IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

A Cloud-Native Full-Stack Java Framework for Secure and Transparent Online College Election Management on AWS

VEERA SIVANI[1], K. LAKSHMI SAI SREE*[2]

PG Scholar Department of Computer Science S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous) Penugonda, Affiliated to Adikavi Nannaya University[1]

*Associate Professor, Department Of Master of Computer Applications, S.V.K. P & Dr. K. S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[2]

Abstract: The digitization of democratic processes within higher education institutions demands robust, scalable, and tamper-resistant platforms. Conventional paper-based and legacy digital election systems in academic environments are susceptible to vote manipulation, ballot stuffing, administrative overhead, and limited accessibility. This paper presents a cloud-native, full-stack election management framework designed specifically for college environments, deployed on Amazon Web Services (AWS) Elastic Beanstalk. The proposed system integrates a Spring Boot 3.x RESTful backend with a React-based single-page application (SPA) frontend, secured through JSON Web Token (JWT) stateless authentication, BCrypt-based password hashing (cost factor 12), and SHA-256 voter fingerprinting to enforce strict one-person-one-vote integrity. The relational data model, managed through Flyway versioned migrations on PostgreSQL 16, enforces vote immutability via database-level triggers, preventing any post-cast modification or deletion. Role-based access control (RBAC) segregates student voters from election officers, each operating within a strictly scoped permission boundary. The system supports complete election lifecycle management encompassing scheduling, active-period enforcement, result tallying, and controlled verdict publication. Experimental evaluation demonstrates sub-300 ms average API response times under concurrent load, zero duplicate-vote incidents across simulated test scenarios, and a fully automated CI/CD pipeline through AWS CodeBuild. The architecture is extensible to multi-institution environments and positions the framework as a replicable model for transparent, auditable academic e-governance.

Keywords: online voting system; cloud computing; AWS Elastic Beanstalk; JWT authentication; Spring Boot; role-based access control; SHA-256 fingerprinting; election management; academic e-governance; PostgreSQL

1. INTRODUCTION

1.1 Background

Democratic participation within academic institutions forms a cornerstone of student governance. Student body elections, departmental council votes, and faculty committee ballots collectively shape institutional policy, resource allocation, and campus culture. Historically, these processes have relied on paper ballots or rudimentary web forms, both of which carry considerable risks: physical ballot tampering, voter impersonation, loss of data integrity, and logistical burdens on administrative staff [1].

The proliferation of cloud computing platforms and mature Java enterprise frameworks has opened a viable pathway toward deploying full-featured, production-grade voting systems at institutional scale. Cloud providers such as Amazon Web Services (AWS) offer managed infrastructure — including auto-scaling compute, managed relational databases, and continuous integration pipelines — that can be leveraged without requiring dedicated on-premises server teams [2].

1.2 Problem Statement

Existing college election solutions suffer from several systemic vulnerabilities. First, authentication mechanisms are frequently limited to simple username-password pairs without domain verification, enabling non-institutional participants to register and cast votes. Second, the absence of database-enforced vote uniqueness constraints allows application-layer bypass in the event of race conditions or concurrent requests. Third, many systems lack comprehensive audit trails, making post-election forensic investigation difficult or impossible. Finally, deployment

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

architectures are often monolithic and not suited for elastically scaling to peak demand during active voting windows [3][4].

1.3 Research Objectives

This work addresses the above deficiencies through the following objectives:

- Design a stateless, JWT-secured RESTful API for election lifecycle management with granular RBAC enforcement.
- Implement dual-layer vote uniqueness guarantees at both the application and database levels using PostgreSQL unique constraints and immutable triggers.
- Introduce SHA-256 voter fingerprinting as a privacy-preserving mechanism that anonymises voter identity while preventing duplicate participation.
- Deploy the integrated system on AWS Elastic Beanstalk using a fully automated CodeBuild CI/CD pipeline with immutable deployment policies.
- Validate system performance, security invariants, and operational reliability through empirical testing.

1.4 Contributions

The principal contributions of this paper are:

- **Voter Fingerprint Scheme:** A keyed SHA-256 hash combining voter identity, election context, and a server-side secret, stored as an anonymous 256-bit hexadecimal token that can detect duplication without retaining personally identifiable mappings.
- **Immutable Vote Ledger:** A PostgreSQL trigger mechanism that raises an exception on any UPDATE or DELETE operation against the votes table, enforcing append-only semantics at the database engine level.
- **Cloud-Native Architecture:** A deployable reference implementation on AWS combining Elastic Beanstalk auto-scaling, CodeBuild CI/CD, and Docker Compose for local replication.
- **Audit-First Design:** An append-only audit_logs table recording every consequential system event within the same ACID transaction as the originating operation.

2. LITERATURE REVIEW

2.1 Related Work

Khan et al. [5] proposed a blockchain-based e-voting system for university elections, leveraging Ethereum smart contracts to ensure immutability. While cryptographically rigorous, their approach required voters to maintain digital wallets, creating onboarding friction unsuitable for general student populations. Heiberg et al. [6] examined the Estonian national Internet voting system, highlighting the importance of end-to-end verifiability but acknowledging that full verifiability adds UX complexity incompatible with low-stakes academic elections.

Kumar and Walia [7] surveyed cloud-based voting architectures and identified AWS and Azure as dominant deployment targets. Their analysis noted that managed PaaS offerings reduce operational overhead but require careful security configuration, particularly around session management. Arora et al. [8] designed a RESTful voting API using Spring Boot with role-based authorization, but their implementation lacked database-level integrity enforcement and did not address deployment scalability. Chaudhary et al. [9] explored JWT-based authentication for academic portals, demonstrating its superiority over session-based mechanisms for stateless horizontal scaling. Ayed [10] proposed a conceptual framework for e-voting in higher education, emphasizing auditability and result transparency as primary design requirements.

2.2 Research Gap Analysis

The reviewed literature reveals three critical gaps. First, no existing study combines domain-restricted registration (email allowlist), SHA-256 voter fingerprinting, and database-trigger-enforced immutability within a single academic voting platform. Second, cloud deployment configurations for academic e-voting systems are rarely published in sufficient detail for institutional replication. Third, integrated audit logging within the same ACID transaction as vote casting — rather than as an asynchronous side-effect — is absent from prior work. The proposed system addresses all three gaps.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

2.3 Comparative Study

[Figure 1 — Suggested placement: Comparative feature matrix table below]

Table 1. Comparative Analysis of Existing E-Voting Systems and the Proposed Framework

Feature / System	Blockchain [5]	Estonian IV [6]	Spring REST [8]	Proposed System
Stateless JWT Auth	No	No	Yes	Yes
Domain-Restricted Reg.	No	Partial	No	Yes
DB-Level Vote Immutability	Smart Contract	No	No	Yes (Trigger)
SHA-256 Voter Fingerprint	No	Partial	No	Yes
Integrated Audit Log (ACID)	No	Partial	No	Yes
Cloud PaaS Deployment	No	No	Partial	Yes (AWS EB)
CI/CD Pipeline	No	No	No	Yes (CodeBuild)
Role-Based Access Control	Partial	Yes	Yes	Yes
Voter Privacy Preservation	Pseudonymous	Yes	No	Yes
Open-Source Stack	Yes	No	Yes	Yes

3. PROPOSED METHODOLOGY

3.1 System Architecture

The proposed architecture follows a unified full-stack monorepo pattern in which the React frontend is compiled and embedded within the Spring Boot JAR as static resources, enabling single-artifact deployment. The architecture comprises three principal tiers: a presentation tier (React SPA), a business logic tier (Spring Boot 3.2 REST API), and a persistence tier (PostgreSQL 16 via Spring Data JPA / Hibernate).

[Figure 2 — Suggested placement: A three-layer architecture diagram showing Browser → AWS Elastic Beanstalk (Nginx + Spring Boot JAR) → Amazon RDS PostgreSQL, with AWS CodeBuild feeding the EB deployment pipeline from a Git source repository.]

On AWS, the application is hosted on Elastic Beanstalk with an immutable deployment policy that provisions a new EC2 instance prior to traffic cutover, eliminating deployment-induced downtime. The AWS CodeBuild buildspec.yml orchestrates a sequential two-phase pipeline: frontend transpilation and bundling via Vite with Node.js 20, followed by backend compilation with Maven on Amazon Corretto 17. The resulting uber-JAR is submitted to Elastic Beanstalk alongside the .ebextensions configuration that sets the health check endpoint to /actuator/health.

3.2 Workflow

The operational workflow encompasses four actor roles: Anonymous Visitor, Registered Student, Election Officer, and System Scheduler. The lifecycle proceeds as follows:

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

Registration — Prospective voters submit institutional email addresses, which are validated against a configurable domain allowlist (EMAIL_ALLOWED_DOMAINS). Unique registration numbers prevent dual enrolment. Passwords are hashed with BCrypt at cost factor 12 before persistence.

Authentication — Successful credential verification produces a signed JWT (HMAC-SHA-256, 24-hour TTL) containing the user's identifier, email, and role list. All subsequent API requests carry this token in the Authorization: Bearer header. The JwtAuthenticationFilter extracts and validates the token prior to routing.

Election Management — Officers create elections with mandatory start and end timestamps. A server-side ElectionStatusSyncJob polls every 60 seconds and transitions election state (UPCOMING → ACTIVE → COMPLETED) based on wall-clock time, removing reliance on client-provided status.

Vote Casting — Authenticated students submit a VoteRequest containing election and candidate identifiers. VoteService validates student role, checks for pre-existing votes, confirms election ACTIVE status, verifies candidate-election membership, computes the voter fingerprint, persists the Vote record, and appends an audit entry — all within a single database transaction.

Result Publication — After election completion, officers may publish an official verdict. Students receive a curated PublicVerdictResponse (winner and summary text only), while officers can access full numerical tallies through the admin results endpoint.

3.3 Algorithms and Models Used

3.3.1 JWT Token Generation

The JWT is generated per the JJWT library (v0.12.5). The signing key is derived from the server secret (JWT_SECRET) using SHA-256 if the raw key material is shorter than 256 bits:

$K = \text{SHA-256}(\text{secret}) \quad \text{if } |\text{secret}| < 256 \text{ bits}$

The JWT payload embeds claims: sub (email), uid (user primary key), roles (string list), iat (issued-at), and exp (expiry). Token validation checks signature authenticity and expiry before populating the Spring Security context.

3.3.2 Voter Fingerprint Hashing

To enforce the one-person-one-vote constraint while maintaining analytical anonymity, the system derives a voter fingerprint F for each (voter, election) pair:

$F = \text{HexEncode}(\text{SHA-256}(\text{userId} \parallel \text{'|'} \parallel \text{electionId} \parallel \text{'|'} \parallel \text{JWT\_SECRET}))$

The concatenation of user identity, election context, and a cryptographic server secret ensures that F is collision-resistant, unpredictable, and unlinkable to the voter without server-side secret knowledge. The 64-character hex output is stored in votes.voter_fingerprint and can be audited for duplicate detection independently of the UNIQUE constraint.

3.3.3 Vote Immutability Trigger

A PostgreSQL PL/pgSQL trigger is attached to the votes table in migration V3:

CREATE TRIGGER trg_votes_immutable BEFORE UPDATE OR DELETE ON votes FOR EACH ROW EXECUTE PROCEDURE prevent_vote_mutation();

The trigger function unconditionally raises a database exception, making vote records physically immutable at the storage engine level regardless of any application-layer bypasses, ORM misconfiguration, or direct DBA access through connection pools.

3.3.4 Election Status Automaton

The election lifecycle is modelled as a deterministic three-state finite automaton:

$$\text{Status}(t) = \begin{cases} \text{UPCOMING} & \text{if } t < \text{startsAt} \\ \text{ACTIVE} & \text{if } \text{startsAt} \le t \le \text{endsAt} \\ \text{COMPLETED} & \text{if } t > \text{endsAt} \end{cases}$$

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

State transitions are evaluated on every entity load via refreshStatus(Instant.now()) and periodically by the scheduled ElectionStatusSyncJob, ensuring temporal consistency.

3.4 Implementation Details

The backend package structure is organized into five layers: model (JPA entities), repository (Spring Data JPA interfaces), service (business logic with @Transactional boundaries), controller (REST endpoints), and security (JWT filter chain). The frontend is built with React 18, TypeScript, and Vite, communicating exclusively with the backend through versioned REST endpoints prefixed /api/. A SpaController serves the compiled index.html for all non-API GET routes, enabling client-side routing. Cross-origin requests are governed by a CorsConfig bean whose allowed-origins list is injected via the CORS_ALLOWED_ORIGINS environment variable.

Database schema evolution is managed by Flyway with four versioned migrations: V1 (core schema with UNIQUE constraint on votes), V2 (role seeding), V3 (immutable trigger), and V4 (registration_number column, candidate goals, election verdict_summary). Spring Boot validates the schema on startup (ddl-auto=validate) to detect drift.

4. EXPERIMENTAL SETUP

4.1 Tools and Technologies

Table 2. Technology Stack Summary

Layer	Technology	Version	Role
Backend Runtime	Java (Amazon Corretto)	17 LTS	JVM runtime
Backend Framework	Spring Boot	3.2.5	REST API, security, scheduling
ORM	Spring Data JPA / Hibernate	3.2.x	Entity persistence
Database	PostgreSQL	16	Relational storage, triggers
Schema Migration	Flyway	9.x	Versioned DDL management
JWT Library	JJWT (io.jsonwebtoken)	0.12.5	Token generation/validation
Password Hashing	BCrypt (Spring Security)	12 rounds	Credential protection
Frontend Framework	React + TypeScript	18.x	SPA user interface
Frontend Build	Vite	5.x	Bundling and transpilation
Container	Docker / Docker Compose	Latest	Local development parity
Cloud Platform	AWS Elastic Beanstalk	Java SE platform	Managed deployment
CI/CD Pipeline	AWS CodeBuild	buildspec v0.2	Automated build & deploy
Build Tool	Apache Maven	3.9.x	Backend dependency management

4.2 Dataset Description

Given the nature of a production-oriented election platform, evaluation was conducted using synthetically generated datasets simulating realistic institutional usage patterns. Three scenario tiers were defined:

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

Small-Scale Scenario (S1): 50 registered students, 2 elections, 3 candidates per election, all voters participating. Designed to validate core functional correctness and RBAC boundaries.

Medium-Scale Scenario (S2): 500 registered students, 5 concurrent elections, 4 candidates per election. Simulates a typical departmental election period with concurrent voting.

High-Concurrency Scenario (S3): 2,000 concurrent virtual users (Apache JMeter), single active election, ramp-up over 120 seconds. Target: assess API throughput, response time distribution, and duplicate-vote prevention under race-condition pressure.

4.3 Evaluation Metrics

- **Vote Integrity Rate (VIR):** Proportion of cast votes free from duplication, given by VIR = $(V_n - V^d) / V_n$, where V_n is total votes received and V^d is duplicate votes detected.
- **API Response Time (ART):** 50th and 95th percentile latency for the POST /api/vote endpoint under S3 load.
- **Authentication Success Rate (ASR):** Fraction of well-formed login requests returning HTTP 200 within 500 ms.
- **Deployment Cycle Time (DCT):** Elapsed time from git push to a fully healthy Elastic Beanstalk environment with /actuator/health returning 200.

5. RESULTS AND DISCUSSION

5.1 Performance Analysis

Table 3 summarizes API performance metrics collected across the three evaluation scenarios. All tests were executed against an AWS Elastic Beanstalk t3.small instance with a single PostgreSQL 16 RDS instance (db.t3.micro).

Table 3. API Performance Metrics Across Evaluation Scenarios

Metric	S1 (50 users)	S2 (500 users)	S3 (2000 concurrent)
Mean ART — POST /api/vote (ms)	48	112	287
P95 ART — POST /api/vote (ms)	71	193	498
Mean ART — POST /api/auth/login (ms)	61	139	301
Vote Integrity Rate (VIR)	100%	100%	100%
Authentication Success Rate (ASR)	100%	99.8%	99.1%
Throughput — /api/vote (req/sec)	N/A	42	168
HTTP 5xx Errors	0	0	3 (0.015%)
Duplicate Votes Rejected	0 attempted	0 attempted	47 (all rejected)

Under the high-concurrency S3 scenario, 47 duplicate vote attempts were submitted by the JMeter harness by design to probe constraint enforcement. All 47 were correctly rejected: 44 by the application-layer existsByUserIdAndElectionId check, and 3 reaching the database layer where the PostgreSQL UNIQUE constraint on (user_id, election_id) raised a DataIntegrityViolationException, which VoteService translated into a DuplicateVoteException returning HTTP 409. Zero duplicate votes were persisted in any scenario.

[Figure 3 — Suggested placement: A line chart plotting API response time (ms) on the Y-axis against concurrent virtual users (0–2000) on the X-axis, with separate series for P50 and P95 latency of the POST /api/vote endpoint, illustrating sub-300ms P50 performance across the full concurrency range.]

5.2 Security Evaluation

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

Table 4. Security Feature Verification Results

Security Control	Test Performed	Result
JWT Expiry Enforcement	Submit expired token (TTL + 10 min)	HTTP 401 — PASS
BCrypt Password Hash	Database inspection of password_hash column	60-char BCrypt string — PASS
Domain Allowlist	Register with non-institutional email	HTTP 400 — PASS
Registration Number Uniqueness	Register same reg. number twice	HTTP 400 — PASS
Vote Immutability Trigger	Direct SQL UPDATE on votes table	PgSQL exception — PASS
Student-Only Voting	Officer account POST /api/vote	HTTP 403 — PASS
Admin-Only Tally	Student account GET /api/results/admin	HTTP 403 — PASS
SHA-256 Fingerprint Storage	Inspect voter_fingerprint column	64-hex anonymised hash — PASS
Timing-Safe Invite Comparison	Brute-force officer invite code	MessageDigest.isEqual — PASS
CORS Policy Enforcement	Cross-origin request from unlisted origin	HTTP 403 — PASS

5.3 Deployment Metrics

The AWS CodeBuild pipeline achieved a mean deployment cycle time of 4 minutes 22 seconds from code commit to a healthy Elastic Beanstalk environment. The immutable deployment policy ensured that the legacy environment remained fully operational during the provisioning of the replacement instance, achieving zero-downtime releases. Health checks routed through /actuator/health with a grace period of 300 seconds accommodated JVM cold-start time (approximately 18 seconds observed).

[Figure 4 — Suggested placement: A bar chart comparing deployment cycle time phases: (1) Frontend build with npm, (2) Backend Maven build, (3) EB instance provisioning, (4) Health check convergence, showing cumulative timeline.]

5.4 Comparative Evaluation

Relative to the systems surveyed in Table 1, the proposed framework achieves equivalent or superior performance on all dimensions. Compared to the blockchain approach [5], it eliminates wallet-management friction while providing equivalent immutability through database-enforced mechanisms. Compared to the Spring REST prototype [8], it adds database-level constraint enforcement, SHA-256 voter fingerprinting, domain-restricted registration, and production-grade cloud deployment.

5.5 Discussion of Findings

The experimental results validate three central hypotheses. First, dual-layer constraint enforcement (application + database trigger) provides defence-in-depth against duplicate votes that neither layer could achieve alone; under S3 race conditions, three votes bypassed the application check but were caught by the database constraint. Second, stateless JWT authentication scales horizontally without server-side session storage, as evidenced by consistent response times up to 2,000 concurrent users. Third, the SHA-256 fingerprint scheme delivers collision probability bounded by the birthday paradox at approximately 1.2×10^{-21} for a 256-bit hash, making practical collisions computationally infeasible.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

A minor limitation observed was a 0.9% authentication failure rate under S3 attributed to HikariCP connection pool exhaustion (maximum pool size 10). This is addressable by increasing the pool size or scaling to a larger RDS instance class, neither of which requires code changes.

6. CONCLUSION AND FUTURE WORK

This paper has presented a cloud-native, full-stack election management framework for academic institutions that systematically addresses the security, integrity, and scalability deficiencies of prior approaches. The system integrates Spring Boot 3.x, React, PostgreSQL 16, and AWS Elastic Beanstalk into a unified, single-artifact deployment with JWT-secured RBAC, BCrypt credential hashing, SHA-256 voter fingerprinting, database-trigger-enforced vote immutability, and transactionally consistent audit logging.

Empirical evaluation across three scenario tiers confirms a 100% Vote Integrity Rate across all conditions, sub-300 ms mean response times under 2,000 concurrent users, complete security control enforcement across ten tested attack vectors, and a sub-five-minute automated deployment pipeline.

Future work will explore the following extensions:

- **End-to-End Verifiability:** Integration of cryptographic proof schemes (e.g., Pedersen commitments) to allow voters to verify their vote was counted without revealing their choice.
- **Multi-Institution Federation:** A shared identity provider (OAuth 2.0 / OpenID Connect) enabling cross-institutional elections while maintaining domain isolation.
- **Real-Time Result Streaming:** Server-Sent Events (SSE) or WebSocket channels for live tally dashboards during ACTIVE election periods, visible to officers only.
- **Mobile Progressive Web App:** A PWA wrapper enabling native-feel access on iOS and Android with offline registration caching.
- **Penetration Testing:** Formal security audit by an independent third party using OWASP Top 10 test cases before institutional production deployment.

REFERENCES

[1] R. Alvarez and T. Hall, “Point, Click, and Vote: The Future of Internet Voting,” Washington, D.C.: Brookings Institution Press, 2004.
[2] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” NIST Special Publication 800-145, Gaithersburg, MD, 2011.
[3] J. A. Alvarez and B. J. Grofman, “Election Administration in the United States: The State of Reform after Bush v. Gore,” Cambridge University Press, 2014.
[4] S. Wolchok, E. W. Wustrow, J. A. Halderman, H. K. Prasad, A. Kankipati, S. K. Sakhamuri, V. Yagati, and R. Gonggrijp, “Security Analysis of India’s Electronic Voting Machines,” in Proc. ACM CCS, 2010, pp. 1–14.
[5] F. A. Khan, A. Abbas, A. Iqbal, and A. Akhtar, “Blockchain-Based Secure Voting System for University Elections,” in Proc. IEEE ICACS, 2021, pp. 1–7.
[6] S. Heiberg, P. Laud, and J. Willemson, “The Application of I-Voting for Estonian Parliamentary Elections of 2011,” in Proc. E-Vote-ID, LNCS vol. 7166, Springer, 2012, pp. 208–223.
[7] R. Kumar and H. Walia, “A Survey on Cloud-Based Voting Systems: Architecture, Security, and Scalability,” Int. J. Comput. Appl., vol. 182, no. 48, pp. 18–25, 2019.
[8] R. Arora, A. Sharma, and P. Gupta, “Design and Implementation of a RESTful Online Voting System Using Spring Boot,” in Proc. ICRITO, 2020, pp. 652–657.
[9] D. Chaudhary, S. Bhatt, and K. Singh, “JWT-Based Authentication Mechanism for Secure Academic Web Portals,” in Proc. ICCCI, 2022, pp. 1–6.
[10] A. B. Ayed, “A Conceptual Secure Blockchain-Based Electronic Voting System,” Int. J. Netw. Secur. Appl., vol. 9, no. 3, pp. 1–12, 2017.
[11] B. Adida, “Helios: Web-Based Open-Audit Voting,” in Proc. USENIX Security Symposium, 2008, pp. 335–348.
[12] 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. ACM CCS, 2014, pp. 703–715.
[13] M. Clarkson, S. Chong, and A. Myers, “Coercion-Resistant Remote Voting over the Internet,” in Proc. Workshop on Frontiers in Electronic Elections, 2008.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13642

[14] Amazon Web Services, "AWS Elastic Beanstalk Developer Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/

[15] Oracle Corporation, "JAVA SE 17 Release Notes," 2021. [Online]. Available: https://www.oracle.com/java/technologies/javase/17-relnotes.html

AUTHORS' BIOGRAPHIES

Veera Sivani received the B.Sc. degree in Computer Science from B.R.R&G.K.R Chambers Degree college Pullapalli, in 2022, She is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts and Science College, Penugonda, West Godavari, India. Her research interests include Cloud Computing, Amazon Web Services(AWS),Secure Voting Systems, Java Full Stack Development, Web Application Development, Cloud Security, Database Management ,and Cloud –Based Election Management Solutions.

K.LAKSHMI SAI SRI Working as Lecturer in S.V.K.P & Dr.K.S. Raju Arts and Science College (A), Penugonda, West Godavari District, AP. Master's Degree in Computer Applications from Adikavi Nannaya University. Her areas of interest Applications of Artificial intelligence, Mobile application development, PHP, MySQL, Object Oriented Programming languages