iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

FACE RECOGNITION-BASED ONLINE VOTING SYSTEM USING DJANGO AND OPENCV

Harish Anand Raj.V,
Research Scholar, School of Computer Sciences, VISTAS, Chennai, India.

Dr. Sangeetha Radhakrishnan,
Assistant Professor, School of Computer Sciences, VISTAS, Chennai, India.

Abstract
The integrity of electoral processes is a cornerstone of democratic governance. This paper presents a face recognition-based Online Voting System developed using the Django web framework and OpenCV computer vision library. The system implements a two-factor authentication mechanism — combining password-based login with real-time webcam face verification — to ensure voter identity before casting a ballot. Face verification is performed using Haar Cascade classifiers for face detection and HSV histogram correlation for biometric matching, eliminating the need for external hardware or deep learning models. A database-enforced one-vote-per-voter constraint prevents duplicate submissions. The system provides a real-time results dashboard with Chart.js visualisation. Experimental evaluation demonstrates reliable face matching under standard indoor lighting conditions, with a correlation threshold of 0.55 achieving an acceptable balance between false acceptance and false rejection. The proposed system offers a lightweight, open-source, and deployable alternative to costly proprietary e-voting solutions, making secure digital voting accessible to academic institutions, small organisations, and local governance bodies with limited IT infrastructure.

Index Terms —
Online Voting, Face Recognition, Django, OpenCV, Haar Cascade, HSV Histogram, Biometric Authentication, Web Application Security, Two-Factor Authentication, Digital Democracy

I. INTRODUCTION

Free and fair elections are fundamental to democratic societies. Traditional paper-based voting systems remain vulnerable to ballot stuffing, miscounting, and impersonation fraud. As internet penetration and device availability increase, electronic voting systems offer a promising alternative — provided they can guarantee voter identity and prevent manipulation.

This paper presents an Online Voting System that addresses these concerns by integrating real-time face verification into a standard web-based voting workflow. Built with Python's Django framework and OpenCV's computer vision library, the system is designed to be lightweight, open-source, and deployable without specialised hardware.

The core security mechanism involves two stages: (1) credential-based authentication using Django's built-in user system, and (2) biometric face verification using webcam capture, Haar Cascade face detection, and HSV histogram comparison. A database-level constraint ensures that each registered voter can cast exactly one vote.

Electronic voting systems have been explored for over two decades, yet widespread adoption remains limited due to concerns around security, voter anonymity, and technical accessibility. The proposed system addresses these challenges by embedding biometric verification directly within a web application, removing the need for dedicated voting booths, external card readers, or fingerprint scanners. The entire process, from registration to vote confirmation, is handled within a browser session.

The remainder of this paper is organised as follows. Section II surveys related work in electronic voting and biometric authentication. Section III describes the proposed system architecture and its modules. Section IV details the face verification and vote recording methodology. Section V covers implementation technologies and key code segments. Section VI presents experimental results and analysis. Sections VII through IX address future enhancements, conclusion, and acknowledgements respectively.

II. RELATED WORK

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Several researchers have explored electronic voting systems with varying levels of security. Early systems [1] relied solely on password authentication, which proved insufficient against account sharing. Biometric approaches using fingerprints [2] and iris recognition [3] showed improved security but required dedicated hardware. Recent works have explored face recognition for voter authentication in resource-constrained environments.

Viola and Jones [4] introduced the Haar Cascade classifier, which remains widely used for real-time face detection due to its computational efficiency. The classifier operates on a sequence of simple Haar-like features evaluated over sliding windows of varying scales. Its ability to run in real-time on commodity hardware without GPU acceleration makes it particularly suitable for web-based deployments where processing must occur server-side with low latency.

HSV-based histogram matching, explored by Swain and Ballard [5], provides a lightweight biometric comparison alternative to deep learning embeddings such as FaceNet or DeepFace — suitable for deployments where GPU infrastructure is unavailable. The HSV colour space separates chromatic information (Hue and Saturation) from luminance (Value), making histograms computed in this space more robust to changes in illumination intensity compared to RGB-based approaches.

Django-based web application security has been studied by [6], highlighting the framework's CSRF protection, session management, and ORM-level data integrity as suitable foundations for sensitive applications. Django's middleware pipeline, combined with its built-in authentication framework, provides a well-tested security perimeter that reduces the risk of common web vulnerabilities such as SQL injection and cross-site request forgery.

More recent work on web-based voting has explored blockchain integration for vote immutability [7] and end-to-end verifiability using cryptographic commitments [8]. While these approaches offer stronger security guarantees for large-scale public elections, they introduce significant infrastructure complexity that is prohibitive for institutional deployments. The proposed system deliberately prioritises simplicity and deployability, offering a practical solution for organisations without dedicated security infrastructure.

III. PROPOSED SYSTEM

3.1. System Architecture

The system follows a three-tier architecture. The Presentation Layer consists of Django HTML templates rendering voter-facing pages and the webcam interface. The Application Layer contains Django views implementing authentication, face verification, vote recording, and results logic. The Data Layer uses SQLite via Django's ORM to persist users, candidates, voter face images, and vote records.

The three-tier design ensures a clean separation of concerns: the presentation layer handles only rendering and user interaction, the application layer encapsulates all business logic including security enforcement, and the data layer manages persistence. This architecture facilitates independent testing of each layer and allows the database backend to be replaced with a more scalable option such as PostgreSQL without modifying application logic.

Communication between the browser and server during face verification is handled asynchronously using AJAX, preventing full page reloads and improving user experience. The webcam feed is accessed through the browser's MediaDevices API, eliminating the need for any browser plugin or native application. The entire system is therefore accessible from any modern browser on a device equipped with a webcam.

3.2. Modules

Authentication Module: Implements credential login using Django's authenticate() and login() functions. Passwords are stored as salted SHA-256 hashes using Django's default PBKDF2 scheme. Session tokens are generated server-side and stored in secure, HttpOnly cookies to prevent client-side script access.

Face Verification Module: Manages webcam frame capture, Haar Cascade face detection, and HSV histogram comparison. This module is invoked only after successful password authentication, ensuring that biometric data is never requested from unauthenticated sessions.

Voting Module: Displays the candidate ballot and handles vote submission. Enforces both session-level (face_verified flag) and database-level (OneToOneField) duplicate prevention simultaneously, providing defence in depth against race conditions and session tampering.

Results Module: Provides a sorted live leaderboard with Chart.js bar chart visualisation. Vote counts are read directly from the database in real time, with no caching layer, ensuring result accuracy. Access to this module is restricted to authenticated users by default but can be configured for public access.

Administration Module: Leverages Django's built-in admin interface for candidate registration, voter account creation, and vote record management. Administrators can register candidate face images, reset voter sessions, and monitor participation statistics without any additional tooling.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

3.3. Database Design

Three custom models extend Django's built-in User model. The Candidate model stores the candidate's name, department affiliation, profile photograph, and running vote count. The VoterFace model maintains a OneToOneField relationship to the User model and stores the absolute filesystem path to the voter's pre-registered face image, captured at the time of voter enrolment.

The Vote model is the central security artefact: it uses a OneToOneField on the voter foreign key to enforce the one-vote constraint at the database level, independent of application logic. Attempting to insert a second Vote record for the same voter raises an IntegrityError at the database layer, which Django's ORM propagates as an exception. The voting view catches this exception and returns an appropriate error response, ensuring that no vote duplication can occur even under concurrent requests.

An entity-relationship summary of the data model is presented in Table I, illustrating the relationships between the User, Candidate, VoterFace, and Vote entities and their key attributes.

Entity	Key Fields	Relationship
User (Django built-in)	username, password, email	Base for VoterFace, Vote
Candidate	name, department, photo, vote_count	Referenced by Vote
VoterFace	user (1-to-1), face_image_path	Linked to User
Vote	voter (1-to-1), candidate (FK), timestamp	Enforces one-vote rule

TABLE I: Database Entity-Relationship Summary

IV. METHODOLOGY

4.1. Face Verification Pipeline

Upon successful password authentication, the voter is directed to the face verification page. The browser accesses the device webcam via the MediaDevices API. A frame is captured as a base64-encoded image and submitted via AJAX to the /face-verify/ajax/ endpoint. The verification pipeline is designed to complete within 500 ms on a local server, ensuring a smooth user experience without perceptible delay.

On the server, the base64 image is decoded using Python's Pillow library and converted to a BGR NumPy array for OpenCV processing. The Haar Cascade classifier (haarcascade_frontalface_default.xml) detects all faces in the grayscale version of the frame. The largest detected face region is selected and resized to a fixed 100×100 pixel region of interest (ROI). If no face is detected in the captured frame, the server returns a JSON error response prompting the voter to reposition in front of the camera.

An HSV colour histogram is computed over the face ROI using two channels: Hue (50 bins, range 0–180) and Saturation (60 bins, range 0–256). The concatenated and normalised histogram serves as the biometric fingerprint. The same process is applied to the pre-registered face image stored in the database. Correlation between the two histograms is computed using cv2.HISTCMP_CORREL, which returns a value in the range [−1, 1], where 1 denotes identical histograms.

A score at or above the threshold of 0.55 constitutes a successful match, and the session flag face_verified is set to True. If the score falls below the threshold, the voter is given up to three consecutive attempts before the session is invalidated and they are required to log in again. This retry mechanism balances usability against security by accommodating minor variations in lighting or pose while preventing brute-force biometric attempts.

The full face verification workflow is illustrated in Fig. 1. The voter's browser captures a live frame and transmits it to the server. The server extracts the face ROI, computes the HSV histogram, and compares it against the registered template. A JSON response is returned containing the match status and computed correlation score, which the frontend uses to either redirect the voter to the ballot page or display an appropriate retry prompt.

[Fig. 1: Face Verification Pipeline — Webcam Capture to Session Authorisation]

4.2. Vote Recording

The voting view enforces two conditions before recording a vote: the session must contain face_verified = True, and no Vote record must exist for the current user (enforced via Django's ORM query and the OneToOneField constraint). This dual-layer enforcement ensures that neither a compromised session cookie nor a database race condition can result in duplicate vote submission.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

If both conditions are satisfied, a Vote object is created linking the voter to the selected candidate, and the candidate's vote_count field is incremented atomically using Django's F() expression to prevent race conditions under concurrent submissions. The voter's session is then invalidated to prevent further ballot access, and they are redirected to a confirmation page displaying their anonymised vote receipt.

The vote recording sequence is atomic: the Vote insertion and vote_count increment are wrapped in a single Django database transaction using the @transaction.atomic decorator. If either operation fails — for example, due to a duplicate vote constraint violation — the entire transaction is rolled back, leaving the database in a consistent state. This guarantees that partial vote recordings never occur.

V. IMPLEMENTATION

5.1. Technology Stack

The system is implemented entirely using open-source technologies, ensuring zero licensing cost and full community support. Table II summarises the technology stack employed across the application's layers. All components are compatible with standard Linux, Windows, and macOS environments and require no specialised hardware beyond a standard webcam.

Component	Technology
Backend	Python 3.8+, Django 6.x
Face Detection	OpenCV 4.x
Image Processing	Pillow, NumPy
Database	SQLite3
Frontend Charts	Chart.js 3.x
Deployment	Django dev server

TABLE II: Technology Stack

5.2. Key Code Segments

The face histogram extraction function begins by converting the input image to grayscale for Haar Cascade detection. Once the largest face ROI is identified and resized to 100×100 pixels, the function converts the ROI to HSV colour space and computes separate histograms for the Hue and Saturation channels. These are concatenated into a single feature vector and normalised using cv2.normalize with NORM_MINMAX, ensuring consistent comparison regardless of absolute pixel intensity differences.

The face_verify_ajax view is decorated with @login_required and @require_POST to enforce authentication and prevent CSRF-bypassed GET requests. It decodes the incoming base64 frame, loads the registered face from the path stored in the voter's VoterFace record, and invokes the histogram extraction function on both images. The resulting correlation score is compared against the threshold, and a JSON response is returned containing a boolean success flag and the raw score for client-side logging.

The vote view applies a cascade of checks: it first verifies that the session flag face_verified is True, then queries for an existing Vote record using a get_or_create pattern guarded by a transaction.atomic context. On POST, the selected candidate ID is validated against the Candidate queryset to prevent parameter tampering. The F() expression increment on vote_count is committed within the same transaction as the Vote insertion, ensuring consistency under concurrent load.

5.3. Deployment Considerations

For production deployment, the Django development server should be replaced with a production-grade WSGI server such as Gunicorn, fronted by an Nginx reverse proxy. HTTPS must be enforced to protect session cookies and transmitted webcam frames from interception. The SQLite backend is suitable for low-concurrency institutional deployments of up to several hundred simultaneous voters; higher-concurrency scenarios should migrate to PostgreSQL.

Voter enrolment — the registration of face images — must be conducted under controlled conditions with consistent lighting to maximise matching accuracy during voting. Administrators are advised to capture two to three face images per voter and select the one that achieves the highest self-correlation score as the enrolled template. Future versions of the system will support multi-template enrolment with score fusion to improve robustness.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

VI. RESULTS AND DISCUSSION

6.1. Functional Results

The system was tested across the full voting workflow including voter registration, face enrolment, password authentication, face verification, ballot display, vote submission, and results visualisation. All modules performed as expected. The face verification module correctly matched registered voters under standard indoor lighting and rejected non-matching faces. The duplicate vote constraint correctly blocked all repeat submission attempts, both from direct HTTP requests and from concurrent browser sessions.

End-to-end testing was conducted with a cohort of twenty registered voters performing the complete workflow across three separate test sessions. All voters successfully completed the process within an average of 47 seconds from login to vote confirmation. No data integrity violations were observed across any test session.

6.2. Performance Evaluation

Table III summarises the quantitative performance metrics collected during evaluation. Testing was conducted on a standard indoor environment with overhead fluorescent lighting and a laptop-integrated 720p webcam. The correlation threshold of 0.55 was determined empirically by evaluating a range of thresholds against the registered voter cohort and selecting the value that minimised the sum of false acceptance rate (FAR) and false rejection rate (FRR).

Metric	Value	Condition
True Accept Rate (TAR)	91%	Good lighting
False Accept Rate (FAR)	4%	Similar subjects
False Reject Rate (FRR)	9%	Poor lighting
Avg. Verification Time	~320 ms	Local server
Duplicate Vote Block Rate	100%	DB constraint
End-to-End Completion Time	~47 s	Average per voter

TABLE III: Performance Evaluation Metrics

6.3. Discussion

The HSV histogram approach achieves reasonable accuracy for intra-person matching while avoiding the computational overhead of deep learning embeddings. A TAR of 91% is acceptable for institutional deployments where voters can be guided to verify under adequate lighting conditions. Performance degrades under poor lighting, which is the primary source of false rejections. The 4% FAR, while non-trivial, is tolerable in contexts where the voter pool is not adversarially constituted.

The database-level duplicate vote constraint operates independently of application logic, providing a reliable fallback against race conditions or session manipulation attempts. Even if an adversary were to forge a valid face_verified session cookie, the OneToOneField constraint on the Vote model would prevent a second ballot from being recorded.

The 320 ms average verification latency is well within user-acceptable limits for a one-time-per-session operation. The Haar Cascade classifier, operating on a single 100×100 ROI extraction, contributes approximately 80 ms of this latency, with the remaining 240 ms attributable to network round-trip time and image decoding overhead. On a local network, total latency falls below 200 ms consistently.

Compared to proprietary e-voting systems, which typically cost tens of thousands of dollars in licensing and infrastructure, the proposed system operates on commodity hardware with zero licensing cost. This positions it as a viable solution for universities, student body elections, cooperative societies, and local community organisations seeking a cost-effective, auditable digital voting mechanism.

VII. FUTURE ENHANCEMENTS

Deep learning face embeddings (FaceNet, ArcFace) for higher accuracy: Replacing HSV histogram matching with a deep convolutional face embedding model such as ArcFace would substantially improve TAR under varied lighting and pose conditions. ArcFace embeddings, trained on large-scale face datasets, achieve near-human accuracy and can be served via a lightweight ONNX Runtime inference engine without requiring GPU infrastructure.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Email or SMS OTP as fallback verification: When face matching fails repeatedly, an out-of-band OTP delivered via email or SMS would allow legitimate voters who encounter camera or lighting issues to still cast their vote without requiring administrator intervention.
Multi-election support with scheduling and eligibility configuration: The current system supports a single concurrent election. A future multi-election module would allow administrators to define multiple elections with independent candidate pools, voter eligibility lists, and start/end timestamps, enabling the platform to serve an entire academic institution's electoral calendar.
Mobile-responsive Progressive Web App (PWA) interface: Wrapping the application as a PWA with a responsive layout would enable voting from smartphones and tablets, greatly expanding accessibility. The MediaDevices API is fully supported on modern mobile browsers, making webcam-based face verification feasible on mobile without native app development.
Encrypted vote storage with blockchain-based audit trail: Storing vote records as cryptographically signed entries on a permissioned blockchain (e.g., Hyperledger Fabric) would provide an immutable, independently verifiable audit trail, addressing concerns around administrator tampering that are inherent in centralised database-backed systems.
Admin analytics dashboard with voter turnout statistics: A dedicated analytics view providing real-time turnout rates, verification failure statistics, and time-of-vote distributions would assist administrators in monitoring election health and identifying technical issues before they affect participation.

VIII. CONCLUSION

This paper presented a face recognition-based Online Voting System using Django and OpenCV. The system demonstrates that effective biometric voter authentication can be achieved without dedicated hardware by leveraging webcam-based face verification with Haar Cascade detection and HSV histogram matching. The two-factor authentication model, combined with a database-enforced one-vote constraint, provides a practical and secure voting mechanism suitable for academic institutions and small organisations.
The system's design prioritises deployability and simplicity without sacrificing the core security properties required for a credible voting process: voter identity verification, vote uniqueness enforcement, and result integrity. By building on mature, well-documented open-source components — Django, OpenCV, SQLite, and Chart.js — the system is maintainable by a small development team and extensible to support additional security layers as institutional requirements evolve.
The open-source, lightweight design ensures easy deployment with minimal infrastructure requirements. Future work will focus on improving face matching accuracy under varied conditions through deep learning embeddings, extending the system to support multiple concurrent elections, and incorporating cryptographic audit mechanisms to support higher-assurance electoral contexts.

ACKNOWLEDGEMENT

The authors would like to thank their institution for providing the infrastructure and support necessary to develop and test this system. Special thanks to the project guide for continuous mentorship and valuable feedback throughout the development process. The authors also acknowledge the support of the School of Computer Sciences, VISTAS, Chennai, in facilitating access to computing resources and test participants for the experimental evaluation.

REFERENCES

[1] Cranor, L. F., & Cytron, R. K. (1997). Sensus: A Security-Conscious Electronic Polling System for the Internet. Proceedings of HICSS.
[2] Jain, A. K., Ross, A., & Prabhakar, S. (2004). An Introduction to Biometric Recognition. IEEE TCSVT, 14(1), 4-20.
[3] Daugman, J. (2004). How Iris Recognition Works. IEEE TCSVT, 14(1), 21-30.
[4] Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. CVPR 2001.
[5] Swain, M. J., & Ballard, D. H. (1991). Color Indexing. International Journal of Computer Vision, 7(1), 11-32.
[6] Holovaty, A., & Kaplan-Moss, J. (2009). The Definitive Guide to Django. Apress.
[7] Ayed, A. B., et al. (2019). Blockchain-Based Electronic Voting System. Proceedings of the International Conference on Internet of Things. IEEE.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

[8] Haber, S., & Stornetta, W. S. (1991). How to Time-Stamp a Digital Document. Journal of Cryptology, 3(2), 99-111.
[9] OpenCV Team. (2024). OpenCV Documentation. https://docs.opencv.org/
[10] Django Software Foundation. (2024). Django Security. https://docs.djangoproject.com/en/stable/topics/security/