



SOFTWARE DESIGN SPECIFICATION

Blockchain Live Project  
  
Blockchain Based Authentication

|  |  |  |  |
| --- | --- | --- | --- |
| **Created By:** | Sheefa Jalali | **Approved By:** | Harshad Topale |
| **Created On:** | 16-09-2023 | **Approved On:** | 25-09-2023 |

**INDEX**

[**1** **PURPOSE** 2](#_Toc142418236)

[**2** **PROJECT SCOPE** 2](#_Toc142418237)

[**3** **SYSTEM OVERVIEW** 2](#_Toc142418238)

[**4** **DESIGN CONSIDERATIONS** 2](#_Toc142418239)

[4.1 Requirements 3](#_Toc142418240)

[4.2 Assumptions 3](#_Toc142418241)

[4.3 Dependencies 3](#_Toc142418242)

[**5** **SYSTEM ARCHITECTURE** 3](#_Toc142418243)

[5.1 Architectural Strategies 4](#_Toc142418244)

[5.2 Structure & Relationships 4](#_Toc142418245)

[**6** **DETAILED DESCRIPTION OF COMPONENTS** 4](#_Toc142418246)

[**7** **INTEGRATION** 5](#_Toc142418247)

[**8** **APPENDICES** 1](#_Toc142418248)

[8.1 Appendix A – Detailed Description of Components 1](#_Toc142418249)

**General Instructions for using the Live Project POC Document**

* This template and the subsequent document created using this template is a confidential document and is the intellectual property of Cloud Counselage Pvt. Ltd. Circulating it outside of the organisation without the consent of Cloud Counselage Pvt. Ltd. is the breach of company policies and will lead to legal actions
* The Design Specification of a software forms the basis of development of software
* The **text between inequality (< >) is to be replaced** by relevant text
* Please **remove the yellow highlight on the Text** between the inequality (< >). This is done to help you notice the text to be changed/replaced
* The text in *italics* highlighted in grey is just for reference and should be removed after adding the relevant text

# **PURPOSE**

This document is created based on the requirement specification document. The purpose of this Software Design Specification (SDS) Document is to break down the project into components to describe in detail what the purpose of each component is and how it will be implemented. The SDS will also serve as a tool for verification and validation of the final product.

# **PROJECT SCOPE**

The scope of the Blockchain Based Authentication includes its distinct features, its benefits, and its limitations. The system's distinct features allow it to securely storing encrypted login credentials on a decentralized, immutable ledger, reducing malicious access risks, ensuring transparent and auditable transactions, eliminating a single point of failure, and aligning with decentralization principles. The aim was to deliver a robust, tamper-resistant authentication system that engenders user confidence in modern digital environments by using Solidity, Remix-IDE, Ganache, Truffle and Github. The system enables the user to securely authenticate their identity and access services with confidence. By leveraging blockchain, it safeguards sensitive data, mitigates unauthorized access risks, ensures data integrity, and upholds a decentralized approach, empowering users with a heightened level of security and trust during authentication processes.

# **SYSTEM OVERVIEW**

This section will provide an outline of the various components and subsystems of Blockchain Based Authentication.

**Users**: The individuals utilizing the system, possessing necessary credentials for authentication - Ethereum address, username, password, and a four-digit code.

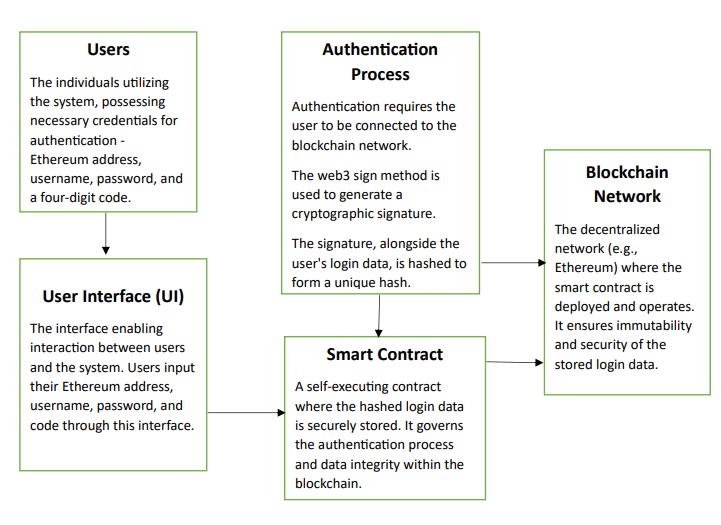
**User Interface (UI):** The interface enabling interaction between users and the system. Users input their Ethereum address, username, password, and code through this interface.

**Authentication Process:**

* Authentication requires the user to be connected to the blockchain network.
* The web3 sign method is used to generate a cryptographic signature.
* The signature, alongside the user's login data, is hashed to form a unique hash.
* This hash is stored within a smart contract on the blockchain.

**Smart Contract:** A self-executing contract where the hashed login data is securely stored. It governs the authentication process and data integrity within the blockchain.

**Blockchain Network:** The decentralized network (e.g., Ethereum) where the smart contract is deployed and operates. It ensures immutability and security of the stored login data.



# **DESIGN CONSIDERATIONS**

This section describes requirements, assumptions and dependencies to be addressed to devise a complete design solution.

## Requirements

**User Authentication Requirements:**

The system must authenticate users based on provided Ethereum address, username, password, and a four-digit code.

**Security Requirements:**

User login data must be securely hashed and stored within the blockchain using cryptographic methods.

**Reliability Requirements:**

The system should reliably authenticate users when connected to the blockchain and utilizing the appropriate cryptographic signatures.

These requirements align with the identified components, emphasizing the need for secure, reliable, and effective user authentication within the blockchain-based system.

## Assumptions

**Blockchain Connectivity Assumptions:**

Users are assumed to have a reliable internet connection to connect to the blockchain network for authentication.

**User Input Assumptions:**

It is assumed that users will provide accurate and valid Ethereum addresses, usernames, passwords, and four-digit codes for authentication.

**Security Assumptions:**

The assumption is made that cryptographic methods utilized, including hashing and digital signatures, are secure and not compromised.

These assumptions align with the identified components, recognizing critical aspects such as user input accuracy, security measures, and blockchain connectivity.

## Dependencies

**Assumption 1 - Blockchain Connectivity:**

Users are assumed to have a reliable and consistent internet connection for interacting with the blockchain network and completing the authentication process.

**Assumption 2 - User Input Reliability:**

Users are assumed to provide accurate and valid Ethereum addresses, usernames, passwords, and four-digit codes during the authentication process.

**Assumption 3 - Secure Cryptographic Operations:**

It is assumed that the cryptographic operations, including hashing and digital signatures, are secure and implemented without vulnerabilities.

These assumptions are aligned with the identified components and highlight critical considerations related to blockchain connectivity, user input reliability, and cryptographic security within the system.

# **SYSTEM ARCHITECTURE**

The software system architecture refers to the logical organization of a distributed system into software components. It defines how components of a software system are assembled, their relationship and communication between them. It serves as a blueprint for software application and development basis for developer team. An effective architecture serves as the conceptual glue that holds every phase of the project together for all of its stakeholders, enabling agility, time and cost savings, and early identification of design risks.

The Software architecture:

* Defines structure of a system
* Defines behaviour of a system
* Defines component relationship.
* Defines communication structure.
* Balances stakeholder’s needs
* Influences team structure
* Focuses on significant elements.
* Captures early design decisions.

Below some important characteristics which are commonly considered are explained.

**Operational Architecture Characteristics:**

* Availability
* Performance
* Reliability
* Low fault tolerance
* Scalability

**Structural Architecture Characteristics:**

* Configurability
* Extensibility
* Supportability
* Portability
* Maintainability

**Cross-Cutting Architecture Characteristics:**

* Accessibility
* Security
* Usability
* Privacy
* Feasibility

## Architectural Strategies

**User Interface (UI):**

This component is the interface through which users interact with the system, providing necessary login information such as Ethereum address, username, password, and a four-digit code.

**Authentication Logic:**

The authentication logic is responsible for processing the user's login data and initiating the authentication process by generating a cryptographic signature using the web3 sign method.

**Blockchain Network:**

The blockchain network, such as Ethereum, is a decentralized and distributed ledger where the authentication logic interacts with a smart contract to securely store hashed login data and perform necessary operations.

**Smart Contract:**

A smart contract is a self-executing contract with the terms of the agreement directly written into code. In this project, the smart contract securely stores hashed login data and manages authentication processes.

**Cryptographic Operations Module:**

This component handles cryptographic operations like hashing and generating digital signatures, essential for securing user login data and the authentication process.

**User Data Storage:**

This component represents the storage mechanism, within the smart contract on the blockchain, where the hashed login data of users is securely stored.

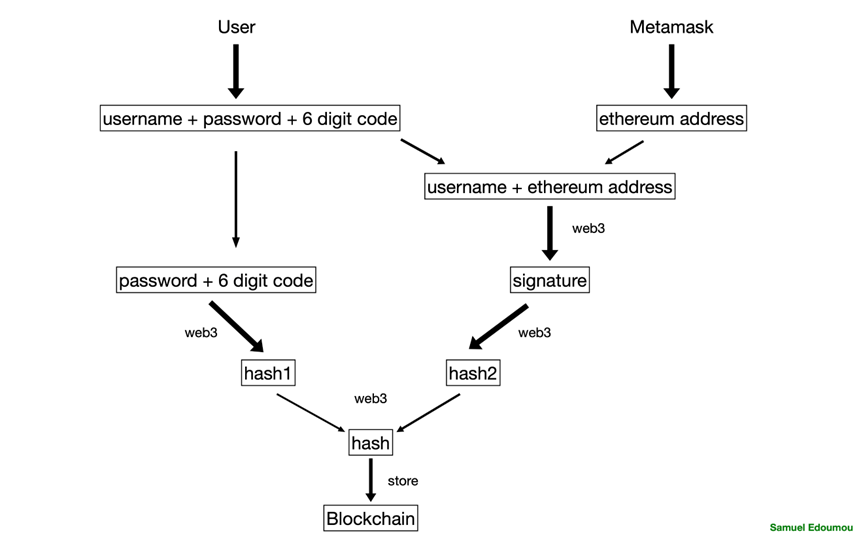
**Blockchain Interaction Layer:**

The interaction layer facilitates communication between the authentication logic and the blockchain network, ensuring seamless integration and data transfer.

**Decentralized Identity Layer:**

This layer ensures that each user is associated with a unique and immutable identity on the blockchain, enhancing security and ownership proof during authentication.

## Structure & Relationships



# **DETAILED DESCRIPTION OF COMPONENTS**

For detailed description of the components, please refer **Appendix A – Detailed Description of Components**

The below template will be used to specify the details of all the components.

**Table 1: Detailed Design Specification Template**

|  |  |
| --- | --- |
| **Identification** | User Interface (UI)  Identification: UIComponent  Location: Front-end layer, directly accessible by users.  Authentication Logic  Identification: AuthLogicComponent  Location: Back-end layer, responsible for processing and authenticating user data.  Blockchain Network  Identification: BlockchainNetworkComponent  Location: External to the system, encompassing the decentralized Ethereum network.  Smart Contract  Identification: SmartContractComponent  Location: Within the BlockchainNetworkComponent, as a self-executing contract.  Cryptographic Operations Module  Identification: BlockchainInteractionLayer  Location: Part of the Authentication Logic, facilitating communication with the blockchain.  Decentralized Identity Layer  Identification: DecentralizedIdentityLayer  Location: Integrated within the Authentication Logic, ensuring decentralized and immutable identity. |
| **Type** | User Interface (UI)  Type: Form  Authentication Logic  Type: Subprogram  Blockchain Network  Type: External System  Smart Contrac  Type: Smart Contract  Cryptographic Operations Module  Type: Module  User Data Storage  Type: Data File  Blockchain Interaction Layer  Type: Module  Decentralized Identity Layer  Type: Module |
| **Purpose** | User Interface (UI)  Purpose: Enable users to input login data and interact with the system for authentication.  Function: Capture Ethereum address, username, password, and four-digit code from users.  Performance Requirements: Responsive design, usability, and compatibility across devices.  Derived Requirement: GUI should guide users to provide accurate login information.  Authentication Logic  Purpose: Process user input, generate cryptographic signatures, and initiate authentication.  Function: Utilize web3.js to generate a cryptographic signature and initiate the authentication process.  Performance Requirements: Efficient cryptographic operations, minimal processing time for authentication.  Derived Requirement: Verify the integrity of the cryptographic signature before proceeding with authentication.  Blockchain Network  Purpose: Serve as a decentralized, immutable ledger for storing hashed login data.  Function: Enable storage and retrieval of hashed login data via smart contracts on the blockchain.  Performance Requirements: High availability, data immutability, low latency in data retrieval.  Derived Requirement: Ensure the blockchain network can handle a large number of authentication requests securely.  Smart Contract  Purpose: Securely store hashed login data and govern the authentication process.  Function: Store hashed login data and manage authentication based on user requests.  Performance Requirements: High security (immutability, encryption), efficient data storage and retrieval.  Derived Requirement: Implement proper access control mechanisms within the smart contract.  Cryptographic Operations Module  Purpose: Perform necessary cryptographic operations for securing login data.  Function: Implement hashing and digital signature generation for user data.  Performance Requirements: High-speed cryptographic operations, resistance to attacks.  Derived Requirement: Ensure the cryptographic operations are compliant with industry standards.  User Data Storage  Purpose: Store hashed login data in a secure and structured manner.  Function: Store hashed login data securely within the smart contract on the blockchain.  Performance Requirements: Efficient data storage, quick data retrieval.  Derived Requirement: Implement mechanisms to detect and mitigate unauthorized data access.  Blockchain Interaction Layer  Purpose: Facilitate communication between the authentication logic and the blockchain.  Function: Handle requests and responses between the system and the blockchain network.  Performance Requirements: Low latency, high reliability in data transmission.  Derived Requirement: Implement failover mechanisms for uninterrupted communication.  Decentralized Identity Layer  Purpose: Ensure users have a unique and immutable identity on the blockchain.  Function: Assign and manage decentralized identities for users.  Performance Requirements: Fast and accurate identity assignment, resistance to identity duplication.  Derived Requirement: Implement a mechanism to recover compromised identities. |
| **Subordinates** | User Interface (UI)  Subordinates:  Input Form  Submit Button  Error Message Display  Internal Structure:  HTML structure for input fields and buttons  CSS for styling and layout  JavaScript for capturing and validating user input  Constituents and Functional Requirements:  Input Form: Captures Ethereum address, username, password, and four-digit code.  Submit Button: Initiates the authentication process.  Error Message Display: Displays errors for incorrect or missing user input.  Authentication Logic  Subordinates:  Cryptographic Signature Generator  Data Processor  Internal Structure:  Functions/methods for cryptographic signature generation  Logic for processing user input  Constituents and Functional Requirements:  Cryptographic Signature Generator: Generates a cryptographic signature using web3.js based on user input.  Data Processor: Processes the generated signature and user input to initiate the authentication process.  Blockchain Network  Subordinates:  Blockchain Node  Data Storage Contracts  Internal Structure:  Nodes distributed across a decentralized network  Smart contracts for data storage and retrieval  Constituents and Functional Requirements:  Blockchain Node: Node maintaining a copy of the blockchain, ensuring data availability.  Data Storage Contracts: Smart contracts responsible for securely storing and managing hashed login data.  Smart Contract  Subordinates:  Data Storage Module  Authentication Control Module  Internal Structure:  Functions for storing and retrieving hashed login data  Logic for authentication control  Constituents and Functional Requirements:  Data Storage Module: Stores hashed login data securely within the contract.  Authentication Control Module: Implements authentication logic based on user requests.  Cryptographic Operations Module  Subordinates:  Hashing Function  Digital Signature Generator  Internal Structure:  Algorithms for hashing and digital signature generation  Constituents and Functional Requirements:  Hashing Function: Performs cryptographic hash on user login data.  Digital Signature Generator: Generates a digital signature based on the hash.  User Data Storage  Subordinates:  Hashed Login Data  Internal Structure:  Storage mechanism for securely storing hashed login data  Constituents and Functional Requirements:  Hashed Login Data: Securely stores the hashed login data.  Blockchain Interaction Layer  Subordinates:  Data Transmission Module  Internal Structure:  Functions/methods for interacting with the blockchain network  Constituents and Functional Requirements:  Data Transmission Module: Handles requests and responses between the system and the blockchain network.  Decentralized Identity Layer  Subordinates:  Identity Assignment Module  Internal Structure:  Functions/methods for assigning decentralized identities  Constituents and Functional Requirements:  Identity Assignment Module: Assigns and manages decentralized identities for users. |
| **Dependencies** | Database Management System (DBMS):  The User Account component relies on a DBMS for storing and managing user account data.  It uses the DBMS's capabilities for efficient data retrieval, storage, and querying.  Function and Performance Relatability to Other Components:  Authentication Logic:  Functionally, the Authentication Logic may utilize user account data to authenticate users during the login process.  Performance-wise, efficient access to user account data impacts the speed and accuracy of the authentication process.  Profile Page UI:  Functionally, the Profile Page UI component fetches and displays user account information for users to view and potentially modify.  Performance-wise, a timely retrieval of accurate data influences the responsiveness and usability of the profile page.  Usage by Other Components:  Authentication Logic:  The Authentication Logic component utilizes the user account data to verify and authenticate users during login attempts.  Profile Page UI:  The Profile Page UI component fetches the user account data to display the user's information on the profile page.  Interaction Details:  Timing:  The Authentication Logic typically interacts with the User Account component during user login attempts.  The Profile Page UI interacts when a user accesses their profile information.  Interaction Conditions:  The Authentication Logic triggers an interaction during login, requiring user account data to authenticate.  The Profile Page UI triggers an interaction when a user intends to view or modify their account information.  Order of Execution:  Authentication Logic typically interacts before granting access, making it an early-stage interaction.  Profile Page UI interaction occurs upon a user's specific request to view or modify their account details.  Data Sharing:  User account data is shared securely between the User Account component and the components using it, ensuring privacy and data integrity.  Responsibility for Creation, Duplication, Use, Storage, and Elimination:  Creation and storage of user account data are responsibilities of the User Account component and the underlying database.  Duplication, use, and elimination follow defined protocols within the User Account component, ensuring secure and accurate handling of data. |
| **Interfaces** | Internal Interfaces:  Database Interface:  Description: Allows the User Account component to communicate with the database for storing and retrieving user account data.  Communication Mechanism: Blockchain Network  External Interfaces:  Profile Page UI Interface:  Description: Facilitates communication between the User Account component and the Profile Page UI for viewing and updating user account information.  Communication Mechanism: JSON or similar data exchange formats.  Communication Mechanisms:  Messages and Parameters:  When the Profile Page UI requests user account data for display or modification, it sends a message containing the user's unique identifier.  Parameters in the message include the user's unique identifier and the requested action (e.g., view account, update account).  Common Data Areas:  Shared data areas within the User Account component hold the user account information temporarily during processing and updates.  Data areas are accessed by internal processes for reading and updating user account data.  Error Messages and Error Codes:  Error Messages:  Error Message 1: "User account not found."  Error Message 2: "Invalid password. Please provide valid parameters."  Error Codes:  Error Code 1: 404 (User account not found)  Error Code 2: 400 (Bad Request)  Screen Formats and Interactive Messages:  Profile Page UI:  Screen Format: HTML/CSS layout displaying user account information in a structured format.  Interactive Messages: "Update successful", "Password changed successfully", "Error: Invalid input." |
| **Resources** | Database Management System (DBMS):  Type: Software Resource  Description: A database management system is essential for storing, managing, and retrieving user account data. The DBMS ensures secure and efficient handling of the data stored by the User Account component.  Here the blockchain is used to store data in a secure and encrepted manner.  Server:  Type: Hardware and/or Software Resource  Description: The User Account component requires server infrastructure or cloud-based services to host the application and provide the necessary computing environment for the component to function optimally. Here the Blockchain Network is being used. |
| **Processing** | **Constructor:**  Description: Initializes the contract and sets the initial number of users to 0.  Pseudocode:  Constructor:  Set nbOfUsers to 0.  **register:**  Description: Registers a user by storing their Ethereum address and a corresponding signature hash.  Pseudocode:  Function: register(\_signature)  Input: \_signature (The signature hash of the user)  Output: None  Pseudocode:  Check if the user is not already registered using their Ethereum address.  If not registered:  Set the user's signature hash to \_signature.  Set the user's Ethereum address to msg.sender.  Increment nbOfUsers by 1.  **getSignatureHash:**  Description: Retrieves the stored signature hash for the calling user.  Pseudocode:  Function: getSignatureHash()  Input: None  Output: Signature hash (string)  Pseudocode:  Check if the caller is the registered user by comparing the Ethereum address.  If caller is registered:  Return the user's signature hash.  **getUserAddress:**  Description: Retrieves the Ethereum address of the calling user.  Pseudocode:  Function: getUserAddress()  Input: None  Output: Ethereum address (address)  Pseudocode:  Return the user's Ethereum address. |
| **Data** | **nbOfUsers:**  Representation Method: Unsigned integer (uint256) representing the total number of registered users.  Initial Value: 0 (initialized in the constructor).  Use: Keeps track of the total number of registered users for the contract.  Semantics: Provides a count of the registered users.  Format: Integer value.  **User Struct:**  Representation Method: Struct with two fields - signatureHash (string) and userAddress (Ethereum address).  Initial Value: Empty struct (initialized when a user registers).  Use: Stores the signature hash and Ethereum address of each registered user.  Semantics: Provides a way to associate a user's Ethereum address with their signature hash.  Format: { "signatureHash": string, "userAddress": Ethereum address }  **mapping(address => User) private user:**  Representation Method: Mapping that associates Ethereum addresses with the corresponding User struct.  Initial Value: Empty mapping (initialized with the contract).  Use: Acts as a lookup to quickly access a user's details using their Ethereum address.  Semantics: Enables efficient retrieval of user information based on their Ethereum address.  Format: Mapping format - address => User struct.  **Signature Hash (\_signature):**  Representation Method: String representing the cryptographic signature hash generated by the user.  Initial Value: Empty string.  Use: Stores the cryptographic signature hash generated by the user during registration.  Semantics: Provides a unique identifier for the user.  Format: Hexadecimal string.  **User Address (msg.sender):**  Representation Method: Ethereum address.  Initial Value: Ethereum address of the transaction sender (msg.sender).  Use: Identifies the user making a transaction or calling a function.  Semantics: Represents the Ethereum address associated with a user.  Format: Ethereum address format. |

# **INTEGRATIONS**

**Web3.js Integration:**

Integration with Web3.js, a JavaScript library for interacting with the Ethereum blockchain, is crucial. It allows seamless communication between the user interface and the Ethereum blockchain, enabling actions like generating cryptographic signatures.

**Metamask or Wallet Integration:**

Integration with popular Ethereum wallets like MetaMask may be required to facilitate Ethereum address management and secure transactions. Users could interact with their Ethereum address through the wallet integrated into the authentication process.

**External Identity Providers:**

Integration with identity providers (e.g., Google, Facebook) for streamlined user registration and login processes can enhance user experience and broaden the user base.

**DApp Browsers and Extensions:**

Integration with decentralized application (DApp) browsers or extensions (e.g., Mist, Brave) ensures compatibility and smooth operation of the authentication system within the decentralized web ecosystem.

**API Integrations:**

Integration with external APIs may be necessary for additional functionalities such as two-factor authentication (2FA), third-party verification, or accessing specific data sources to enhance the overall security and functionality of the system.

**Database Integration (if required):**

Integration with a traditional database may be needed for storing non-sensitive data or for specific functionalities outside the blockchain scope, ensuring a comprehensive and efficient data management approach.

**Logging and Monitoring Tools:**

Integration with logging and monitoring tools (e.g., Splunk, ELK Stack) is essential for real-time tracking of system behavior, identifying potential issues, and ensuring system health and performance.

**Identity Verification Services:**

Integration with identity verification services (e.g., KYC providers) may be necessary for enhancing identity validation and authentication processes, especially in compliance-sensitive applications

# **APPENDICES**

## Appendix A – Detailed Description of Components

|  |  |
| --- | --- |
| **Identification** | **A screenshot of a computer  Description automatically generatedSign-up form** |
| **Type** | Class/Form/ |
| **Purpose** | Enable users to input their login credentials, including username, password, and four-digit code, to initiate the authentication process. |
| **Subordinates** | Username Input Field: Gathers the user's username.  Password Input Field: Allows users to input their password.  Four-Digit Code Input Field: Captures the four-digit code for authentication.  Submit Button: Initiates the authentication process.  Error Message Display: Displays errors for incorrect or missing input. |
| **Dependencies** | Authentication Logic: The sign-in form relies on the Authentication Logic to process and authenticate the user's input once the Submit Button is clicked. It sends the gathered login credentials to the Authentication Logic for further processing.  Cryptographic Operations Module: The form may utilize cryptographic operations to securely transmit the input data, depending on the implementation. If used, it depends on this module to handle the cryptographic aspects of the input data. |
| **Interfaces** | User Interface (UI): The sign-in form interfaces directly with the UI, gathering user input and presenting error messages.  Authentication Logic Interface: It interfaces with the Authentication Logic to transmit the collected user login data for authentication. |
| **Resources** | UI Resources: Styling, assets, and elements required to render the form adequately for user interaction.  Authentication Logic Resources: Functions and methods needed to process the input from the form. |
| **Processing** | Captures the user's provided Ethereum address, username, password, and four-digit code.  Validates the input data for completeness and correctness.  Sends the validated data to the Authentication Logic to initiate the authentication process. |
| **Data** | Input Data:  Ethereum Address  Username  Password  Four-Digit Code  Output Data:  Error Messages (if any)  Validated Login Credentials (sent to Authentication Logic) |

|  |  |
| --- | --- |
| **Identification** | **Home PageA screenshot of a computer  Description automatically generated** |
| **Type** | Class/Form/ |
| **Purpose** | The home page aims to illustrate and inform users about the authentication system based on blockchain technology, emphasizing the secure storage and hashing of login data within a smart contract. |
| **Subordinates** | Information Display Section: Displays the provided information about the authentication system. |
| **Dependencies** | Authentication Logic: May depend on the Authentication Logic to extract information to be displayed on the home page accurately. |
| **Interfaces** | User Interface (UI): Interacts directly with the UI to display information and illustrations to the user. |
| **Resources** | UI Resources: Styling, assets, and elements required to render the information and illustrations effectively.  Content Resources: Text and media resources presenting the information about the blockchain-based authentication system. |
| **Processing** | Retrieves the provided information about the authentication system.  Renders the content and illustrations on the home page for user visibility. |
| **Data** | Information about the authentication system is provided in the description. Rendered content and illustrations presented on the home page. |

|  |  |
| --- | --- |
| **Identification** | A screenshot of a login form  Description automatically generatedSign-in Form |
| **Type** | Class/Form |
| **Purpose** | The sign-in page allows users to input their login credentials, including Ethereum address, username, password, and a four-digit code. It initiates the authentication process by gathering essential information for secure login. |
| **Subordinates** | Ethereum Address Input Field: Captures the user's Ethereum address.  Username Input Field: Gathers the user's username.  Password Input Field: Allows users to input their password.  Four-Digit Code Input Field: Captures the four-digit code for authentication.  Submit Button: Initiates the authentication process.  Error Message Display: Displays errors for incorrect or missing input. |
| **Dependencies** | Authentication Logic: The sign-in page relies on the Authentication Logic to process and authenticate the user's input once the Submit Button is clicked. It sends the gathered login credentials to the Authentication Logic for further processing.  Cryptographic Operations Module: The page may utilize cryptographic operations to securely transmit the input data, depending on the implementation. If used, it depends on this module to handle the cryptographic aspects of the input data. |
| **Interfaces** | User Interface (UI): The sign-in page interfaces directly with the UI, gathering user input and presenting error messages.  Authentication Logic Interface: It interfaces with the Authentication Logic to transmit the collected user login data for authentication. |
| **Resources** | UI Resources: Styling, assets, and elements required to render the page adequately for user interaction.  Authentication Logic Resources: Functions and methods needed to process the input from the page. |
| **Processing** | Captures the user's provided Ethereum address, username, password, and four-digit code.  Validates the input data for completeness and correctness.  Sends the validated data to the Authentication Logic to initiate the authentication process. |
| **Data** | Input Data:  Username  Password  Four-Digit Code  Output Data:  Error Messages (if any)  Validated Login Credentials (sent to Authentication Logic) |

|  |  |
| --- | --- |
| **Identification** | A screenshot of a computer  Description automatically generatedUser Account |
| **Type** | Class |
| **Purpose** | The User Account component stores essential personal information of a user, including name, occupation, date of birth, age, city, state, and country. It serves as a centralized repository for managing and retrieving user-specific data. |
| **Subordinates** | Data Fields:  Name: Stores the user's full name.  Occupation: Stores the user's occupation or profession.  Date of Birth: Stores the user's date of birth.  Age: Automatically calculated based on the date of birth.  City: Stores the city of residence.  State: Stores the state or region of residence.  Country: Stores the country of residence. |
| **Dependencies** | Database Management System: Relies on a database management system to store and manage the user account data securely |
| **Interfaces** | User Interface (UI): Interfaces with the UI when users update or view their account information.  Database Interface: Interfaces with the database for data retrieval and storage. |
| **Resources** | Database Resources: Tables and fields required to store user account information securely.  UI Resources: Interface elements and forms to interact with users for account management. |
| **Processing** | Stores and updates user account data based on user interactions.  Automatically calculates the user's age based on the provided date of birth. |
| **Data** | Input Data:  User's Name  Occupation  Date of Birth  City  State  Country  Output Data:  Updated user account information |