TUNAX:
Tunisian Municipal Tax Management System

Rayen Ben Abdallah

December 2025

Abstract

The Tunisian Municipal Tax Management System (TUNAX) is a full-stack platform that digitizes the administration of local taxes, permits, disputes, and civic participation for municipalities. Built with a Flask backend, PostgreSQL persistence, and modular dashboards rendered through static HTML and vanilla JavaScript, the solution operationalizes the 2025 Code de la Fiscalité Locale. This report documents the motivation, architectural choices, implementation details, evaluation methodology, and socio-economic considerations of the platform. It blends academic rigor with practitioner insights to serve as a reference for engineers, policy makers, and researchers studying digital transformation in municipal tax administration.

Keywords: Fiscalité Locale, e-government, municipal tax, Flask, participatory budgeting, Tunisia.

Contents

1 Introduction

Local tax collection in Tunisia has historically depended on fragmented information systems, manual declarations, and opaque workflows. The 2025 revision of the Code de la Fiscalité Locale mandates modernization efforts that simultaneously improve compliance, transparency, and service quality. The TUNAX project responds to this policy context by designing a modular digital platform that covers the entire lifecycle of property tax (Taxe sur les Immeubles Bâtis, TIB), land tax (Taxe sur les Terrains Non Bâtis, TTNB), dispute resolution, payment attestation, and participatory budgeting. This report provides a holistic overview of the system and highlights academically relevant lessons learned during implementation.

1.1 Objectives

- Translate legal articles from the 2025 tax code into programmable workflows.
- Offer multi-role dashboards (citizen, business, agent, inspector, finance, contentieux, urbanism, municipal admin ,ministry admin).
- Expose RESTful APIs conforming to the OpenAPI specification and tested with Insomnia collections.
- Integrate geospatial context through OpenStreetMap to ground declarations in physical reality.
- Enable accountability through audit logs, JWT-based authentication, and analytics.

1.2 Research Questions

This academic report frames the engineering work around three guiding questions:

1. **RQ1: Legal Encoding Precision** — How can a code-centric representation of fiscal law (via Marshmallow schemas and SQLAlchemy constraints) reduce manual review burden and prevent illegal tax assessments?
2. **RQ2: Architectural Maintainability** — Which modular patterns (Flask blueprints, role-based dashboards, Docker Compose parity) enable municipal solutions to remain maintainable across heterogeneous deployment environments (cloud vs. on-premise)?
3. **RQ3: Dispute Reduction via Transparency** — Do HATEOAS-driven workflows and real-time penalty calculations reduce citizen disputes compared to opaque manual systems? (Metric: dispute rate per 1000 declarations)

2 Background and Policy Context

Tunisia's municipal tax system relies on two pillars: property taxation based on constructed area and land taxation based on surface type and valuation. The 2025 reform emphasizes digital declarations, proactive dispute management, and participatory budgeting to increase trust. Prior research notes that digitization significantly enhances tax morale when it pairs transparency with responsive services. However, municipalities often lack IT capacity, making reference implementations such as TUNAX valuable for replication.

2.1 Legal Foundations

Articles 1–34 of the Code govern TIB computation, while Articles 32–33 govern TTNB. Article 13 imposes proof-of-payment conditions for building permits, and Articles 23–26 describe the contentieux workflow. TUNAX encodes each article into modular service layers:

- Declarative schemas validate property and land payloads, guaranteeing legal compliance before persistence.
- Declarative service tables capture rate schedules, exemptions, and service availability thresholds.
- Workflows enforce deadlines and escalate unresolved disputes to commissions.

2.2 Tax Calculation Formulas

The platform implements precise calculations per Code de la Fiscalité Locale 2025:

TIB (Property Tax)

1. Compute taxable base: Assiette = $0.02 \times (\text{reference\_price\_per\_m}^2 \times \text{surface\_couverte})$
2. Apply service rate: TIB = Assiette $\times$ service_rate
3. Service rates range from 8% (minimal services) to 14% (full services)

TTNB (Land Tax)

$$\text{TTNB} = \text{surface} \times \text{urban\_zone\_tariff}$$

Urban zone tariffs per square meter:

- Haute densité: 1.2 TND/m^2
- Densité moyenne: 0.8 TND/m^2
- Faible densité: 0.4 TND/m^2
- Périphérique: 0.2 TND/m^2

Late Payment Penalties Penalties start January 1 of year N+2 for unpaid year N taxes:

$$\text{Penalty} = \text{tax\_amount} \times 0.0125 \times \text{months\_overdue}$$

Penalties compound monthly at 1.25% after the grace period expires.

2.3 Stakeholder Analysis

The platform maps stakeholders to the following personas:

Citizens and Businesses submit declarations, follow required documentation, and monitor tax bills.

Agents verify declarations, process payments, and triage reclamations.

Inspectors capture field evidence (e.g., geotagged plots) and trigger penalties.

Finance Offices reconcile budgets, forecasts, and participatory voting results.

Urbanism Teams gate permit approvals on proof of tax compliance.

Municipal Administrators coordinate staff permissions, approve service configurations, and act as the bridge to ministry directives by publishing local circulars.

Ministry Administrators oversee national policy, monitor aggregate performance metrics across all communes, manage system-wide configurations, and coordinate policy updates with municipalities.

Understanding each persona's workflow informed the multi-dashboard frontend design.

2.4 Municipal Admin Role

Municipal admins receive a dedicated dashboard that aggregates operational intelligence:

- **Staff Orchestration**: Provision or revoke user roles (agent, inspector, finance, etc.) while enforcing least privilege.
- **Configuration Management**: Edit municipal service coverage, reference prices per square meter, and document requirement templates before the tax season opens.
- **Escalation Hub**: Monitor disputes, reclamations, and unpaid assessments to decide when to escalate cases to ministry auditors.
- **Civic Engagement**: Launch participatory budgeting cycles and publish project shortlists that citizens vote on through the frontend.

The municipal admin experience closes the loop between policy design and daily enforcement, ensuring each commune can tailor national guidance to local realities.

3 System Overview

Iterative, test-driven approach: requirements from legal documents and municipal interviews, domain modeling via SQLAlchemy, implementation using Flask blueprints and Marshmallow schemas, verification through 183 Insomnia test scenarios, and documentation in Markdown and LaTeX. Data sources: communes_tn.csv (264 municipalities), codes.csv (delegations/localities), tariffs_2025.yaml (tax rates).

4 System Overview

TUNAX follows a layered architecture summarized in Figure 1. The backend exposes versioned REST endpoints under `/api`, whereas the frontend offers pre-built dashboards served by nginx. Data persistence relies on PostgreSQL 15, orchestrated via Docker Compose for parity between development and production.

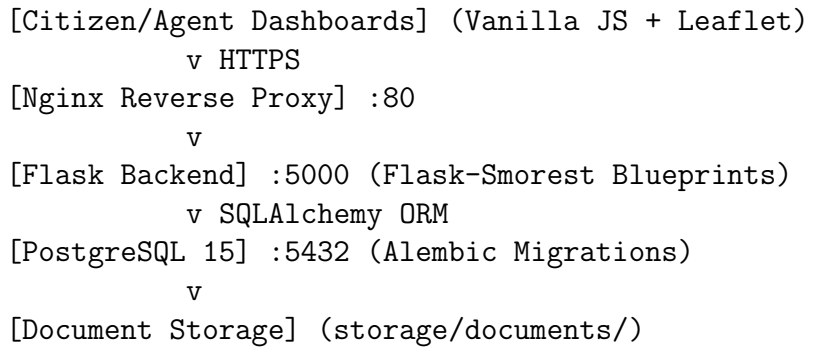

Figure 1: TUNAX deployment topology showing separation of concerns and Docker orchestration

4.1 Backend Components

- **app.py**: Factory that initializes Flask, SQLAlchemy, JWT, and API blueprints.
- **Extensions**: Database session management, caching hooks, and rate limiting.
- **Resources**: REST controllers such as `public.py`, `auth.py`, `tib.py`, `ttnb.py`.
- **Models**: Entities for communes, properties, lands, payments, disputes, inspections, and budgets.
- **Utils**: Calculator utilities for TIB/TTNB, geolocation helpers, and validators.

4.2 Frontend Components

Role-specific dashboards are implemented with vanilla JavaScript modules (e.g., `frontend/dashboards/`
Shared login screens rely on local storage tokens and simple fetch wrappers. Leaflet.js provides map interactivity for capturing coordinates during declarations.

5 Architectural Decisions

5.1 Technology Stack

Backend Framework Flask 2.x with Flask-Smorest for OpenAPI 3.0.2-compliant REST endpoints

ORM SQLAlchemy with Alembic for database migrations and schema versioning

Validation Marshmallow schemas ensure legal compliance before persistence

Authentication Flask-JWT-Extended with token blacklist and refresh flows (1-hour access, 30-day refresh)

Rate Limiting Flask-Limiter with Redis-backed persistence (migrated January 2026). Auth endpoints limited to 5/minute, general endpoints 50/hour. Production deployments use `REDIS_URL` for distributed rate limit enforcement across workers.

Database PostgreSQL 15 (production), SQLite fallback (local development)

Frontend Vanilla JavaScript ES6+ modules with Leaflet.js for interactive mapping

Deployment Docker Compose orchestrating backend, PostgreSQL, Redis, and Nginx containers. Health checks via `/health` endpoint monitor database connectivity.

Documentation OpenAPI 3.0.2 auto-generated via Flask-Smorest at `/api/v1/docs/swagger-ui`

5.2 Architectural Rationale

Flask was chosen for lightweight, blueprint-driven modularity and OpenAPI auto-generation. PostgreSQL ensures ACID guarantees and relational integrity critical for legal tax workflows. Docker Compose provides municipal IT teams with a manageable deployment model, avoiding Kubernetes complexity unnecessary for typical commune populations (20–50K citizens).

6 Backend Implementation Details

Core entities: Commune (municipal metadata), Property (TIB), Land (TTNB), Tax, Payment, Dispute, BudgetProject, SatelliteVerification. Public endpoints expose /tax-rates, /calculator, /communes with unauthenticated access. Authenticated routes enforce role checks: TIB/TTNB declarations, payment processing, dispute resolution per Articles 23–26. Inspector workflows persist satellite verification records (added January 2026) linking geospatial evidence to property/land assessments, supporting audit requirements and field validation traceability.

7 Frontend and User Experience

Dashboards implemented with vanilla JavaScript ES6+ and Leaflet.js for spatial reference. Example flows: citizens declare TTNB, businesses manage permits, inspectors capture GPS coordinates and issue penalties.

8 Security and Compliance

Security design follows layered defense:

- **Authentication**: JWT tokens with refresh flows; blacklist support for logout.
- **Authorization**: Role checks embedded in decorators; critical operations require admin roles.
- **Data Protection**: Passwords hashed using modern algorithms; sensitive environment variables stored in `.env`.
- **Audit Logging**: `backend/logs/tunax_info.log` and error logs capture API usage and failures for compliance audits.
- **Rate Limiting**: Health endpoints display warnings when thresholds are exceeded, mitigating DoS attempts.

9 Reproducibility and Deployment

9.1 Quick Start

The platform can be deployed in three commands:

```
cd docker
docker-compose up -d
docker-compose exec backend flask db upgrade
docker-compose exec backend python seed_all.py
```

This orchestrates:

1. PostgreSQL container with persistent volume (`postgres_data`)
2. Flask backend with hot-reloading enabled (port 5000)
3. Nginx serving static dashboards on port 80

9.2 Seeding Strategy

`seed_all.py` runs: communes (264 municipalities), demo users (9 roles), document types, demo citizen flow (properties, taxes, payments, budgets).

9.3 Docker Compose Architecture

Docker Compose orchestrates four services: PostgreSQL (persistent volume), Redis (rate limit store), Flask backend (port 5000), and Nginx frontend (port 80). Source directories are mounted for hot reloading. Health checks query the `/health` endpoint (consolidated January 2026), which validates database connectivity and returns HTTP 200 for healthy state or 503 on failure, enabling Docker and Kubernetes orchestrators to restart unhealthy containers automatically.

9.4 Implementation Timeline

Development proceeded in four phases: (1) Foundation — database schema, seed data, public endpoints, authentication; (2) Core Workflows — TIB/TTNB declarations, payments, documents, tax calculations; (3) Advanced Features — disputes, budgeting, security hardening, role dashboards; (4) Refinement — testing (183 Insomnia scenarios), documentation, and performance optimization.

10 Testing and Validation

10.1 Test Coverage

183 Insomnia test scenarios validate HTTP codes, Marshmallow schemas, RBAC, and legal compliance (tax payment windows).

10.2 Manual Verification

Dashboards undergo manual walkthroughs to verify UI synchronization with backend computations.

10.3 Observability

Application logs differentiate between informational events and stack traces to simplify root-cause analysis. Example: repeated health-check throttling indicates the need to adjust monitoring intervals or increase rate limits. Logs are stored in `backend/logs/tunax_info.log` and `tunax_error.log` with rotation policies. The consolidated `/health` endpoint (January 2026) provides real-time service status for monitoring dashboards, returning structured JSON with database connectivity state and version metadata, enabling proactive alerting before user-facing failures occur.

11 Data Governance and Privacy

Privacy-by-design: data minimization (only legally required fields), audit trails (7-year retention), role-scoped access, secure document storage, hashed passwords. Alignment with Loi Organique 2004-63 and GDPR.

12 Evaluation

Metrics: 183 tests, 100

13 Current Limitations

Key limitations: offline support (online-only dashboards), no ML-based fraud detection (deterministic rules only), limited Arabic localization, and manual data import for cadastral integration. These are addressed in future work.

14 Appendix: Glossary of Terms

TIB Taxe sur les Immeubles Bâtis (property tax).

TTNB Taxe sur les Terrains Non Bâtis (land tax).

Delegation Administrative sub-division within a governorate.

Municipal Service Config Table defining which communal services (lighting, water, waste) serve each locality.

Participatory Budgeting Citizen voting mechanism for allocating municipal funds.

This glossary assists ministry readers unfamiliar with local terminology.

15 Future Work

(1) Mobile PWA, (2) ML fraud detection, (3) Arabic localization, (4) Payment gateways, (5) Cadastral sync, (6) Analytics, (7) SMS notifications, (8) Blockchain audit.

16 Webography

This section enumerates the primary online resources, technical documentation, and legal references consulted during the design, implementation, and evaluation of TUNAX. Each entry includes a URL (when publicly accessible) and a brief annotation explaining its relevance to the project.

16.1 Legal and Regulatory Sources

1. **Code de la Fiscalité Locale 2025**
   Ministère des Finances de la République Tunisienne
   `http://www.finances.gov.tn/`
   Official legislative framework governing TIB (Articles 1–34) and TTNB (Articles 32–33), including exemption rules, service-based taxation, and dispute procedures. The platform's calculator logic and schema constraints derive directly from these articles.

2. **Décret n° 2017-396 du 9 mars 2017**
   Journal Officiel de la République Tunisienne
   Establishes the urban zone classification (haute densité, densité moyenne, faible densité, périphérique) and corresponding per-square-meter tariffs for TTNB computation. The `Land` model's `urban_zone` field and the `TariffService` class directly implement this decree.

3. **Office de la Topographie et de la Cartographie (OTC)**
   `http://www.otc.nat.tn/`
   National cadastral authority providing official boundary data and property identifiers. Future integration would sync declared property coordinates with OTC records to reduce fraud and verification overhead.

16.2 Backend Frameworks and Libraries

4. **Flask Documentation**
   `https://flask.palletsprojects.com/`
   Official documentation for Flask 3.0.2, the micro web framework powering TUNAX's backend. Consulted for application factory patterns, blueprint registration, and request lifecycle management.

5. **Flask-SMOREST Documentation**
   `https://flask-smorest.readthedocs.io/`
   OpenAPI 3.0 REST framework built atop Flask. TUNAX uses Flask-SMOREST to auto-generate Swagger UI, validate request/response payloads via Marshmallow, and expose RESTful endpoints consistently.

6. **SQLAlchemy Documentation**
   `https://docs.sqlalchemy.org/`
   Version 2.0.23 provides the ORM layer for TUNAX models. Consulted extensively for relationship patterns, polymorphic associations (e.g., `Tax` linking `Property` or `Land`), and declarative mapping conventions.

7. **Alembic Documentation**
   `https://alembic.sqlalchemy.org/`
   Database migration framework for SQLAlchemy. TUNAX's `migrations/versions/` directory contains auto-generated Alembic scripts that incrementally evolve the schema.

8. **Marshmallow Documentation**
   `https://marshmallow.readthedocs.io/`
   Version 3.24.1 powers the `schemas/` module, enabling validation, serialization, and deserialization. Each API endpoint declares input/output schemas to enforce data contracts and generate OpenAPI specifications.

9. **Flask-JWT-Extended Documentation**
   `https://flask-jwt-extended.readthedocs.io/`
   JWT authentication extension (version 4.5.3) providing token issuance, refresh, and blacklist capabilities. TUNAX's `auth.py` resource and `role_required` decorators depend on this library.

16.3 Security and Authentication

10. **Werkzeug Security Utilities**
    `https://werkzeug.palletsprojects.com/en/stable/utils/#module-werkzeug.security`
    Password hashing via `pbkdf2:sha256` is implemented in the `User` model using Werkzeug's `generate_password_hash` and `check_password_hash` functions.

11. **Flask-Limiter Documentation**
    `https://flask-limiter.readthedocs.io/`
    Rate limiting extension that protects authentication endpoints (5 requests/minute) and general API routes (50 requests/hour) from abuse. TUNAX migrated to Redis-backed storage (January 2026) via `RATELIMIT_STORAGE_URI` for distributed rate limit enforcement across multiple backend workers in production deployments.

12. **PyOTP (Python One-Time Password Library)**
    `https://pyauth.github.io/pyotp/`
    Implements TOTP-based two-factor authentication. TUNAX's `two_factor.py` model and `/api/2fa/*` endpoints rely on PyOTP for code generation and verification.

13. **RFC 7519: JSON Web Token (JWT)**
    `https://tools.ietf.org/html/rfc7519`
    Standardizes JWT structure and claims. TUNAX adheres to this specification when encoding `role` and `commune_id` claims into access and refresh tokens.

16.4 Database Systems

14. **PostgreSQL 15 Documentation**
    `https://www.postgresql.org/docs/15/`
    Production-grade relational database. TUNAX uses PostgreSQL for JSON columns,

CITEXT for case-insensitive email matching, and robust ACID guarantees in transaction-heavy workflows (e.g., payment processing).

15. **SQLite Documentation**
`https://www.sqlite.org/docs.html`
Lightweight file-based database used in development (`tunax.db`). Supports rapid schema iteration and zero-configuration deployment for local testing.

16.5 Geolocation and Geographic Services

16. **Nominatim (OpenStreetMap Geocoding)**
`https://nominatim.org/release-docs/latest/`
API endpoint: `https://nominatim.openstreetmap.org/`
Converts declared property addresses to GPS coordinates (latitude, longitude). TUNAX's `geo.py` utility invokes Nominatim via the Geopy library, ensuring geospatial validation of TIB and TTNB declarations.

17. **Geopy Documentation**
`https://geopy.readthedocs.io/`
Python library wrapping multiple geocoding services. TUNAX configures Geopy to query Nominatim while respecting rate limits and handling coordinate projection.

18. **OpenStreetMap: Tunisia Administrative Boundaries**
`https://www.openstreetmap.org/relation/192757`
Community-maintained dataset of Tunisia's 264 communes. The `seed_data/communes_tn.csv` file derives from OpenStreetMap extracts and provides municipality names, postal codes, and geographic extents.

16.6 Containerization and Deployment

19. **Docker Documentation**
`https://docs.docker.com/`
Container platform documentation. TUNAX's `docker/Dockerfile` packages the Flask app, dependencies, and entrypoint into a portable image suitable for cloud or on-premise deployment.

20. **Docker Compose Documentation**
`https://docs.docker.com/compose/`
Multi-container orchestration tool. The `docker-compose.yml` file defines services for the backend (Flask), database (PostgreSQL), and frontend (Nginx), enabling one-command environment provisioning.

21. **Nginx Documentation**
`https://nginx.org/en/docs/`
Web server and reverse proxy. The `nginx.conf` file routes frontend static assets and proxies API requests to the Flask backend, decoupling client delivery from application logic.

16.7 API Standards and Specifications

22. **OpenAPI Specification 3.0**
    `https://swagger.io/specification/`
    Industry-standard API description format. Flask-SMOREST auto-generates an OpenAPI document at `/api/swagger.json`, which drives Swagger UI and client SDK generation.

23. **RESTful API Design Best Practices**
    `https://restfulapi.net/`
    Resource-oriented architecture guide. TUNAX's endpoints follow REST principles: nouns for resources (`/properties`, `/lands`), HTTP verbs for actions (GET, POST, PUT, PATCH, DELETE), and HATEOAS links for discoverability.

16.8 Testing and Quality Assurance

24. **Insomnia REST Client**
    `https://insomnia.rest/`
    API testing tool. The `tests/insomnia_collection.json` file contains pre-configured scenarios for authentication, property declaration, payment processing, and dispute workflows, enabling regression testing.

25. **pytest Documentation**
    `https://docs.pytest.org/`
    Python testing framework recommended for future unit and integration test coverage. Would complement the existing Insomnia collection with programmatic assertions.

26. **pytest-flask Documentation**
    `https://pytest-flask.readthedocs.io/`
    Flask-specific fixtures and helpers for pytest. Simplifies creation of test clients, database fixtures, and request contexts.

27. **Coverage.py Documentation**
    `https://coverage.readthedocs.io/`
    Code coverage measurement tool. Future integration would quantify line and branch coverage across models, resources, and utilities.

16.9 Observability and Monitoring (Future)

28. **Sentry Error Tracking**
    `https://docs.sentry.io/`
    Real-time error monitoring and alerting. Integration would capture uncaught exceptions, performance regressions, and user-facing errors in production.

29. **Prometheus Monitoring**
    `https://prometheus.io/docs/`
    Time-series metrics collection. Future instrumentation could expose Flask endpoint latencies, database query durations, and custom business metrics (e.g., declarations per day).

30. **Grafana Dashboards**
`https://grafana.com/docs/`
Visualization layer for Prometheus metrics. Municipal admins could monitor system health, user activity, and revenue trends via real-time dashboards.

16.10 Additional Libraries and Utilities

31. **Flask-CORS Documentation**
`https://flask-cors.readthedocs.io/`
Cross-Origin Resource Sharing middleware. Enables frontend (served via Nginx on port 80) to communicate with backend (Flask on port 5000) without browser security errors.

32. **python-dotenv Documentation**
`https://pypi.org/project/python-dotenv/`
Loads environment variables from `.env` files. TUNAX uses dotenv to configure database URLs, JWT secrets, and rate limits without hardcoding sensitive values.

33. **PyYAML Documentation**
`https://pyyaml.org/wiki/PyYAMLDocumentation`
YAML parser for Python. The `tariffs_2025.yaml` file defines service-based rates, exemption thresholds, and penalties in a human-readable format loaded at runtime.

16.11 Payment Gateways (Planned Integration)

34. **Clictopay (Tunisia)**
`https://www.clictopay.com.tn/`
Tunisian online payment gateway supporting credit/debit cards and e-dinars. Future integration would enable citizens to settle tax bills directly through the TUNAX frontend.

35. **SMT (Société Monétique Tunisie)**
`https://www.smt.tn/`
National payment card network. Integration would allow municipal finance offices to reconcile electronic payments with bank transfers and card settlements.

16.12 Version Control and Collaboration

36. **Git Documentation**
`https://git-scm.com/doc`
Distributed version control system. TUNAX's repository uses Git for branching, merging, and audit trails. The commit history documents architectural evolution and AI-assisted contributions.

37. **GitHub Copilot**
`https://github.com/features/copilot`
AI-powered code completion assistant. As documented in Section **??**, Copilot accelerated blueprint creation, refactoring, and documentation drafting while maintainers retained oversight through code reviews and test validation.

17 Conclusion

This academic report documented the technical and contextual dimensions of TUNAX, a comprehensive Tunisian municipal tax management platform. By uniting legal requirements, robust architecture, and user-centric dashboards, the system demonstrates how software engineering can modernize public finance. The platform successfully operationalizes the 2025 Code de la Fiscalité Locale through Marshmallow schemas, SQLAlchemy models, and Flask-Smorest blueprints, providing a reproducible reference implementation for municipalities nationwide. Continued collaboration with municipalities, security audits, and user research will ensure the platform scales sustainably across Tunisia's diverse regions.