Chapter 1

Introduction

In today's rapidly evolving digital landscape, blockchain technology, and more specifically, the Ethereum blockchain, have emerged as revolutionary platforms with the potential to reshape traditional systems of finance and transaction processing. The concept of decentralized transactions, where transactions occur directly between users without the need for intermediaries, has garnered considerable attention and interest.

This project, titled "Decentralized Transaction Application", delves into the world of decentralized finance (DeFi) and smart contracts to explore the possibilities and advantages of conducting transactions securely and efficiently on the Ethereum blockchain. The significance of this project lies in its aim to develop a user-friendly application that harnesses the power of Ethereum's decentralized network to facilitate peer-to-peer transactions.

In an era marked by increasing concerns about data privacy, security, and trust, decentralized transactions offer a promising solution. By eliminating reliance on centralized institutions and intermediaries, decentralized applications (dApps) on Ethereum provide users with greater control over their assets and transactions. This project seeks to empower individuals by offering a seamless and secure means of conducting financial transactions within a trustless environment.

Throughout this documentation, we will delve into the objectives, scope, methodology, and technologies employed in the development of this decentralized transaction application. We will also explore the challenges encountered, the progress made, and the potential implications of this project, as we embark on a journey into the world of decentralized finance and Ethereum-powered transactions.

1.1 Background

Blockchain technology, pioneered by Bitcoin, has evolved beyond its initial use as a cryptocurrency and has found applications across various industries. One of the most notable developments in the blockchain space is Ethereum, a platform designed to enable the creation of decentralized applications (dApps) through the use of smart contracts.

Ethereum's smart contracts are self-executing agreements with the terms of the contract directly written into code. This innovation opened the door to a wide range of possibilities, including decentralized finance (DeFi), decentralized autonomous organizations (DAOs), and more. Ethereum's decentralized nature and robust security features have made it a key player in the blockchain ecosystem.

The concept of decentralized transactions represents a fundamental shift in how financial and data transactions are conducted. Traditionally, centralized intermediaries such as banks, payment processors, and clearinghouses have facilitated transactions, which has led to concerns about trust, security, and inefficiency. Decentralized transactions aim to address these issues by allowing users to interact directly with each other on a trustless and transparent network.

Key concepts and technologies underpinning this project include :

1. Ethereum Blockchain : Ethereum is a decentralized blockchain platform that allows developers to build and deploy smart contracts and dApps. It features a robust and secure network that ensures the integrity of transactions.

2. Smart Contracts : Smart contracts are self-executing contracts with predefined rules and conditions. They automatically enforce agreements without the need for intermediaries.

3. Web3 : Web3 is a set of libraries and protocols that enables the creation of decentralized applications that interact with the Ethereum blockchain. It provides the necessary tools for building user-friendly, blockchain-based applications.

4. Decentralized Finance (DeFi) : DeFi is a rapidly growing sector within the blockchain space that aims to recreate traditional financial services such as lending, borrowing, and trading on decentralized platforms. It relies heavily on Ethereum and smart contracts.

5. Security and Trustlessness : Decentralized transactions offer enhanced security and trust as they are recorded on an immutable blockchain, making fraud and manipulation extremely difficult.

This project seeks to leverage the power of Ethereum and the principles of decentralized transactions to develop a user-friendly application that empowers users to conduct secure and efficient financial transactions in a decentralized environment. By doing so, it addresses some of the challenges and limitations associated with centralized financial systems and opens up new possibilities for trustless peer-to-peer interactions.

1.2 Motivation

The motivation for creating the **"Decentralized Transaction Application"** project can be summarized as follows:

1. Empowerment : We aim to empower individuals by providing a decentralized platform that eliminates the need for traditional intermediaries, giving users direct control over their financial interactions.

2. Enhanced Security : The project is motivated by the desire to enhance security and privacy for users through blockchain's cryptographic security features and immutable ledger.

3. Financial Inclusion : We seek to contribute to financial inclusion efforts by offering a borderless and inclusive financial ecosystem accessible to anyone with an internet connection.

4. Technology Exploration : Our motivation includes the opportunity to gain hands-on experience with emerging technologies like smart contracts, Web3 development, and DeFi.**5.**

5. Pioneering Change : We aspire to be pioneers in transforming traditional financial systems into more transparent, efficient, and user-centric ecosystems.

1.3 Objectives

1. Application Development : Develop a user-friendly web application that allows users to initiate and complete decentralized transactions on the Ethereum blockchain.

2. Smart Contract Integration : Implement Ethereum smart contracts to facilitate secure and automated transaction processing, eliminating the need for traditional intermediaries.

3. User-Friendly Interface : Create an intuitive and accessible user interface (UI) that enables users, including those with minimal blockchain experience, to navigate and utilize the application seamlessly.

4. Enhanced Security : Ensure robust security measures within the application, including secure user authentication and encryption, to protect users' assets and data.

5. Decentralized Identity : Explore and implement decentralized identity solutions to enhance user privacy and control over their personal information.

6. Cross-Platform Compatibility : Ensure the application is compatible with a variety of devices and operating systems, promoting widespread accessibility.

7. Scalability : Design the application with scalability in mind to accommodate a growing user base and increasing transaction volume.

8. Performance Optimization : Optimize transaction processing speed and efficiency to provide users with a smooth and responsive experience.

9. Documentation and Education : Create comprehensive documentation and educational resources to guide users in using the application and understanding the benefits of decentralized transactions.

10. Testing and Quality Assurance : Conduct rigorous testing, including security audits and user testing, to identify and address any issues or vulnerabilities.

11. Community Engagement : Foster a supportive user community around the application, encouraging user feedback and continuous improvement.

1.4 Purpose, Scope & Applicability

1.4.1 Purpose

The **"Decentralized Transaction Application"** project is driven by a dual purpose. Firstly, it aims to empower individuals by providing a user-friendly platform for conducting decentralized transactions on the Ethereum blockchain. By eliminating traditional intermediaries, the project seeks to grant users greater control over their financial interactions, aligning with the principles of user autonomy and self-reliance inherent to blockchain technology.

Secondly, the project aspires to enhance security and trust in financial transactions. Traditional financial systems face ongoing challenges related to security breaches, fraud, and data manipulation. By leveraging blockchain's cryptographic security measures and its immutable ledger, the project seeks to establish a secure and transparent environment that inspires confidence and mitigates risks. Additionally, the project's overarching goal is to promote financial inclusion, explore emerging technologies, and serve as a catalyst for industry transformation by pioneering the adoption of decentralized transactions and applications. Through these objectives, it aims to shape a more equitable and user-centric financial landscape.

1.4.2 Scope

The scope of the **"Decentralized Transaction Application"** project is to develop a user-friendly web application for decentralized transactions on the Ethereum blockchain. This encompasses the creation, execution, and verification of secure and efficient peer-to-peer transactions. The project will integrate Ethereum smart contracts to automate transaction processes, reducing the reliance on intermediaries and ensuring transparency and security. It will also include the development of a user-friendly interface accessible across various devices and operating systems.

The project's focus extends to security, encompassing robust measures to protect user data, assets, and transactions, including secure authentication and encryption. Additionally, the application will undergo rigorous testing, including security audits and user testing, to ensure stability and reliability. Comprehensive documentation and educational resources will be provided to guide users and promote understanding of decentralized transactions. Community engagement efforts will foster a supportive user community, while compliance with relevant legal and regulatory frameworks will ensure transparency and user trust. In summary, the project's scope aims to deliver a secure, accessible, and innovative platform for decentralized transactions, empowering users to engage in peer-to-peer transactions on the Ethereum blockchain.

1.4.3 Applicability

The **"Decentralized Transaction Application"** project boasts versatile applicability across a spectrum of domains, driven by its potential to disrupt and enhance conventional financial systems. Foremost, this application is particularly relevant to individuals seeking a secure, efficient, and user-friendly platform for conducting financial transactions. By eliminating the need for intermediaries, it liberates users from the constraints of traditional financial institutions. Its accessibility accommodates users with varying levels of blockchain knowledge, making decentralized transactions accessible to a broader audience and catering to both novice and experienced users.

Secondly, the project holds significant promise in the context of financial inclusion. In regions where traditional banking services remain inaccessible, the application can act as a gateway to the global financial ecosystem. It provides a conduit for previously marginalized populations to partake in peer-to-peer transactions and access decentralized financial services. This inclusivity aligns with global efforts to reduce financial disparities and promote economic equity, making the application particularly relevant for underserved communities.

Furthermore, the **"Decentralized Transaction Application"** is an invaluable resource for blockchain enthusiasts and developers eager to explore emerging technologies. It serves as an educational and experimental platform, offering insights into smart contract development, Web3 interactions, and the dynamics of decentralized finance (DeFi). This educational aspect makes the project applicable in fostering blockchain expertise and contributes to the broader blockchain community's continuous evolution and innovation.

1.5 Significant Contribution

The **"Decentralized Transaction Application"** project is poised to make substantial contributions across several critical dimensions, each playing a pivotal role in reshaping the landscape of financial transactions and decentralized applications.

1. Empowering Financial Autonomy : One of the project's most significant contributions lies in empowering individuals. By providing a user-friendly platform for decentralized transactions, it liberates users from the reliance on traditional financial intermediaries. This shift towards user empowerment not only aligns with the core principles of blockchain but also grants users greater control over their financial interactions, fostering financial autonomy and self-reliance.

2. Enhancing Security and Trust : The project's implementation of robust security measures significantly contributes to enhancing the security and trustworthiness of financial transactions. In a world plagued by data breaches and fraud, the application's utilization of blockchain's cryptographic security and its immutable ledger establishes a more secure and transparent environment. This contribution directly addresses trust issues associated with conventional financial systems, instilling confidence among users.

3. Promoting Financial Inclusion : A paramount contribution is the promotion of financial inclusion. By offering a decentralized financial ecosystem accessible to anyone with an internet connection, the project plays a vital role in reducing financial disparities. It enables underserved populations to participate in peer-to-peer transactions, thus contributing to broader global efforts to achieve financial inclusivity and economic equity.

4. Exploring Emerging Technologies : The project's exploration and experimentation with emerging technologies, including smart contracts, Web3 development, and DeFi, contribute to the advancement of blockchain knowledge and capabilities. It serves as an educational resource for blockchain enthusiasts and developers, fostering innovation and driving progress within the broader blockchain community.

5. Leading Industry Transformation : On a broader scale, the project leads the transformation of the financial industry by pioneering the adoption of decentralized transactions and applications. It sets an example of how blockchain technology can make financial systems more transparent, efficient, and user-centric. This contribution serves as a catalyst for change within the financial sector and inspires further innovation and adoption.

1.6 Gantt Chart

	5/7/23	11/7/23	20/7/23	12/8/23	26/8/23	29/12/23	27/2/24
Topic selection	■						
Submission of Topic		■					
Studying Existing System			■				
Planning				■			
Analysis					■		
Design & Coding						■	
Documentation completion							■

Chapter 2

System Analysis

Systems analysis is the study of sets of interacting entities, including computer systems analysis. This field is closely related to requirement analysis or operations research. It is also "an explicit formal inquiry carried out to help someone (referred to as the decision maker). It identifies a better course of action and make a better decision than he might otherwise have made."

The development of a computer-based information system includes a systems analysis phase which produces or enhances data model which itself is a precursor to creating or enhancing a database.

There are a number of different approaches to system analysis. When a computer- based information system is developed, systems analysis would constitute the following steps:

- The development of a feasibility study, involving determining whether a project is economically, socially, technologically and organizationally feasible.
- Conducting fact-finding measures, designed to ascertain the requirements of the system's end-users. These typically span interviews, questionnaires, or visual observations of work on the existing system.

2.1 Existing System

The **"Decentralized Transaction Application"** project aims to improve upon the existing financial transaction systems, particularly those in the traditional centralized financial sector. Here is more detailed information about the existing system based on the project's context:

In the current financial landscape, traditional centralized financial systems dominate the scene, posing several challenges and limitations:

1. Intermediaries : Traditional financial transactions often require multiple intermediaries, such as banks, payment processors, and clearinghouses. These intermediaries add complexity, increase transaction costs, and introduce potential points of failure and delays.

2. Trust-Based Model : Users are reliant on centralized institutions and intermediaries to process and validate transactions. This trust-based model can be problematic, as it necessitates placing faith in third parties to execute transactions accurately and securely.

3. Security Vulnerabilities : Centralized systems store sensitive user data and transaction information on their servers, making them attractive targets for hackers and data breaches. Security incidents in centralized institutions can lead to unauthorized access, identity theft, and financial fraud.

4. Exclusionary Practices : Traditional financial systems can be exclusionary, requiring users to meet specific criteria or have a physical presence to access financial services. This results in limited financial inclusion, particularly for marginalized or underbanked populations.

5. Opaque Transactions : Lack of transparency is a significant concern in centralized systems. Users often have limited visibility into the transaction process, making it challenging to verify the authenticity of transactions and resolve disputes.

6. Geographic Barriers : Cross-border transactions within traditional financial systems are often slow, costly, and cumbersome. These challenges arise due to currency conversion, compliance requirements, and the involvement of multiple financial intermediaries.

7. Inefficiencies : Traditional financial systems can suffer from inefficiencies, such as extended settlement times and high transaction costs, which can hinder the overall efficiency of financial transactions.

The **"Decentralized Transaction Application"** project recognizes these limitations and aims to leverage blockchain technology, specifically the Ethereum blockchain, to create a decentralized system that overcomes these challenges. It seeks to eliminate intermediaries, enhance security through cryptography and decentralization, eliminate trust dependencies, promote financial inclusion by providing accessible financial services, ensure transaction transparency via the immutable blockchain ledger, and enable cross-border transactions in a seamless and cost-effective manner. In doing so, the project seeks to offer a more efficient, secure, and inclusive alternative to the existing centralized financial systems.

2.2 Proposed System

The **"Decentralized Transaction Application"** project's proposed system represents a groundbreaking shift towards decentralized financial transactions. It addresses the limitations of centralized financial systems by eliminating intermediaries, enhancing security, promoting trustless transactions, fostering financial inclusion, ensuring transaction transparency, streamlining cross-border transactions, and offering a user-centric experience. This proposed system aims to revolutionize the way financial transactions are conducted, making them more efficient, secure, and accessible to a global audience.

2.3 Requirement Analysis

Requirement analysis is a pivotal phase in the development of the **"Decentralized Transaction Application"** project. It involves a comprehensive examination of the project's functional, non-functional, and technical requirements to ensure that the system meets user expectations and operates effectively.

2.3.1 Functional Requirements

1. User Registration and Authentication : Users must be able to create accounts securely and authenticate their identities to access the application.

2. Transaction Initiation : Users should be able to initiate various types of decentralized transactions, including cryptocurrency transfers and asset swaps.

3. Smart Contract Integration : The application must seamlessly integrate with Ethereum smart contracts to automate and execute transactions based on predefined conditions.

4. Transaction History : Users should have access to a transaction history feature that displays the details of past transactions, including timestamps and transaction IDs.

5. User Interface (UI) : The UI must be intuitive and user-friendly, catering to users with varying levels of blockchain expertise.

6. Security Measures : Robust security features, such as encryption of user data and two-factor authentication, must be in place to protect user information and assets.

7. Documentation and Educational Resources : Comprehensive documentation and educational materials should be available to guide users, explaining the functionalities of the application and blockchain technology.

2.3.2 Non-Functional Requirements

1. Security : The system must prioritize security with measures to protect against hacking, data breaches, and unauthorized access.

2. Scalability : It should be capable of handling a growing user base and increased transaction volume without compromising performance.

3. Performance : The application should be responsive, with minimal transaction processing times, ensuring a seamless user experience.

4. Cross-Platform Compatibility : The application must be accessible across various devices and operating systems, promoting widespread usability.

5. Compliance : The system must adhere to relevant legal and regulatory frameworks, ensuring transparency and user trust.

6. Reliability : Users should have confidence in the system's reliability, with minimal downtime and robust error-handling mechanisms.

2.3.3 Technical Requirements

1. Blockchain Integration : The application must be compatible with the Ethereum blockchain, utilizing relevant APIs and libraries.

2. Database Management : An efficient database management system should be in place to store and retrieve transaction data.

3. Hosting Infrastructure : Reliable hosting infrastructure with adequate resources to support the system's scalability and performance requirements.

4. Encryption : Implementation of strong encryption protocols to secure user data and transactions.

5. Audit and Monitoring : Tools for auditing and monitoring system performance and security should be incorporated.

2.4 Hardware Requirement

- **Processor :** Quad-core processor or higher for smooth development.
- **RAM :** 8 GB RAM or more for running development tools and environments efficiently.
- **Storage :** At least 256 GB SSD for fast read/write operations during development.
- **Display :** A monitor with a resolution of 1920x1080 pixels or higher for comfortable coding and design work.
- **Internet :** High-speed internet connection for accessing blockchain data.

2.5 Software Requirement

1. Operating System :

- **Server OS :** Linux-based operating system (e.g., Ubuntu Server 20.04 LTS) for production servers (if self-hosted).
- **Development Workstations :** Windows 10, macOS, or Linux-based operating systems for developers' workstations.

2. Web Server :

- **Development Server :** Vite development server for local frontend development.
- **Production Server :** Netlify for hosting the production version of the web application, providing scalability and reliability.

3. Backend Framework :

- **Node.js :** JavaScript runtime environment for building the backend of the web application.
- **Express.js :** Node.js framework for developing robust and scalable web applications.

4. Frontend Technologies :

- **JavaScript Framework :** React.js for building the interactive user interface.
- **Styling :** Tailwind CSS for efficient and responsive styling of the web application.
- **Frontend Build Tool :** Vite for fast development and efficient production builds.

5. Blockchain Integration :

- **Ethereum APIs :** Integration with Alchemy Ethereum API for querying blockchain data and interacting with Ethereum smart contracts.
- **Metamask Integration :** Integration of Metamask for user authentication and authorization through Ethereum wallet addresses.
- **Testing Network :** Sepolia Test Network for testing Ethereum smart contracts and interactions.

6. Ethereum Smart Contract Development :

- **Hardhat :** Development environment for Ethereum smart contracts, including compiling, testing, and deploying contracts.

7. Security Tools and Libraries :

- **Cryptography Libraries :** Libraries for implementing encryption, hashing, and digital signatures to ensure data security.

8. Development Tools :

- **Integrated Development Environment (IDE) :** Visual Studio Code (VS Code) for coding and debugging.
- **Version Control :** Git repositories (e.g., GitHub, GitLab) for collaborative development and version control.

9. Ethereum Test Network :

- **Blockchain Testing :** Sepolia Test Network for testing Ethereum smart contracts and interactions.

2.6 Justification of selection of Technology

The selection of technology for the "Decentralized Transaction Application" project is driven by a careful consideration of factors that ensure efficiency, security, scalability, and user-friendliness. The technologies chosen align seamlessly with the project's objectives and requirements.

1. Ethereum Blockchain : Ethereum stands as the cornerstone of this project due to its established position as a leading blockchain platform for decentralized applications (dApps). Ethereum's robust and well-documented ecosystem offers the following justifications:

- Smart Contract Support : Ethereum's native support for smart contracts is paramount for automating and executing decentralized transactions securely.
- Developer Community : Ethereum boasts a vibrant and knowledgeable developer community, providing access to a wealth of resources and expertise.
- Interoperability : Ethereum's compatibility with various blockchain networks and token standards ensures seamless integration with existing blockchain ecosystems.

2. Node.js for Backend Development : Node.js offers numerous advantages that make it a strong choice for backend development:

- Non-blocking Architecture : Its non-blocking I/O architecture ensures high concurrency, ideal for handling multiple user transactions simultaneously.
- Scalability : Node.js is known for its scalability, accommodating increasing user loads without compromising performance.
- JavaScript : Node.js allows developers to use JavaScript on both the frontend and backend, streamlining development and fostering code reusability.

3. JavaScript (React) for Frontend Development : React, a JavaScript library, was chosen for frontend development due to the following justifications:

- Component-Based Architecture : React's component-based architecture simplifies UI development, making it modular, maintainable, and efficient.
- Rich Ecosystem : React's ecosystem includes a wealth of third-party libraries and resources that expedite frontend development.
- User Experience : React's focus on a responsive and engaging user experience aligns with the project's user-centric approach.

4. Solidity for Smart Contract Development : Solidity is a natural choice for writing Ethereum smart contracts, with justifications including:

Ethereum Compatibility : Solidity is specifically designed for Ethereum, ensuring seamless compatibility with the Ethereum Virtual Machine (EVM).

- Security : It incorporates security features to help developers write secure smart contracts, mitigating common vulnerabilities.
- Community Support : Solidity enjoys strong community support, offering readily available documentation and tools for development and testing.

Chapter 3

System Design

3.1 Module Division

1. User Authentication Module :

- Responsible for user authentication and authorization using Metamask.
- Manages user accounts and permissions for accessing the application.
- Ensures secure and trusted transactions by verifying user identities.

2. Transaction Processing Module :

- Handles the core functionality of creating, processing, and validating decentralized transactions.
- Manages the interaction between users and Ethereum smart contracts.
- Ensures the security and integrity of transactions.

3. Blockchain Interaction Module :

- Integrates with the Ethereum blockchain and interacts with Alchemy Ethereum API.
- Retrieves blockchain data, such as transaction history and contract information.
- Facilitates communication with the Ethereum network for transaction execution.

4. Frontend User Interface Module :

- Focuses on the design and development of the user interface (UI) using React.js and Tailwind CSS.

- Includes components for user interaction, transaction history display, and account management.
- Provides a user-friendly and responsive interface for seamless user experience.

5. Backend Server Module :

- Implements the backend server using Node.js and Express.js.
- Serves as an intermediary between the frontend and the Ethereum blockchain.
- Handles API requests, data validation, and business logic.

6. Smart Contract Integration Module :

- Manages the integration of Ethereum smart contracts into the application.
- Develops and deploys smart contracts for handling decentralized transactions.
- Ensures the security and reliability of smart contract interactions.

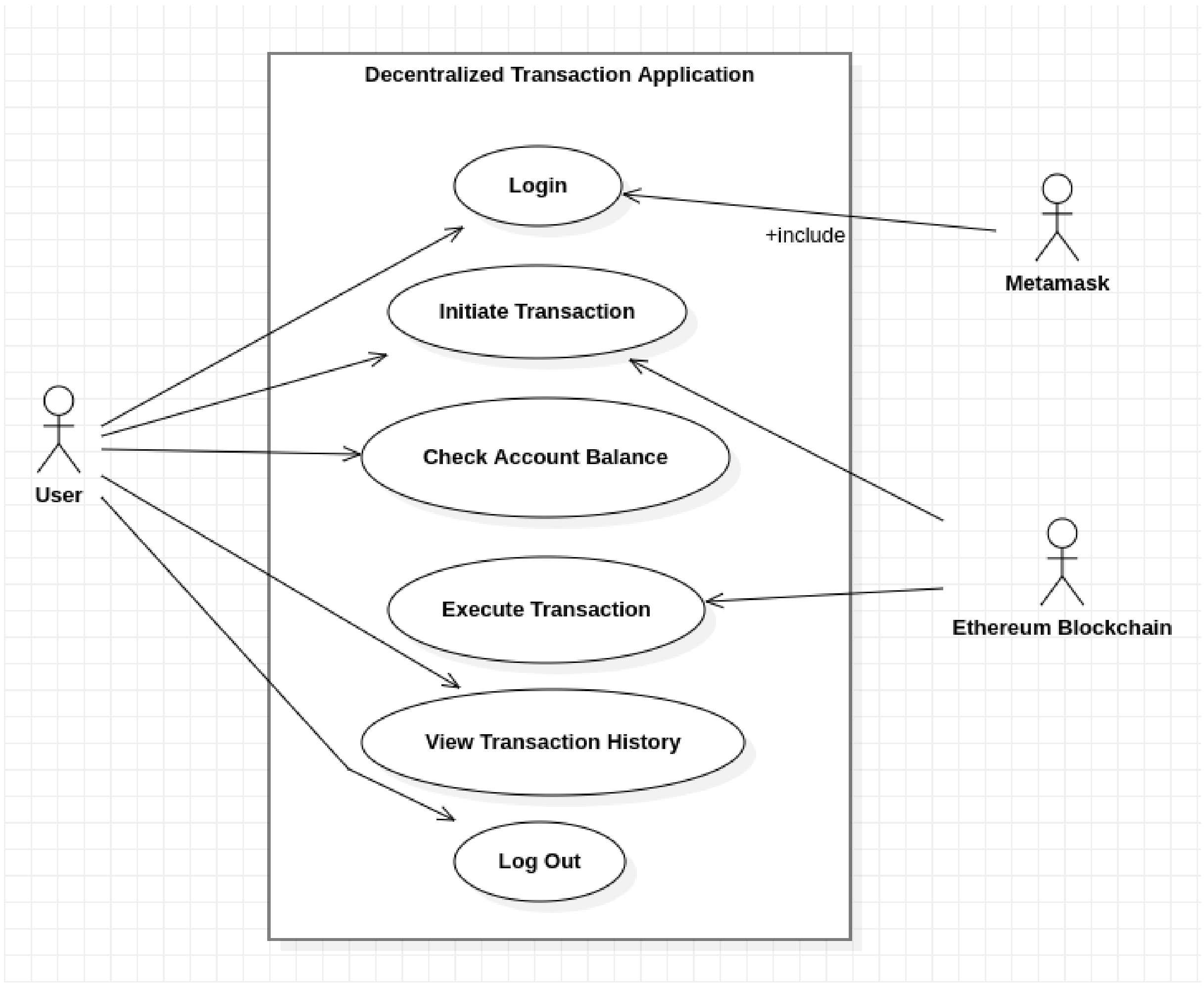
7. Testing and Quality Assurance Module :

- Focuses on testing strategies and methodologies for the entire application.
- Includes unit testing, integration testing, and end-to-end testing of components.
- Ensures the application functions correctly and meets quality standards.

8. Deployment and Hosting Module :

- Manages the deployment process using Netlify for the frontend and cloud-based servers for the backend.
- Sets up continuous integration/continuous deployment (CI/CD) pipelines.
- Ensures the application is accessible to users on the internet.

3.2 Use Case Diagram



1. Actors :

- User : Represents the end-users who interact with the application.
- Metamask : An external entity for authentication and transaction signing.
- Ethereum Blockchain : An external entity representing the Ethereum network.

2. Use Cases :

- Log In : Users can log into the application, which includes authentication and authorization.
- View Transaction History : Users can view their transaction history.
- Initiate Transaction : Users can initiate financial transactions.

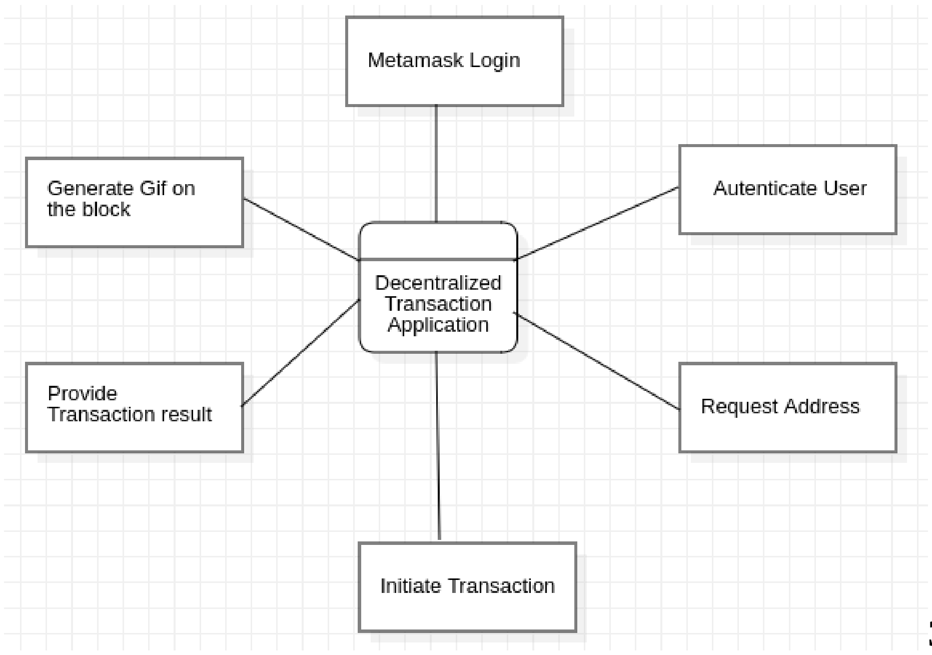
- Check Account Balance : Users can check their account balance.
- Log Out : Users can log out of the application.
- Execute Transaction : This use case represents the execution of a financial transaction, which involves interactions with the Ethereum blockchain.

3. Associations :

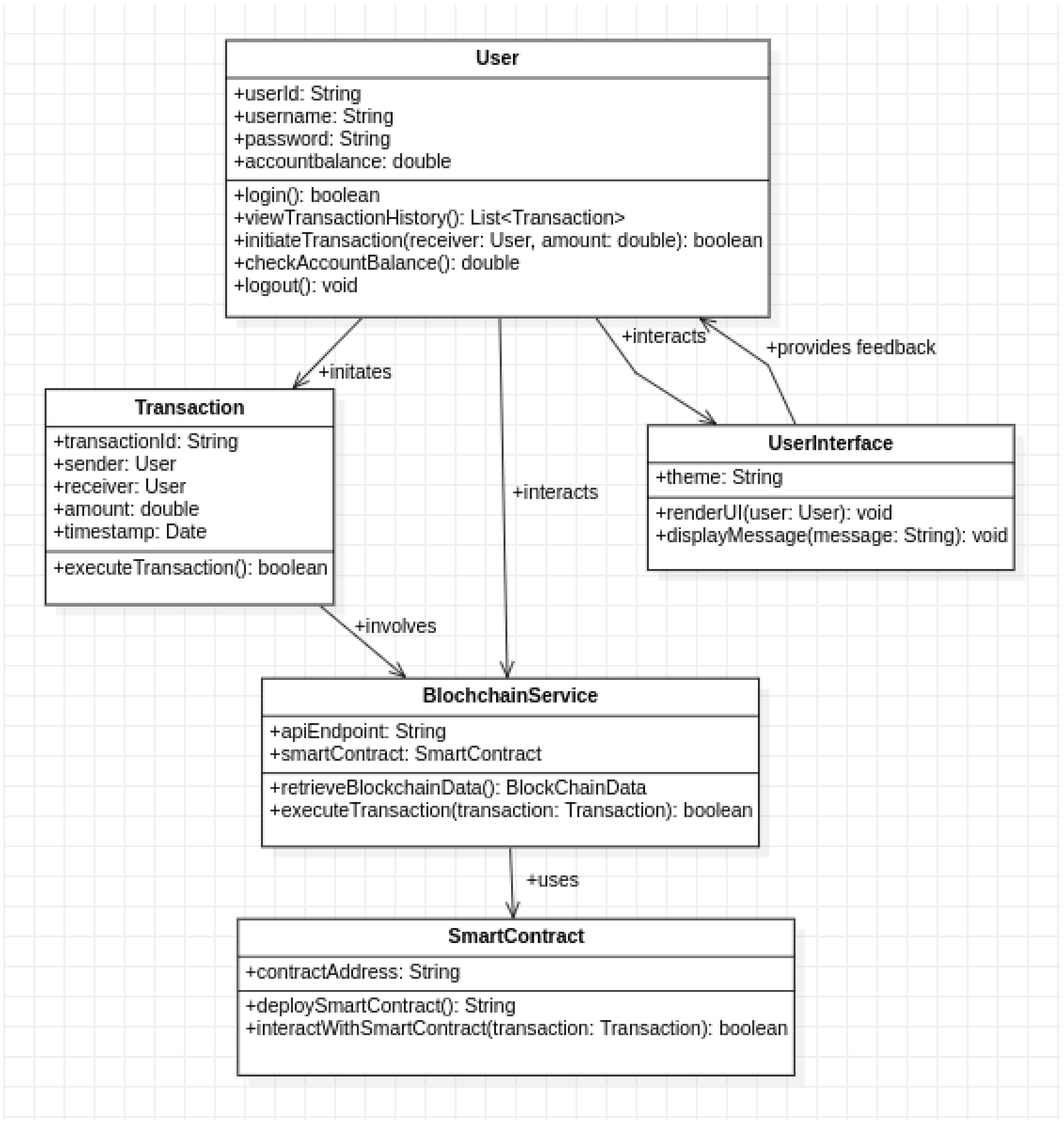
- Users interact with various use cases, such as logging in, viewing transaction history, initiating transactions, checking their balance, and logging out.
- Use cases like "Log In" include interactions with the external entity Metamask for authentication.

"Initiate Transaction" and "Execute Transaction" include interactions with the external entity Ethereum Blockchain for blockchain-based transaction execution.

3.3 Data Flow Diagram


Metamask Login
Generate Gif on the block
Autenticate User
Decentralized Transaction Application
Provide Transaction result
Request Address
Initiate Transaction

3.4 Class Diagram


User
+userId: String
+username: String
+password: String
+accountbalance: double
+login(): boolean
+viewTransactionHistory(): List<Transaction>
+initiateTransaction(receiver: User, amount: double): boolean
+checkAccountBalance(): double
+logout(): void
+initates
+interacts
+interacts
+provides feedback
Transaction
+transactionId: String
+sender: User
+receiver: User
+amount: double
+timestamp: Date
+executeTransaction(): boolean
UserInterface
+theme: String
+renderUI(user: User): void
+displayMessage(message: String): void
+involves
BlochchainService
+apiEndpoint: String
+smartContract: SmartContract
+retrieveBlockchainData(): BlockChainData
+executeTransaction(transaction: Transaction): boolean
+uses
SmartContract
+contractAddress: String
+deploySmartContract(): String
+interactWithSmartContract(transaction: Transaction): boolean

Classes :

1. User :

- This class represents the users of your application. It includes attributes like "userId", "username", "password", and "accountBalance". Users can log in, view their transaction history, initiate transactions, check their account balance, and log out. These methods allow users to interact with the application.

2. Transaction :

- The "Transaction" class represents financial transactions. It has attributes such as "transactionId", "sender", "receiver", "amount", and "timestamp". Transactions can be executed using the "executeTransaction" method.

3. BlockchainService :

- This class manages interactions with the blockchain and the Ethereum network. It has attributes like "apiEndpoint" and "smartContract". Methods in this class include "retrieveBlockchainData" for retrieving data from the blockchain and "executeTransaction" for executing transactions on the blockchain.

4. SmartContract :

- The "SmartContract" class is responsible for handling Ethereum smart contracts. It has an attribute "contractAddress". Methods include "deploySmartContract" for deploying smart contracts and "interactWithSmartContract" for interacting with smart contracts during transactions.

5. UserInterface :

- This class manages the user interface (UI) of your application. It has an attribute "theme" for UI customization. Methods like "renderUI" are used to display the UI to users, and "displayMessage" is used to provide feedback to users.

6. Metamask :

- This class represents an external entity, the Metamask browser extension, used for user authentication and transaction signing. It doesn't have specific attributes or methods in the application but plays a critical role in user authentication.

7. EthereumBlockchain :

- Another external entity, this class represents the Ethereum network. It is involved in blockchain transactions but doesn't have specific attributes or methods within the application.

Relationships :

- The "User" class is associated with the "Transaction" class to show that users initiate transactions.
- Users interact with the "UserInterface" class to use the application and receive feedback.
- The "User" class interacts with the "BlockchainService" class to perform various blockchain-related activities.
- The "Transaction" class involves the "BlockchainService" class for executing transactions.
- The "BlockchainService" class uses the "SmartContract" class for managing smart contracts on the blockchain.

Methods :

1. User :

- login() : This method allows a user to log into the application. It likely involves authentication and authorization checks.
- viewTransactionHistory() : Users can use this method to retrieve and view their transaction history.
- initiateTransaction(receiver : User, amount : double) : Users can initiate a transaction by specifying the receiver and the transaction amount.
- checkAccountBalance() : This method returns the current account balance of the user.
- logout() : Users can log out of the application using this method.

2. Transaction :

- executeTransaction() : This method executes the financial transaction. It could involve deducting funds from the sender's account and transferring them to the receiver. Transaction execution might also involve interacting with the blockchain.

3. BlockchainService :

- retrieveBlockchainData() : This method retrieves data from the blockchain. It's essential for fetching transaction-related information.
- executeTransaction(transaction : Transaction) : This method executes a transaction by interacting with the blockchain, ensuring the transaction is recorded securely.

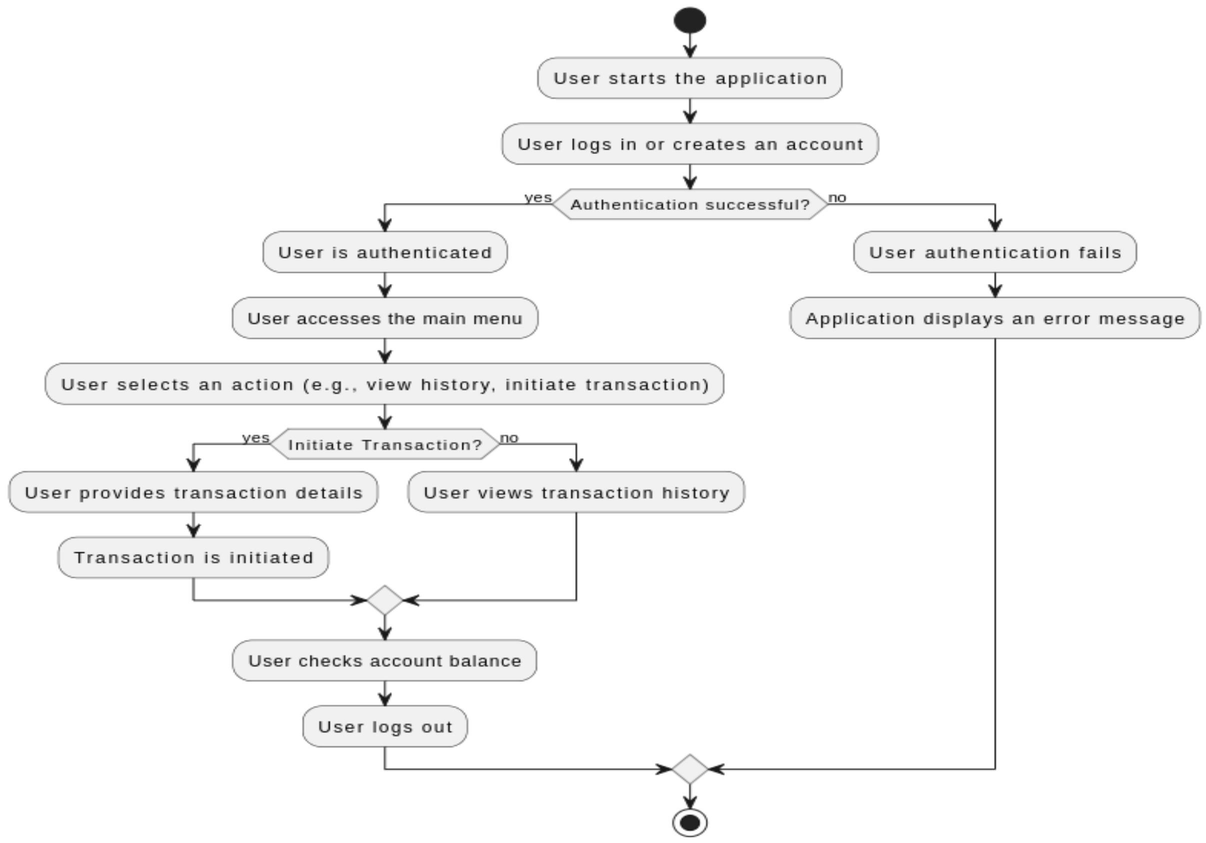
4. SmartContract :

- deploySmartContract() : This method deploys a smart contract on the blockchain, which can be used for various purposes like managing transactions.
- interactWithSmartContract(transaction : Transaction) : This method is responsible for interacting with a deployed smart contract to execute transactions on the blockchain.

5. UserInterface :

- renderUI(user : User) : This method renders the user interface based on the user's information. It's responsible for displaying the application's interface.
- displayMessage(message : String) : Users receive feedback and messages from the application through this method, which is crucial for user communication.

3.5 Activity Diagram


User starts the application
User logs in or creates an account
Authentication successful?
yes
no
User is authenticated
User authentication fails
User accesses the main menu
Application displays an error message
User selects an action (e.g., view history, initiate transaction)
Initiate Transaction?
yes
no
User provides transaction details
User views transaction history
Transaction is initiated
User checks account balance
User logs out

This Activity Diagram describes the following steps in the application's workflow :

1. The user starts the application.

2. The user can either log in or create an account.

3. If the user's authentication is successful, they are authenticated and gain access to the main menu.

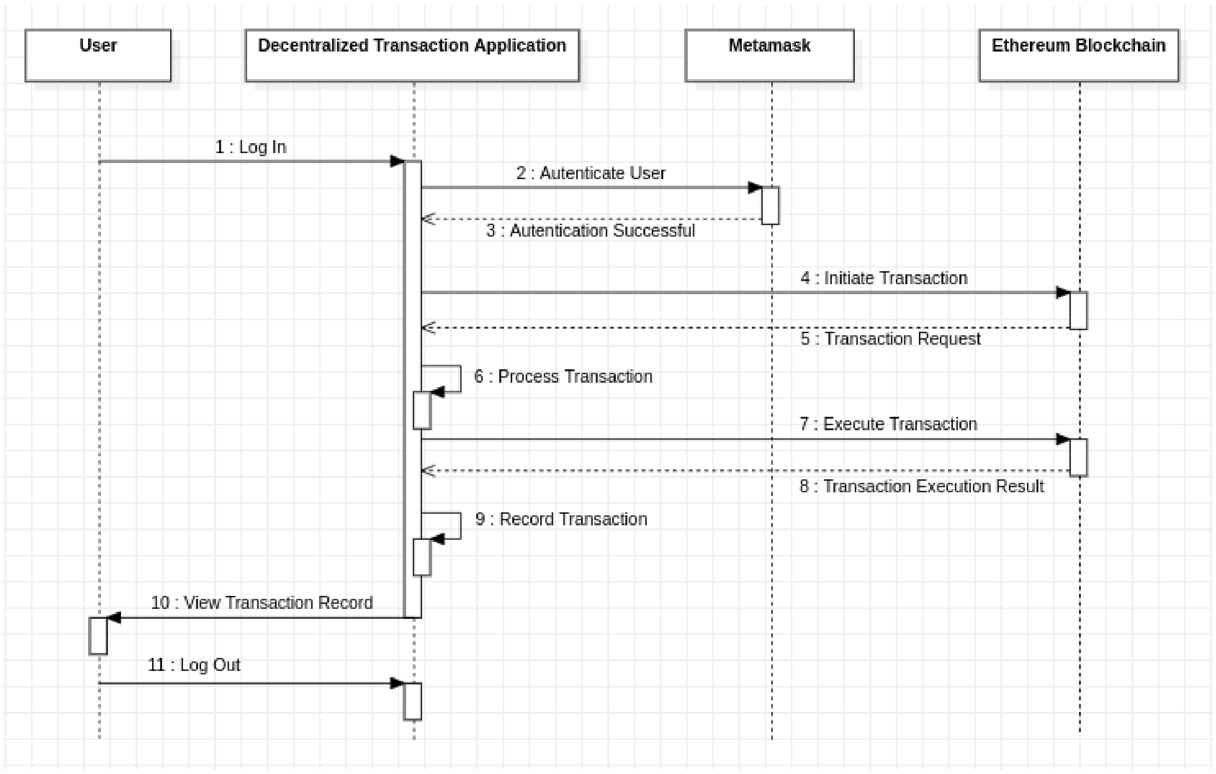
4. The user can then choose various actions, such as viewing transaction history, initiating a transaction, checking their account balance, or logging out.

5. If the user chooses to initiate a transaction, they provide the necessary transaction details, and the transaction is initiated.

6. The user can check their account balance.

7. Finally, the user logs out of the application.

3.6 Sequence Digram



In this Sequence Diagram :

1. The User starts the application and logs in.

2. The Application initiates a transaction.

3. The Application authenticates the user using Metamask, which interacts with the Ethereum Blockchain for authentication.

4. The Ethereum Blockchain processes the transaction request and sends the result back to the Application.

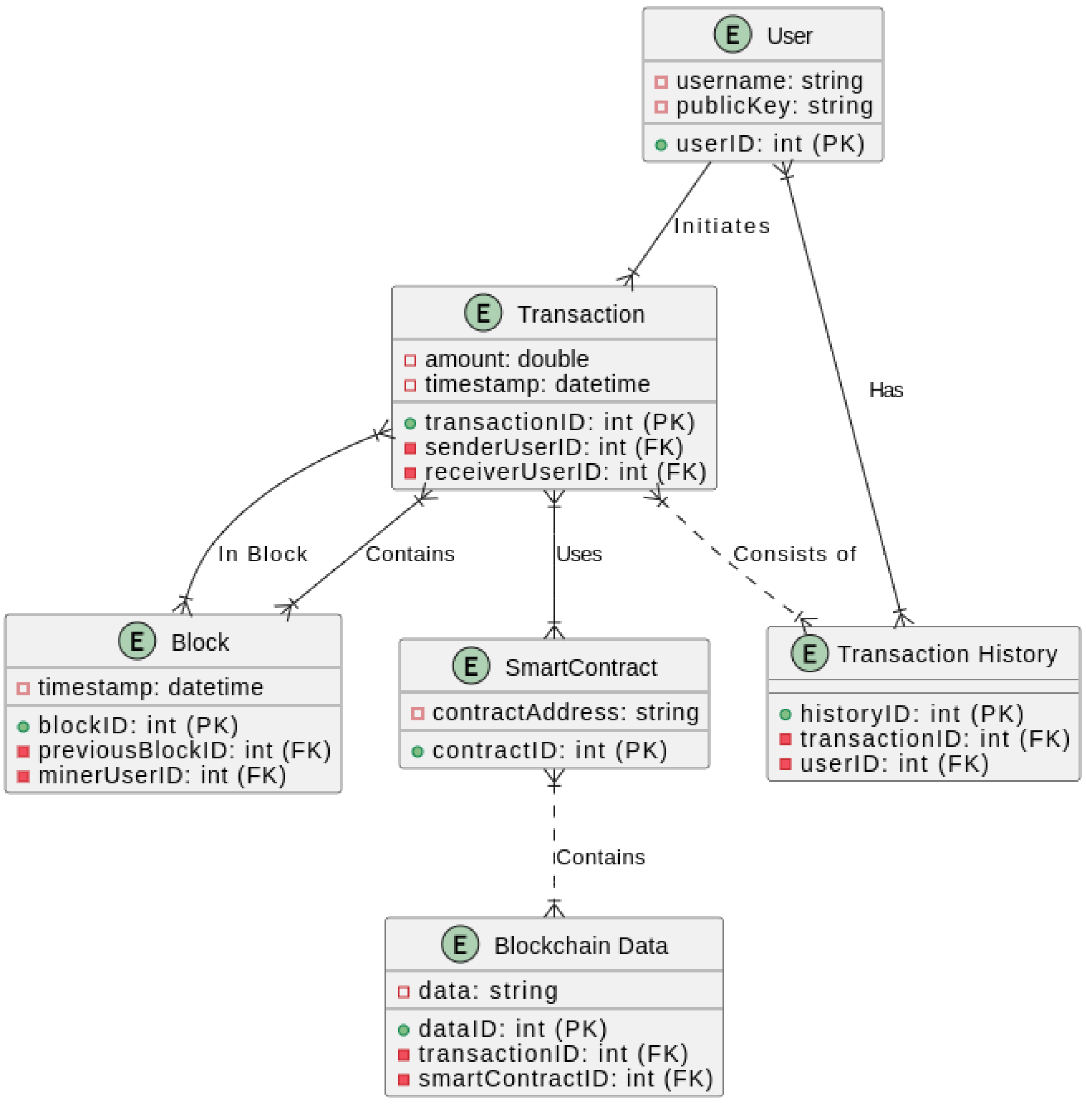
5. The Application executes the transaction, which involves interaction with the Ethereum Blockchain.

6. The Application records the transaction.

7. The User views the transaction result.

8. The User logs out.

3.7 ER Diagram



Entities :

1. User : Represents users of the application.

- Attributes :

- userID : A unique identifier for each user (Primary Key).

- username : The username of the user.

- publicKey : The public key associated with the user.

Users initiate transactions and have a role in transactions and transaction history.

2. Transaction : Represents financial transactions.

- Attributes :

- transactionID : A unique identifier for each transaction (Primary Key).

- senderUserID : A foreign key referencing the user initiating the transaction.

- receiverUserID : A foreign key referencing the user receiving the transaction.

- amount : The transaction amount.

- timestamp : The timestamp of the transaction.

Transactions are related to users and blocks.

3. Block : Represents blocks in the blockchain.

- Attributes :

- blockID : A unique identifier for each block (Primary Key).

- previousBlockID : A foreign key referencing the previous block in the blockchain.

- minerUserID : A foreign key referencing the user responsible for mining the block.

- timestamp : The timestamp of the block creation.

Blocks contain transactions and are connected in a chain.

4. SmartContract : Represents smart contracts used for transactions.

- Attributes :

- contractID : A unique identifier for each smart contract (Primary Key).

- contractAddress : The address of the smart contract on the blockchain.

Smart contracts can be used in transactions.

5. Transaction History : Stores the history of user transactions.

- Attributes :

- historyID : A unique identifier for each transaction history entry (Primary Key).

- transactionID : A foreign key referencing the associated transaction.

- userID : A foreign key referencing the user involved in the transaction.

Transaction history entries are linked to users and transactions.

6. Blockchain Data : Represents data stored on the blockchain related to transactions.

- Attributes :

- dataID : A unique identifier for each data entry (Primary Key).

- transactionID : A foreign key referencing the related transaction.

- smartContractID : A foreign key referencing the smart contract associated with the data.

- data : The actual data stored on the blockchain.

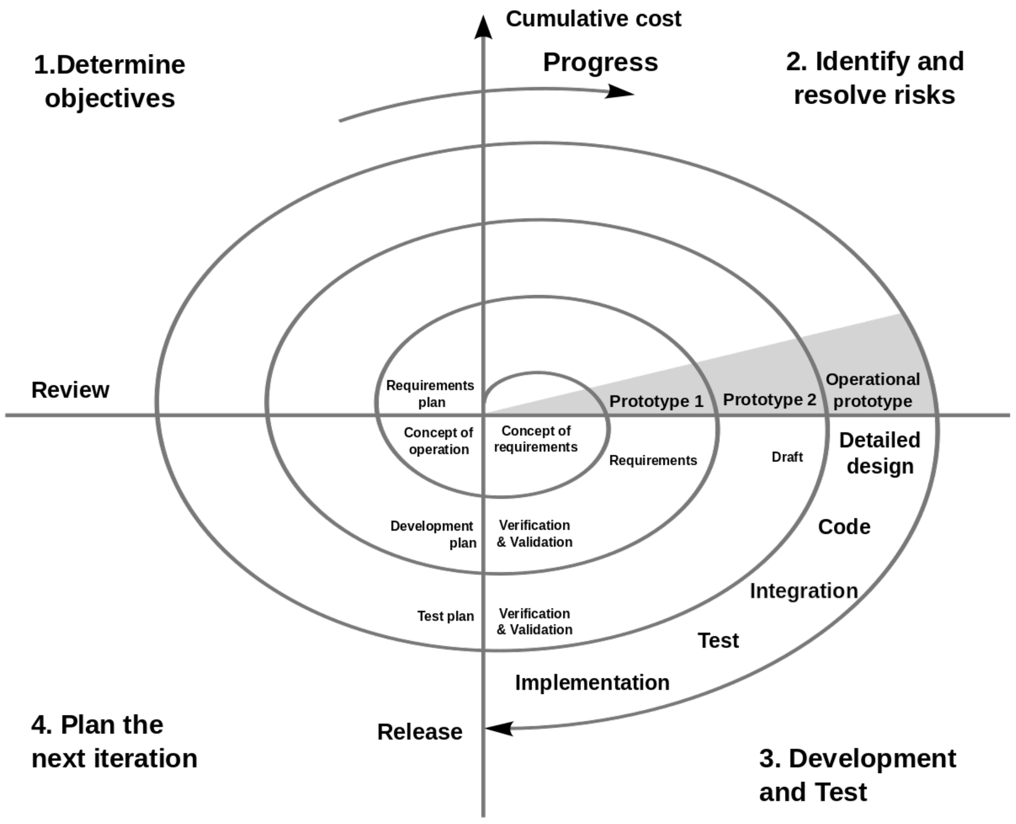
Blockchain data entries are connected to transactions and smart contracts.

Relationships :

- User initiates Transaction : Users initiate transactions, creating a relationship between the User entity and the Transaction entity.
- Transaction involves User (for both sender and receiver) : Transactions involve users, creating relationships between the Transaction entity and the User entity for both the sender and receiver.
- Transaction is contained within Block : Transactions are contained within blocks, forming a relationship between the Transaction and Block entities.
- Block forms a chain with the previous Block : Blocks in the blockchain are linked to the previous block through a relationship, forming a chain of blocks.

- Transaction uses SmartContract : Transactions can use smart contracts, creating a relationship between the Transaction and SmartContract entities.
- SmartContract contains Blockchain Data : Smart contracts can contain data stored on the blockchain, creating a relationship between the SmartContract and Blockchain Data entities.

3.8 Spiral Model



Spiral Cycle 1 : Project Initiation and Planning

Objective : Define project objectives, stakeholders, and high-level requirements.

Activities:

- Identify project goals and key stakeholders.
- Conduct a preliminary analysis of the decentralized transaction concept.
- Define initial project scope and requirements.
- Create a high-level project plan and schedule.
- Assess the feasibility of using blockchain technology.

Spiral Cycle 2 : Risk Analysis and Requirements Refinement

Objective : Analyze risks, refine requirements, and evaluate the feasibility.

Activities :

- Conduct an in-depth risk analysis, particularly related to blockchain and security.
- Refine project requirements based on insights from the first cycle.
- Validate the feasibility of implementing decentralized transactions.
- Prepare a detailed project plan with milestones for the next cycle.

Spiral Cycle 3 : Design and Prototyping

Objective : Create a prototype of the "Decentralized Transaction Application."

Activities:

- Develop a proof of concept (PoC) for the application.
- Design the application's user interface and transaction management.
- Build a prototype for blockchain communication and data storage.
- Test the prototype to ensure basic functionality.

Spiral Cycle 4 : Development and Testing

Objective : Develop and test the application incrementally.

Activities :

- Begin the application's development based on the PoC.
- Conduct continuous testing and debugging.
- Implement security measures to protect user transactions.
- Seek feedback from users and stakeholders.

Spiral Cycle 5 : User Feedback and Refinement

Objective : Gather user feedback and refine the application.

Activities :

- Collect feedback from users and stakeholders.
- Identify areas for improvement and enhancement.
- Refine the application based on feedback.
- Continue testing and quality assurance.

Spiral Cycle 6 : Deployment and Evaluation

Objective : Deploy the application and evaluate its performance.

Activities :

- Deploy the application to a production environment.
- Monitor its performance in a real-world setting.
- Evaluate user satisfaction and transaction success.
- Plan for maintenance and future enhancements.

Chapter 4

Implementation & Testing

4.1 Implementation Approach

The implementation approach for the "Decentralized Transaction Application" involves leveraging blockchain technology, specifically the Ethereum blockchain, to develop a decentralized system for financial transactions. The approach consists of the following key steps:

1. Requirements Analysis and Feasibility Study:

- Conduct a detailed analysis of project requirements, including functional, non-functional, and technical requirements.
- Evaluate the feasibility of implementing a decentralized transaction system using blockchain technology, considering factors such as scalability, security, and user experience.

2. Technology Selection:

- Choose appropriate technologies and frameworks aligned with the project's objectives and requirements.
- Select Ethereum as the primary blockchain platform due to its support for smart contracts and widespread adoption in decentralized applications.

3. Architecture Design:

- Design the system architecture, including frontend and backend components, smart contracts, and blockchain integration.

- Define the data model and data flow within the application, ensuring seamless interaction between different modules.

4. Development Iterations:

- Adopt an iterative development approach, dividing the development process into manageable iterations or sprints.
- Implement core functionalities incrementally, starting with user authentication, transaction processing, and smart contract integration.

5. Code Quality Assurance:

- Enforce coding standards, best practices, and code reviews to maintain code quality and consistency.
- Implement automated testing practices, including unit testing, integration testing, and end-to-end testing, to ensure the reliability and robustness of the application.

6. Security Measures:

- Implement security measures such as encryption, authentication, and authorization to protect user data and transactions.
- Conduct security audits and vulnerability assessments to identify and address potential security threats.

7. Deployment and Continuous Integration:

- Deploy the application in stages, starting with a test environment for internal testing and validation.
- Utilize continuous integration/continuous deployment (CI/CD) pipelines to automate the deployment process and ensure smooth deployment of updates and enhancements.

4.2 Coding Details

4.2.1 Client Folder

1. App.jsx

```
import {
  Welcome,
  EthereumChart,
  Footer,
  AboutCard,
  Transactions,
  Cursor,
} from "./components";

const App = () => (
  <div className="min-h-screen">
    <div className="gradient-bg-welcome">
      <Cursor />
      <Welcome />
      <EthereumChart />
    </div>
    <AboutCard />
    <Transactions />
    <Footer />
  </div>
);

export default App;
```

2. main.jsx

```
import React from "react";
import ReactDOM from "react-dom";

import App from "./App";
import { TransactionsProvider } from "./context/TransactionContext";
import "./index.css";

ReactDOM.render(
  <TransactionsProvider>
    <App />
  </TransactionsProvider>,
  document.getElementById("root"),
);
```

3. index.css

```
@import url("https://fonts.googleapis.com/css2?family=Open+Sans:wght@300;400;500;600;700&display=swap");

:root {
  --scrollbar-primary: #000000;
  --scrollbar-secondary: #f6e27a;
}

* {
  padding: 0;
  margin: 0;
  scroll-behavior: smooth;
  cursor: none;
  box-sizing: border-box;
}
```

```
/* Firefox */
* {
  scrollbar-width: thin;
  scrollbar-color: var(--scrollbar-secondary) var(--scrollbar-primary);
  cursor: none;
}

/* Chrome, Edge, and Safari */
*::-webkit-scrollbar {
  width: 15px;
}

*::-webkit-scrollbar-track {
  background: var(--scrollbar-primary);
  border-radius: 0px;
}

*::-webkit-scrollbar-thumb {
  background-color: var(--scrollbar-secondary);
  border-radius: 14px;
  border: 3px solid var(--scrollbar-primary);
}

body {
  margin: 0;
  font-family: "Open Sans", sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

.gradient-bg-welcome {
  background-color: #0f0e13;
  background-image: radial-gradient(
      at 0% 0%,
      rgb(109, 58, 17) 0,
      transparent 50%
    ),
    radial-gradient(at 50% 0%, rgba(255, 255, 255, 0.575) 0, transparent 50%),
    radial-gradient(at 100% 0%, rgba(66, 6, 71, 0.493) 0, transparent 50%);
}

.gradient-bg-aboutcard {
  background-color: #0f0e13;
  background-image: radial-gradient(
      at 0% 0%,
      hsla(253, 16%, 7%, 1) 0,
      transparent 50%
    ),
    radial-gradient(at 50% 100%, rgba(253, 238, 102, 0.459) 0, transparent 50%);
}

.gradient-bg-transactions {
  background-color: #0f0e13;
  background-image: radial-gradient(
      at 0% 100%,
      hsla(253, 16%, 7%, 1) 0,
      transparent 50%
    ),
    radial-gradient(at 50% 0%, rgba(253, 238, 102, 0.459) 0, transparent 50%);
}

.gradient-bg-footer {
  background-color: #0f0e13;
  background-image: radial-gradient(
      at 0% 100%,
      hsla(253, 16%, 7%, 1) 0,
      transparent 53%
    ),
    radial-gradient(at 50% 150%, rgba(255, 0, 0, 0.555) 0, transparent 50%);
}

.blue-glassmorphism {
  background: rgba(0, 0, 0, 0.178);
  border-radius: 16px;
  box-shadow: 0px 0px 30px rgba(255, 255, 255, 0.493);
  backdrop-filter: blur(10px);
  -webkit-backdrop-filter: blur(5px);
  border: 1px solid rgba(240, 233, 233, 0.3);

  opacity: 0;
  animation: slideLeft 1s ease forwards;
  animation-delay: 1.3s;
}

.features {
  opacity: 0;
  animation: zoomIn 1s ease forwards;
  animation-delay: 2s;
}

span {
  transform: translate(-50%, -50%);
  top: 50%;
  left: 50%;
  font-family: "Times New Roman", serif;
  letter-spacing: 5px;
  font-size: 130px;
  font-weight: bold;
  background: linear-gradient(
    to right,
    #462523bd 0,
    #cb9b51 22%,
    #f6e27a 45%,
    #f5ed7f 50%,
    #f6e27a 55%,
    #cb9b51 78%,
    #462523bd 100%
  );
  color: transparent;
  -webkit-background-clip: text;
}

.Ether-back {
  text-shadow: 5px 2px 10px #000000;
}

.border-clr {
  border-color: #cb9b51;
}

.btn-box {
  opacity: 0;
  animation: zoomIn 1s ease forwards;
  animation-delay: 0.6s;
```

```
}

.btn-box:hover {
  box-shadow: 0 0 5px #daa455e8, 0 0 25px #daa455e8, 0 0 50px #daa455e8,
    0 0 100px #daa455e8, 0 0 200px #daa455e8;
}

.dec-ani {
  opacity: 0;
  animation: slideRight 1s ease forwards;
  animation-delay: 0.7s;
}

.tag-ani {
  opacity: 0;
  animation: slideLeft 1s ease forwards;
  animation-delay: 0.7s;
}

/* white glassmorphism */
.white-glassmorphism {
  background: rgba(255, 255, 255, 0.05);
  border-radius: 16px;
  backdrop-filter: blur(5px);
  -webkit-backdrop-filter: blur(5px);
  border: 1px solid rgba(255, 255, 255, 0.3);
  width: 90%;
  height: auto;

  opacity: 0;
  animation: zoomIn 1s ease forwards;
  animation-delay: 1.3s;
}

.eth-card {
  background-image: linear-gradient(
    to right,
    #462523bd 0,
    #cb9b51 22%,
    #f6e27a 45%,
    #f5ed7f 50%,
    #f6e27a 55%,
    #cb9b51 78%,
    #462523bd 100%
  );
  color: transparent;

  opacity: 0;
  animation: zoomIn 1s ease forwards, floatImage 4s ease-in-out infinite;
  animation-delay: 0.6s, 2.1s;
}

.text-gradient {
  background-color: #fff;
  background-image: radial-gradient(
      at 4% 36%,
      rgb(255, 255, 255) 0,
      transparent 53%
    ),
    radial-gradient(at 100% 60%, rgb(253, 255, 114) 0, transparent 50%);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
```

```
}

.border-company {
  border-color: #cb9b51;
}

.rotate-3d {
  animation: rotate 3s infinite linear;
  transform-origin: center;
}

.about {
  margin-top: 5px;
  margin-bottom: 5px;
}

.about-img {
  border-width: 1px;
  /* border-color: rgba(0, 0, 0, 0.644); */
  border-color: rgba(240, 233, 233, 0.3);
}

.about:hover {
  transform: scale(6); /* Adjust the scale factor as needed */
  transition: transform 0.3s ease; /* Add a smooth transition effect */
  box-shadow: 0 0 5px rgba(240, 233, 233, 0.3),
    0 0 25px rgba(240, 233, 233, 0.3), 0 0 50px rgba(240, 233, 233, 0.3),
    0 0 100px rgba(240, 233, 233, 0.3), 0 0 200px rgba(240, 233, 233, 0.3);
}

.transaction {
  border-radius: 10px;
  background: #050505c7;
  box-shadow: inset 20px 20px 67px #040404, inset -20px -20px 67px #060606;
}

@keyframes rotate {
  0% {
    transform: rotateY(0deg);
  }
  100% {
    transform: rotateY(360deg);
  }
}

@keyframes slideTop {
  0% {
    opacity: 0;
    transform: translateY(100px);
  }

  100% {
    opacity: 1;
    transform: translateY(0);
  }
}

@keyframes slideRight {
  0% {
    opacity: 0;
    transform: translateX(-100px);
  }
```

```
 100% {
   opacity: 1;
   transform: translateX(0);
 }
}

@keyframes slideLeft {
 0% {
   opacity: 0;
   transform: translateX(100px);
 }

 100% {
   opacity: 1;
   transform: translateX(0);
 }
}

@keyframes zoomIn {
 0% {
   opacity: 0;
   transform: scale(0);
 }

 100% {
   opacity: 1;
   transform: scale(1);
 }
}
```

```
@keyframes floatImage {
  0% {
    transform: translateY(0);
  }

  50% {
    transform: translateY(-24px);
  }

  100% {
    transform: translate(0);
  }
}

@keyframes circleRotate {
  0% {
    transform: rotate(0);
  }

  100% {
    transform: rotate(360deg);
  }
}

@keyframes fadeIn {
  0% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
}

@tailwind base;
@tailwind components;
@tailwind utilities;
```

- **Client Folder – context**

1. TransactionContext.jsx

```
import React, { useEffect, useState } from "react";
import { ethers } from "ethers";

import { contractABI, contractAddress } from "../utils/constants";

export const TransactionContext = React.createContext();

const { ethereum } = window;

const createEthereumContract = () => {
  const provider = new ethers.providers.Web3Provider(ethereum);
  const signer = provider.getSigner();
  const transactionsContract = new ethers.Contract(contractAddress, contractABI, signer);

  return transactionsContract;
};

export const TransactionsProvider = ({ children }) => {
  const [formData, setformData] = useState({ addressTo: "", amount: "", keyword: "", message: "" });
  const [currentAccount, setCurrentAccount] = useState("");
  const [isLoading, setIsLoading] = useState(false);
```

```
    const [transactionCount, setTransactionCount] = useState(localStorage.getItem("transactionCount"));
  const [transactions, setTransactions] = useState([]);

  const handleChange = (e, name) => {
    setformData((prevState) => ({ ...prevState, [name]: e.target.value }));
  };

  const getAllTransactions = async () => {
    try {
      if (ethereum) {
        const transactionsContract = createEthereumContract();

        const availableTransactions = await transactionsContract.getAllTransactions();

        const structuredTransactions = availableTransactions.map((transaction) => ({
          addressTo: transaction.receiver,
          addressFrom: transaction.sender,
          timestamp: new Date(transaction.timestamp.toNumber() * 1000).toLocaleString(),
          message: transaction.message,
          keyword: transaction.keyword,
          amount: parseInt(transaction.amount._hex) / (10 ** 18)
        }));

        console.log(structuredTransactions);

        setTransactions(structuredTransactions);
      } else {
        console.log("Ethereum is not present");
      }
    } catch (error) {
```

```
    console.log(error);
  }
};

const checkIfWalletIsConnect = async () => {
  try {
    if (!ethereum) return alert("Please install MetaMask.");

    const accounts = await ethereum.request({ method: "eth_accounts" });

    if (accounts.length) {
      setCurrentAccount(accounts[0]);

      getAllTransactions();
    } else {
      console.log("No accounts found");
    }
  } catch (error) {
    console.log(error);
  }
};

const checkIfTransactionsExists = async () => {
  try {
    if (ethereum) {
      const transactionsContract = createEthereumContract();
      const currentTransactionCount = await transactionsContract.getTransactionCount();

      window.localStorage.setItem("transactionCount", currentTransactionCount);
    }
  } catch (error) {
    console.log(error);

    throw new Error("No ethereum object");
  }
};

const connectWallet = async () => {
  try {
    if (!ethereum) return alert("Please install MetaMask.");

    const accounts = await ethereum.request({ method: "eth_requestAccounts", });

    setCurrentAccount(accounts[0]);
    window.location.reload();
  } catch (error) {
    console.log(error);

    throw new Error("No ethereum object");
  }
};

const sendTransaction = async () => {
  try {
    if (ethereum) {
      const { addressTo, amount, keyword, message } = formData;
      const transactionsContract = createEthereumContract();
      const parsedAmount = ethers.utils.parseEther(amount);

      await ethereum.request({
        method: "eth_sendTransaction",
        params: [{
```

```
        from: currentAccount,
        to: addressTo,
        gas: "0x5208",
        value: parsedAmount._hex,
      }],
    });

          const transactionHash = await transactionsContract.addToBlockchain(addressTo,
parsedAmount, message, keyword);

    setIsLoading(true);
    console.log(`Loading - ${transactionHash.hash}`);
    await transactionHash.wait();
    console.log(`Success - ${transactionHash.hash}`);
    setIsLoading(false);

    const transactionsCount = await transactionsContract.getTransactionCount();

    setTransactionCount(transactionsCount.toNumber());
    window.location.reload();
   } else {
    console.log("No ethereum object");
   }
  } catch (error) {
   console.log(error);

   throw new Error("No ethereum object");
  }
 };

 useEffect(() => {
    checkIfWalletIsConnect();
    checkIfTransactionsExists();
  }, [transactionCount]);

  return (
    <TransactionContext.Provider
      value={{
        transactionCount,
        connectWallet,
        transactions,
        currentAccount,
        isLoading,
        sendTransaction,
        handleChange,
        formData,
      }}
    >
      {children}
    </TransactionContext.Provider>
  );
};
```

- **Client Folder – hooks**

1. useFetch.jsx

```
import { useEffect, useState } from "react";

const APIKEY = import.meta.env.VITE_GIPHY_API;

const useFetch = ({ keyword }) => {
  const [gifUrl, setGifUrl] = useState("");
  const fetchGifs = async () => {
    try {
      const response = await fetch(`https://api.giphy.com/v1/gifs/search?api_key=${APIKEY}&q=$
{keyword.split(" ").join("")}&limit=1`);
      const { data } = await response.json();

      setGifUrl(data[0]?.images?.downsized_medium.url);
    } catch (error) {
      setGifUrl("https://metro.co.uk/wp-content/uploads/2015/05/pokemon_crying.gif?
quality=90&strip=all&zoom=1&resize=500%2C284");
    }
  };

  useEffect(() => {
    if (keyword) fetchGifs();
  }, [keyword]);

  return gifUrl;
};

export default useFetch;
```

- **Client Folder - components**

1. About.jsx

```
import React from "react";
import { saket, sai, ether } from "../assets";

const About = ({ color, title, image, subtitle1, subtitle2 }) => (
   <div className="flex flex-row justify-start items-start white-glassmorphism p-3 m-2 cursor-
pointer hover:shadow-xl">
    <div
      className={`w-10 h-10 rounded-full flex justify-center items-center ${color} about`}
    >
      {image ? (
        <img
          src={image}
          alt={`Image of ${title}`}
          className="w-10 h-10 about-img rounded-full"
        />
      ) : null}
    </div>
    <div className="ml-5 flex flex-col flex-1">
      <h3 className="mt-2 text-white text-lg">{title}</h3>
      <p className="mt-2 text-white text-sm md:w-9/12">
        {subtitle1}
        <p>{subtitle2}</p>
      </p>
    </div>
  </div>
);

const AboutCard = () => (
  <div className="flex w-full justify-center items-center gradient-bg-aboutcard">
    <div className="flex mf:flex-row flex-col items-center justify-between md:p-20 py-12 px-4">
      <div className="flex-1 flex flex-col justify-start items-start">
        <h1 className="text-white text-3xl sm:text-5xl py-2 text-gradient ">
          Name of the College, Student and Project.
        </h1>
        <p className="text-left my-2 text-white font-light md:w-9/12 w-11/12 text-base">
```

```
          The purpose of this crypto exchange project is to enable the exchange
          of Ethereum, offering users a platform to transfer their digital
          assets securely and conveniently.
        </p>
      </div>
      <div className="flex-1 flex flex-col justify-start items-center">
        <About
          color=""
          title="SAKET COLLEGE OF ARTS, SCIENCE & COMMERCE"
          image={saket}
          subtitle=""
        />
        <About
          color=""
          title="TERUKULA SAI"
          image={sai}
          subtitle1="PNR No : 2021016400730531"
          subtitle2="Roll No : 233026 "
        />
        <About
          color=""
          title="Decentralized Transaction Web3 Application"
          image={ether}
          subtitle=""
        />
      </div>
    </div>
  </div>
);
export default AboutCard;
```

2. EthereumChart.jsx

```
import React, { useEffect, useRef, useState } from "react";
import Chart from "chart.js/auto";
import axios from "axios";

const EthereumChart = () => {
  const chartRef = useRef(null);
  const [priceData, setPriceData] = useState([]);
  const [dataToShow, setDataToShow] = useState(7);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get(
          `https://api.coingecko.com/api/v3/coins/ethereum/market_chart?vs_currency=usd&days=$
{dataToShow}&interval=daily`
        );
        const data = response.data.prices;
        setPriceData(data);
      } catch (error) {
        console.error("Error fetching Ethereum price data:", error);
      }
    };

    fetchData();
  }, [dataToShow]);

  useEffect(() => {
    if (priceData.length === 0 || !chartRef.current) {
      return;
```

```
}

const timestamps = priceData.map((entry) =>
  new Date(entry[0]).toLocaleDateString()
);
const prices = priceData.map((entry) => entry[1]);
const ctx = chartRef.current.getContext("2d");

new Chart(ctx, {
  type: "line",
  data: {
    labels: timestamps,
    datasets: [
      {
        label: "Ethereum Price (USD)",
        data: prices,
        borderColor: "rgba(203, 155, 81, 1)", // Adjust color for a 3D-like effect
        borderWidth: 1.5,
        fill: true,
        backgroundColor: "rgba(203, 155, 81, 0.1)", // Adjust color for a 3D-like effect
        pointRadius: 3,
        pointBackgroundColor: "rgba(203, 155, 81, 1)", // Adjust color for a 3D-like effect
        pointBorderWidth: 2,
      },
    ],
  },
  options: {
    responsive: true,
    maintainAspectRatio: true,
    scales: {
      x: {
```

```
    display: true,
    title: {
      display: true,
      text: "Date",
      color: "white",
      font: {
        size: 18,
      },
    },
    ticks: {
      color: "white",
      font: {
        size: 14,
      },
      maxRotation: 0,
      maxTicksLimit: 10, // Number of visible ticks
    },
  },
  y: {
    display: true,
    title: {
      display: true,
      text: "Price (USD)",
      color: "white",
      font: {
        size: 18,
      },
    },
    ticks: {
      color: "white",
      font: {
            size: 14,
          },
        },
      },
    },
    plugins: {
      legend: {
        display: true,
        labels: {
          color: "rgba(203, 155, 81, 1)", // Adjust color for a 3D-like effect
          font: {
            size: 18,
          },
        },
      },
      tooltip: {
        enabled: true,
      },
    },
  },
 });
}, [priceData]);

return (
  <div>
    <canvas ref={chartRef} className="ethereum-chart" />
  </div>
);
};
export default EthereumChart;
```

3. Loader.jsx

```
const Loader = () => (
  <div className="flex justify-center items-center py-3">
    <div className="animate-spin rounded-full h-32 w-32 border-b-4 border-clr" />
  </div>
);

export default Loader;
```

4. Transaction.jsx

```
import React, { useContext } from "react";

import { TransactionContext } from "../context/TransactionContext";

import useFetch from "../hooks/useFetch";
import dummyData from "../utils/dummyData";
import { shortenAddress } from "../utils/shortenAddress";

const TransactionsCard = ({
  addressTo,
  addressFrom,
  timestamp,
  message,
  keyword,
  amount,
  url,
}) => {
  const gifUrl = useFetch({ keyword });

  return (
<div
 className="m-4 flex flex-1
 2xl:min-w-[450px]
 2xl:max-w-[500px]
 sm:min-w-[270px]
 sm:max-w-[300px]
 min-w-full
 flex-col p-3 hover:shadow-2xl transaction"
>
 <div className="flex flex-col items-center w-full mt-3">
  <div className="display-flex justify-start w-full mb-6 p-2">
   <a
    href={`https://sepolia.etherscan.io/address/${addressFrom}`}
    target="_blank"
    rel="noreferrer"
   >
    <p className="text-white text-base">
     From: {shortenAddress(addressFrom)}
    </p>
   </a>
   <a
    href={`https://sepolia.etherscan.io/address/${addressTo}`}
    target="_blank"
    rel="noreferrer"
   >
    <p className="text-white text-base">
     To: {shortenAddress(addressTo)}
    </p>
   </a>
   <p className="text-white text-base">Amount: {amount} ETH</p>
   {message && (
          <>
            <br />
            <p className="text-white text-base">Message: {message}</p>
          </>
        )}
      </div>
      <img
        src={gifUrl || url}
        alt="nature"
        className="w-full h-64 2xl:h-96 rounded-md shadow-lg object-cover"
      />
      <div className="bg-black p-3 px-5 w-max rounded-3xl -mt-5 shadow-2xl">
        <p className="text-[#c8a849] font-bold">{timestamp}</p>
      </div>
    </div>
  </div>
 );
};

const Transactions = () => {
  const { transactions, currentAccount } = useContext(TransactionContext);

  return (
    <div className="flex w-full justify-center items-center 2xl:px-20 gradient-bg-transactions">
      <div className="flex flex-col md:p-12 py-12 px-4">
        {currentAccount ? (
          <h3 className="text-white text-3xl text-center my-2">
            Latest Transactions
          </h3>
        ) : (
          <h3 className="text-white text-3xl text-center my-2">
          Connect your account to see the latest transactions
        </h3>
      )}

      <div className="flex flex-wrap justify-center items-center mt-10">
        {[...dummyData, ...transactions].reverse().map((transaction, i) => (
          <TransactionsCard key={i} {...transaction} />
        ))}
      </div>
    </div>
  </div>
 );
};

export default Transactions;
```

5. Welcome.jsx

```
import React, { useContext } from "react";
import {
  AiFillAccountBook,
  AiFillPlayCircle,
  AiOutlineApi,
} from "react-icons/ai";
import { SiEthereum } from "react-icons/si";
import { BsInfoCircle } from "react-icons/bs";

import { TransactionContext } from "../context/TransactionContext";
import { shortenAddress } from "../utils/shortenAddress";
import { Loader } from ".";

const companyCommonStyles =
  "min-h-[70px] sm:px-0 px-2 sm:min-w-[120px] flex justify-center items-center border-[0.2px] border-company text-sm font-light text-white";


const Input = ({ placeholder, name, type, value, handleChange }) => (
  <input
    placeholder={placeholder}
    type={type}
    step="0.000001"
    value={value}
    onChange={(e) => handleChange(e, name)}
    className="my-2 w-full rounded-sm p-2 outline-none bg-transparent text-white border-[1px] border-[#c8a849] text-sm white-glassmorphism"
  />
);


const Welcome = () => {
  const {
    currentAccount,
    connectWallet,
    handleChange,
    sendTransaction,
    formData,
    isLoading,
  } = useContext(TransactionContext);


  const handleSubmit = (e) => {
    const { addressTo, amount, keyword, message } = formData;


    e.preventDefault();


    if (!addressTo || !amount || !keyword || !message) return;
  sendTransaction();
};

return (
  <div className="flex w-full justify-center items-center">
    <div className="flex mf:flex-row flex-col items-start justify-between md:p-20 py-12 px-4">
      <div className="flex flex-1 justify-start items-start flex-col mf:mr-10">
        <h1 className="text-3xl sm:text-5xl text-white text-gradient py-1 dec-ani">
          Connecting you to the <span>Decentralize</span> future.
        </h1>
        <p className="text-left mt-5 text-white font-light md:w-9/12 w-11/12 text-base tag-ani">
          Experience a secure, user-friendly platform for seamless
          cryptocurrency exchange. Unlock the potential of digital assets and
          shape your financial future with confidence.
        </p>
        {!currentAccount && (
          <button
            type="button"
            onClick={connectWallet}
             className="flex flex-row justify-center items-center my-5 bg-[#c0c0c06c] p-3 rounded-
full cursor-pointer hover:bg-[#daa455e8] btn-box"
          >
            <AiOutlineApi className="text-white mr-2" />
            <p className="text-white text-base font-semibold">
              Connect Wallet
            </p>
          </button>
        )}

        <div className="grid sm:grid-cols-3 grid-cols-2 w-full mt-10 features">
        <div className={`rounded-tl-2xl ${companyCommonStyles}`}>
          Web 3.0
        </div>
        <div className={companyCommonStyles}>Security</div>
        <div className={`sm:rounded-tr-2xl ${companyCommonStyles}`}>
          Ethereum
        </div>
        <div className={`sm:rounded-bl-2xl ${companyCommonStyles}`}>
          Reliability
        </div>
        <div className={companyCommonStyles}>Low Fees</div>
        <div className={`rounded-br-2xl ${companyCommonStyles}`}>
          Blockchain
        </div>
      </div>
    </div>


    <div className="flex flex-col flex-1 items-center justify-start w-full mf:mt-0 mt-10">
        <div className="p-3 flex justify-end items-start flex-col rounded-xl h-40 sm:w-72 w-full
my-5 eth-card .white-glassmorphism ">
        <div className="flex justify-between flex-col w-full h-full">
          <div className="flex justify-between items-start">
                <div className="w-10 h-10 rounded-full border-2 border-white flex justify-center
items-center">
              <SiEthereum fontSize={21} color="#fff" />
            </div>
            <BsInfoCircle fontSize={17} color="#fff" />
          </div>
          <div>
            <p className="text-white font-light text-sm">
              {shortenAddress(currentAccount)}
            </p>
            <p className="text-white font-semibold text-lg mt-1 Ether-back">
              Ethereum
            </p>
          </div>
        </div>
      </div>

        <div className="p-5 sm:w-96 w-full flex flex-col justify-start items-center blue-
glassmorphism">
        <Input
          placeholder="Address to Transfer"
          name="addressTo"
          type="text"
          handleChange={handleChange}
        />
        <Input
          placeholder="Amount (ETH)"
          name="amount"
          type="number"
          handleChange={handleChange}
        />
        <Input
          placeholder="Keyword (Gif)"
          name="keyword"
          type="text"
          handleChange={handleChange}
        />
        <Input
          placeholder="Enter Message"
          name="message"
          type="text"
          handleChange={handleChange}
        />

        <div className="h-[1px] w-full bg-gray-400 my-2" />

        {isLoading ? (
          <Loader />
        ) : (
          <button
            type="button"
            onClick={handleSubmit}
            className="text-white w-full mt-2 border-[1px] p-2 border-[#c8a849] hover:bg-[#c8a849] hover:text-black rounded-full cursor-pointer"
          >
            Transfer
          </button>
        )}
      </div>
    </div>
  </div>
</div>
);
};

export default Welcome;
```

4.2.2 Smart Contract Folder

1. Transactions.sol

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.0;

import "hardhat/console.sol";

contract Transactions {
   uint256 transactionCount;

   event Transfer(address from, address receiver, uint amount, string message, uint256 timestamp, string keyword);

   struct TransferStruct {
      address sender;
      address receiver;
      uint amount;
      string message;
      uint256 timestamp;
      string keyword;
   }

   TransferStruct[] transactions;

     function addToBlockchain(address payable receiver, uint amount, string memory message, string memory keyword) public {
      transactionCount += 1;
         transactions.push(TransferStruct(msg.sender, receiver, amount, message, block.timestamp, keyword));
```

```
        emit Transfer(msg.sender, receiver, amount, message, block.timestamp, keyword);
    }

    function getAllTransactions() public view returns (TransferStruct[] memory) {
        return transactions;
    }

    function getTransactionCount() public view returns (uint256) {
        return transactionCount;
    }
}
```

2. hardhat.config.js

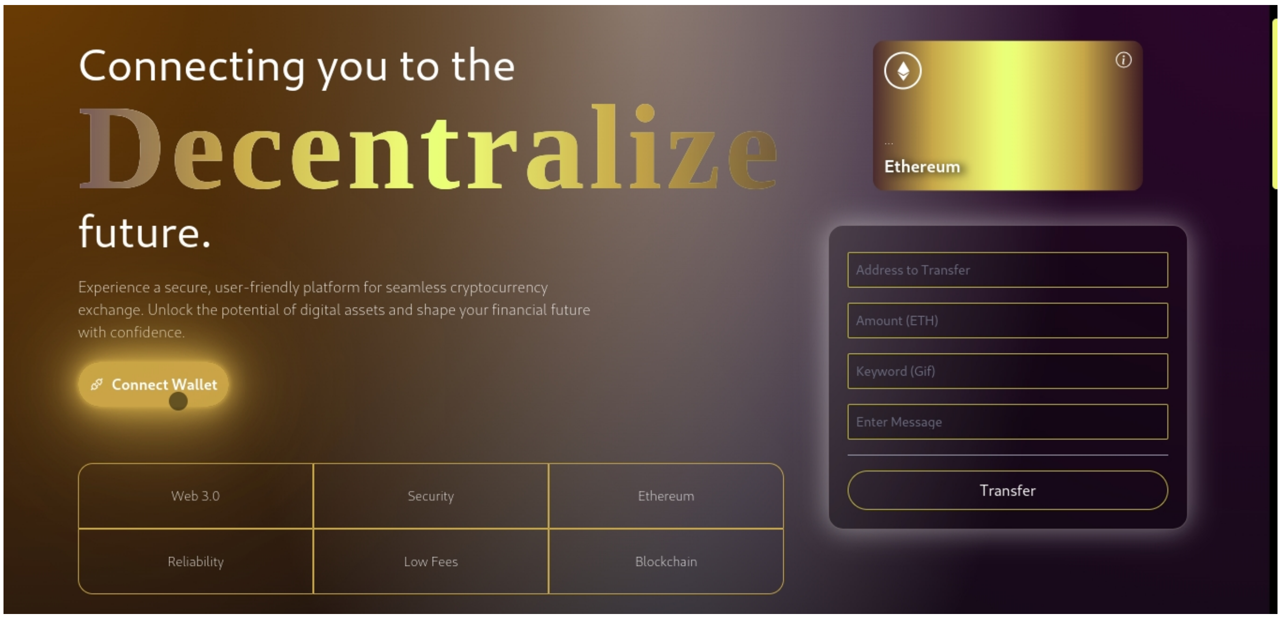
```
require('@nomiclabs/hardhat-waffle');

module.exports = {
  solidity: '0.8.0',
  networks: {
    sepolia: {
      url: 'https://eth-sepolia.g.alchemy.com/v2/mRTQxofIkLI_GF7HZZoqN1fWO6TLS7Ky',
      accounts: ['1a5c0009993bd25d28988d107c71b2ad6a5526db8e856sea9i25393cd9409c62'],
    },
  },
};
```

Chapter 5

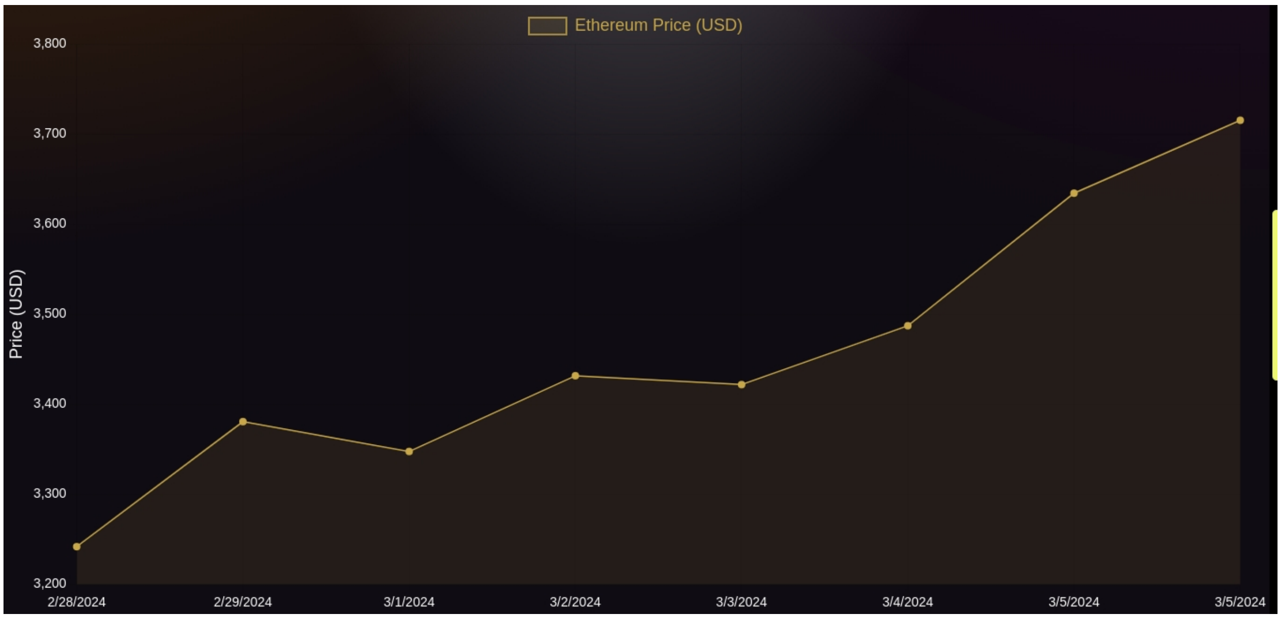
Results & Discussions

5.1 Interface before the connection of wallet

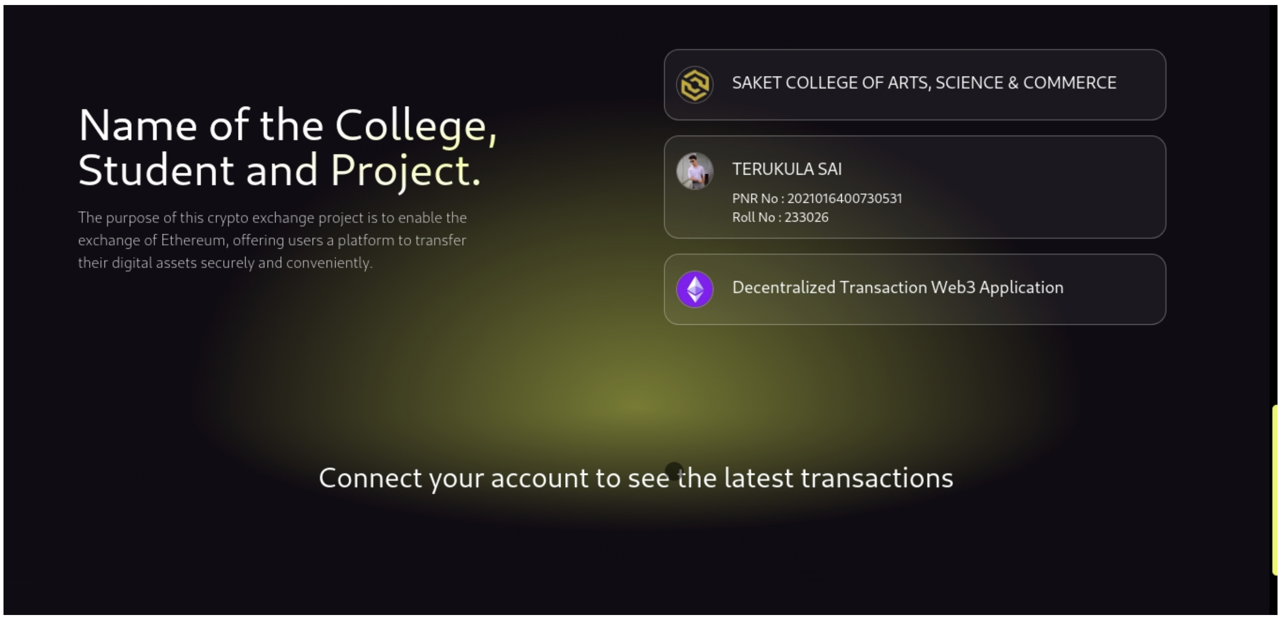
1. Home/Welcome Section



2. Ethereum Chart Section



3. About Section



Name of the College, Student and Project.

The purpose of this crypto exchange project is to enable the exchange of Ethereum, offering users a platform to transfer their digital assets securely and conveniently.

SAKET COLLEGE OF ARTS, SCIENCE & COMMERCE

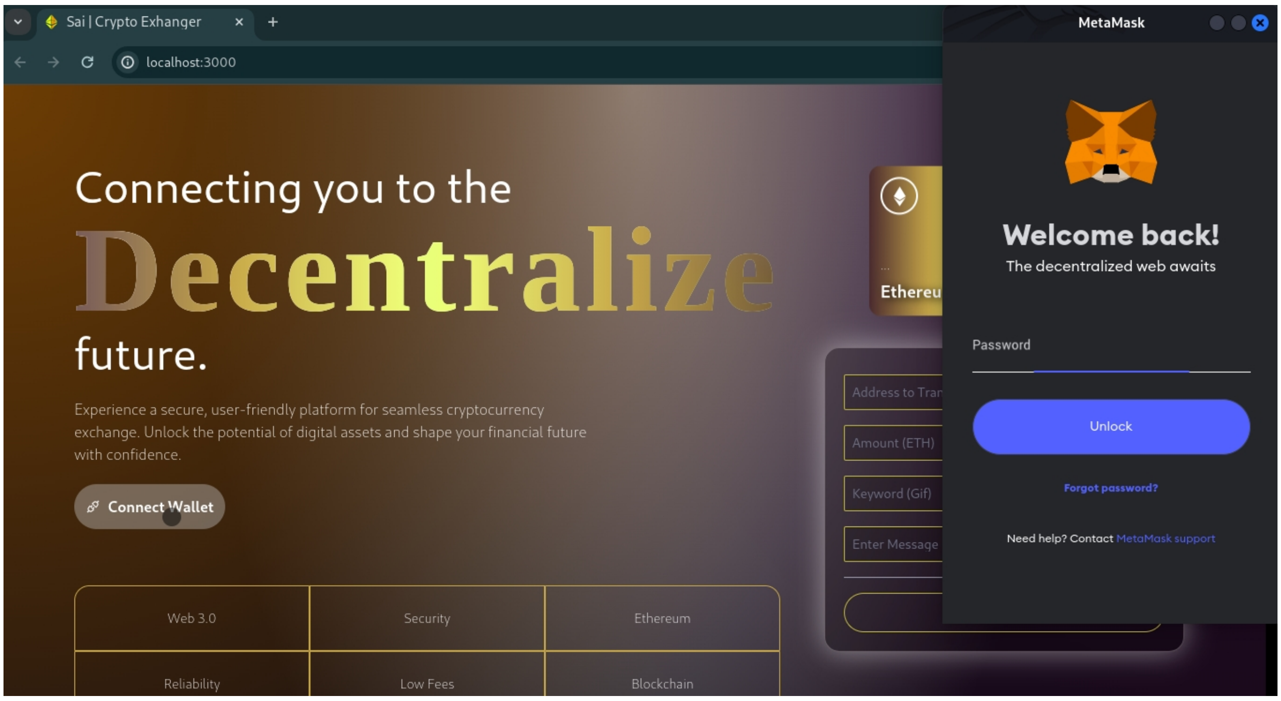
TERUKULA SAI
PNR No : 2021016400730531
Roll No : 233026

Decentralized Transaction Web3 Application

Connect your account to see the latest transactions

5.2 Interface after the connection of wallet

1. Account Login (Metamask Wallet)



Sai | Crypto Exhanger
localhost:3000

Connecting you to the Decentralize future.

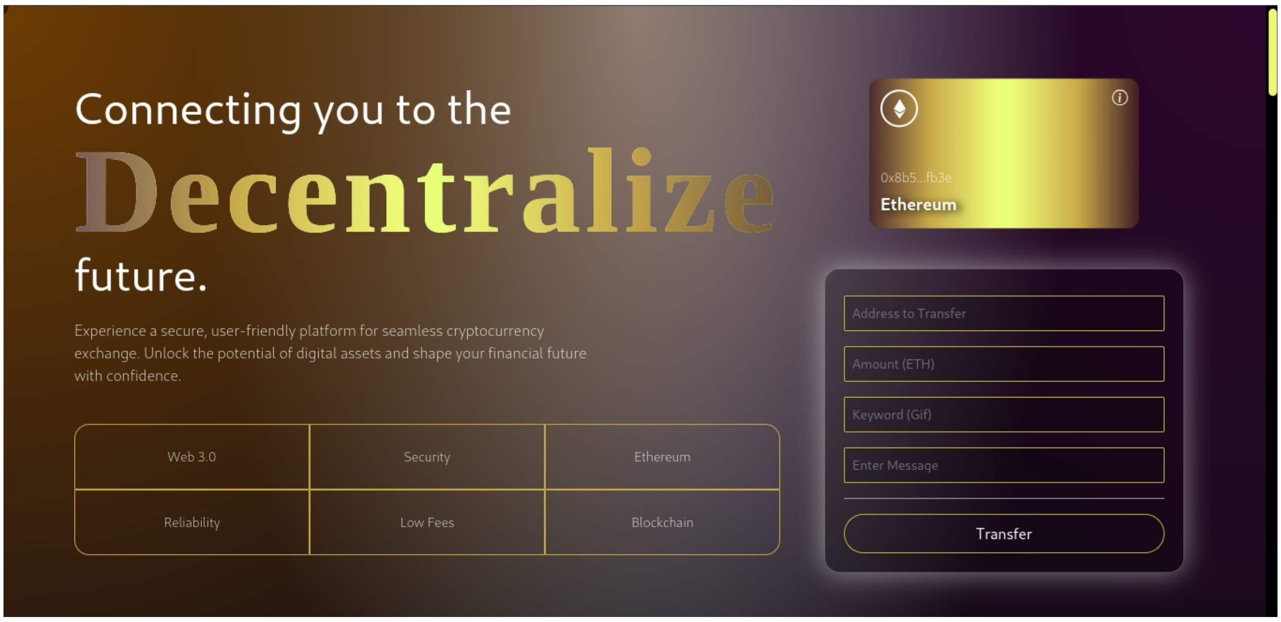
Experience a secure, user-friendly platform for seamless cryptocurrency exchange. Unlock the potential of digital assets and shape your financial future with confidence.

Welcome back!
The decentralized web awaits

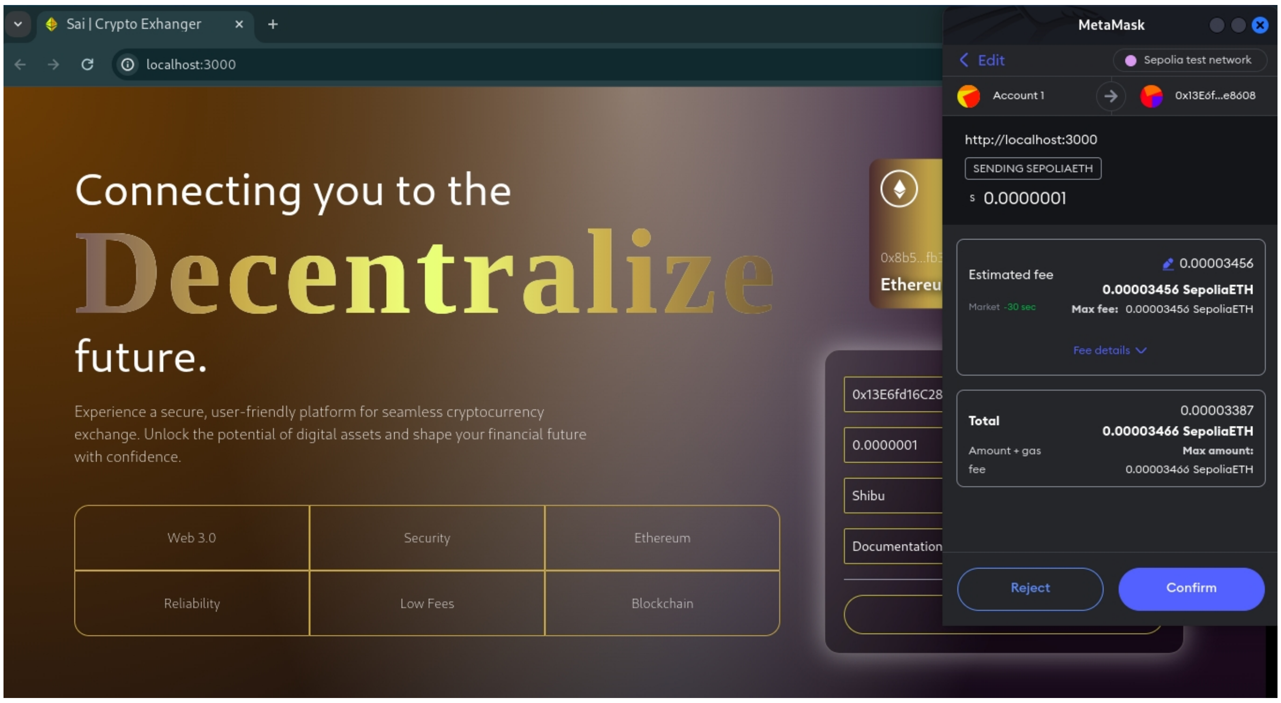
Password

Need help? Contact MetaMask support

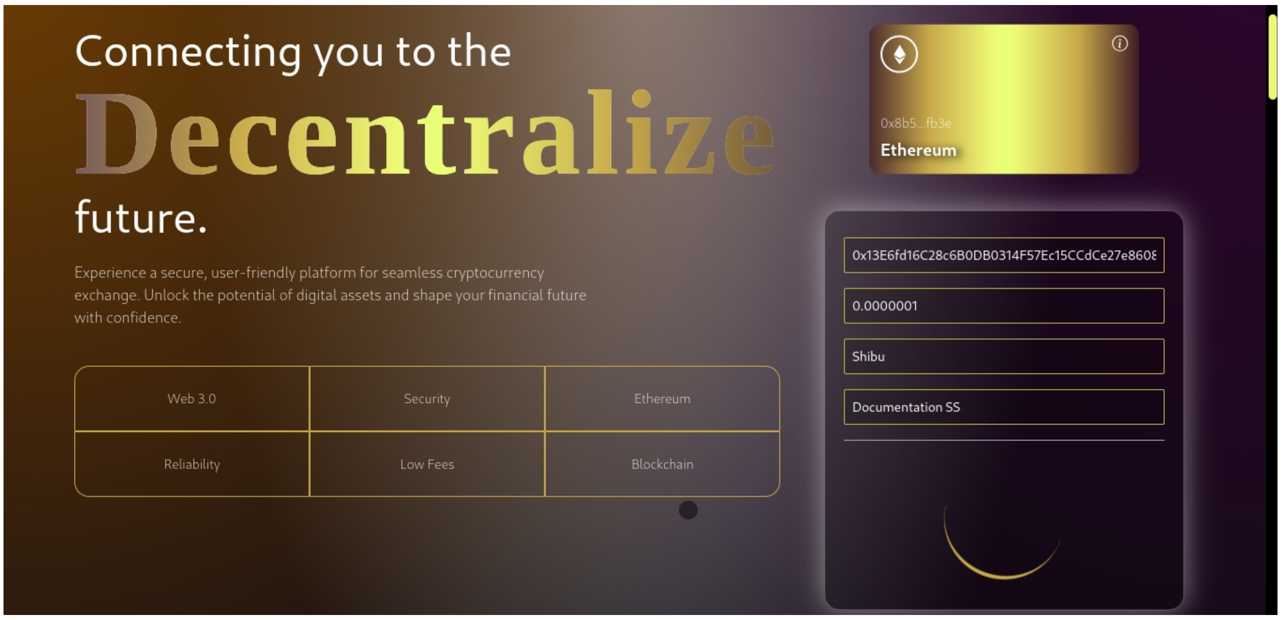
2. Home/Welcome Section



3. Transaction Confirmation



4. Transaction Loader



Connecting you to the

Decentralize

future.

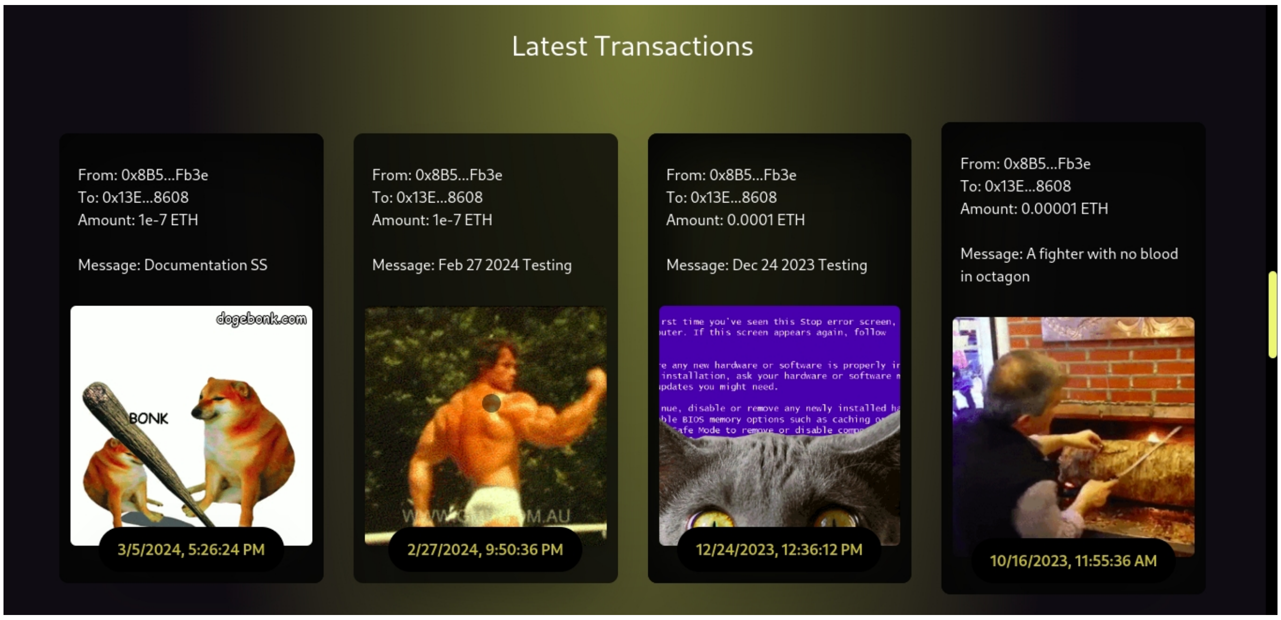
Experience a secure, user-friendly platform for seamless cryptocurrency exchange. Unlock the potential of digital assets and shape your financial future with confidence.

Web 3.0	Security	Ethereum
Reliability	Low Fees	Blockchain

0x8b5...fb3e
Ethereum

0x13E6fd16C28c6B0DB0314F57Ec15CCdCe27e8608
0.0000001
Shibu
Documentation SS

5. Transaction History



Latest Transactions

From: 0x8B5...Fb3e
To: 0x13E...8608
Amount: 1e-7 ETH

Message: Documentation SS

3/5/2024, 5:26:24 PM

From: 0x8B5...Fb3e
To: 0x13E...8608
Amount: 1e-7 ETH

Message: Feb 27 2024 Testing

2/27/2024, 9:50:36 PM

From: 0x8B5...Fb3e
To: 0x13E...8608
Amount: 0.0001 ETH

Message: Dec 24 2023 Testing

12/24/2023, 12:36:12 PM

From: 0x8B5...Fb3e
To: 0x13E...8608
Amount: 0.00001 ETH

Message: A fighter with no blood in octagon

10/16/2023, 11:55:36 AM

Chapter 6

Conclusion

The "Decentralized Transaction Application" project represents a significant leap forward in the world of blockchain technology and decentralized finance. With a clear focus on empowering individuals, enhancing security, promoting financial inclusion, and exploring emerging technologies, this project aims to reshape traditional financial systems and set an example for industry transformation. By eliminating traditional intermediaries, implementing robust security measures, and providing a user-friendly platform for secure transactions, it addresses the core challenges and limitations associated with centralized financial systems.

The project's versatile applicability extends to a broad audience, from individuals seeking secure and efficient financial transactions to underserved communities in need of accessible financial services. Moreover, it serves as an educational resource for blockchain enthusiasts and developers, contributing to the advancement of blockchain knowledge and capabilities.

By pioneering the adoption of decentralized transactions and applications, the "Decentralized Transaction Application" project serves as a catalyst for change within the financial sector. It leads the way in making financial systems more transparent, efficient, and user-centric, inspiring further innovation and adoption in the blockchain and decentralized finance space. This project represents a promising step towards a more equitable and inclusive financial landscape and has the potential to significantly impact the way financial transactions are conducted in the future.

Chapter 7

Reference

- https://research.csiro.au/blockchainpatterns/general-patterns/deployment-patterns/dapp/
- https://www.alchemy.com/create-web3-dapp
- https://hardhat.org/docs
- https://ethereum.org/pdfs/EthereumWhitePaper.pdf
- https://ethereum.org/en/developers/docs/
- https://web3js.readthedocs.io/en/v1.10.0/
- https://etherscan.io/
- https://consensys.github.io/smart-contract-best-practices/