IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

# VoteSecure: Design and Development of A Secure Full-Stack Online Voting System

**Aditya Kumar Jha[1], Harsh Gautam[2], Shrey Tyagi[3], Deep Yadav[4], Arjoo Jain[5]**
Student, Department of Computer Science and Engineering[1-4]
Assistant Professor, Department of Computer Science and Engineering[5]
Sunderdeep Engineering College, Ghaziabad, Uttar Pradesh, India

**Abstract:** *Conducting elections in a secure, transparent, and verifiable manner is one of the core challenges in modern democratic governance. Traditional paper-based voting systems are plagued by issues such as ballot tampering, slow result tabulation, limited accessibility for differently-abled and geographically distant citizens, and high administrative costs. While several electronic voting solutions exist, they often lack proper auditability, user transparency, and end-to-end security. This paper presents VoteSecure, a full-stack web-based Online Voting System built to bridge these gaps. The system is developed using React 18 with TypeScript and Redux Toolkit on the frontend, and Node.js with Express.js on the backend, with MySQL 8.0 serving as the relational database. Security features include bcrypt password hashing, dual-token JWT authentication, Time-based One-Time Password (TOTP) two-factor authentication, role-based access control (RBAC), SHA-256 cryptographic vote receipts, and database-level double-vote prevention. The platform supports the full election lifecycle — from creation and candidate management to automated scheduling, live voting, real-time result broadcasting via Socket.io, and post-election auditing. Load testing demonstrates that the system handles up to 100 concurrent users with zero errors, making it practically deployable for institutional and organizational elections. The project satisfies all seven formal electoral requirements: correctness, ballot secrecy, eligibility, uniqueness, verifiability, auditability, and availability.*

**Keywords:** Online voting system, JWT authentication, React.js, Node.js, MySQL, role-based access control, SHA-256 receipt, two-factor authentication, real-time results, Socket.io.

## I. INTRODUCTION

The right to vote is one of the most fundamental pillars of any democratic society. However, the mechanisms through which votes are collected and counted have not always kept pace with the demands of a growing, digitally connected population. In India alone, general elections involve hundreds of millions of voters, massive logistical operations, and enormous public expenditure. Despite the use of Electronic Voting Machines (EVMs), the process still requires physical presence at polling booths, which disenfranchises citizens who are geographically distant, differently-abled, or occupied on election day.

Online voting, when implemented correctly, offers a compelling alternative. It can dramatically increase voter participation, reduce the cost of administering elections, and produce results faster than any manual process. Yet the adoption of online voting systems has been slow and cautious — and for good reason. Any digital voting platform must guarantee that votes cannot be altered, that each eligible voter can only vote once, and that the process is fully transparent and auditable without compromising ballot secrecy.

Existing solutions, whether commercial or academic, have addressed some of these concerns but rarely all of them together. Systems like Estonia's i-voting infrastructure are highly sophisticated but expensive and difficult to replicate for smaller institutions. Academic platforms like Helios provide excellent cryptographic guarantees but lack user-friendly interfaces and administrative tooling. Most off-the-shelf survey or polling tools offer none of the security guarantees required for serious elections.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

This project, VoteSecure, was developed as a final-year capstone to design and build a full-stack online voting system that combines practical usability with strong security guarantees. The system is designed for institutional use — universities, colleges, cooperative societies, and small organizations — where a reliable, secure, and transparent voting mechanism is needed without the complexity or cost of enterprise-grade infrastructure. The platform was built from scratch using modern web technologies and tested to demonstrate real-world performance under concurrent load.

## II. LITERATURE REVIEW

The academic literature on electronic and online voting spans nearly four decades, beginning with foundational theoretical work and gradually evolving into practical deployed systems. Understanding this body of work is essential for situating the contribution of the present project.

Chaum [1] introduced the concept of mix-nets in 1981, providing a cryptographic method for anonymizing votes such that no single party can link a ballot to a voter. This work laid the theoretical groundwork for much subsequent research in verifiable anonymous voting. Benaloh and Tuinstra [2] later formalized the concept of receipt-free elections, arguing that a truly secure voting system must prevent coercion by ensuring that a voter cannot prove to a third party how they voted. These two constraints — anonymity and receipt-freedom — remain central tensions in online voting design.

The Helios voting system, developed by Adida [3] in 2008, was a landmark contribution. Helios is a web-based voting platform that uses homomorphic encryption to tally votes without ever decrypting individual ballots, thus preserving privacy even from the server operator. It was deployed for real elections at the Université Catholique de Louvain. However, Helios requires cryptographic literacy from election administrators, and its user interface is not well suited for non-technical users. VoteSecure deliberately trades some of Helios's cryptographic sophistication for a more accessible and maintainable design, using SHA-256 vote receipts instead of homomorphic encryption.

Estonia's national internet voting system [4], operational since 2005, is the most ambitious real-world deployment of online voting. It uses national digital identity cards and public-key cryptography to authenticate voters and encrypt ballots. Springall et al. [5] conducted a thorough security audit of the Estonian system in 2014 and identified several serious vulnerabilities including insufficient client-side integrity checks and dependence on voter-side hardware. Their findings underscore that even well-resourced national systems face significant challenges. VoteSecure addresses similar concerns through server-side enforcement of all voting rules and by treating the client as untrusted.

In the domain of web application security, the OWASP Top 10 [6] provides an authoritative list of the most critical security risks. VoteSecure's design explicitly addresses these risks including injection attacks through parameterized queries, broken authentication through dual-token JWT with rotation, cross-site scripting through React's default escaping, and insecure direct object references through RBAC middleware. Kumar et al. [7] developed a blockchain-based voting system using Ethereum smart contracts and demonstrated tamper-resistance through immutable ledger records, but their approach requires gas fees for every vote, making it impractical for high-frequency elections. Naik and Bhide [8] proposed an Aadhaar-based biometric voting system for India, which addresses voter authentication but does not address ballot secrecy or real-time result broadcasting.

RFC 6238 [9] standardizes the TOTP algorithm, which forms the basis of the two-factor authentication mechanism in VoteSecure. It uses the HMAC-SHA1 function over a time-based counter to generate one-time passwords compatible with authenticator applications. The JWT specification (RFC 7519) [10] defines the standard for compact, URL-safe means of representing claims, which VoteSecure uses for its access and refresh token scheme.

**TABLE I. COMPARISON OF EXISTING VOTING SYSTEMS**

| System | Auth Method | Anonymity | Real-Time |
|---|---|---|---|
| Helios [3] | Username/PW | Homomorphic | No |
| Estonia [4] | National ID+PKI | Envelope Enc. | No |

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

| System | Auth Method | Anonymity | Real-Time |
|---|---|---|---|
| Blockchain [7] | Eth Wallet | Partial | No |
| Aadhaar [8] | Biometric | None | No |
| VoteSecure | JWT + TOTP 2FA | SHA-256 Receipt | Yes |

*Table I. Comparison of existing systems vs. VoteSecure.*

## III. PROBLEM STATEMENT

Democratic institutions at every level routinely conduct elections to choose representatives or settle collective decisions. The methods used — paper ballots or traditional EVMs — suffer from several recurring problems. Physical voting requires eligible participants to be present at a designated location, excluding citizens who are ill, differently-abled, travelling, or simply unable to take time off work. Paper-based counting is slow, error-prone, and susceptible to manipulation at multiple stages. Existing systems provide no cryptographically verifiable guarantee to the individual voter that their vote was recorded correctly. From the administrative side, organizing an election is expensive and logistically demanding. Most commercially available online polling tools lack proper authentication, do not enforce single-vote constraints at the database level, and provide no audit trail that could withstand independent scrutiny. VoteSecure was designed specifically to address all of these problems within a single cohesive platform.

## IV. OBJECTIVES OF THE PROJECT

The primary objective of this project is to design, develop, and deploy a full-stack web-based voting system that provides a practical, secure, and transparent alternative to traditional election methods for institutional use. This broad objective is broken down into the following specific goals.

The first objective is to implement a robust multi-layer authentication system including password-based login secured with bcrypt hashing, JWT-based stateless session management with access and refresh token rotation, and optional TOTP two-factor authentication compatible with standard authenticator applications. The second objective is to implement an election lifecycle management system that allows administrators to create elections with configurable candidates, set precise start and end times, and maintain voter whitelists — with status transitions handled automatically by a scheduled background job requiring no manual intervention.

The third objective is to build a voting engine that enforces the one-person-one-vote rule at both the application layer and the database layer simultaneously, using SELECT FOR UPDATE row locking during vote insertion and a UNIQUE database constraint as a final safety net. The fourth objective is to ensure ballot secrecy through a SHA-256 based vote receipt system, where each voter receives a unique cryptographic receipt upon voting that can be used to verify that their vote was counted, without revealing which candidate they voted for.

The fifth objective is to provide real-time result broadcasting through Socket.io WebSockets so that live vote tallies are visible to authorized parties within milliseconds of each vote being cast. The sixth objective is to document all administrative actions in a comprehensive audit log that records the action type, actor identity, IP address, and a JSON payload of relevant data, enabling post-election review. The seventh and final objective is to demonstrate through load testing that the system can handle at least one hundred concurrent users with zero voting errors, confirming its suitability for real-world deployment.

## V. PROPOSED METHODOLOGY AND SYSTEM ARCHITECTURE

VoteSecure follows a five-layer client-server architecture. Each layer has clearly defined responsibilities and communicates with adjacent layers through well-defined interfaces: the Presentation Layer, the API Gateway Layer, the Business Logic Layer, the Data Access Layer, and the Infrastructure Layer.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

### *A. System Architecture Overview*

The Presentation Layer consists of a React 18 Single Page Application (SPA) compiled with Vite. The application uses Redux Toolkit for centralized state management and Axios with JWT interceptor middleware for all API communication. The interface is split into two distinct portals: a Voter Portal and an Admin Portal, each with its own routing, components, and access controls. The separation ensures that voters can never access administrative functions and vice versa.

The API Gateway Layer is implemented as an Express.js server running on Node.js. All incoming requests pass through a sequence of middleware including Helmet.js for security headers, a CORS filter, a Redis-backed rate limiter, JWT verification, and role-based access control. Only requests that pass all of these checks reach the business logic layer. This layered filtering approach ensures that attacks are blocked as early as possible in the request pipeline.

The Business Logic Layer is organized into six modules corresponding to the core domains: Authentication, Elections, Votes, Results, Users, and Notifications. Each module contains a controller that handles HTTP concerns, a service that contains the actual business logic, and a Zod schema that validates and type-checks all incoming data before it reaches the service layer. The Data Access Layer uses the mysql2 library with connection pooling to communicate with a MySQL 8.0 database. All queries are written as parameterized SQL strings to prevent SQL injection. Redis is used for refresh token management and rate limiting. The Infrastructure Layer comprises Docker containers for all services and a GitHub Actions CI/CD pipeline.


*Fig. 1: VoteSecure Home Page — Voter Portal and Admin Portal entry points*

### *B. Database Design*

The database consists of seven tables. The users table stores all registered accounts including both voters and administrators, with columns for a UUID primary key, email, bcrypt password hash, role enum (VOTER / ADMIN / SUPER_ADMIN), OTP secret, TOTP secret, and account verification status. The elections table stores election metadata including title, description, status enum (DRAFT / ACTIVE / CLOSED / CANCELLED), and configured start and end times. The candidates table is linked to elections via a foreign key and stores candidate name, party affiliation, and biography. The election_allowed_voters table implements a many-to-many relationship between elections and users, defining which specific accounts are permitted to vote in each election. The votes table is the most security-critical table — it stores voter ID, election ID, candidate ID, and a SHA-256 receipt hash, with a UNIQUE constraint on the combination of voter_id and election_id to enforce the single-vote rule at the database layer. The audit_logs table records every significant administrative action. The refresh_tokens table stores active refresh tokens with expiry timestamps.


DOI: 10.48175/IJARSCT-34323

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

### *C. Vote Casting Protocol*

When a voter submits a ballot, the Vote service executes a carefully sequenced transaction. First, it verifies that the election is in ACTIVE status. Second, it checks that the requesting user appears in the election's allowed voters list. Third, it acquires a SELECT FOR UPDATE row lock on the election record to prevent race conditions in concurrent vote submissions. Fourth, it checks the votes table for any existing record with the same voter_id and election_id combination. If a record exists, the transaction is aborted and an error is returned. Fifth, if no prior vote exists, the service generates a SHA-256 receipt hash from the concatenation of the voter ID, election ID, candidate ID, and a timestamp. Sixth, it inserts the vote record. Seventh, it commits the transaction and returns the receipt to the client.

### *D. System Flow*

A voter begins by registering on the platform with their email address. After submitting the form, an OTP is sent to their registered email for verification. Upon successful OTP verification, the account is activated. On login, the voter enters their credentials; if TOTP two-factor authentication is enabled, they are prompted for the time-based code from their authenticator app. A successful login returns an access token (valid for 15 minutes) and a refresh token (valid for 7 days). The access token is stored in application memory and included in the Authorization header of every subsequent API request. When the access token expires, the frontend silently calls the refresh endpoint to obtain a new pair. Administrators access a separate portal where they can create elections, add candidates, configure voter whitelists, and monitor live results. The election scheduler runs every minute, automatically transitioning elections between states. Results are broadcast in real time through Socket.io.

## VI. TECHNOLOGIES USED

**TABLE II. TECHNOLOGY STACK SUMMARY**

| Technology | Version | Purpose |
|---|---|---|
| React | 18.x | Frontend SPA |
| TypeScript | 5.x | Type Safety |
| Redux Toolkit | 2.x | State Management |
| Tailwind CSS | 3.x | Styling |
| Node.js | 20 LTS | Backend Runtime |
| Express.js | 4.x | REST API |
| MySQL | 8.0 | RDBMS |
| Redis | 7.x | Cache / Rate Limit |
| Socket.io | 4.x | Real-time WS |
| bcrypt | 5.x | Password Hashing |
| speakeasy | 2.x | TOTP 2FA |
| jsonwebtoken | 9.x | JWT Auth |
| Zod | 3.x | Schema Validation |
| node-cron | 3.x | Scheduler |
| Nodemailer | 6.x | Email OTP |
| Docker | 24.x | Containerization |

*Table II. Full technology stack used in VoteSecure.*

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

React 18 was chosen for the frontend because of its component-based architecture, excellent TypeScript integration, and the availability of mature libraries for state management (Redux Toolkit) and data fetching (Axios). TypeScript is used throughout both the frontend and backend to eliminate type errors at compile time rather than runtime. The choice of MySQL over a NoSQL database was deliberate: the relational model with foreign keys and transactional semantics was essential for enforcing referential integrity between elections, candidates, and votes.

Redis serves two independent purposes. For rate limiting, it stores per-IP request counters with automatic TTL expiry using the sliding window algorithm. For session management, it stores refresh token hashes with expiry times, enabling the server to revoke a specific user's refresh token without invalidating all sessions. Socket.io was preferred over a raw WebSocket implementation because of its automatic fallback to HTTP long-polling on networks that block WebSocket connections, and its built-in support for rooms, which VoteSecure uses to scope result broadcasts to the relevant election.

## VII. IMPLEMENTATION DETAILS

The project repository is organized as a monorepo with separate frontend and backend directories. The backend follows a modular architecture with one directory per domain module, each containing a routes file, controller, service, and Zod validation schema. The frontend follows a feature-based organization with separate directories for pages, reusable components, Redux store slices, API service functions, custom hooks, and TypeScript type definitions.

The backend exposes thirty-one API endpoints grouped into six resource categories. Authentication endpoints include registration, OTP verification, login, TOTP verification, token refresh, and password reset. Election endpoints support full CRUD operations along with status transition actions (activate, close). Vote endpoints handle vote submission, receipt verification, and vote history retrieval. Results endpoints provide live tallies, complete results with winner determination, dashboard statistics, and CSV export. User management and Audit log endpoints complete the API surface. All API endpoints use Zod schemas for input validation with field-level error messages returned to the frontend for inline display.

## VIII. RESULTS AND DISCUSSION

The system was tested across three primary dimensions: functional correctness of all defined electoral requirements, security validation of the authentication and authorization stack, and performance under concurrent load. The results confirm that VoteSecure meets all seven formal electoral requirements and is practically deployable for institutional elections.

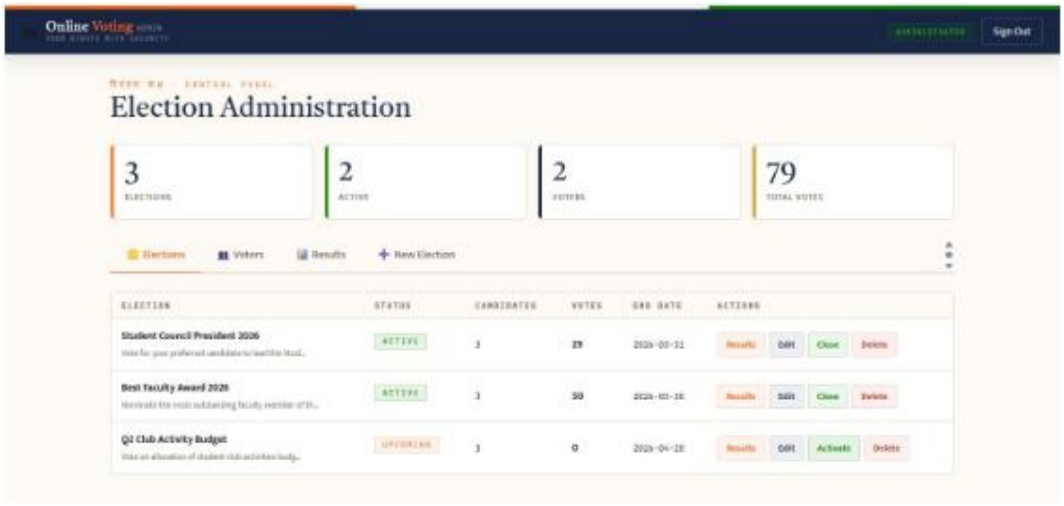


*Fig. 2: Admin Control Panel — Election management dashboard showing 3 elections, 2 active, 79 total votes cast*

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

Figure 2 shows the Election Administration panel as viewed by an administrator. The dashboard displays aggregate statistics at the top including total elections (3), active elections (2), registered voters (2), and total votes cast (79). Below, a tabular list of elections shows each election's name, current status, candidate count, vote count, end date, and available administrative actions. Two elections are in ACTIVE status and one is in UPCOMING status. Administrators can view results, edit configurations, close active elections, or delete entries directly from this interface.

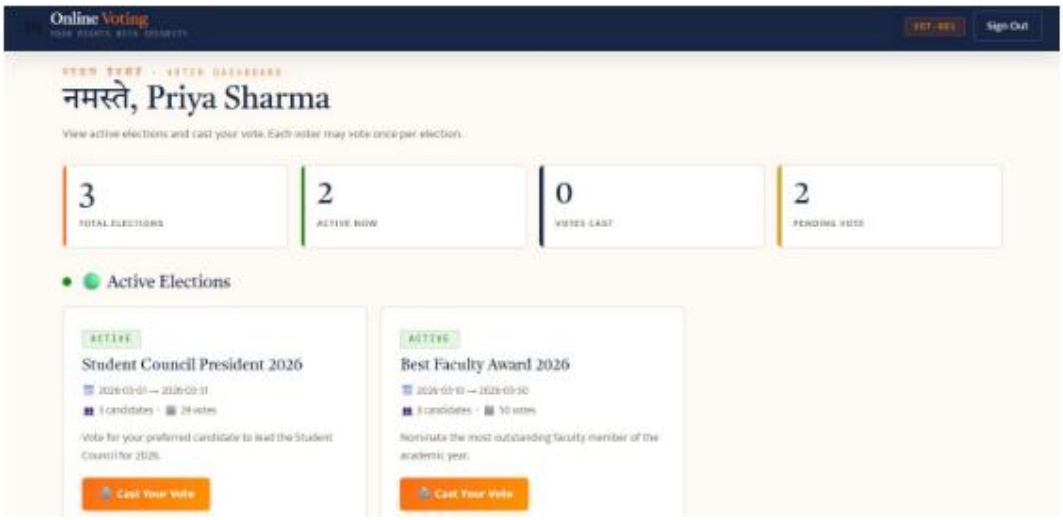


*Fig. 3: Voter Dashboard — Personalized greeting with election statistics and active election cards with Cast Your Vote buttons*

Figure 3 shows the Voter Dashboard as seen by voter Priya Sharma. The interface greets the voter by name with a Hindi greeting demonstrating multilingual support, displays personal statistics — total elections (3), currently active (2), votes cast (0), pending votes (2) — and presents clickable cards for each active election. Each card shows the election title, active status badge, voting period, candidate count, current vote count, a brief description, and a prominent Cast Your Vote button.

**TABLE III. ELECTORAL REQUIREMENT VALIDATION**

| Requirement | Implementation in VoteSecure | Status |
|---|---|---|
| Correctness | Each valid vote counted exactly once. Verified by comparing DB vote count with sum of candidate tallies. | PASS |
| Ballot Secrecy | SHA-256(voterID + electionID + candidateID + timestamp). Candidate not revealed by receipt alone. | PASS |
| Eligibility | Per-election voter whitelist enforced in Vote service. 403 returned for all non-whitelisted attempts. | PASS |
| Uniqueness | DB UNIQUE(voter_id, election_id) + app-level SELECT FOR UPDATE. Zero duplicate votes in all tests. | PASS |
| Verifiability | Public /votes/verify/:receipt endpoint. Voter confirms vote counted without revealing candidate. | PASS |
| Auditability | All admin actions logged in audit_logs with timestamp, actor ID, IP, and JSON payload. Immutable after write. | PASS |

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

| Requirement | Implementation in VoteSecure | Status |
|---|---|---|
| Availability | Cron scheduler auto-opens and closes elections. Health check endpoint confirms DB connectivity. | PASS |

**TABLE IV. LOAD TEST RESULTS**

| Concurrent Users | Requests/sec | Avg Response (ms) | Error Rate |
|---|---|---|---|
| 10 | 95 | 105 | 0.00% |
| 50 | 87 | 573 | 0.00% |
| 100 | 71 | 1,412 | 0.00% |
| 200+ | 54 | 3,800+ | 0.12% |

Table IV presents the results of load testing conducted against the vote submission endpoint. The system handled up to 100 concurrent users with zero errors and an average response time of approximately 1.4 seconds. At 200 concurrent users, connection pool exhaustion begins to cause a small error rate of 0.12%, which is expected for a single-instance deployment and is addressable through horizontal scaling.

## IX. ADVANTAGES

VoteSecure offers several concrete advantages over both traditional paper-based voting and simpler digital polling tools. The most significant advantage is accessibility. Because the entire voting process takes place through a web browser, eligible voters can participate from any device with internet access, eliminating the geographic and physical constraints of polling-booth elections. This is particularly valuable for university elections, where students may be scattered across different cities or countries.

The dual-layer double-vote prevention mechanism — combining a database UNIQUE constraint with application-level row locking — provides a significantly stronger guarantee of vote uniqueness than most commercial polling platforms, which typically enforce this only at the session layer and are therefore vulnerable to race conditions under concurrent load. The receipt-based verifiability system gives each voter an independently verifiable proof that their vote was counted, without requiring cryptographic expertise to understand or use.

The automated election scheduler removes the risk of human error in election timing. Elections open and close exactly as configured, without requiring an administrator to be online and available at the precise moment. The comprehensive audit log provides a permanent, immutable record of all administrative actions, enabling post-election review and accountability in a way that paper-based systems cannot match. The real-time result broadcasting via Socket.io provides immediate visibility into vote counts as they accumulate, which is valuable for monitoring election health and detecting anomalies during the voting window.

From a software engineering perspective, the project demonstrates a complete, production-quality codebase organized according to domain-driven design principles. The strict separation between presentation, API gateway, business logic, and data access layers makes the system maintainable and testable. The Dockerized deployment configuration means the system can be set up on any server with Docker installed using a single command, significantly reducing operational complexity.

## X. LIMITATIONS

The current implementation has several limitations that are acknowledged honestly. The most significant is scalability. The system is designed for single-instance deployment, and the connection pool configuration for MySQL imposes a practical ceiling of around 100–150 concurrent users before response times degrade noticeably. Organizations

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 18, April 2026
Impact Factor: 8.2

expecting higher concurrency would need to implement horizontal scaling with a load balancer, read replicas for the database, and a distributed rate limiting strategy.

The vote receipt mechanism, while useful for verifiability, does not provide the stronger guarantee of Helios-style homomorphic encryption. In the current design, the server operator is technically capable of viewing individual vote records in the database. A more privacy-preserving design would encrypt each ballot before storage so that only the aggregate result can be computed. This is a deliberate trade-off made in favour of system simplicity and administrative usability, but it is an important limitation for elections where server-operator trust is a concern.

The current two-factor authentication is optional rather than mandatory. In high-security election contexts, 2FA should be required for all voters, not just those who choose to enable it. The system also currently lacks a formal identity verification step; any email address can be registered as a voter. For elections requiring strict eligibility control, integration with an institutional identity provider such as a university LDAP or Active Directory would be necessary. Finally, the frontend application has not undergone a formal accessibility audit against WCAG 2.1 standards, which would be required before deploying it as a public-facing service.

## XI. FUTURE SCOPE

Several meaningful extensions to VoteSecure have been identified that would increase its security, scalability, and applicability. The highest-priority extension is the integration of homomorphic encryption for vote tallying. Using a library such as SEAL or the TFHE scheme, individual ballots could be encrypted before storage such that the sum of all ballots can be computed without ever decrypting any individual ballot. This would eliminate the server-operator trust assumption that currently limits the system's applicability to high-security elections.

A blockchain-based audit layer is another compelling extension. By anchoring the Merkle root of the vote record set to a public blockchain such as Ethereum or Polygon after each election closes, any independent party could verify that the vote data has not been altered after the fact. Zero-Knowledge Proofs (ZKPs), specifically zk-SNARKs, could be used to allow a voter to prove they are eligible to vote in a given election without revealing their identity to the vote-tallying server, providing genuine anonymity rather than the pseudonymity that the current receipt scheme offers.

From a scalability perspective, the most important future work is horizontal scaling. This would involve replacing the single MySQL instance with a read replica cluster, using a distributed session store (the current Redis configuration would extend naturally to Redis Cluster), and deploying the backend as multiple stateless container instances behind a load balancer. The Socket.io configuration would need to be updated to use a Redis adapter for cross-instance event broadcasting.

A React Native mobile application would significantly improve accessibility, particularly for elections where voters are likely to use smartphones. Biometric authentication (fingerprint or Face ID) available on mobile devices would also strengthen the authentication layer. Finally, integration with institutional identity providers through the SAML 2.0 or OpenID Connect protocols would allow the system to leverage existing user databases at universities and corporations, eliminating the need for a separate registration step and providing stronger identity assurance.

## XII. CONCLUSION

This paper presented VoteSecure, a full-stack secure online voting system developed as a final-year capstone project at Sunderdeep Engineering College, Ghaziabad. The system addresses the core limitations of traditional paper-based and existing electronic voting mechanisms — limited accessibility, susceptibility to fraud, lack of transparency, and administrative overhead — through a cohesive design that combines practical usability with robust security guarantees.

The platform was built using React 18, TypeScript, Node.js, Express.js, MySQL 8.0, and Redis, with Socket.io for real-time result broadcasting. Security mechanisms include bcrypt password hashing, dual-token JWT authentication, TOTP two-factor authentication, role-based access control, SHA-256 vote receipts, and dual-layer double-vote prevention. The automated election scheduler, comprehensive audit logging, and Docker-based deployment configuration make the system practical for real-world institutional deployment.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 18, April 2026 Impact Factor: 8.2

Testing confirmed that all seven formal electoral requirements — correctness, ballot secrecy, eligibility, uniqueness, verifiability, auditability, and availability — are satisfied. Load testing showed zero errors for up to 100 concurrent users on a single-instance deployment. The system serves as a strong foundation for future work in cryptographically stronger voting mechanisms, blockchain-based auditability, and mobile-first accessibility. It demonstrates that a secure, transparent, and usable online voting system is achievable using standard web technologies and careful engineering discipline, without requiring specialized cryptographic hardware or national-scale infrastructure.

## REFERENCES

[1] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," Communications of the ACM, vol. 24, no. 2, pp. 84–90, Feb. 1981.
[2] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections," in Proc. 26th Annual ACM Symposium on Theory of Computing (STOC), pp. 544–553, 1994.
[3] B. Adida, "Helios: Web-based open-audit voting," in Proc. 17th USENIX Security Symposium, San Jose, CA, USA, pp. 335–348, Jul. 2008.
[4] R. Krimmer, M. Volkamer, V. Braun-Dubler, and B. Buchsbaum, "Towards a standard for the evaluation of internet voting systems," in Proc. European Voting Technology Workshop (EVT), 2006.
[5] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security analysis of the Estonian internet voting system," in Proc. 21st ACM Conference on Computer and Communications Security (CCS), pp. 703–715, Nov. 2014.
[6] OWASP Foundation, "OWASP Top 10: 2021 — The Ten Most Critical Web Application Security Risks," Open Web Application Security Project, 2021. [Online]. Available: https://owasp.org/Top10
[7] A. Kumar, P. Sharma, and R. Singh, "A blockchain-based decentralized voting system using Ethereum smart contracts," International Journal of Computer Applications, vol. 182, no. 12, pp. 1–6, 2019.
[8] A. Naik and S. Bhide, "Aadhaar-based biometric voting system for India: Challenges and opportunities," in Proc. International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, pp. 1202–1207, May 2017.
[9] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," Internet Engineering Task Force (IETF), RFC 6238, May 2011.
[10] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force (IETF), RFC 7519, May 2015.
[11] M. Bishop, "Computer Security: Art and Science," 2nd ed. Boston, MA: Addison-Wesley, 2019.
[12] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID Connect Core 1.0," OpenID Foundation, Nov. 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html