## 

## **Decentralised Voting System Using Ethereum, IPFS & Biometric Verification**

## **Submitted By**

| **Name** | **Roll No.** | **Responsibilities** |
| --- | --- | --- |
| **Chaitanya Kumar** | **23BCS064** | **Lead Developer – Designed and implemented core architecture: Smart Contracts, Frontend-Backend Integration,Face Verification, Voter Authentication (on-chain & off-chain), Vote Casting Logic, Result Aggregation & System Workflows** |
| **Aryan Pansare** | **23BCS040** | **Cybersecurity enhancements, migration from Ganache/Truffle to Hardhat, and contributed to issue resolution, verification and Smart Contract** |
| **Ayush Agarwal** | **23BCS046** | **Backend & Database development, MongoDB schema setup, admin dashboard bug fixes** |
| **Tanay Baviskar** | **23BCS247** | **Decentralized Identity (DID) setup, IPFS Pinning via Pinata, and integration with the file upload pipeline** |
| **Vaibhav Sahu** | **23BCS253** | **Admin Dashboard UI development, interface design improvements, and testing across admin workflows** |

## 

**Mentors:**

* **Atul Gupta Sir**
* **Dr. Ranjeet K. Ranjan Sir**

**Institute: Indian Institute of Information Technology, Design and Manufacturing (IIITDM) Jabalpur  
 Academic Year: 2024–2025  
 Date of Submission: 7 August 2025**

## **2. Abstract**

**This report presents the design and development of a fully functional Decentralised Voting System, implemented as a team project for the course *Blockchain and Decentralised Applications* at IIITDM Jabalpur.**

**The system leverages Ethereum smart contracts to ensure secure, tamper-proof, and transparent vote recording. To make the application user-friendly and scalable, we integrated a complete off-chain backend using Next.js APIs, MongoDB for persistence, and IPFS (via Pinata) for decentralized file storage. The application supports two roles — Admins (who manage elections, upload CSVs, and process candidates/voters) and Voters (who authenticate and cast votes securely).**

**By combining blockchain's immutability with the efficiency of off-chain infrastructure, our project offers a fast, cost-effective, and transparent voting platform. This report documents the complete technical workflow, architecture, modules, APIs, contract logic, and deployment strategy involved in building the system.**

## **3. Table of Contents**

1. **Title Page**
2. **Abstract**
3. **Table of Contents**
4. **Introduction**
5. **Objectives**
6. **Tech Stack Used**
7. **System Architecture Overview**
8. **Detailed Module-wise Description**
   * **Smart Contracts (Voting.sol)**
   * **Backend API (Next.js)**
   * **MongoDB Database**
   * **IPFS & Pinata Integration**
   * **Frontend (Next.js App)**
9. **Workflow of the System**
10. **Admin Flow**
11. **Voter Flow**
12. **On-Chain vs Off-Chain Summary**
13. **Security Considerations**
14. **Deployment Strategy (Docker)**
15. **Challenges Faced**
16. **Conclusion**
17. **Screenshots**
18. **References**

## **4. Introduction**

**Modern voting systems demand not only reliability and security but also public verifiability, tamper resistance, and decentralization. Traditional e-voting platforms often fall short in these areas due to centralized control, opacity, and vulnerability to manipulation.**

**To address these shortcomings, our team developed a Decentralised Voting System built on Ethereum smart contracts that handle critical on-chain logic such as candidate registration, voter authorization, and vote tallying. Off-chain layers using Next.js APIs, MongoDB, and IPFS provide the supporting infrastructure for data management, authentication, and file storage.**

**This hybrid approach allows the system to be secure, cost-effective, and scalable while maintaining decentralization where it matters most — the voting process itself.**

**This report outlines the entire lifecycle of the project: from architecture and module design to smart contract logic, backend workflows, UI design, and deployment using Docker. The goal is to present a comprehensive and transparent account of the system's implementation and functionality.**

## **5. Objectives**

**The primary objectives of this project are:**

* **✅ To build a secure, tamper-proof voting platform using blockchain technology.**
* **✅ To ensure transparency and public verifiability of the voting process via Ethereum smart contracts.**
* **✅ To integrate off-chain infrastructure (MongoDB, IPFS, Next.js) for efficient authentication, file storage, and database operations.**
* **✅ To support two roles in the system:**
  + **Admins: Upload and process candidates/voters, manage the election.**
  + **Voters: Authenticate, cast votes, and view results.**
* **✅ To provide real-time UI feedback and seamless user experience using modern React and Tailwind components.**
* **✅ To deploy the entire system using Docker for local testing and reproducibility.**

## **6. Tech Stack Used**

**The following technologies were used in the project:**

| **Layer** | **Technology Used** | **Purpose** |
| --- | --- | --- |
| **Smart Contract** | **Solidity, Hardhat** | **Voting logic, vote casting, candidate/voter management** |
| **Blockchain** | **Local Ethereum network (Hardhat node)** | **On-chain deployment & testing** |
| **Backend** | **Next.js (API Routes)** | **Admin & voter workflows, CSV processing, contract interactions** |
| **Frontend** | **React (Next.js App Directory), Tailwind CSS** | **Admin dashboard, Voter interface, results page** |
| **Database** | **MongoDB (via Mongoose)** | **Stores admin data, voter info, IPFS hashes** |
| **Storage** | **IPFS via Pinata API** | **Decentralized storage for candidate/voter CSVs** |
| **Deployment** | **Docker, Docker Compose** | **Containerized full-stack deployment** |
| **Libraries** | **ethers.js, bcrypt.js, JWT, mongoose** | **Contract interaction, password hashing, token auth** |
| **Others** | **Context API, Custom Hooks, Toast System** | **State management, UI feedback** |

## **7. System Architecture Overview**

**The system is structured in four core layers, each with a clear responsibility:**

### **1. Smart Contract Layer (On-chain)**

* **Written in Solidity and deployed on a local Ethereum (Hardhat) network.**
* **Handles:**
  + **Candidate registration**
  + **Voter authorization**
  + **Vote casting and counting**
  + **Result querying**

### **2. Backend API Layer (Off-chain)**

* **Built using Next.js API routes.**
* **Handles:**
  + **Admin login, password management**
  + **Uploading and processing candidate/voter CSVs**
  + **Voter authentication**
  + **Calling on-chain contract functions using ethers.js**
  + **Session management via JWT**

### **3. Persistence Layer**

* **MongoDB for storing:**
  + **Admin credentials**
  + **Voter records (email, hasVoted, isAuthorized)**
  + **IPFS hashes of uploaded CSVs**
* **Connected via a shared lib/mongodb.js client and Mongoose models**

### **4. Frontend Layer**

* **Built using Next.js App directory, React, and Tailwind CSS.**
* **Pages:**
  + **app/(auth)/: Admin and Voter sign-in/change-password**
  + **app/admin\_page/: Admin dashboard for managing elections**
  + **app/candidates/: Voter ballot UI with “Vote” button**
  + **app/results/: Displays vote tallies**
* **Reusable components (headers, footers, loaders, toasts)**

### **🧭 System Flow Diagram**

**[Admin Panel - Frontend]**

**│**

**▼**

**[Next.js API → upload.ts]**

**│**

**▼**

**[Pinata (IPFS Upload)]**

**│**

**▼**

**[MongoDB stores IPFS hash]**

**Then...**

**[Next.js API → processCandidates.ts/processVoters.ts]**

**│**

**├── [On-chain: call addCandidate/authorizeVoter]**

**└── [Off-chain: store records in MongoDB]**

**For Voter:**

**[Voter UI → /api/verify-voter]**

**▼**

**[Checks voter eligibility (MongoDB)]**

**▼**

**[Fetches ballot → casts vote]**

**▼**

**[On-chain: vote()]**

**+**

**[Off-chain: mark hasVoted = true]**

**Results Page:**

**[Fetch voteCount from Ethereum]**

**+**

**[Fetch candidate names from Mongo]**

## **8. Detailed Module-wise Description**

**This section explains each core module in-depth, including its file structure, responsibilities, and how it interacts with other parts of the system.**

### **8.1 Smart Contract Layer (Voting.sol)**

**📁 contracts/Voting.sol**

**This Solidity contract contains the core voting logic and stores all crucial election data on-chain.**

#### **🔐 Key Features:**

* **State Variables:**
  + **owner: Address of the contract deployer (admin rights).**
  + **candidates: Mapping of candidate ID → Candidate struct.**
  + **authorizedVoters: Mapping of address → eligibility.**
  + **hasVoted: Mapping to prevent double-voting.**
* **Structs:**
  + **Candidate { id, name, voteCount }**
* **Functions:**
  + **addCandidate(name): Admin adds candidate.**
  + **authorizeVoter(address): Admin authorizes a voter.**
  + **vote(candidateId): Casts a vote if eligible and not voted already.**
  + **getVoteCount(id): Returns the number of votes.**
  + **getWinner(): (Optional) Returns the candidate with highest votes.**
* **Events:**
  + **CandidateAdded, VoterAuthorized, VoteCast (for off-chain syncing)**
* **Security:**
  + **Uses onlyOwner modifier.**
  + **Uses ReentrancyGuard to protect against double-spend or call stacking attacks.**

**📄 Deployment Script:  
 📁 scripts/deploy.js — Compiles and deploys Voting.sol using Hardhat. Saves ABI and contract address to app/artifacts/.**

**📄 Config:  
 📁 hardhat.config.js — Contains local/devnet configuration, gas limits, and Solidity version.**

### **8.2 Backend API Layer (Next.js)**

**📁 pages/api/...**

**This layer handles communication between the front-end and:**

* **MongoDB**
* **IPFS (via Pinata)**
* **Ethereum Smart Contract**

#### **🔧 Admin APIs (/api/admin/)**

* **login.ts / logout-admin.ts / me.ts: Manage admin sessions via JWT cookies.**
* **upload.ts: Uploads candidate/voter CSV to IPFS using Pinata API.**
* **processCandidates.ts:**
  + **Reads candidate CSV from IPFS.**
  + **Registers each candidate on-chain via addCandidate().**
  + **Stores them in MongoDB.**
* **processVoters.ts:**
  + **Reads voter CSV from IPFS.**
  + **Calls authorizeVoter() on-chain.**
  + **Seeds voter records in MongoDB.**

#### **👤 Voter APIs**

* **verify-voter.ts: Authenticates voter off-chain using email & optional code.**
* **getCandidates.ts: Fetches list of candidates from MongoDB.**
* **processVoters.ts: Casts the vote both on-chain (vote()) and sets hasVoted = true off-chain.**

#### **📦 Utilities**

* **lib/mongodb.js: Shared MongoDB connector.**
* **models/: Mongoose schemas:**
  + **Admin.js, Voter.js, IpfsHash.js**

### **8.3 Persistence Layer (MongoDB & IPFS)**

#### **🧠 MongoDB Collections:**

* **admins: Stores hashed passwords and email of admins.**
* **voters: Stores email, hasVoted status, authorization status.**
* **ipfshashes: Maps uploaded CSV filename to returned IPFS hash.**

**🔌 Connected via mongoose using lib/mongodb.js.**

#### **📂 Pinata / IPFS Integration**

* **CSVs for candidates and voters are pinned to IPFS via REST API using:**
  + **PINATA\_API\_KEY and PINATA\_SECRET\_KEY**
* **The resulting IPFS hash is stored in MongoDB for retrieval and processing.**
* **All downloads use https://gateway.pinata.cloud/ipfs/<hash>.**

### **8.4 Front-End Layer (Next.js App + Tailwind)**

**📁 app/ — Based on the Next.js App Directory architecture.**

#### **🎯 Key Pages:**

* **app/(auth)/signin/page.tsx: Admin login UI.**
* **app/(auth)/change-password/page.tsx: Admin password update.**
* **app/signinusers/page.tsx: Voter login/verification.**
* **app/admin\_page/page.tsx: Admin dashboard.**
* **app/candidates/page.tsx: Voter interface with ballot and vote button.**
* **app/results/page.tsx: Displays vote tally and winner.**

#### **🧩 UI Components:**

**📁 components/ui/…**

* **Header, Footer, Loaders, Toast messages.**
* **Utility components styled with Tailwind CSS.**

#### **⚙️ State Management:**

* **Uses useState, useEffect, and Context API.**
* **Includes custom hooks like useMasonry.tsx, useMousePosition.tsx.**

### **8.5 How a Typical Flow Works**

#### **🧑‍💻 Admin:**

1. **Logs into dashboard.**
2. **Uploads CSV → backend pins to IPFS → saves hash in MongoDB.**
3. **Clicks Process → reads CSV from IPFS → adds entries to:**
   * **Blockchain (via addCandidate() / authorizeVoter())**
   * **MongoDB collections**

#### **🧑‍🦱 Voter:**

1. **Signs in via email → verified against MongoDB.**
2. **Fetches candidate list.**
3. **Clicks “Vote” → vote is:**
   * **Cast on-chain via vote()**
   * **hasVoted marked as true in MongoDB**
4. **Sees real-time confirmation and result page.**

## **9. Workflow of the System**

**The entire system integrates smart contract logic, off-chain database, file storage, and frontend UI through a structured series of workflows.**

**Below is a step-by-step breakdown of how the system functions from deployment to vote tallying.**

### **🔄 System Initialization:**

* **Smart contract (Voting.sol) is compiled and deployed using Hardhat.**
* **ABI and deployed contract address are saved to /app/artifacts/.**
* **Environment variables are configured in .env.local or docker-compose.**

### **🧑‍💼 Admin Workflow:**

* **Admin logs into the dashboard.**
* **Uploads CSVs for candidates and voters.**
* **Backend pins the files to IPFS via Pinata and stores the hash in MongoDB.**
* **Admin processes CSV:**
  + **Candidates → registered on-chain using addCandidate(), and stored off-chain.**
  + **Voters → authorized on-chain using authorizeVoter() and added to MongoDB.**

### **🧑‍🦱 Voter Workflow:**

* **Voter accesses the /signinusers page and enters email (and optional code).**
* **Backend verifies:**
  + **Email exists in MongoDB**
  + **Voter is authorized and has not already voted**
* **If successful, a session is created.**
* **Voter fetches candidate list via /api/getCandidates.**
* **Votes by clicking the vote button:**
  + **Triggers vote() on-chain**
  + **Marks hasVoted = true off-chain**
* **Confirmation shown via toast.**
* **Voter is redirected to results or a thank-you message.**

### **📊 Result Generation:**

* **Results page fetches vote counts:**
  + **On-chain: getVoteCount() from Ethereum contract.**
  + **Off-chain: Candidate metadata from MongoDB.**
* **Both datasets are merged and rendered for real-time tally display.**

## **10. Admin Flow**

**Here's a detailed breakdown of the admin's actions:**

| **Step** | **Action** | **Backend API** | **Description** |
| --- | --- | --- | --- |
| **1** | **Login** | **/api/admin/login** | **Authenticates admin using hashed credentials stored in MongoDB** |
| **2** | **Upload CSV** | **/api/admin/lists/upload** | **File sent to Pinata via API → returns IPFS hash → saved in ipfshashes** |
| **3** | **Process Candidates** | **/api/admin/lists/processCandidates** | **Reads IPFS hash → downloads CSV → for each entry:**  **• Calls addCandidate() on-chain**  **• Saves to candidates collection** |
| **4** | **Process Voters** | **/api/admin/lists/processVoters** | **Reads voter CSV → for each row:**  **• Calls authorizeVoter() on-chain**  **• Creates MongoDB voter record** |
| **5** | **Change Password** | **/api/admin/change-password** | **Verifies old password and saves new hashed password** |
| **6** | **View Logs & Toasts** | **Frontend** | **Displays confirmation or error messages using ToastProvider** |

**Admins can view real-time logs, disable buttons after one click (to avoid double submissions), and get feedback on processing status.**

## **11. Voter Flow**

**Voters interact only with the frontend (Next.js) and vote via smart contract in the background.**

| **Step** | **Action** | **API Involved** | **Description** |
| --- | --- | --- | --- |
| **1** | **Visit Login Page** | **/signinusers/page.tsx** | **Voter enters email/code** |
| **2** | **Verify Voter** | **/api/verify-voter** | **Backend checks:**  **• Email exists in voters**  **• Is authorized**  **• Has not voted already** |
| **3** | **Fetch Candidates** | **/api/getCandidates** | **Returns candidate list from MongoDB** |
| **4** | **Cast Vote** | **/api/processVoters** | **Backend:**  **• Sends vote(candidateId) to Ethereum**  **• Waits for transaction to be mined**  **• Sets hasVoted = true in MongoDB**  **• Returns transaction hash** |
| **5** | **UI Feedback** | **ToastProvider** | **Voter sees confirmation message like: “Vote Cast! Tx: 0xABC123…”** |
| **6** | **View Results** | **/results/page.tsx** | **Combines on-chain vote count with off-chain names from MongoDB** |

## **12. On-Chain vs Off-Chain Summary**

**The system intelligently separates on-chain and off-chain responsibilities to optimize for cost, performance, and security.**

| **Feature / Operation** | **On-Chain (Ethereum)** | **Off-Chain (MongoDB, IPFS, Next.js)** |
| --- | --- | --- |
| **Voting logic** | **vote() in Voting.sol — records immutable vote count** | **MongoDB mirror: sets hasVoted = true for UI responsiveness** |
| **Candidate registration** | **addCandidate() — stores candidate on-chain** | **Candidate metadata stored in MongoDB for display** |
| **Voter authorization** | **authorizeVoter() — sets eligibility flag** | **MongoDB stores voter details (email, status)** |
| **Vote tallying** | **getVoteCount() or getWinner()** | **Fetched from contract and matched with names from Mongo** |
| **CSV Uploads** | **❌ (not possible on-chain)** | **CSVs are pinned to IPFS via Pinata API** |
| **Authentication** | **❌ (public blockchain unsuitable)** | **JWT + cookies in Next.js backend with Mongo-stored credentials** |
| **Admin Management** | **❌** | **Handled fully off-chain (login, password change, session cookies)** |
| **Session handling** | **❌** | **JWT + Secure HttpOnly Cookies** |
| **File Storage** | **❌** | **IPFS (via Pinata) for decentralized CSV storage** |

**🔍 Why this hybrid model?**

* **✅ Gas-efficient: Off-chain DB avoids unnecessary gas costs.**
* **✅ Fast: MongoDB and sessions provide real-time feedback.**
* **✅ Secure: On-chain voting ensures immutability and auditability.**
* **✅ Scalable: Logic like verification and uploads don’t need blockchain.**

## **13. Security Considerations**

**Security is critical for any voting system. Here’s how our architecture defends against common vulnerabilities:**

### **🛡️ Smart Contract Security**

* **Access Control: Only contract owner (admin) can add candidates or authorize voters via onlyOwner modifier.**
* **Reentrancy Protection: vote() is wrapped with nonReentrant modifier using ReentrancyGuard from OpenZeppelin.**
* **Immutable Vote Records: Votes, once cast, cannot be altered or removed.**

### **🔐 Backend & API Security**

* **Hashed Passwords: Admin passwords are stored hashed via bcrypt.**
* **JWT Tokens: Sessions use JSON Web Tokens with secure, HttpOnly cookies.**
* **Input Validation: Email/code and file validations are performed before processing.**
* **Environment Isolation: Sensitive secrets (JWT, DB URL, Pinata keys, private keys) are stored in .env.local.**

### **🔏 Database Security**

* **Role-based Logic: Voter entries are validated before allowing vote.**
* **Vote Flag Mirror: Off-chain hasVoted flag helps avoid re-voting attempts instantly.**

### **🗂️ Decentralized File Storage**

* **All CSV files are stored on IPFS using Pinata—ensures tamper-proof, decentralized backups of election data.**

## **14. Deployment Strategy (Docker)**

**We used Docker to containerize and run all components locally, ensuring consistency, portability, and zero setup errors.**

### **🧱 Dockerfile (App Container)**

* **Based on node:18-alpine**
* **Copies project files**
* **Installs dependencies**
* **Builds the Next.js project**
* **Serves the app using next start**

### **⚙️ docker-compose.yaml**

**Defines three services:**

| **Service** | **Image/Build** | **Purpose** |
| --- | --- | --- |
| **voting-app** | **Built via Dockerfile** | **Next.js app (frontend + backend APIs)** |
| **voting-mongo** | **mongo:6.0** | **MongoDB database** |
| **voting-hardhat** | **Custom Hardhat image** | **Local Ethereum RPC network for Voting.sol** |

### **📁 .env.local**

**Stores environment variables:**

**env**

**MONGODB\_URI=mongodb://mongo:27017/decentralised-voting-system**

**JWT\_SECRET=your\_jwt\_secret**

**PINATA\_API\_KEY=your\_pinata\_api\_key**

**PINATA\_SECRET\_KEY=your\_pinata\_secret**

**NEXT\_PUBLIC\_PINATA\_GATEWAY=https://gateway.pinata.cloud/ipfs**

**VOTING\_ADDRESS=deployed\_contract\_address**

**PRIVATE\_KEY=deployer\_wallet\_key**

**HARDHAT\_RPC=http://hardhat:8545**

### **🧪 Run the Full Stack**

**bash**

**docker-compose up --build**

* **Wait until:**
  + **voting-app (Next.js) on port 3000**
  + **voting-mongo on 27017**
  + **voting-hardhat on 8545**

**Then:**

* **Visit http://localhost:3000**
* **Admin logs in → Uploads CSVs → Processes data**
* **Voter signs in → Casts vote → Results visible**

## **15. Challenges Faced**

**Building a decentralized voting system involved tackling multiple technical and architectural challenges across the stack:**

### **🔄 Smart Contract Development**

* **Designing minimal, gas-efficient storage structures (e.g., mappings) that still offer all necessary voting functionalities.**
* **Preventing double-voting via hasVoted logic both on-chain and off-chain.**
* **Securing admin-only functions using modifiers and avoiding reentrancy attacks.**

### **🧠 Off-Chain Voter Verification**

* **Ensuring that voter verification (via email/code) was done securely and without exposing sensitive data on-chain.**
* **Balancing between off-chain speed and on-chain immutability.**

### **⚠️ IPFS Integration**

* **Pinning and retrieving files using the Pinata API involved dealing with asynchronous operations, IPFS gateway delays, and error-handling.**
* **Mapping hashes to CSV files and syncing with MongoDB for accurate candidate/voter processing.**

### **🔐 Security & Authentication**

* **Implementing secure session management via JWTs + HttpOnly cookies to prevent tampering or spoofing.**
* **Hashing passwords properly with bcrypt and validating all user inputs to avoid vulnerabilities.**

### **🐳 Containerization with Docker**

* **Configuring services (Mongo, Hardhat, App) to run within the same Docker network and ensuring proper boot order.**
* **Volume handling for MongoDB persistence and contract deployments during development.**

### **🔄 Sync Between On-Chain and Off-Chain**

* **Ensuring that every on-chain action (e.g., vote, addCandidate) had a corresponding off-chain reflection for instant UI feedback.**
* **Managing cases where on-chain vote succeeded but Mongo update failed, or vice versa.**

## **16. Conclusion**

**The Decentralised Voting System demonstrates how blockchain technology can be effectively combined with modern web tools to solve real-world problems like election transparency and vote integrity.**

**By carefully splitting responsibilities between on-chain (immutability, transparency) and off-chain (efficiency, flexibility) layers, our system:**

* **Guarantees tamper-proof vote recording**
* **Enables secure, fast voter verification**
* **Offers admin-controlled election setup**
* **And provides a real-time, user-friendly frontend**

**This project not only helped us understand core blockchain principles, but also taught us how to build scalable, modular applications using full-stack web development and decentralized storage protocols like IPFS.**

**Our system is now a fully deployable, end-to-end blockchain voting solution, suitable for small-scale elections in communities, universities, and DAOs.**

## **17. References**

1. **Ethereum Documentation**
2. **Hardhat - Ethereum Development Environment**
3. **Next.js Documentation**
4. [**MongoDB Documentation**](https://www.mongodb.com/docs/)
5. **Pinata IPFS API Docs**
6. **ethers.js - Ethereum JavaScript Library**
7. **OpenZeppelin Smart Contract Library**