TECHNICAL SYSTEM OVERVIEW — CONFIDENTIAL

GSCX — Global Supply Chain Exchange

A regulated digital exchange platform connecting importers, exporters, financiers, logistics providers, and compliance officials across global trade corridors.

Version 1.0.0 **Runtime** Node.js / Express 4.21 **Database** MariaDB **Classification** Investor / Stakeholder **Date** March 2026

NAVIGATION

Table of Contents

01 — EXECUTIVE SUMMARY

Platform Overview

Key metrics and strategic positioning of the GSCX platform.

MEMBER ROLES	CORE MODULES	API ENDPOINTS	SECURITY LAYERS
8 entity types	6 platform modules	22 REST endpoints	5 defence layers

MISSION STATEMENT

GSCX is a permissioned exchange infrastructure for international trade. It enforces a structured onboarding pipeline — identity verification (KYC), OFAC sanctions screening, bond collateralisation, and multi-party transaction matching — before any entity may transact. The platform is designed for institutional participants: trading houses, logistics operators, commercial banks, insurers, and customs inspectors. All activity is cryptographically authenticated, role-scoped, and persisted in a relational ledger.

VALUE PROPOSITION

- Single platform spanning buyer, seller, bank, insurer, logistics, and inspector
- Mandatory KYC + OFAC compliance gate before any market access
- Bond collateral module reduces counterparty settlement risk
- Deterministic transaction matching engine for bilateral trade pairs
- TOTP-enforced MFA for all privileged operations

TARGET PARTICIPANTS

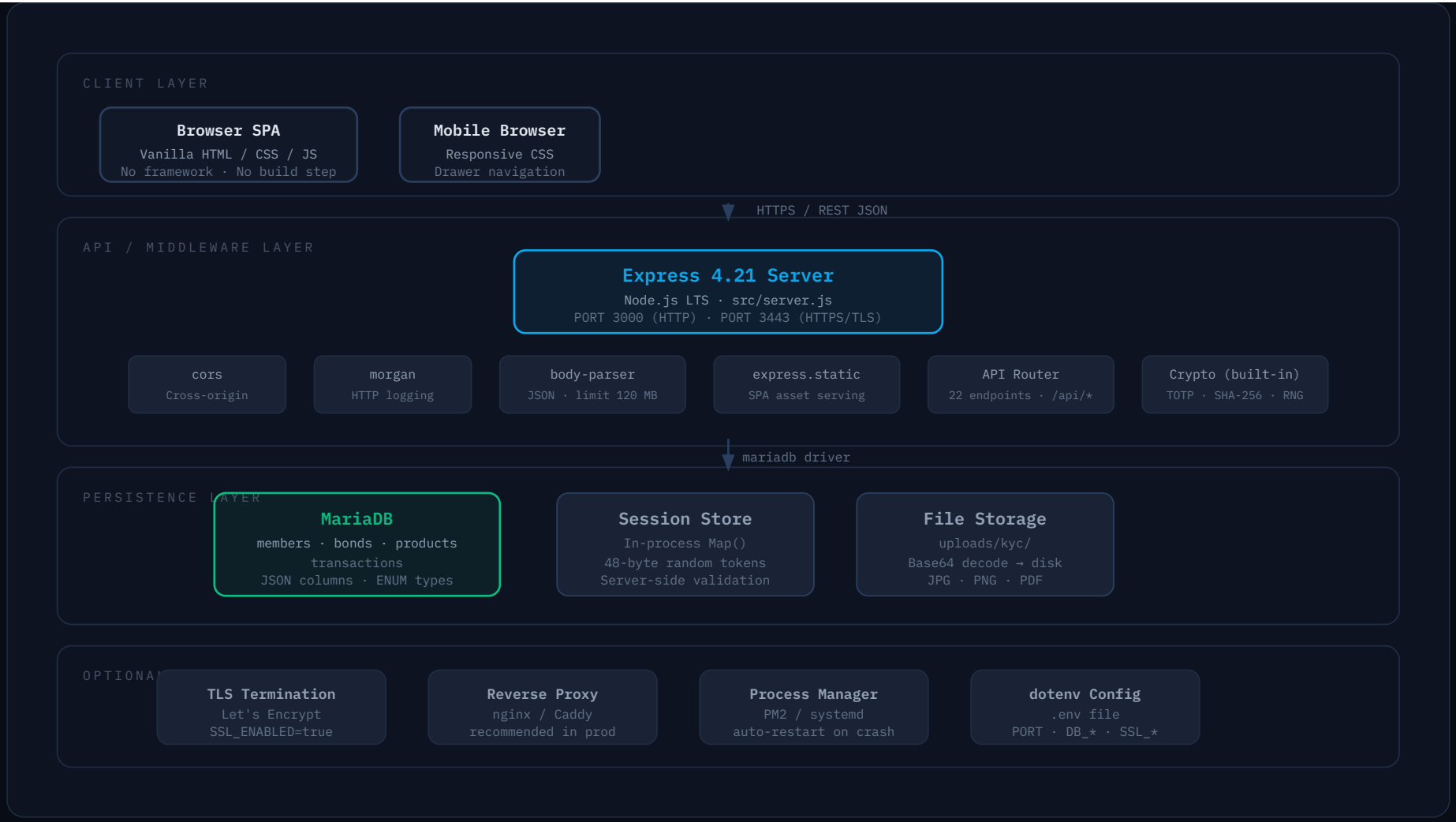
ADMIN EXPORTER IMPORTER BANK INSURANCE TRANSPORT REMITTER INSPECTOR

All non-admin participants self-register and pass through a gated compliance workflow before gaining exchange access.

02 — SYSTEM ARCHITECTURE

Layered Architecture Diagram

Three-tier deployment: stateless API server, relational persistence layer, browser-rendered SPA client.



03 — CORE MODULES

Platform Feature Set

Six integrated modules form the complete exchange lifecycle.

IDENTITY & AUTHENTICATION

KYC & ONBOARDING

PRODUCT MARKETPLACE

BOND MANAGEMENT

Session token issuance via `crypto.randomBytes(24)`. Two-step login flow — credentials first, TOTP second. Password hashing via SHA-256. Optional TOTP 2FA using RFC 6238 implemented natively with Node.js `crypto`.

- TOTP MFA
- SESSION AUTH
- SHA-256

Four-stage KYC lifecycle: NOT_STARTED → SUBMITTED → VERIFIED / FAILED. Government-issued ID upload (JPG, PNG, PDF, ≤5 MB). Admin review queue with approve/reject controls. OFAC sanctions status tracked per member.

- KYC GATE
- OFAC CHECK
- DOC UPLOAD

Six product categories: Machinery, Equipment, Hardware, Software, Commodities, Merchandise. Exporter-only listing. Weighted rolling average ratings. Per-product OFAC screening flag. License-required toggle with proof URL.

- 6 CATEGORIES
- RATINGS
- OFAC SCREEN

Two bond types: member-submitted or GSCX-issued. Tracks compensation amount, currency, contract URL, and active/expired status. Bonds form the collateral backbone of the exchange, reducing settlement risk.

- COLLATERAL
- USD SUPPORT
- MULTI-CURRENCY

⚡

TRANSACTION ENGINE

Role-typed transactions (one per member category). JSON details payload for flexible per-type metadata. Four lifecycle states: DRAFT → MATCHED → COMPLETED / ERROR. Full audit trail persisted in MariaDB.

- LIFECYCLE STATES
- JSON PAYLOAD
- AUDIT LOG

🔗

MATCHING ENGINE

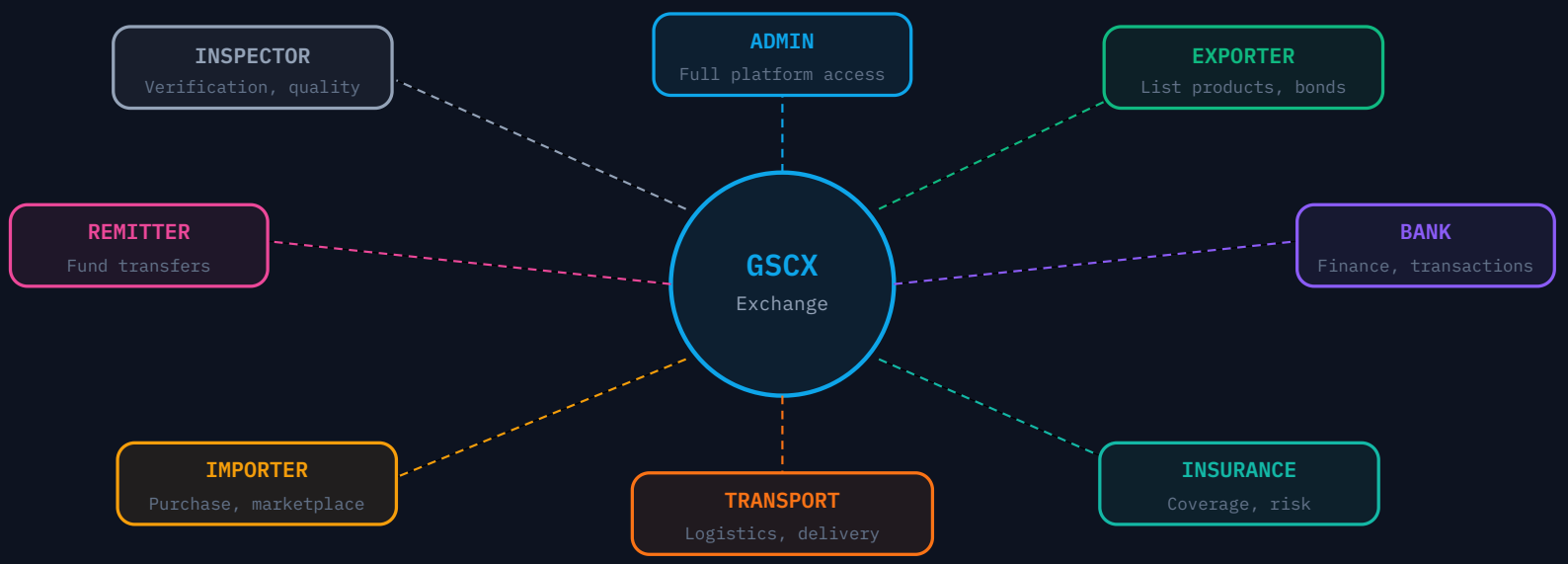
Admin-controlled bilateral matching of two DRAFT transactions into a MATCHED pair in a single atomic SQL update. Designed for trade-finance pairing: e.g. an importer's purchase order against an exporter's shipment confirmation.

- BILATERAL
- ATOMIC
- ADMIN ONLY

04 — MEMBER ROLES & PERMISSIONS

Role Access Matrix

Eight entity types with differentiated access to platform modules and actions.



ROLE	REGISTER	MARKETPLACE	LIST PRODUCTS	TRANSACTIONS	BONDS	ADMIN PANEL	MATCHING ENGINE
ADMIN	✓ Setup only	✓	—	✓	✓	✓ Full	✓
EXPORTER	✓	✓	✓ Own	✓	✓	—	—
IMPORTER	✓	✓	—	✓	✓	—	—
BANK	✓	✓	—	✓	✓	—	—
INSURANCE	✓	✓	—	✓	✓	—	—
TRANSPORT	✓	✓	—	✓	✓	—	—

ROLE	REGISTER	MARKETPLACE	LIST PRODUCTS	TRANSACTIONS	BONDS	ADMIN PANEL	MATCHING ENGINE
REMITTER	✓	✓	—	✓	—	—	—
INSPECTOR	✓	✓	—	✓	—	—	—

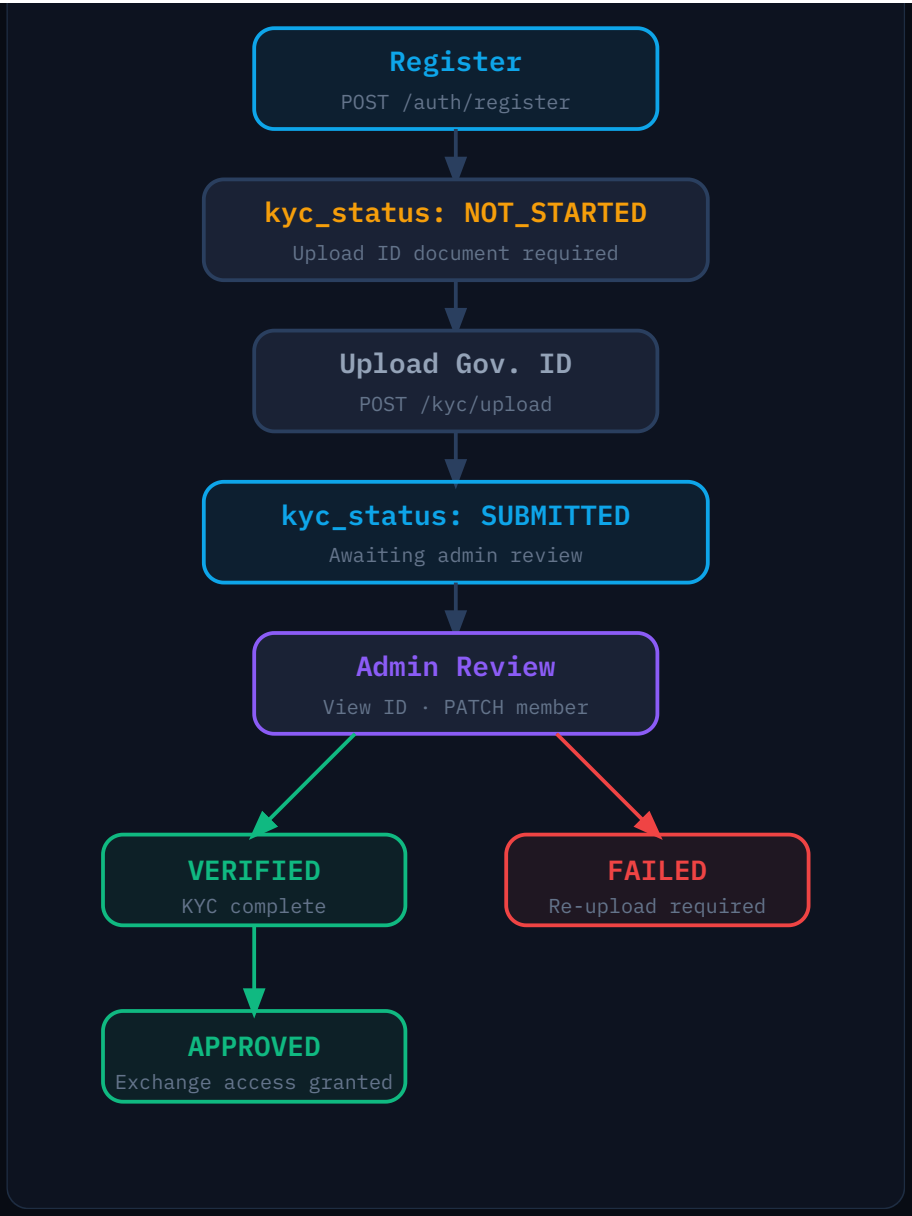
05 — TRANSACTION FLOWS

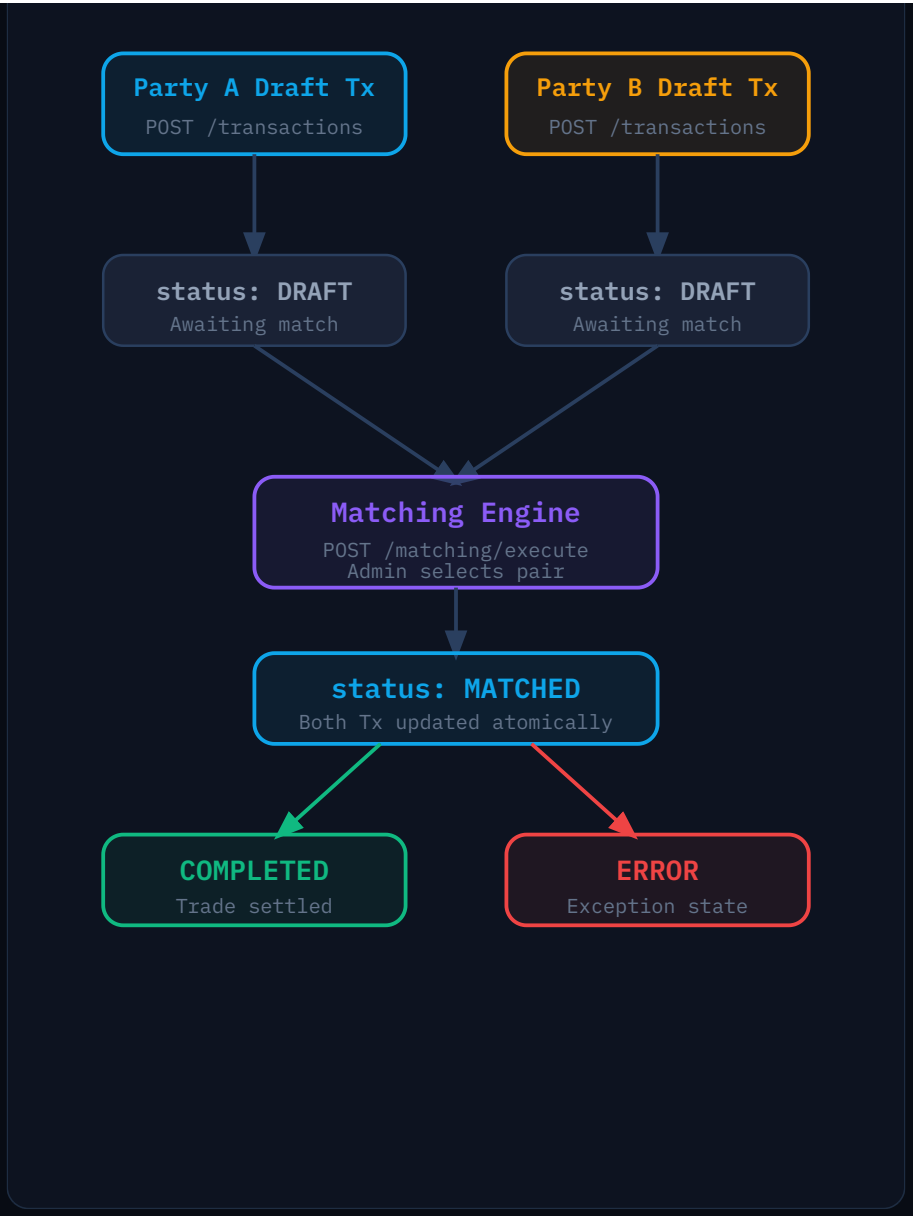
Lifecycle Diagrams

Member onboarding pipeline and transaction matching flow.

MEMBER ONBOARDING PIPELINE

TRANSACTION MATCHING FLOW





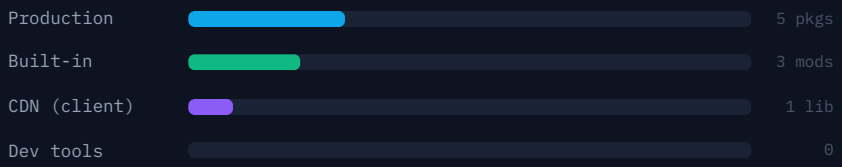
06 — TECHNOLOGY STACK

Runtime & Dependencies

Minimal, auditable dependency surface. All cryptography uses Node.js built-in modules.

COMPONENT	TECHNOLOGY	VERSION	ROLE	NOTES
Runtime	`Node.js`	LTS	Server execution environment	No transpilation required
Web Framework	`express`	^4.21.2	HTTP routing, middleware pipeline	JSON body parsing, static file serving
Database Client	`mariadb`	^3.4.0	MariaDB / MySQL connection pool	Parameterised queries, async/await
CORS	`cors`	^2.8.5	Cross-origin request handling	Configured globally on all routes
HTTP Logger	`morgan`	^1.10.0	Request/response logging	dev format in development
Config	`dotenv`	^16.4.7	.env file loading	PORT, DB_*, SSL_* variables
Cryptography	`crypto` (built-in)	Node built-in	TOTP, password hash, session tokens	Zero external crypto packages
TLS	`https` (built-in)	Node built-in	HTTPS server (optional)	Let's Encrypt cert paths via env
Frontend	Vanilla HTML/CSS/JS	—	Single-page application	No framework, no build tool, no bundler
QR Codes	`qrcode@1.5.4`	1.5.4	TOTP setup QR rendering	CDN browser bundle, client-side only
Fonts	Google Fonts CDN	—	Space Grotesk, IBM Plex Mono	Exchange terminal aesthetic
Database	MariaDB	10.x+	Persistent relational storage	JSON columns, ENUM types, FK constraints

DEPENDENCY COUNT



FRONTEND ARCHITECTURE

Single HTML File
index.html mounts the SPA shell. All rendering is client-side via innerHTML.

Zero Build Step
No webpack, no Vite, no TypeScript transpilation. Edit and reload.

CSS Custom Properties

Full dark exchange theme via :root variables. Responsive drawer navigation.

07 — API REFERENCE

REST Endpoint Catalogue

All endpoints are prefixed `/api`. Request and response bodies are `application/json`.

METHOD	ENDPOINT	AUTH	DESCRIPTION
POST	/init	PUBLIC	Run schema migrations, check admin setup status
POST	/admin/setup	PUBLIC	Create first administrator account (one-time)
POST	/auth/register	PUBLIC	Self-register a new non-admin member
POST	/auth/login	PUBLIC	Authenticate; returns session token or OTP challenge
POST	/auth/logout	PUBLIC	Invalidate session token
POST	/auth/settings	SESSION	Change password or disable TOTP
POST	/auth/totp/setup	SESSION	Generate TOTP secret and otpauth:// URI
POST	/auth/totp/enable	SESSION	Verify code and activate TOTP on account
POST	/kyc/upload	SESSION	Upload base64 ID document, sets kyc_status=SUBMITTED
GET	/kyc/document/:memberId	SESSION	Stream KYC document (admin or self only)
GET	/members	PUBLIC	List all members
POST	/members	PUBLIC	Create member (admin seeding)

METHOD	ENDPOINT	AUTH	DESCRIPTION
PATCH	/members/:id	PUBLIC	Update member fields (status, KYC, etc.)
GET	/products	PUBLIC	List all products
POST	/products	PUBLIC	Create product listing (exporter)
PATCH	/products/:id/ofac	PUBLIC	Toggle OFAC screening status
PATCH	/products/:id/rate	PUBLIC	Submit star rating (rolling weighted average)
GET	/bonds	PUBLIC	List all collateral bonds
POST	/bonds	PUBLIC	Create bond record
GET	/transactions	PUBLIC	List all transactions ordered by date
POST	/transactions	PUBLIC	Create DRAFT transaction
PATCH	/transactions/:id/status	PUBLIC	Update transaction status
POST	/matching/execute	PUBLIC	Atomically match two DRAFT transactions
GET	/dashboard	PUBLIC	Aggregate stats and recent activity
GET	/health	PUBLIC	Liveness probe — confirms DB connectivity

08 — SECURITY ARCHITECTURE

Defence-in-Depth Model

Five security layers from transport to compliance, all implemented without external security libraries.

Layer 1 — Transport Security

Optional HTTPS via Node.js built-in `https` module. Let's Encrypt certificate paths configured through environment variables.

- TLS 1.2+
- HTTPS
- LET'S ENCRYPT

Layer 2 — Authentication & Session Management

Session tokens issued as 48-character hex strings from `crypto.randomBytes(24)`. Stored server-side in a `Map()`. Never persisted to database. Password hashed as SHA-256 before storage. Two-step login enforces TOTP before session issuance.

- SESSION TOKENS
- SHA-256
- TWO-STEP LOGIN

Layer 3 — Multi-Factor Authentication (TOTP)

RFC 6238 TOTP implemented from scratch using only `crypto.createHmac`. Base32 encode/decode, HMAC-SHA1, dynamic truncation, 30-second time steps with ±1 window tolerance. Compatible with Google Authenticator, Authy, and any TOTP app. QR code provisioning via `otpauth://totp/` URI.

- RFC 6238
- HMAC-SHA1
- 30S WINDOW
- ±1 TOLERANCE

Layer 4 — Input Validation & File Security

KYC document uploads restricted to JPG, PNG, PDF extensions. Maximum 5 MB enforced on decoded binary (not base64 length). Express JSON body limit set to 120 MB to accommodate base64 overhead. File stored outside the `public/` directory; served only through authenticated API endpoint.

- TYPE ALLOWLIST
- 5 MB LIMIT
- PRIVATE STORAGE

Layer 5 — Compliance Controls

OFAC sanctions status tracked per member and per product. KYC four-stage lifecycle enforced at the data model level with ENUM constraints. Member status (PENDING/APPROVED/REJECTED) prevents unauthorized market participation. Admin approval required for all non-admin accounts.

- OFAC SCREENING
- KYC GATE
- APPROVAL WORKFLOW

TOTP IMPLEMENTATION DETAIL

```
step  = floor(Date.now() / 30000)
key   = base32Decode(secret)
hmac  = HMAC-SHA1(key, uint64BE(step))
off   = hmac[19] & 0x0f
code  = ((hmac[off] & 0x7f) << 24)
      | (hmac[off+1] << 16)
      | (hmac[off+2] << 8)
      |  hmac[off+3]
token = (code % 1_000_000).padStart(6,'0')
verify: check step-1, step, step+1
```

KNOWN SECURITY CONSIDERATIONS

⚠ **Session Store is In-Process**

Sessions reset on server restart. Production deployments should use Redis or a persistent session store for HA setups.

⚠ **SHA-256 Password Hashing**

Production hardening should upgrade to bcrypt or Argon2 for PBKDF with cost factor resistance to brute force.

⚠ **API Authentication Scope**

Several admin-only mutation endpoints rely on client-side role enforcement. Server-side role guards on all write endpoints are recommended for hardened production deployment.

Relational Schema

Four core tables with foreign key relationships enforced at the database level.

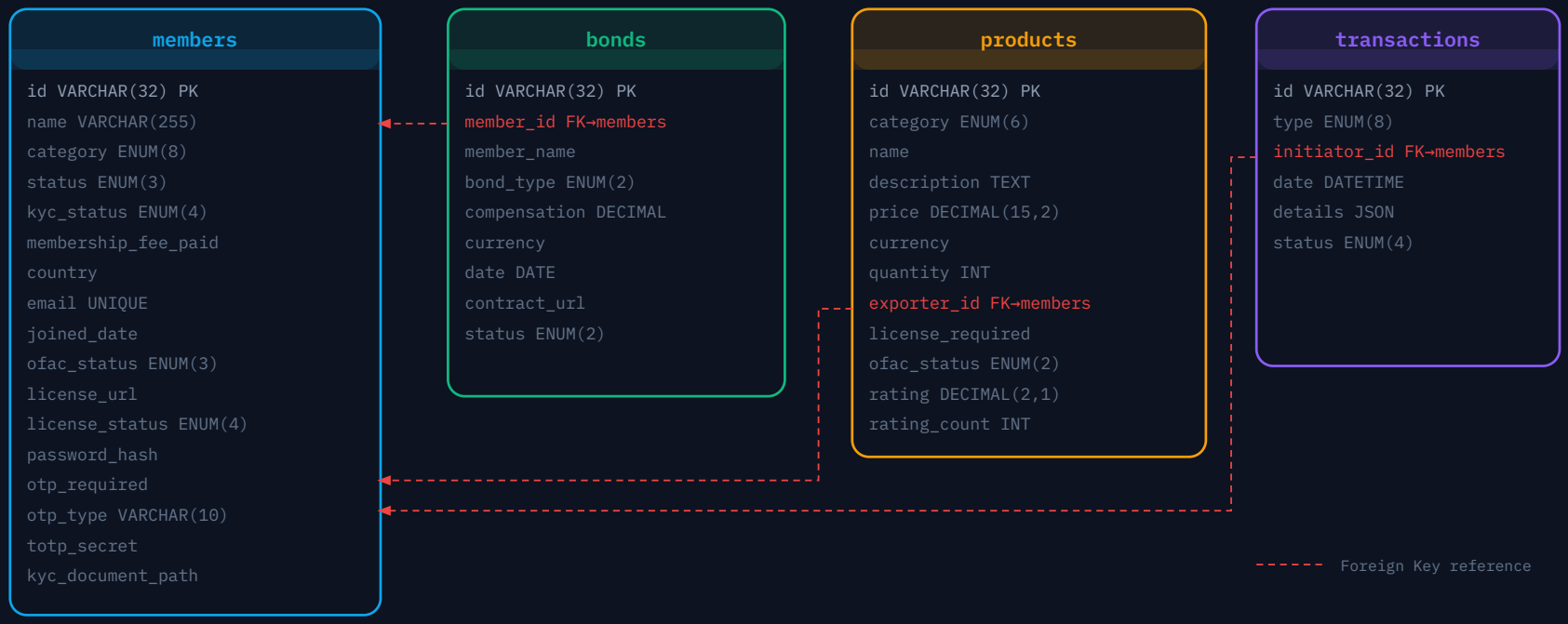


TABLE	PRIMARY KEY	FOREIGN KEYS	NOTABLE CONSTRAINTS	ROW VOLUME
members	`id VARCHAR(32)`	—	UNIQUE email, ENUM category/status/kyc_status	Low (tens–hundreds)
bonds	`id VARCHAR(32)`	`member_id → members.id`	ENUM bond_type (2), ENUM status (2)	Low–Medium
products	`id VARCHAR(32)`	`exporter_id → members.id`	ENUM category (6), ENUM ofac_status (2)	Medium (thousands)
transactions	`id VARCHAR(32)`	`initiator_id → members.id`	JSON details, ENUM status (4)	High (ongoing)

Environment & Configuration

Twelve-factor app design. All configuration is environment-variable driven with no hardcoded values.

ENVIRONMENT VARIABLES

VARIABLE	DEFAULT	PURPOSE
`PORT`	3000	HTTP listen port
`SSL_ENABLED`	false	Enable HTTPS server
`SSL_PORT`	3443	HTTPS listen port
`LETSENCRYPT_PRIVKEY_PATH`	—	TLS private key path
`LETSENCRYPT_FULLCHAIN_PATH`	—	TLS certificate chain
`DB_HOST`	localhost	MariaDB host
`DB_PORT`	3306	MariaDB port
`DB_USER`	—	Database user
`DB_PASS`	—	Database password
`DB_NAME`	—	Database name

DEPLOYMENT CHECKLIST

Database Init

Call `POST /api/init` on first boot. Runs schema migrations with `IF NOT EXISTS` guards — safe to call repeatedly.

Admin Setup

First load presents first-time setup screen. Admin credentials are created by the deployer, not hardcoded in source.

TLS in Production

Set `SSL_ENABLED=true` with valid Let's Encrypt paths. Place nginx/Caddy in front for HTTP→HTTPS redirect.

Process Supervision

Use PM2 (`pm2 start src/server.js`) or systemd for auto-restart, log rotation, and cluster mode.

QUICK START

```
# Clone and install
git clone <repo> && cd GSCXNodeJS
npm install

# Configure environment
cp .env.example .env
```

```
# edit DB_HOST, DB_USER, DB_PASS, DB_NAME

# Start server
npm run dev          # development
npm start            # production

# Initialise database (browser or curl)
curl -X POST http://localhost:3000/api/init

# First-time admin is created via the web UI setup screen
```