Swinburne University of Technology

Ho Chi Minh Campus

Decentralised Trading System

Project Design Document

COS30049 - Computing Technology Innovation Project

Project Group 3TL:

1. Nguyen Do Nhat Nam - 104061616
2. Nguyen Quoc Thang - 104193360
3. Tran Thanh Minh - 103809048

March 2024

Word Count: 6729 words

Table of contents

1. Project background and introduction

1.1 Introduction

Decentralized Trading System (DTS) is a new and promising financial trading platform. DTS, which uses blockchain technology, removes intermediaries, empowering consumers directly and delivering several benefits like increased control, security, and wider accessibility.

The DTS system is built on a sophisticated backend system and blockchain technology. The backend system consists of a smart order matching engine that automatically matches buy and sell orders, an order book that records all open buy and sell orders to ensure market transparency and efficiency, and infrastructure. The advanced security layer helps to safeguard systems against network assaults and illegal access. Blockchain technology acts as a public ledger, permanently and immutably recording all transactions to provide transparency, prevent manipulation, and foster confidence in the system. Smart contracts perform transactions automatically in accordance with predefined circumstances, removing the need for manual intervention and human mistake.

The combination of a secure backend system with transparent, immutable blockchain technology enables DTS to build a safe, efficient, and user-centric trading environment.

Every day, the DTS system is being built and improved, with the goal of providing consumers with a novel and potentially viable financial transaction option. More information regarding the system will be released in the future. Encourage users to properly understand the system before engaging in transactions.

1.2 Project Background

The conventional financial system relies on centralized organizations, such as banks and brokers, to support trade activity. These middlemen handle user funds and transactions, adding a degree of trust and possibly control. Decentralized trading systems, also known as Decentralized Exchanges (DEXs), seek to disrupt the current economy by allowing users to trade directly with one another.

DEXs use blockchain technology to provide a transparent and immutable record of transactions. This eliminates dependency on trustworthy third parties and increases trust in the system.

Users on a DEX keep control of their funds throughout the trading process. This removes the possibility of exchange hacking or the abuse of deposited money.

DEXs are often permissionless, allowing anybody with an internet connection to trade. This promotes greater financial inclusion and lowers obstacles to access.

Decentralized trading systems are a fast expanding subset of the wider topic of Decentralized Finance. As technology advances, DEXs have the potential to transform the way financial markets work.

1.3 Motivation

As potential owners of a Decentralized Trading System (DTS), we may be motivated by a variety of considerations, including personal values, business objectives, and a desire to address specific challenges in the existing financial environment.

Personal values such as conviction in decentralization, love for financial inclusion, and desire for openness and security might inspire us to create a DTS. We may believe in the notion of decentralization and seek to build a system that empowers individuals while reducing reliance on centralized authority. We may be enthusiastic about developing an accessible and inclusive financial system that opens up chances for individuals who are now excluded from existing financial systems. We may also be motivated by a desire to build a more open and secure trading environment, which addresses concerns about manipulation and mismanagement in existing systems.

Business objectives like innovation and competitive advantage, cost efficiency and scalability, and satisfying market expectations can all drive us to develop a DTS. A DTS can be a pioneering endeavor in the financial technology field, with a distinct selling point and a competitive edge in the market. The possibility for lower operating expenses and the inherent scalability of blockchain technology might be appealing advantages, enabling for more efficient expansion and service delivery. We may also see an increase in demand for decentralized financial solutions among individuals and institutions, and we want to meet this changing market need.

Frustration with existing systems, as well as a desire to tackle specific problems, might drive us to develop a DTS to address the deficiencies of the current financial system. We may have had personal or professional situations that highlighted the limitations or holes in established financial systems, prompting us to create an alternate solution. We may also be motivated by a specific financial problem that we want to solve by establishing a Decentralized Trading System (DTS).

Finally, the reasons we are compelled to establish a DTS are special to us and our team. It is most likely a mix of these elements, and recognizing our own motivations might be critical for staying focused and motivated throughout the growth process.

1.4 Core function requirements

The decentralized trading system will provide users with safety and transparency during use. Below are some core functional requirements. My system provides users with the login vs signup function to provide users with the best and most comprehensive experience.

1. Signup allows new users to register for an account by providing first name, last name, username, email, password and phone number. There is integrated authentication to check validity and prevent duplicate accounts. clause
2. Login allows users to access the system if the user uses a username with a registered password. We also use the password hash mechanism to protect user information more strictly
3. My system also has transactions for users to make money transfers. Users can enter the recipient address with the recipient address and the amount of money they make the transaction, and update the balance after making the transaction.
4. In addition, our system also allows users to track and view information and transaction times so that users can easily manage their own transaction performance.

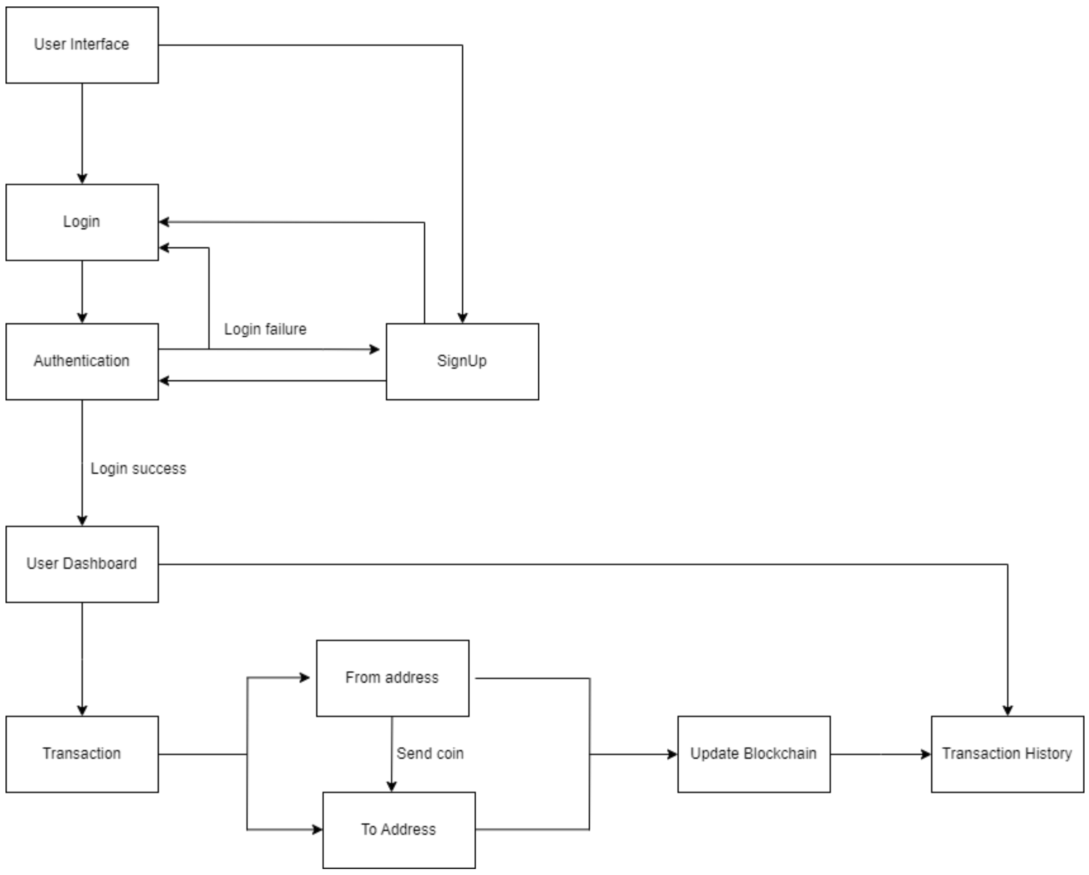


Image 1 : Diagram of core function requirement

2. Team introduction

The project team consists of three people, each of whom specializes in one of the project's domains, such as cybersecurity, software development, and software design:

Table 1: Team Responsibilities

Team members	Responsibilities
Tran Thanh Minh	Software development, Blockchain development
Nguyen Quoc Thang	Back-end development, Cybersecurity
Nguyen Do Nhat Nam	Software development, Front-end development

3. Project Description

This application is a decentralized money transfer system which will be implemented with technologies such as React, Django, Solidity to enable the users to securely send and receive money in a decentralized manner. This project will be executed in two major phases for software development. The first phase involves the development of frontend components, including the static web pages and the user interface design. The second phase focuses on the backend logic and components where the core functional requirements are implemented to handle decentralized money transfer functionalities.

4. Project requirement list

4.1 Functional Requirements

4.1.1 User Authentication and Registration:

- Users can create accounts by providing details
- Users can log in to the system using their credentials
- Passwords should be securely stored using appreciate encryption techniques

4.1.2 Wallet Creation and Management:

- Users have a digital wallet associated with their account
- Wallets support basic functionalities such as balance inquiry and transaction

4.1.3 Decentralised Money Transfer:

- Users can send money to other users within the system using decentralized technologies.
- Transactions should be secure, transparent and verifiable.

4.1.4 Smart Contracts:

- Integration of smart contracts using Solidity for executing transactions
- Smart contracts should define rules for money transfers and ensure trust among users.

4.1.4 Transaction Confirmation:

- Users receive notifications or confirmations once money transfer is successfully completed.
- Confirmation should include details such as transition ID, amount and timestamp.

4.1.6 User Dashboard:

- A user-friendly dashboard displaying wallet balance, user's wallet address, transaction history, account details.
- Real-time updates on wallet status and recent transactions.

4.1.7 Security Measures:

- Implementation of security protocols to safeguard user data and transactions.
- Use HTTPS, encryption, and secure authentication methods.

4.1.8 Database Design and Implementation:

- Develop a robust and efficient database structure compatible with the frontend requirements
- Ensure proper normalization and indexing to optimize data storage and retrieval.
- Implement mechanisms for secure data storage and user privacy protection, adhering to industry best practices and compliance standards.

4.1.19 Server-side Logic:

- Design and implement backend logic to handle user interactions, process data, and respond to frontend requests
- Develop APIs for seamless communication between the frontend and backend components, supporting CRUD (Create, Read, Update, Delete) operations.
- Implement authentication and authorization mechanisms to secure endpoints and protect sensitive data from unauthorized access.

4.1.10 Integration with Front-end:

- Ensure seamless integration between the backend functionalities and frontend user interface.
- Test and validate data flow and communication channels to guarantee real-time updates and synchronization between frontend and backend components.
- Implement error handling mechanisms to gracefully manage communication failures and ensure data consistency.

4.1.11 User-friendly Design:

- Implement dynamic content generation based on user actions and data changes, providing a personalized and engaging user experience.
- Craft informative and user-friendly messages to guide users through the application workflows and provide feedback on their user interactions.

4.1.12 Testing and debugging:

- Conduct thorough testing of backend functionalities to identify and address any issues or bugs.
- Utilize debugging tools and techniques to diagnose and resolve backend errors, ensuring the stability and reliability of the application.
- Collaborate with frontend developers to perform integration testing and validate end-to-end functionality across the entire application stack.

4.1.13 Documentation:

- Prepare comprehensive documentation covering database design, API specifications, backend functionality descriptions and deployment instructions.
- Provide clear guidelines and examples for using backend API and interacting with backend services.
- Ensure documentation is updated and maintained throughout the lifecycle to facilitate future maintenance and troubleshooting efforts.

4.2 Non-functional Requirements

4.2.1 Scalability:

- The system should be scalable to handle an increasing number of users and transactions.
- Implement scalable infrastructure components and distributed computing techniques to handle spikes in user traffic and workload demands.

4.2.2 Performance:

- Efficient handling of transactions with minimal latency.
- Quick response times for user interactions.
- Optimize backend processes and algorithms for efficient resource utilization and minimal response times.
- Conduct performance testing and profiling to identify bottlenecks and areas for optimization, ensuring a smooth and responsive user experience.

4.2.3 Reliability:

- The system should be available and reliable, minimizing downtime.
- Backups and recovery plans should be in place.
- Implement fault-tolerant and redundant architectures to ensure high availability and resilience against system failures or disruptions.

4.2.4 Security:

- Implementation of robust security measures to protect user data and financial transitions.
- Enforce robust security measures at the backend to protect against common threats such as unauthorized access, data breaches, and injection attacks.
- Implement encryptions mechanisms to secure data transmission and storage, ensuring confidentiality and integrity of sensitive information

- Regularly audit and update security protocols to address emerging threats and vulnerabilities.

4.2.5 Usability:

- User-friendly interfaces for both web and mobile platforms.
- Clear instructions and error messages for users.

4.2.6 Compatibility:

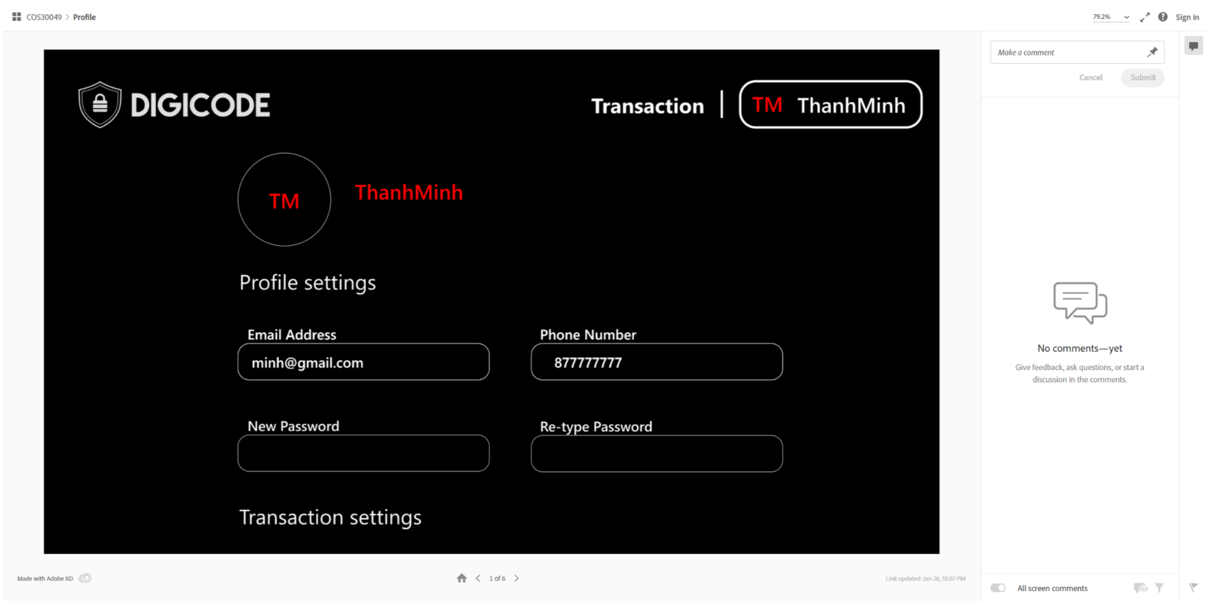
- Ensure backend systems are compatible with a wide range of browsers, devices and operating systems, providing a consistent user experience across different platforms.
- Ensure a consistent user experience across different platforms.

5. Project design

5.1. Frontend Prototype

Link to our frontend Prototype: Link

5.1.1. Setting page


COS30049 > Profile

DIGICODE

Transaction | TM ThanhMinh

TM ThanhMinh

Profile settings

Email Address: minh@gmail.com

Phone Number: 877777777

New Password:

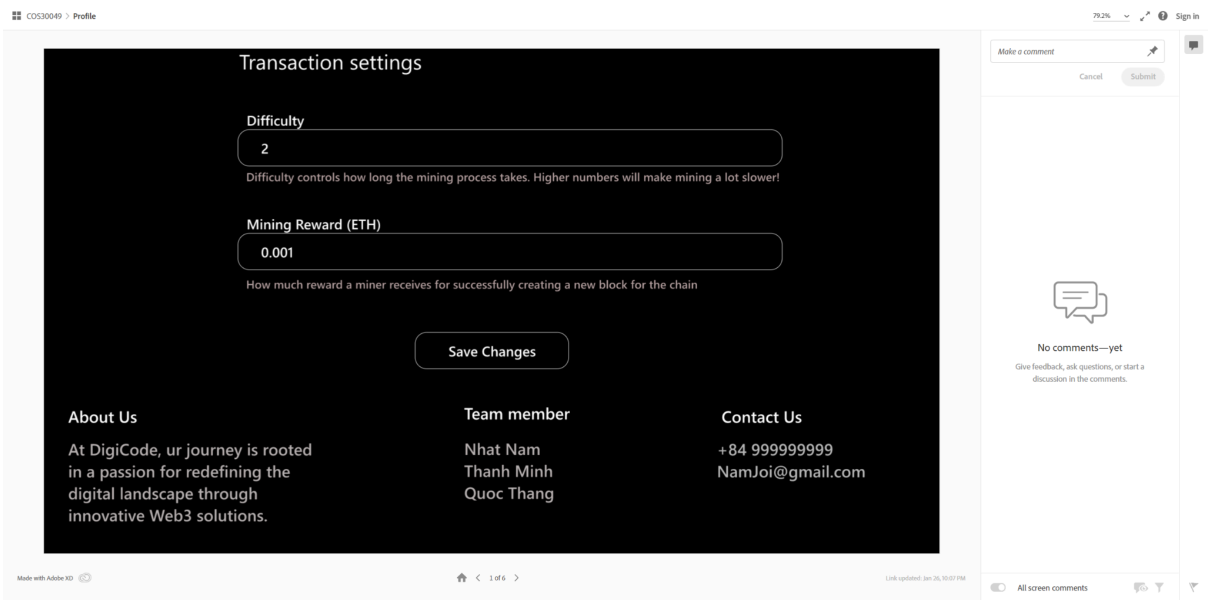
Re-type Password:

Transaction settings

No comments—yet

Give feedback, ask questions, or start a discussion in the comments.

Image 2: Profile settings page


COS30049 > Profile

Transaction settings

Difficulty: 2

Difficulty controls how long the mining process takes. Higher numbers will make mining a lot slower!

Mining Reward (ETH): 0.001

How much reward a miner receives for successfully creating a new block for the chain

Save Changes

About Us

At DigiCode, ur journey is rooted in a passion for redefining the digital landscape through innovative Web3 solutions.

Team member

Nhat Nam
Thanh Minh
Quoc Thang

Contact Us

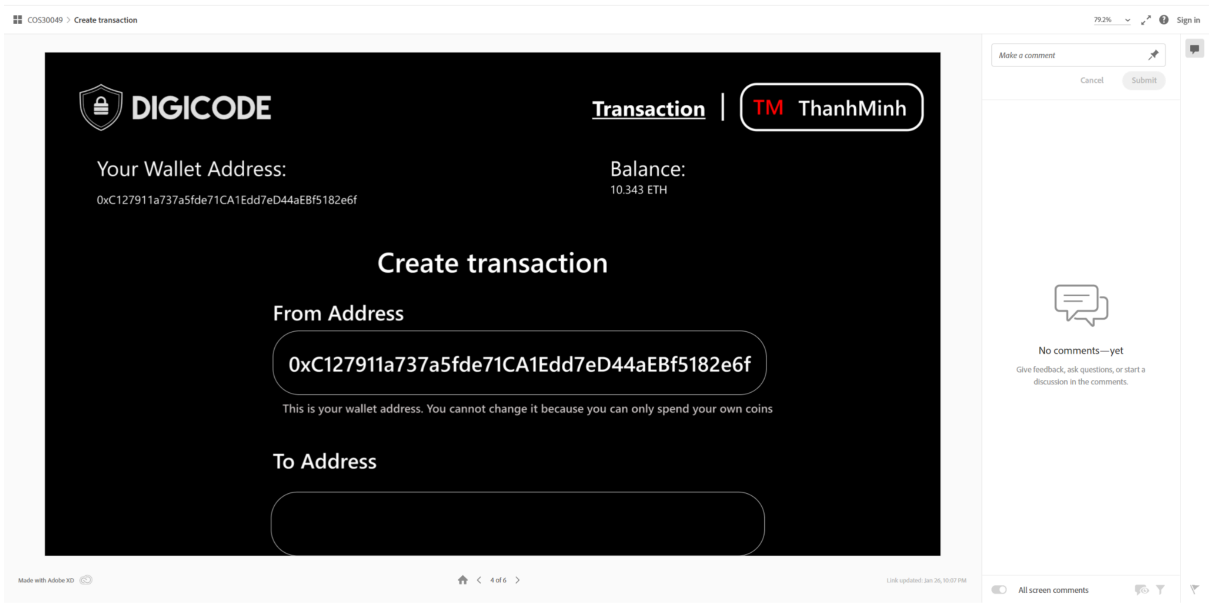
+84 999999999
NamJoi@gmail.com

No comments—yet

Give feedback, ask questions, or start a discussion in the comments.

Image 3: Transaction settings page

5.1.2. Transaction page


COS30049 > Create transaction
79.2%
Sign in
Make a comment
Cancel
Submit
No comments—yet
Give feedback, ask questions, or start a discussion in the comments.
Made with Adobe XD
4 of 6
Link updated: Jan 26, 10:07 PM
All screen comments
DIGICODE
Transaction
TM ThanhMinh

Your Wallet Address:
0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f

Balance:
10.343 ETH

Create transaction

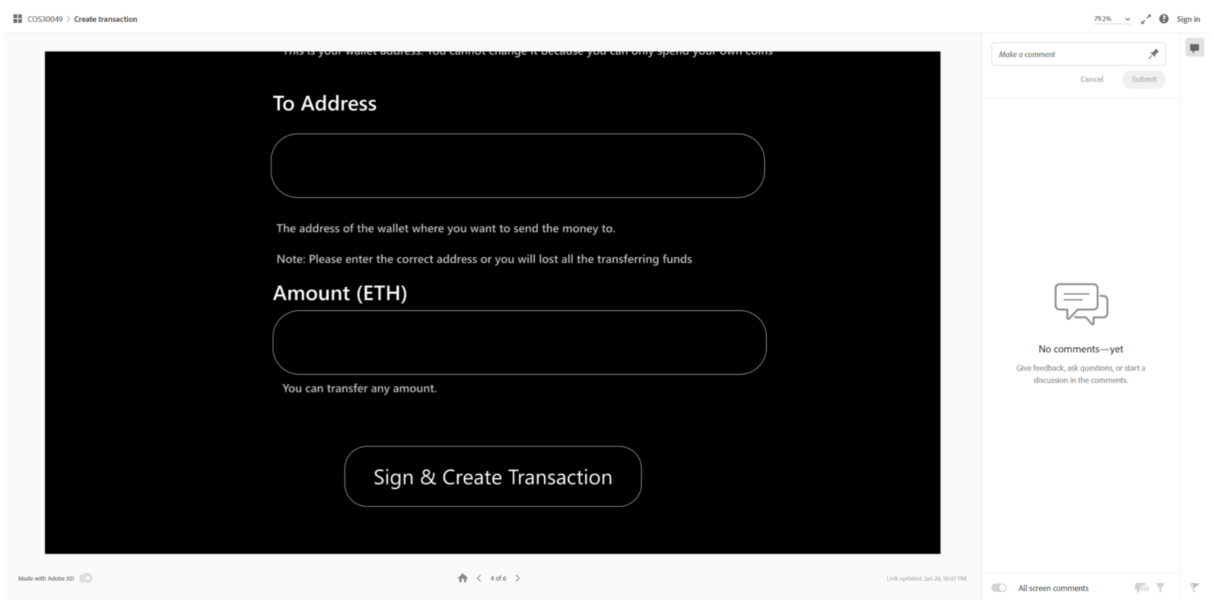
From Address

0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f

This is your wallet address. You cannot change it because you can only spend your own coins

To Address

Image 4: Create transaction form


COS30049 > Create transaction
79.2%
Sign in
Make a comment
Cancel
Submit
No comments—yet
Give feedback, ask questions, or start a discussion in the comments.
Made with Adobe XD
4 of 6
Link updated: Jan 26, 10:07 PM
All screen comments
Sign & Create Transaction

This is your wallet address. You cannot change it because you can only spend your own coins

To Address

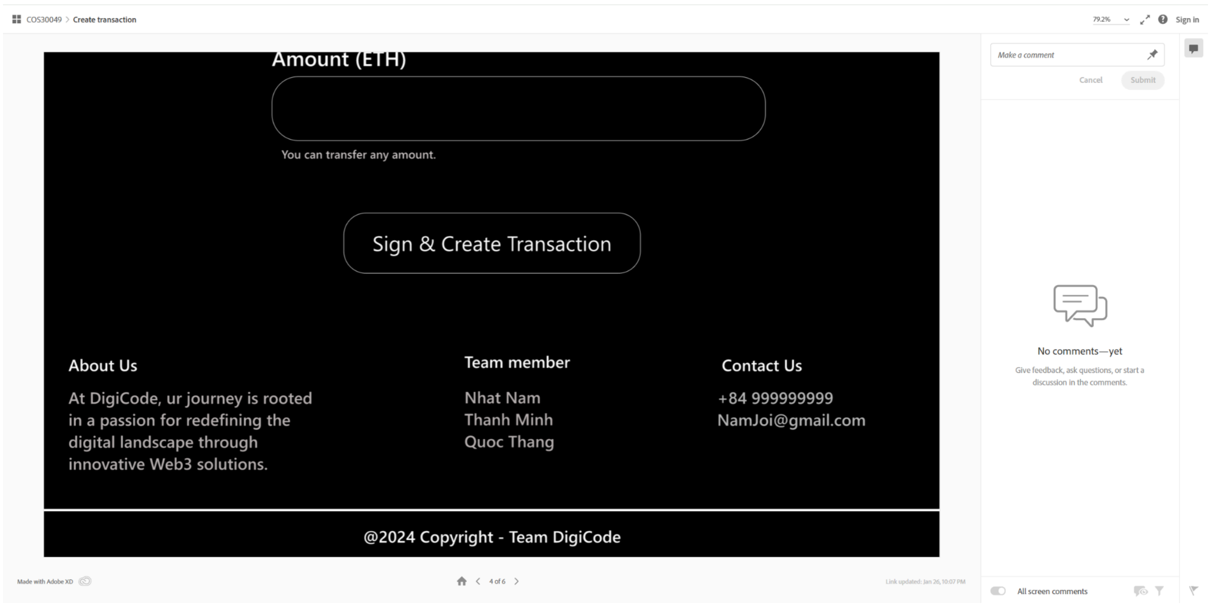
The address of the wallet where you want to send the money to.

Note: Please enter the correct address or you will lost all the transferring funds

Amount (ETH)

You can transfer any amount.

Image 5: Create transaction form


COS30049 > Create transaction
79.2%
Sign in
Amount (ETH)
You can transfer any amount.
Sign & Create Transaction
Make a comment
Cancel
Submit
No comments—yet
Give feedback, ask questions, or start a discussion in the comments.
Made with Adobe XD
4 of 6
All screen comments

About Us

At DigiCode, ur journey is rooted in a passion for redefining the digital landscape through innovative Web3 solutions.

Team member

Nhat Nam
Thanh Minh
Quoc Thang

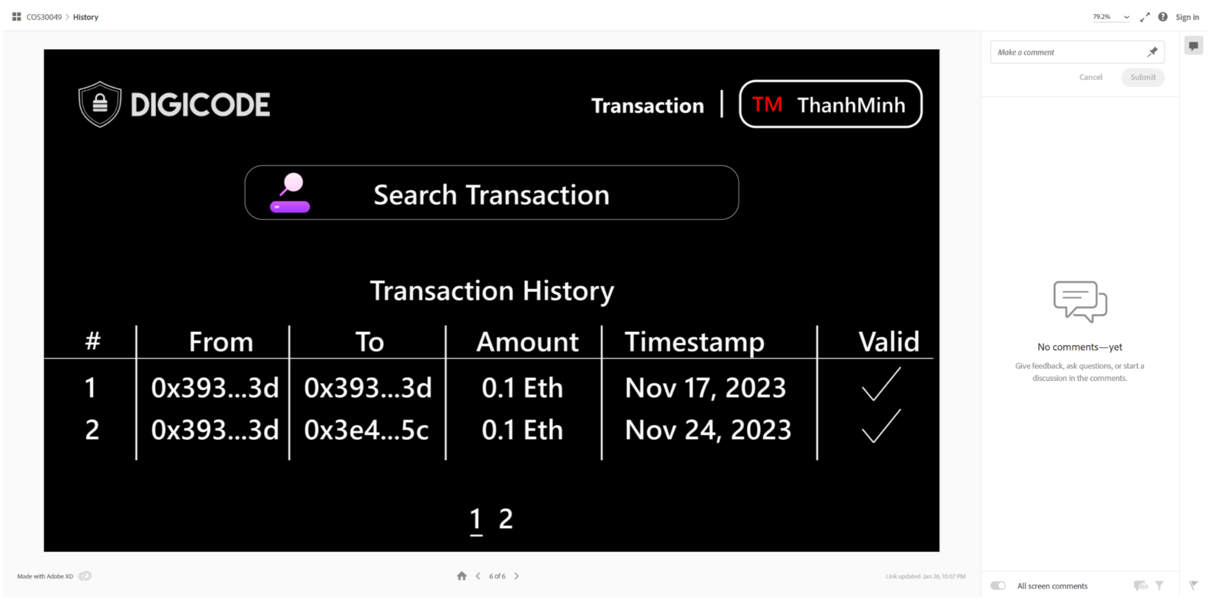
Contact Us

+84 999999999
NamJoi@gmail.com

@2024 Copyright - Team DigiCode

Image 6: Create transaction button

5.1.3. Transaction History page

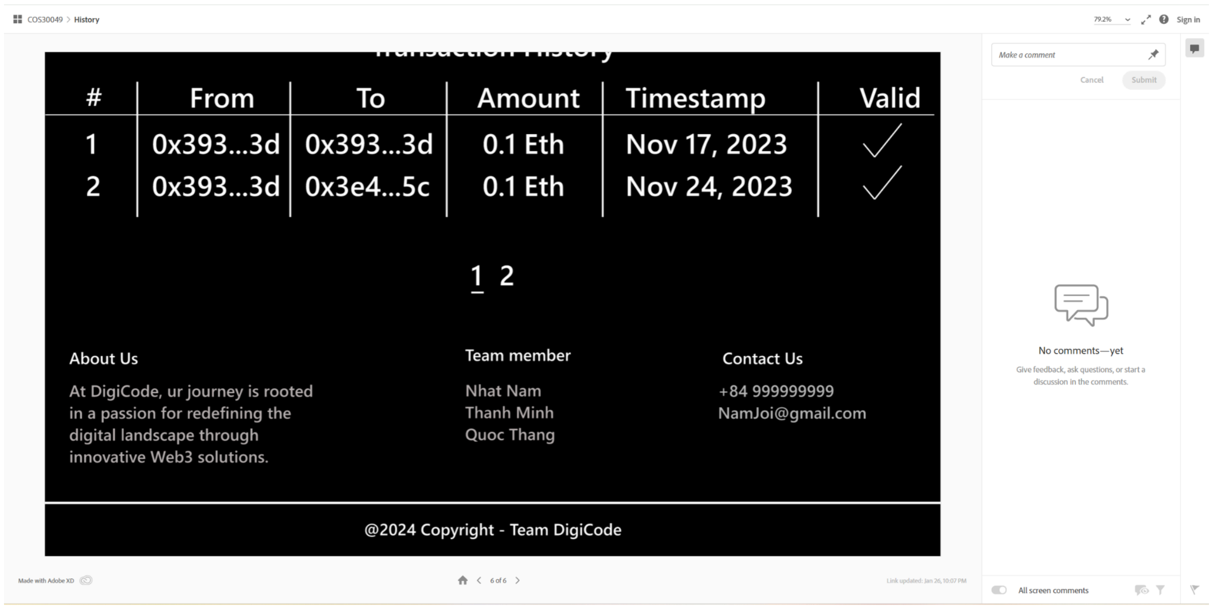


Transaction History

#	From	To	Amount	Timestamp	Valid
1	0x393...3d	0x393...3d	0.1 Eth	Nov 17, 2023	✓
2	0x393...3d	0x3e4...5c	0.1 Eth	Nov 24, 2023	✓

1 2

Image 7: Transaction history table



Transaction History

#	From	To	Amount	Timestamp	Valid
1	0x393...3d	0x393...3d	0.1 Eth	Nov 17, 2023	✓
2	0x393...3d	0x3e4...5c	0.1 Eth	Nov 24, 2023	✓

1 2

About Us

At DigiCode, ur journey is rooted in a passion for redefining the digital landscape through innovative Web3 solutions.

Team member

Nhat Nam
Thanh Minh
Quoc Thang

Contact Us

+84 999999999
NamJoi@gmail.com

@2024 Copyright - Team DigiCode

Image 8: Transaction history pagination

5.2 Overall system architecture design

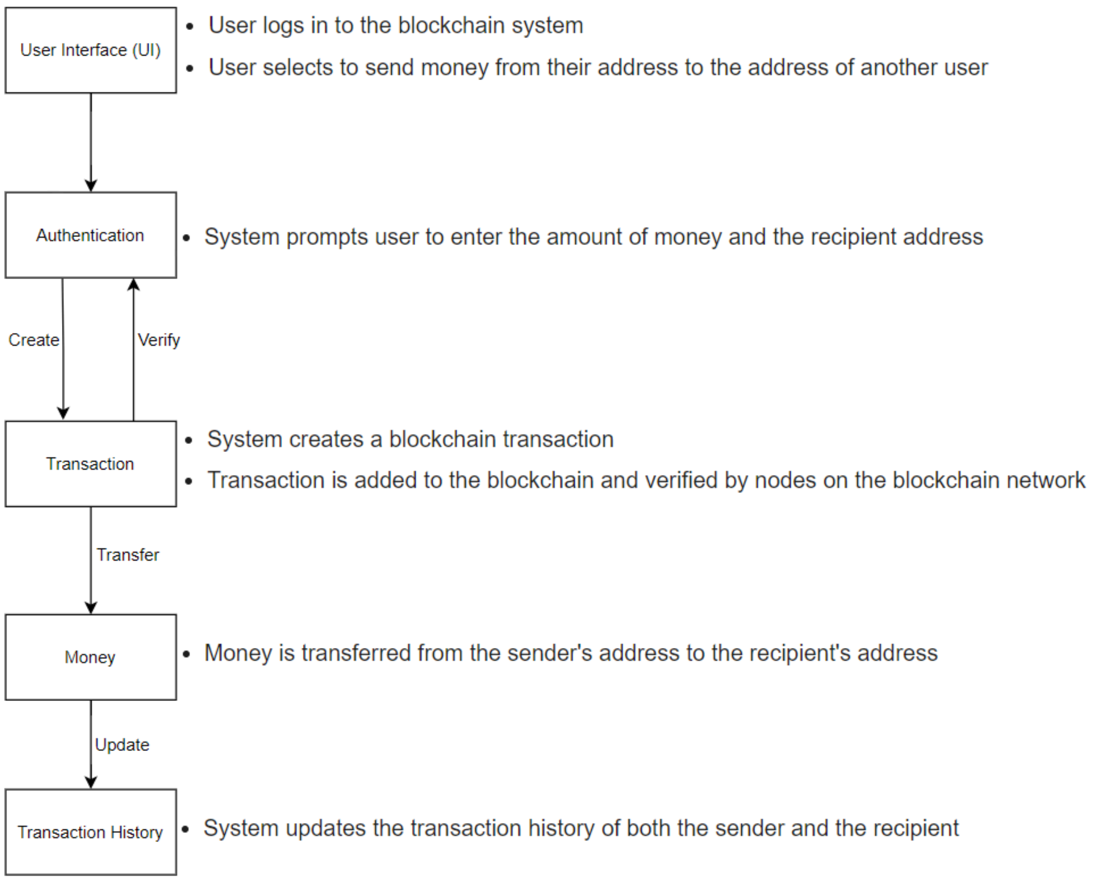

User Interface (UI)
User logs in to the blockchain system
User selects to send money from their address to the address of another user
Authentication
System prompts user to enter the amount of money and the recipient address
Create
Verify
Transaction
System creates a blockchain transaction
Transaction is added to the blockchain and verified by nodes on the blockchain network
Transfer
Money
Money is transferred from the sender's address to the recipient's address
Update
Transaction History
System updates the transaction history of both the sender and the recipient

Image 9: Overall system diagram

5.3 Backend Database Design

- We are using the sqlite3 database for our project, sqlite3 is a relational database. The sqlite3 database is built-in in Django, allowing you to use sqlite3 as the main database in Django without complex configuration. Using sqlite3 in Django is convenient for us during the development and construction of our project.

- **Database**
  - ***User Model***: Used to store information about users in the system. It includes fields like username, email, last_name, first_name, phoneNumber, pin, user_address, data, etc. The User table is used to store user personal information, manage information, and authenticate user information during registration and login processes
  - ***OneTimePassword***: This table stores the OTP code associated with a specific user. It includes fields like user, code, expiration_time. It is used to store the user otp used for authentication and to store the time that the otp was created to use to calculate how long the otp can last
  - ***SaveEmailModel***: This table stores email addresses and related code. It includes fields email, code, expiration_time. This table is used to temporarily store the email and OTP of users, for use in the process of resetting a user's password when they forget it.
  - ***HistoryModel***: This table stores information about the user's transaction history. It includes fields like user_address, username, hash_block, contract_address, transaction_hash, etc. This table is used to store information used in retrieving the user's transaction history.

5.4 API Design

Base URL: http://localhost:8000/ or http://127.0.0.1:8000/

Endpoints:

❖ POST /login

This endpoint is used for user authentication. It verifies the provided username and password and returns user details along with a JWT token upon successful authentication.

- ➢ **Body**:
  - ■ username: the username of the user.
  - ■ password: the password input of the user.
- ➢ **Response**:

  Return a JSON object with the following properties:

  - ■ status: The status of the response.
  - ■ data: A dictionary of user information, each with the following properties:
    - ● id: The unique identifier of the user.
    - ● username: The email address of the user.
    - ● email: The email address of the user.
    - ● lastname: The last name of the user.
    - ● name: The full name of the user (combination of first name and last name).
    - ● balance: The balance of the user's wallet in Ether.
    - ● phone: The phone number of the user.
    - ● address: The Ethereum address associated with the user
    - ● refresh: The refresh token for the user's session.
    - ● token: The access token for the user's session.
  - ■ message: A message indicating the outcome of the login attempt.
- ➢ **Example**:

Request:

```
POST /login
{
 "username": "Minh123",
 "password": "qwerty"
}
```

Response:

```
{
  "status": "200 OK",
  "data": {
    "id": 1,
    "username": "example_user",
    "email": "example@example.com",
    "lastname": "Doe",
    "name": "John Doe",
    "balance": 10.0,
    "phone": "1234567890",
    "address": "0x123abc...",
    "refresh": "example_refresh_token",
    "token": "example_access_token"
  },
  "message": "Login successful"
}
```

- ➢ **Error**:
  - ■ 401 Unauthorized: The user types the wrong username or password.

❖ POST /verify

This endpoint is used for verifying the OTP (one-time password) sent to the user's email during the registration process. It checks if the provided OTP matches the OTP stored in the database and verifies the user's email address. If the OTP is valid, it generates a PIN (personal identification number) for the user and sends it to their email address.

- **Body**:
  - otp: The one-time password (OTP) provided by the user for verification.
- **Response**:
  - status: The status of the response.
  - message: A message indicating the outcome of the verification attempt.
  - data: Additional data related to the response.
- **Example**:

Request:

```
POST /verify
{
  "otp": "123456"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Account verified successfully",
  "data": "Please check and remember your PIN is being sent to your email"
}
```

- **Error**:
  - 400 Bad Request: The OTP is invalid.

❖ PUT /updateProfile

This endpoint is used to update the user's profile information such as password, phone number, etc.

- **Body**:
  - token: The access token of the authenticated user.
  - email: The updated email address of the user.

- password: The updated password of the user.
- confirm_password: The confirmation of the updated password.
- phone: The updated phone number of the user.

➢ **Response**:

- status: The status of the response.
- message: A message indicating the outcome of the update attempt.

➢ **Example**:

Request:

PUT /updateProfile

```
{
  "token": "example_access_token",
  "email": "new_email@example.com",
  "password": "new_password",
  "confirm_password": "new_password",
  "phone": "1234567890"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Change successfully"
}
```

➢ **Error**:

- 401 Unauthorized: When the token is missing or other fields are invalid.

❖ POST /forgotPassword

This endpoint is used to initiate the password reset process. It sends a one-time password (OTP) to the user's email address for verification.

➢ **Body**:

    - username: The username of the user for whom the password reset is requested.
- **Response**:
    - status: The status of the response.
    - message: A message indicating the outcome of the request.
- **Example**:

Request:

POST /forgotPassword

```
{
  "username": "example_username"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "OTP sent successfully"
}
```

- **Error**:
    - 401 Unauthorized: When the user enters an incorrect username.

❖ PUT /changePassword

This endpoint is used to change the password after initiating a password reset process using the one-time password (OTP) sent to the user's email address.

- **Body**:
    - password: The new password to be set.
    - confirm_password: The confirmation of the new password.
    - otp: The one-time password (OTP) received by the user via email.
- **Response**:
    - status: The status of the response.
    - message: A message indicating the outcome of the password change request.

➢ **Example**:

Request:

```
PUT /changePassword
{
  "password": "new_password",
  "confirm_password": "new_password",
  "otp": "123456"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Change password successfully"
}
```

➢ **Error**:

- 401 Unauthorized: When the password not match or OTP is invalid

❖ POST /pending

This endpoint is used to retrieve pending transactions for the authenticated user.

➢ **Body**:

- token: The authentication token of the user.

➢ **Response**:

- status: The status of the response.
- message: A message indicating the outcome of the request.
- data: An array containing information about pending transactions, each with the following properties:
- id: The unique identifier of the transaction.
- transaction_hash: The hash of the transaction.
- receiver: The recipient of the transaction.
- amount: The amount of Ether involved in the transaction.

        - contract_address: The address of the contract associated with the transaction.
        - timestamp: The timestamp of the transaction.

    - **Example**:

Request:

```
POST /pending
{
  "token": "your_authentication_token_here"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Successfully retrieved pending transactions",
  "data": [
    {
      "id": 1,
      "transaction_hash": "0x123456789...",
      "receiver": "0xrecipient_address_here",
      "amount": "0.1234",
      "contract_address": "0xcontract_address_here",
      "timestamp": "2024-02-25T12:00:00Z"
    },
    {
      "id": 2,
      "transaction_hash": "0xabcdef123...",
      "receiver": "0xrecipient_address_here",
      "amount": "0.5678",
      "contract_address": "0xcontract_address_here",
      "timestamp": "2024-02-25T12:30:00Z"
    }
  ]
}
```

- **Error**:
  - 401 Unauthorized: Invalid or expired token

❖ <u>POST /history</u>

This endpoint is used to retrieve transaction history for the authenticated user.

- **Body**:
  - token: The authentication token of the user.
- **Response**:
  - status: The status of the response.
  - message: A message indicating the outcome of the request.
  - data: An array containing information about transaction history, each with the following properties:
  - id: The unique identifier of the transaction.
  - timestamp: The timestamp of the transaction.
  - amount: The amount of Ether involved in the transaction. Positive amounts indicate incoming transactions, while negative amounts indicate outgoing transactions.
  - valid: A boolean indicating whether the transaction was successful (true) or not (false).
  - to: The recipient address of the transaction.
  - from: The sender address of the transaction.
- **Example**:

Request:

```
POST /history
{
  "token": "your_authentication_token_here"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Successfully retrieved history",
  "data": [
    {
      "id": 1,
      "timestamp": "2024-02-25T12:00:00Z",
      "amount": "+0.1234",
      "valid": true,
      "to": "0xrecipient_address_here",
      "from": "0xsender_address_here"
    },
    {
      "id": 2,
      "timestamp": "2024-02-25T12:30:00Z",
      "amount": "-0.5678",
      "valid": true,
      "to": "0xrecipient_address_here",
      "from": "0xsender_address_here"
    }
  ]
}
```

- ➢ **Error**:
  - ■ 401 Unauthorized: Invalid or expired token

❖ POST /execute

This endpoint is used to execute a transaction action for the authenticated user.

- ➢ **Body**:
  - ■ token: The authentication token of the user.

    - pin: The user's PIN for verification.
    - item: The address or identifier of the transaction.
    - action: The action to be performed on the transaction.
- **Response**:
    - status (string): The status of the response.
    - message (string): A message indicating the outcome of the request.
    - data (object): An object containing the following properties:
    - history (array): An array containing information about transaction history.
    - id (integer): The unique identifier of the transaction.
    - timestamp (string): The timestamp of the transaction.
    - amount (string): The amount of Ether involved in the transaction. Positive amounts indicate incoming transactions, while negative amounts indicate outgoing transactions.
    - valid (boolean): A boolean indicating whether the transaction was successful (true) or not (false).
    - to (string): The recipient address of the transaction.
    - from (string): The sender address of the transaction.
    - balance (float): The updated balance of the user's account.
- **Example**:

Request:

POST /execute

```
{
  "token": "your_authentication_token_here",
  "pin": "user_pin_here",
  "item": "transaction_address_or_identifier",
  "action": "transaction_action_here"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Successfully retrieved pending transactions",
  "data": {
    "history": [
      {
        "id": 1,
        "timestamp": "2024-02-25T12:00:00Z",
        "amount": "+0.1234",
        "valid": true,
        "to": "0xrecipient_address_here",
        "from": "0xsender_address_here"
      },
      {
        "id": 2,
        "timestamp": "2024-02-25T12:30:00Z",
        "amount": "-0.5678",
        "valid": true,
        "to": "0xrecipient_address_here",
        "from": "0xsender_address_here"
      }
    ],
    "balance": 123.45
  }
}
```

- ➢ **Error**:
  - ■ 401 Unauthorized: Invalid or expired token.

❖ POST /transaction

This endpoint is used to initiate a transaction from the authenticated user's account to a specified recipient address.

- **Body**:
  - token: The authentication token of the user.
  - to_address: The recipient address of the transaction.
  - amount: The amount of Ether to be sent in the transaction.
  - pin: The user's PIN for verification.
- **Response**:
  - status (string): The status of the response.
  - message (string): A message indicating the outcome of the request.
  - data (object): An object containing the following properties:
  - balance (float): The updated balance of the user's account after the transaction.
- **Example**:

Request:

POST /transaction

```
{
  "token": "your_authentication_token_here",
  "to_address": "recipient_address_here",
  "amount": "amount_in_ethers_here",
  "pin": "user_pin_here"
}
```

Response:

```
{
  "status": "200 OK",
  "message": "Transaction was made successfully",
  "data": {
    "balance": 123.45
  }
```

}

- **Error**:
  - 400 Bad Request: Invalid address or not enough money
  - 401 Unauthorized: Invalid PIN
  - 500 Internal Server Error

❖ <u>POST /signup</u>

This endpoint is used for user registration. It accepts user information such as username, email, password, and phone number, and sends a verification email containing an OTP (one-time password) to the provided email address.

- **Body**:
  - username: Username for logging in
  - first_name: Firstname of user
  - last_name: Lastname of user
  - email: Email of user for verifying
  - password: Password of user for logging in
  - retypePassword: Retype password.
  - phoneNumber: Mobile phone of user
- **Response**:
  - status: The status of the response.
  - message: A message indicating the outcome of the request.
  - data: An object containing the following properties:
  - id: The unique identifier of the newly registered user.
  - username: The username of the newly registered user.
  - email: The email address of the newly registered user.
  - lastname: The last name of the newly registered user.
  - name: The full name of the newly registered user.
  - phone (string): The phone number of the newly registered user.
  - message (string): A message guiding the user to verify their account.
- **Example**:

Request:

{

```
        email: "tranthanhminh17072003@gmail.com"
        first_name: "Thanh Minh"
        last_name: "Tran"
        password: "Minh103@"
        phoneNumber: "0902566135"
        retypePassword: "Minh103@"
        username: "admin"
}
```

Response:

```
{
  "status": "200 OK",
  "data": {
    "id": 123,
    "username": "example_username",
    "email": "example@example.com",
    "lastname": "Example",
    "name": "Example User",
    "phone": "1234567890",
    "message": "Please do not skip the last step to verify your account."
  }
}
```

- ➢ **Error**:
  - ■ 401 Unauthorized: Password and retype password do not match or some input fields are blank.

❖ GET /block

This endpoint is used to fetch all blocks from the blockchain.

- ➢ **Response**:
  - ■ status: The status of the response.

- message: A message indicating the outcome of the request.
- data: An array containing information about each block, with each block represented as an object with the following properties:
- number: The number of the block.
- hash : The hash of the block.
- previous_hash: The hash of the previous block.
- nonce: The nonce of the block.
- timestamp: The timestamp of the block.

➢ **Example**:

Request:

GET /block

Response:

```
{
  "status": "200 OK",
  "message": "Fetch all blocks successfully",
  "data": [
    {
      "number": 123,
      "hash": "0x123456789abcdef",
      "previous_hash": "0xabcdef123456789",
      "nonce": 123456,
      "timestamp": 1645346800
    },
    {
      "number": 124,
      "hash": "0x23456789abcdef12",
      "previous_hash": "0xbcdef123456789a",
      "nonce": 234567,
      "timestamp": 1645346900
    },
    ...
  ]
}
```

- **Error**:
  - 400: Fetch block failed.

❖ <u>GET /block/:id</u>

This endpoint is used to fetch details of a specific block from the blockchain.

- **Params**:
  - id: The identifier of the block to fetch.
- **Response**:
  - status (string): The status of the response.
  - message (string): A message indicating the outcome of the request.
  - data (array): An array containing information about each transaction in the block, with each transaction represented as an object with the following properties:
  - id (integer): The identifier of the transaction.
  - from (string): The sender address of the transaction.
  - to (string): The recipient address of the transaction.
  - hash (string): The hash of the transaction.
  - value (float): The value of the transaction in Ether
- **Example**:

Request:

GET /block/0x123456

Response:

```
{
  "status": "200 OK",
  "message": "Block detail fetched successfully",
  "data": [
    {
      "id": 1,
      "from": "0x123456789abcdef",
      "to": "0xabcdef123456789",
      "hash": "0x123456789abcdef",
      "value": 1.2345
    },
    {
      "id": 2,
      "from": "0x23456789abcdef12",
      "to": "0xbcdef123456789a",
      "hash": "0x23456789abcdef12",
      "value": 5.6789
    },
    ...
  ]
}
```

- **Error**:
    - 400: Error with fetching transactions inside block.

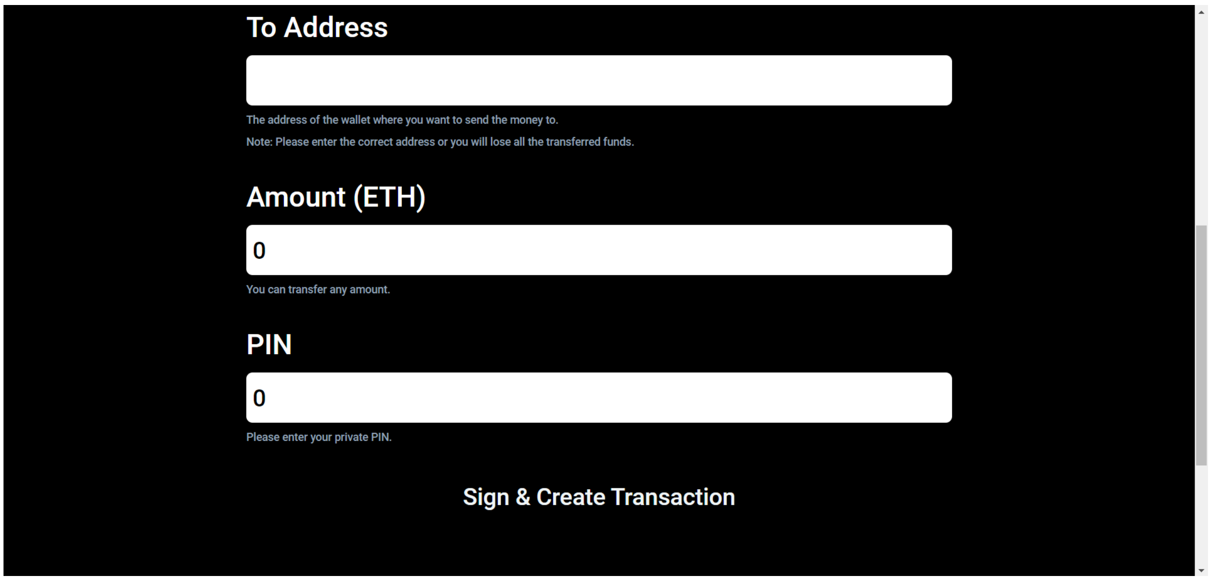
5.5 Function Description

5.5.1 PIN and Password Decryption

In a Decentralized Trading System (DTS), a PIN is essential for confirming transactions, assuring security and transparency. The PIN code will be sent via the user's email entered on the system. The PIN is the final verification step before performing a trade, guaranteeing that the user desires to proceed and preventing fraudulent transactions.

While the PIN does not directly safeguard the user's account, it helps to improve security by providing an extra layer of protection to the transaction, lowering the danger of sensitive information being hacked. PIN is used for decrypting the private key in the database. If the user forgets the PIN then no transaction can be processed.

For the password, we use bcrypt for one way encryption which no one can decrypt the password stored in the database. Only when the user enters, the input password will be compared with the encrypted one stored in the database with python bcrypt function, this happens the same for PIN. If the PIN is the same that the input PIN will be used for decrypting the private key for the user to sign up for the transaction.


To Address

The address of the wallet where you want to send the money to.

Note: Please enter the correct address or you will lose all the transferred funds.

Amount (ETH)

0

You can transfer any amount.

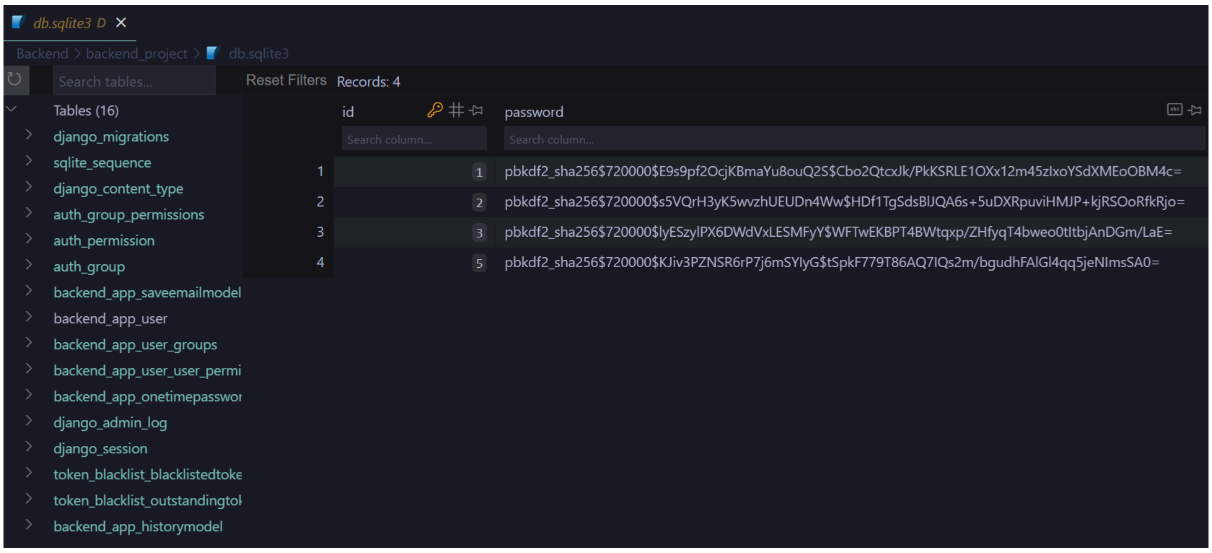
PIN

0

Please enter your private PIN.

Sign & Create Transaction

Image 10: PIN input form


db.sqlite3 D
Backend > backend_project > db.sqlite3

Tables (16)
- django_migrations
- sqlite_sequence
- django_content_type
- auth_group_permissions
- auth_permission
- auth_group
- backend_app_saveemailmodel
- backend_app_user
- backend_app_user_groups
- backend_app_user_user_permi
- backend_app_onetimepasswoi
- django_admin_log
- django_session
- token_blacklist_blacklistedtoke
- token_blacklist_outstandingtol
- backend_app_historymodel

Reset Filters Records: 4

	id	password
1	1	pbkdf2_sha256$720000$E9s9pf2OcjKBmaYu8ouQ2S$Cbo2QtcxJk/PkKSRLE1OXx12m45zlxoYSdXMEoOBM4c=
2	2	pbkdf2_sha256$720000$s5VQrH3yK5wvzhUEUDn4Ww$HDf1TgSdsBlJQA6s+5uDXRpuviHMJP+kjRSOoRfkRjo=
3	3	pbkdf2_sha256$720000$lyESzylPX6DWdVxLESMFyY$WFTwEKBPT4BWtqxp/ZHfyqT4bweo0tItbjAnDGm/LaE=
4	5	pbkdf2_sha256$720000$KJiv3PZNSR6rP7j6mSYlyG$tSpkF779T86AQ7IQs2m/bgudhFAlGl4qq5jeNImsSA0=

Image 11: Stored passwords in the database

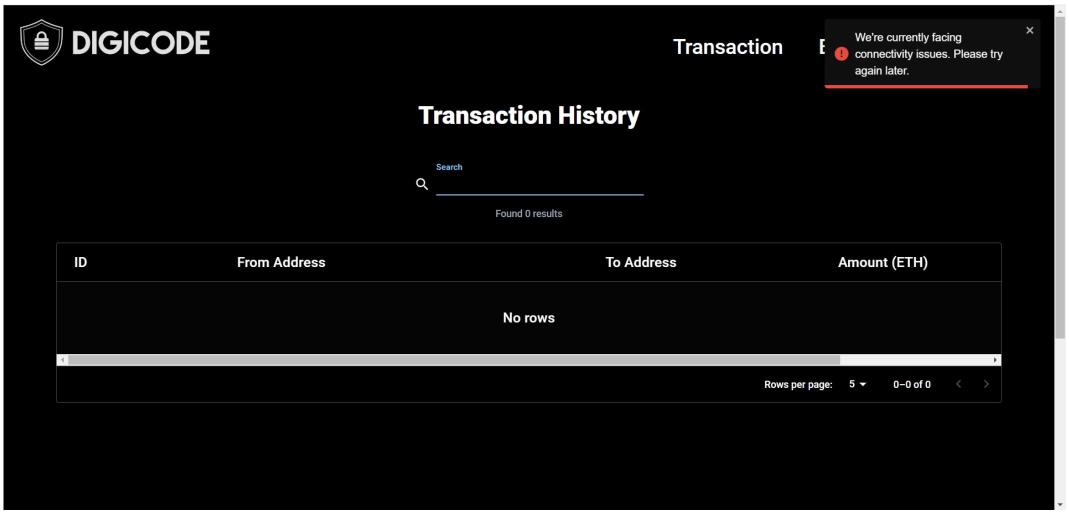
5.5.2 Search bar

The search bar is an essential function in Decentralized Trading Systems (DTS), allowing users to quickly and easily find the coins they want to trade. There are two prevalent search methods: by token name and by wallet address.

Searching by token name: Users may enter the token name, and the system will automatically recommend and show detailed information, such as price, trading volume, liquidity, and so on. This approach is quick and simple, although it might be difficult when looking for new or similarly named tokens.

Searching by wallet address: This approach allows users to discover tokens accurately by providing their wallet addresses. The system will show extensive information, transaction history, and provide access to DTS for trading tokens. However, users must memorize the wallet address and cannot search for tokens that have not yet been exchanged.

Utilizing the search bar effectively will save users time and enhance their overall DTS trading experience.



We're currently facing connectivity issues. Please try again later.

Transaction History

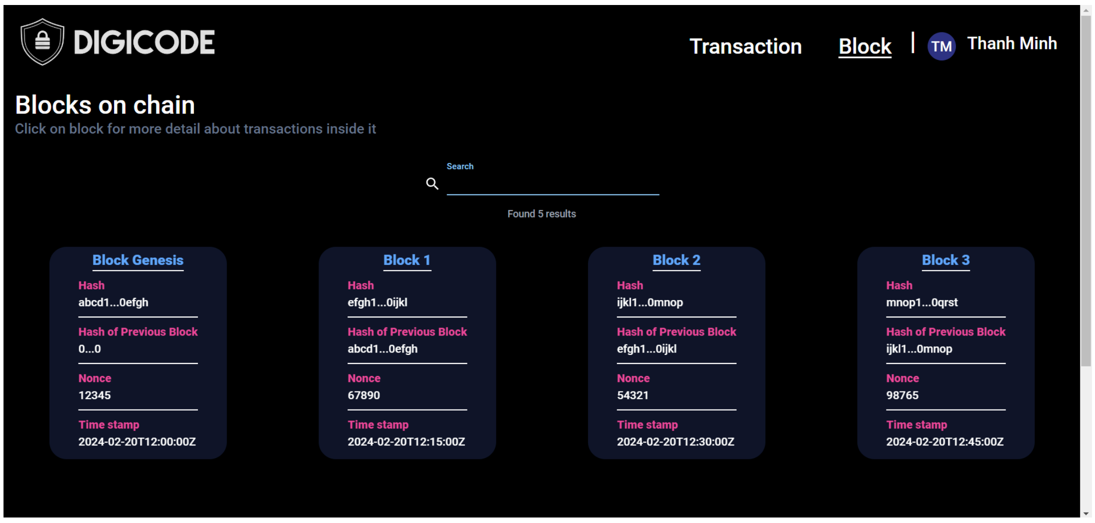
Search

Found 0 results

ID	From Address	To Address	Amount (ETH)
No rows			

Rows per page: 5 0–0 of 0

Image 12: Searchbar of Transaction History page



Blocks on chain

Click on block for more detail about transactions inside it

Search

Found 5 results

Block Genesis
Hash: abcd1...0efgh
Hash of Previous Block: 0...0
Nonce: 12345
Time stamp: 2024-02-20T12:00:00Z

Block 1
Hash: efgh1...0ijkl
Hash of Previous Block: abcd1...0efgh
Nonce: 67890
Time stamp: 2024-02-20T12:15:00Z

Block 2
Hash: ijkl1...0mnop
Hash of Previous Block: efgh1...0ijkl
Nonce: 54321
Time stamp: 2024-02-20T12:30:00Z

Block 3
Hash: mnop1...0qrst
Hash of Previous Block: ijkl1...0mnop
Nonce: 98765
Time stamp: 2024-02-20T12:45:00Z

Image 13: Searchbar of Blocks page

5.5.3 History page

The Transaction History page plays an important role in managing users' transaction information. It acts as a secure and easily accessible repository that stores all performed trading activities, helping you track, manage and understand your financial situation effectively.

Transaction History stores all your transactions, including purchases, transfers, deposits, withdrawals, etc. Each transaction is recorded with complete information such as date and time, transaction type, amount, status, transaction code,... You can easily sort and filter transactions by time, type, status to find information quickly and accurately.

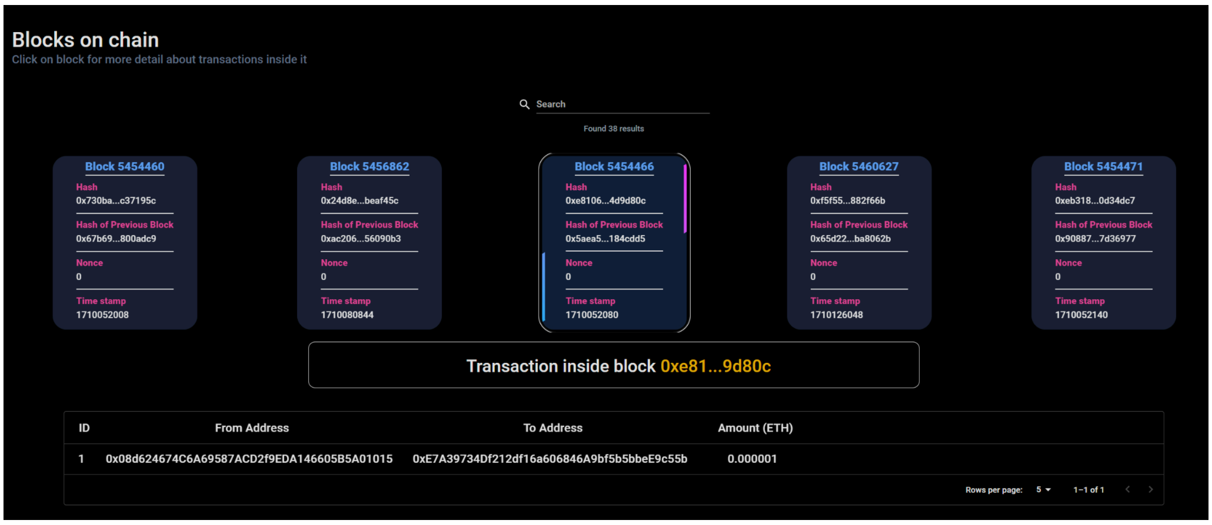
The convenient search bar on the Transaction History page helps you quickly find specific transactions. The system will automatically suggest suitable results when you enter information, saving time and effort. Advanced filters allow you to search by many criteria such as date and time, transaction type, amount, status,...

5.5.4 Block page

The Block Page functionality provides users with an organized view of all the blocks that store transactions within the system. By clicking on each block, users can access detailed information about the transactions contained within that block, displayed neatly in a table format.

Use Cases:

1. Navigate to the Block Page within the application's interface.
2. Upon accessing the Block Page, users are presented with a list of blocks, each representing a set of transactions.
3. Click on a specific block to view its details.
4. Within the block detail view, transactions are neatly displayed in a table format, showing relevant information such as sender, recipient, amount.
5. Users can easily browse through the transactions within the block and analyze their details.
6. Optionally, users can interact with additional functionalities or perform actions related to the displayed transactions, such as filtering, sorting …



Blocks on chain

Click on block for more detail about transactions inside it

Block 5454460	Block 5456862	Block 5454466	Block 5460627	Block 5454471
Hash: 0x730ba...c37195c	Hash: 0x24d8e...beaf45c	Hash: 0xe8106...4d9d80c	Hash: 0xf5f55...882f66b	Hash: 0xeb318...0d34dc7
Hash of Previous Block: 0x67b69...800adc9	Hash of Previous Block: 0xac206...56090b3	Hash of Previous Block: 0x5aea5...184cdd5	Hash of Previous Block: 0x65d22...ba8062b	Hash of Previous Block: 0x90887...7d36977
Nonce: 0	Nonce: 0	Nonce: 0	Nonce: 0	Nonce: 0
Time stamp: 1710052008	Time stamp: 1710080844	Time stamp: 1710052080	Time stamp: 1710126048	Time stamp: 1710052140

Transaction inside block 0xe81...9d80c

ID	From Address	To Address	Amount (ETH)
1	0x08d624674C6A69587ACD2f9EDA146605B5A01015	0xE7A39734Df212df16a606846A9bf5b5bbeE9c55b	0.000001

Image 14: Page block

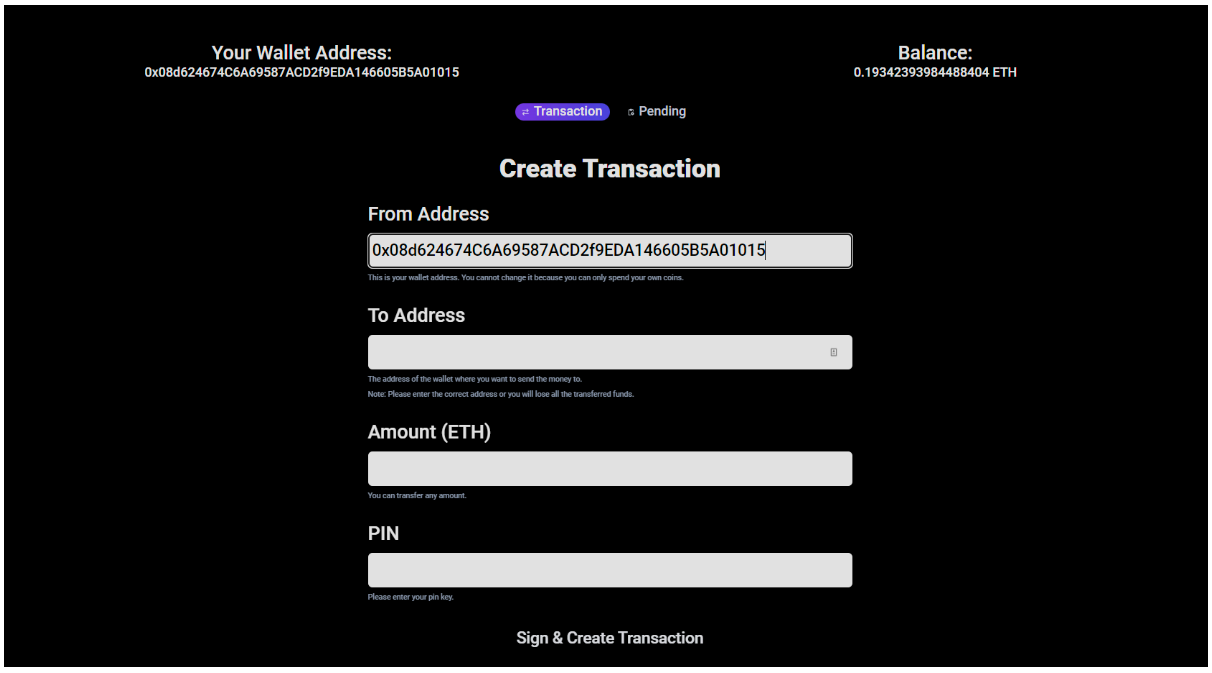
5.5.5 Transaction page

The Transaction Page functionality enables users to send cryptocurrency to a specific user's address. It provides a seamless interface for users to interact with the backend, verify transactions using a private PIN sent through email, and execute transactions securely. Additionally, the functionality facilitates the viewing of pending transactions for execution or withdrawal, with real-time updates to the user's account balance after each transaction.

Use Cases:

1. Access the Transaction Page within the application's interface.
2. Enter the recipient's address and the amount of cryptocurrency to be sent.
3. Input the private PIN received via email for transaction verification.
4. Upon successful verification, a suitable notification pops up confirming the transaction's initiation.
5. View the list of pending transactions awaiting execution or withdrawal.
6. Select a pending transaction to execute or withdraw, providing the necessary confirmation.
7. After executing the transaction, the account balance is updated immediately to reflect the changes.
8. Users can also track transaction history, including timestamps, recipient addresses, amount.

The Transaction Page functionality streamlines the process of cryptocurrency transactions, ensuring security, transparency, and real-time updates for users. It enhances user confidence in managing their cryptocurrency assets and facilitates seamless interaction with the platform's backend services.



Your Wallet Address:
0x08d624674C6A69587ACD2f9EDA146605B5A01015

Balance:
0.19342393984488404 ETH

Create Transaction

From Address

0x08d624674C6A69587ACD2f9EDA146605B5A01015

This is your wallet address. You cannot change it because you can only spend your own coins.

To Address

The address of the wallet where you want to send the money to.

Note: Please enter the correct address or you will lose all the transferred funds.

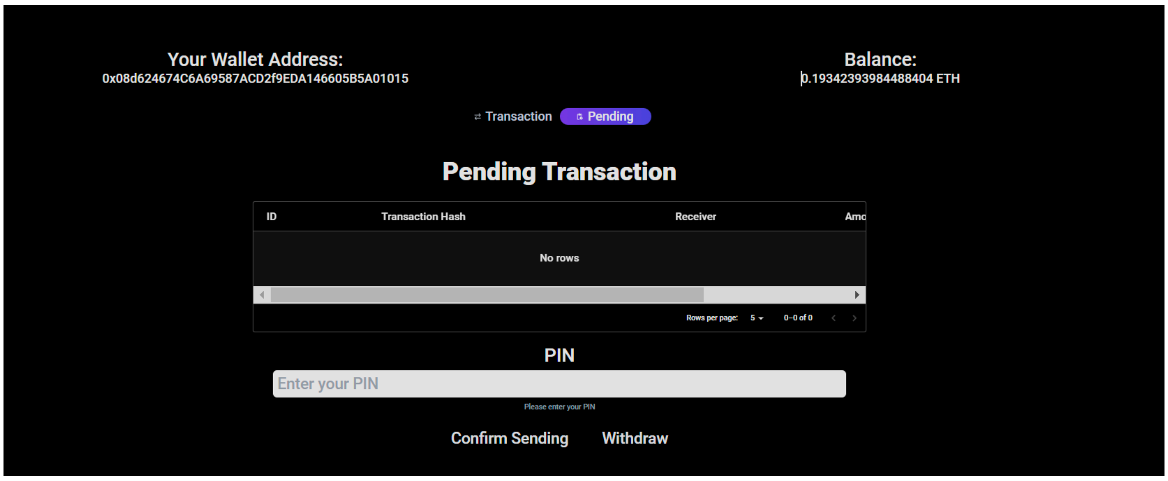
Amount (ETH)

You can transfer any amount.

PIN

Please enter your pin key.

Image 15 : Transaction section in transaction page



Your Wallet Address:
0x08d624674C6A69587ACD2f9EDA146605B5A01015

Balance:
0.19342393984488404 ETH

Pending Transaction

ID	Transaction Hash	Receiver	Amo
No rows			

Rows per page: 5 0–0 of 0

PIN

Enter your PIN

Please enter your PIN

Image 16: Pending section in Transaction page

5.5.6 Forgot password and change password page

The Forgot Password and Change Password pages play an essential role in protecting your account. These two sites work together to ensure you always have secure access to your account. This page will help you regain access when you forget your password and change your current password for added security.

Use Cases:

1. Enter the username of the account.
2. The system will send the password reset code to the email used to create your account, and the page will be forwarded to the password change page.

3. Set up a new password and confirm by re-entering the new password along with the pin code just sent.
4. The system updates your account with the new password after confirmation.

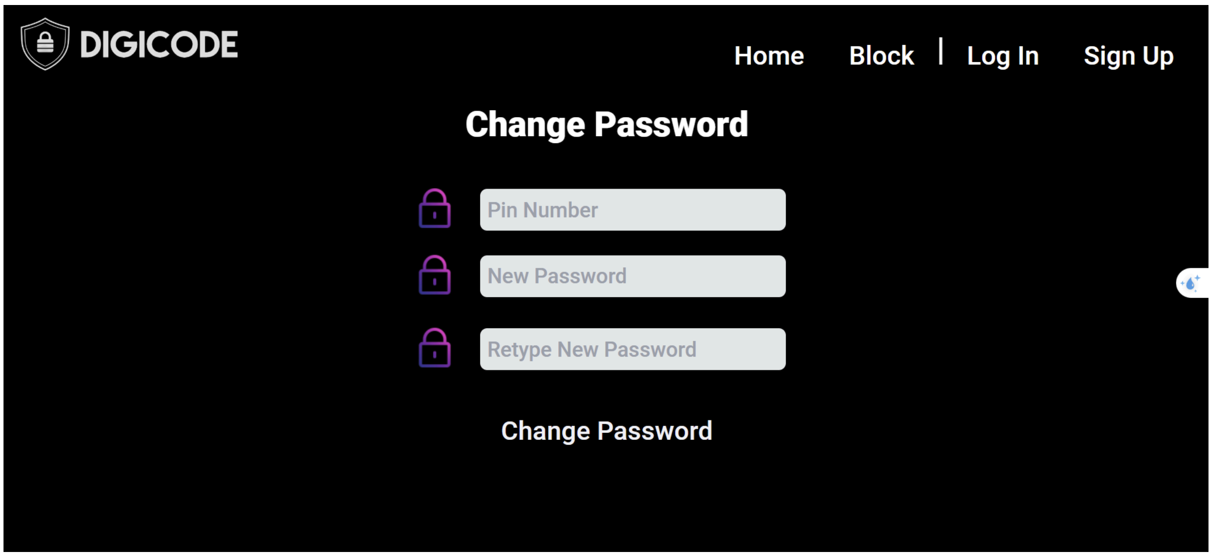


Image 17: Change password page

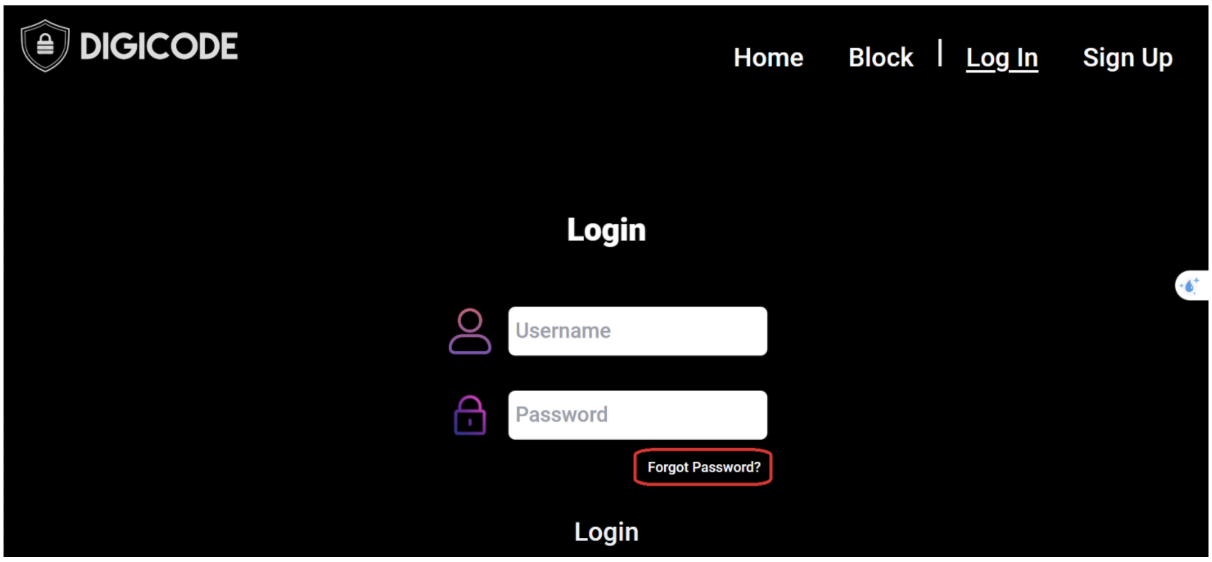


Image 18: Forgot Password button connect to Change Password page

5.5.7 Verify and Sign Up page

The Verify and Sign Up page plays an important role in creating and authenticating your account, opening the door to a complete experience on the platform. It can work in two ways, depending on your current account status.

Use Cases:

1. Fill out all required fields.

2. When you complete the page, it will be forwarded to the Verify page, and a pin code will be sent to the email you just registered.
3. Enter the pin code in the Verify section (pin code only lasts for 1 minute and the new code will be automatically sent back to your email).
4. You have successfully registered and the system has recorded.

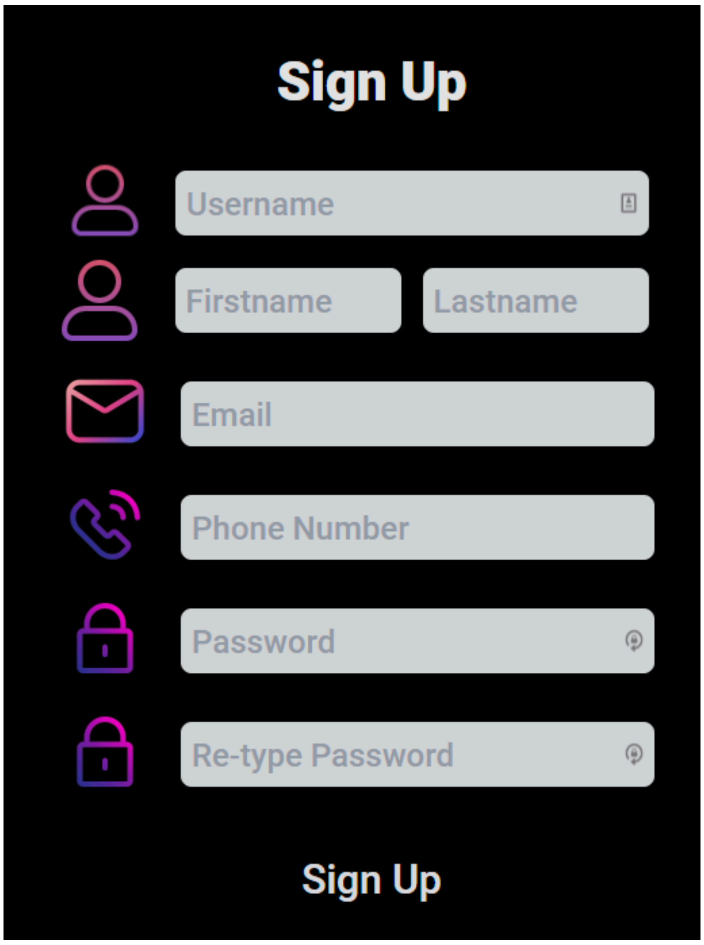


Image 19: Sign Up Page

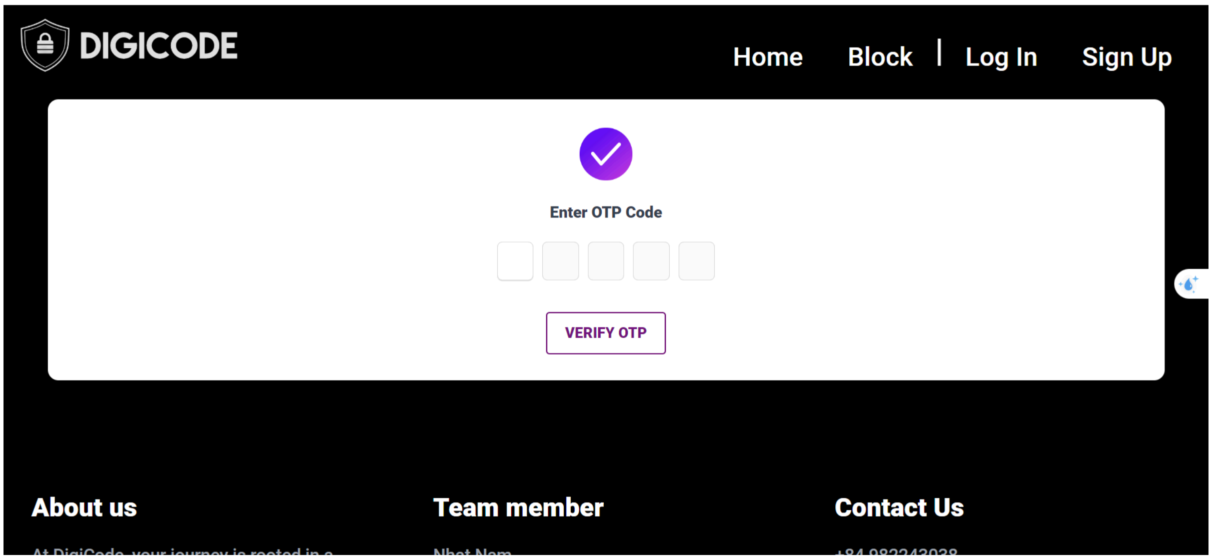


Image 20: Verify Page

5.5.8 Smart contracts

The Smart Contract Interaction with Web3.py functionality facilitates the interaction between a Python application and a Solidity smart contract deployed on the Ethereum blockchain. This functionality enables users to create, manage, and interact with transactions securely and transparently.

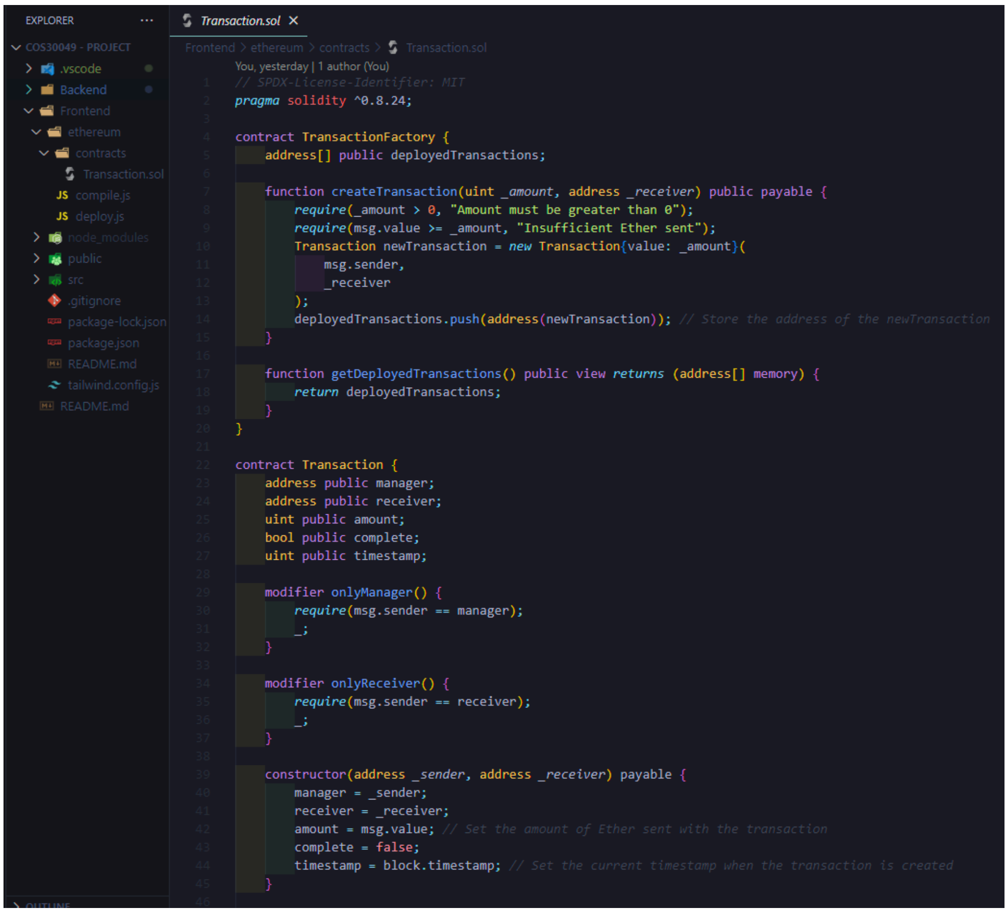
Key Components:

- TransactionFactory Contract: This contract serves as a factory for creating individual Transaction contracts. It includes functions for creating new transactions and retrieving a list of deployed transactions.
- Transaction Contract: Each transaction is represented by an instance of the Transaction contract. It contains details such as the transaction manager, receiver, amount, completion status, and timestamp. Users can execute or withdraw transactions, retrieve transaction information, and receive event notifications upon completion.
- Smart Contract Deployment: The solidity code is compiled and deployed onto the Ethereum blockchain using NodeJs. After deployment, the contract addresses are obtained for interaction.
- Web3.py Integration: Web3.py, a Python library for interacting with Ethereum, is utilized to communicate with the deployed smart contracts. It provides

functionalities to send transactions, call contract methods, and retrieve contract data.

Use Cases:

- Deploy the Smart Contract: Compile the solidity code and deploy the TransactionFactory and Transaction contracts onto the Ethereum blockchain using appropriate deployment tools.
- Obtain Contract Addresses: Retrieve the addresses of the deployed contracts for use in the Python application.
- Initialize Web3.py: Configure Web3.py to connect to the Ethereum network and provide the necessary credentials for transaction signing and contract interaction.
- Create Transactions: Utilize the TransactionFactory contract to create new transactions by specifying the amount and receiver's address.
- Execute or Withdraw Transactions: Interact with individual Transaction contracts to execute or withdraw transactions based on the user's actions.
- Retrieve Transaction Information: Fetch transaction details such as manager, receiver, amount, completion status, timestamp, and balance using contract methods provided by Web3.py.
- Handle Transaction Events: Subscribe to transaction completion events emitted by the smart contracts to receive real-time notifications.



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract TransactionFactory {
    address[] public deployedTransactions;

    function createTransaction(uint _amount, address _receiver) public payable {
        require(_amount > 0, "Amount must be greater than 0");
        require(msg.value >= _amount, "Insufficient Ether sent");
        Transaction newTransaction = new Transaction{value: _amount}(
            msg.sender,
            _receiver
        );
        deployedTransactions.push(address(newTransaction)); // Store the address of the newTransaction
    }

    function getDeployedTransactions() public view returns (address[] memory) {
        return deployedTransactions;
    }
}

contract Transaction {
    address public manager;
    address public receiver;
    uint public amount;
    bool public complete;
    uint public timestamp;

    modifier onlyManager() {
        require(msg.sender == manager);
        _;
    }

    modifier onlyReceiver() {
        require(msg.sender == receiver);
        _;
    }

    constructor(address _sender, address _receiver) payable {
        manager = _sender;
        receiver = _receiver;
        amount = msg.value; // Set the amount of Ether sent with the transaction
        complete = false;
        timestamp = block.timestamp; // Set the current timestamp when the transaction is created
    }

```

Image 20: Part of the file Transaction.sol

5.6 Project Deployment Instruction

- **Frontend/Blockchain:**

From root folder here is instruction to run Frontend , compile and deploy smart contracts

- **cd Frontend/** → *locate the Frontend folder.*
- **npm i** → *install all libraries that are necessary for the system.*
- **node compile.js** → *Using to compile file Transaction.sol*
- **cd ethereum** → *locate the ethereum folder.*
- **node deploy.js** → *Using to deploy contract to Infura Sepolia testnet network*
- **npm start** → *using this command to start the Frontend server.*
- The project will run at localhost : ***http://localhost:3000/.***

- **Backend**

From root folder here is instruction to run install packages and run server

- **cd Backend/** → *locate the Backend folder.*
- **Source .venv/Scripts/activate** → *active virtual environment.*
- **pip install -r requirements.txt** → *install all libraries that are necessary for the system.*
- **cd backend_project/** → *go to the backend_project folder.*

***Note**: This folder needs to have the file .env which is not uploaded to Github due to security reasons, and only has it in the submitted folder.*

- **py manage.py runserver** → *using this command to start the Backend server.*

6. Result and Performance Evaluation

Link to access the web page online: https://cos30049-digicode.netlify.app/

6.1 Library used

6.1.1 Libraries used for Front-end

Here are list of the library used with React to develop the frontend phase:

- *@mui/material*: This is the official Material UI library for React Offering a comprehensive set of pre-built, customizable UI components.
- *@mui/x-data-grid*: This library provides a powerful data grid component for React applications.
- *eslint-plugin-react* and *eslint-plugin-react-hooks*: These plugins for ESLint to help enforce React coding conventions and catch any potential errors in the development stage to ensure the code quality and consistency.
- *react-fast-marquee*: This library provides a component for creating scrolling text marquees in React.
- *react-router-dom*: This library is used for client-side routing in React applications, enabling navigation between different views and components.
- *react-toastify*: This library provides a way to display toast notifications in React applications, informing users about events or actions.
- *tailwindcss*: This is a utility-first CSS framework that provides a vast collection of CSS classes for styling React components.
- *axios*: This library helps to make HTTP requests such as fetching data from a server or sending data to a server.
- *@mui/icons-material*: This library provides a lot of icons for the Material UI components.

All the above libraries can be install in the project with the command line *npm install [library]*

6.1.2 Libraries used for Back-end

Here are list of the library used with Django to develop the Backend phase:

- *axios*: This library helps to make HTTP requests such as fetching data from a server or sending data to a server.
- *djangorestframework-simple* : This module offers straightforward JSON Web Token (JWT) integration with the Django REST Framework, complete with token refresh and validation.
- *django-cors-headers* : With the help of this library, Django can handle CORS (Cross-Origin Resource Sharing) headers to accept requests from different origins.
- *django-environ :* With Django, this library facilitates the management of application configuration from environment variables and.env files.
- *bcrypt* : used for encryption. It uses a strong hash algorithm for encryption
- *python-decouple* : used for Python application configuration management. It offers a straightforward method for reading values and environment variables from configuration files.
- *web3* : used to interact with the Ethereum blockchain via web protocol 3

6.2 Package Installation

The client should have the NodeJS installed on the computer, if not the client can follow this link to install the NodeJS (https://nodejs.org/en/download).

1. Navigate to the folder
2. Execute this in the command line to install packages: *npm install*
3. Execute this in the command line to start the server: *npm start*
4. Then navigate to http://localhost:3000/ to open the web view.

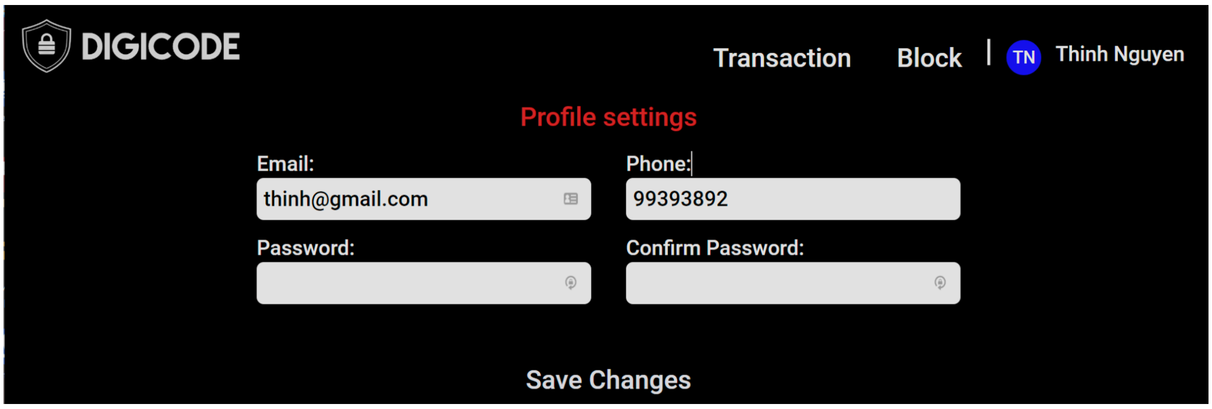
6.3 Result

Link to our website which is deployed online: https://cos-30049-project.vercel.app/
Link to our source code on Github: https://github.com/NamJoi/COS30049-Project.git
There is a user that have been created with the following credentials:

- Username: Thinh
- Password: Thinh@123

6.3.1 Settings page

In the first section of the "Profile and Transaction Settings" page, we have designed the Profile section to allow users to edit personal information such as "Email", "Phone", and "Password".


DIGICODE

Transaction Block | TN Thinh Nguyen

Profile settings

Email: thinh@gmail.com

Phone: 99393892

Password:

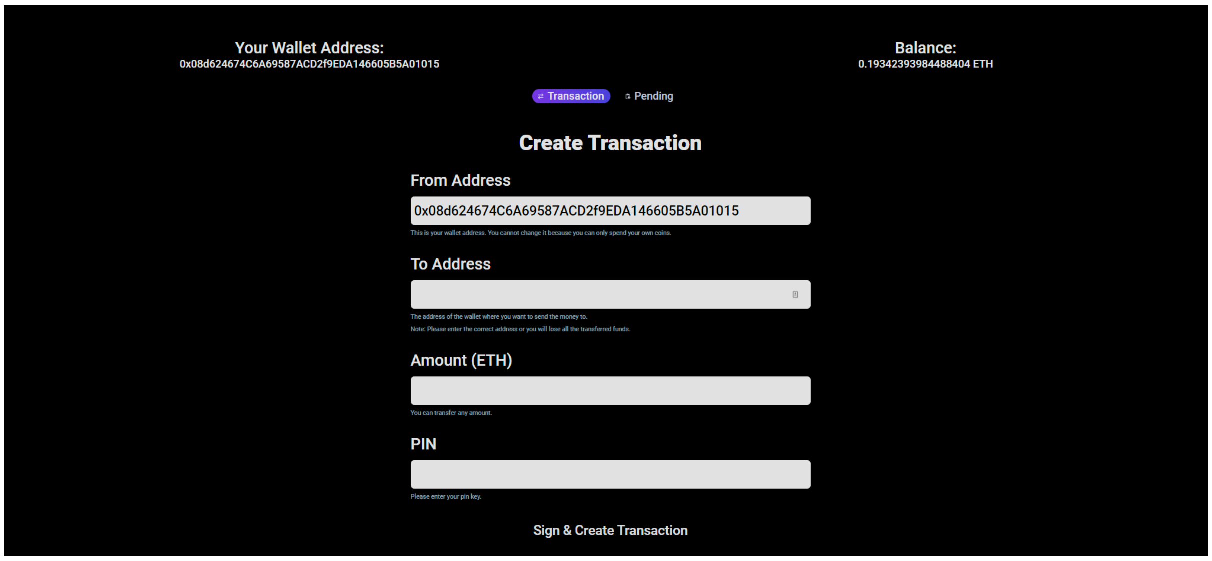
Confirm Password:

Save Changes

Image 21: Settings section

6.3.2 Transaction page

On this page, we provide users with the functionality to transfer ETH from one address to another. The page will display the default address and the total amount of ETH the user currently possesses. Users can initiate the process of transferring ETH from one address to another based on their decision.


Your Wallet Address:
0x08d624674C6A69587ACD2f9EDA146605B5A01015

Balance:
0.19342393984488404 ETH

Transaction Pending

Create Transaction

From Address
0x08d624674C6A69587ACD2f9EDA146605B5A01015
This is your wallet address. You cannot change it because you can only spend your own coins.

To Address

The address of the wallet where you want to send the money to.
Note: Please enter the correct address or you will lose all the transferred funds.

Amount (ETH)

You can transfer any amount.

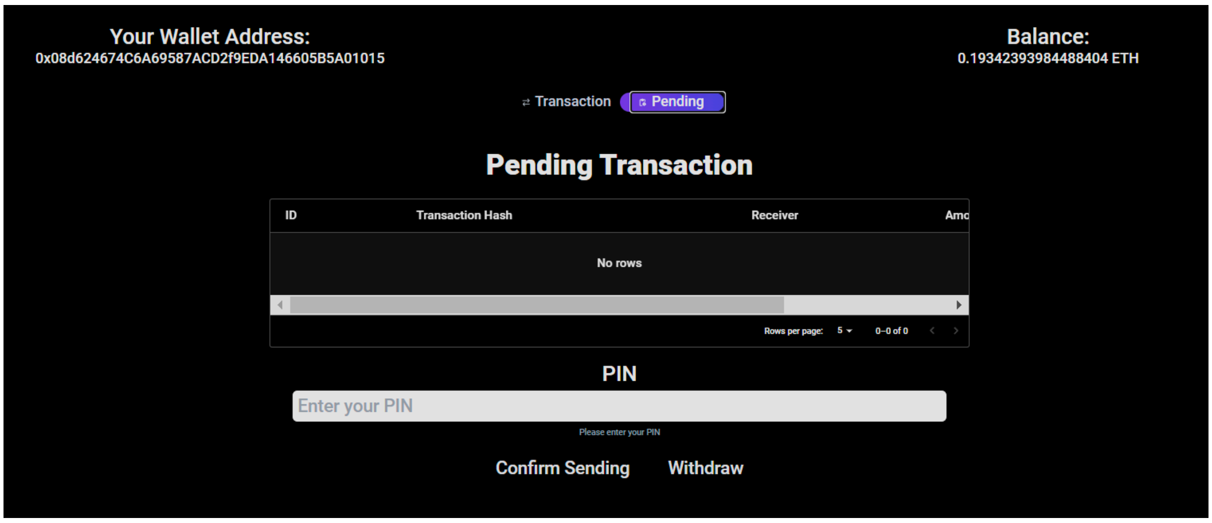
PIN

Please enter your pin key.

Sign & Create Transaction

Image 22: Create Transaction section

When the user presses the button, the system will receive the information and execute the transaction.



Your Wallet Address:
0x08d624674C6A69587ACD2f9EDA146605B5A01015

Balance:
0.19342393984488404 ETH

Pending Transaction

ID	Transaction Hash	Receiver	Amo
No rows			

Rows per page: 5 0–0 of 0

PIN

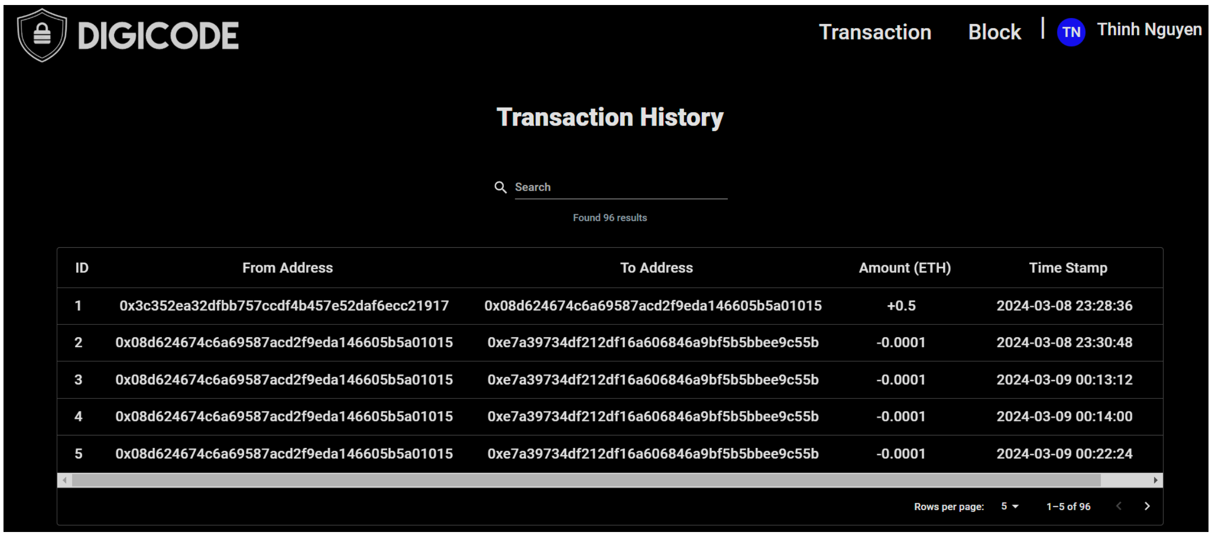
Enter your PIN

Please enter your PIN

Image 23: Pending section

6.3.3 Transaction History page

This page will enable users to view and manage all the transaction history they have conducted. All transaction history will be displayed in a table format. This table provides comprehensive details such as "ID", "From Address", "To Address", "Amount (ETH)", "Time Stamp", and "Valid".



Transaction History

Search

Found 96 results

ID	From Address	To Address	Amount (ETH)	Time Stamp
1	0x3c352ea32dfbb757ccdf4b457e52daf6ecc21917	0x08d624674c6a69587acd2f9eda146605b5a01015	+0.5	2024-03-08 23:28:36
2	0x08d624674c6a69587acd2f9eda146605b5a01015	0xe7a39734df212df16a606846a9bf5b5bbee9c55b	-0.0001	2024-03-08 23:30:48
3	0x08d624674c6a69587acd2f9eda146605b5a01015	0xe7a39734df212df16a606846a9bf5b5bbee9c55b	-0.0001	2024-03-09 00:13:12
4	0x08d624674c6a69587acd2f9eda146605b5a01015	0xe7a39734df212df16a606846a9bf5b5bbee9c55b	-0.0001	2024-03-09 00:14:00
5	0x08d624674c6a69587acd2f9eda146605b5a01015	0xe7a39734df212df16a606846a9bf5b5bbee9c55b	-0.0001	2024-03-09 00:22:24

Rows per page: 5 1–5 of 96

Image 24: Transaction History table

Moreover, for users looking to save time in searching, we have implemented a search bar, allowing users to search for specific transaction histories.

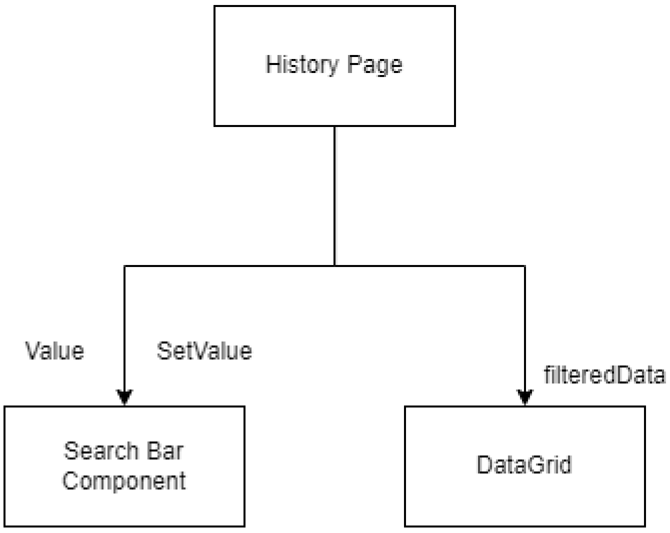
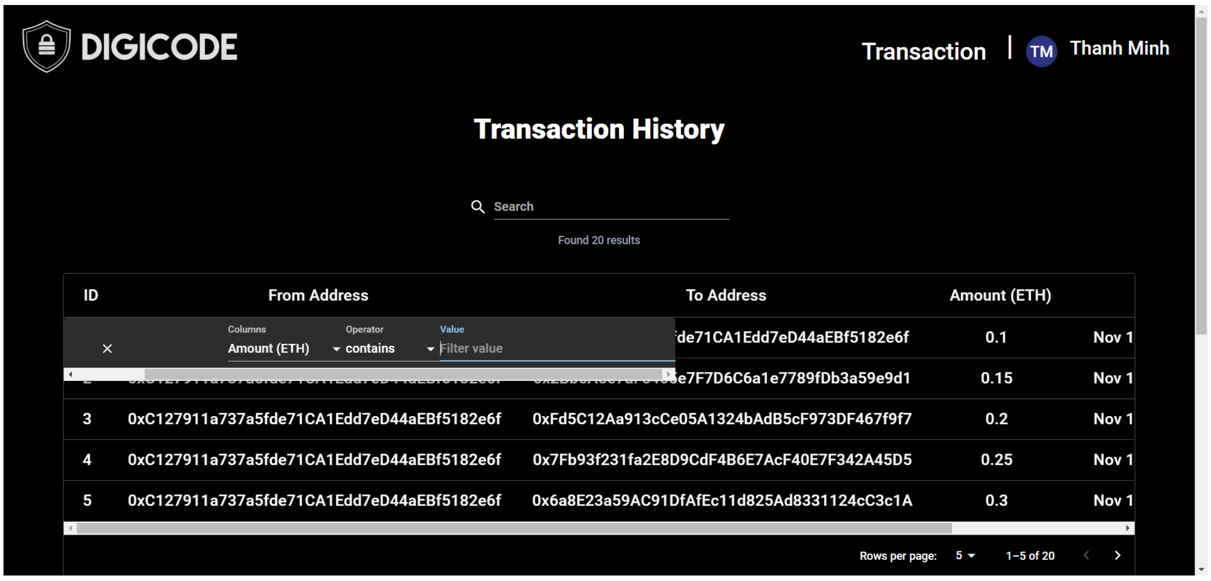


Image 25: Structure of History page.

In the page History, it will fetch the data of transaction history. Then users can search for any transaction that they are interested in. The Search bar component will get the input from the user and send it back to the History component and then there will be a function for filtering the user input to match with the data then it will display in the Data Grid (Image 16).

Additionally, we offer several user-friendly features like "Sort", "Filter", or "Hide and Show column" (Image 16 to Image 20).



Transaction History

Search

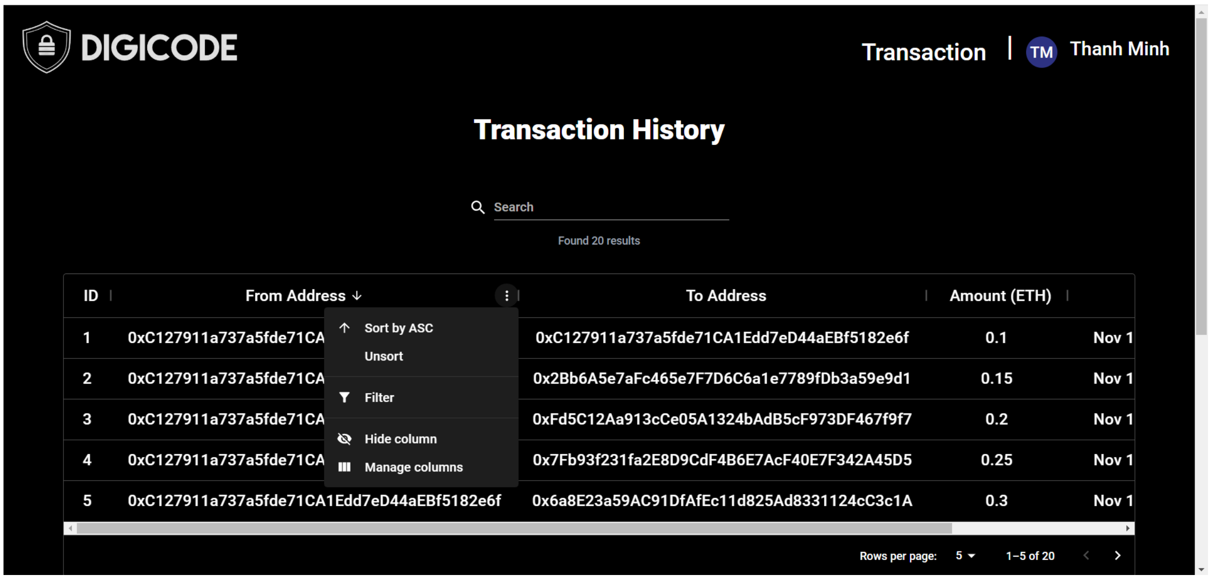
Found 20 results

Columns: Amount (ETH) | Operator: contains | Value: Filter value

ID	From Address	To Address	Amount (ETH)	
		de71CA1Edd7eD44aEBf5182e6f	0.1	Nov 1
		e7F7D6C6a1e7789fDb3a59e9d1	0.15	Nov 1
3	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0xFd5C12Aa913cCe05A1324bAdB5cF973DF467f9f7	0.2	Nov 1
4	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0x7Fb93f231fa2E8D9CdF4B6E7AcF40E7F342A45D5	0.25	Nov 1
5	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0x6a8E23a59AC91DfAfEc11d825Ad8331124cC3c1A	0.3	Nov 1

Rows per page: 5 1–5 of 20

Image 26: Filter function

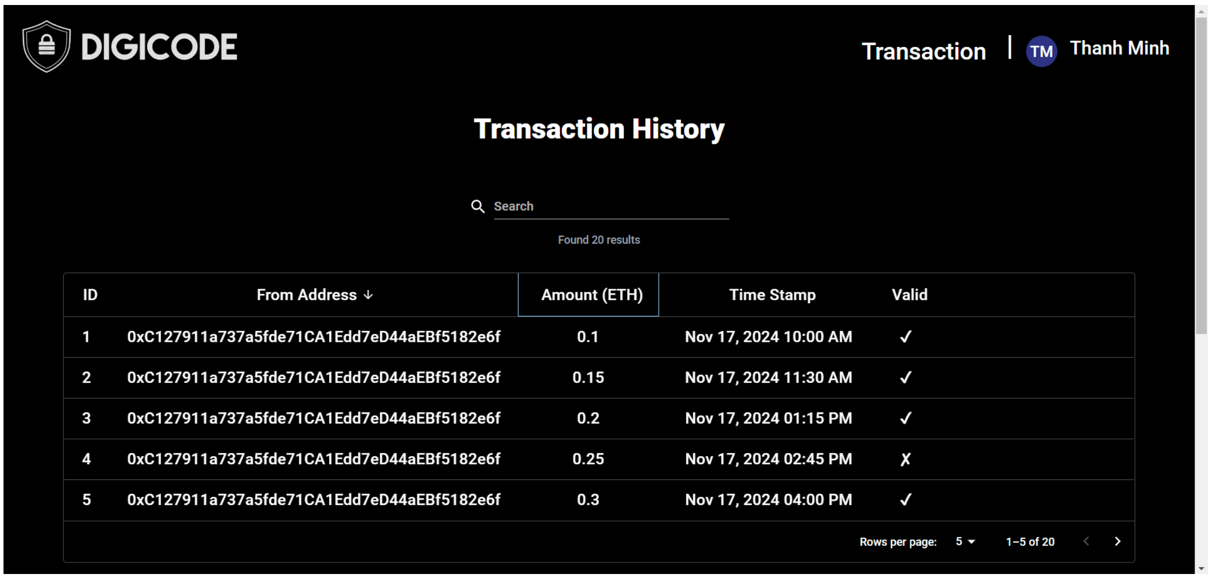


Transaction History

Found 20 results

ID	From Address ↓	To Address	Amount (ETH)	
1	0xC127911a737a5fde71CA	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0.1	Nov 1
2	0xC127911a737a5fde71CA	0x2Bb6A5e7aFc465e7F7D6C6a1e7789fDb3a59e9d1	0.15	Nov 1
3	0xC127911a737a5fde71CA	0xFd5C12Aa913cCe05A1324bAdB5cF973DF467f9f7	0.2	Nov 1
4	0xC127911a737a5fde71CA	0x7Fb93f231fa2E8D9CdF4B6E7AcF40E7F342A45D5	0.25	Nov 1
5	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0x6a8E23a59AC91DfAfEc11d825Ad8331124cC3c1A	0.3	Nov 1

Image 27: Sort function

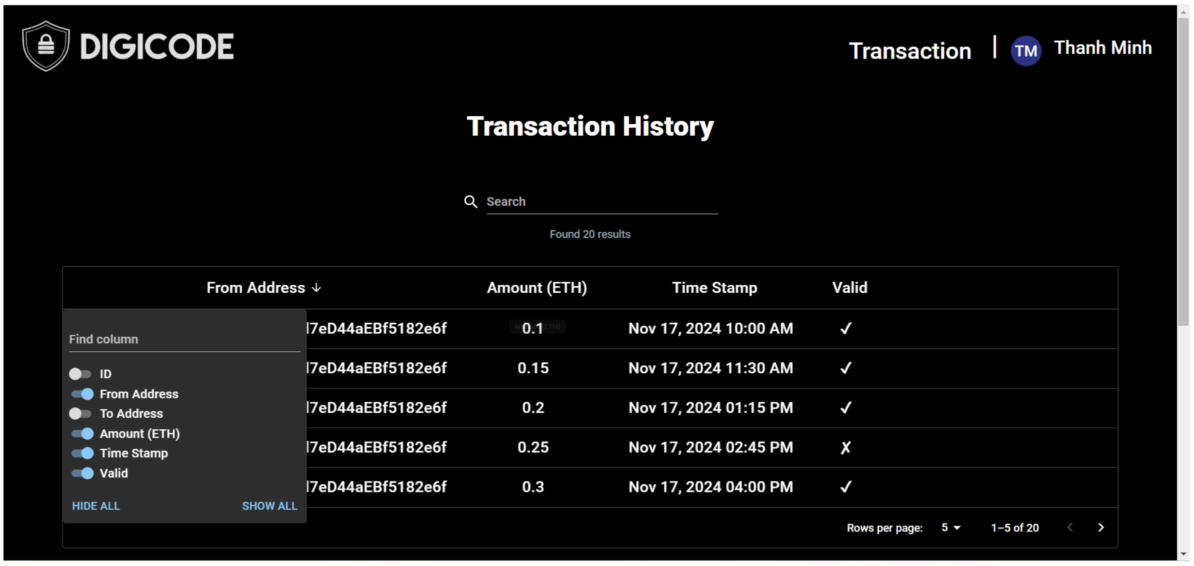


Transaction History

Found 20 results

ID	From Address ↓	Amount (ETH)	Time Stamp	Valid
1	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0.1	Nov 17, 2024 10:00 AM	✓
2	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0.15	Nov 17, 2024 11:30 AM	✓
3	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0.2	Nov 17, 2024 01:15 PM	✓
4	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0.25	Nov 17, 2024 02:45 PM	✗
5	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0.3	Nov 17, 2024 04:00 PM	✓

Image 28: Hide column function



Transaction History

Search

Found 20 results

Find column

ID
From Address
To Address
Amount (ETH)
Time Stamp
Valid

HIDE ALL SHOW ALL

From Address ↓	Amount (ETH)	Time Stamp	Valid
l7eD44aEBf5182e6f	0.1	Nov 17, 2024 10:00 AM	✓
l7eD44aEBf5182e6f	0.15	Nov 17, 2024 11:30 AM	✓
l7eD44aEBf5182e6f	0.2	Nov 17, 2024 01:15 PM	✓
l7eD44aEBf5182e6f	0.25	Nov 17, 2024 02:45 PM	✗
l7eD44aEBf5182e6f	0.3	Nov 17, 2024 04:00 PM	✓

Rows per page: 5 1–5 of 20

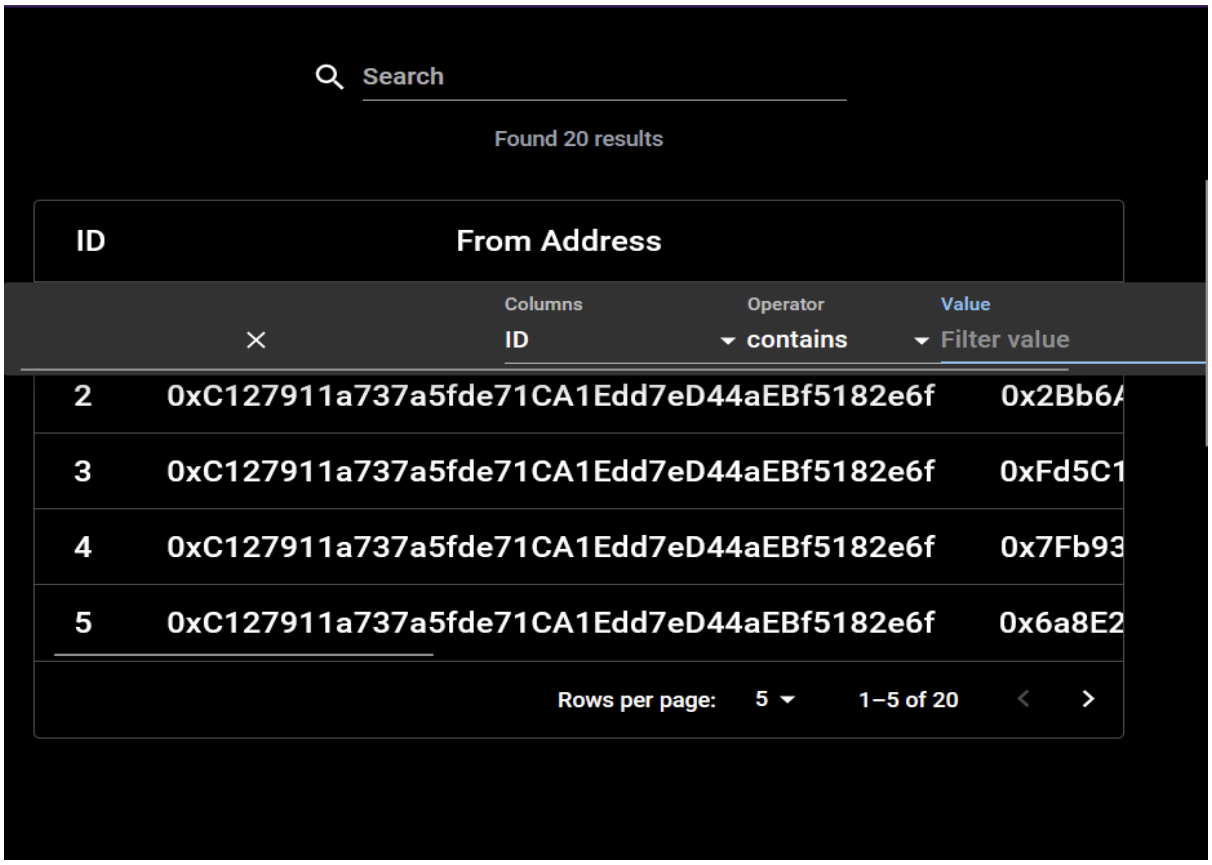
Image 29: Manage column section

6.4 Drawbacks and Improvement:

6.4.1 Drawback 1

Unable responsive when enabled Filter in Transaction History page.

Improvement: We will use other libraries that are more suitable for website responsiveness. Thereby giving users the most optimized experience possible.



Search

Found 20 results

Columns: ID | Operator: contains | Value: Filter value

ID	From Address	
2	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0x2Bb6
3	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0xFd5C1
4	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0x7Fb93
5	0xC127911a737a5fde71CA1Edd7eD44aEBf5182e6f	0x6a8E2

Rows per page: 5 1–5 of 20

Image 30: Error responsive Filter

6.4.2 Drawback 2

Unsuitable library horizontal scrolling animation.

Improvement: We will use other libraries to improve the lag effect. Users can also improve their web experience by accessing it with other browsers such as Firefox.

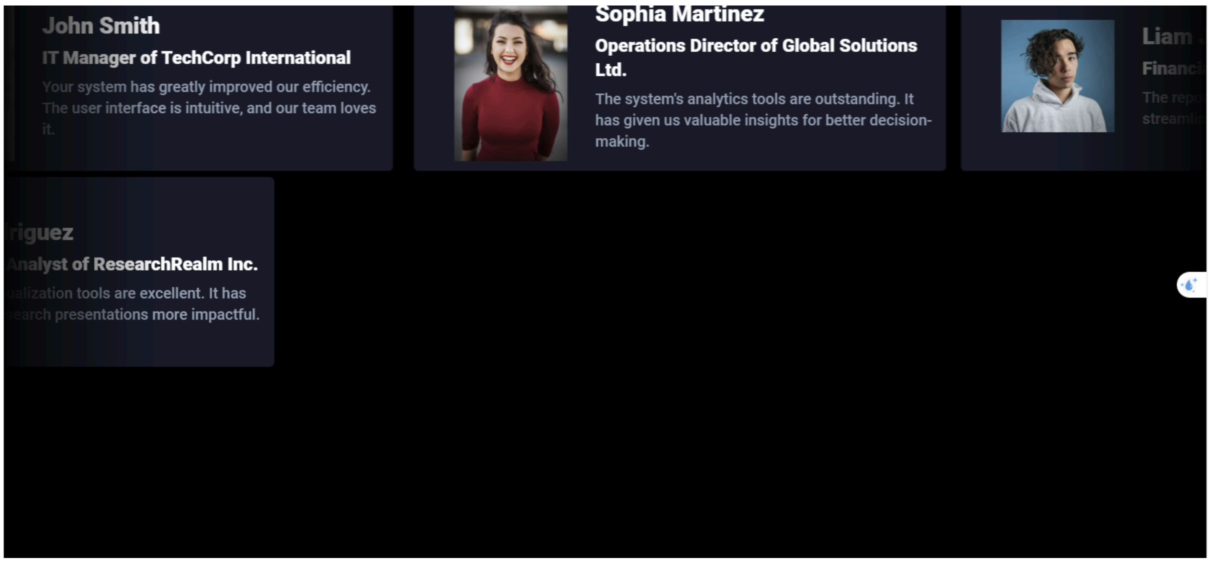

John Smith
IT Manager of TechCorp International
Your system has greatly improved our efficiency. The user interface is intuitive, and our team loves it.
Sophia Martinez
Operations Director of Global Solutions Ltd.
The system's analytics tools are outstanding. It has given us valuable insights for better decision-making.
Liam
Financ
riguez
Analyst of ResearchRealm Inc.
alization tools are excellent. It has
earch presentations more impactful.

Image 31: Lagging animation

6.4.3 Drawback 3

Slow API request to Infura, the more transaction on the system, the more time need to fetch on the page block

Improvement: Optimize API queries and use asynchronous requests to allow multiple requests to be processed simultaneously, thereby reducing the overall response time.

6.4.4 Drawback 4

Limitation of only 5 requests per second for the Sepolia testnet API. Exceeding the request limit may result in rejected requests or throttling.

Improvement: Upgrade to premium testnet API, or can optimize the data retrieval which involves batching multiple requests into a single call.

6.4.5 Drawback 5

The current reliance on a client-side framework introduces a potential drawback as it may need additional effort and resources to maintain compatibility.

Improvement: Develop a comprehensive migrations plan outlining the steps involved in transitioning from the existing client-side framework to NextJs.

7. Conclusion

Blockchain technology ensures decentralization, reducing the need for a central authority to control funds or manipulate trades. Blockchain technology ensures the security and immutability of transaction data, promoting trust and transparency. All transactions on a DTS are publicly viewable, fostering greater transparency in the trading process.

The frontend of DTS provides a user-friendly interface, allowing users to connect their crypto wallets, browse available assets, and initiate trades. DTS also integrates with various crypto wallets, allowing users to retain control of their private keys and manage their funds directly. Compared to centralized exchanges, DTS may offer a less feature-rich frontend experience, but ongoing development is continuously enhancing functionality.

Decentralized Trading Systems (DTS) are revolutionizing finance by enabling peer-to-peer trading without intermediaries. DTS utilizes smart contracts, self-executing code stored on the blockchain, to define trading rules and automatically match buy and sell orders according to pre-programmed algorithms. They often rely on liquidity pools, where users contribute their assets, creating a pool of funds available for trading.

8. Reference

Dmitry Efanov and Pavel Roschin (2018). The All-Pervasiveness of the Blockchain Technology. *Procedia Computer Science*, [online] 123, pp.116–121. doi:https://doi.org/10.1016/j.procs.2018.01.019.

GeeksforGeeks. (2020). *Features of Blockchain*. [online] Available at: https://www.geeksforgeeks.org/features-of-blockchain/.

Investopedia. (2024). *Decentralized Market Definition*. [online] Available at: https://www.investopedia.com/terms/d/decentralizedmarket.asp [Accessed 29 Jan. 2024].

Investopedia. (2024). *What Is Decentralized Finance (DeFi) and How Does It Work?*. [online] Available at:What Is Decentralized Finance (DeFi) and How Does It Work? (investopedia.com) [Accessed 1 Feb. 2024].

Investopedia. (2024).Blockchain Facts: What Is It, How It Works, and How It Can Be Used. [online] Available at:Blockchain Facts: What Is It, How It Works, and How It Can Be Used (investopedia.com) [Accessed 1 Feb. 2024].

Tardi, C. (2019). *Decentralized Market Definition*. [online] Investopedia. Available at: https://www.investopedia.com/terms/d/decentralizedmarket.asp.

Magas, J. (2019). *DEX, Explained*. [online] Cointelegraph. Available at: https://cointelegraph.com/explained/dex-explained.

Magas (n.d.). *What is a DEX*. [online] Coinbase Help. Available at: https://help.coinbase.com/en/coinbase/trading-and-funding/trade-on-dex/what-is-a-dex.