Decentralizing Meetings

July 25, 2022

Wahaj Javed:

Background:

Every company has a board meeting every quarter in which they discuss the terms and conditions. We want those terms and conditions to be saved. But if we save it in a centralized way it can be tempered. So what we want is that we have to make the data centralized.

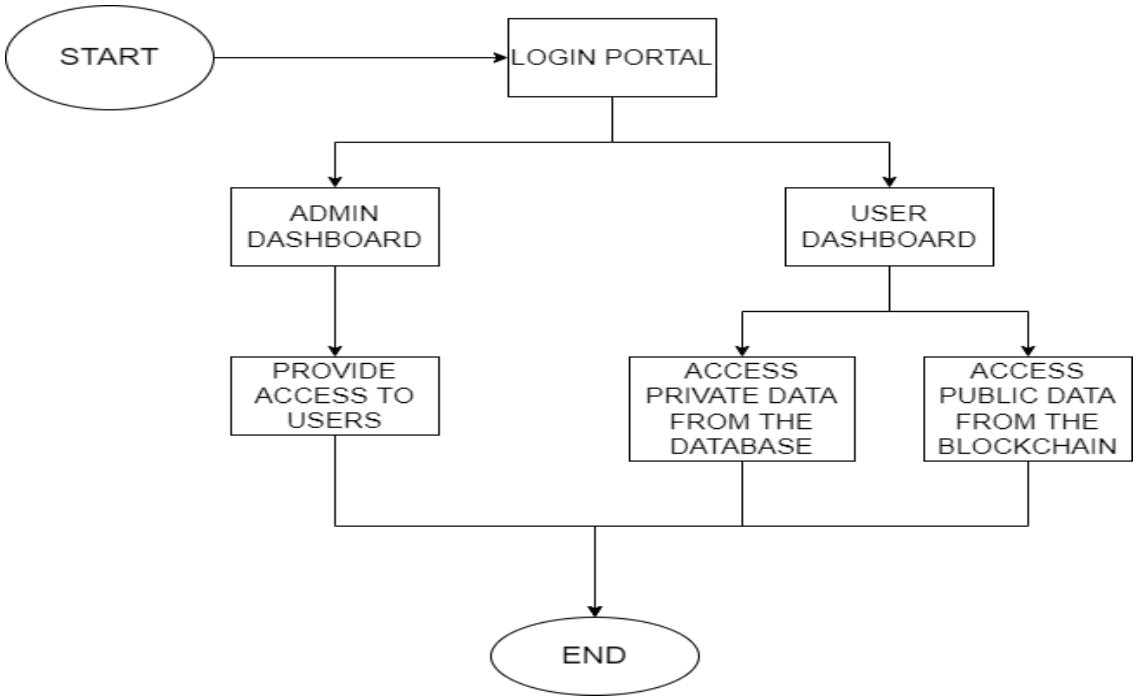
High Level Architecture:

The High Level Architecture of the blockchain comprises 2 main components, Authorization, the private blockchain, and the smart contracts.

- **Authorization:**
  - The Authorization mechanism can be designed in a typically centralized way like Email and password Login Credentials. This way, only those having the credentials can access the blockchain data and mine data blocks to it.
- **Blockchain:**
  - This chain will store the respective meeting data with timestamps so the records are easily accessible and verifiable. The data can be stored by pinning it on IPFS[defined later on] and storing the link in the metadata

in order to access it. And storing such contents in masked form so only those with private keys can access it. This way, the data is only accessible to certain individuals while the storing mechanism remains decentralized.

- **Smart Contracts**
  - The smart contract is a programmatic set of rules which is intended to automatically execute, control or document relevant events without the interference of a third-party. There is typically going to be 1 smart contract.
    - Main Contract
      - The main contract will hold the relevant rules and mechanisms to programmatically pin the PDF of the meeting terms and conditions onto IPFS and send it to the blockchain.

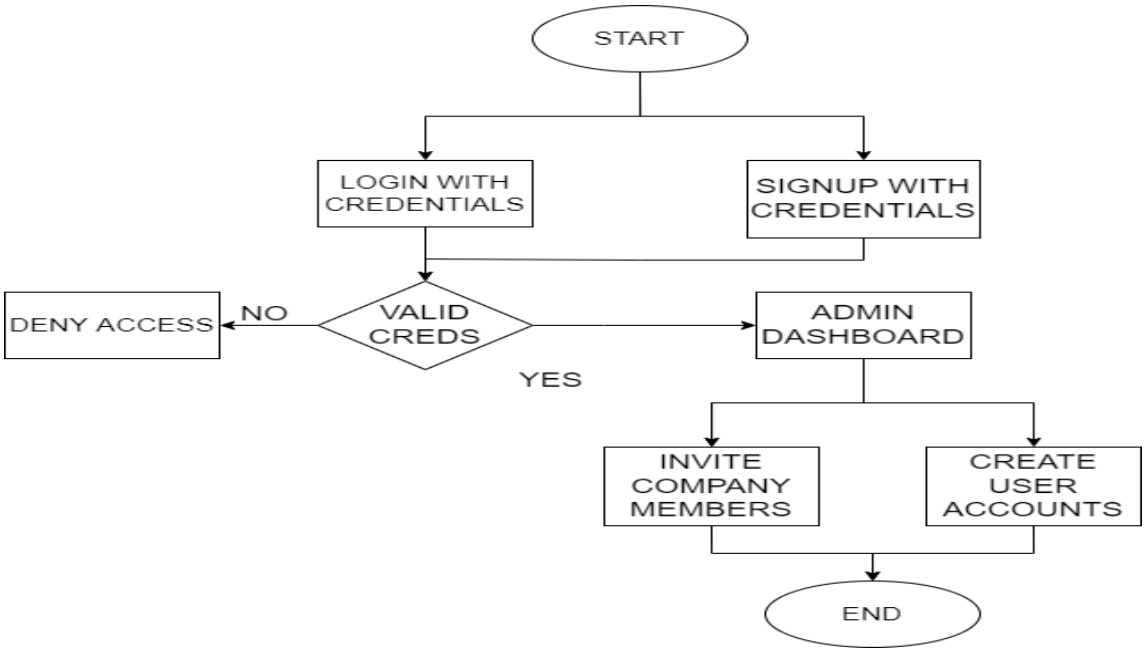


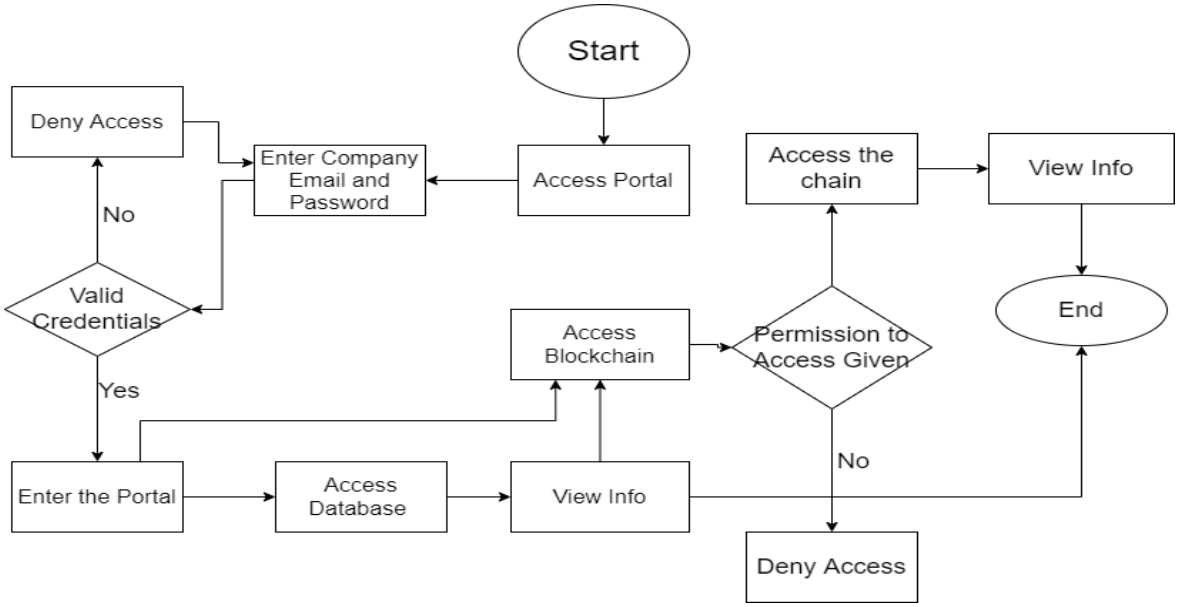
User stories:

ADMIN ACCESS			
7/25/22	1	I want to able to access the portal	
7/25/22	2	I want to be able to register a new account if I am new to the site.	
7/25/22	3	I want to be able to access the admin login page.	
7/25/22	4	I want to be able to Log in with my valid credentials.	
7/25/22	5	I want to be able to access the dashboard page.	
7/25/22	6	I want to be able to invite my team.	
7/25/22	7	I want to be able to make accounts for users.	
7/25/22	8	I want to be able to give access to the blockchains to users.	
USER LOG IN			
7/25/22	1	I want to be able to access the portal	
7/25/22	2	I want to be able to access the user login page	
7/25/22	3	I want to be able to log in to my account with my valid credentials.	
7/25/22	4	I want to be able to go to the dashboard page.	
ACCESSING INFORMATION AS A USER			
7/25/22	1	I want to be able to click on the company whose data I want to view.	
7/25/22	1	I want to be able to access the database for the residing information.	

7/25/22	2	I want to be able to access the blockchain for minutes if I have the access.	

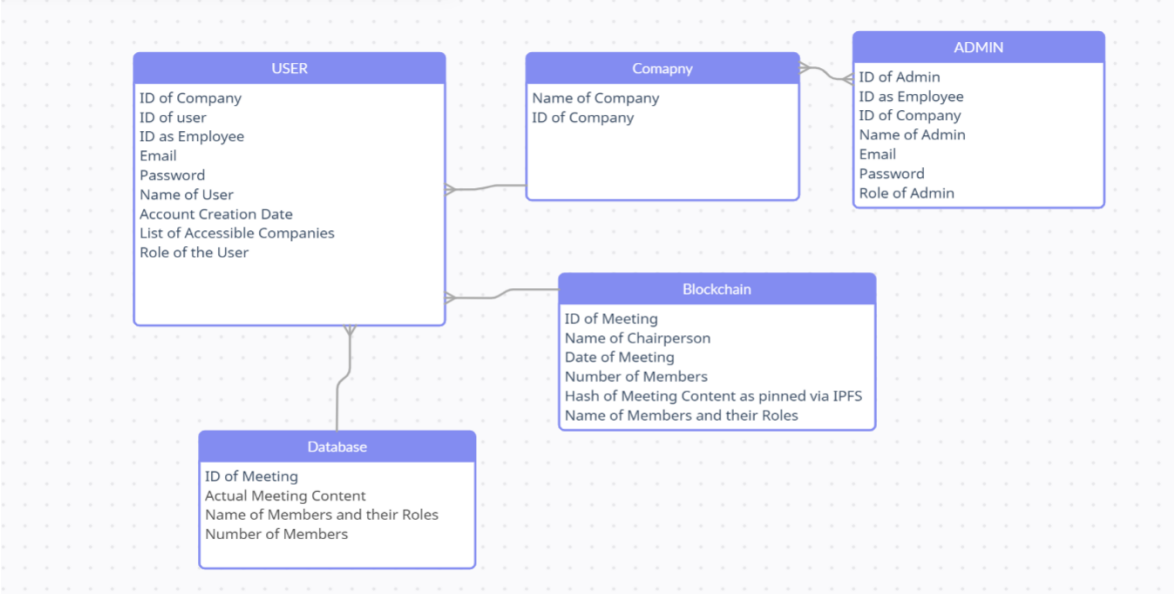
User flows:



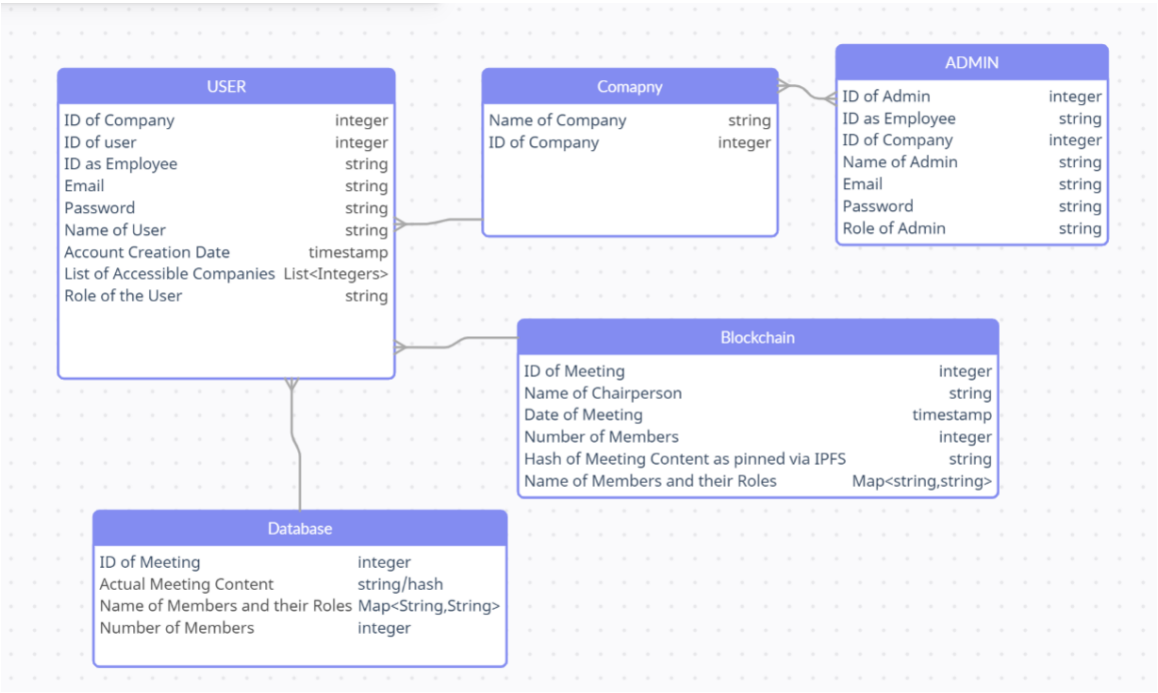


Data Model:

Logical Model:



Physical Model:



Data Dictionary:

Entity	Type	Mandatory	Frontend Exposed	Description
ADMIN				
Admin_ID	Integer	Y	N	Primary Key
Employee_ID	String	Y	Y	ID of Employee in the company
Company_ID	Integer	Y	N	Foreign Key
Name	String	Y	Y	Name of the Admin
Email	String	Y	N	Email credential for logging in
Password	String	Y	N	Password credential for logging in
Role	String	Y	Y	Role of the admin in the company
COMPANY				
Company_ID	Integer	Y	N	Primary Key
Name	String	Y	Y	Name of the Company
USER				
User_ID	Integer	Y	N	Primary Key
Company_ID	Integer	Y	N	Foreign Key
Employee_ID	String	N	Y	ID of the Employee in the company
Name	String	Y	Y	Name of the user
Email	String	Y	N	Email credential for logging in
Password	String	Y	N	Password credential for logging in
Created_At	timestamp	Y	N	The Timestamp at which account was created
Accessible Companies	List<Integers>	Y	Y	The list of companies whose meetings data is accessible to this user
Role	String	Y	Y	Role of the user in the company

DATABASE				
Meeting_ID	Integer	Y	N	Primary Key
Meeting Content	String/hash/PDF	Y	Y	The Actual Content related to the meeting
Member Names->Roles	Map<String,String>	Y	Y	A map consisting of the member names pointing to their roles
Number of Members	Integer	Y	Y	The Number of members in the meeting
BLOCKCHAIN				
Meeting_ID	Integer	Y	N	ID of the meeting from the database
Chairperson name	String	Y	Y	Name of the chairperson in the meeting
Date	timestamp	Y	Y	The timestamp at which the meeting was held
Number of Members	Integers	Y	Y	The Number of members in the meeting
Content Hash	String	Y	N	The Hashed IPFS Address of the Meeting content.
Member Names->Roles	Map<String,String>	Y	Y	A map consisting of the member names pointing to their roles

Resources:

- **Web3UIKit:**
  - A simple kit for Web3 apps with many built-in features like connect buttons, cards, input forms etc.
- **Tailwind CSS:**
  - A library with utility tools available for easy CSS markups.
- **Polygon Mainnet:**
  - A Layer2 side-chain solution to the Ethereum Mainnet for faster transactions and cheap gas fee for deploying the dapp.

- **Polygon Mumbai(Testnet):**
  - Testnet for testing and deployments.
- **MongoDB:**
  - A Centralized NoSQL Database to store the data related to Web2 contents.
- **Alchemy:**
  - A website for handling RPC URLs for blockchain test and main nets.
- **Polygon Faucet:**
  - Faucets to get test MATIC tokens for polygon.
- **PolygonScan**
  - Block explorer for polygon contracts

API documentation:


```
{

  "author": "Wahaj Javed",

  "details": "This implements simple mappings and events to accomplish the task",

  "kind": "dev",

  "methods": {

    "addAddressToCompany(address,string)": {

      "params": {

        "companyID": ": the ID of the company",

        "walletAddress": ": The address of the wallet of the user"
```

```
  }

},

"getCompanyIDFromWallet(address)": {

  "params": {

    "walletAddress": ": The address of the wallet of the user"

  },

  "returns": { "_0": "The list of company IDs" }

},

"getMeetingData(string,string)": {

  "params": {

    "companyID": ": the ID of the company",

    "meetingID": ": the ID of the meeting to be accessed"

  },

  "returns": { "_0": "The requested meeting data" }

},

"storeContent(string,string,string,string,uint256,string,string[],string[])": {
```

```
    "params": {

      "chairpersonName": ": The Name of the chairperson in the meeting",

      "companyID": ": The ID of the company",

      "contentHash": ": The Hash value of the content PDF",

      "meetingID": ": The ID of the meeting recieved from the database",

      "memberNames": ": The List of the names of all members",

      "numberOfMembers": ": The number of members who attended the meeting",

      "roles": ": The List of the roles of all members",

      "timestamp": ": The time at which meeting was held"

    }

   }

 },

 "title": "Safe Meetings",

 "version": 1

}
```

Smart Contracts:

Safe Meeting.sol:

```
// SPDX-License-Identifier:MIT
pragma solidity ^0.8.8;

/* Errors */
error SafeMeeting__DoesNotHaveAccess(address walletAddress, string companyID);
error SafeMeeting__AlreadyListed(address walletAddress, string companyID);
error SafeMeeting__MeetingNotListed(
    address walletAddress,
    string companyID,
    string meetingID
);

/**
 * @title Safe Meetings
 * @author Wahaj Javed
 * @notice This contract is for creating an untemperable storage for board meeting contents.
 * @dev This implements simple mappings and events to accomplish the task
 */
contract SafeMeeting {
    /* Events */
    event WalletListed(address indexed walletAddress, string indexed companyID);
    event DataUploaded(
        string indexed companyID,
        string indexed meetingID,
        string chairpersonName,
        string timestamp,
        uint256 numberOfMembers,
        string indexed contentHash,
        MemberRole[] memberToRoles
    );
```

```
    /* Structures */
    struct MemberRole {
        string memberName;
        string memberRole;
    }
    struct MeetingData {
        string meetingID;
        string chairpersonName;
        string timestamp;
        uint256 numberOfMembers;
        string contentHash;
        MemberRole[] memberToRoles;
    }
```

```
/* State Variables*/
// List of the company IDs accessible to the specific wallet
mapping(address => string[]) private s_addressToCompanyID;
// the company ID pointing to the data of the meetings list
mapping(string => mapping(string => MeetingData)) private s_idToMeetingData;
/* Modifiers */
modifier hasAccess(address walletAddress, string memory companyID) {
    string[] memory arr = s_addressToCompanyID[walletAddress];
    bool exists = false;
    for (uint256 i = 0; i < arr.length; i++) {
        if (keccak256(bytes(arr[i])) == keccak256(bytes(companyID))) {
            exists = true;
            break;
        }
    }
    if (!exists) {
        revert SafeMeeting__DoesNotHaveAccess(walletAddress, companyID);
    }
    _;
}
modifier alreadyListed(address walletAddress, string memory companyID) {
    string[] memory arr = s_addressToCompanyID[walletAddress];
    bool exists = false;
    for (uint256 i = 0; i < arr.length; i++) {
        if (keccak256(bytes(arr[i])) == keccak256(bytes(companyID))) {
            exists = true;
            break;
        }
    }
    if (exists) {
        revert SafeMeeting__AlreadyListed(walletAddress, companyID);
    }
    _;
}
```

```
/**
 * @notice : Method for returning the requested meeting data
 * @param companyID: the ID of the company
 * @param meetingID : the ID of the meeting to be accessed
 * @return The requested meeting data
 */
function getMeetingData(string memory companyID, string memory meetingID)
    public
    view
    hasAccess(msg.sender, companyID)
    returns (MeetingData memory)
{
    return s_idToMeetingData[companyID][meetingID];
}
```

```
/**
 * @notice : Method for storing the meeting data into the specific company ID List
 * @param companyID: The ID of the company
 * @param meetingID: The ID of the meeting recieved from the database
 * @param chairpersonName: The Name of the chairperson in the meeting
 * @param timestamp: The time at which meeting was held
 * @param numberOfMembers: The number of members who attended the meeting
 * @param contentHash: The Hash value of the content PDF
 * @param memberNames: The List of the names of all members
 * @param roles: The List of the roles of all members
 */
function storeContent(
    string memory companyID,
    string memory meetingID,
    string memory chairpersonName,
    string memory timestamp,
    uint256 numberOfMembers,
    string memory contentHash,
    string[] memory memberNames,
    string[] memory roles
) public hasAccess(msg.sender, companyID) {
    MeetingData storage meetData = s_idToMeetingData[companyID][meetingID];
    meetData.meetingID = meetingID;
    meetData.chairpersonName = chairpersonName;
    meetData.timestamp = timestamp;
    meetData.numberOfMembers = numberOfMembers;
    meetData.contentHash = contentHash;
    for (uint256 i = 0; i < memberNames.length; i++) {
        meetData.memberToRoles.push();
        MemberRole memory mem = MemberRole(memberNames[i], roles[i]);
        meetData.memberToRoles[i] = mem;
    }
    emit DataUploaded(
        companyID,
        meetingID,
        chairpersonName,
        timestamp,
        numberOfMembers,
        contentHash,
        meetData.memberToRoles
    );
}
```

```
    /**
     * @notice Method for registering a wallet address to allow access to company data
     * @param walletAddress: The address of the wallet of the user
     * @param companyID : the ID of the company
     */
    function addAddressToCompany(address walletAddress, string memory companyID)
        external
        alreadyListed(walletAddress, companyID)
    {
        s_addressToCompanyID[walletAddress].push(companyID);
        emit WalletListed(walletAddress, companyID);
    }

    /**
     * @notice Method for returning the list of company IDs accessible to the user
     * @param walletAddress: The address of the wallet of the user
     * @return The list of company IDs
     */
    function getCompanyIDFromWallet(address walletAddress)
        external
        view
        returns (string[] memory)
    {
        return s_addressToCompanyID[walletAddress];
    }
}
```