and responsibility, and their contributions were equal, reflecting their overall contribution to the successful completion of the project.

4.5.1 Frontend development

Igor Mahlinovski
Igor actively participated in developing the design, layout, and UI components development, integration with API, and implementation of Feature-Sliced Design. His work was done with high quality and attention to detail, contributing to the successful completion of the project.

4.5.2 Backend developement

Alexander Khrushkov
Alexander made a significant contribution to restructuring the back-end architecture to a microservices-based one, updating class structures, setting up new endpoints and integrating data types into the frontend. Despite the complexities associated with transitioning to a new architecture, Alexander successfully completed his tasks, ensuring system reliability and performance.

5. Conclusion

5.1 Summary

The aim in creating an explorer for the Alphabill blockchain was to increase the transparency of the network. The goal of the project was to develop a reliable and scalable blockchain explorer tool for efficiently processing and displaying blockchain data. For the implementation, modern technologies were chosen, such as microservice architecture, Go programming language and BoltDB for the backend, React, Vite and TypeScript for the frontend. The development process included phases of planning, design, implementation, integration, testing and documentation.

The main achievement was the successful implementation of the functionality for the interaction with the money partition of the Alphabill blockchain, allowing clients to track transactions and blocks. The analysis of the work showed that the selected technologies and architectural solutions provide high performance and flexibility of the system, which is a good basis for further expansion of functionality.

5.2 Future work

Within the scope of this work, only a part of the functionality related to the Money partition has been implemented. The current version does not provide users with detailed information about what exactly happened to the unit during a transaction. In the future, it is planned to:

- Add other partitions, such as User Tokens and EVM, each with own shards, blocks and transactions.
- Implement detailed flow for all types of units, allowing users to see the complete history.
- Improve the UI and UX, including data visualization and integration of additional analytical tools.
- Optimize performance and scalability of the system to support growing data volume.
- Conduct additional testing and implement features to enhance the security and reliability of the blockchain explorer.

References

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Whitepaper. Accessed: 24-04-2024. bitcoin.org, 2008. URL: https://bitcoin.org/bitcoin.pdf.

[2] Joseph Abadi and Markus Brunnermeier. *Blockchain economics*. Tech. rep. National Bureau of Economic Research, 2018.

[3] *Understanding Peer-to-Peer Architecture*. [Accessed: 24-04-2024]. URL: https://systemdesignschool.io/blog/peer-to-peer-architecture.

[4] Simanta Shekhar Sarmah. "Understanding blockchain technology". In: *Computer Science and Engineering* 8.2 (2018), pp. 23–29.

[5] Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[6] Ingolf Gunnar Anton Pernice and Brett Scott. "Cryptocurrency". In: *Internet Policy Review, Glossary of decentralised technosocial systems* 10.2 (2021).

[7] Shubhani Aggarwal and Neeraj Kumar. "Cryptocurrencies". In: *Advances in Computers*. Vol. 121. Elsevier, 2021, pp. 227–266.

[8] *Altcoin Explained: Pros and Cons, Types, and Future*. [Accessed: 24-04-2024]. URL: https://www.investopedia.com/terms/a/altcoin.asp#:~:text=Pros%20and%20Cons%20of%20Altcoins&text=Altcoins%20are%20%22improved%20versions%22%20of,uses%2C%20such%20as%20Ethereum's%20ether..

[9] Douglas W Arner, Raphael Auer, and Jon Frost. "Stablecoins: risks, potential and regulation". In: (2020).

[10] Andrei-Dragos Popescu. "Non-fungible tokens (nft)–innovation beyond the craze". In: *5th International Conference on Innovation in Business, Economics and Marketing Research*. Vol. 32. 2021.

[11] Robert Werner, Sebastian Lawrenz, and Andreas Rausch. "Blockchain analysis tool of a cryptocurrency". In: *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*. 2020, pp. 80–84.

[12] Daniel Fernando Arevalo Espinel. "How Usable are Block Explorers for Attesting Agreements Registered on Blockchains?" MA thesis. NTNU, 2019.

[13] *Alphabill Whitepaper: Public Token Infrastructure*. Whitepaper. Accessed: 10-03-2024. alphabill.org, 2024. URL: https://docs.alphabill.org/files/ab-whitepaper-2024-v15.pdf.

[14] *Alphabill Documentation*. [Accessed: 10-03-2024]. URL: https://docs.alphabill.org/docs/welcome.

[15] Ahto Buldas et al. "A Unifying Theory of Electronic Money and Payment Systems". In: (July 2021). DOI: 10.36227/techrxiv.14994558.v3.

[16] Ahto Buldas et al. "An Ultra-Scalable Blockchain Platform for Universal Asset Tokenization: Design and Implementation". In: *IEEE Access* 10 (2022), pp. 77284–77322. DOI: 10.1109/ACCESS.2022.3192837.

[17] *The full Ubuntu experience, now available on Windows*. [Accessed: 24-04-2024]. URL: https://ubuntu.com/desktop/wsl.

[18] *Why Vite*. [Accessed: 24-04-2024]. URL: https://vitejs.dev/guide/why.html.

[19] *React - The library for web and native user interfaces*. [Accessed: 24-04-2024]. URL: https://react.dev/.

[20] *The TypeScript Handbook*. [Accessed: 24-04-2024]. URL: https://www.typescriptlang.org/docs/handbook/intro.html.

[21] *Find and fix problems in your code*. [Accessed: 24-04-2024]. URL: https://eslint.org/.

[22] *Core Concepts*. [Accessed: 24-04-2024]. URL: https://tailwindcss.com/docs/utility-first.

[23] *Getting Started*. [Accessed: 24-04-2024]. URL: https://axios-http.com/docs/intro.

[24] *Overview*. [Accessed: 24-04-2024]. URL: https://tanstack.com/query/latest/docs/framework/react/overview.

[25] *Overview*. [Accessed: 24-04-2024]. URL: https://tanstack.com/table/latest/docs/overview.

[26] *Overview*. [Accessed: 24-04-2024]. URL: https://reactrouter.com/en/main/start/overview.

[27] *Chart.js*. [Accessed: 24-04-2024]. URL: https://www.chartjs.org/docs/latest/.

[28] *What is Etherscan*. [Accessed: 24-04-2024]. URL: https://info.etherscan.com/what-is-etherscan/.

[29] Wren Chan and Aspen Olmsted. “Ethereum transaction graph analysis”. In: *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. 2017, pp. 498–500. DOI: 10.23919/ICITST.2017.8356459.

[30] *Relentlessly building the future of finance since 2011*. [Accessed: 24-04-2024]. URL: https://www.blockchain.com/about.

[31] *bitinfocharts*. [Accessed: 24-04-2024]. URL: https://bitinfocharts.com/.

Appendices

A. Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Igor Mahlinovski and I Alexander Khrushkov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Developing a block explorer for the next-generation blockchain: a tool to enhance the network transparency", supervised by Pavel Grigorenko
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. We are aware that the authors also retain the rights specified in clause 1 of the non-exclusive licence.
3. We confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

20.05.2024

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

B. Example of API endpoints

```
// version v1 router
apiV1 := apiRouter.PathPrefix("/v1").Subrouter()

// block
apiV1.HandleFunc("/blocks/{blockNumber}", api.getBlock).Methods("GET", "OPTIONS")
apiV1.HandleFunc("/blocks", api.getBlocks).Methods("GET", "OPTIONS")

// tx
apiV1.HandleFunc("/txs/{txHash}", api.getTx).Methods("GET", "OPTIONS")
apiV1.HandleFunc("/txs", api.getTxs).Methods("GET", "OPTIONS")
apiV1.HandleFunc("/blocks/{blockNumber}/txs",
    api.getBlockTxsByBlockNumber).Methods("GET", "OPTIONS")
apiV1.HandleFunc("/units/{unitID}/txs", api.getTxsByUnitID).Methods("GET",
    "OPTIONS")

// bill
apiV1.HandleFunc("/address/{pubKey}/bills", api.getBillsByPubKey).Methods("GET",
    "OPTIONS")
```

C. Example of API endpoint

```
// @Summary Retrieve a blockchain block by number, or the latest if unspecified
// @Description Retrieves a block by using the provided block number as a path
    parameter, or retrieves the latest block if no number is specified.
// @Tags Blocks
// @Accept json
// @Produce json
// @Param blockNumber path string false "Block number ('latest' or a specific
    number)"
// @Success 200 {object} api.BlockInfo "Block information successfully retrieved"
// @Failure 400 {object} string "Missing or invalid block number"
// @Failure 404 {object} string "No block found with the specified number"
// @Failure 500 {object} string "Internal server error, such as a failure to
    retrieve the block"
// @Router /blocks/{blockNumber} [get]
func (api *MoneyRestAPI) getBlock(w http.ResponseWriter, r *http.Request) {
    // Implementation of the endpoint
}
```

D. Example of a test case

```
func TestGetBlock_Success(t testing.T) {
r := mux.NewRouter()
 restapi := &MoneyRestAPI{Service: &MockExplorerBackendService{
  getBlockFunc: func(blockNumber uint64) (api.BlockInfo, error) {
    require.Equal(t, uint64(1), blockNumber)
    return &api.BlockInfo{TxHashes: []api.TxRecordHash{{0xFF}}}, nil
  },
}}
r.HandleFunc("/blocks/{blockNumber}", restapi.getBlock)
ts := httptest.NewServer(r)
defer ts.Close()

res, err := http.Get(fmt.Sprintf("%s/blocks/1", ts.URL))
require.NoError(t, err)

require.Equal(t, http.StatusOK, res.StatusCode)

body, err := io.ReadAll(res.Body)
require.NoError(t, err)

result := &api.BlockInfo{}
require.NoError(t, json.Unmarshal(body, result))
require.Equal(t, &api.BlockInfo{TxHashes: []api.TxRecordHash{{0xFF}}}, result)
}
```