Elect: A Secure, Modular Digital Elections Platform

Version 1.0

Abstract

Elect is a full-stack digital elections platform designed for organizations to create, manage, and run secure elections at scale. It supports traditional CSV-driven workflows and API-managed integrations, enabling organizations to onboard candidates and voters, verify voter identity with OTPs, cast votes securely, and view results. The backend leverages FastAPI, SQLAlchemy 2, Alembic, PostgreSQL, Redis, and Stripe. The frontend is built with React, Vite, and Tailwind CSS. The system enforces granular roles (admin, organization owner, organization admin, voter), email verification for organizational accounts, OTP verification for voters (Twilio), payment top-ups via Stripe, and extensible notifications. While basic analytics and summaries exist, future AI integrations are planned for insights and summaries to assist election organizers.

1. Introduction

1.1 Business Logic and Problem Statement

Organizations (e.g., sports clubs, student unions, professional bodies, NGOs, businesses) routinely need to run elections in a secure, repeatable, and cost-effective way. Traditional offline processes are slow, error-prone, and costly; ad-hoc digital solutions often lack robust identity verification, auditability, role-based access, or integrations with payment and messaging systems. Elect addresses this by providing:

- **Role-based governance**: system admins, organization owners, organization admins, and voters each follow tailored flows.
- **Flexible onboarding**: either upload CSVs (candidates/voters) or integrate with an external API (or internal dummy service) to verify voters and derive eligibility dynamically.
- **Voter verification**: OTP-based verification via Twilio; voters cannot vote twice and can only vote while an election is running.
- **Secure vote casting**: controlled by election runtime windows, vote limits per voter, and strict relationship checks.
- **Operational guardrails**: email verification for organizations and org admins; optional payment gating; notifications trail for key events.
- **Extensibility**: Stripe wallet top-ups and future pricing per potential voter; Cloudinary-backed image uploads; planned AI insights and summaries.

1.2 Importance

Elect streamlines the entire election lifecycle:

1. **Registration and verification**: organizations register, verify email, and are accepted by a system admin before operating.
2. **Election setup**: organizations create elections (CSV or API method), onboard candidates and voters, and define time windows and vote limits.
3. **Voter identity and participation**: voters receive OTPs via SMS to authenticate; the system ensures 1-person-1-vote constraints.
4. **Results and transparency**: results endpoints compute winners, turnout, and statistics, while notifications create an operational audit trail.

2. Team Work

2.1 Collaboration with Git

The project uses Git for version control and collaborative development. Teams can adopt:

- **Branching strategy**: feature branches and pull requests to review changes, ensuring code quality and traceability.
- **Code reviews**: enforce standards and knowledge sharing via PRs and issues.

2.2 Reproducible Environments with Docker

Docker and docker-compose standardize the development environment across machines:

- Identical service versions (PostgreSQL, Redis, Nginx).
- Hot-reload for backend/frontend during development via bind mounts.
- Simplified bootstrap: one command to provision dependencies and internal networking.

3. Docker Architecture

3.1 Services

As defined in `docker-compose.yml`:

- **backend** (`./backend`):
  - FastAPI app (Uvicorn) exposed at `8000`, with volume mount for live dev.
  - Depends on **postgres** and **redis**.
  - Loads environment from `.env.example` (supply a real `.env` in production).
- **frontend** (`./frontend`):
  - React + Vite app exposed at `3000`, bind-mounted for live dev.
- **postgres** (`postgres:15`):
  - Primary database with a named volume `postgres`.
  - Exposed to host on `55555:5432` for optional local tools.
- **redis** (`redis:alpine`):
  - Used by **fastapi-limiter** for rate limiting.
- **nginx** (`nginx:alpine`):
  - Reverse proxy (`nginx/nginx.conf`) exposed on `80`.
  - Proxies to backend and frontend within the Docker network.
- **pgadmin** (`dpage/pgadmin4`):
  - Optional DB admin UI on `5051:80`, with `pgadmin` volume.

3.2 Interactions

- Frontend calls backend (`/api/...`) through Nginx.
- Backend talks to PostgreSQL (SQLAlchemy async engine) and Redis (rate limiting/limiter init).
- The `/api/proxy/dummy-service/...` endpoint allows the UI to appear as if it calls an external service while the backend proxies to an internal dummy service for API-managed elections.

4. Technical Stack

4.1 Backend

- **FastAPI** (ASGI), **Uvicorn**.
- **SQLAlchemy 2.0** ORM, **Alembic** for migrations.
- **PostgreSQL** (`asyncpg`, `psycopg2-binary`).
- **Auth and Security**: `python-jose` (JWT), `passlib[bcrypt]`.
- **Email**: `fastapi-mail` (SMTP), verification tokens with **Pydantic** models.
- **Messaging/OTP**: `twilio` for SMS OTP.
- **Payments**: `stripe` integration for wallet top-ups and webhooks.
- **Rate Limiting**: `fastapi-limiter` backed by **Redis**.
- **Media**: `cloudinary` for image uploads (candidate photos, symbols).
- **Async HTTP**: `httpx`, `aiohttp` for API integrations.

4.2 Frontend

- **React** (Vite), **React Router**, **@tanstack/react-query**.
- **Tailwind CSS** and utility plugins for responsive UI.
- Pages for organization onboarding, dashboards, payment flows, voter login/experience, and election management/results.

5. ERD and ORM (SQLAlchemy)

5.1 Core Entities and Relationships

Below is a textual ERD overview (all models are SQLAlchemy declarative classes):

- **User** (`users`)
  - Fields: `id` (PK), `email` (unique), `password`, `first`, `last_name`, `role` (`admin`|`organization`|`organiz`
    `is_active`, `wallet`, `stripe_session_id`, timestamps.
  - Relations: one-to-one with **Organization** (if role=`organization`); one-to-many **VerificationToken**.
- **Organization** (`organizations`)

  - PK is `user` (FK → `users.id`), plus `name` (unique), `country`, `status` (`pending`|`accepted`|`rejected`), `is_paid`.
  - Relations: has many **Election**, **Candidate**, **Notification**.

- **OrganizationAdmin** (`organization`)
  - Composite concept: `user` (PK, FK → `users.id`) belongs to `organization_user_id` (FK → `organizations.user_id`).

- **Election** (`elections`)
  - Fields: `id` (PK), `title`, `types` (e.g., `simple`/`district`/`governorate_based`/`api_managed`), `status`, `starts_at`, `ends_at`, `num_of_votes_per_voter`, `potential_number_of_voters`, `method` (`api`|`csv`), `api_endpoint`, `organization_id` (FK → `organizations.user_id`).
  - Relations: has many **Voter**, **VotingProcess**, **CandidateParticipation**, **Notification**.

- **Candidate** (`candidates`)
  - PK: `hashed_id`; attributes include `name`, `district`, `governorate`, `country`, `party`, `symbol_icon_url`, `symbol_name`, `photo_url`, `birth_date`, `description`, `organization_id` (FK → `organizations.user_id`).
  - Relations: many-to-many to **Election** via **CandidateParticipation**.

- **CandidateParticipation** (`candidate`)
  - Composite PK: `candidate_national_id` (FK → `candidates.hashed_national_id`), `election_id` (FK → `elections.id`).
  - Fields include `vote`, `has_won`, `rank`.

- **Voter** (`voters`)
  - Composite PK: `voter_national_id`, `election_id` (FK → `elections.id`).
  - Attributes: `phone0`, `governerate`, OTP fields (`is_verified`, `otp_code`, `otp_expires_at`, `last_verified_at`); API elections: `eligible_candidates` (JSON string), `is_api_voter`.

- **VotingProcess** (`voting`)
  - Composite PK: `voter_national_id`, `election_id` (FK → `elections.id`); records each successful vote event.

- **Notification** (`notifications`)
  - Fields: `id` (PK), `organization` (FK), `type` (rich enum covering election/candidate/voter/system events), `priority`, `title`, `message`, optional `election_id`, `candidate_id`, `voter_id`, `additional_data` (JSON), timestamps, read flags.

- **VerificationToken** (`verification`)
  - Used for email verification and password resets, linked to **User**.

- **Transaction** (`transactions`)
  - Wallet top-ups and spending records linked to **User**.

- **DummyCandidate** / **DummyVoter**

  - Internal testing tables for the dummy service supporting API-managed elections (simulate an external organization's API).

- **ApprovalRequest**
  - Generic approval workflow entity (create/update/delete against `election` or `candidate`); used primarily for staged operations when needed.

5.2 Modeling Notes

- Organization's primary key is the owning `users.id` (`organizations.user`); this simplifies linking elections/candidates directly to an organization owner identity.
- `CandidateParticipation` captures vote counts and final ranking (set after the election ends).
- Voter OTP state is stored on `voters` to gate `/voting/election/{id}/vote`.
- Enumerations: `UserRole`, `Country`, `Status`, and rich `NotificationType`.

6. User Types and Flows

6.1 System Admin

- **Approves organizations**: sets `accepted`/`rejected`, toggling `users.is,`.
- **Oversees activity**: dashboards, list active elections, inspect election details, manage organization admins.

6.2 Organization (Owner)

1. **Register** via `/api/auth/register`.
2. **Email verification**: `EmailService` sends a tokenized link, `/api/auth/verify-email`.
3. **Admin acceptance**: system admin accepts organization (status `accepted`) which activates login.
4. **Payment (optional gating)**: Stripe wallet top-up; model field `organizations.is` available to gate election creation if enabled.
5. **Create elections**:
   - *CSV method*: upload candidates/voters CSVs; server hashes national IDs, validates required columns per election type.
   - *API method*: set `api`; voter eligibility is verified against the external or dummy API.
6. **Manage candidates**: CRUD with Cloudinary-backed images and participation links; constraints prevent editing while election is running.
7. **Notifications**: on creation/update/delete events for elections/candidates and other activities.

6.3 Organization Admin

- Created by the organization owner (or system admin) and **must verify email** before login.
- Has operational privileges paralleling the owner for day-to-day tasks (election/candidate management), with notifications capturing admin actions.

6.4 Voter

1. **Onboarding**:
   - *CSV-based elections*: voters are preloaded via CSV.
   - *API-based elections*: backend (or UI via proxy) verifies eligibility by national ID against the configured API (dummy or external); creates/updates voter records and eligible candidates.
2. **OTP login**: `/api/voters/login/request-otp` generates a 6-digit OTP via Twilio SMS; rate-limited and time-bound.
3. **OTP verification**: `/api/voters/login/verify-otp` marks the voter verified if code and timing checks pass and no prior vote recorded.
4. **Voting**: `/api/voting/election/{id}/vote` enforces:
   - election is running (`starts`$\leq$ `now` $\leq$ `ends_at`);
   - voter exists, is verified, and has not yet voted in that election;
   - exact number of candidate selections equals `num_votes_per_voter`;
   - each selected candidate is a participant in that election.

6.5 Identity & Security Highlights

- JWT-based auth for organization roles; email verification gating for organization and organization admin accounts.
- OTP via Twilio; centralized SHA-256 hashing for national IDs (`core.shared.hash_id`).
- Rate limiting via Redis-backed `fastapi-limiter`.

7. Payment Using Stripe

7.1 Current Implementation

- **Wallet top-up** flow:
  1. `/api/payment/create-checkout-session` creates a Stripe Checkout Session for the selected amount (EGP).
  2. `/api/payment/payment-success` is the success redirect handler; verifies session, credits user wallet, records a **Transaction**, and sets `Organization.is= true`.
  3. `/api/payment/webhook` handles `checkout.session.completed` to ensure idempotent wallet crediting even if the redirect is interrupted.
  4. `/api/payment/wallet` and `/api/payment/transactions` expose basic wallet info and history.
- Robust key checks ensure `sk_.../sk_live_...` is configured before Stripe calls.

7.2 Planned Pricing Model

- **Per-election charge** proportional to **potential number of voters**.
- UX: during election creation or publishing, compute amount based on `potential0_of_voters`, initiate a dedicated Checkout session, and lock creation/publish until payment success.
- Back-office: store pricing tiers and audit charges; reconcile via Transactions and webhooks.

8. AI Integration

8.1 Current Status

- A frontend component (`SummaryGenerator.jsx`) produces an *AI-like* narrative summary for results as a placeholder.
- The repository includes an `ai/` directory with scripts and a scikit-learn model artifact (`participation.pkl`) to bootstrap future analytics (e.g., turnout prediction).

8.2 Planned Capabilities

- **Summaries**: natural-language narratives contextualizing winners, margins, turnout, and comparisons to past elections.
- **Insights**:
  - predicted turnout vs. actual (using features from historical CSVs and live tallies);
  - anomaly detection flags (sudden spikes, geographic irregularities);
  - driver analysis (which candidate attributes correlated with success).
- **APIs and UI**: secure endpoints returning insight objects; UI components rendering explainable charts and text.

9. Conclusion

Elect provides a secure, modular, and extensible foundation for digital elections. It balances ease-of-use (CSV workflows), enterprise integration (API-managed elections), robust identity checks (email and OTP), operational governance (roles, notifications), and revenue enablement (Stripe). Future AI features will deepen organizers' understanding, making elections not just operationally efficient but strategically insightful.