SECUREPATH
SOLUTIONS

ATP DOCUMENT

Fraud & Risk Detection System (FRDS)

Version 1.0 | Fall 2025 | SE 305

Date: 18 November’ 2025

Table of Contents

1. INTRODUCTION 3

A. Purpose of Acceptance Test 3
B. Proposed System Overview or Configuration Chart 4
C. Definitions, Acronyms, and Abbreviations 6
D. Testing Principles Used 7
E. Overview of Rest of ATP 8

2. HARDWARE AND SOFTWARE USED FOR TESTING 9

A. Hardware Configuration 10
B. Core Software Stack 10
C. Integration and Testing Tools 11

3. ERROR HANDLING POLICY 12

A. Error Categories and Expected System Behavior 12
B. Core Error Handling Objectives 13
C. Role in Acceptance Testing 14
D. Functions vs. Tests Mapping Table 15
***Functions vs. Tests Table** 15*

4. TEST SETS W/ SCHEDULE (TABLE) 16

Overview of Test Set Design 16
A. Acceptance Test Schedule 16
B. Test Set Execution Rationale 17

5. INDIVIDUAL TEST CASES 18

7. LOG OF MEETINGS, REVIEWS AND MEETINGS 26

8. PROJECT ACCEPTANCE SIGNATURES FOR CLIENT AND DEVELOPER 26

1. Introduction

The purpose of this Acceptance Test Plan (ATP) is to formally validate that the Fraud & Risk Detection System (FRDS) satisfies every functional and non-functional requirement defined in the Medium-Level Design (MLD) and Requirements Document (RD2). This ATP serves as the final validation step in the V-Model and confirms that the fully integrated system meets the client's expectations from a user perspective. All test cases in Section 5 follow the exact same numbering as the MLD, ensuring one-to-one traceability between requirements and acceptance tests.

This ATP focuses exclusively on **validation** ("Are we building the right product?"). It verifies that the complete transaction workflow: Data Upload → Cleansing (WBS-W1) → Risk Scoring (WBS-W2) → Dashboard Visualization and Decision Flow (WBS-W3), operates exactly as described in the MLD. Each acceptance test demonstrates a single promised behavior, executed by the client, and signed individually, consistent with Pack's acceptance-testing methodology.

Items Not Tested in This ATP:
This ATP does not test any functionality not explicitly defined in the MLD. The following areas are excluded from acceptance testing: mobile interfaces, external third-party integrations beyond the Plaid Sandbox mock, advanced machine-learning enhancements planned for Phase II, internal developer-oriented metrics (code coverage, mutation testing), and any system-level performance or stress testing beyond the NFR limits defined in RD2. These items are outside the validated scope of the MLD and therefore not part of formal acceptance.

a. Purpose of Acceptance Test

The primary objective is **Validation** ("Are we building the right product?"). This final testing phase confirms that the integrated FRDS not only functions technically but also satisfy the client's needs for accuracy, transparency, and operational readiness. This ATP will confirm that:

1. **End-to-End Transaction Flow:** We will confirm the entire transaction processing pipeline operates seamlessly, from **Data Upload** $\rightarrow$ **Cleansing** (WBS-W1) $\rightarrow$ **Risk Scoring** (WBS-W2) $\rightarrow$ **Dashboard Alerting** (WBS-W3). This validates the interface communication between all modular components as designed in the DD.

2. **Critical Requirements Met:** All key functional and non-functional deliverables are demonstrable and verifiable by the client. This includes validating the **Fraud Risk Score accuracy** ($\ge 95\%$ per NFR-3.2.1), the system's **low latency** (≤ 2 seconds per transaction per NFR-3.2.2), and the **Dashboard visualization** (FR-3.3.1) showing alerts correctly.

3. **Client Usability and Transparency:** The user interface (UI) and core workflows are **intuitive** for the analyst user, and the system delivers **Explainable AI** by providing clear reasons for flagged transactions (NFR-3.2.4).

4. **Operational Resilience:** The system can successfully handle edge cases and failures, specifically validating the **graceful degradation** mechanism (EX-3.3.1) if an external dependency, such as the Plaid API, fails.

b. Proposed System Overview or Configuration Chart

The FRDS is architected as a **Modular Layered System**, ensuring separation of concerns, scalability, and maintainability. The entire system lifecycle, from data ingestion to final analyst decision, is executed through three secure, integrated modules.

1. Data Cleansing Module (WBS-W1)

- **Purpose:** Acts as the system's quality gateway. This Python-based backend module is responsible for the input validation, deduplication, and normalization of all incoming

transactions (FR-3.1.2/3.1.3).

- **Input/Output:** Ingests raw CSV/JSON files (simulating the Plaid API) and outputs a **Cleaned Dataset** that is ready for downstream analytical processing.

- **Technology Stack:** Python 3.10 with Flask/FastAPI for API exposure.

2. Business Logic / ML Module (WBS-W2)

- **Purpose:** The analytical core, responsible for determining fraud risk by implementing a hybrid detection approach.

- **Key Functions (FR-3.2.1/3.2.3):** It executes both configurable **Rule-Based Analysis** and trained **Machine Learning models** (scikit-learn) to calculate a final **Fraud Risk Score (0-100)**. Each flagged transaction is automatically tagged with an **Explanation/Reason Code** to ensure transparency (NFR-3.2.4).

- **Input/Output:** Consumes the Cleaned Dataset and outputs the **Risk-Scored Dataset** and **Alerts** to the database for visualization.

- **Technology Stack:** Python 3.10, scikit-learn/TensorFlow Lite.

3. Dashboard Module (WBS-W3)

- **Purpose:** Provides the secure, interactive visualization and decision-making environment for the end-user (Analyst).

- **Key Functions (FR-3.3.1/3.3.3):** Displays **real-time risk heatmaps**, transaction tables, summary statistics, and supports user actions (Approve/Reject). It is also responsible for **Report Export** (PDF/CSV).

- **Interface:** Built using **React JS** for responsiveness and **Tailwind CSS** for adherence to modern UI/UX standards.

- **Technology Stack:** React JS, Node.js runtime.

The system utilizes **PostgreSQL** for storing transaction data and **JWT** for secure authentication across all layers, ensuring reliable, end-to-end data flow.

c. Definitions, Acronyms, and Abbreviations

Term	Definition
Acceptance Test Plan (ATP)	The formal document (this file) outlining the final validation of the system against client requirements prior to sign-off and deployment.
Fraud & Risk Detection System (FRDS)	The complete system developed by SecurePath Solutions for identifying, scoring, and mitigating fraudulent financial transactions.
Requirements Document (RD2)	The official source document containing all approved Functional (FR) and Non-Functional (NFR) system requirements.
Design Document (DD)	The technical blueprint detailing the FRDS architecture, component specifications, and technology choices (e.g., Python/React).
Functional Requirement (FR)	A requirement specifying a function that the system must perform (e.g., file upload, report export).
Non-Functional Requirement (NFR)	A requirement specifying criteria that can be used to judge the operation of a system, rather than specific behaviors (e.g., performance, security, accuracy).
Fraud Risk Score	A calculated value (0-100) representing the likelihood that a transaction is fraudulent, derived from rules and machine learning models.

Validation	The process of ensuring that the final product meets the client's expectations and original purpose ("Are we building the right product?").
Work Breakdown Structure (WBS)	A project management tool that organizes and defines the total scope of the project, linking tasks to modules (W1, W2, W3).
JWT	**JSON Web Token**. Used for secure, stateless **authentication** and authorization across the FRDS APIs.
API	**Application Programming Interface**. A set of protocols defining how the system's modules (and the Plaid Sandbox mock) communicate and exchange data.
Plaid Sandbox	The simulated external API environment used to provide **mock financial transaction data** for development and testing, restricted to sandboxed mock data only.
Graceful Degradation	A fault tolerance strategy (EX-3.3.1) where the system maintains partial functionality (e.g., using cached data) when a critical dependency fails.
UI/UX	**User Interface / User Experience**. Refers to the analyst-facing dashboard, evaluated for **usability** and clarity.

d. Testing Principles Used

The Acceptance Testing stage represents the final validation activity in the V-Model and is executed strictly from the user's perspective. Consistent with industry practice and Prof. Pack's lecture guidance, this phase employs **only Black-Box Testing techniques**, meaning that test cases are designed solely from the Functional and Non-Functional Requirements (RD2), without visibility into code or internal logic.

Acceptance Testing in the FRDS project follows four core principles:

1. Requirement-Based Testing (Primary Principle)
Every acceptance test case is mapped directly to a corresponding FR or NFR in the RD2 and traceable through the DD. This ensures that **100% of client-committed functionality is tested**,

and the client can verify that all promised behaviors are demonstrably implemented. The ATP thereby serves as the formal link between the RD2 requirements and the delivered system.

2. Validation over Verification
Acceptance Testing focuses exclusively on *validation* ("Are we building the right product?"). Internal implementation correctness (verification) was already established during System Testing. At this stage, the goal is to demonstrate to the client that the system behaves exactly as expected in realistic, end-to-end workflows—specifically the Data Cleansing → Risk Scoring → Dashboard Alerting sequence.

3. Positive-Test–First Principle for Acceptance Testing
In alignment with Pack's Acceptance Testing rules, **acceptance testing consists primarily of positive tests**, demonstrating normal, expected behavior of the system. Negative or exception-based tests are **only included if the client explicitly requests them**. This protects the client experience by ensuring acceptance testing remains focused, predictable, and aligned with the agreed-upon feature set.
Accordingly, this ATP includes negative test scenarios *only* where the RD2 explicitly identifies required exception behaviors (e.g., EX-3.3.1 graceful degradation).

4. Defined Test Data Supplied by the Project Team
Per acceptance-testing standards, **all test data used in this ATP is prepared and supplied by the project team**. This includes clean datasets, borderline-risk datasets, high-volume performance files, and failure-simulation datasets (for Plaid API outage recovery). The client does not generate test data for the ATP; instead, the client executes the scripted actions using predefined, controlled inputs to ensure consistent and reproducible validation outcomes.

Together, these testing principles ensure that the FRDS Acceptance Test Plan provides an objective, requirements-driven, client-centered mechanism for final system validation and formal project sign-off.

e. Overview of Rest of ATP

The remaining sections of this Acceptance Test Plan define the complete structure of the client-facing validation process for the Fraud & Risk Detection System (FRDS). Each section builds toward the central purpose of Acceptance Testing: providing measurable, traceable evidence that every promised feature in the Requirements Document (RD2) is fully implemented and ready for client sign-off.

Section 2 (Hardware and Software) establishes the standardized, Dockerized environment used during Acceptance Testing. This ensures reproducibility of all test results and maintains alignment with the technology stack documented in the Design Document (DD). It also confirms that all functional and performance validation occurs using the Plaid Sandbox mock environment rather than real banking data.

Section 3 (Error Handling Policy) outlines the system's required resilience behaviors, especially the Graceful Degradation requirement (EX-3.x). It clarifies how the system must respond to invalid files, API failures, or incomplete data—behaviors that the client will explicitly evaluate during Acceptance Testing.

Section 4 introduces the Acceptance Test Schedule and the **Functions vs. Tests Table**, which maps each Functional Requirement (FR-3.x) and Non-Functional Requirement (NFR-3.x) to one or more acceptance tests. This satisfies the requirement-traceability expectations outlined in Prof. Pack's Testing II lectures and ensures 100% client-visible coverage.

Section 5 contains the **Individual Acceptance Test Cases**, written step-by-step and tied directly to RD2 requirement IDs. **This section is the only portion of the ATP that the client (Prof. Pack) will formally sign.** Each test case is structured so that the client can observe the system behavior, compare it to the expected results, and sign each test independently. This aligns with the Acceptance Testing method taught in class: many small signatures rather than one large "approve all or none" signature.

Section 6 (Log of Reviews and Meetings) maintains traceability for all design clarifications and requirement discussions leading up to acceptance. Section 7 (Final Acceptance Signatures) records the concluding agreement that FRDS has met all requirements and is approved for delivery.

Together, these sections ensure the ATP is complete, traceable, and aligned with the V-Model: RD2 → DD → ATP. This structure guarantees that the final decision-making process for acceptance is transparent, auditable, and centered entirely on the test cases validated and signed by the client in Section 5.

2. Hardware and Software used for Testing

As college juniors with budget constraints, we designed our **Acceptance Test environment** to be a perfectly reproducible, low-cost simulation of a real-world system. Since the final demonstration is tied to the **performance NFRs** (like processing 10,000 records in ≤ 3 seconds), **consistency is everything**. We achieved this by making the entire stack **Dockerized**. This means what we test on our laptops is exactly what the client (Prof. Pack) will review, eliminating those annoying "it works on my machine" issues.

A. Hardware Configuration

This setup adheres to the limitations of the typical university lab PC environment while still providing enough juice to hit our performance targets.

Component	Specification	Purpose in ATP
Server/Backend (Docker Host)	**Min. Intel i5/Ryzen 5** (Quad-Core+), **16GB RAM** (8GB minimum), SSD Storage, Ubuntu 22.04 LTS OS.	This is the dedicated host for our Docker containers. The increased RAM is critical for running the Python ML Engine, PostgreSQL instance, and the FastAPI service concurrently without throttling, ensuring we meet our ≤ 2 **second latency target**.
Client Workstation	Standard PC (Intel i5+, 8GB RAM, Windows/Mac OS).	Simulates the end-user (Analyst) experience. Used for **UI validation**, cross-browser checks, and confirming dashboard responsiveness (FR-3.3.1).
Network Condition	Stable LAN/Wi-Fi connection (100 Mbps minimum).	Necessary to accurately measure the time taken for file uploads (Cleansing Module) and API calls without network interference skewing the **latency results** (NFR-3.2.2).

B. Core Software Stack

Our development and testing stack is entirely based on high-performance, open-source technologies, aligning with our academic budget constraint.

Component	Specification	Purpose in ATP
Backend/Core Logic	**Python 3.10**, **FastAPI/Flask**, Scikit-learn, TensorFlow Lite.	Hosts the Business Logic and ML Engine. FastAPI handles the high-volume, **asynchronous API requests** from the Dashboard, minimizing latency.

Database	**PostgreSQL 15** (Containerized Instance).	Serves as the persistence layer for the **Risk-Scored Dataset** and audit logs. We use a real relational DB, not just flat files, to validate **data integrity** (FR-3.2.4).
Frontend/UI	**React 18** and **Tailwind CSS**.	Provides the modern, responsive **Dashboard** (WBS-W3) where the client observes the final system output (alerts, heatmaps).
Containerization	**Docker 25** and **Docker Compose**.	Guarantees the entire FRDS application, including the database and API, boots up identically for both testing and the final client demo, fulfilling the **reproducibility principle**.

C. Integration and Testing Tools

During the ATP, the client and test team will use the following tools to manually verify system behavior and gather objective evidence.

Component	Specification	Purpose in ATP
External Dependency	**Plaid Sandbox** (Mock API data).	Provides the synthetic transaction stream input for end-to-end testing (TFT-01). **Crucially, this is where we simulate the API failure for the Recovery Check (UAT Set 3).**
API Validation	**Postman** (Executed via Newman or manually).	Used to verify that the API endpoints, specifically the `/api/score` and authentication endpoints, correctly return **HTTP 200/403** and validate the **JWT tokens** (NFR-3.1.3).
Test Output Monitoring	**Browser DevTools (Network Tab/Console)**.	Used during the Latency Test (Test Case 3) to precisely measure the time taken for the largest JSON/data payload transfer from the backend to the React dashboard.

Reporting Tool	Built-in PDF/CSV Export Utility.	Validated via **Test Case 4** to ensure reports are generated correctly and contain accurate audit information.

3. Error Handling Policy

The Fraud & Risk Detection System (FRDS) incorporates a robust, multi-layered error handling strategy designed to preserve system stability, protect data integrity, and maintain user trust during unexpected operational events. This policy is derived directly from the Non-Functional Requirements (NFRs) and Exception Handling Requirements (EX-3.x) documented in the RD2, and forms an essential component of the Acceptance Testing criteria.

The foundational principle guiding the system's error management is **graceful degradation**: the FRDS must continue to provide partial or fallback functionality in the presence of failures, rather than terminating unexpectedly or presenting the user with incomplete or misleading results. Each potential error type is paired with a clearly defined system response to ensure predictable behavior, transparent communication with users, and alignment with overall system reliability goals.

A. Error Categories and Expected System Behavior

Error Type (Source)	Policy & Expected Behavior (Reference: RD2 EX-3.x)
Invalid Input Data (FR-3.1.1 / FR-3.1.3)	When a malformed, incomplete, or schema-violating CSV/JSON file is uploaded, the system will reject only the invalid entries while allowing valid records to proceed through the pipeline. The cleansing module must log all anomalies—including missing fields, invalid timestamps, and mismatched formats—and provide a user-readable notification summarizing the issues. This ensures that data integrity is maintained without halting the full workflow for minor issues.

Model Failure (EX-3.2.2)	If the Machine Learning inference module becomes unavailable (e.g., corrupted model file, loading failure, missing dependency), the FRDS will immediately fall back to the deterministic rule-based scoring engine. This guarantees continuity of fraud detection operations and prevents the analyst from receiving incomplete risk assessments. The system will also alert the administrator, log the failure with diagnostic details, and continue operating in "Fallback Mode" until the ML component is restored.
Dashboard Data Feed Failure (EX-3.3.1)	In scenarios where the dashboard cannot retrieve new risk-scored data—due to backend timeout, service unavailability, or network instability—the user interface will automatically shift to the last successfully cached dataset. A clearly visible "Data Unavailable / Using Cached Data" banner will be displayed. This prevents UI errors, broken charts, or blank tables, ensuring that the analyst remains informed and able to continue working with the most recent valid dataset.
Security / Access Denied (NFR-3.1.3)	All API endpoints enforce strict authentication using JWT tokens. If an incoming request carries an invalid, expired, or tampered token, the FRDS must return an HTTP 403 "Forbidden" response without revealing any sensitive details about internal logic or user accounts. Each failed attempt is logged with timestamp, IP address, and reason code to support auditability and potential threat analysis. The system must never expose stack traces or internal identifiers to the end user.

B. Core Error Handling Objectives

The FRDS error handling framework is designed to meet the following objectives:

1. **Maintain Operational Continuity:**
   Even when certain components fail, essential workflows—such as viewing alerts, reviewing transactions, and performing risk assessments—must remain available.

2. **Preserve Data Integrity:**
   Under no circumstance should corrupted or ambiguous data be allowed to enter the scoring engine or dashboard.
3. **Provide Clear and Non-Technical User Feedback:**
   Analysts should receive concise notifications describing the issue and its impact, without exposure to internal system details.
4. **Support Post-Event Diagnosis:**
   All errors, warnings, and fallback activations must be logged in a structured format to support debugging, regression testing, and future optimization.
5. **Align With Security Best Practices:**
   Error messages must never disclose implementation details that could assist an attacker (e.g., encryption keys, module paths, stack traces, or DB schema information).

C. Role in Acceptance Testing

This error handling policy directly informs several Acceptance Test Cases, especially those under UAT Set 3: Recovery Testing. Specifically, the ATP must verify that:

- Invalid input files trigger appropriate warnings without disrupting valid data flow.
- The ML-to-rule-engine fallback mechanism correctly activates upon simulated model failure.
- The dashboard properly displays cached data when backend calls fail.
- Unauthorized requests are rejected securely and consistently with NFR-3.1.3.

Successful demonstration of these behaviors will confirm that the FRDS meets its resilience, reliability, and transparency commitments as defined in the RD2

D. Functions vs. Tests Mapping Table

The Acceptance Test Plan must provide a traceable mapping between each Medium-Level Design (MLD) function and the corresponding Acceptance Test Case(s). This table ensures that **every client-visible function** described in the Design Document (DD) is explicitly validated during Acceptance Testing, as required by Pack's ATP checklist.

This table guarantees full coverage of WBS-linked FRs/NFRs and demonstrates that all promised functionalities will be formally verified during UAT.

Functions vs. Tests Table

MLD Function ID	Module / Description	Corresponding ATP Test Case(s)	WBS Reference
F-1.0	Upload Raw Transaction File	**ATC-01: File Upload & Schema Validation**	W1 – Data Cleansing
F-1.1	Detect & Remove Duplicate Records	**ATC-02: Data Cleansing Accuracy**	W1 – Data Cleansing
F-1.2	Normalize Timestamp, Currency, Category Labels	**ATC-02: Data Cleansing Accuracy**	W1 – Data Cleansing
F-2.0	Execute Rule-Based Fraud Checks	**ATC-03: Rule Engine & Threshold Validation**	W2 – Business Logic / ML
F-2.1	Machine Learning Risk Score Calculator (0–100)	**ATC-04: ML Risk Score Accuracy Test**	W2 – Business Logic / ML
F-2.2	Generate Explanation Code (Explainable AI)	**ATC-04: ML Risk Score Accuracy Test**	W2 – Business Logic / ML
F-3.0	Display Alerts on Dashboard (Real-Time)	**ATC-05: Dashboard Visualization Test**	W3 – Dashboard
F-3.1	User Approve/Reject Action w/ Audit Logging	**ATC-05 & ATC-06: Decision Flow Test**	W3 – Dashboard
F-3.2	Export Report (PDF/CSV)	**ATC-07: Report Export Test**	W3 – Dashboard
F-4.0	API Connection to Plaid Sandbox	**ATC-08: External API Integration Test**	W4 – API Integration
EX-3.3.1	Graceful Degradation w/ Cached Data	**ATC-09: Recovery & Failover Test**	W4 – API Integration
NFR-3.2.2	System Latency ≤ 2s for 10,000 Records	**ATC-10: Performance / Latency Validation**	All Modules

4. Test Sets w/ Schedule (table)

This section defines the structured groups of Acceptance Tests that the **Client will execute** to validate the complete functionality of the Fraud & Risk Detection System (FRDS). All test sets follow the V-Model principle that Acceptance Testing directly validates the Medium-Level Design (MLD) functions mapped in Section 3.5. Each test set is designed to demonstrate full end-to-end functionality, confirm non-functional performance commitments, and validate the system's recoverability under controlled fault conditions.

Overview of Test Set Design

Each test set groups logically related acceptance tests to simplify execution for the Client and ensure end-to-end coverage. These sets match Pack's ATP checklist and incorporate the Functions vs. Tests mapping established in Section 3.5.

Test Set	Objective	Included Test Cases
Set 1: Core Functional Flow	Validate all primary functions required for basic system operation.	ATC-01, ATC-02, ATC-03, ATC-04
Set 2: Dashboard & User Interaction	Validate user-visible actions, alerts, and decision workflows.	ATC-05, ATC-06, ATC-07
Set 3: External Dependency & Recovery	Validate Plaid API integration and graceful degradation mechanisms.	ATC-08, ATC-09
Set 4: Performance & Accuracy Validation	Confirm latency, risk score accuracy, and NFR compliance.	ATC-10

This structure ensures every MLD function is validated through user-visible acceptance tests.

A. Acceptance Test Schedule

- **Client (Primary Role):**
  Executes and signs off each acceptance test. The ATP is a client-driven process; only the Client's completion and signature constitute acceptance.
- **Project Team (Support Role):**
  Prepares the environment, provides datasets, and stands by to answer questions. The team **does not** execute acceptance tests unless explicitly requested.

Execution Date	Test Set	Primary Executor	Support	Notes

Day 1 – Morning	Set 1: Core Functional Flow	**Client**	Project Team	End-to-End functional validation.
Day 1 – Afternoon	Set 2: Dashboard & User Interaction	**Client**	Project Team	Includes audit log verification.
Day 2 – Morning	Set 3: External Dependency & Recovery	**Client**	Project Team	API failure simulation + recovery test.
Day 2 – Afternoon	Set 4: Performance & Accuracy Validation	**Client**	Project Team	Latency measurement for 10,000-record file.

B. Test Set Execution Rationale

- **Sequential Order:**
  The sets are arranged so that foundational functionality is validated first (Set 1), ensuring downstream performance and recovery checks are meaningful and not confounded by basic system defects.
- **Client Involvement:**
  Each set includes tasks that are executed by both the QA team and the client to ensure transparency and promote confidence in the delivered product. The participation model is consistent with acceptance testing best practices outlined in Prof. Pack's Testing I & II lectures.
- **Sign-off at Each Stage:**
  Successful completion of each UAT set is required before moving to the next. Any deviation from expected results will trigger immediate defect logging, correction, and re-execution of the affected test cases before progressing.

5. Individual Test cases

ATC-01 — File Upload & Schema Validation

5A. Module Name & WBS Reference

Module: Data Cleansing Module
WBS Reference: WBS-W1
Function(s): F-1.0 (Upload Raw Transaction File)

5B. Purpose

To confirm that the system correctly accepts a valid CSV/JSON file, validates the schema, rejects invalid formats, and loads the file into the staging database as specified in FR-3.1.1.

5C. Setup

1. Launch Docker environment using:
2. `docker compose up`
3. Open the Dashboard at:
   `http://localhost:3000/`
4. Ensure PostgreSQL container is running.
5. Prepare the test files in the test data folder:
   - `valid_transactions.csv`
   - `invalid_missing_column.csv`
   - `invalid_format.txt`
6. Login using the test analyst account:
7. `username: analyst_test`
8. `password: test1234`

5D. Actions

1. Click “Upload Transactions.”
2. Select `valid_transactions.csv`.
3. Click “Submit.”
4. Observe schema verification results.
5. Repeat with `invalid_missing_column.csv`.
6. Repeat with `invalid_format.txt`.

5E. Expected Results

- System accepts valid file.
- Success message displays: **“File uploaded and validated successfully.”**

- Database table `staging_transactions` contains the same number of records as the CSV.
- Invalid CSV produces error: **"Missing required column: amount."**
- Invalid TXT file is rejected: **"Unsupported file format."**
- No crashes or unhandled exceptions occur.

5F. Signatures

Client Signature: _____ Date: _____
Developer Signature: _____ Date: _____

ATC-02 — Data Cleansing Accuracy (Deduplication & Normalization)

5A. Module Name & WBS Reference

Module: Data Cleansing
WBS: WBS-W1
Function(s): F-1.1, F-1.2

5B. Purpose

To verify that duplicates are removed and timestamps/currency formats are normalized exactly as defined in FR-3.1.2.

5C. Setup

1. Use dataset: `duplicates_and_format_errors.csv`.
2. Load file from ATC-01 step.
3. Confirm cleansing engine is running (`docker logs cleansing-service`).
4. Expected unique rows from file = **480**.

5D. Actions

1. Upload `duplicates_and_format_errors.csv`.
2. Click "Run Cleansing."
3. After execution, download the "Cleaned Output."
4. Open the log file: `/logs/cleansing/cleansing_log.txt`.

5E. Expected Results

- Duplicate count detected ≥ 1 and removed.
- Final record count = **480** (must match expectation).
- All timestamps converted to `YYYY-MM-DD HH:MM:SS`.

- All currency values normalized to float with two decimals.
- Log file includes lines:
  `"Normalization completed"`
  `"Duplicates removed"`.

5F. Signatures

Client: _____ Date: _____
Developer: _____ Date: _____

ATC-03 — Rule Engine Threshold Validation Module & WBS

Business Logic / ML Module
WBS-W2
Function: F-2.0

Purpose

Verify that the rule engine correctly flags high-risk transactions:
Threshold = **$5,000** or foreign country or new IP address.

Setup

Use dataset: `rule_test_set.csv` containing:

- 3 high-amount cases
- 2 new-IP cases
- 1 foreign-country case
  Expected risk flags: **6**.

Actions

1. Upload dataset.
2. Click “Run Rule Analysis.”
3. Navigate to Dashboard → Flagged Transactions.

Expected Results

- All 6 transactions flagged.
- Flag reason codes match rule IDs (e.g., “R1: High Amount”).
- Dashboard displays each flag in “High Risk” (red indicator).

Signatures

Client: _____ Developer: _____ Date: _____

ATC-04 — Machine Learning Risk Score Accuracy

Module/WBS: WBS-W2
Function: F-2.1, F-2.2
Purpose

To validate that the ML model outputs risk scores matching labeled test data within ±3 tolerance, and generates Explainable AI reason codes per NFR-3.2.4.

Setup

Dataset: `ml_labeled_test_set.csv`
Contains 20 transactions with known expected scores.

Actions

1. Upload file.
2. Click "Run ML Scoring."
3. Export ML scoring results.

Expected Results

- For all 20 transactions:
- `| actual_score − expected_score | ≤ 3`
- Explanation codes appear in column `reason_code`.
- No missing scores or NaN values.

Signatures

Client: _____ Developer: _____ Date: _____

ATC-05 — Dashboard Visualization Test

Module: Dashboard (WBS-W3)
Purpose

Validate real-time rendering of risk flags, charts, and summary statistics (FR-3.3.1).

Setup

Dataset: `mixed_risk_set.csv`
30 low-risk, 10 medium-risk, 5 high-risk transactions.

Actions

1. Upload and score dataset.
2. Navigate to Dashboard.
3. Inspect:
    - Heatmap
    - Summary cards
    - Risk distribution chart
    - Flagged table

Expected Results

- High-risk transactions show **red** badges.
- Tooltip shows risk score and timestamp.
- Summary statistics match the dataset:
- `High: 5, Medium: 10, Low: 30`

Signatures

Client: _____ Developer: _____ Date: _____

ATC-06 — User Decision Flow (Approve/Reject)

Module: Dashboard (WBS-W3)

Purpose

To validate the end-user decision workflow updates audit logs correctly (FR-3.3.3).

Setup

Dataset: `decision_test_set.csv`.

Actions

1. On Dashboard, select the first flagged transaction.
2. Click “Approve.”
3. Select another flagged transaction.
4. Click “Reject.”
5. Open audit log in DB:
6. `SELECT * FROM audit_log ORDER BY timestamp DESC LIMIT 2;`

Expected Results

- DB shows status = `approved` for first transaction.
- Status = `rejected` for second.
- Both logs contain:
    - user_id
    - timestamp
    - decision

- reason_code (if applicable)

Signatures

Client: _____ Developer: _____ Date: _____

ATC-07 — Report Export Validation
Module: Dashboard / Reporting (WBS-W3)
Purpose

Validate PDF/CSV export generates correct content (FR-3.3.4).

Setup

Dataset: `q4_transactions.csv`.

Actions

1. After scoring, click "Generate Report."
2. Export both PDF and CSV.
3. Open files.

Expected Results

- PDF contains:
  - summary tables
  - charts
  - total flagged count
- CSV contains all columns with no missing values.
- Filename format:
- `report_YYYYMMDD_HHMM.pdf`

Signatures

Client: _____ Developer: _____ Date: _____

ATC-08 — External API Integration (Plaid Sandbox)
Module: API Integration (WBS-W4)
Purpose

Verify system retrieves transactions from Plaid Sandbox and handles failed authentication.

Setup

Use API credentials stored in `.env`:

- Valid key
- Invalid key

Actions

1. Run "Fetch from Plaid."
2. Observe results.
3. Replace API key with invalid one.
4. Run again.

Expected Results

- Valid key → records imported successfully.
- Invalid key → system shows: **"Authentication Failed – Invalid API Key"**
- No crash; error logged in `/logs/api/api_errors.log`.

Signatures

Client: _____ Developer: _____ Date: _____

ATC-09 — Recovery & Failover (Graceful Degradation)

Module: API Integration (WBS-W4)

Purpose

Test the fallback mechanism EX-3.3.1 when the Plaid API is unavailable.

Setup

Dataset: `cached_transactions.json`.
Disable network for Plaid container:

```
docker network disconnect bridge plaid_api_container
```

Actions

1. Click "Fetch Transactions."
2. System should attempt API call → fail.
3. System automatically switches to cached mode.

Expected Results

- Message displayed: **"Plaid unavailable. Using cached data."**
- Dashboard loads normally.

- Log file contains:
  `"API failure detected — switched to cached mode"`.

Signatures

Client: _____ Developer: _____ Date: _____

ATC-10 — Performance & Latency Validation
Module: All (System-Wide)
Purpose

Validate that FRDS processes a 10,000-record dataset in ≤ 2 seconds (NFR-3.2.2).

Setup

Dataset: `ten_k_transactions.csv`.
Open Browser DevTools → Network tab.

Actions

1. Upload dataset.
2. Click "Run Full Pipeline."
3. Measure request time to `/api/score`.

Expected Results

- Response time ≤ **2.0 seconds**.
- CPU usage ≤ 80% in Docker stats.
- No timeout or failure.

Signatures

Client: _____ Developer: _____ Date: _____

7. Log of Meetings, Reviews and Meetings

This log provides traceability for key client/reviewer interactions related to requirements and design, confirming alignment throughout the project lifecycle.

Meeting ID	Date	Participants	Summary of Key Decision/Feedback
M-01	Sep 11, 2025	Client (Prof. Pack), Team	Confirmed modular structure (Cleansing, Logic, Dashboard) and emphasized the necessity of detailed UAT/ATP.
M-02	Sep 26, 2025	Client (Prof. Pack), Team	Requirements Document (RD2) drafted; confirmed accuracy target of 95 and latency target of 2 seconds.
M-03	Oct 17, 2025	Team Lead, Client	Design Document (DD) reviewed; emphasis placed on using JWT for security (NFR-3.1.3) and visual heatmap implementation (FR-3.3.5).
M-04 (STP Sign-Off)	Nov 4, 2025	Client (Prof. Pack), Team	System Test Plan (STP) approved, setting the stage for Acceptance Testing as the next and final formal phase.

8. Project Acceptance Signatures for Client and Developer

We, the undersigned, acknowledge that the Fraud & Risk Detection System (FRDS) has been formally tested according to the Acceptance Test Plan (ATP) outlined above and verify that all necessary acceptance criteria have been met.

Client Acknowledgment: The system meets the documented requirements and is approved for final submission/deployment.

Role	Name	Signature	Date
Client/Reviewer	Prof. Pack		

Developer Acknowledgment: All outstanding issues have been resolved and the system is delivered as promised in the RD2 and DD.

Role	Name	Signature	Date
Developer Lead	SecurePath Solutions		