



International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Design and Implementation of a Web-Based Voting System Using E-mail OTP Verification and Webcam Face Capture A *Prototype for a Prime Ministerial Election Demonstration*

Assistant Professor Smita Mangesh Junnarkar¹, Yadav Hari Haran Vellaisamy²

¹Department of Data Science, SIES College of Arts, Science and Commerce (Empowered Autonomous), Sion(W), Mumbai. smitaj@sies.edu.in

²Department of Data Science, SIES College of Arts, Science and Commerce (Empowered Autonomous), Sion(W), Mumbai. yadavsushila37194@gmail.com

ABSTRACT

Casting a vote from a phone or laptop is convenient, but that convenience is worthless if the system cannot tell who is voting or whether they have already voted. I built a small web-based voting system in Python using Flask, as a demonstration election where voters pick a Prime Minister from three candidates. The idea was simple: a voter signs up with their name and email, gets a six-digit OTP sent to that email, proves they actually own the inbox, takes a photo through their webcam, and only then can they cast one vote. The OTP is stored only as a keyed hash, expires after five minutes and locks after three wrong attempts. The vote is recorded through one atomic database operation so that a voter cannot be counted twice, and the table of votes holds no reference to the voter. The same code runs on SQLite during development and on PostgreSQL when deployed to a serverless platform. We checked the system with fourteen automated functional tests, all of which passed, and we are explicit about what those tests do not cover. The analysis is equally frank about the design itself: e-mail is a weak channel for OTPs, a photograph is evidence rather than identity verification, and the secrecy of the ballot depends on database design instead of cryptography. The system suits classroom, club or organisational elections and works as a teaching prototype. It is not suitable for statutory elections.

Keywords: online voting, one-time password, face capture, Flask, ballot secrecy, web security, serverless deployment

1. Introduction

1.1 Background and motivation

Indian general elections are conducted by the Election Commission of India (ECI) using electronic voting machines placed in polling stations, and the scale of that exercise, with hundreds of millions of eligible voters, is hard to appreciate until one thinks about the logistics. Away from those national contests, a far larger number of small elections take place every year: student councils, housing societies, professional associations, clubs and company committees. Most of them are settled by a show of hands, paper slips or a shared online form. Each of these has an obvious weakness. Raised hands are public, paper needs a room and someone to count it, and an ordinary online form will accept ten submissions from the same person without complaint.

This project began with a simple question: how much of the trust that a polling booth provides can be rebuilt in a browser using only free tools? We do not claim an answer that would satisfy an election authority. We do think the exercise is worth writing down, because even a modest system forces a long list of decisions about identity, secrecy, duplication and failure that are easy to overlook when one only reads about electronic voting.

A note on the demonstration ballot: the three candidates and their party symbols were used as realistic placeholders so that the interface could be tested with a familiar ballot layout. The system is not connected with, endorsed by or affiliated to the Election Commission of India or any political party, and it collects no real votes.

1.2 Problem statement

Really, the project had five goals. Only someone who actually controls a registered email should be able to vote. Nobody should be able to vote twice. The system should keep a person's identity separate from what they actually voted for. There should still be some record of who participated. And, importantly, it had to be something I could realistically run and deploy myself, on a laptop or a free hosting tier, without needing a team or expensive infrastructure.

1.3 Objectives

- Build the full voter journey, registration, OTP check, photo capture, voting and confirmation, so that someone using it for the first time doesn't need anyone to explain how it works.
- 1. To make duplicate voting impossible at the database level and not merely hidden in the interface.
- 2. To keep the tally and the voter list structurally separate.
- 3. To give an administrator live results and a searchable log of who voted.
- 4. To deploy the system on a public serverless platform and record what had to change to make that work.

1.4 Contributions

The paper offers three things. First, a small reference implementation, a few hundred lines of Python plus templates, that is easy to read end to end. Second, a data layer written so the same SQL runs on SQLite and PostgreSQL, which lowers the barrier between a classroom laptop and a hosted deployment. Third, a candid security analysis, including a worked estimate of OTP guessing risk and a list of weaknesses that remain, together with practical lessons from deployment. The rest of the paper is organised as follows. Section 2 reviews related work, Section 3 sets out requirements and threats, Sections 4 and 5 describe design and implementation, Section 6 reports deployment experience, Section 7 presents testing, and Sections 8 to 11 discuss security, limitations, future work and conclusions.

2. Related Work

2.1 Security research on electronic voting

The history of electronic voting is, to a large extent, a history of flaws found after deployment. Kohno and colleagues analysed the source code of a touch-screen voting machine that was already in use and reported serious weaknesses in its design [1]. The lesson many readers took from that study was not that computers cannot count votes, but that a system trusted by default and closed to inspection is difficult to trust at all. Internet voting has been tried at national scale, most prominently in Estonia. An independent evaluation by Springall et al. concluded that the system's security rested heavily on assumptions about voters' computers and on the operational discipline of election staff, and it raised concerns about both [2]. On the research side, Helios showed a different path: a web-based system in which ballots are encrypted, each voter can check that their ballot was included, and anyone can audit the tally [3]. Its authors were open about its target, which is elections in which coercion is unlikely, such as those held by clubs and universities. Rivest's idea of software independence gives a compact way to state what such systems aim for: an undetected change in the software should not cause an undetected change in the outcome [4].

Our prototype is not in this class. It has no cryptographic verifiability and no software independence. We mention these works because they define what a serious system should aim for, and therefore where a prototype like ours falls short.

2.2 One-time passwords

The two best-known OTP standards are HOTP, which derives a code from a shared secret and a counter [5], and TOTP, which replaces the counter with the time [6]. Our scheme is simpler and different in kind: the server generates a random code, delivers it over a separate channel and checks that the user can return it. There is no shared secret between the user and the server. This is closest to what NIST's digital identity guidelines call an out-of-band authenticator [7].

Two parts of that guidance matter here. The verifier is expected to limit failed attempts and to treat codes as short-lived and single-use, and we follow both. However, the same document says that channels which do not prove possession of a particular device, e-mail being the named example, should not be used for out-of-band authentication [7]. We accept this as a real limitation. E-mail was chosen because it is free, universal and needs no phone-network agreement, which suits small elections, but it makes the OTP a proof of inbox access and nothing stronger.

2.3 Photographs and biometrics

It helps to separate two things that are often lumped together. Face capture stores an image as a record. Face recognition compares an image against a reference and produces a match decision. Capture adds accountability and a mild deterrent, because a person is less likely to vote for someone else if their own face is logged. Recognition adds genuine identity assurance but brings error rates, uneven accuracy across groups and heavier privacy duties. This paper implements capture only.

A face photograph is personal data. India's Digital Personal Data Protection Act, 2023, places obligations of notice, consent and purpose limitation on those who process such data [8]. For that reason the registration page asks for explicit consent to photo capture, the photo is used only to record who voted, and it is visible only to the administrator.

2.4 The gap this work addresses

Most of the literature concerns large-scale systems and their failures. There is much less that walks a student or a small organisation through a working, understandable system, including the unglamorous parts: how to send an e-mail reliably, why a serverless host loses a SQLite file, why the browser refuses to open the camera on a plain HTTP address. This paper tries to fill part of that space.

3. Requirements and Threat Model

3.1 Functional requirements

Table 1. Functional requirements

ID	Requirement
FR1	A visitor can register with a full name, an e-mail address and consent to photo capture.
FR2	The system e-mails a six-digit OTP to the address entered.
FR3	The OTP expires, can be tried a limited number of times and can be requested again only after a short wait.
FR4	A verified voter must capture a webcam photo before voting.
FR5	The voter selects one candidate and confirms the choice in a review step.
FR6	At most one vote is recorded for each e-mail address.
FR7	An administrator can sign in, see live totals and turnout, and search the log of voters.
FR8	Every candidate is shown with the same layout: name, party and party symbol.

3.2 Non-functional requirements

Usability was treated as a security property, since a confusing interface produces mistakes and abandoned votes. The pages work on phones and laptops, can be operated by keyboard, and switch off decorative motion for users who ask their system to reduce it. On the security side we set the goals of limiting guessing, blocking double voting and separating identity from choice. Portability meant one code base for SQLite and PostgreSQL. Maintainability meant keeping all server logic in one readable file with short functions.

3.3 Threat model

The assets we want to protect are the integrity of the tally, the eligibility of voters and the secrecy of each choice. We considered four kinds of adversary:

- an impersonator who tries to vote as somebody else by guessing or intercepting an OTP;
- 1. a repeat voter who tries to vote more than once, for example from two browser tabs;
- 2. a curious insider, such as an administrator with database access, who wants to learn how a particular person voted;
- 3. a network or web attacker who tries to steal a session, forge a request or upload a malicious payload as a photo.

We deliberately left out coercion or vote-selling, malware on the voter's device, and well-resourced attackers who can compromise the hosting provider. These are hard problems that even mature systems only partly solve, and pretending to address them in a prototype would be dishonest.

4. System Design

4.1 Architecture

The system has three layers, shown in Figure 1. The browser renders the pages, runs a small amount of JavaScript for the OTP boxes and the camera, and talks to the server over HTTP or, when deployed, HTTPS. The Flask application holds all rules: who may vote, how OTPs are issued and checked, and

how a vote is stored. A relational database stores voters, candidates, votes and pending OTPs. A separate SMTP server is used only to deliver the OTP to the voter's inbox.

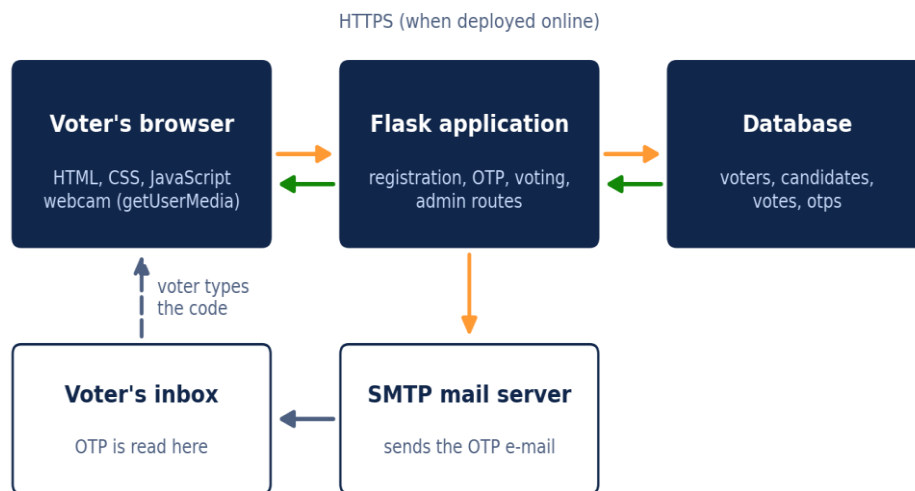


Figure 1. Overall architecture of the voting system

4.2 The voter journey

Figure 2 traces a voter from registration to a recorded vote. I kept the path deliberately linear. Each step only unlocks after the one before it actually succeeds, and this isn't checked in the browser, since that's easy to bypass. The server itself checks it on every single request, so nobody can skip ahead just by typing a different URL. A person who tries to open the voting page directly without verifying an OTP first is simply sent back to the start.

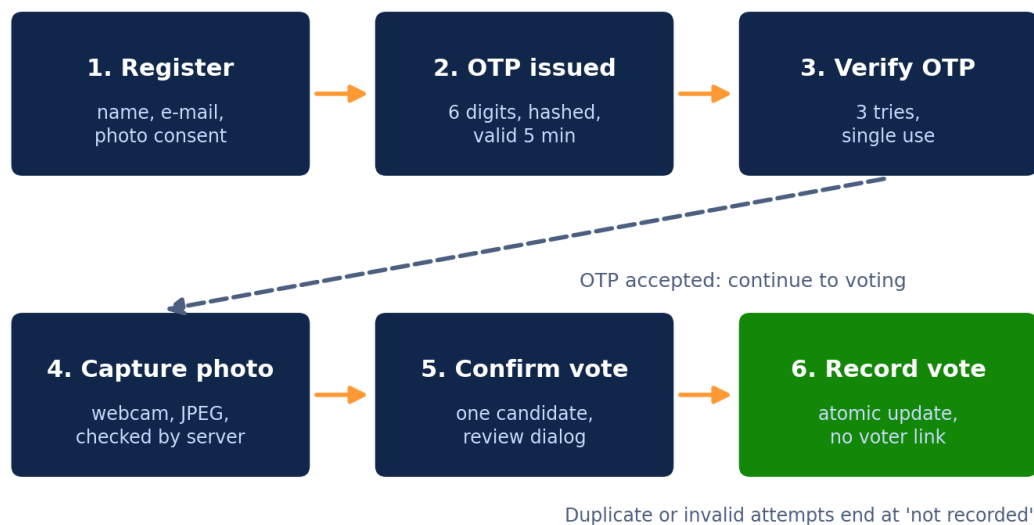


Figure 2. The six stages of the voter journey

4.3 Data model

Four tables carry the whole system (Table 2). The most important design choice is what the votes table leaves out. It stores only a candidate reference and its own identifier. The link between a person and the act of voting exists in a different table, as a flag and a timestamp.

Table 2. Database tables and their purpose

Table	Key columns	Purpose
voters	id, name, email (unique), has_voted, face_photo, voted_at	One row per registered person; records whether and when they voted.
candidates	id, name (unique), party, logo	The ballot: three candidates with their party and symbol image.
votes	id, candidate_id	One row per vote cast. No voter reference.
otps	voter_pk, otp_hash, expires, attempts, sent_at	The current pending OTP for a voter, stored only as a hash.

4.4 OTP design

An OTP is a six-digit number drawn with Python's secrets module, which uses the operating system's cryptographic random source. It is never stored in readable form. What is stored is an HMAC-SHA256 of the voter's identifier and the code, keyed with the server's secret key, so that a copy of the database alone does not reveal a valid code. The record carries an expiry time of five minutes, a counter of wrong attempts, capped at three, and the time it was sent. A new code cannot be requested until thirty seconds have passed, and issuing a new code replaces the old one. On success the record is deleted, so a code works exactly once. Comparison uses a constant-time function to avoid leaking information through timing.

4.5 Preventing duplicate votes

A common mistake is to check whether a person has voted and then, in a separate step, to record the vote. Two requests arriving together can both pass the check. We avoid this with a single conditional update: the voter's row is changed to `has_voted = 1` only if it currently holds 0. The database reports how many rows were changed. Only when that number is one is the vote inserted, and both statements are committed together. A second request for the same person changes nothing and is told that no vote was recorded. Because the e-mail column is unique, registering twice with the same address returns the same voter record, not a new one.

4.6 Ballot secrecy

The votes table has no voter column, so nothing in the tally can be joined back to a person through a foreign key. We call this structural secrecy. It protects against casual inspection and against the administrator page, which shows who voted but not for whom. It does not protect against a determined insider, a point we return to in Section 8.

4.7 Face capture

The browser asks for camera permission through the standard media-devices interface. When the voter presses the capture button, one frame is drawn onto a canvas scaled to 320 pixels wide and encoded as a JPEG at 70 per cent quality, which keeps a typical photo to a few tens of kilobytes. The image is placed in a hidden form field as a data URL. The server does not trust it. It checks the data-URL prefix, decodes the base64 strictly, confirms that the decoded bytes begin with the JPEG signature and rejects anything above a size cap. Requests larger than one megabyte are refused before they are read. The photo is stored with the voter's record and shown only on the administrator's log.

4.8 Interface design

We treated the interface as part of the security design. The OTP is entered in six separate boxes that move focus forward automatically, accept a pasted code and submit when the sixth digit is entered. A countdown shows how long the code remains valid, and the resend button stays disabled until the waiting period has passed. On the voting page, the candidate cards are equal in size and layout; selecting one shows a tick, and a sticky bar lists which of the two required steps (photo, candidate) is still open. The vote is not sent until the voter has seen a confirmation dialog with the chosen symbol and their own photo. After a successful vote a short confetti animation plays, which is skipped for users who have asked for reduced motion.

4.9 Administrator dashboard

The administrator signs in with a password compared in constant time. The results page shows a bar and a percentage for each candidate, the number of votes cast and the number of registered voters. It fetches a small JSON document every eight seconds and updates the bars without reloading. Below the results is the voter log with photo, name, address and time, which can be filtered by typing part of a name or address.

5. Implementation

5.1 Technology stack

Table 3. Technologies used

Layer	Technology and role
Server	Python 3 with Flask: routing, sessions, templates.
Database	SQLite for development; PostgreSQL (through psycopg) for the hosted deployment.
E-mail	Python smtplib with STARTTLS, configured through environment variables.
Front end	Jinja templates, plain CSS and JavaScript; no front-end framework.
Camera	Browser media-devices interface and the canvas element.
Hosting	Vercel serverless functions, with the database on a managed PostgreSQL service.

5.2 Two short excerpts

The two excerpts below carry most of the system's guarantees. The first issues an OTP and stores it as a hash, replacing any earlier one for the same voter with an upsert that both SQLite and PostgreSQL understand.

```
otp = f'{secrets.randbelow(10**6):06d}'
db().execute("""INSERT INTO otps
(voter_pk, otp_hash, expires, attempts, sent_at)
VALUES (?, ?, ?, 0, ?) ON CONFLICT (voter_pk) DO UPDATE SET
otp_hash=excluded.otp_hash, expires=excluded.expires,
attempts=0, sent_at=excluded.sent_at""",
(voter["id"], hash_otp(voter["id"], otp), now + OTP_TTL, now))
```

The second records the vote. The row count tells the program whether this request was the one that flipped the voter's flag.

```
done = conn.execute(
"UPDATE voters SET has_voted=1, face_photo=?, voted_at=? "
"WHERE id=? AND has_voted=0", (photo, now, pk))
if done.rowcount == 1:
    conn.execute("INSERT INTO votes (candidate_id) VALUES (?)", (cid,))
conn.commit()
```

5.3 One code base, two databases

A small wrapper class opens a SQLite connection or a PostgreSQL connection depending on whether a connection string is present in the environment. It rewrites the question-mark placeholders to the style PostgreSQL expects and otherwise passes SQL through unchanged. Three habits keep the SQL portable: the upsert syntax shown above, timestamps generated in Python instead of by a database function, and integers converted explicitly where PostgreSQL returns a different numeric type from SQLite. The schema is created automatically on start-up and the three candidates are inserted if the table is empty, so a fresh deployment needs no set-up commands.

6. Deployment Experience and Lessons

Running the application on a laptop took an afternoon. Putting it on a public serverless platform took longer and taught more. Five problems are worth recording because anyone repeating the exercise is likely to meet them.

- 1) The file system does not persist. A serverless function cannot keep a SQLite file between requests, so votes would silently disappear. Moving to a hosted PostgreSQL database was not optional.
- The secret key must be fixed. Sessions are signed with it, and in our design the OTP hash is keyed with it. If each function instance generates its own random key, a code issued by one instance cannot be verified by another. The application now refuses to start online unless a key is set.
- Static files live elsewhere. The platform serves a public folder from its content network, so we moved the stylesheet and party symbols there and set the static path accordingly, which also works unchanged on a laptop.
- The camera needs a secure context. Browsers allow camera access on HTTPS pages and on localhost but not on plain HTTP addresses, so the camera step worked at home and would have failed on any unencrypted test server.
- Sending e-mail is its own project. A mail account normally used through a web page cannot be used by a program with its ordinary password, and needs a separate app-specific password with two-step verification switched on. Even then, messages to other recipients did not always arrive and may have been filtered as spam, while messages sent to our own address did, which made early testing misleadingly successful. The program now catches sending errors, deletes the pending code and tells the user, instead of leaving them waiting for an e-mail that will not arrive. For anything beyond a demonstration, a transactional mail service is the better choice.

None of these is deep computer science, but each one, on its own, would have ended a live demonstration.

7. Testing and Results

7.1 Method

We wrote fourteen functional tests that drive the application through Flask's built-in test client against a fresh SQLite database. In development mode the OTP is printed to the console instead of being e-mailed, and the tests read it from there. The photographs used were synthetic JPEG byte strings, so the tests exercise validation logic and not image quality. Each test states a scenario and an expected outcome, and passes only if that outcome occurs. Separately, the developer walked through the browser flow, including the camera, the animation and the layout, on a desktop computer. That walk-through was informal and is not counted as a test.

7.2 Results

All fourteen tests passed (Table 4). The most informative was T9, which registers and verifies the same person in two independent browser sessions and then submits a vote from each. The first vote was recorded and the second was reported as not recorded, and the vote count stayed at one. T14 checked the structure of the votes table and confirmed that the number of votes equalled the number of voters flagged as having voted. Computing the keyed hash of an OTP took about two microseconds in our test environment, so the hashing adds no noticeable delay.

Table 4. Functional test cases and outcomes

ID	Scenario	Expected outcome	Result
T1	Register, verify OTP, capture photo, vote	Vote recorded; tally increases by one	Pass
T2	Register again with an e-mail that has voted	Refused with a clear message	Pass
T3	Three wrong OTPs, then the correct one	Code locked; correct code refused	Pass
T4	Enter the OTP after its expiry time	Refused	Pass
T5	Request a new OTP within 30 seconds	Refused with a wait message	Pass
T6	Open the voting page without verifying	Redirected to the start page	Pass
T7	Submit a vote with no photo, or a PNG photo	Refused	Pass

ID	Scenario	Expected outcome	Result
T8	Submit an unknown candidate identifier	Refused	Pass
T9	Same voter in two sessions, both vote	First recorded; second not recorded	Pass
T10	Invalid e-mail format, or consent not ticked	Refused	Pass
T11	Open the admin page without signing in	Redirected to admin sign-in	Pass
T12	Sign in with a wrong admin password	Refused	Pass
T13	Sign in with the correct admin password	Results page shown	Pass
T14	Inspect votes table and compare counts	No voter column; votes equal voters flagged	Pass

7.3 What the tests do not show

The evidence is narrow and it is better to say so plainly. We ran no load test, so we cannot state how many simultaneous voters the system supports. We ran no penetration test and no independent code review. There was no usability study with real participants, so the claim that a first-time user can finish unaided remains a design goal and not a measured result. We did not run the automated tests against PostgreSQL for this paper; the hosted path uses the same SQL apart from placeholder conversion, but confirming that with the full test set is on the list of future work.

8. Security Discussion

Table 5 sets each threat from Section 3 beside the control that addresses it and the risk that remains. The remaining risk column is the more important one.

Table 5. Threats, controls and residual risk

Threat	Control in the prototype	Residual risk
Guessing an OTP	Random six digits; 3 tries; 5-minute expiry; 30-second resend gap	About 0.036% per hour against one address; no per-IP throttling
Voting twice	Unique e-mail; conditional atomic update	One person with many e-mail addresses
Reading the OTP table	Codes stored as keyed hashes	If the secret key leaks, a million values can be tested quickly
Session theft or forged requests	Signed cookies; HttpOnly; SameSite=Lax; Secure online	No anti-CSRF tokens
Learning how someone voted	No voter reference in the votes table	An insider could align vote order with voting times
Malicious or oversized photo	Prefix, strict base64, JPEG signature, size cap	No liveness check; any JPEG is accepted
Unauthorised access to results	Password-protected admin pages, constant-time compare	One shared password; no audit log
Tampering with the tally	Database access control only	No way for a voter or auditor to verify the count

8.1 How risky is a six-digit OTP here?

Each issued code has one million possible values and allows three guesses, so a blind attacker succeeds with probability of about 3 in 1,000,000 per code. Since I set the resend gap to thirty seconds, one address can receive at most 120 codes in an hour, which works out to roughly a 0.036 per cent chance of someone guessing a valid code for that specific address within the hour. On its own that is a small risk. But I only rate-limited by address, not by network address, so an attacker holding a list of thousands of registered e-mails and trying them all at once could expect a few to succeed purely by chance. A per-IP limit and a longer code would close this cheaply.

8.2 The weakest points

Three weaknesses deserve more attention than the others. The first is the multiple-identity problem. Open registration means anyone with several e-mail addresses can vote several times, and no amount of OTP checking changes that. The earlier version of our system avoided it by requiring a voter identifier from a pre-approved list. The trade-off is administrative effort against openness.

The second is secrecy against insiders. Vote rows are numbered in the order they are created, and voter rows carry a voting time with one-second resolution. Someone with read access to the database could line the two lists up and, in a quiet election, infer individual choices. Removing sequential identifiers, dropping timestamps or shuffling stored votes would reduce this, but none of them would make it impossible.

The third is the absence of verifiability. A voter has no way to confirm that their vote was counted, and an auditor cannot confirm that the totals match what voters cast. In Rivest's terms the system is not software independent [4]. Fixing that requires cryptographic techniques of the kind Helios uses [3], and it is a research-sized task, not a patch.

9. Limitations

4. E-mail is a weak second channel and does not satisfy the out-of-band requirement in NIST's guidance [7].
5. Face capture records an image but does not verify that it belongs to the voter or that a live person is present.
6. Open registration cannot stop one person from creating several voter records.
7. The ballot stays secret because of how the database is structured, not because of any cryptography, and there is no way for a voter to verify afterwards that their vote was actually counted.
8. The photographs are sensitive personal data, and right now the prototype just stores them in the same database as everything else, with no separate encryption and no automatic deletion once the election is over.
9. Testing was functional only, on synthetic data, without load, security or usability evaluation.

10. Future Work

Several extensions follow directly from the weaknesses above. A pre-approved voter list, imported from a spreadsheet, would close the multiple-identity gap in organisations that already know their members. SMS or authenticator-app codes would provide a stronger second factor than e-mail. Face matching against an enrolled reference photograph, combined with a liveness check, would move the photo from evidence to verification, although it would also raise the privacy and fairness questions discussed in Section 2.3. On the operational side, a fixed voting window, an audit log of administrator actions, per-IP rate limiting, anti-CSRF tokens and scheduled deletion of photographs after the result is declared are all modest changes with clear benefit. Longer term, a cryptographic receipt scheme in the spirit of Helios would address verifiability. Finally, the automated tests should be extended to run against PostgreSQL and complemented by a small usability study.

11. Conclusion

We set out to see how much of a polling booth's trust could be rebuilt with free tools and a few hundred lines of Python. The answer is: some, but not all. The prototype reliably enforces the parts that can be enforced by careful programming. It verifies inbox access with a short-lived, hashed, rate-limited code, blocks double voting with a single atomic database operation, keeps the tally free of voter references, and records a photo as evidence of participation. All fourteen functional tests passed.

What it cannot do is equally instructive. It cannot prove that a voter is who they say they are, cannot stop a person with several mailboxes from voting several times, and cannot let anyone check the count. These are not bugs to be fixed in the next release. They mark the point at which a demonstration ends and a real election system, with its legal, cryptographic and organisational machinery, begins. For a classroom, a club or a society where members know one another and the stakes are modest, the design is a sensible starting point. For anything larger, it is a teaching aid that shows why the harder problems are hard.

References

- [1] T. Kohno, A. Stubblefield, A. D. Rubin and D. S. Wallach, "Analysis of an electronic voting system," in Proc. IEEE Symposium on Security and Privacy, 2004.
- [2] 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 Conference on Computer and Communications Security (CCS), 2014.
- [3] B. Adida, "Helios: Web-based open-audit voting," in Proc. 17th USENIX Security Symposium, 2008.
- [4] R. L. Rivest, "On the notion of 'software independence' in voting systems," Philosophical Transactions of the Royal Society A, vol. 366, no. 1881, 2008.
- [5] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache and O. Ranen, "HOTP: An HMAC-based one-time password algorithm," RFC 4226, IETF, 2005.
- [6] D. M'Raihi, S. Machani, M. Pei and J. Rydell, "TOTP: Time-based one-time password algorithm," RFC 6238, IETF, 2011.
- [7] P. A. Grassi et al., "Digital identity guidelines: Authentication and lifecycle management," NIST Special Publication 800-63B, 2017 (see section on out-of-band authenticators).
- [8] Government of India, The Digital Personal Data Protection Act, 2023.
- [9] Pallets Projects, "Flask documentation," <https://flask.palletsprojects.com>.
- [10] OWASP Foundation, "OWASP Top 10: 2021," <https://owasp.org/Top10/>.

Note: references [9] and [10] informed implementation choices (session handling, input validation, and the classification of authentication and access-control failures) and are listed for completeness.