



CIS 4951 Capstone II - Fall 2025

Green Vault

Project Documentation

Members:

Giorgio Abboud, B.S, Malik Dagher, B.S
Audrey Gamero, B.S, Daniella Lacotera, B.S
Rodrigo Munoz Salas, B.S



CIS 4951 Capstone II

Giorgio Abboud

Malik Dagher

Audrey Gamero

Daniella Lacotera

Rodrigo Munoz Salas

December 1, 2025

1 Executive Summary

The main problem Green Vault solves is that most traders, especially beginners and intermediate-level users, receive a trade fill but have no clear way to understand whether their execution was good or bad. Existing tools are either too complex or focused only on professionals, leaving everyday traders without guidance on slippage, timing, or how market movement impacted their trade.

The users who benefit most are:

- Beginner traders, who want a simple explanation of their trade in plain English.
- Intermediate traders, who want deeper performance metrics to learn and improve over time.

Our solution is a full-stack platform that takes a single user trade and produces a clear performance analysis. The React UI allows users to sign up, log in, enter their trade details, and view results. The FastAPI analyzer service fetches 1-minute Open, High, Low, Close, and Volume (OHLCV) market data from the Twelve Data API around the trade time and calculates metrics such as VWAP slippage, shortfall, effective and realized spreads, impact, and drift. The Go internal API handles authentication, validation, and data storage in PostgreSQL, acting as the only service with database access to ensure clean system boundaries.

Key results include a fully containerized system deployed via Docker Compose, strong validation rules with detailed error feedback, and successful testing of calculation logic, API behavior, and data persistence. Users can now enter a trade and receive not only metric-level insight but also a short explanation of what went right or wrong.

Green Vault turns **complex trading analytics** into **practical guidance** for real users.

2 Problem & Requirements (O1)

Green Vault is a full-stack post-trade analytical platform that serves to provide performance resorts based on the user's data and stock values of the time. A React/Vite UI collects a single user fill, then a FastAPI analyzer fetches 1-minute OHLCV from the Twelve Data API around the fill time, followed by a Go API with Postgres database that persists users accounts and their trade performances. It calculates overall execution, impact and timing performance by computing key metrics including VWAP slippage, shortfall, realized spread, effective spread, market impact, and drift. Then a resulting performance explanation is returned through a rule-based review so traders can see their execution quality.

2.1 Notation and Data Required

2.1.1 User Input

- Timestamp: RFC3339 datetime string that is not in the future, between 10:30 and 17:00 EST, weekdays only, and not on US holidays.
- Price: Positive number with at most 8 decimal places.
- Quantity: Positive number with no decimal places.
- Side: Either Buy or Sell.
- Symbol: Uppercase ticker, letters/digits only (A-Z, 0-9), real symbol, and a length of 1-5.
- Mode: Either Analyze or Estimate.

2.1.2 Open, High, Low, Close, and Volume (OHLCV)

- Open is the price at the beginning of a trade session.
- High is the highest price during a trade session.
- Low is the lowest price during a trade session.
- Close is the final price of a trade session.
- Volume is the amount of stocks traded during a session.
- Retrieved from Twelve Data using an API key.

2.1.3 Typical Price

- Since this project relies on Open, High, Low, Close, Volume (OHLCV) 1 minute bars, which are a combination of individual quotes or trades over a 1 minute period.
- In order to approximate market prices over that period, we calculate the typical price with the formula ([OANDA, 2024](#))

$$TP = \frac{H + L + C}{3}$$

- Arrival Price = Typical Price in our project.

2.1.4 Side Variables

- Positive if a trade order is a buy.
- Negative if a trade order is a sell.
- Basis points (BPS) are units of measurement to describe the percentage change in the value of financial trades. For example 1 bps is equivalent to 0.01% ([Langager, 2024](#)).
- In both cases a negative bps value calculated is better for the trader. A positive value means you paid above the average asking price or sold below the average asking price. A negative value means you paid below the average asking price or sold above the average asking price.
- The [ts-1min, ts+1min] windows are required to find appropriate 1 minute bars according to the user's trade. For example, a minute bar aggregates everything from 10:01:00 to 10:01:59.999. A fill at 10:01:05 is somewhere inside that bar; a fill at 10:01:59 is almost at the end of it.

2.1.5 Hidden Metrics

- VWAP slippage
 - Reveals how much better or worse your performance was than the market average at the time. It compares your trade Volume Weighted Average Price (VWAP) vs the market VWAP over your trade interval. ([Makrides, 2024](#))

$$VWAPSlippage(bps) = 10000 \times side \times \frac{TradeVWAP - MarketVWAP}{MarketVWAP}$$

- Implementation Shortfall
 - It calculates the difference between the initial price when a trader wants to buy or sell, and the final price after the trade is executed, considering all commissions, fees and taxes. ([Hayes, 2023](#))

$$Shortfall(bps) = 10000 \times side \times \frac{TradeVWAP - ArrivalPrice}{ArrivalPrice}$$

- Effective Spread
 - Calculates the difference between the bid and ask price to see how much you paid on the trade. Represents how far from the mid price you traded at each fill. ([frd, 2025](#))

$$EffectiveSpread(bps) = 10000 \times 2 \times side \times \frac{TradeVWAP - TypicalPrice}{TypicalPrice}$$

- Realized Spread
 - Calculates how much of the spread is temporary and how much of it is reminded after the trade. It finds the price after it had time to revert after the trade. We look ahead 5 minutes after the trade and find its typical price to compare it to the current typical price.

$$RealizedSpread(bps) = 10000 \times 2 \times side \times \frac{TradeVWAP - FutureTypicalPrice}{FutureTypicalPrice}$$

- Market Impact
 - The permanent component of the trade. It is how much the stock price moves in your favor during the trade. The post typical price is after the end of the execution window. (Wu, 2022)

$$MarketImpact(bps) = 10000 \times side \times \frac{PostTP - TypicalPrice}{TypicalPrice}$$

- Drift
 - Determines how the market moves before your trade. Shows if you paid or sold at the right time. The pre typical price is the typical price one minute before the arrival time.

$$Drift(bps) = 10000 \times side \times \frac{TypicalPrice - PreTypicalPrice}{PreTypicalPrice}$$

2.1.6 Analyze vs Estimate

- In our API we have two functions, the analyze and estimate.
- Analyze uses historical trade data, including prices after the trade takes place. That allows us to calculate all six metrics.
- Estimate runs in near real time, taking for input the current datetime instead of a user inputted one, meaning it only has access to currently existing trade data. We are not able to look 1 or 5 minutes ahead into the future. Any attempt to guess would be too inaccurate. As a result, we cannot calculate realized spread and market impact since we require data after the trade. It will also rely on computing the metrics based on the current typical price, but will use the user's inputted price as a backup if the TP could not be found.

2.2 Stakeholders

- Primary
 - Beginner traders who simply want to understand what went right and wrong in plain English and in a way they can easily understand without being overwhelmed.
 - Intermediate traders who require a more detailed review of their past trades to learn from their mistakes. They will have access to individual metrics being broken down and calculated to further assess their performance.
- Secondary
 - The product owner who sets the scope of the project, prioritizes features, and declares acceptance criteria.
 - The backend engineers who focus on the main features like calculating metrics, determining performance, handling APIs and data persistence.
 - The frontend engineer who designed the UI and made a user friendly interface.
 - The QA engineers who thoroughly tested each feature and ensured proper validations were set in place.
 - The DevOps member who set up Docker and containerized the project.

2.3 Personas

- Giovanni Johnson (Beginner): Enters his fill on the desktop application and expects a plain-language performance review of what went right and wrong without TCA jargon.
- Paul Rogers (Intermediate): Wants more detailed explanations for his performance, access to the calculated metrics, and the ability to review past trades.

2.4 Functional Requirements

- UI Flow
 - Signup/login/logout/profile. User history view, calculate page calls analyzer then POST fills and metrics to Go API when authenticated.
- Auth & Validation, Internal
 - Signup /v1/users: field-level 422 for bad email/password/name, 409 if email exists, bcrypt hashing, and JWT session issuance.
 - Login /v1/login: 422 on empty/format issues, 401 on invalid credentials, and session JWT set on success.
 - Logout /v1/logout: clears session.
 - Profile /v1/users/me GET/PUT/DELETE: auth required, PUT enforces old password + character complexity + retype password, DELETE clears session and returns 204 payload.
 - History /v1/history/data: auth required, filter with optional fields like symbol and returns fills and metrics.
 - Protected routes return 401 when unauthenticated.
- Analyzer
 - Validate fills (RFC3339 -04:00, trading hours, no weekends or holidays, symbol exists, price>0 with ≤ 8 decimal points, qty>0, side set to buy/sell).
 - Fetch OHLCV [ts-1m, ts+1m] and a realized horizon for analyze, timezone normalization to America/New_York, handle Twelve Data API errors with HTTP 429/500.
 - Metrics: VWAP slippage using HLC3, implementation shortfall, effective spread, realized spread, market impact and drift are properly calculated and returned.
 - Estimate mode: Only VWAP slippage using HLC3, implementation shortfall, effective spread, and drift. Realized spread and market impact are marked as undeterminable.
 - Review builder: axis bands (execution/impact/timing), conclusion, how to improve block, and axis summary.
- UI Error Handling
 - Render 422 field errors inline, 409 duplicate email, 401 invalid login/expired session, 429 data provider rate limit and 500 generic fallback.
 - Gate save-history behind auth and show provisional/undeterminable labels when metrics are missing.

2.5 Non-Functional Requirements

- Accuracy: Signed bps consistent with side, overlap weighting avoids boundary bias, reject invalid inputs, replace uncalculable values for estimate with “undeterminable”.
- Availability: Docker Compose services on 5173 (UI), 8000 (analyzer), 8080 (API); 99% during trading hours if hosted.
- Privacy: Only OHLCV requests hit Twelve Data but the user’s data remains local in Postgres.

2.6 Constraints

- Data: OHLCV only. Using HLC3 mid proxy instead of NBBO/venue.
- Timezones: Inputs must be timezone aware and are expected to be normalized to America/New_York (Eastern Time Zone).
- Vendors: Twelve Data rate limits are 8 calls per minute.

2.7 Success Criteria

- Auth correctness: invalid signup/login/profile inputs yield 422 with field errors, duplicate signup returns 409, bad creds return 401, protected routes return 401 when unauthenticated.
- Data validation: analyzer rejects bad timestamps (future dates, weekend/holiday, out of hours), invalid symbols, price/qty/side errors with structured 422 JSON responses, and Go API rejects malformed fills/metrics payloads.
- Metric integrity: when OHLCV bars are present, analyzer returns finite values for available metrics. Missing or non-finite values surface as undeterminable.
- Mode behavior: estimate mode marks future-dependent metrics, realized spread and market impact, as undeterminable, and returns softened review.
- Output stability: responses include request_id and consistent JSON structure.
- Persistence: authenticated users can save their fills and metrics to later retrieve them via history endpoint with optional symbol filter.
- Error surfacing: Twelve Data rate-limit/empty-responses return 4xx or 5xx errors with explanatory payloads which are displayed by the UI.

2.8 Risks & Mitigations

- Invalid inputs/outputs: validation functions within the code catch invalid values and cleanly return handled 4xx or 5xx errors with an error message to be displayed in the UI.
- Persistence/security: relying on JSON Web Token and bcrypt. Mitigation: require JSON Web Token set per env, session middleware on protected routes, do not leak which auth field failed on 401.

2.9 Prioritized Backlog

- **User Story 1:** Complete and test the metric calculating functions.
- **User Story 2:** Integrate real market data from the TwelveData API.
- **User Story 3:** Build a backend API to connect to the frontend.
- **User Story 4:** Develop strong validation for calculate
- **User Story 5:** Automated Request Validation Tests
- **User Story 6:** Twelve Data API Endpoint Validation
- **User Story 7:** Standardized error contact to UI error responses
- **User Story 8:** Guarantee formats for calculate & save fill validation
- **User Story 9:** Database connectivity and migrations.
- **User Story 10:** Provide a clean and accessible UI:
- **User Story 11:** Provide instant feedback when email/password/name/last name are invalid (validate and guide signup form inputs).
- **User Story 12:** Display server-side field errors on signup/login
- **User Story 13:** Provide instant feedback when fill input is invalid.
- **User Story 14:** Display server-side field errors on calculate
- **User Story 15:** Create the project documentation file.

3 System Overview & Architecture (O2)

Green Vault is a containerized, service-oriented system composed of three application services (UI, Analyzer, and Internal) and one database (Postgres), all running inside a Docker Compose network. The UI service is the central orchestrator, communicating directly to the Analyzer for market calculations and to the Internal API for authentication and persistence. Only the Internal API service has direct access to the database. The design ensures separation of concerns, secure data persistence, real-time analytics, and dependable deployment across environments.

3.1 High-Level Architecture

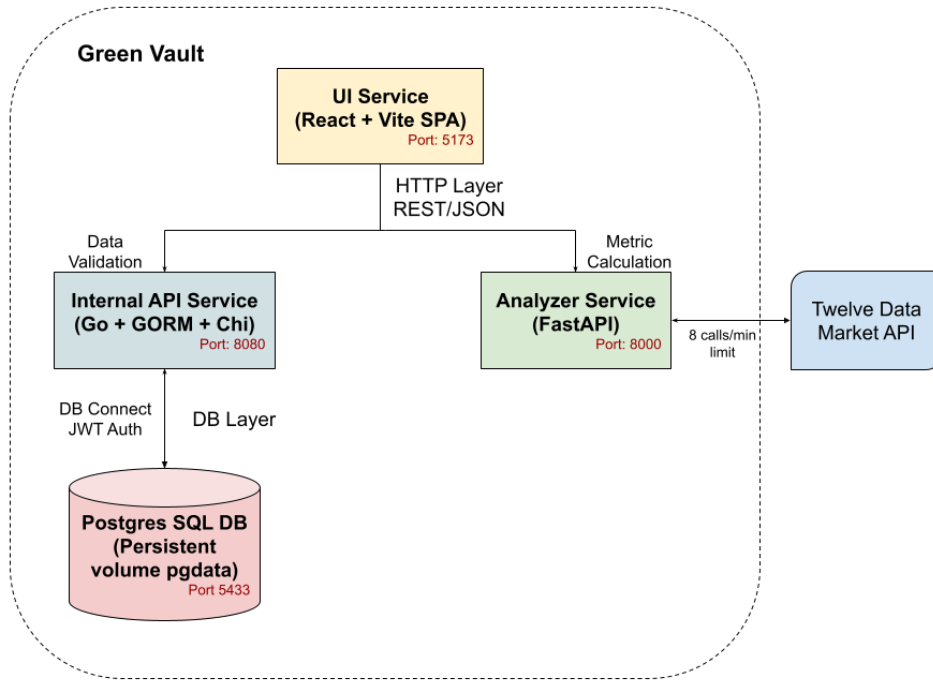


Figure 1: Green Vault - System Architecture

1. UI Service (React + VITE SPA)

- Runs in the browser, served from the ui container (port 5173).
- Exposes screens for signup, login, calculation, profile editing, and history.
- Acts as the orchestrator:
 - Sends auth and persistence requests to the Internal API.
 - Sends calculation requests to the Analyzer Service, then forwards the returned metrics to the Internal API when the user chooses to save them.

2. Internal API Service (Go + GORM + Chi)

- Runs in the internal container (port 8080).
- Exposes a REST/JSON API under /v1 for:
 - User and session management (/v1/users, /v1/login, /v1/logout, /v1/me, /v1/users/me, /v1/users/me DELETE).
 - Persistence of trade fills and metrics (/v1/fills, /v1/history/data).
- Implements:
 - HTTP layer (chi router, handlers, middleware).
 - Authentication and authorization (JWT-based).
 - Datastore layer using GORM, mapped to PostgreSQL.

- Only this service has access to the Postgres DB, via a DB URL connection string injected through environment variables.
3. Analyzer Service (FastAPI)
- Runs in the analyzer container (port 8000).
 - Provides a dedicated calculation endpoint consumed by the UI:
 - The UI sends symbol, side, quantity, timestamp, price, and mode.
 - Analyzer calls the Twelve Data Market API (up to 8 calls per minute) to fetch OHLCV/time-series data.
 - Analyzer computes trade metrics (VWAP Slippage, Shortfall, Effective Spread, Realized Spread, Market Impact, and Drift) and returns a purely computational JSON response to the UI.
 - Analyzer is stateless and has no direct DB access. All persistence decisions are left to the UI + Internal API.
4. Postgres SQL Database (Persistent volume pgdata)
- Runs in the db container (port 5432 inside the network, exposed as 5433 on host).
 - Uses the pgdata Docker volume to ensure durable storage of:
 - Users, auth data.
 - User fills.
 - Metrics saved after an analysis.
 - Configuration (user, password, DB name) comes from /.env.db.

3.2 Key Technology Stack

Component	Technology
Frontend UI	React, Vite, Docker, TailwindCSS, React Router
Analytics Service	Python, FastAPI, Twelve Data API
Auth & Internal Service	Go, GORM, Chi, JWT, REST/JSON, PostgreSQL, testcontainers
Persistence	Docker Volume for Postgres
Images	Chainguard Images (Golang , Python , Node , Postgres)
System Integration	Docker Compose

Table 1: Green Vault - Key Technology Stack

3.3 Architectural Behavior

- Requests flow using explicit services boundaries:
 - UI ↔ Internal ↔ DB is the secure path for auth + validation + persistence.
 - UI ↔ Analyzer ↔ TwelveDataAPI is the real-time analytics path for computational trade metrics.
- Credentials are set securely through environment files expected at container startup, including:
 - /.env.db for database auth.
 - internal/.env.docker for backend connectivity and JSON Web Token.
 - ui/.env and analyzer/.env for service routing and api key usage.
- The system avoids direct dependency between analytics and storage
- All services communicate over a Docker internal bridge network, where service-to-service connections use DNS-based discovery.

- Dependencies are boot-gated using healthcheck and depends_on conditions to prevent startup race failures.
- The Analyzer service is intentionally stateless, ensuring that all generated analytics results are ephemeral until explicitly persisted by the UI via the Internal API layer.
- The Internal API is the system’s single source of truth for persistent data, controlling schema migrations through GORM and durable storage through the pgdata Docker volume.
- Service reliability is reinforced using auto-restart policies (unless-stopped) for both the Analyzer and Internal API, preventing unexpected downtime without manual intervention.

3.4 API Surface & Storage Overview

The user-facing service layer is exposed through a RESTful API implemented in Go using the Chi framework. The Green Vault Internal API Service acts as the system’s sole gateway to persistence, enforcing validation and ownership over all stored entities, while the Analyzer remains stateless and isolated from database concerns.

- Public Endpoints
 - POST /v1/users: Signup.
 - POST /v1/login: Login.
 - POST /v1/logout: Logout
- Protected Endpoints (JWT-gated via middleware)
 - GET /v1/me: Get authenticated user profile.
 - POST /v1/fills: Save validated trade fills + computed metrics.
 - PUT /v1/users/me: Edit profile.
 - DELETE /v1/users/me: Delete user.
 - GET v1/history/data: Fetch fill history + saved metrics.
- Persistence & Data Ownership
 - All persistent data is stored in a PostgreSQL database, accessed exclusively by the Internal API Service.
 - Database schemas and models are auto-migrated using GORM to ensure consistency between the application layer and storage.
 - Trade and user data durability is guaranteed using the pgdata Docker volume, ensuring zero data loss between container restarts or redeployments.

4 Implementation Notes O2

4.1 Repository Structure

Green Vault at the root-level:

```
Green-Vault/
  analyzer/
  internal/
  ui/
  docker-compose.yml
  README.md
```

The **analyzer** folder contains the stateless Python service that computes trade metrics on-demand. Within the analyzer folder are the following files:

```

analyzer/
  cmd/
    main.py          # FastAPI entry point; exposes POST /calculate
  src/
    service.py       # Orchestrates analyze/estimate modes; fetches market data
    metric.py        # Computes VWAP, spreads, impact, drift, etc.
    twelve_client.py # Adapter for Twelve Data API (1-minute market bars)
    data_client.py   # Helper for external API calls
    validate.py      # Input validation logic
  reviewer/
    axes.py          # Grades execution, impact, timing axes
    band.py          # Maps numeric metrics to qualitative performance bands
    conclusion.py    # Generates conclusion text from axis scores
    formatter.py     # Formats metrics and feedback strings for the UI
  tests/             # Unit & integration tests (pytest)
  requirements.txt   # Python dependencies
  Dockerfile        # Container build config
  .env              # Twelve Data API key (not tracked in git)

```

The key purpose of **analyzer** is to respond to the UI when it sends a trade (symbol, price, quantity, timestamp). Analyzer returns structured metrics and a human-readable review. Analyzer fetches live/historical market data from Twelve Data to compute benchmarks (VWAP, spreads, etc.).

The **internal** folder is the REST API & Database Layer. Internal Go service handles all HTTP requests, user authentication, session management, and database persistence. Within the **internal** folder are the following files:

```

internal/
  cmd/api/
    main.go          # API startup; mounts routes and middleware
  datastore/
    datastore.go      # Database abstraction layer (GORM store)
    datastore_test.go # Datastore tests
  package/
    models/
      models.go       # GORM models: User, UserFill, Metric
    handlers/
      auth.go         # Signup, login, logout, JWT validation handlers
      auth_test.go    # Auth tests
      fills.go        # Save fills & metrics to DB
      fills_test.go   # Fill handler tests
      session.go      # Session & cookie utilities
      common.go       # Shared JSON helpers, error writers, and constants
      store_interface.go # Repository interface definitions
      testutil.go     # Test helpers
  go.mod
  go.sum             # Go module dependencies
  .env.docker / .env.local # DB & JWT config (not tracked in git)

```

The key purpose of **internal** is to serve as an entry point for all client requests. Internal validates auth tokens, manages user accounts, and persists fills & metrics to Postgres. Additionally, it runs GORM migrations automatically on startup.

The **ui** folder is the React Frontend layer. The **ui** is a React + Vite single-page application for user interactions (signup, login, trade analysis.) Within the **ui** folder are the following files:

```

ui/
  src/
    main.jsx         # React entry; mounts router and global layout
    api.js           # HTTP client for analyzer and internal API calls
    auth.jsx         # React authentication context and utilities
    index.css
    styles.css       # Global styles

```

```

pages/
  Home.jsx
  Login.jsx
  Signup.jsx
  Calculate.jsx    # Trade analysis form (Analyze/Estimate modes)
  EditProfile.jsx
  History.jsx      # Past trade metrics and reviews view
index.html
package.json
tailwind.config.js
postcss.config.js
.env # Proxy target URLs (not tracked in git)

```

The key purpose of **ui** is to serve as the user-facing interface. **UI** provides forms to input trades and view analyses. **UI** communicates with both the analyzer (for metrics) and the internal API (for authentication and persistence).

Reasons for this structure:

- Separation of Concerns: Each service has a single responsibility (UI, computation, auth/DB).
- Independent Scaling: A busy analyzer won't block auth requests; each service scales independently.
- Technology Fit: React for interactivity, Python for numerical computation, Go for fast HTTP low-level DB operations.
- Docker Orchestration: docker-compose.yml brings all three up together; easy local dev and production readiness.
- Testability: Each service has tests; can be developed and tested in isolation.

4.2 Coding Conventions

Green Vault follows standard coding conventions that will be briefly outlined below. Across the three services (Analyzer, UI, Internal), there was a lightweight but consistent coding conventions used to maintain clarity and team consistency. Each service follows the idiomatic style of its language, and all secrets, environment variables, and service configurations are kept out of version control.

1. Python (Analyzer)

- Follows PEP 8 standards for formatting and function naming.
- Use type hints in function signatures.
- FastAPI entrypoint in cmd/main.py; core logic in src/.
- Errors returned as structured JSON (ok, error, field_errors).
- Reviewer/analysis logic separated into reviewer/ and src/metric.py.
- Wrap third-party API calls in dedicated modules (twelve_client.py, data_client.py).
- Tests written with pytest.
- Tests are in tests/ directory.

2. Go (Internal API)

- Idiomatic Go naming: PascalCase for exported symbols, camelCase otherwise.
- Handlers separated in package/handlers/; models in package/models/.
- GORM models with automatic migrations.
- chi router used for clean, modular route definitions.
- Consistent JSON error responses and appropriate HTTP status codes.
- All queries go through the datastore abstraction (datastore.go).
- Handlers follow signature: `func NameHandler(app *App) http.HandlerFunc return func(w http.ResponseWriter, r *http.Request) ...`

- For testing, using Go's built-in testing package.
3. React/JavaScript (UI)
- Functional components with Hooks; components named in PascalCase; functions in camelCase.
 - Pages in pages/, shared utilities in src/api.js and src/auth.jsx.
 - Environment variables (VITE_*) used for API base URLs.
 - Tailwind CSS for styling consistency.
 - Centralize HTTP calls in api.js.
 - Handle errors gracefully; show user feedback for failed requests.
 - Store JWT/session token in memory or secure storage.
 - Dependencies in package.json; lock file tracked.
4. Cross-Service Practices
- All environment variables stored in .env files excluded via .gitignore.
 - Shared logging strategy with clear messages and request IDs where applicable.
 - Clear separation between presentation (UI), business logic (Analyzer), and persistence/auth (Internal).
 - Meaningful commit messages and isolated feature branches.

4.3 Notable Patterns

- In-memory test double
 - fakeStore implements a simple in-memory store used for unit tests
 - Data kept in maps: users keyed by email, fills and metrics keyed by UUID
- Concurrency safety
 - All fakeStore methods lock a mutex (f.mu) to make the fake safe for concurrent test usage.
- Defensive input validation
 - Methods validate inputs (nil checks, uuid.Nil, empty strings) and return clear errors for invalid params.
- CRUD helpers with simple semantics
 - Create/Get/Update/Delete methods for users, plus CreateUserFill/CreateMetric and listing methods.
 - updateUser finds the existing record by ID and replaces it in place.
 - DeleteUser cascades: remove a user and also deletes related fills and metrics
- Copies returned to avoid aliasing
 - Methods return copies of stored structs (or copies before return) to prevent tests from mutating shared in-map instance
- List helps assemble related data
 - ListUserFills attaches the first matching Metric to each UserFill and supports optional symbol filtering.
 - List methods return empty slices rather than nil where appropriate
- Test-specific app utilities
 - newTestApp constructs an App configured for tests (fake store, test JWT secret)
 - With User injects a user ID into request context for handler tests
- JSON/test helpers

- `mustJSONBody` builds JSON request bodies and fails the test on marshal errors
- `stdResp` and `decodeStdResp` standardize reading/decode of JSON responses from handlers
- `t.Helper()` used to make test failures point at caller code
- Context and UUID conventions
 - Methods accept `context.Context` and use `github.com/google/uuid` for IDs.
 - Error messages are short and explicit (e.g. “not found”, “invalid user params”).

4.4 Tradeoffs

- Test In-memory vs real DB
 - Tradeoff: fast, simple, no external deps.
 - Implication: tests are deterministic and fast but won’t catch SQL/transaction/ORM bugs or DB-specific constraints.
- Mutex for concurrency in Tests
 - Tradeoff: single global lock (sync. `Mutex`) protects all maps.
 - Implication: safe for concurrent tests but serializes all operations
- Map keys and lookup choices
 - Tradeoff: users keyed by email; fills/metrics keyed by UUID.
 - Implication: updating user email requires scanning/moving entries; getting an user turns $O(1)$ via Primary Key.
- Test helpers convenience vs realism
 - Tradeoff: `newTestApp` hardcodes JWT secret and uses the fake store for convenience.
 - Implication: quick setup for handler tests
- PKs & FKs implementation
 - Tradeoff: Faster DB look ups
 - Implication: Avoid low search when DB increases

5 Testing & Evaluation

Green-Vault employs a service-specific, multi-layered testing strategy where each microservice is validated using the most appropriate tooling for its runtime and language environment. Testing is split into two independent domains to preserve architectural boundaries and data ownership:

- Analyzer Service (Python – tested via PyTest)
 - Uses `pytest` and a rate-limited client to validate analytics workflows.
 - Remains stateless and intentionally isolated from database access.
- Internal API Service (Go – tested via `go test` + `Testcontainers`)
 - Validates HTTP routing, JWT authentication, and PostgreSQL persistence.
 - Acts as the only DB gateway, ensuring data durability and system integrity.

By aligning testing responsibilities to service boundaries, the system prevents analytic components from affecting storage concerns, improves test reliability, and reduces environment drift during experimentation.

5.1 Internal API - Go

Container-backed integration and routing test harness was implemented by using native Go tools from the Go ecosystem. The test strategy guarantees that all persistent data flows exclusively through the Internal API.

5.1.1 Test Architecture & Principles

- Isolation of persistence: Only Go integration tests provision a controlled PostgreSQL instance using testcontainers-go.
- Ephemeral test DB ensures clean migrations and repeatable schema state for every suite run.
- Parallelizable HTTP fakes use an in-memory fakeStore based on Go maps to simulate data-store behavior deterministically.

5.1.2 DB Layer Tests

A real Postgres container simulates production semantics, validating all core persistence functions:

- CreateUser()
- GetUserByEmail()
- GetUserByID()
- UpdateUser()
- DeleteUser()
- CreateUserFill()
- CreateMetric()
- ListUserFillsByUserID()
- ListUserFillsWithMetrics()
- ListUserFillsAndMetrics()

Coverage yielded 84.9% of internal statements, with 100% passing method assertions for correctness and integration reliability.

5.1.3 HTTP Layer Tests

The public and protected HTTP surface is verified using the fakeStore test harness. The following endpoints are validated across all input branches, error cases, and auth requirements, registered under the Chi router:

- POST /v1/users
- POST /v1/login
- POST /v1/logout
- GET /v1/me
- POST /v1/fills
- PUT /v1/users/me
- DELETE /v1/users/me
- GET v1/history/data

HTTP coverage reached 71.3%, with 100% passing method assertions for correction and integration reliability.

5.1.4 Performance, Reliability & Coverage Dashboards

To collect measurable evidence and visualize test quality, **go test** and **go tool** were employed.

```
• rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/fall25/Green-Vault/internal/datastore$ go test ./... -coverprofile=coverage.out
ok      github.com/Giorgio-Abboud/Green-Vault/internal/datastore 14.920s coverage: 84.9% of statements
• rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/fall25/Green-Vault/internal/datastore$ go tool cover -func=coverage.out
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:16:      New                                100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:28:      CreateUser                          100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:38:      GetUserByEmail                      100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:49:      GetUserByID                        100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:60:      UpdateUser                         60.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:70:      DeleteUser                         88.9%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:87:      CreateUserFill                     80.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:100:     CreateMetric                       80.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:113:     ListUserFillsByUserID              75.0%
github.com/Giorgio-Abboud/Green-Vault/internal/datastore/datastore.go:122:     ListUserFillsWithMetrics           71.4%
total:                                (statements)                        84.9%
```

Figure 2: DB - Test Coverage

```
• rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/fall25/Green-Vault/internal/package/handlers$ go test ./... -coverprofile=coverage.out
ok      github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers 0.409s coverage: 71.3% of statements
• rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/fall25/Green-Vault/internal/package/handlers$ go tool cover -func=coverage.out
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:32:      Signup                                78.9%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:158:     Login                               68.8%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:227:     Logout                              0.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:239:     Me                                  50.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:263:     EditProfile                         67.3%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:376:     DeleteUser                          83.3%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/auth.go:416:     ListUserFillsAndMetrics             83.3%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/common.go:35:     WriteJSON                           100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/common.go:50:     WriteError                          100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/fills.go:38:      SaveFill                            74.5%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/session.go:20:     issueSession                        85.7%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/session.go:45:     clearSession                        100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/session.go:58:     parseSession                        0.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/session.go:81:     AuthMiddleware                      0.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/session.go:96:     CurrentUserID                       100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:27:     newFakeStore                        100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:35:     CreateUser                          77.8%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:50:     GetUserByEmail                      100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:61:     GetUserByID                        100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:73:     CreateUserFill                     83.3%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:84:     CreateMetric                       83.3%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:95:     UpdateUser                         87.5%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:127:    newTestApp                          100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:135:    withUser                           100.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:142:    mustJSONBody                       80.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:157:    decodeStdResp                      80.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:166:    DeleteUser                         68.2%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:211:    ListUserFillsByUserID              0.0%
github.com/Giorgio-Abboud/Green-Vault/internal/package/handlers/testutil.go:225:    ListUserFillsWithMetrics           94.7%
total:                                (statements)                        71.3%
```

Figure 3: HTTP - Test Coverage

5.1.5 KPI Evidence & Coverage Backlog

Domain	Result
Datastore test pass rate	100%
DB Isolation boundary	maintained
HTTP test pass rate	100%
Coverage DB	84.9%
Coverage HTTP	71.3%

Table 2: Green Vault - KPI Evidence & Coverage Backlog

5.1.6 Defect Summary

Component	Coverage
Logout handler	0%
Parse Session	0%
AuthMiddleware	0%
ListUserFillsByUserID	0%

Table 3: Green Vault - Defect Summary

Logout and **AuthMiddleware** coverage may register 0.0% in granular unit reports because logout operates by issuing a **Set-Cookie** directive that **clears the session cookie at the HTTP transport layer** without mutating persistent state. The helper `parseSession()` similarly lacks direct line coverage signals because in production it is evaluated only when handlers process a presented cookie, and cookie clearing behavior is validated in **end-to-end UI-driven flows**, where the browser replaces the previous session. **AuthMiddleware** exists strictly as an **HTTP routing guard** that injects authentication into request context, and does not require datastore assertions since it never accesses the database; its correctness is confirmed indirectly by protected routes rejecting unauthenticated calls during live execution.

The method `ListUserFillsByUserID()` is part of the handler domain because the shared storage interface requires an implementation for both the real and in-memory test store, even though the DB-backed version of this logic has already been fully validated at the datastore layer and does not need to be re-tested here. This method satisfies interface contracts by using Go map implementations during tests, preventing redundancy while enabling Router and Handler wiring, maintaining abstraction consistency with deployment environments.

All Go handlers are actively tested and operational as designed, including signup, login, profile editing, fill and metric persistence, and historical data retrieval, while only the Internal API holds database access to PostgreSQL and associated volume durability boundaries, validated externally by the testcontainers-go pipeline.

5.2 Analyzer - Python

5.2.1 Scope and isolation

- All analyzer tests live in `analyzer/tests` and run with `pytest`. The analyzer is stateless and DB-free, so tests focus on pure functions, validation, and the Twelve Data adapter. Shared fixtures in `analyzer/tests/conftest.py` build tz-aware timestamps, overlapping bars, and fills to keep math deterministic.

5.2.2 Metric Math Coverage

- `analyzer/tests/test_market_trade_vwap.py`: checks trade VWAP, overlap-weighted market VWAP, side-signed slippage, plus a partial-overlap scenario that exercises fractional volume weighting across bar boundaries.
- `analyzer/tests/test_spreads_and_impact.py`: validates effective spread proxy per-fill then qty-weighted, realized spread proxy at a future horizon, and derived `impact = effective realized`.
- `analyzer/tests/test_is_timing_participation.py`: verifies implementation shortfall vs arrival TP and timing drift over the execution window.
- `analyzer/tests/test_integration_compute_all.py`: end-to-end `compute_all_metrics` with synthetic bars/fills; asserts every numeric output (trade/market VWAP, slippage, spreads, impact, shortfall, drift) matches hand-computed expectations.

5.2.3 Input Validation Coverage

- `analyzer/tests/test_fills.py`: parametrized cases for future timestamps, off-hours, weekend/holiday, negative/over-precision prices, non-integer or negative qty, invalid side, bad symbol format/length/nonexistent symbol. Confirms 422 HTTPException payloads with field-specific errors exactly as raised by `analyzer/src/validate.py`.

5.2.4 Data Provider Adapter

- analyzer/tests/test_twelve_client.py: marked integration and skipped unless API key is set. Calls Twelve Data, asserting non-empty rows, required columns, datetime typing, and numeric types. Keeps the default suite offline-fast.

5.2.5 Tooling and helpers

- Fixtures (conftest.py) ensure every bar/fill/window is tz-aware and reproducible.
- Markers: use -m “not integration” for the offline suite; add real API key to include the live data test.

5.3 Gaps

- FastAPI surface (/calculate) is untested: no request/response schema checks, status codes, or error-path assertions.
- Estimate mode is untested: no confirmation that future-dependent metrics become “undeterminable” or that estimate-mode review text returns.
- Reviewer logic is untested: no coverage for axis banding, conclusion selection, or bps string formatting (analyzer/reviewer).
- Twelve Data error paths lack tests: rate-limit and empty-payload branches aren’t exercised to confirm 429/500 behavior.
- Time edge cases: DST transitions and non-04:00 offsets aren’t covered.
- Performance/regression: no timing guardrails or fuzz/property tests for stability.
- Potential mismatch: test_integration_compute_all.py expects numeric keys like trade_vwap and vwap_slippage_bps, while the current compute_all_metrics returns stringified fields under different names and this should be reconciled before relying on the suite.

6 Security, Privacy, Accessibility & Compliance (O5)

6.1 Ethical and Societal Impact

Green Vault is built to give traders honest feedback about their execution, not to tell them what to buy or sell. The system focuses on explaining:

- How the user’s fill compares to the market around that time.
- What went right, what went wrong, and how they might improve.

This helps:

- Beginner traders avoid “blind” trading where they never learn from past fills.
- Intermediate traders get more structure and numbers behind their decisions.

6.1.1 Avoiding Harmful Behavior

- The app does not give trade recommendations or “signals.” It only explains past trades.
- It does not encourage overtrading, leverage, or “get rich quick” promises.
- Metrics like slippage and shortfall are shown as informational, not as a score of “good vs bad person.”

6.1.2 Transparency & Honesty

- The metrics are based on OHLCV and HLC3 mid prices, which are approximations. This is clearly documented in the limitations so users understand the numbers are estimates, not exact market truth.
- When metrics cannot be reliably calculated (for example, in estimate mode), the system returns “undeterminable” instead of faking a number. This avoids misleading the user.

6.2 Security

Security is handled at multiple layers: authentication, data handling, and deployment.

6.2.1 Authentication and sessions

- Users sign up with email and password.
- Passwords are hashed with bcrypt, not stored in plain text.
- Sessions are managed with JWTs, which are required for any protected route (profile, history, fills).
- Protected endpoints return 401 Unauthorized when a user is not logged in, preventing access to other people's data.

6.2.2 Input validation and safe failures

- The analyzer validates timestamps, symbols, price, quantity, and side before doing any calculations.
- The internal API validates request bodies for signup, login, profile changes, and fill/metrics payloads.
- Invalid or suspicious input returns structured 4xx responses (mostly 422) instead of causing crashes or leaking stack traces.

6.2.3 Service boundaries and least privilege

- Only the Internal API talks to Postgres. The Analyzer never sees raw user data in the database.
- The UI only calls the Internal API and Analyzer over defined REST endpoints.
- DB credentials are stored in environment files (not in code) and injected into the containers at runtime.
- JWT secret, DB password, and Twelve Data API key are all configured through .env files that are excluded from version control.

6.2.4 Deployment and resilience

- Docker Compose runs four containers (UI, Internal API, Analyzer, DB) on an internal bridge network instead of exposing the DB directly to the internet.
- Auto-restart policies ensure services restart if they crash.
- Health and dependency ordering (depends_on) are used so services do not start in a broken state.

6.3 Privacy

6.3.1 Data Collected and Stored

Green Vault keeps the data surface intentionally small:

- User account info: email, hashed password, basic profile fields.
- Trade data: symbol, side, quantity, price, timestamp, and computed metrics.
- No high-risk personal data is collected (no SSN, address, payment details, etc.).

6.3.2 Local Data Ownership

- All user fills and metrics are stored in Postgres owned by the Internal API.
- The Twelve Data API only receives OHLCV requests by symbol and time window; it does not receive user identity from Green Vault.
- This separation keeps external vendors away from the user's personal account data.

6.3.3 Principles Applied

- Data minimization: Store only the fields needed for auth and trade analysis.
- Purpose limitation: Data is used strictly for showing trade history and analysis back to the same user.
- No unnecessary sharing: There is no feature to export or share user data with third parties.

6.4 Accessibility

The current implementation focuses on clear layout and error feedback to provide full guidance when using Green Vault, open to any user with internet access

- The UI provides inline error messages for invalid fields (signup, login, calculate).
- Server-side errors (422, 401, 409, 429, 500) are mapped to user-friendly messages instead of raw codes.
- Forms use simple, step-by-step flows (login → calculate → history), which helps cognitive load.

6.5 Compliance Considerations

Green Vault is a course-level project, not a regulated production system, but it is designed in a way that points toward compliance practices:

- Data isolation: Only the Internal API can reach Postgres, which is similar to patterns used in SOC 2-style architectures where a service owns a data store.
- Environment-based secrets: Credentials and keys are kept out of source control, which is a basic requirement in most compliance frameworks.
- Auditability potential: Request IDs are included in responses, which makes it easier to trace requests through logs once a logging pipeline is added.

7 Deployment & Operations

Green-Vault is deployed using container-first orchestration via Docker Compose, enabling automated service discovery, dependency health gating, and simplified operational control. The system is operated by building and running 3 service images alongside 1 database image, all connected through an internal Compose network.

7.1 Deployment Guide

7.1.1 Two main steps

1. The services are created by running **docker compose build ui analyzer internal**
2. The system is started by running **docker compose up**

7.1.2 After Startup

1. The UI is reachable at port 5173 on the host machine once the container starts.
2. The Go service manages schema migrations automatically on boot using GORM models, so no manual database migrations are required.
3. All durable data is owned by the Go internal API, which communicates with PostgreSQL through internal container DNS (db:5432 inside the network).
4. Service uptime is maintained with restart policies (unless-stopped) to auto-recover from unexpected container shutdowns.

7.2 Required Environment Files

- When running **docker compose up**, all services will be run. However, the four of them require independent environment files that need to be set up for a successful startup. The following files must be created before creating the images.
- **/.env.db**: setup the database credentials for the postgres instance. Note: the first time the system boots up, it will automatically create the postgres account and database with the credentials that are given, no extra steps are needed [Installation Guide \(A.3.1, A.3.2\)](#)
 - `POSTGRES_USER=postgres`
 - `POSTGRES_PASSWORD={DB_PASSWORD}`
 - `POSTGRES_DB={DB_NAME}`
- **/ui/.env**: setup frontend service routing
 - `VITE_CALC_BASE_URL=http://localhost:8000`
 - `VITE_API_BASE_URL=http://localhost:8080`
- **/internal/.env.docker**: internal api connection & jwt secret. Note: `DB_PASSWORD` and `DB_NAME` are the same ones that were set up on `/.env/db`. `HEX_32` is the randomly generated value for the JSON Web Token, using openssl [Installation Guide \(A.3.11\)](#)
 - `DB_URL=postgres://postgres:{DB_PASSWORD}@db:5432/{DB_NAME}?sslmode=disable`
 - `JWT_SECRET={HEX_32}`
 - `API_PORT=8080`
- **analyzer/.env**: twelve data api key setup. Generate a Twelve Data API Key by creating an account in the [twelve data api official website](#) [Installation Guide \(A.3.4, A.3.5, A.3.6, A.3.7\)](#)
 - `TWELVE_DATA_API_KEY={KEY}`

8 Limitations & Future Work

Known limitations, shortcuts, technical debt, and prioritized roadmap items for future iterations.

8.1 Limitations

- In Transaction Cost Analysis (TCA), execution metrics are usually calculated using the National Best Bid and Offer midpoint (NBBO) midquote using tick-level Trade and Quote records data (TQR). That gives the exact inside market and lets you see if each fill was at/inside/outside NBBO and what the mid did 5 minutes later. (Hu, 2018)
- We only have Open, High, Low, Close, Volume (OHLCV) bars, so we set the typical price equal to $(\text{High} + \text{Low} + \text{Close})/3$ as a midprice proxy inside each bar. That makes our slippage and spread metrics bar-based approximations of NBBO-based TCA.
- With 1-minute bars they're usually within a few bps of a NBBO-based TCA, but with 5-minute bars the margin of error can grow into the low tens of bps. Generally speaking, a few bps difference is a negligible margin of error.
- External dependency on Twelve Data for 1-minute bars causes rate limits (8 calls per minute) and data gaps are surfaced as errors since there is no retry, caching, or fallback logic.
- Execution window is fixed to ± 1 minute around a single fill with a +1 minute horizon, meaning multi-fill or longer execution windows are not supported.
- Trading-hours validation is hard-coded to 10:30–17:00 America/New_York and a fixed -04:00 offset, which ignores daylight saving time changes and must be manually changed.
- Estimate mode cannot compute future-dependent metrics; realized spread and market impact are returned as undeterminable.
- Analyzer returns undeterminable strings for missing values.

- Auth and user flows lack email verification and brute-force protection.
- JWT_SECRET defaults to “change-me” if unset and no HTTPS or CSRF protection is set for cookie usage.
- Persistence layer stores each fill and metric without deduplication or idempotency guards and client_request_id is not enforced as unique.
- UI error handling paths are not under automated tests.
- Deployment is local Docker only and there is no migration strategy for production secrets, TLS termination, or scaling guidance.

8.2 Future Work

- Resilience for data fetching: add retry/backoff logic, cache recent OHLCV windows, and support a fallback data source, surface a clear provisional state when the latest bar may be incomplete.
- Auth hardening: require JWT_SECRET configuration, add email verification, implement rate limiting on auth and calculate endpoints, and document HTTPS/secure cookies/CSRF expectations.
- Time handling: switch validation to proper market hours with DST awareness, accept offset-aware timestamps beyond -04:00, and add unit tests for DST boundaries.
- Multi-fill and longer window support: accept arrays of fills, compute participation rate and better arrival benchmarks, and enforce a strict 10-minute window for consistency.
- UI clarity: add badges for unavailable/undeterminable metrics, clear messaging for Twelve Data 429 or empty bars, and inline mapping of backend field errors to form inputs.
- Idempotency and deduplication: use client_request_id or a hash to prevent duplicate fill inserts and store analyzer request_id for traceability across services.
- Observability: add structured logging, latency/error metrics, and request_id propagation across UI, analyzer, and Go API, and a basic dashboards for error rates and data-provider failures.
- Security and compliance: document secret management for docker-compose, and add basic audit logs for saved fills.
- UX polish: richer history views with filters, export, and comparisons. Introduce mobile UI.

References

- (2025). Spread and price impact. Accessed 29 November 2025.
- Hayes, A. (2023). Implementation shortfall: Meaning, examples, shortfalls. Investopedia. Accessed 23 December 2023.
- Hu, E. et al. (2018). Tick size pilot plan and market quality. Technical report, U.S. Securities and Exchange Commission. Published 31 January 2018.
- Langager, C. (2024). Basis points: Understanding what they are and how they are used. Investopedia. Accessed 26 November 2024.
- Makrides, D. (2024). What is slippage in trading? - definition. Quadcode. Published 18 December 2024.
- OANDA (2024). Seven moving average price options for advanced trading analysis. Published 24 September 2024.
- Wu, S. (2022). Market impact models. Pretty Quant. Published 3 September 2022.

A Installation Guide

The following document provides a step-by-step guide on how to set up and run Green-Vault locally in a machine. The primary deployment model uses Linux guidelines, including container bootstrapping, environment configuration, and service wiring, supported by screenshots to reduce environment drift during installation. Although this guide is centered on Linux systems, the system is cross-platform compatible and can run in any operating system that supports the Docker Engine. Alternative setup paths are also documented for:

- MacOS.
- Microsoft Windows.

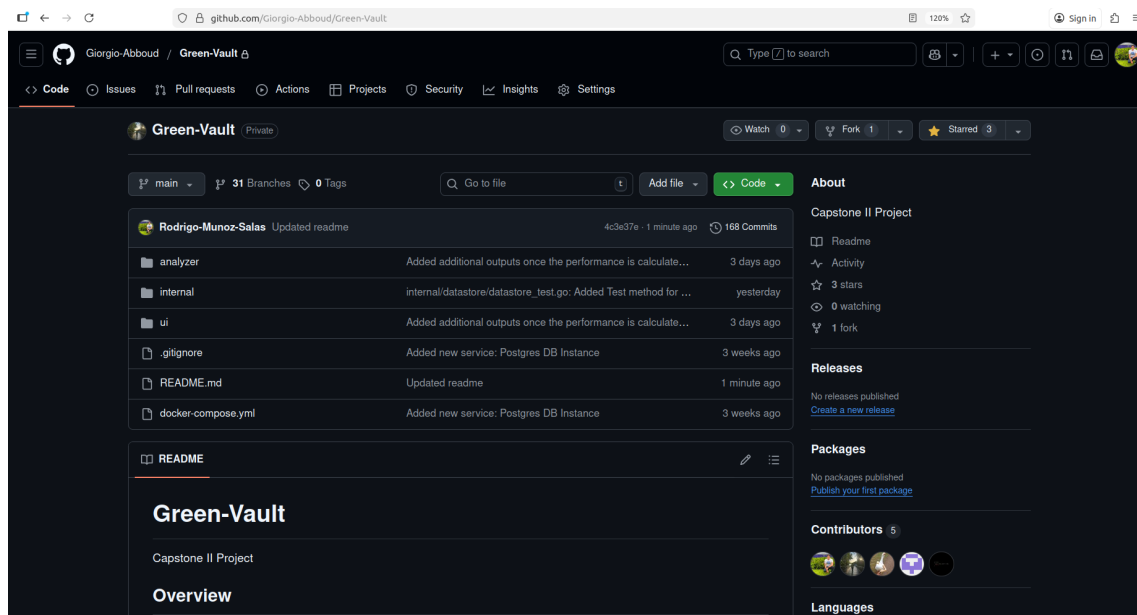
All application components (UI, Analyzer, and Internal API) are deployed as isolated containers managed by Docker Compose, while only the Go Internal API service owns and accesses the PostgreSQL database layer for persistence (which is also launched by Docker Compose).

A.1 Prerequisites

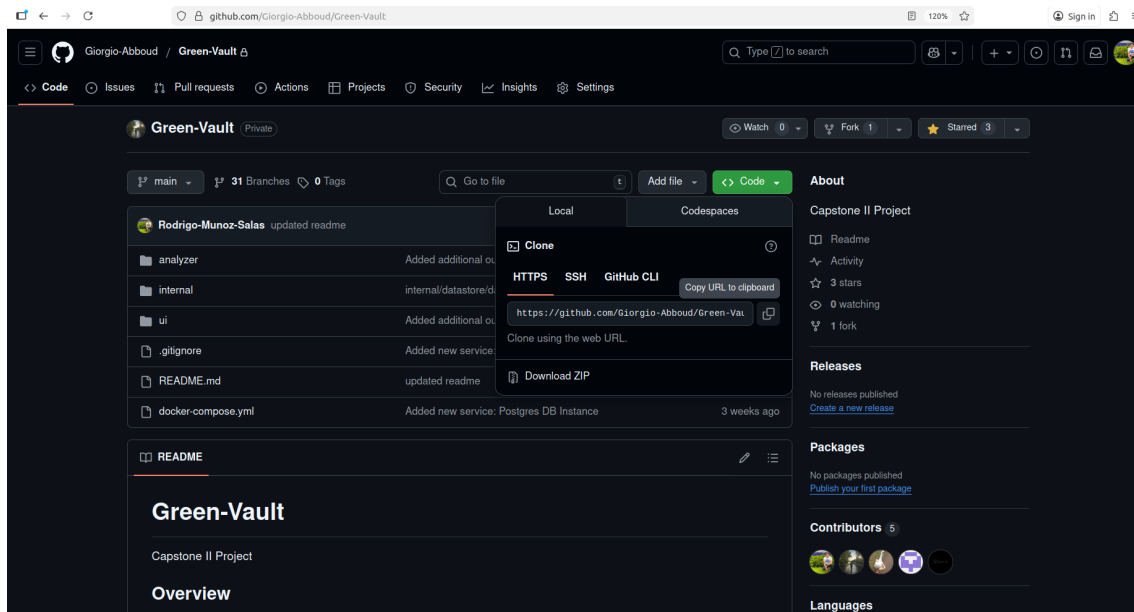
1. Install Git: <https://git-scm.com/install/>
2. Install Docker: <https://docs.docker.com/engine/install/>
3. Install Docker Compose: <https://docs.docker.com/compose/install/linux/>

A.2 Cloning Green Vault

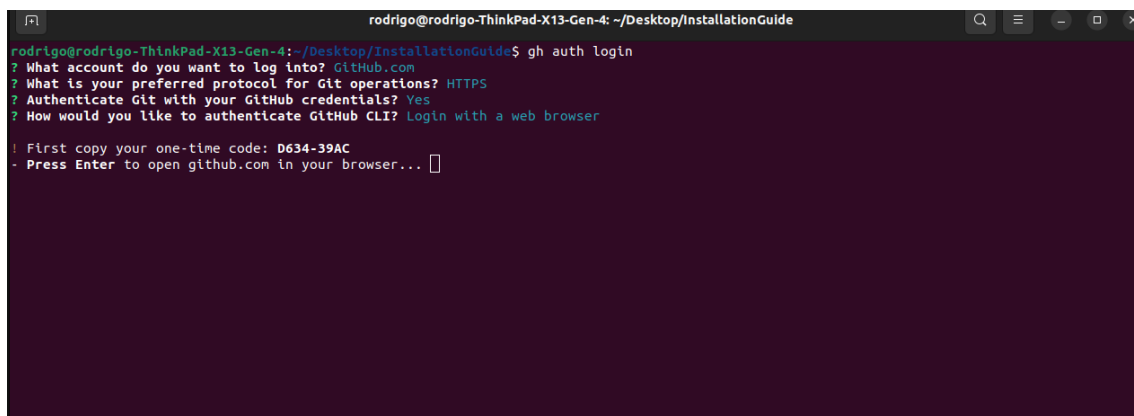
1. Access Green Vault: <https://github.com/Giorgio-Abboud/Green-Vault>

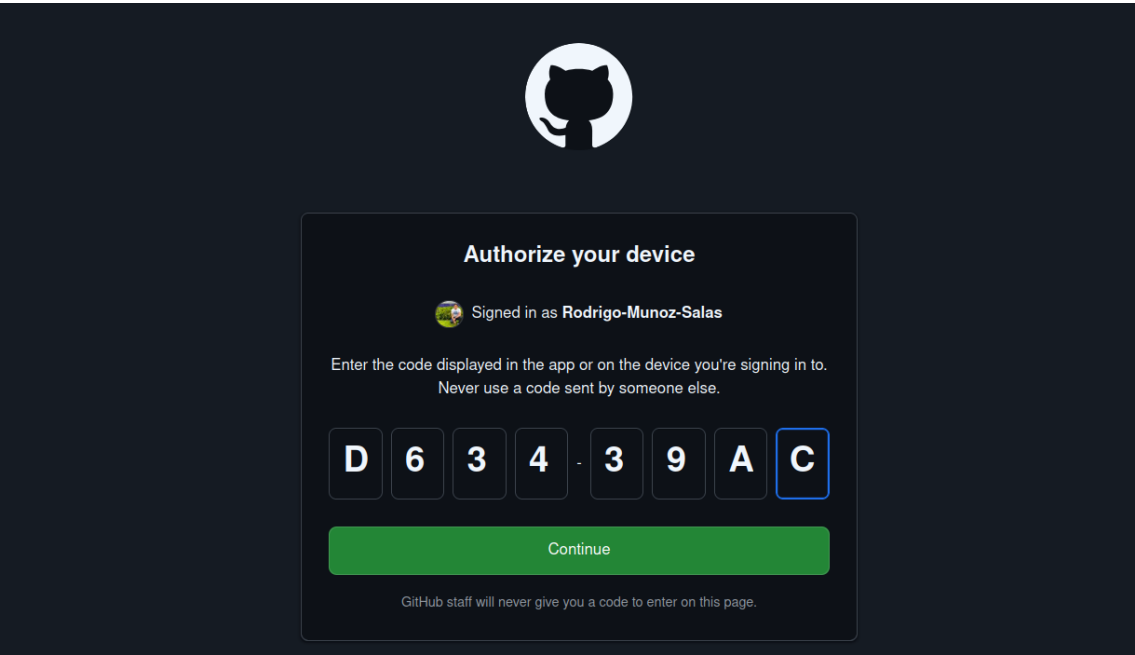
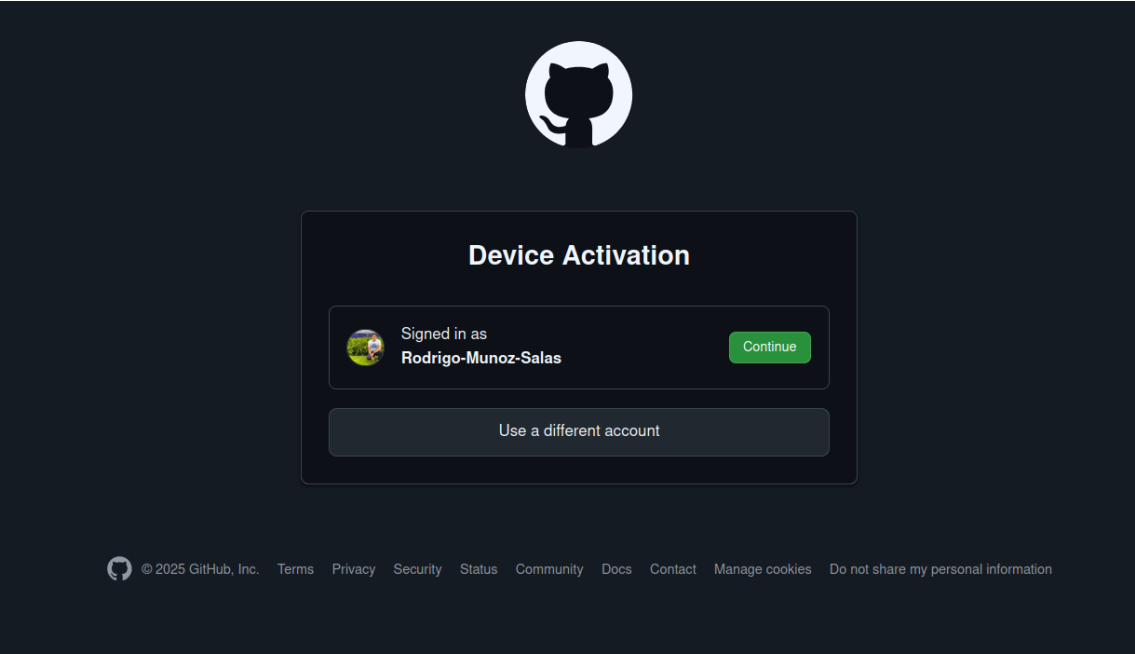


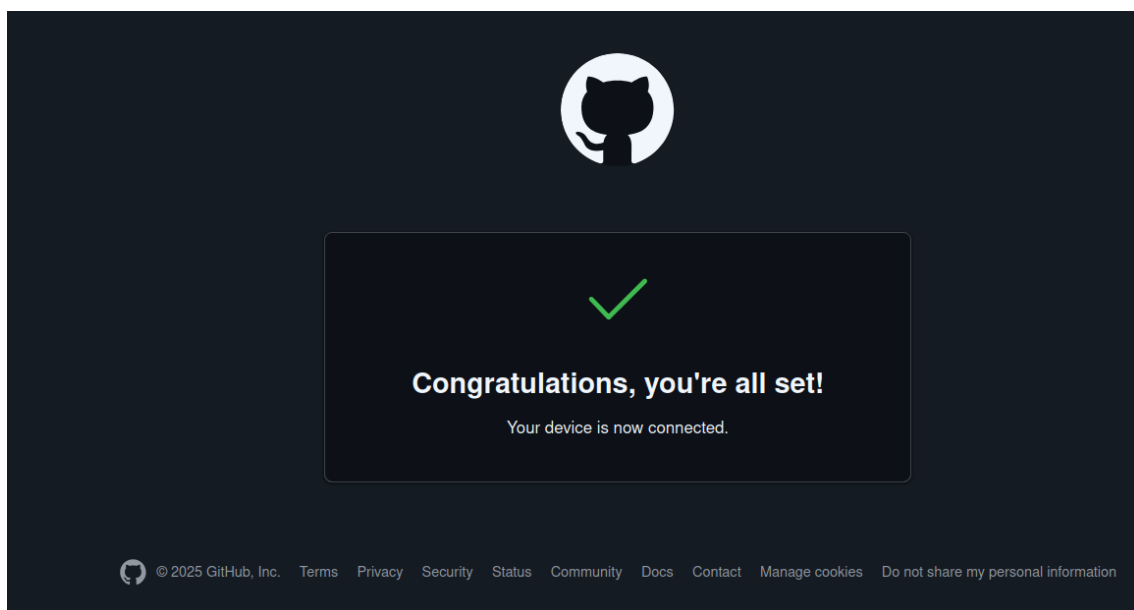
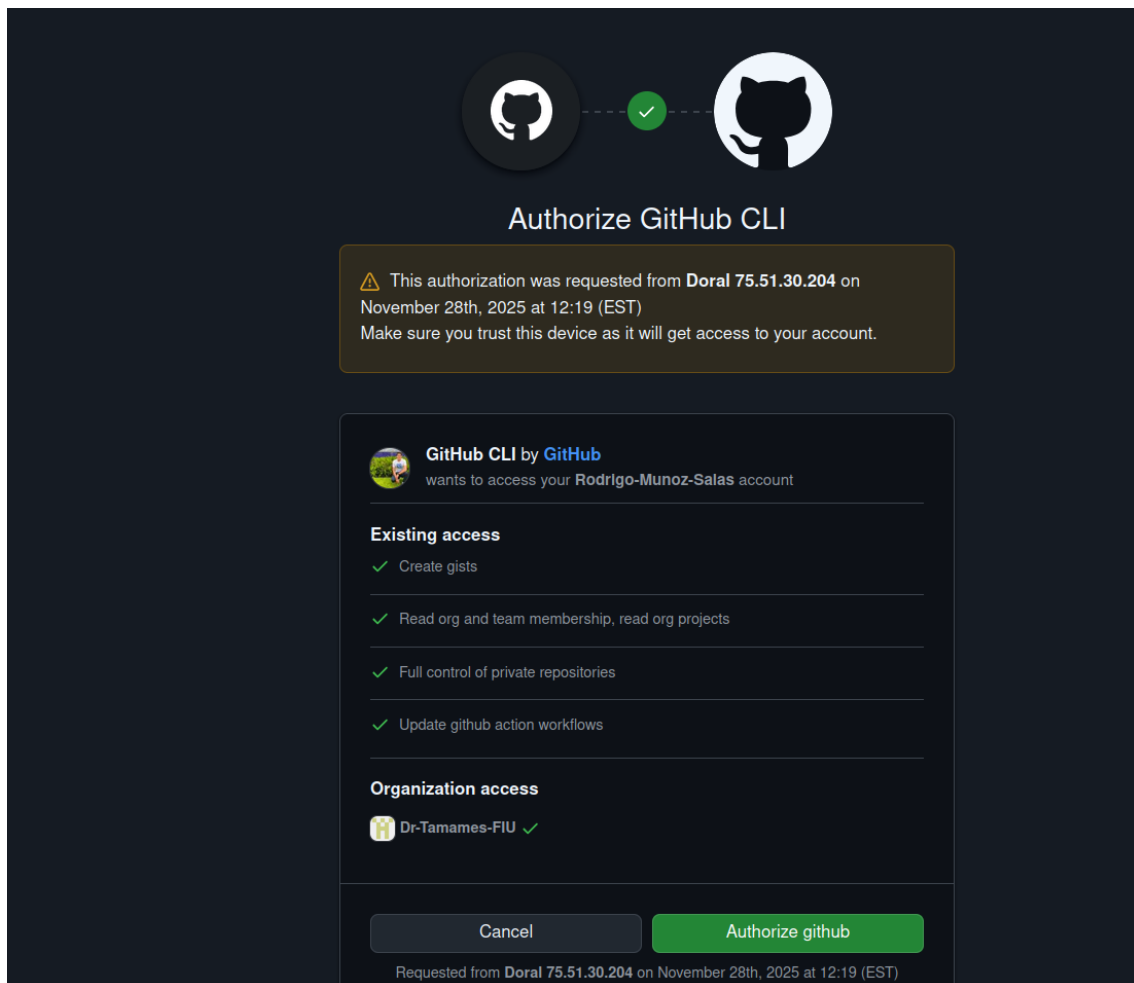
2. Click on 'Code' button and copy the URL



3. Install gh cli and log in to your GitHub account (skip if already installed)







```
ask: https://github.com/rodrigo-munoz-salas/rodrigo-munoz-salas.git
✓ Authentication complete. Press Enter to continue...
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as Rodrigo-Munoz-Salas
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide$
```

4. Clone the repository

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide$ git clone https://github.com/Giorgio-Abboud/Green-Vault.git
Cloning into 'Green-Vault'...
remote: Enumerating objects: 1213, done.
remote: Counting objects: 100% (297/297), done.
remote: Compressing objects: 100% (201/201), done.
remote: Total 1213 (delta 155), reused 164 (delta 89), pack-reused 916 (from 1)
Receiving objects: 100% (1213/1213), 371.53 KiB | 2.32 MiB/s, done.
Resolving deltas: 100% (696/696), done.
```

A.3 Setup Environment Files

1. Go to **Green-Vault/** and create a file named **.env.db**

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ ls
analyzer  docker-compose.yml  internal  README.md  ut
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ touch .env.db
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ la
analyzer  docker-compose.yml  .env.db  .git  .gitignore  internal  README.md  ut
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$
```

2. Inside the file, set up the **POSTGRES_USER**, **POSTGRES_PASSWORD**, and **POSTGRES_DB**.
Note: Here you are setting up your DB account, which will be created at startup, so you can choose your **POSTGRES_PASSWORD** and **POSTGRES_DB** or just use this:

- **POSTGRES_USER**=postgres
- **POSTGRES_PASSWORD**=InstallationGuideGreenVault25!
- **POSTGRES_DB**=greenvaultdb

If you have previously used the **postgres** image from **Chainguard** to create an account, use those credentials for this setup

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ ls
analyzer  docker-compose.yml  internal  README.md  ut
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ touch .env.db
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ la
analyzer  docker-compose.yml  .env.db  .git  .gitignore  internal  README.md  ut
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ nano .env.db
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ cat .env.db
POSTGRES_USER=postgres
POSTGRES_PASSWORD=InstallationGuideGreenVault25!
POSTGRES_DB=greenvaultdb
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$
```

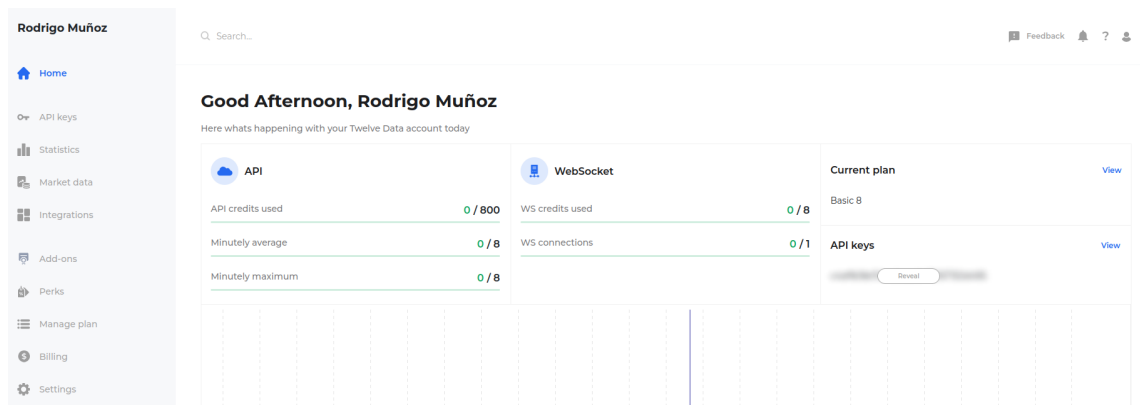
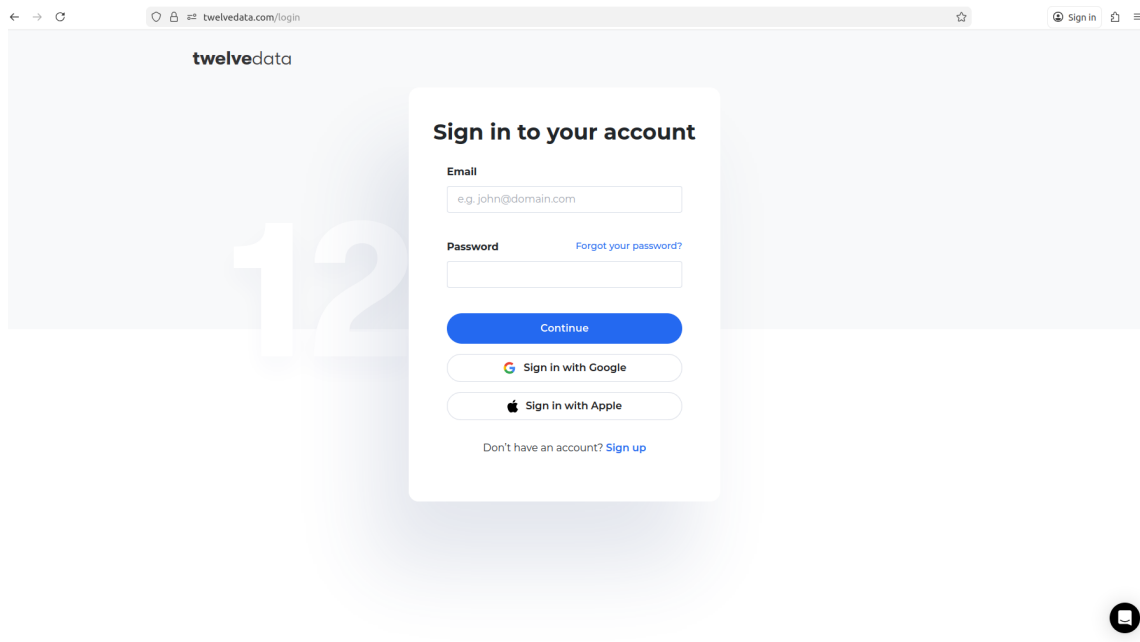
3. Go to **Green-Vault/analyzer** and create a file named **.env**

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/analyzer
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ cd analyzer/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ la
cmd  Dockerfile  .dockerignore  __init__.py  requirements.txt  reviewer  src  tests
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ touch .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ la
cmd  Dockerfile  .dockerignore  .env  __init__.py  requirements.txt  reviewer  src  tests
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$
```

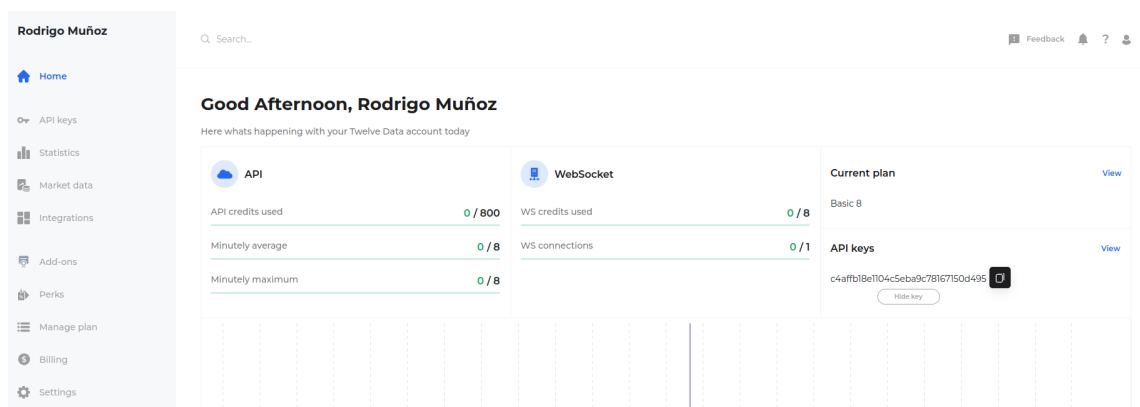
4. Inside the file, set up **TWELVE_DATA_API_KEY**

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/analyzer
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ cd analyzer/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ la
cmd  Dockerfile  .dockerignore  __init__.py  requirements.txt  reviewer  src  tests
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ touch .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ la
cmd  Dockerfile  .dockerignore  .env  __init__.py  requirements.txt  reviewer  src  tests
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ nano .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ cat .env
TWELVE_DATA_API_KEY={API_KEY}
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$
```

5. To get **{API_KEY}**, go to <https://twelvedata.com/login> and create and account/log in



6. Click on 'Reveal' in API Keys and copy it. Note: Do not share your api key with anyone.



7. Copy and paste YOUR API Key in the .env file

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/analyzer
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ cd analyzer/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ la
cmd Dockerfile .dockerignore __init__.py requirements.txt reviewer src tests
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ touch .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ la
cmd Dockerfile .dockerignore .env __init__.py requirements.txt reviewer src tests
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ nano .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ cat .env
TWELVE_DATA_API_KEY={API_KEY}
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ nano .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$ cat .env
TWELVE_DATA_API_KEY=c4affb18e1104c5eba9c78167150d495
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/analyzer$
```

8. Go to **Green-Vault/internal** and create a file named **.env.docker**

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/internal
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ cd internal/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ la
cmd datastore Dockerfile .dockerignore go.mod go.sum package
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ touch .env.docker
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ la
cmd datastore Dockerfile .dockerignore .env.docker go.mod go.sum package
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$
```

9. Inside the file, set up **DB_URL**, **JWT_SECRET**, and **API_PORT**.

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/internal
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ cd internal/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ la
cmd datastore Dockerfile .dockerignore go.mod go.sum package
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ touch .env.docker
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ la
cmd datastore Dockerfile .dockerignore .env.docker go.mod go.sum package
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ nano .env.docker
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ nano .env.docker
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ cat .env.docker
DB_URL={DB_URL}

JWT_SECRET={JWT_SECRET}

API_PORT=8080
```

10. For **{DB_URL}** you will use **POSTGRES_USER**, **POSTGRES_PASSWORD**, and **POSTGRES_DB** that were previously set in this format:

postgres://{POSTGRES_USER}:{POSTGRES_PASSWORD}@db:5432/{POSTGRES_DB}?sslmode=disable

If you used the same credentials that were configured earlier, you can safely copy and paste this connection string directly:

postgres://postgres:InstallationGuideGreenVault25!@db:5432/greenvaultdb?sslmode=disable

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ nano .env.docker
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ cat .env.docker
DB_URL=postgres://postgres:InstallationGuideGreenVault25!@db:5432/greenvaultdb?sslmode=disable

JWT_SECRET={JWT_SECRET}

API_PORT=8080
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$
```

11. For **{JWT_SECRET}**, run **openssl rand -hex 32** in your terminal and copy the generated random value. Note: if you don't have openssl in your machine, you can just use the provided one:

bd9417b640ab3232c0e47acb9298922e9ee37c5c89b7b82d6bc5f1fc9374683b

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/internal
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/internal$ openssl rand -hex 32
bd9417b640ab3232c0e47acb9298922e9ee37c5c89b7b82d6bc5f1fc9374683b
```

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/Internal
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/Internal$ openssl rand -hex 32
bd9417b640ab3232c0e47acb9298922e9ee37c5c89b7b82d6bc5f1fc9374683b
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/Internal$ nano .env.docker
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/Internal$ cat .env.docker
DB_URL=postgres://postgres:InstallationGuideGreenVault251@db:5432/greenvaultdb?sslmode=disable

JWT_SECRET=bd9417b640ab3232c0e47acb9298922e9ee37c5c89b7b82d6bc5f1fc9374683b

API_PORT=8080
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/Internal$
```

12. Go to **Green-Vault/ui** and create a file named **.env**

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/ui
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ cd ui/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ la
Dockerfile .dockerignore index.html package.json package-lock.json postcss.config.js src tailwind.config.js
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ touch .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ la
Dockerfile .dockerignore .env index.html package.json package-lock.json postcss.config.js src tailwind.config.js
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$
```

13. Inside the file, set up **VITE_CALC_BASE_URL** and **VITE_API_BASE_URL** as:

- **VITE_CALC_BASE_URL**=http://localhost:8000
- **VITE_API_BASE_URL**=http://localhost:8080

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault/ui
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ cd ui/
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ la
Dockerfile .dockerignore index.html package.json package-lock.json postcss.config.js src tailwind.config.js
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ touch .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ la
Dockerfile .dockerignore .env index.html package.json package-lock.json postcss.config.js src tailwind.config.js
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ nano .env
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$ cat .env
VITE_CALC_BASE_URL=http://localhost:8000
VITE_API_BASE_URL=http://localhost:8080
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault/ui$
```

A.4 Run Green Vault!

1. Go to the top of the directory (Green-Vault/) where **docker-compose.yml** is located

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ ls
analyzer docker-compose.yml internal README.md ui
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$
```

2. Run **docker compose build ui analyzer internal**, you will get a successful message at the end.

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ docker compose build ui analyzer internal
```

```
[+] Building 3/3
✔ green-vault-ui Built 0.0s
✔ green-vault-analyzer Built 0.0s
✔ green-vault-internal Built 0.0s
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$
```

3. Verify the images were created by running **docker images**

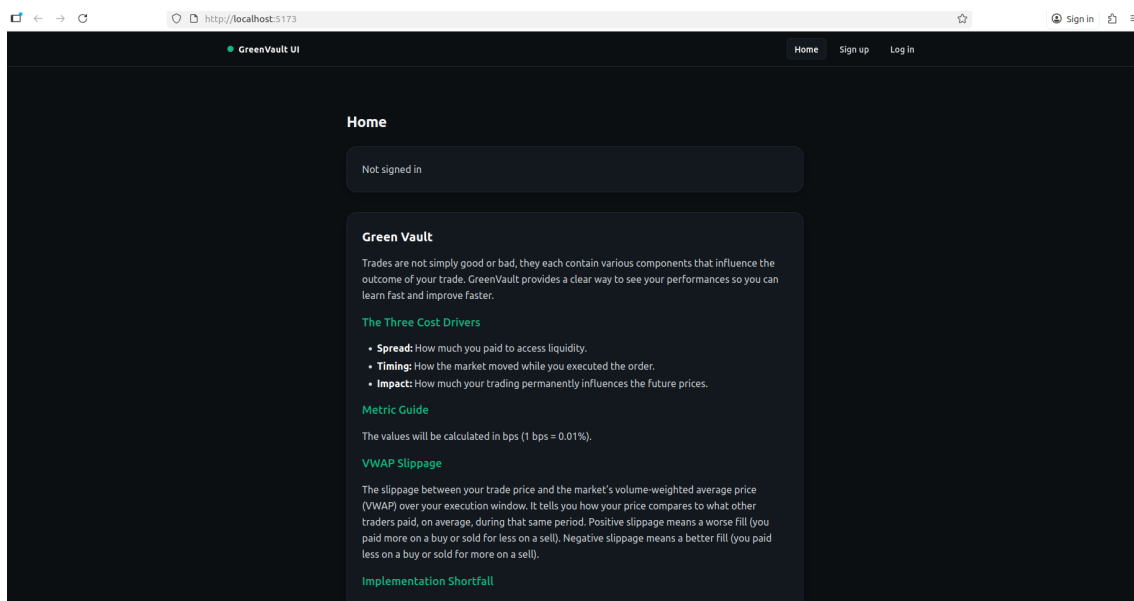
```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
green-vault-ui       latest             c1a7c8527156       About a minute ago 759MB
green-vault-analyzer latest             49ea304b7bcf       About a minute ago 767MB
green-vault-internal latest             918685d0929f       10 days ago        21.6MB
postgres             16                08d5fe3fa89c       13 days ago        451MB
cgr.dev/chainguard/postgres latest             45fff01d4e95       3 weeks ago        369MB
```

4. Startup Green Vault by running **docker compose up**, and wait for services to start (takes a few seconds).

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ docker compose up
```

```
rodrigo@rodrigo-ThinkPad-X13-Gen-4: ~/Desktop/InstallationGuide/Green-Vault
rodrigo@rodrigo-ThinkPad-X13-Gen-4:~/Desktop/InstallationGuide/Green-Vault$ docker compose up
[+] Running 5/5
 ✓ Network green-vault_default Created                                0.1s
 ✓ Container postgres-db Created                                    0.1s
 ✓ Container green-vault-internal-1 Created                        0.1s
 ✓ Container green-vault-analyzer-1 Created                       0.1s
 ✓ Container green-vault-ui-1 Created                             0.1s
Attaching to analyzer-1, internal-1, ui-1, postgres-db
postgres-db      PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres-db      2025-11-28 18:12:50.489 UTC [1] LOG:  starting PostgreSQL 18.0 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (W
postgres-db      olfi 15.2.0-r4) 15.2.0, 64-bit
postgres-db      2025-11-28 18:12:50.489 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgres-db      2025-11-28 18:12:50.489 UTC [1] LOG:  listening on IPv6 address "::", port 5432
postgres-db      2025-11-28 18:12:50.504 UTC [1] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
postgres-db      2025-11-28 18:12:50.518 UTC [32] LOG:  database system was shut down at 2025-11-28 18:10:31 UTC
postgres-db      2025-11-28 18:12:50.528 UTC [1] LOG:  database system is ready to accept connections
internal-1       2025/11/28 18:12:56 [✓] Database connected and migrations applied!
internal-1       2025/11/28 18:12:56 Go API listening on :8080
ui-1             npm warn Unknown builtin config "globalignorefile". This will stop working in the next major version of npm.
ui-1             npm warn Unknown builtin config "python". This will stop working in the next major version of npm.
ui-1             > dev
ui-1             > vite --host 0.0.0.0 --port 5173
ui-1             VITE v5.4.0  ready in 161 ms
ui-1             → Local:   http://localhost:5173/
ui-1             → Network: http://172.18.0.5:5173/
analyzer-1      INFO:  Started server process [1]
analyzer-1      INFO:  Waiting for application startup.
analyzer-1      INFO:  Application startup complete.
analyzer-1      INFO:  Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

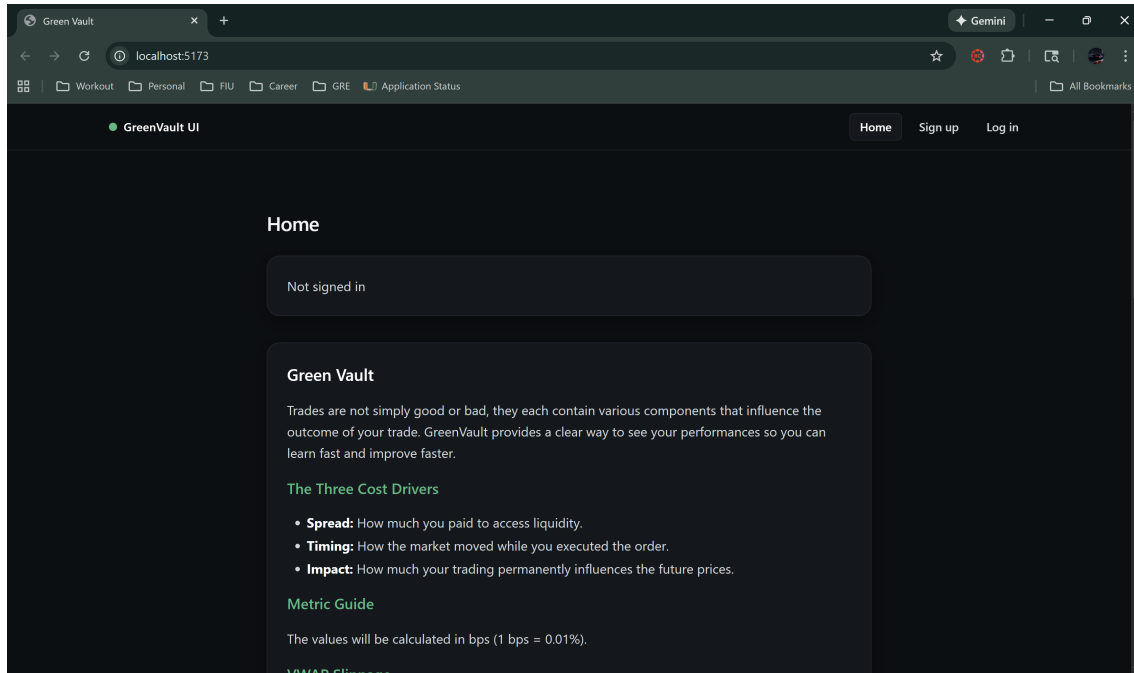
5. Open <http://localhost:5173/> and enjoy Green Vault!



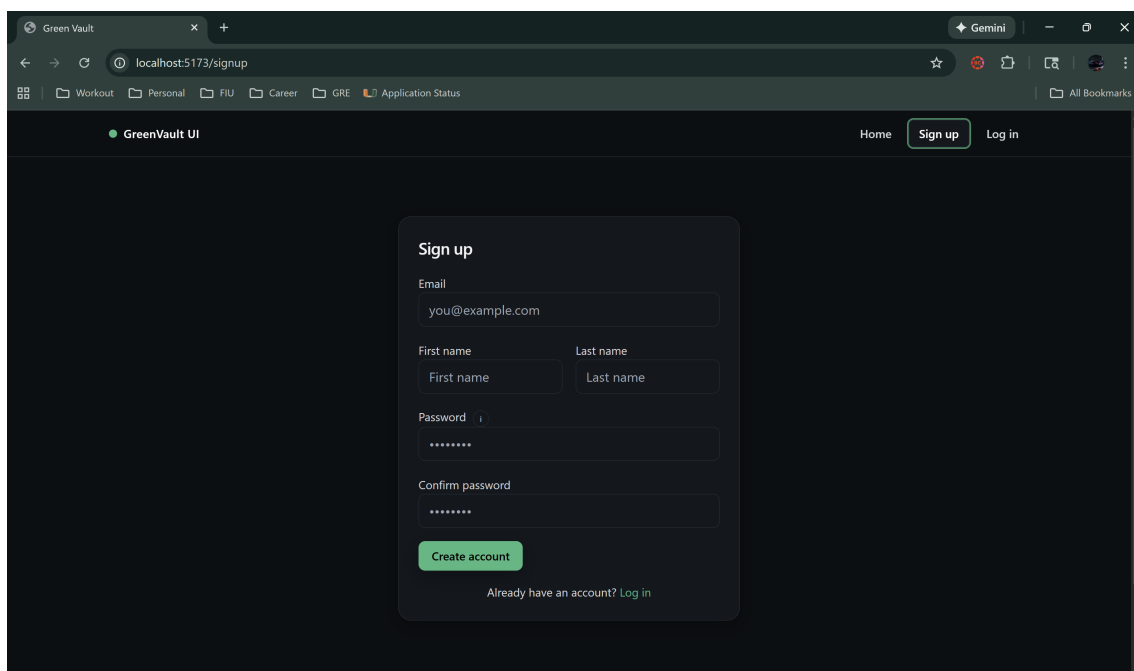
B User Manual

The walkthrough begins at the homepage as can be seen below. We will demonstrate through screenshots how to use our web application, our various features, where to find your stored data, and how to resolve error messages.

1. Begin at the homepage which initially sets your signed in status to “not signed in”. You may take your time on this page familiarizing yourself with our goal, what is being calculated, and terminology definitions.



2. Following is the signup page. Here you will be prompted to enter your information like email, first and last name, password, and a password confirmation. You will be provided with an info circle next to the password to disclose what types of characters your password must include. You are also given the option to press the login link below the “create account” button if you already have an account.



The screenshot shows a web browser window with the address bar displaying 'localhost:5173/signup'. The page title is 'Green Vault UI'. The navigation bar includes 'Home', 'Sign up', and 'Log in' links. The main content area features a 'Sign up' form with the following fields: 'Email' (containing 'you@example.com'), 'First name' (containing 'First name'), 'Last name' (containing 'Last name'), 'Password' (containing '*****'), and 'Confirm passw' (containing '*****'). A tooltip for the password field lists requirements: 'At least 12 characters', '1 uppercase letter', '1 lowercase letter', '1 number', and '1 symbol'. A green 'Create account' button is at the bottom of the form, and a link 'Log in' is provided for users who already have an account.

3. Entering invalid details will raise an error in the form of red boxes, and a text below explaining why the format provided was incorrect. Please take the time to fix these fields to successfully create your account.

The screenshot shows the same 'Sign up' form as the previous image, but with invalid input and error messages. The 'Email' field contains 'invalid' and has a red border with the message 'invalid format' below it. The 'First name' field contains '0' and has a red border with the message 'invalid characters' below it. The 'Last name' field contains '1' and has a red border with the message 'invalid characters' below it. The 'Password' field contains a single dot '.' and has a red border with the message 'too short (min 12 chars)' below it. The 'Confirm password' field also contains a single dot '.'. A green 'Create account' button is at the bottom of the form, and a link 'Log in' is provided for users who already have an account. At the bottom of the form, there is a red error message: 'The information you entered doesn't meet the requirements. Please fix the highlighted fields.'

The screenshot shows a web browser window with the URL `localhost:5173/signup`. The page title is "GreenVault UI". The navigation bar includes "Home", "Sign up", and "Log in". The "Sign up" form contains the following fields and messages:

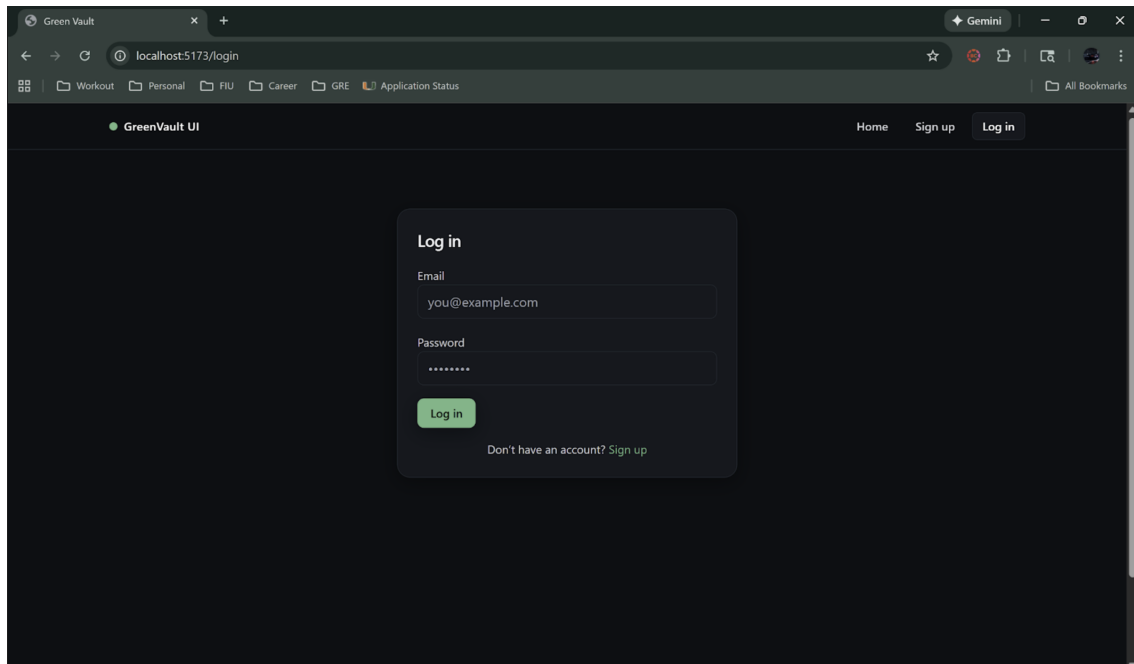
- Email:** `invalid` (highlighted in red)
- First name:** `0` (highlighted in red)
- Last name:** `1` (highlighted in red)
- Password:** (empty, highlighted in red)
- Confirm password:** (empty, highlighted in red)
- Message:** "Passwords do not match" (highlighted in red)
- Buttons:** "Create account" (green), "Already have an account? Log in" (blue)
- Error Message:** "Please fix the highlighted fields." (red)

4. Once you have provided valid inputs, you will receive a message confirming that your account is now successfully created. For this dummy account, I have chosen the password "Password123!@", which fulfills all password requirements. Please ensure to select a more secure password when you create an account.

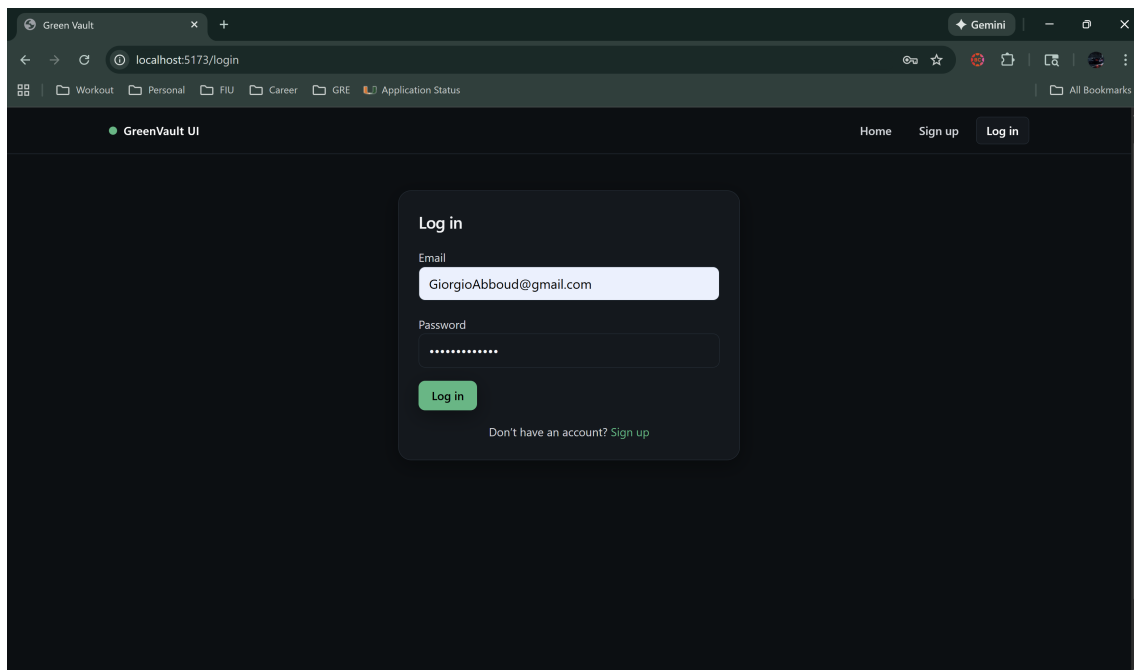
The screenshot shows the same web browser window with the URL `localhost:5173/signup`. The page title is "GreenVault UI". The navigation bar includes "Home", "Sign up", and "Log in". The "Sign up" form contains the following fields and messages:

- Email:** `GiorgioAbboud@gmail.com`
- First name:** `Giorgio`
- Last name:** `Abboud`
- Password:** (filled with dots)
- Confirm password:** (filled with dots)
- Buttons:** "Create account" (green), "Already have an account? Log in" (blue)
- Success Message:** "Your profile was successfully created!" (green)

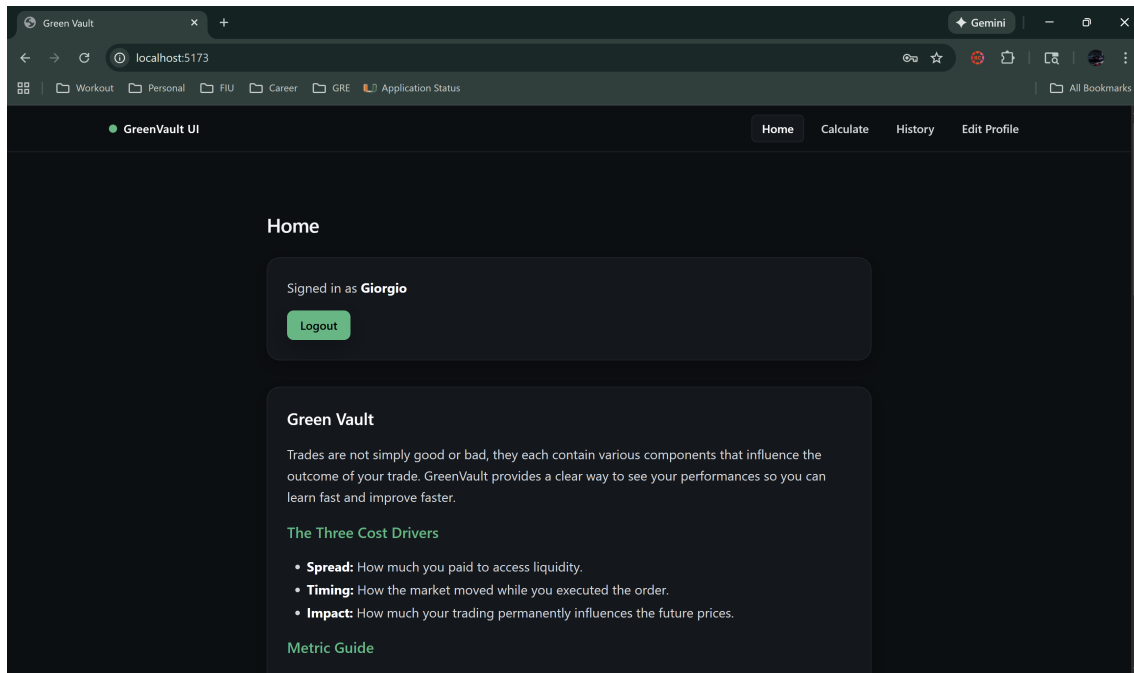
5. Now we may proceed to the login page where we are urged to enter our chosen email and password.



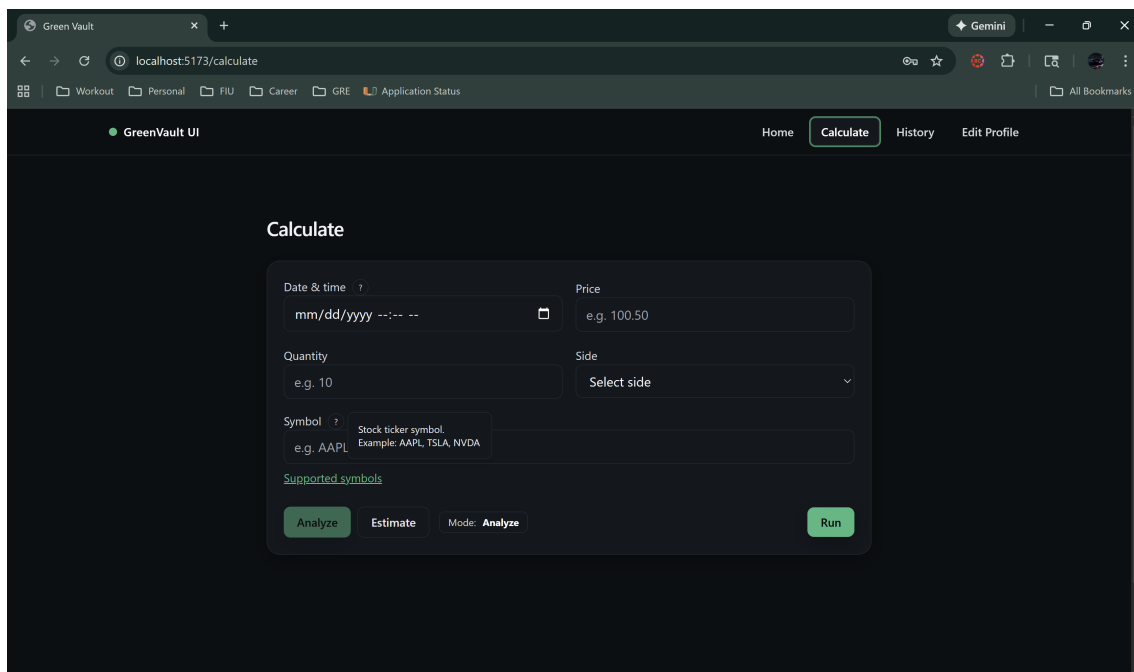
6. After entering your account details, press the login button and you will be redirected to the homepage. You also have the option to be redirected to the sign up page if you do not have an account created.

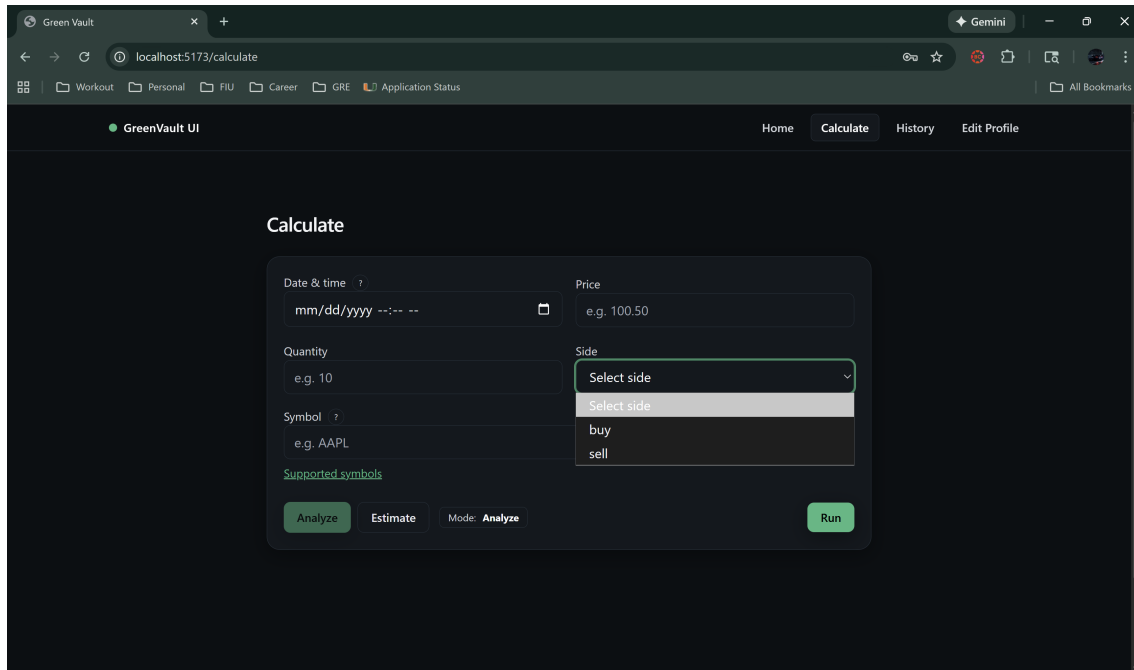
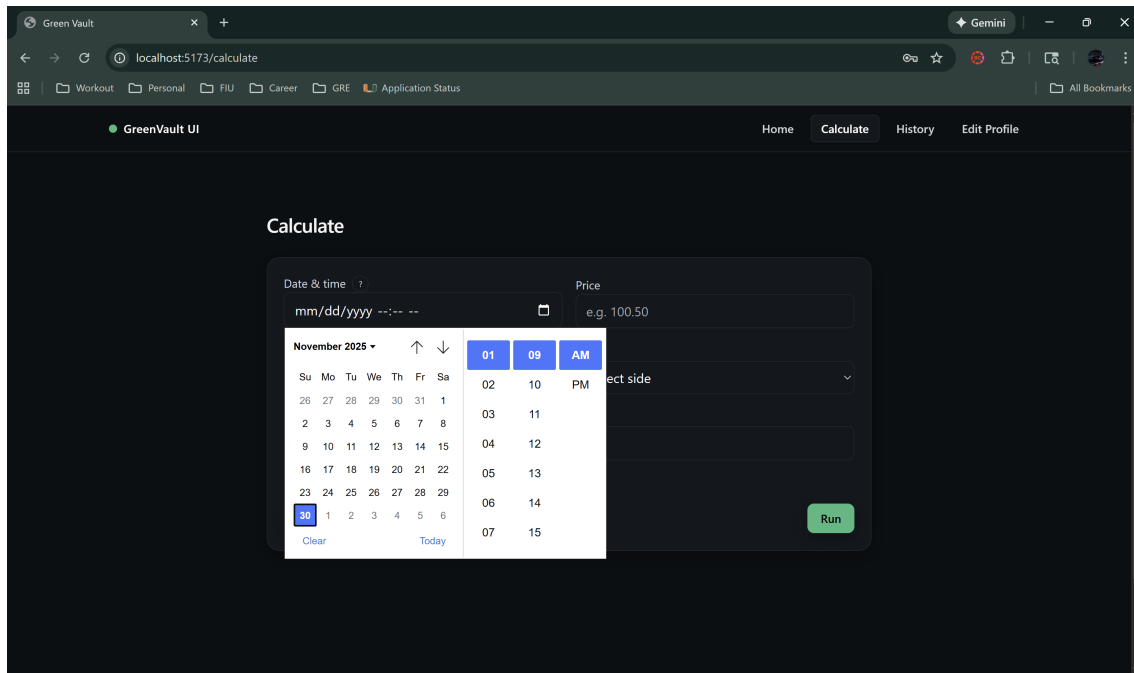


7. Now the homepage will change and show who is the currently logged in user. Now you can proceed to the main component of our web application by pressing the “Calculate” button on the top right corner.



8. Here is the calculate page, where you possess the tools to analyze your past trades, or estimate how well a trade now would do. You are prompted to enter a datetime, price, quantity, side (buy or sell), and a symbol. You are provided with examples to know in which format to enter your values. In this example, we are set to the analyze mode.





9. Pressing the “support symbols” link will send you to a NASDAQ page where you can search for valid symbols.

Stock Screener

Nov 30, 2025 - Market: Closed

Exchange

- ☐ NASDAQ
- ☐ Global Select
- ☐ Global Market
- ☐ Capital Market
- ☐ ADR
- ☐ NYSE
- ☐ AMEX

Market Cap

- ☐ Mega (>\$200B)
- ☐ Large (\$10B-\$200B)

Filtered by:
No filter applied

[Download CSV](#)

Symbol	Name	Last Sale	Net Change	% Change	Market Cap
NVDA	NVIDIA Corporation Common Stock	\$177.00	-3.26	-1.808%	4,301,100,000,000
AAPL	Apple Inc. Common Stock	\$278.85	1.30	0.468%	4,120,386,034,050
GOOGL	Alphabet Inc. Class A Common Stock	\$320.18	0.23	0.072%	3,863,612,060,000
GOOG	Alphabet Inc. Class C Capital Stock	\$320.12	-0.16	-0.05%	3,862,888,040,000
MSFT	Microsoft Corporation Common Stock	\$492.01	6.51	1.341%	3,656,804,130,037

Symbol Search

Search for a Symbol

[Screener](#)

[Cookies Settings](#) [Accept All Cookies](#)

10. Here we can see that I have entered valid inputs for each field and pressed the “Run” button to calculate the six metrics you saw in the homepage, and to return the result of your performance. There will also include an explanation, telling how each aspect of your trade performed and how you can improve.

GreenVault UI

Home [Calculate](#) [History](#) [Edit Profile](#)

Calculate

Date & time ? 11/13/2025 03:10 PM

Price 256.7

Quantity 15

Side buy

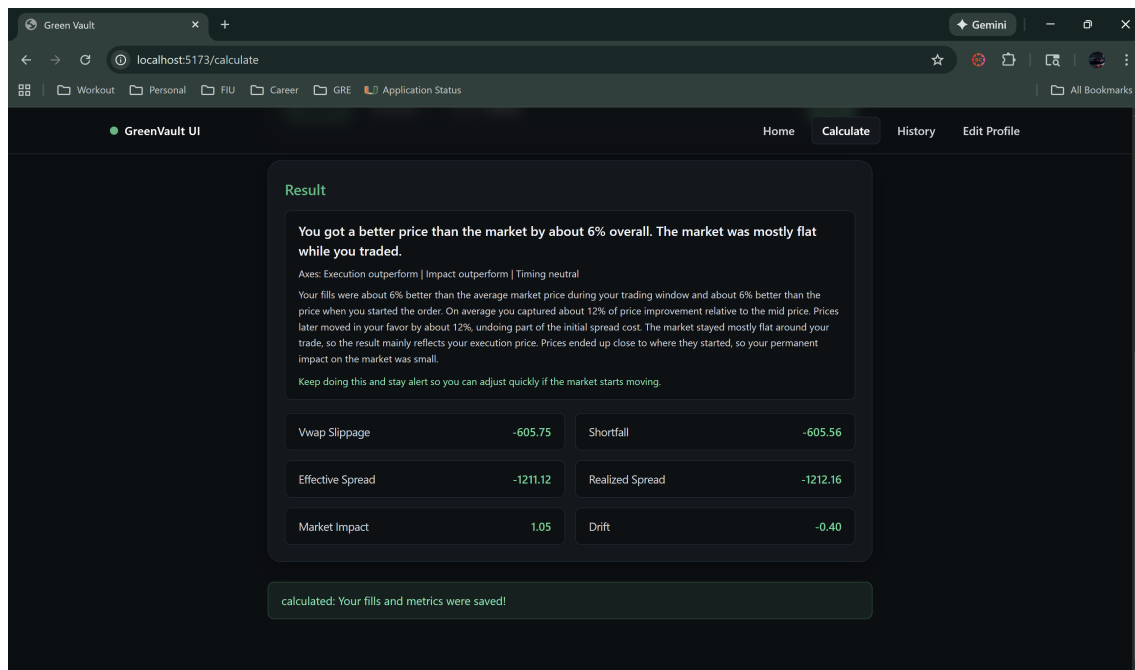
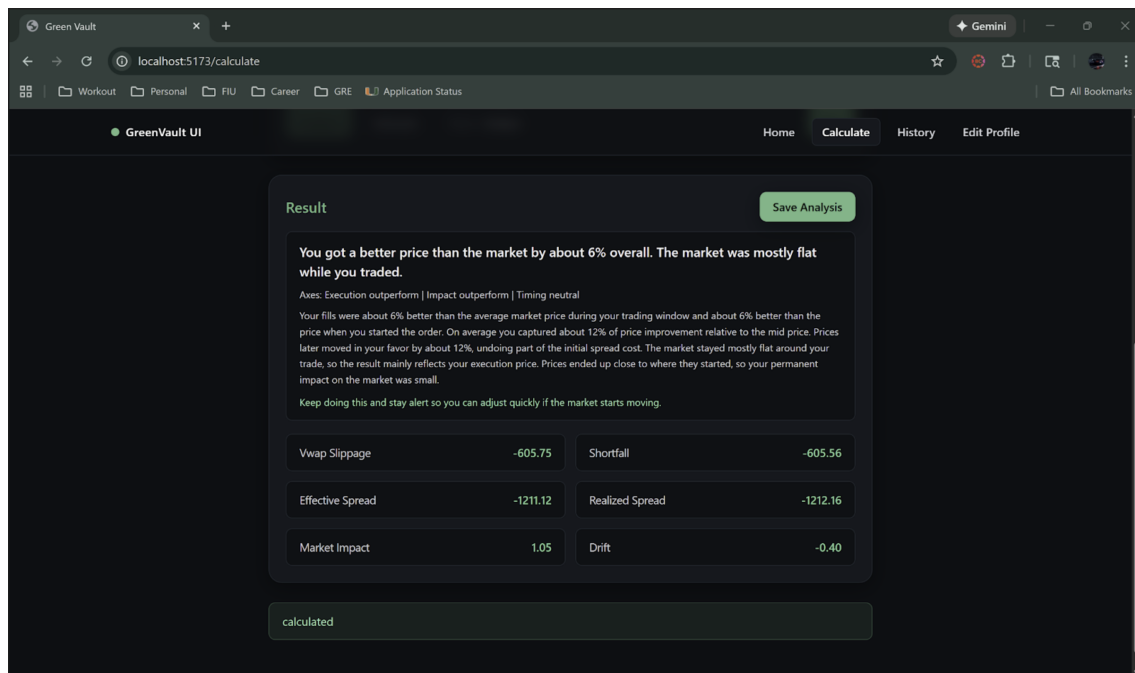
Symbol ? AAPL

[Supported symbols](#)

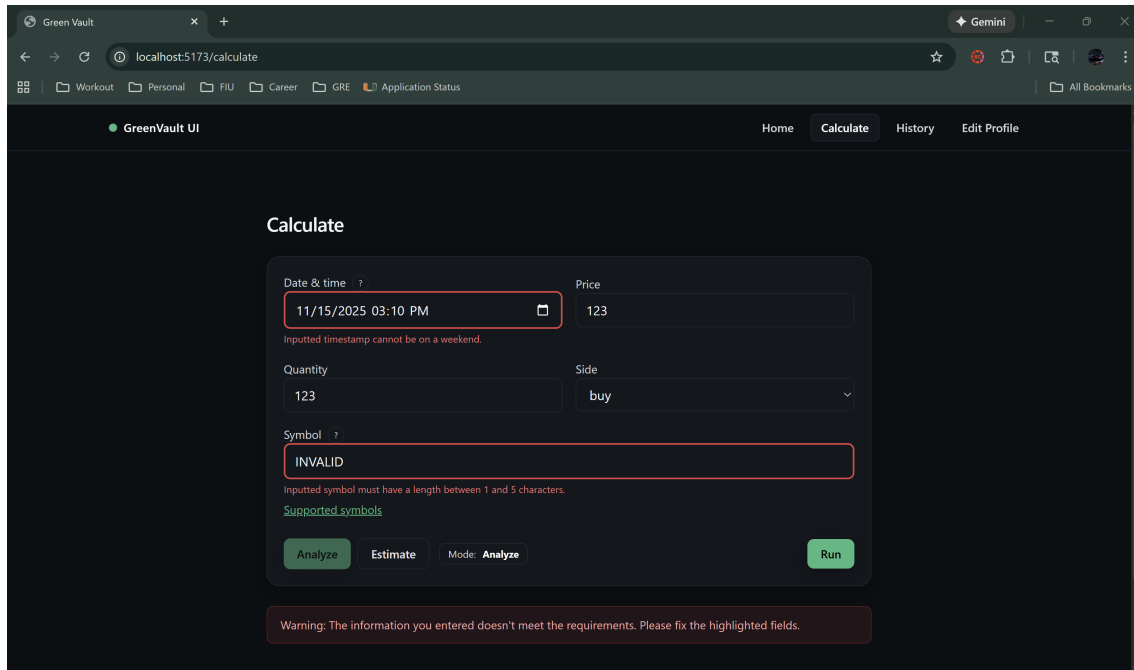
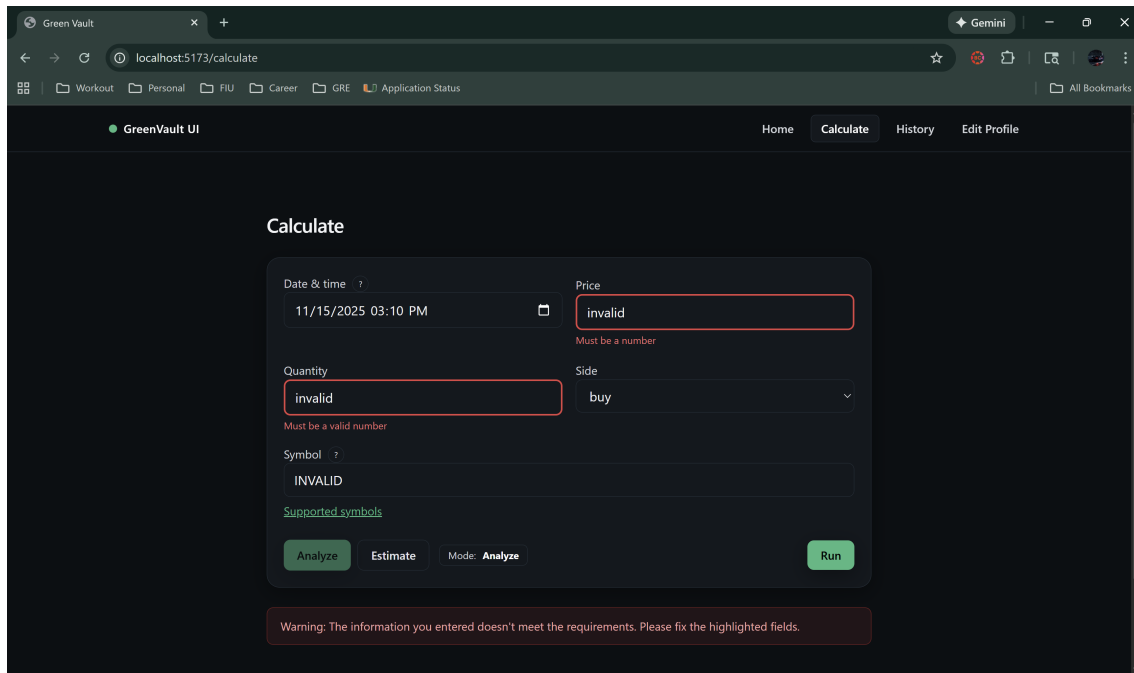
[Analyze](#) [Estimate](#) Mode: [Analyze](#) [Run](#)

Result [Save Analysis](#)

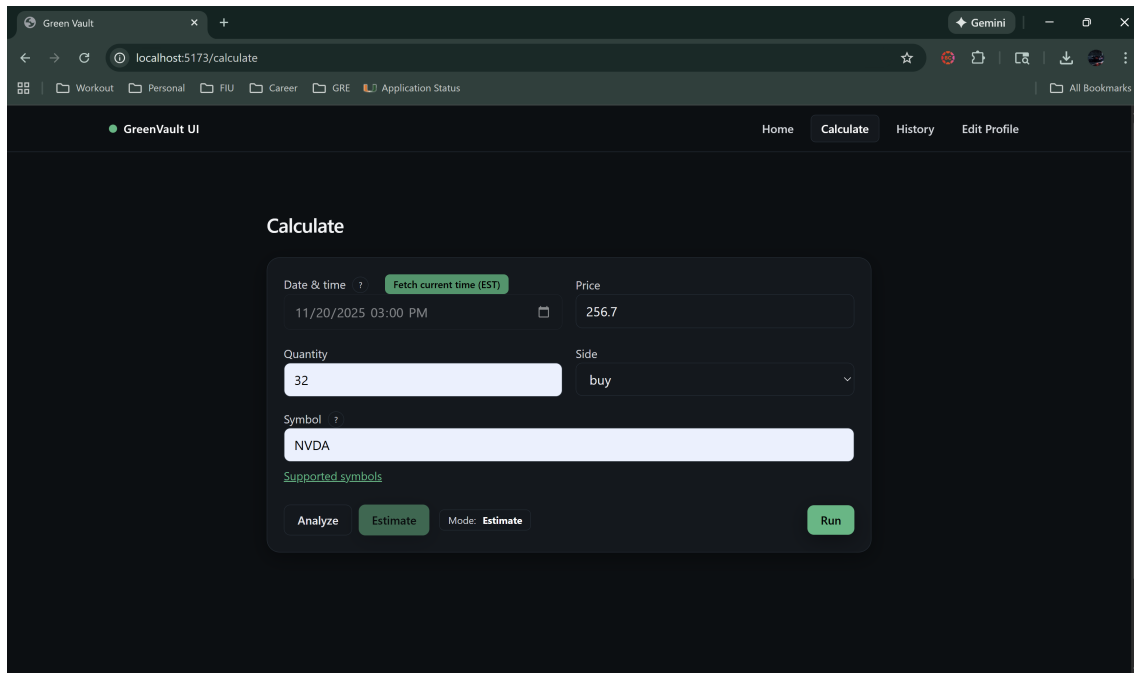
You got a better price than the market by about 6% overall. The market was mostly flat while you traded.



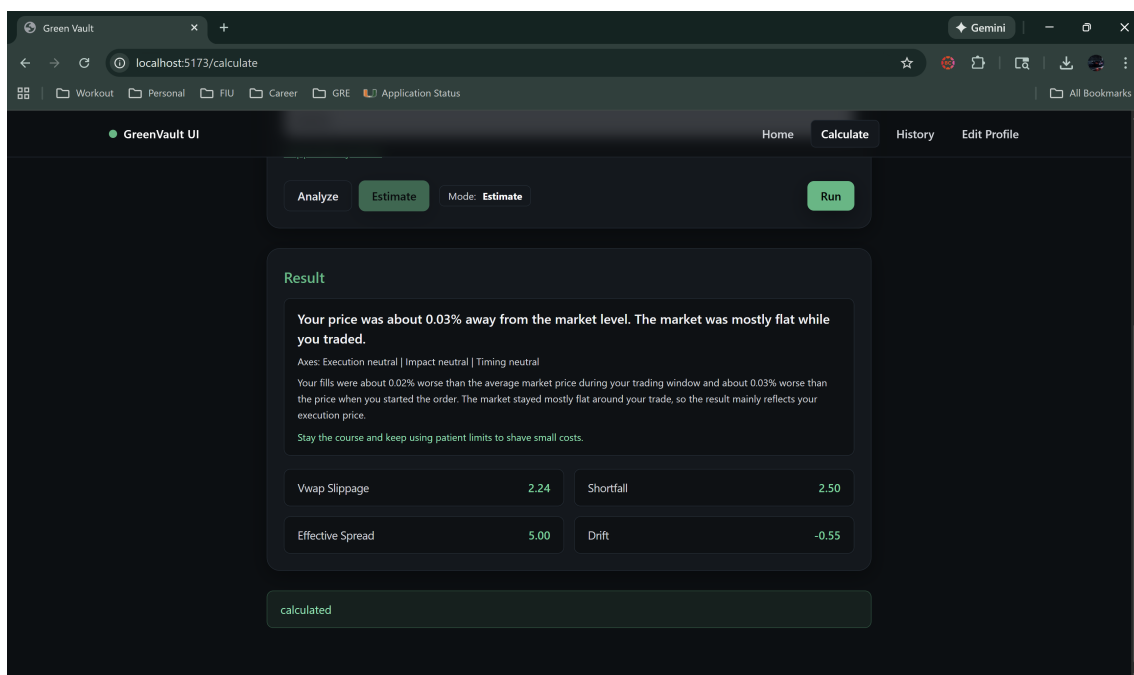
- Once again, invalid inputs will return errors and you will have to edit your inputs in the appropriate fields.



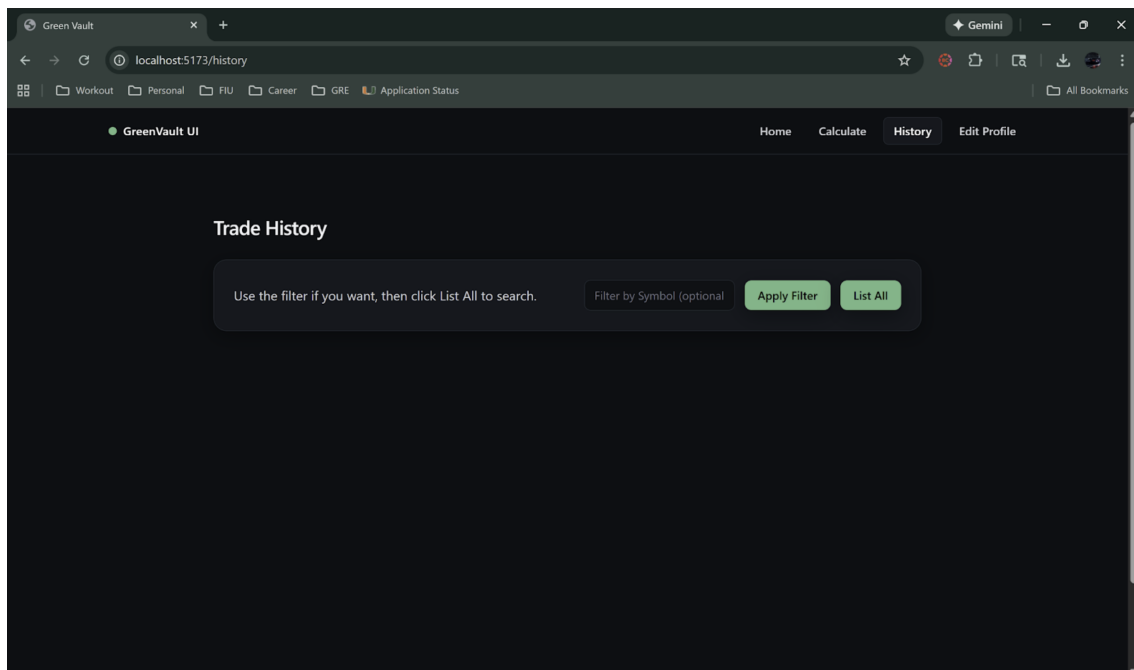
12. The estimate mode works similarly to the analyze mode in functionality and error handling. The two differences are that the output contains indeterminable fields, since they cannot be calculated, and you will also not have the ability to store your results.



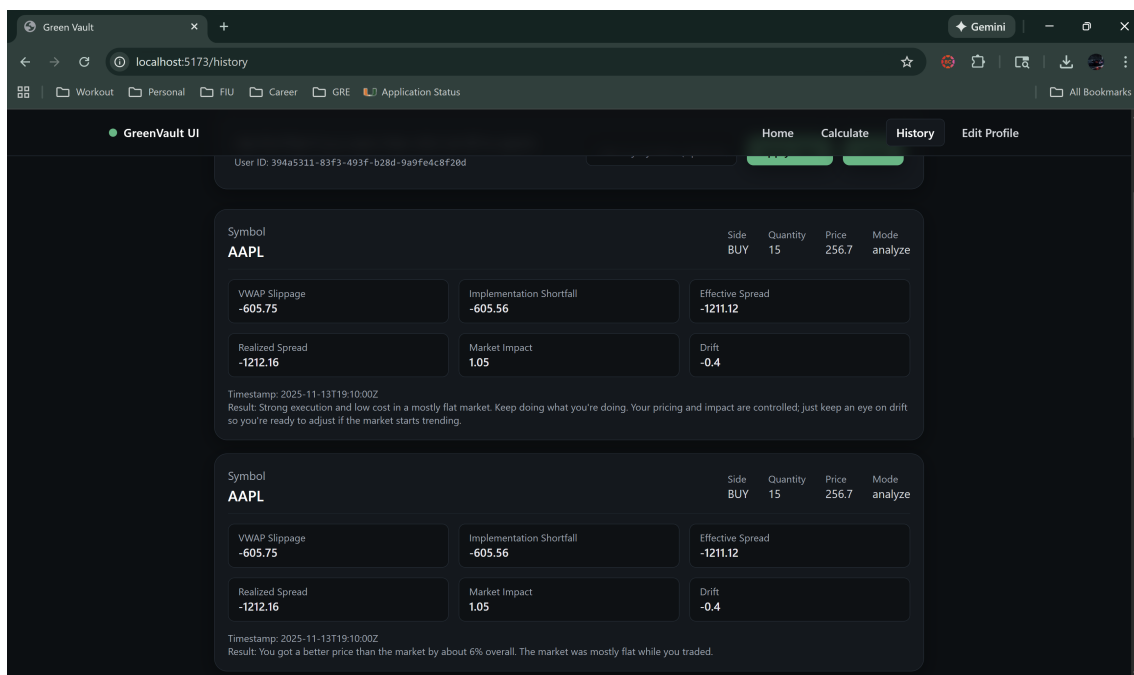
13. Once again the results will contain the bps of the calculated metrics and a review of your performance. These results cannot be stored in the database but will give you an approximation of how well you will perform by paying the typical price for your trade.



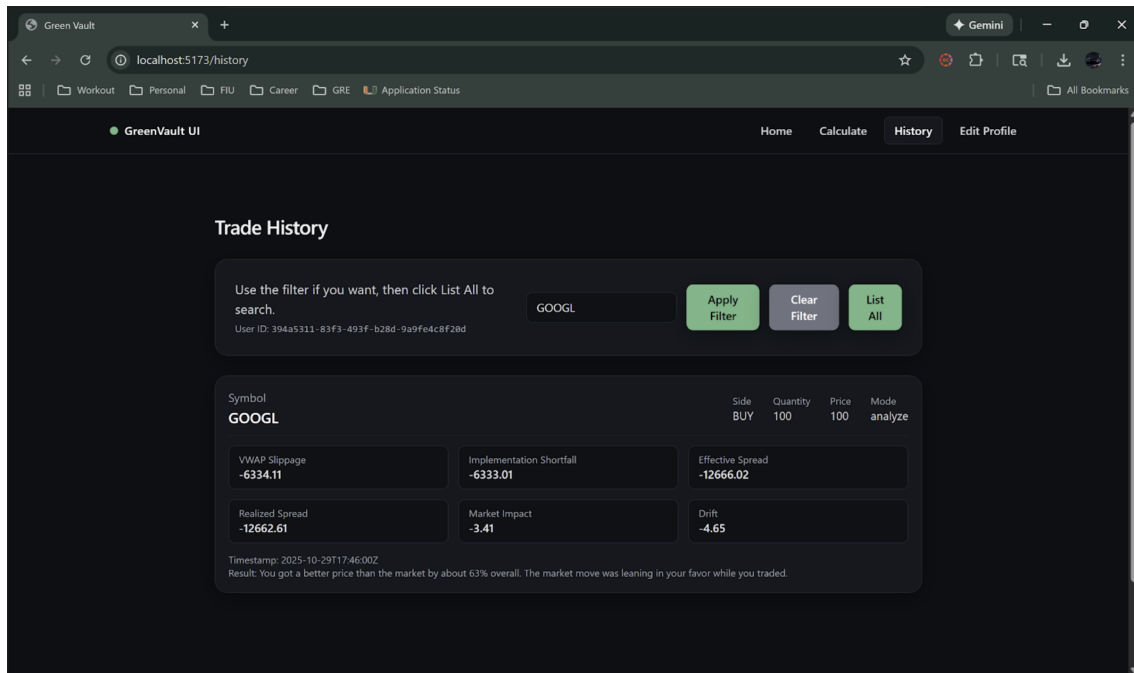
14. By heading to the top right corner and clicking on the “History” button, you will be redirected to the trade history page.



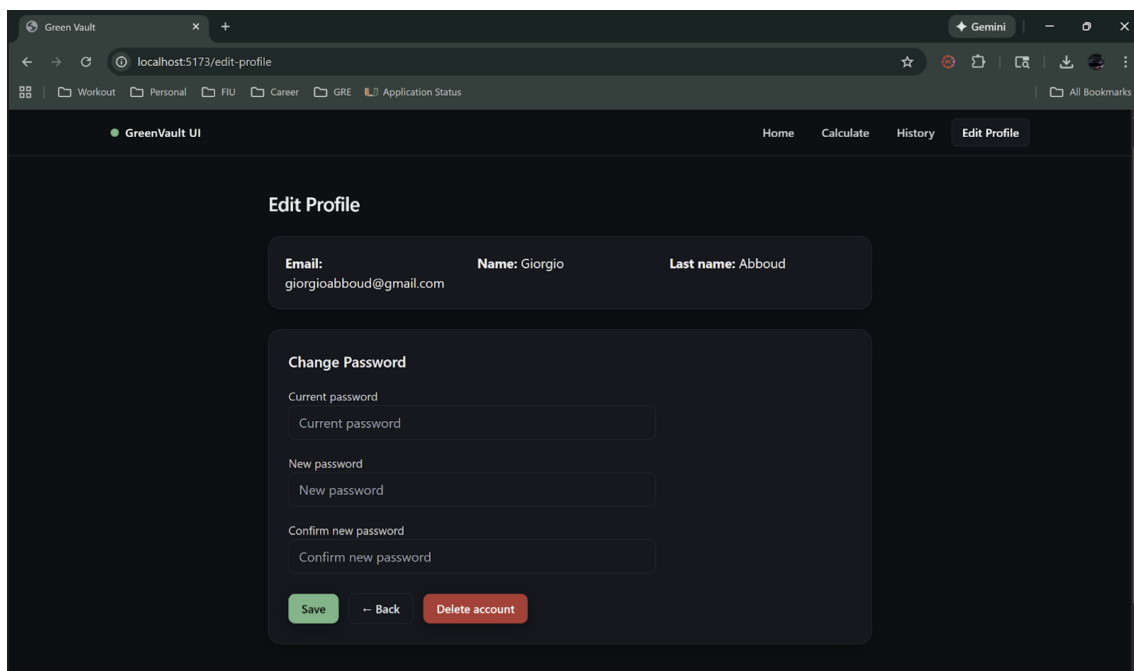
15. By pressing the “List all” button, you will be able to see the complete history of the trades you saved. This includes the calculated metrics, your inputs, and your performance’s review.



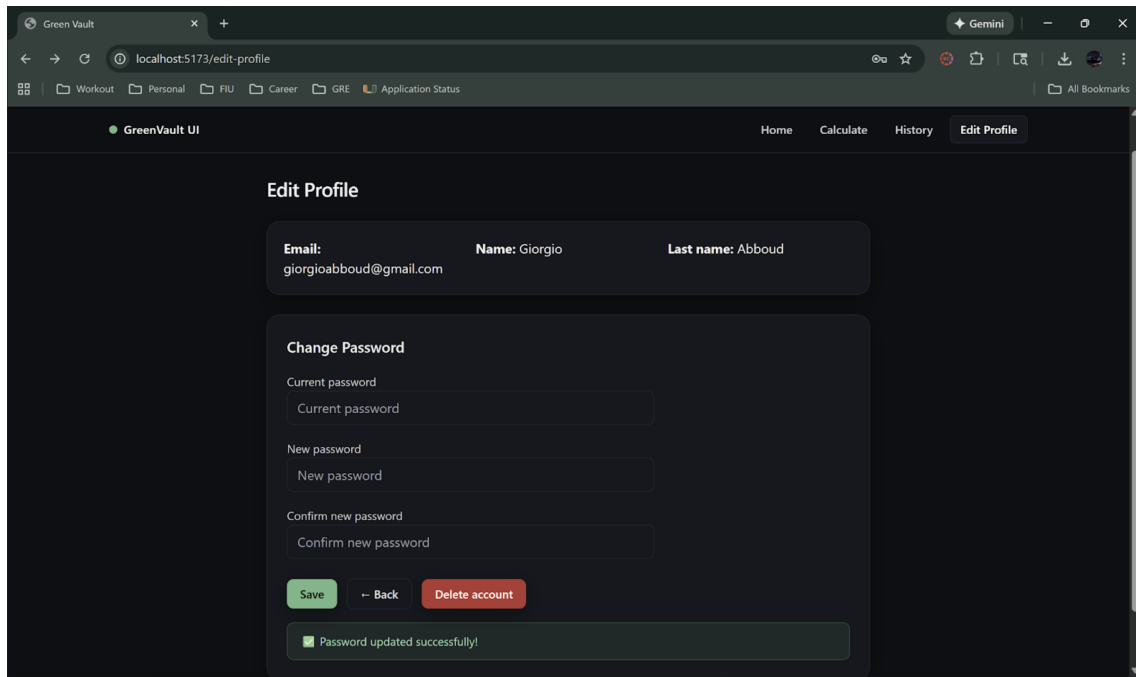
16. By inputting a symbol to filter by, you can now search for only specific stocks that you traded for. You always have the option to clear your inputted filter or apply a new one.



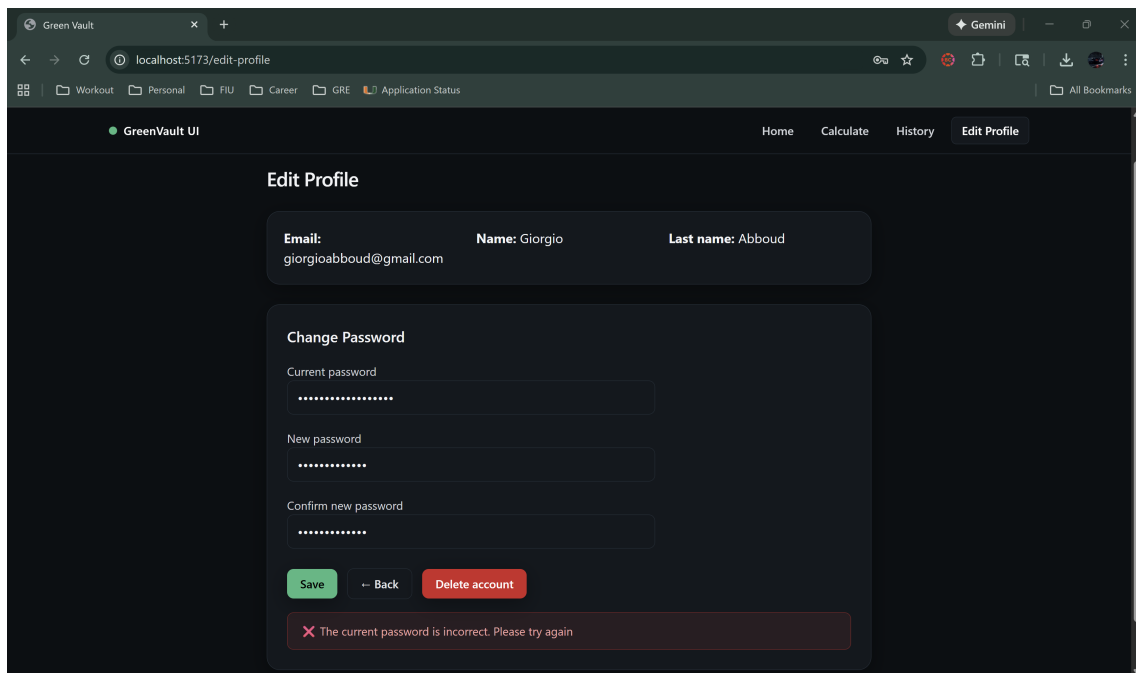
17. Going back to the top right, we see the “Edit Profile” button. Here you have the option to change your password or delete your account. You can also press the “Back” button to return to the homepage.

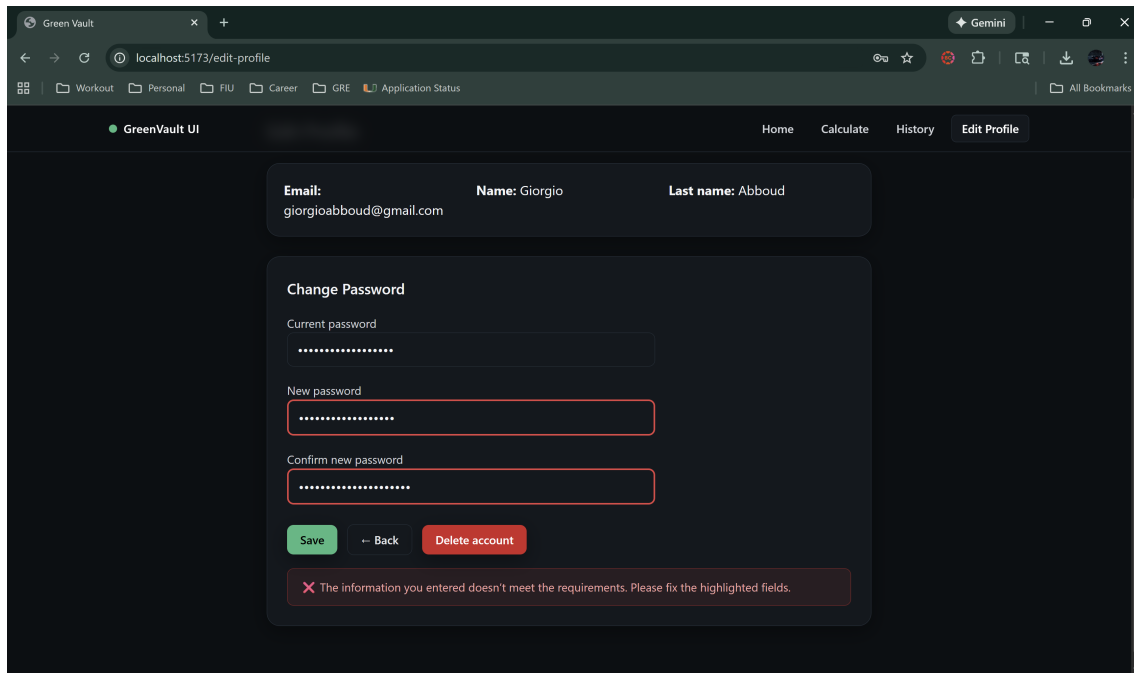


18. To change your password, simply enter your previous password, and enter your new password twice before pressing the “Save” button. This will automatically redirect you to the homepage once completed.

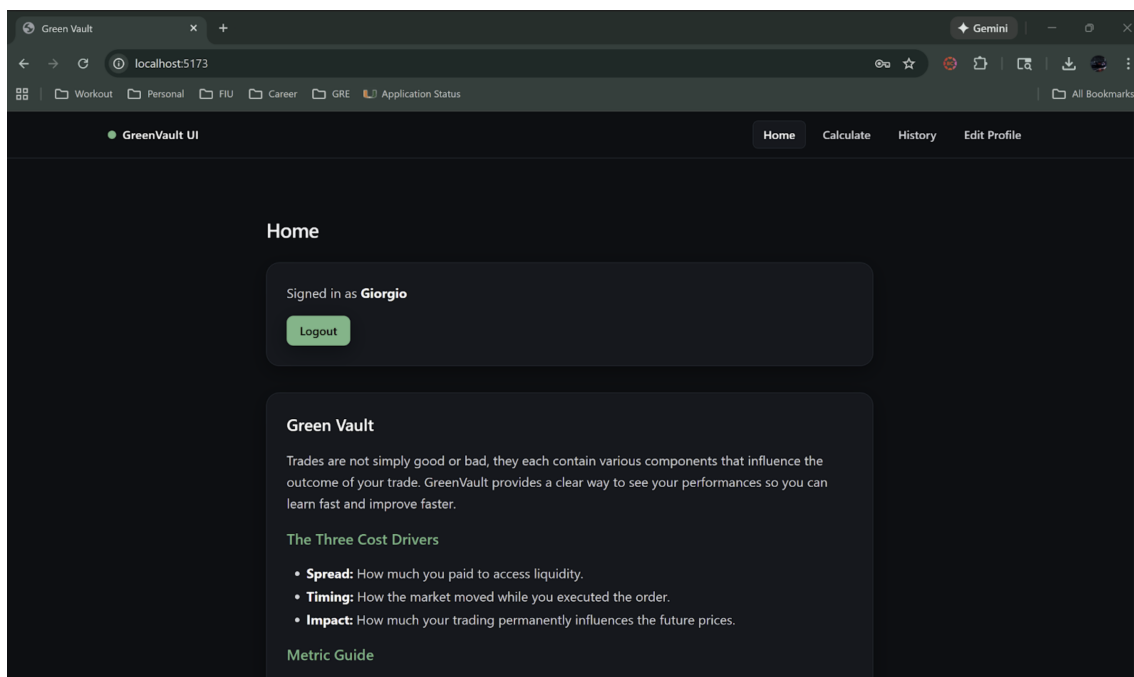


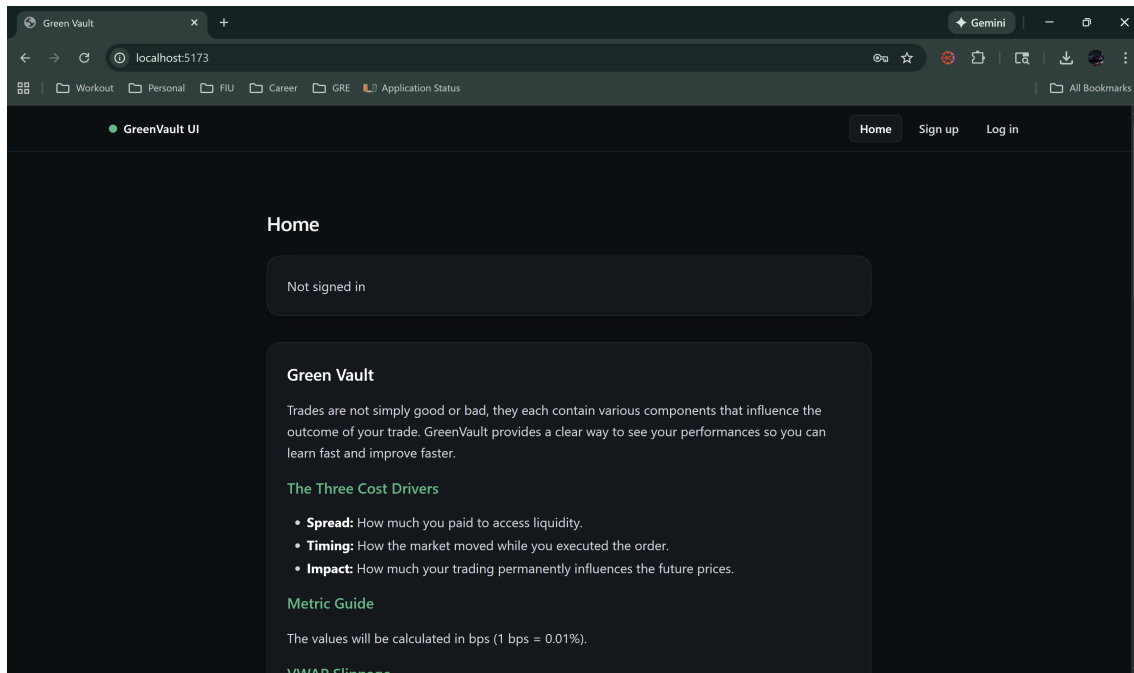
19. Failing to input your correct previous password or inputting contrasting new passwords will result in an error you must resolve.



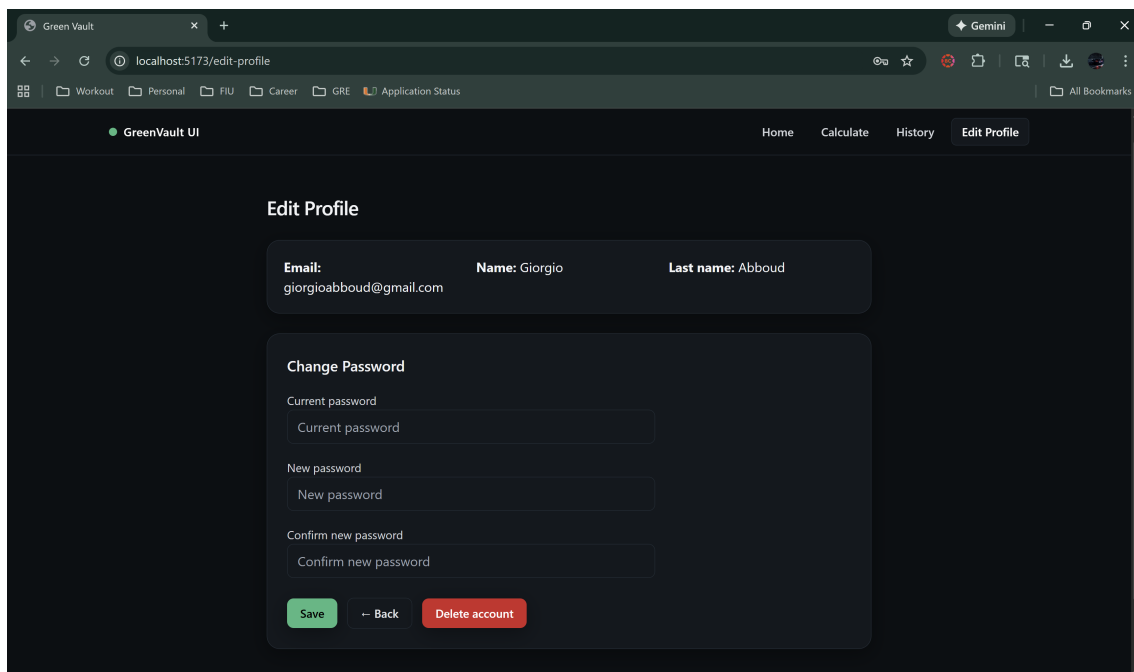


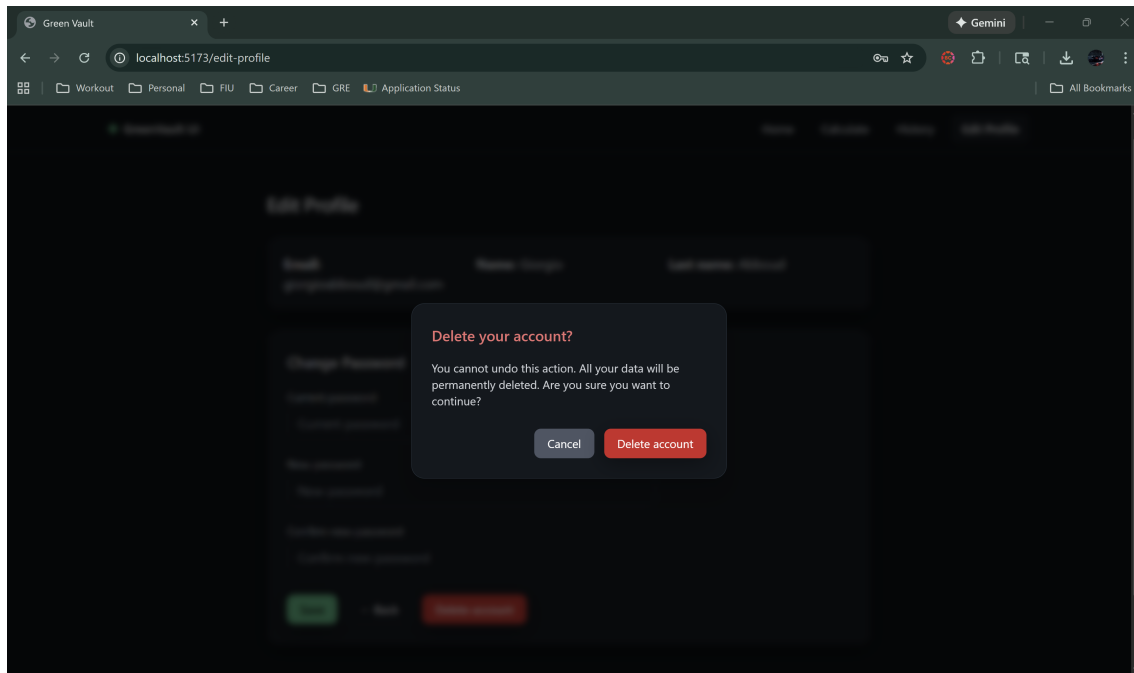
20. Back in the homepage, pressing the “Logout” button logs you out of your account. This will keep you in the homepage but you simply will not be logged in.



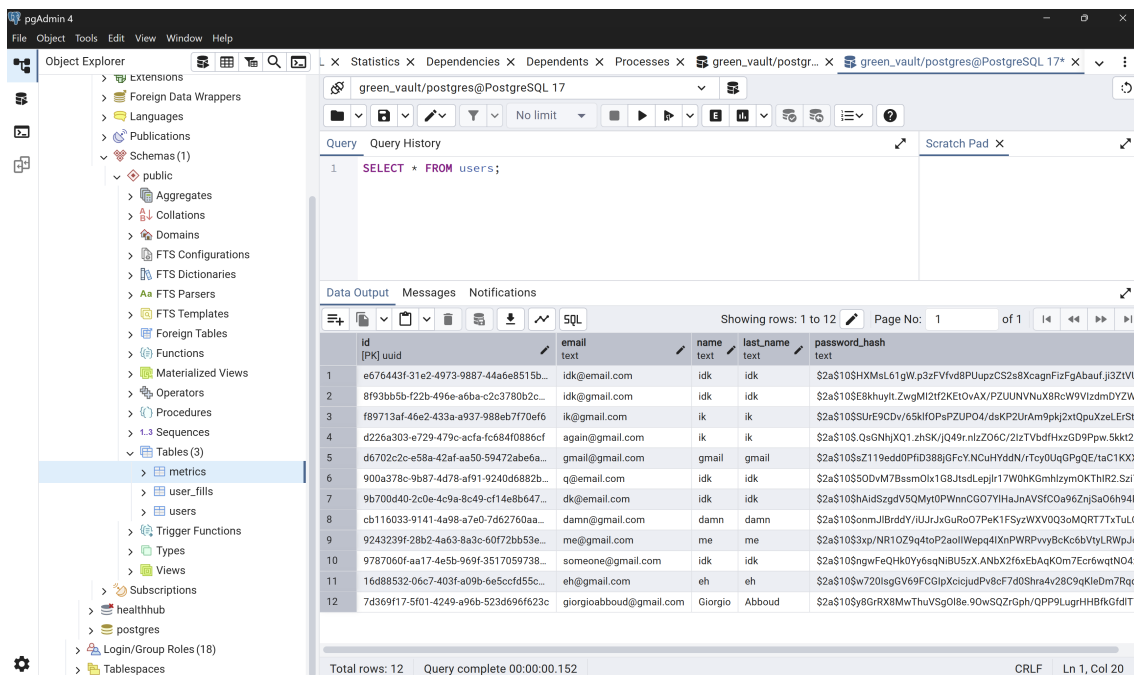


21. Should you want to delete your account, return to the edit profile page and select the “Delete account” button. A pop up will appear ensuring that you are confident in your decision. You can select “Back” to return to the edit profile page. Otherwise, press “Delete account” and you will be redirected to the login page.





22. To verify that your saved results in your database, you can open pgAdmin4, select Databases, your GreenVault database, Schemas, Public and Tables. From there you can use the SELECT clause to query for your data. Note: run **docker exec -it postgres-db psql -U postgres -d greenvault** in the terminal to access the DB if you don't have pgAdmin4.



pgAdmin 4 interface showing a query result for the 'metrics' table. The query is `SELECT * FROM metrics;`. The result shows 8 rows of data with columns: user_fill_id, vwap_slippage, shortfall, effective_spread, realized_spread, market_impact, and drift.

	user_fill_id	vwap_slippage	shortfall	effective_spread	realized_spread	market_impact	drift
1	-8718-ce91e62ca9...	5f51a240-3c62-468b-ac8d-e4e334192f4e	0	0	0	0	0
2	a403-09fd98895404	db72a5fa-86a3-4f60-b18b-c94573dfd153	0	0	0	0	0
3	-a516-faa515dfefca	04e4cf77-c33a-4854-9c34-57340e7832...	0	0	0	0	0
4	3e3-b03f5817d0c0	9b729117-7a28-4418-9db5-6752b2ed6...	0	0	0	0	0
5	b48f-6a67bfb0d68fc	6f98801c-d1bc-4935-b416-3d01d970e7...	0	0	0	0	0
6	-a2c6-cc9ad057b3ff	fab16d1c-479a-4ecf-8e8c-c3ddf05a56c5	-34832.05	-34835.82	-69671.64	-69678.42	6.78
7	-b2e2-c2b4a02b17...	f9ec9fae-02cf-4c52-a5c4-0d2b03441ae9	-8955.19	-8955.28	-17910.55	-17907.22	-3.33
8	-8041-f0a52eb3b1b9	ec7055f0-3d12-49db-95ee-2822a9f16258	3.63	2.5	5	0	2.4

Total rows: 8 Query complete 00:00:00.104

pgAdmin 4 interface showing a query result for the 'user_fills' table. The query is `SELECT * FROM user_fills;`. The result shows 8 rows of data with columns: KJ uuid, user_id, symbol, timestamp, price, side, quantity, mode, and result.

	KJ uuid	user_id	symbol	timestamp	price	side	quantity	mode	result
1	51a240-3c62-468b-ac8d-e4e334192f4e	e676443f-31e2-4973-9887-44a6e8515b...	AAPL	2025-10-10 14:07:00-04	123	buy	12	Analyze	SUCCESS
2	72a5fa-86a3-4f60-b18b-c94573dfd153	e676443f-31e2-4973-9887-44a6e8515b...	AAPL	2025-10-10 10:15:00-04	256.7	buy	1000	Analyze	SUCCESS
3	4e4cf77-c33a-4854-9c34-57340e7832...	e676443f-31e2-4973-9887-44a6e8515b...	AAPL	2025-10-10 10:15:00-04	256.7	buy	1000	Analyze	SUCCESS
4	729117-7a28-4418-9db5-6752b2ed6...	e676443f-31e2-4973-9887-44a6e8515b...	AAPL	2025-10-10 10:15:00-04	256.7	buy	1005	Analyze	SUCCESS
5	98801c-d1bc-4935-b416-3d01d970e7...	e676443f-31e2-4973-9887-44a6e8515b...	AAPL	2025-10-10 10:15:00-04	256.7	buy	1005	Analyze	SUCCESS
6	b16d1c-479a-4ecf-8e8c-c3ddf05a56c5	9243239f-28b2-4a63-8a3c-60f72bb53ed5	AAPL	2025-11-12 14:14:00-05	1233	buy	23	analyze	
7	ec9fae-02cf-4c52-a5c4-0d2b03441ae9	9243239f-28b2-4a63-8a3c-60f72bb53ed5	NVDA	2025-11-05 13:28:00-05	21	buy	32	analyze	
8	7055f0-3d12-49db-95ee-2822a9f16258	9243239f-28b2-4a63-8a3c-60f72bb53ed5	AAPL	2025-11-19 13:25:00-05	25	buy	321	estimate	

Total rows: 8 Query complete 00:00:00.086

C Admin/Operations Guide

C.1 Environment File Maintenance

C.1.1 Files to Maintain

- `/.env.db` → PostgreSQL credentials
- `internal/.env.docker` → Internal API DB URL + JWT signing key
- `ui/.env` → Vite server URLs for UI
- `analyzer/.env` → Twelve Data API key

C.1.2 Best Practices

- Rotate keys quarterly: update any `JWT_SIGNING_KEY`, database password, or API key every 90 days.

- Never commit real secrets: only store `.env.example` (no values) in git history.
- File permissions: allow read/write only for the owner for all env files. This blocks access from other users/app on the system

C.2 Deploying and Rotating API Keys Safely

C.2.1 Twelve Data API Key (`analyzer/.env`)

- When updating the Twelve Data API key:
 - Log in to the official dashboard of Twelve Data.
 - Generate a new key, set expiration if available.
 - Update `analyzer/.env` with the new value.
 - Restart analyzer service via Docker: **`docker compose restart analyzer`**
- If key is exposed or leaked:
 - Immediately revoke it in Twelve Data dashboard.
 - Generate a replacement and update the `.env` file.

C.2.2 JWT Compromise (`internal/.env.docker`)

- Symptom: 401 spikes, suspicious internal API calls, or reports of stolen token
- When updating JWT Key:
 - Open `internal/.env.docker`
 - Replace `JWT_SECRET=NEW_SECURE_RANDOM`
 - Generate a new secure key with openssl: **`openssl rand -hex 32`**
 - Restart all services: **`docker compose restart`**
- If key is stolen:
 - Treat it as an active incident:
 - Rotate signing key immediately.
 - Invalidate all issued tokens if your API has a logout/denylist strategy.
 - Communicate to the team via your incident channel.

C.3 Services Images

C.3.1 Images Versions

All services employ the latest images from Chainguard for each service. This could cause some issues in the future as Chainguard provides more updates. The current setup successfully runs with no dependency issues:

- Golang 1.25.4
- Python 3.14.0
- Node 25.2.1
- Postgres 18.1

C.3.2 Minimal Images

- All services (Internal, Analyzer, UI, and Postgres DB) run on minimal & secure images.
- Meeting 0 Common Vulnerability Exposures (CVEs), keeping the system free of flaws and weaknesses.

C.4 Credentials, Backup, and Restore Postgres

C.4.1 Credentials

- Update all user, password, and db name in `/.env.db` if change of db is required

C.4.2 Backup

- Run to create a compressed export file you can restore from:
`docker compose exec insert bash -c "pg_dump $POSTGRES_DB -U $POSTGRES_USER -Fc -f /backup/green-vault_latest.dump"`

C.4.3 Restore

- Run to restore:
`docker compose exec insert bash -c "pg_restore -d $POSTGRES_DB -U $POSTGRES_USER /backup/green-vault_latest.dump"`

C.4.4 Tips

- Store backups in a mounted volume like:

```
volumes:
  backup:
    driver: local
```

C.5 Common On-Call & Quick Fixes

Scenario	Action
API key leaked	Revoke and regenerate key in provider dashboard, update <code>/.env</code> , then restart analyzer service
Internal JWT stolen	Rotate signing key (<code>env.docker</code>) and restart stack
DB credentials need update	Update <code>env.db</code> and restart stack
Services not responding	Check logs using <code>docker compose logs -f</code> , then restart services with <code>docker compose restart</code>
Backups failing	Validate mounted <code>/backup</code> volume and container permissions

Table 4: Green Vault - Common Scenarios & Fixes

D API Reference

D.1 Twelve Data API

<https://twelvedata.com/docstime-series>

Sample output:

```
{
  "meta": {
    "symbol": "AAPL",
    "interval": "1min",
    "currency": "USD",
    "exchange_timezone": "America/New_York",
    "exchange": "NASDAQ",
    "mic_code": "XNAS",
```

```
    "type": "Common Stock"
  },
  "values": [
    {
      "datetime": "2021-09-16 15:59:00",
      "open": "148.73500",
      "high": "148.86000",
      "low": "148.73000",
      "close": "148.85001",
      "volume": "624277"
    }
  ],
  "status": "ok"
}
```

E Poster



Knight Foundation
School of Computing
and Information Sciences

Knight Foundation School of Computing and Information Sciences

Fall 2025 Capstone 2 Project



GREEN VAULT

Green Vault

Students: Giorgio Abboud, Malik Dagher, Audrey Gamero, Daniella Lacotera, Rodrigo Munoz
Instructor/Faculty: Dr. Masoud Sadjadi / Associate Professor, Florida International University

PROBLEM

The main problem Green Vault solves is that most traders, especially beginners and intermediate-level users, receive a trade fill but have no clear way to understand whether their execution was good or bad. It leaves everyday traders without guidance on slippage, timing, or how market movement impacted their trade.

Our solution takes the provided user trader and outputs a clear analysis. Our application fetches 1-minute Open, High, Low, Close, and Volume (OHLCV) market data from a Twelve Data API and computes hidden metrics, such as Volume-Weighted Average Price (VWAP) slippage, shortfall, effective and realized spreads, impact, and drift. These are then used to evaluate the user's performance and output a clear review with explanations.

CURRENT SYSTEM

Most traders today do not measure their execution quality in any structured way. The majority of traders make a trade and check their fill price and what it was around the time they traded. Experienced traders may check the VWAP for some of their trades, but it is time-consuming and often insufficient. On the other hand, large institutions use expensive Transaction Cost Analysis tools to compute detailed metrics. These are often inaccessible and too detailed for most traders. As a result, there is not a consistent way for traders to analyze and understand their performances.

REQUIREMENTS

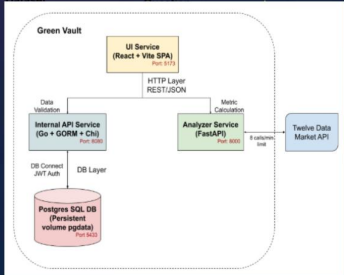
Functional:

- GO handles auth, account creation, updates, deletion, and data storing in the database.
- Analyzer accepts and verifies user inputs, and returns errors if necessary.
- UI handles errors and outputs them in a handled verbose response.

Nonfunctional:

- Service always available as long as the trades are performed within valid hours.
- User data remains local and private.
- Users authenticated with JWT tokens, no exposure of keys, and credentials are safe.
- 0 Common Vulnerability Exposures (CVEs) from Chainguard Images usage.

SYSTEM DESIGN



The diagram illustrates the architecture of Green Vault. It shows a UI Service (React + Vite SPA) connected to an Internal API Service (Go + GORM + Chi) via HTTP Layer (REST/JSON). The Internal API Service is connected to a PostgreSQL DB (Persistent volume pgdata) via DB Layer (JWT Auth). The Internal API Service also connects to an Analyzer Service (FastAPI) via Metrics Connection. The Analyzer Service connects to a Twelve Data Market API via Evaluation API.

OBJECT DESIGN

- UI: Displays all application features. (Malik)
- Internal: API Auth validation and database insertions. (Rodrigo & Daniella)
- Analyzer: Calculates metrics and determines performance. (Giorgio & Audrey)
- Database: Stores user data. (Rodrigo & Daniella)



IMPLEMENTATION

Front UI:

- React
- Vite
- TailwindCSS
- React Router

Programming Language:

- python

Analytics Service:

- Python
- Twelve Data API

Auth & Internal Backend:

- Go
- Chi
- GORM
- JWT
- PostgreSQL
- REST/JSON
- testcontainers

Persistence:

- Docker Volume for Postgres

Images:

- Chainguard Images (Golang, Python, Node, Postgres)

System Integration:

- Docker Compose



VERIFICATION

- The internal API service validates routing, authentication, and database persistence via Go Tests, serving as the system's sole gateway to PostgreSQL to ensure data integrity and durability.
- The analyzer service uses a rate-limited client and pytest to validate its analytics workflows, operating as a stateless component intentionally isolated from all database access.



SUMMARY

Green Vault addresses the lack of accessible execution-quality tools by providing clear trade analysis for beginners and deeper metrics for intermediate users. The platform ingests a user's trade, retrieves relevant market data, computes execution metrics like slippage and spreads, and stores results through a structured backend designed for secure authentication, validation, and data management.

REFERENCES

- Hayes, Adam. "Implementation Shortfall: Meaning, Examples, Shortfalls." Investopedia, Investopedia, www.investopedia.com/terms/i/implementation-shortfall.asp. 23 Dec. 2023.
- Hu, Edwin, et al. "Tick Size Pilot Plan and Market Quality." Tick Size Pilot Plan and Market Quality, 31 Jan. 2018, www.sec.gov/files/marketstructure/research/dora_wp_tick_size_market_quality.pdf.
- Langager, Chad. "Basis Points: Understanding What They Are and How They Are Used." Investopedia, Investopedia, www.investopedia.com/ask/answers/what-basis-point-bps/. 26 Nov. 2024.
- Makrides, Demetris. "What Is Slippage in Trading? - Definition." Quadcode, 18 Dec. 2024, quadcode.com/blog/what-is-slippage.
- Oanda. Seven Moving Average Price Options for Advanced Trading Analysis. OANDA, 24 Sept. 2024, propttrader.oanda.com/en/lab-education/trading-knowledge/technical-analysis/exploring-seven-moving-average-price-options-for-advanced-analysis/.
- "Spread and Price Impact." Fxds, fxds.io/measures/spread_and_price_impact/. Accessed 29 Nov. 2025.
- Stephanie Wu. "Market Impact Models." Pretty Quant, Pretty Quant, 3 Sept. 2022, www.prettyquant.com/post/2022-09-03-market-impact-models/.



**Engineering
& Computing**

ACKNOWLEDGEMENT

The material presented in this poster is based upon the work supported by the Knight Foundation School of Computing and Information Sciences, Florida International University. We thank Dr. Masoud Sadjadi for his guidance throughout this project.

FLORIDA INTERNATIONAL UNIVERSITY

F Scrum Evidence

Sprint Planning Meeting Minutes - September 2

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **22 story points** per sprint. It was decided that the team would focus on the higher priority sprints during the first week, and completing the smaller user stories the following week.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- **User Story 1:** Research data sources/fields to compute spread cost (effective spread), market impact, timing/drift, and optionally opportunity cost.
- **User Story 2:** Compare 1, 5, and 15 minute stock window bars.
- **User Story 3:** Define output metrics and UI fields.
- **User Story 4:** Finalize formulas for hidden-cost metrics.
- **User Story 5:** Research service split design (multiple localhost ports).
- **User Story 6:** Research python backend framework and libraries, API contract, frontend routing.
- **User Story 7** Research frontend (React + Tailwind) architecture, backend routing, and user flow.
- **User Story 8** Research database selection and schema design.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**
 - **User Story 1:** Research data sources/fields to compute spread cost (effective spread), market impact, timing/drift, and optionally opportunity cost.
 - **User Story 4:** Finalize formulas for hidden-cost metrics.
 - **User Story 6:** Research python backend framework and libraries, API contract, frontend routing.
- **Malik Dagher**
 - **User Story 2:** Compare 1, 5, and 15 minute stock window bars.
 - **User Story 3:** Define output metrics and UI fields.
 - **User Story 7** Research frontend (React + Tailwind) architecture, backend routing, and user flow.
- **Audrey Gamero**
 - **User Story 1:** Research data sources/fields to compute spread cost (effective spread), market impact, timing/drift, and optionally opportunity cost.

- **User Story 6:** Research python backend framework and libraries, API contract, frontend routing.
- **Daniella Lacotera**
 - **User Story 3:** Define output metrics and UI fields.
 - **User Story 7** Research frontend (React + Tailwind) architecture, backend routing, and user flow.
- **Rodrigo Munoz**
 - **User Story 5:** Research service split design (multiple localhost ports).
 - **User Story 6:** Research python backend framework and libraries, API contract, frontend routing.
 - **User Story 8** Research database selection and schema design.

Sprint Planning Meeting Minutes - September 15

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **22 story points**. It was decided to continue with this number of user stories as the team had enough time to finish the primary objectives, as well as ask further questions and conduct additional research afterwards.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- **User Story 1:** Database connectivity and migrations.
- **User Story 2:** Confirm developer machines connect to the database and can run the docker compose.
- **User Story 3:** Confirm that data persists permanently in the database, and temporarily in the broker.
- **User Story 4:** Compute the hidden cost metrics for testing purposes with dummy data.
- **User Story 5:** Conduct unit tests and create drivers (TDD testing)
- **User Story 6:** Create the frontend UI to accept user data and return feedback.
- **User Story 7:** Insert a feedback option to receive user comments.
- **User Story 8:** Backend accepts user fills and performs computations.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**
 - **User Story 2:** Confirm developer machines connect to the database and can run the docker compose.
 - **User Story 4:** Compute the hidden cost metrics for testing purposes with dummy data.

- **User Story 5:** Conduct unit tests and create drivers (TDD testing)
- **User Story 8:** Backend accepts user fills and performs computations.
- **Malik Dagher**
 - **User Story 2:** Confirm developer machines connect to the database and can run the docker compose.
 - **User Story 6:** Create the frontend UI to accept user data and return feedback.
 - **User Story 7:** Insert a feedback option to receive user comments.
- **Audrey Abboud**
 - **User Story 2:** Confirm developer machines connect to the database and can run the docker compose.
 - **User Story 4:** Compute the hidden cost metrics for testing purposes with dummy data.
 - **User Story 5:** Conduct unit tests and create drivers (TDD testing)
 - **User Story 8:** Backend accepts user fills and performs computations.
- **Daniella Lacotera**
 - **User Story 2:** Confirm developer machines connect to the database and can run the docker compose.
 - **User Story 6:** Create the frontend UI to accept user data and return feedback.
 - **User Story 7:** Insert a feedback option to receive user comments.
- **Rodrigo Munoz**
 - **User Story 1:** Database connectivity and migrations.
 - **User Story 2:** Confirm developer machines connect to the database and can run the docker compose.
 - **User Story 3:** Confirm that data persists permanently in the database, and temporarily in the broker.

Sprint Planning Meeting Minutes - September 29

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **22 story points**. It was decided to continue with this number of user stories as the team had enough time to finish the primary objectives, as well as ask further questions and conduct additional research afterwards.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- **User Story 1:** Complete and test the metric calculating functions.
- **User Story 2:** Integrate real market data from the TwelveData API.

- **User Story 3:** Build a backend API to connect to the frontend.
- **User Story 4:** Polish the frontend pages.
- **User Story 5:** Connect the frontend React to the backend endpoints.
- **User Story 6:** Develop database infrastructure to connect it to every member's machine.
- **User Story 7:** Create the broker to temporarily store metric data.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**
 - **User Story 1:** Complete and test the metric calculating functions.
 - **User Story 2:** Integrate real market data from the TwelveData API.
 - **User Story 3:** Build a backend API to connect to the frontend.
- **Malik Dagher**
 - **User Story 4:** Polish the frontend pages.
 - **User Story 5:** Connect the frontend React to the backend endpoints.
- **Audrey Gamero**
 - **User Story 1:** Complete and test the metric calculating functions.
 - **User Story 3:** Build a backend API to connect to the frontend.
- **Daniella Lacotera**
 - **User Story 4:** Polish the frontend pages.
 - **User Story 5:** Connect the frontend React to the backend endpoints.
- **Rodrigo Munoz**
 - **User Story 6:** Develop database infrastructure.
 - **User Story 7:** Create the broker to temporarily store metric data.

Sprint Planning Meeting Minutes - October 13

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **32 story points**. It was decided to continue with this number of user stories as the team had enough time to finish the primary objectives, as well as ask further questions and conduct additional research afterwards.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

UI:

- **User Story 1:** Provide instant feedback when email/password/name/last name are invalid (validate and guide signup form inputs).

Acceptance Criteria:

- Email shows error for invalid format.
- Password meter + rule hints (at least 8 chars, 1 digit, 1 letter; configurable).
- Name/Last name required; trims leading/trailing spaces.
- Submit button disabled while invalid; shows inline errors per field.
- **User Story 2:** Display server-side field errors on signup/login.
Acceptance Criteria:
 - When API returns 422 with JSON: { ok:false, error:"...", field_errors: { email:"...", password:"..." } }, UI maps errors near the correct field.
 - Depends on API returning structured field errors.
- **User Story 3:** Provide instant feedback when fill input is invalid.
Acceptance Criteria:
 - Timestamp must be valid RFC3339 or ISO8601; otherwise block submit + show error.
 - side dropdown supports buy/sell (case-insensitive), normalized to lowercase.
 - price/quantity accept only numeric input (decimals allowed).
 - symbol uppercased client-side; non-alphanumeric rejected.
- **User Story 4:** Display server-side field errors on calculate.
Acceptance Criteria:
 - When API returns 422 with JSON: { ok:false, req_id:"...", error:"validation failed", field_errors:{ timestamp:"...", ...} }, UI maps errors near correct fields.
 - Depends on Analyzer returning structured field errors.
- **User Story 5:** Auth UX polish before and after authentication.
Acceptance Criteria:
 - Hide “Sign up / Login” when logged in.
 - Show “Logout” and “Calculate” only when logged in.
 - Dashboard/Home displays “Signed in as {name}”.
- **User Story 6:** Provide a clean and accessible UI.
Acceptance Criteria:
 - Consistent buttons, spacing, focus states, and visible error field mapping.
 - UI remains responsive and readable across devices.

Go API:

- **User Story 7:** Develop strong authoritative validation for signup/login.
Acceptance Criteria:
 - Validate email format, max length 254, normalize lowercase.
 - Enforce password rules (8 chars, 1 digit, 1 letter, max length cap).
 - Trim and require name/last_name (max 100).
 - On failure return 422 with JSON: { “ok”:false, “error”:“validation failed”, “field_errors”:{...} }
- **User Story 8:** Standardized error contract and helper adoption.
Acceptance Criteria:
 - All handlers use WriteValidation(w, fieldErrors) → 422.
 - Non-field errors return WriteError(w, status, err).
 - Success responses retain { ok:true, data:{...} }
- **User Story 9:** Guarantee timestamp/side/mode/number formats for calculate+save fill.
Acceptance Criteria:

- Validate fill.timestamp is RFC3339.
- Validate fill.side {buy,sell} normalized lowercase.
- Validate numeric formats or enforce numeric types.
- Validate mode {analyze,estimate}.

- **User Story 10:** Profile retrieval GET /v1/me.

Acceptance Criteria:

- Returns { id, email, name, last_name }.
- Returns 401 when not authenticated.

- **User Story 11:** Edit profile PUT /v1/users/me.

Acceptance Criteria:

- Accepts password update.
- Requires current password + new password.
- Returns updated profile.
- Depends on US4.

Python Analyzer:

- **User Story 12:** Develop strong authoritative validation for calculate.

Acceptance Criteria:

- timestamp: tz-aware ISO-8601 only; reject naive timestamps.
- price: positive float >0, decimals okay, max 8dp precision.
- side: literal “buy” or “sell”.
- quantity: integer >0 (no floats/strings).
- mode: literal “analyze” or “estimate”.
- On error return 422 with JSON: { ok:false, req_id:“...”, error:“validation failed”, field_errors:{...}}

- **User Story 13:** Automated Request Validation Tests.

Acceptance Criteria:

- pytest suite uses fixtures for valid/invalid payloads.
- Invalid requests return HTTP 422 with { ok:false, req_id:“...”, error:“validation failed”, field_errors:{...}}
- Valid payloads pass without raising validation errors.

- **User Story 14:** Validation Metrics Alerts.

Acceptance Criteria:

- Lightweight metrics/log hooks detect bad client behavior.
- Invalid floats become numpy NaN → UI receives None.
- Must preserve float realism or return null-safe values.

- **User Story 15:** Twelve Data API Endpoint Validation.

Acceptance Criteria:

- Validate request params before calling TwelveData.
- Responses validate required numeric+field contract.
- Calls mocked in unit tests + 1 sandbox integration test.
- Log latency, errors, and enforce 422 failures with field errors.

- **User Story 16:** Finalize Backend Python Connectivity.

Acceptance Criteria:

- Initialize DB/adapters at startup; teardown gracefully.

- Health-check verifies connectivity and validation readiness.
- Errors bubble via middleware using standard error contract.
- Documentation updates include required config/env vars.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**

- **User Story 12:** Develop strong validation for calculate.
- **User Story 13:** Automated Request Validation Tests.
- **User Story 14:** Validation Metrics Alerts.
- **User Story 15:** Twelve Data API Endpoint Validation.
- **User Story 16:** Finalize Backend Python Connectivity.

- **Malik Dagher**

- **User Story 3:** Provide instant feedback when fill input is invalid.
- **User Story 4:** Display server-side field errors on calculate.
- **User Story 5:** Auth UX polish before/after authentication.
- **User Story 6:** Provide a clean and accessible UI.

- **Audrey Gamero**

- **User Story 12:** Develop strong validation for calculate.
- **User Story 13:** Automated Request Validation Tests.
- **User Story 14:** Validation Metrics Alerts.
- **User Story 15:** Twelve Data API Endpoint Validation.

- **Daniella Lacotera**

- **User Story 9:** Guarantee formats for calculate save fill validation.
- **User Story 10:** Returns profile GET /v1/me.
- **User Story 11:** Edit profile PUT /v1/users/me.

- **Rodrigo Munoz**

- **User Story 1:** Instant feedback for invalid email/password/name/last name.
- **User Story 2:** Display server-side field errors.
- **User Story 7:** Strong authoritative validation.
- **User Story 8:** Standardized error contract.
- **User Story 9:** Guarantee formats for calculate save.
- **User Story 10:** Guide Daniella.
- **User Story 11:** Guide Daniella.

Sprint Planning Meeting Minutes - October 27

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **25 story points**. It was decided to continue with this number of user stories as the team had enough time to finish the primary objectives, as well as ask further questions and conduct additional research afterwards.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

UI:

- **User Story 1:** Create a page to allow users to check previously calculated results.
- **User Story 2:** Create a button to filter previous results.
- **User Story 3:** Complete the implementation of the calendar for naive datetime input.
- **User Story 4:** Implement a drop down to select timezone and return as string.
- **User Story 5:** Block the calendar when estimate function is selected.

Go API:

- **User Story 6:** Set database to run on a server (not locally).
- **User Story 7:** Set the system to run on a server (not locally).
- **User Story 8:** Allow users to check their previous results.
- **User Story 9:** Allow filtering of previous results.

Python Analyzer:

- **User Story 10:** Create / Review Twelve Data API validation.
- **User Story 11:** Create test cases for the API validation.
- **User Story 12:** Create the explanations for the calculated metrics.
- **User Story 13:** Implement the estimate function.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**
 - **User Story 10:** Create / Review Twelve Data API validation.
 - **User Story 11:** Create test cases for the API validation.
 - **User Story 12:** Create the explanations for the calculated metrics.
 - **User Story 13:** Implement the estimate function.
- **Malik Dagher**
 - **User Story 1:** Create a page to allow users to check previously calculated results.
 - **User Story 2:** Create a button to filter previous results.
 - **User Story 3:** Complete the calendar implementation for naive datetime input.
 - **User Story 4:** Implement timezone selection dropdown.
 - **User Story 5:** Block the calendar when estimate mode is active.
- **Audrey Abboud**
 - **User Story 10:** Create / Review Twelve Data API validation.
 - **User Story 11:** Create test cases for the API validation.
- **Daniella Lacotera**
 - **User Story 8:** Allow users to check their previous results.

- **User Story 9:** Allow filtering of previous results.
- **Rodrigo Munoz**
 - **User Story 6:** Set database to run on a remote server.
 - **User Story 7:** Set the system to run on a remote server.
 - **User Story 8:** Help frontend retrieve stored results.
 - **User Story 9:** Implement result filtering at API layer.

Sprint Planning Meeting Minutes - November 10

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **25 story points**. It was decided to continue with this number of user stories as the team had enough time to finish the primary objectives, as well as ask further questions and conduct additional research afterwards.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

UI:

- **User Story 1:** Create a page to allow users to check previously calculated results.
- **User Story 2:** Create a button to filter previous results (based on symbol, new/old, etc.)
- **User Story 3:** Create a front page to introduce the services of the web app.
- **User Story 4:** Update the estimate function page to grey out certain fields.

Go API:

- **User Story 5:** Set the system to run a VM with a public API (maybe also DB)
- **User Story 6:** Allow users to check their previous results.
- **User Story 7:** Allow filtering (by symbols, time, etc.) for previous results

Python Analyzer:

- **User Story 8:** Create test cases for the API validation.
- **User Story 9:** Create the explanations for the calculated metrics.
- **User Story 10:** Implementing the estimate function.
- **User Story 11:** Implement the LLM and Vector DB.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**
 - **User Story 8:** Create test cases for the API validation.
 - **User Story 9:** Create the explanations for the calculated metrics.

- **User Story 10:** Implementing the estimate function.
- **User Story 11:** Implement the LLM and Vector DB.
- **Malik Dagher**
 - **User Story 1:** Create a page to allow users to check previously calculated results.
 - **User Story 2:** Create a button to filter previous results.
 - **User Story 3:** Create a front page to introduce the services of the web app.
 - **User Story 4:** Update the estimate function page to grey out certain fields.
 - **User Story 5:** Block the calendar when estimate mode is active.
- **Audrey Gamero**
 - **User Story 8:** Create test cases for the API validation.
 - **User Story 9:** Create the explanations for the calculated metrics.
- **Daniella Lacotera**
 - **User Story 6:** Allow users to check their previous results.
 - **User Story 7:** Allow filtering (by symbols, time, etc.) for previous results.
- **Rodrigo Munoz**
 - **User Story 5:** Set the system to run a VM with a public API (maybe also DB)
 - **User Story 6:** Allow users to check their previous results.
 - **User Story 7:** Allow filtering (by symbols, time, etc.) for previous results.

Sprint Planning Meeting Minutes - November 24

Attendees

- Abboud, **Giorgio**
- Dagher, **Malik**
- Gamero, **Audrey**
- Lacotera, **Daniella**
- Munoz, **Rodrigo**

Meeting Time

- **Start time:** 9:00 AM
- **End time:** 12:00 PM

After discussion, the velocity of the team was estimated to be **20 story points**. It was decided to continue with this number of user stories as the team had enough time to finish the primary objectives, as well as ask further questions and conduct additional research afterwards.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- **User Story 1:** Finalize changes to display performance outcomes in the UI.
- **User Story 2:** Thoroughly test analyzer and estimate for response output.
- **User Story 3:** Create the showcase presentation poster.
- **User Story 4:** Create the project documentation file.
- **User Story 5:** Create the showcase presentation slides.
- **User Story 6:** Create the team introduction video.
- **User Story 7:** Create the project demo video.

The team members indicated their willingness to work on the following user stories.

- **Giorgio Abboud**

- **User Story 1:** Finalize changes to display performance outcomes in the UI.
- **User Story 2:** Thoroughly test analyzer and estimate for response output.
- **User Story 3:** Create the showcase presentation poster.
- **User Story 4:** Create the project documentation file.
- **User Story 6:** Create the team introduction video.
- **User Story 7:** Create the project demo video.

- **Malik Dagher**

- **User Story 3:** Create the showcase presentation poster.
- **User Story 5:** Create the showcase presentation slides.
- **User Story 6:** Create the team introduction video.

- **Audrey Gamero**

- **User Story 3:** Create the showcase presentation poster.
- **User Story 5:** Create the showcase presentation slides.
- **User Story 6:** Create the team introduction video.

- **Daniella Lacotera**

- **User Story 3:** Create the showcase presentation poster.
- **User Story 5:** Create the showcase presentation slides.
- **User Story 6:** Create the team introduction video.

- **Rodrigo Munoz**

- **User Story 1:** Finalize UI performance display.
- **User Story 3:** Sprint poster creation.
- **User Story 4:** Project documentation file.
- **User Story 6:** Team introduction video.