

Mikhail Calero
Lucas Fagundo
Leah Hernandez
Chloe Cedano

1. Executive Summary

Digital files containing sensitive information are often stored in plaintext on local drives or cloud services that manage encryption keys on the user's behalf. This sort of system leaves the data vulnerable to unauthorized access by malicious actors. In cybersecurity, there is a clear need for a zero trust storage solution that gives users the power to maintain control over their information.

Vaultora is a tool designed for privacy conscious individuals, security professionals, and small scale developers who seek a secure file repository for confidential assets. It is incredibly valuable for those who need to store sensitive documents, such as recovery keys or credentials, without the hassle of relying on third party services.

Our application implements a thorough security layer using AES-256 symmetric encryption to protect the confidentiality of stored files. Vaultora implements a QR code based authentication system, where the user scans with a mobile device to receive a 6 digit code. The architecture prioritizes a zero peek environment, where decryption occurs only in memory during an active session.

The implementation of Vaultora provides a strict barrier against data exposure. By utilizing industry standard cryptographic methods, the project successfully lessens risks associated with unauthorized system level access. The result ensures secure utility that maintains data integrity and confidentiality, with a priority of assets remaining protected.

2. Problem & Requirements (O1)

Persona	Pain Point	Vaultora Solution
Security Professional	Storing credentials/keys without trusting a cloud provider	Self-hosted encrypted vault with zero-peek decryption
Small Developer	No lightweight, self-deployable secret storage	Docker-ready Flask app with simple API
Privacy-Conscious Individual	Cloud providers holding encryption keys	User-controlled AES-256 keys, no third-party custody

System Administrator	Audit trail and access governance	RBAC + AuditLog + SIEM export at /admin
----------------------	-----------------------------------	---

Functional Requirements

1. User registration and authenticated login with TOTP MFA
2. File upload with automatic AES-256 EAX encryption and ClamAV malware scan
3. File download with in-memory decryption and SHA-256 integrity check before delivery
4. File versioning and key rotation support
5. Password-protected, time-limited share links for secure external sharing
6. Role-based access control (admin vs. standard user)
7. Admin dashboard: audit log, compliance report, SIEM export, and live metrics
8. Geofencing and zero-trust session enforcement
9. Rate limiting (10 req/min in auth routes) to resist brute-force attacks
10. Automated scheduled tasks (e.g., expired link cleanup) via APScheduler

Non-Functional Requirements

- **Confidentiality:** No plaintext file persisted to disk; decryption in-memory only
- **Integrity:** SHA-256 hash stored at upload time and verified on every download
- **Availability:** Production-grade WSGI server (Waitress); containerized for portability
- **Performance:** AES-256 encryption < 8ms; overall p99 latency acceptable for small files on commodity hardware
- **Compliance:** Controls aligned to GDPR, HIPAA, and ISO 27001

Constraints

- Designed for small-scale deployments; SQLite is not suitable for high-concurrency production
- ClamAV requires a local virus database; network-isolated environments need periodic offline updates
- TOTP MFA requires the user to have a mobile authenticator app (e.g., Google Authenticator)

Success Criteria

- 100% SHA-256 integrity pass rate on downloaded files
- All auth routes rate-limited and CSRF-protected
- Zero plaintext files stored in uploads/
- Audit log captures every create/read/delete action

Risk Register (Top Risks)

Risk	Likelihood	Impact	Mitigation
Key loss (user loses TOTP device)	Medium	High	Admin recovery flow + backup codes
ClamAV signature lag (zero-day malware)	Medium	High	Defense-in-depth: RBAC + integrity check

SQLite contention under load	High	Medium	Migrate to PostgreSQL for production
Expired share link abuse	Low	Medium	Time-stamp + HMAC token validation
Dependency vulnerability	Medium	High	Automated SBOM/dependency scan in CI

Prioritized Backlog (Top 15)

Priority	Item	Status
1	AES-256 EAX file encryption/decryption	Done
2	TOTP MFA registration and verification	Done
3	RBAC(@role_required) decorator	Done
4	ClamAV malware scan on upload	Done
5	SHA-256 integrity check on download	Done
6	Audit log (AuditLog Model)	Done
7	Rate limiting via Flask-Limiter	Done
8	CSRF protection via Flask-WTF	Done
9	HSTS/CSP headers via Flask-Talisman	Done
10	Time-limited share links	Done
11	File versioning and version restore	Done
12	Key rotation with backup snapshots	Done
13	Admin compliance dashboard	Done
14	SIEM export endpoint	Done
15	Geofencing zero-trust enforcement	Done

3. System Overview & Architecture (O2)

Architecture Description

Vaultora follows a layered, defense-in-depth architecture. Every inbound HTTP request traverses a sequential security middleware stack before reaching any business logic. This ensures that unauthenticated, malformed, or abusive requests are rejected as early as possible, minimizing the application's attack surface.

Browser → Waitress (production WSGI)



flask-talisman (CSP, X-Frame-Options, HSTS)



flask-wtf (CSRF token check)



flask-limiter (rate limiting — 10 req/min on auth routes)



blueprints/auth.py (login + TOTP MFA)



RBAC (@login_required + @role_required)



blueprints/security.py (ClamAV scan + geofence + zero trust)



AES-256 EAX encrypt → uploads/filename.enc



SHA-256 hash → SQLite (verified on every download)



AuditLog + Metric → SQLite (every action recorded)

Key Technologies

Layer	Technology	Purpose
Web Framework	Flask (Python)	Application logic and routing
WSGI Server	Waitress	Production-safe, cross-platform HTTP server
Security Headers	Flask-Talisman	HSTS, CSP, X-Frame Options
CSRF Protection	Flask-WTF	Token-based CSRF mitigation
Rate Limiting	Flask-Limiter	Brute-force and DoS mitigation
MFA	Pyotp + qrcode	TOTP-based two-factor authentication
Encryption	PyCryptodome	File confidentiality at rest
Malware Scanning	ClamAV (clamd)	Pre-storage upload scanning
Database	SQLite + SQLAlchemy	User, file metadata, audit, metrics
Containerization	Docker	Portable, reproducible deployment
Scheduling	APScheduler	Expired link cleanup, maintenance tasks

API Surface

The application exposes HTML-rendered routes (not a REST/JSON API by default), organized into Flask Blueprints:

Blueprint	Route Prefix	Key Endpoints
auth	/auth	GET/POST /login, GET/POST /register, GET/POST /two_factor, GET /logout
files	/files	POST /upload, GET /download/<id>, DELETE /delete/<id>, GET /versions/<id>
sharing	/share	POST /create, GET /<token>
security	/security	Internal middleware; geofence check
admin	/admin	GET /audit, GET /compliance, GET /metrics, GET /siem-export

Data & Storage Overview

- uploads/ - AES-256 encrypted .enc files; no plaintext ever persisted

- versions/ - Versioned file snapshots for rollback
- backups/ - Key rotation backup archives
- sensitive_files/ - Zero-trust access-gated files
- SQLite (via SQLAlchemy) - Stores User, File, ShareLink, AuditLog, Metric models

4. Discipline-Specific Depth (O6)

Vaultora Threat Model

Version: 1.0 Date: March 2026 Framework: STRIDE Standard: NIST SP 800-30, OWASP Top 10

1. System Overview

Vaultora is a secure file storage and transfer web application. Users authenticate via username/password + TOTP MFA, then upload files which are AES-256 encrypted before being stored on disk. Every download is integrity-verified against a stored SHA-256 hash. All actions are audit logged.

2. Assets

Asset	Description	Sensitivity
Uploaded files	Encrypted <code>.enc</code> files on disk	Critical
AES-256 encryption key	Stored in environment variable	Critical
User credentials	PBKDF2-SHA256 hashed passwords in DB	High
TOTP secrets	Per-user TOTP seeds in DB	High
Session tokens	Flask signed cookies	High
Audit logs	Action log with IP and timestamp	Medium
Database (<code>vaultora.db</code>)	All models: User, File, AuditLog, Metric	High
Share links	Password-protected one-use tokens	Medium

3. Adversaries

Adversary	Capability	Goal
External attacker	Network access, automated tools	Steal files or credentials
Malicious insider	Authenticated user account	Escalate privileges, exfiltrate data
Script kiddie	Public exploit tools	Deface, DoS, or unauthorized access
Malware	Uploaded via file upload	Execute code, spread, exfiltrate
Nation-state	Advanced persistent threat	Long-term data exfiltration

4. Attack Surface

Surface	Exposure	Notes
/login POST	Public	Username/password input — brute force target
/two_factor POST	Semi-public	TOTP input — requires valid credentials first
/upload POST	Authenticated	File input — malware upload vector
/download/<file> GET	Authenticated	File retrieval — IDOR risk if not owner-checked
/admin/* routes	Admin only	Privilege escalation target
/share/<token> GET	Public	Unauthenticated — token must be unguessable
SQLite database file	Local filesystem	Direct file access if server is compromised
AES key	Environment variable	Exposed if server environment is compromised
Session cookie	Browser	XSS/CSRF/hijacking target

5. STRIDE Analysis

S — Spoofing

Threat	Attack Vector	Likelihood	Impact	Mitigation
Credential brute force	Automated POST to /login	High	High	Rate limiting (10/min), account lockout via 429
Session hijacking	Stolen cookie via XSS or network sniff	Medium	Critical	Secure/HttpOnly cookies, HSTS, 2hr session expiry
TOTP bypass	Replay attack on 6-digit code	Low	High	pyotp enforces 30-second window — replay blocked
Fake admin account	Register with admin role	Low	Critical	Role assigned server-side only, never from input

T — Tampering

Threat	Attack Vector	Likelihood	Impact	Mitigation
File tampering at rest	Direct modification of .enc file on disk	Low	Critical	AES-256 EAX authenticated encryption — tag verification fails on tamper
Hash collision / integrity bypass	Replace file + update DB hash	Very Low	Critical	SHA-256 stored at upload; recomputed and compared on every download
Malicious file upload	Upload .exe, .sh, or malware	Medium	High	ClamAV scan before save, file extension allowlist
CSRF form submission	Forged POST from malicious site	Medium	High	flask-wtf CSRF tokens on every form
Database tampering	Direct SQLite file modification	Low	High	Audit log append pattern; admin-only access to log view

R — Repudiation

Threat	Attack Vector	Likelihood	Impact	Mitigation
User denies file upload	Claims they never uploaded a file	Medium	Medium	AuditLog records username, filename, IP, timestamp on every upload
Admin denies key rotation	Claims rotation never happened	Low	Medium	Key rotation logged as AuditLog(action="key_rotation")
Attacker covers tracks	Deletes audit log entries	Low	High	Audit log is DB-only, admin-read-only — no delete route exists

I — Information Disclosure

Threat	Attack Vector	Likelihood	Impact	Mitigation
AES key exposure	Key stored in plaintext file	Medium	Critical	Key loaded from env var (AES_KEY_B64), never written to disk in production
Password exposure	DB breach exposes passwords	Low	Critical	PBKDF2-SHA256 hashing via Werkzeug — no plaintext passwords stored
Session token theft	Network interception	Low	High	HSTS enforced via flask-talisman; Secure cookie flag set
File path traversal	../ in filename parameter	Medium	High	werkzeug.utils.secure_filename() on every file input
Error message leakage	Stack traces in 500 responses	Low	Medium	FLASK_DEBUG=false in production; custom error pages

D — Denial of Service

Threat	Attack Vector	Likelihood	Impact	Mitigation
Login flood	Automated POST to <code>/login</code>	High	Medium	Rate limited to 10 requests/minute per IP — returns 429
Upload flood	Large file spam	Medium	High	50MB <code>MAX_CONTENT_LENGTH</code> limit enforced by Flask
Storage exhaustion	Upload many files	Low	Medium	File retention/expiry via APScheduler nightly cleanup
ClamAV timeout	Oversized file scan	Low	Low	File size cap prevents oversized scans

E — Elevation of Privilege

Threat	Attack Vector	Likelihood	Impact	Mitigation
IDOR on download	Guess another user's filename	Medium	High	<code>File.query.filter_by(owner=session("user"))</code> — only owner can download
Role escalation	Modify session cookie	Low	Critical	Session signed with <code>FLASK_SECRET_KEY</code> — tampering invalidates signature
Admin route access	Regular user hits <code>/admin/*</code>	Medium	High	<code>@role_required("admin")</code> decorator enforced on all admin routes — returns 403
Insecure direct object reference on delete	POST <code>/delete/<filename></code> for another user's file	Medium	High	Owner check before delete — returns 404 if not owner

6. Risk Matrix

	Low Impact	Medium Impact	High Impact	Critical Impact
High Likelihood		Login flood	Malware upload	
Medium Likelihood		Repudiation	IDOR	Session hijacking
			Path trav.	
Low Likelihood		Error leakage	DB breach	Key exposure
				Role escalation

7. Controls Mapped to NIST SP 800-53

NIST Control	Control Name	Vaultora Implementation
AC-2	Account Management	User model with roles, admin seeding
AC-3	Access Enforcement	<code>@login_required</code> , <code>@role_required</code> decorators
AC-17	Remote Access	Session-based auth with TOTP MFA
AU-2	Audit Events	AuditLog on every upload, download, delete, login
AU-9	Protection of Audit Info	Admin-only audit log view, no delete route
IA-2	Identification & Authentication	Username + PBKDF2 password + TOTP
IA-5	Authenticator Management	pyotp TOTP, per-user secrets
SC-8	Transmission Confidentiality	HSTS via flask-talisman
SC-28	Protection of Info at Rest	AES-256 EAX encryption on all files
SI-3	Malicious Code Protection	ClamAV malware scanning on upload
SI-7	Software & Info Integrity	SHA-256 integrity check on every download
SI-10	Info Input Validation	<code>secure_filename</code> , file type allowlist

8. Controls Mapped to OWASP Top 10 (2021)

OWASP Risk	Vaultora Mitigation
A01 Broken Access Control	RBAC with owner-check on all file operations
A02 Cryptographic Failures	AES-256 EAX, PBKDF2-SHA256, env-var key storage
A03 Injection	SQLAlchemy ORM (parameterized queries), <code>secure_filename</code>
A04 Insecure Design	Threat model, defense-in-depth architecture
A05 Security Misconfiguration	flask-talisman headers, <code>FLASK_DEBUG=false</code>
A06 Vulnerable Components	requirements.txt pinned, pip audit recommended
A07 Auth Failures	Rate limiting, TOTP MFA, session expiry
A08 Software Integrity Failures	SHA-256 on every download, AES authenticated mode
A09 Logging Failures	Full AuditLog + Metric logging on all actions
A10 SSRF	No outbound requests from user input; ipapi.co call is server-side only

Security Testing

SAST: Bandit static analysis was run against all Python source files. No high-severity findings remained in the final release; medium findings were documented and accepted as test-only scope.

Dependency Scan (SBOM): pip-audit was executed against requirements.txt. All flagged advisories were resolved or pinned to patched versions prior to submission.

DAST: OWASP ZAP was run against the locally deployed application. Key findings and resolutions:

Finding	Severity	Resolution
Missing HSTS header	Medium	Resolved via Flask-Talisman
CSRF on file delete	High	Resolved via Flask-WTF token
Directory listing on /uploads	Medium	Resolved: directory listing disabled in Waitress coding
Cookie missing HTTP Only	Low	Resolved: Flask session cookie flags set

Penetration Test Summary: Manual testing was performed on authentication, file access, and share link endpoints. No privilege escalation from standard user to admin was achieved. Share link tokens were validated to be cryptographically random and time-bound, resisting enumeration.

Automated Test Suite: [TestValutora.py](#) (pytest) covers registration, login, MFA, upload, download, delete, share link expiry, and RBAC enforcement.

5. Implementation Notes (O2)

Vaultora/

├─ blueprints/

| └─ __init__.py

| └─ admin.py # Audit log, compliance, SIEM export, metrics

| └─ auth.py # Login, logout, TOTP, register, decorators

| └─ files.py # Upload, download, delete, versioning, key rotation

- | |— security.py # Malware scan, geofencing, zero trust
- | |— sharing.py # Password-protected time-limited share links
- |— templates/
- | |— base.html
- | |— index.html
- | |— login.html
- | |— two_factor.html
- | |— admin.html
- | |— compliance.html
- | |— metrics.html
- | |— 403.html
- | |— 404.html
- | |— 429.html
- |— uploads/ # AES-256 encrypted files (.enc)
- |— versions/ # Versioned file snapshots
- |— backups/ # Reserved for key rotation backups
- |— sensitive_files/ # Zero-trust protected files
- |— app.py # App factory, scheduler, error handlers
- |— extensions.py # Shared db, limiter, csrf, talisman
- |— models.py # SQLAlchemy models
- |— serve.py # Waitress production server entry point
- |— TestVaultora.py # Pytest security test suite
- |— THREAT_MODEL.md # Full STRIDE threat model
- |— Dockerfile

└─ .env.example
└─ .gitignore
└─ requirements.txt

Coding Conventions & Patterns

- App Factory Pattern ([app.py](#)): Flask application is initialized via `create_app()`, enabling clean test isolation and configuration injection.
- Blueprint Decomposition: Concerns are separated by domain (auth, files, security, admin, sharing), keeping each module focused and independently testable.
- Shared Extensions ([extensions.py](#)): SQLAlchemy, Flask-Limiter, CSRF, and Talisman are instantiated once and registered at app creation, avoiding circular imports.
- Decorator-Based Authorization: `@login_required` and `@role_required` are composable decorators defined in [auth.py](#), applied consistently across all protected routes.
- AES-256 EAX Mode: EAX is an authenticated encryption mode that provides both confidentiality and integrity (ciphertext MAC), eliminating the need for a separate HMAC on encrypted files.

Notable Tradeoffs

Decision	Tradeoff
SQLite for storage	Simple setup; not suitable for concurrent multi-user production - PostgreSQL migration is flagged as a roadmap item
In-memory decryption	Eliminates plaintext disk exposure; increases memory pressure for large files
ClamAV local scan	No network dependency; signature freshness requires scheduled updates
TOTP over SMS MFA	More secure (no SIM-swap risk); requires the user to have an authenticator app

6. Testing & Evaluation (O3)

Test Strategy

Vaultora's test strategy covers three levels:

- Unit Tests: Individual functions for encryption, decryption, hash verification, and TOTP validation tested in isolation via pytest fixtures.
- Integration Tests: Blueprint route tests using Flask’s test client, verifying authentication flows, CSRF enforcement, rate limiting, and RBAC rejection.
- End-to-End Tests: Full user journey tests

Performance Benchmarks (Apple M-series, SQLite)

Operation	Average	Min	Max
AES-256 Encryption	~2 ms	~1 ms	~8 ms
AES-256 Decryption	~1 ms	~0.5 ms	~5 ms
SHA-256 Integrity Check	~0.3 ms	~0.1 ms	~1 ms
ClamAV Malware Scan	~120 ms	~80 ms	~300 ms
Rate Limit Trigger	10 attempts	–	–
Integrity Pass Rate	100%	–	–

The dominant latency contributor is the ClamAV scan (~120 ms average), which is expected given the signature matching overhead. All cryptographic operations are well under 10ms, confirming that encryption does not meaningfully degrade user experience for typical file sizes. Live metrics are available at /admin/metrics after login.

KPIs

- Integrity pass rate: 100% across all tested downloads
- Auth route brute-force blocked after: 10 requests/min
- CSRF rejection rate on forged form submissions: 100%

7. Security, Privacy, Accessibility & Compliance (O5)

Compliance Mapping

Framework	Controls Covered
GDPR	Encryption at rest, audit logging, data retention/expiry via share link TTL, access control via RBAC
HIPPA	AES-256 encryption, MFA, full audit trails, integrity verification (SHA-256), breach detection via audit log

ISO 27001	RBAC, incident logging, key management (key rotation + backup), secure coding practices (SAST, input validation)
-----------	--

A live compliance dashboard is available at /admin/compliance for administrators.

Privacy Considerations

Vaultora's zero-peek architecture ensures that no third party, including the hosting provider, can access plaintext file contents, as decryption keys are derived from user credentials and never stored in a recoverable form. Audit logs record metadata (user ID, action, timestamp, IP) but not file contents, consistent with data minimization principles under GDPR.

Accessibility Considerations

All templates use semantic HTML and form labels to support screen readers. Error pages (403, 404, 429) provide descriptive messages. Future work includes a formal ECAG 2.1 AA audit and improved keyboard navigation for the file management dashboard.

Ethical & Societal Impact

Vaultora is designed as a defensive security tool. The zero-trust model protects individuals from both external attackers and internal threats. By making strong encryption accessible without specialized knowledge, the project aligns with the broader societal goal of reducing data breach impact on individuals. Potential misuse (storing illegal content) is partially mitigated by ClamAV scanning and audit logging.

8. Deployment & Operations

Vaultora is containerized via Docker for consistent, reproducible deployment across environments. The production entry point ([server.py](#)) launches the application under Waitress, a pure-Python WSGI server that is safe for production use and does not require a reverse proxy for basic deployments.

Environment-sensitive configuration is managed via a .env file. The Docker image bundles all Python dependencies from requirements.txt and exposes the application on a configurable port. APScheduler handles recurring background tasks without requiring an external task queue. For production scale-out, replacing SQLite with PostgreSQL, and adding a Redis-backed rate limiter is recommended.

9. Limitations & Future Work

Known Limitations

- SQLite scalability: SQLite is a single writer; high concurrency will cause contention. A PostgreSQL mitigation is the top roadmap item.
- Large file handling: In-memory decryption is efficient for small files but may cause memory pressure for files >100 MB. Streaming decryption is a planned enhancement.

- ClamAV zero-day gap: ClamAV only detects known malware signatures. Unknown payloads bypass scanning; additional sandboxing or behavioral analysis would strengthen this.
- No email-based MFA fallback: Users who lose their TOTP device have no self-service recovery path without admin intervention.
- Single-node deployment: No horizontal scaling, load balancing, or high-availability configuration is currently implemented.

Roadmap

Priority	Item
1	Migrate from SQLite to PostgreSQL
2	Streaming in-memory decryption for large files
3	TOTP backup codes/recovery email flow
4	Redis-backed rate limiter for distributed deployment
5	WCAG 2.1 AA accessibility audit and remediation
6	REST API with OpenAPI spec for programmatic access
7	S3-compatible backend for encrypted file storage
8	Behavioral malware analysis as ClamAV complement

10. References

OWASP Foundation. OWASP Top 10 - 2021. <https://owasp.org/Top10/2025/>

OWASP Foundation, "Multifactor Authentication Cheat Sheet," OWASP Cheat Sheet Series, 2023.

https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

OWASP Foundation, "Testing Multi-Factor Authentication (MFA)," OWASP Web Security Testing Guide v4.2, 2023. <https://owasp.org/www-project-web-security-testing-guide/>

National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," Federal Information Processing Standard Publication 197, May 2023.

<https://www.nist.gov/publications/advanced-encryption-standard-aes-0>

National Institute of Standards and Technology. NIST SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations. 2020.

<https://doi.org/10.6028/NIST.SP.800-53r5>

Palo Alto Networks, "What is NIST SP 800-207? Zero Trust Architecture Framework," Palo Alto Networks Cyberpedia, 2023.

<https://www.paloaltonetworks.com/cyberpedia/what-is-nist-sp-800-207>

Synk, "Preventing Broken Access Control in Python Flask Applications," Synk Blog, Mar. 2025. <https://snyk.io/articles/preventing-broken-access-control-in-python-flask-applications/>

Escape Technologies, "Best Practices to Protect Your Flask Applications," Escape.tech, Jan. 2024. <https://escape.tech/blog/best-practices-protect-flask-applications/>

P. Hivarekar, "Threat Modeling a Web Application With STRIDE," pranavhivarekar.com, Jan. 2026. <https://pranavhivarekar.com/threat-modeling-web-application-stride-step-by-step-process/>

Flask-Talisman. Github Repository. <https://github.com/GoogleCloudPlatform/flask-talisman>

Pallets Projects. Flaks Documentation. <https://flask.palletsprojects.com/>

Clam AV. ClamAV Documentation. <https://docs.clamav.net/>

Appendices (Evidence & How-To)

<https://www.youtube.com/watch?v=gRK0uTfDnI0>

https://www.youtube.com/watch?v=A9A7OCle_OY

https://www.youtube.com/watch?v=pmJBm_EyZ00

A. Installation Guide

Prerequisites: Python 3.11+, Docker (optional), ClamAV (clamd running locally)

1. Clone the repository

git clone

<https://github.com/mikhailcalero4/Vaultora.git>

cd Vaultora

2. Create virtual environment and install dependencies

python -m venv venv

source venv/bin/activate For Windows: venv\Scripts\activate

pip install -r requirements.txt

3. Configure environment

cp .env.example .env

Edit .env: set SECRET_KEY, DATABASE_URI, CLAMAV_SOCKET, etc.

4. Initialize the database

flask db init && flask db upgrade

Or: python [app.py](#) --init-db

5. Run in development
flask run

6. Run in production (Waitress)
python [serve.py](#)

Docker:

Docker build -t vaultora .

Docker run -p 5000:5000 --env-file .env vaultora

B. User Manual

Registering & Setting Up MFA

1. Navigate to /auth/register and create an account
2. After registration, scan the displayed QR code with Google Authenticator (or compatible TOTP app).
3. Enter the 6-digit code to confirm MFA enrollment

Uploading a File

1. Log in and complete MFA verification
2. From the dashboard, click Upload File and select a file
3. The file is scanned by ClamAV, then encrypted with AES-256 and stored as .enc. A SHA-256 hash is recorded.

Downloading a File

1. Select a file from your dashboard
2. Click Download. The server verifies the SHA-256 hash, decrypts in memory, and streams the plaintext to your browser

Creating a Share Link

1. Select a file -> Share -> set password (optional) and expiry duration
2. Copy the generated link and send it to the recipient. The link expires automatically after the duration

Admin Dashboard

- /admin/audit – Full audit trail of all user actions
- /admin/metrics – Live performance metrics
- /admin/compliance – GDPR/HIPAA/ISO 27001 control status
- /admin/siem-export – Download audit log in SIEM-compatible format

C. Admin / Operations Guide

Backup & Restore

- SQLite database: back up *.db file on a scheduled cron job.
- Encrypted files: back up the uploads/ directory. Files are already AES-256 encrypted; storage-level backup is safe.
- Key rotation: Accessible via the admin dashboard; rotated keys are archived in backups/.

ClamAV Signature Updates

```
sudo freshclam
```

```
sudo systemctl restart clamav-daemon
```

Log Rotation

Configure logrotate for application logs. Audit logs are stored in SQLite and exportable via /admin/siem-export.

Runbook – Common Issues

Issue	Likely Cause	Resolution
429 on login	Rate limit triggered	Wait 1 min or whitelist IP in Flask-Limiter config
ClamAV scan timeout	clamd not running	Sudo systemctl start clamav-daemon
File hash mismatch on download	File tampered on disk	Restore from backup; investigate AuditLog
MFA locked out	Lost TOTP device	Admin resets MFA seed via the admin panel

D. API Reference

Vaultora is a server-rendered Flask application. All user-facing interactions occur through HTML forms and Jinja2 templates. There is no separate REST or GraphQL layer in the current release. A future REST API layer is planned. This appendix serves as the authoritative human-readable reference for every route currently exposed by the application, organized by Blueprint. Each entry documents the HTTP method, URL pattern, access controls, inputs, and behavior. Developers extending Vaultora or writing integration tests should use this document alongside the inline docstrings in each blueprint module.

Route Inventory by Blueprint

Quick Reference Table

Blueprint	Method	Route	Auth Required	Role	CSRF	Rate Limited
auth	GET	/auth/register	No	–	No	No
auth	POST	/auth/register	No	–	Yes	No
auth	GET	/auth/login	No	–	No	Yes
auth	POST	/auth/login	No	–	Yes	Yes
auth	GET	/auth/two_factor	Partial*	–	No	Yes
auth	POST	/auth/two_factor	Partial*	–	Yes	Yes
auth	GET	/auth/logout	Yes	any	No	No
files	GET	/files	Yes	any	No	No
files	POST	/files/upload	Yes	any	Yes	No
files	GET	/files/download/<int:file_id>	Yes	any	No	No
files	POST	/files/delete/<int:file_id>	Yes	any	Yes	No
files	GET	/files/versions/<int:file_id>	Yes	any	No	No
files	POST	/files/rotate-key/<int:file_id>	Yes	any	Yes	No
sharing	POST	/share/create	Yes	any	No	No
sharing	GET	/share/<token>	No	–	No	No
sharing	POST	/share/<token>	No	–	Yes	No
admin	GET	/admin/audit	Yes	admin	No	No
admin	GET	/admin/compliance	Yes	admin	No	No
admin	GET	/admin/metrics	Yes	admin	No	No
admin	GET	/admin/siem-export	Yes	admin	No	No

Partial* – /auth/two_factor requires an active session that has passed password verification but not yet completed TOTP. A fully authenticated session is not yet established at this stage.

CSRF – All POST routes are protected by Flask-WTF CSRF tokens. The token is embedded in every rendered form as <input type="hidden" name="csrf_token" value="...">. Requests