LT RISK ENGINE

ARCHITECTURE DOCUMENT

Document ID LT-RE-ARCH-001

Version 1.0

Classification Internal

Date 2026-02-16

Executive Summary

The LT Risk Engine is a centralized risk assessment microservice responsible for evaluating and scoring risk across business processes and customer-related operations. It provides rule-driven risk decisions through a synchronous REST API, enabling consistent and auditable risk evaluation across the platform.

This document provides a comprehensive architectural reference covering system design, data model, API contracts, security model, integration patterns, and operational considerations for the LT Risk Engine service.

Service Name: lt-risk-engine	**Language:** Python
Database: PostgreSQL	**Auth Provider:** Keycloak (OAuth2/JWT)
API Protocol: REST / JSON (Synchronous)	**Messaging:** Apache Kafka

System Overview

Purpose and Scope

The LT Risk Engine provides risk scoring based on predefined rules and configurable criteria. It ensures consistent decision-making across the platform by processing requests synchronously, facilitates integration with external data providers when required, and maintains comprehensive traceability of all risk calculations and decisions.

Capability	Description
Rule-Driven Logic	Configurable risk rules stored in the database, allowing logic changes without code deployment.
External Validation	Integration with external data providers for enriched risk assessment.
Full Auditability	Complete decision traceability via input snapshots, rule execution logs, and distributed tracing.
Duplicate Detection	Idempotent evaluation using request hashing to prevent redundant external API calls.
In-Memory Caching	Rule lookups cached to minimize database load and reduce evaluation latency.

Architecture Diagram

The following diagram illustrates the layered architecture of the LT Risk Engine, showing the API layer, business logic layer, and data layer along with external system integrations.

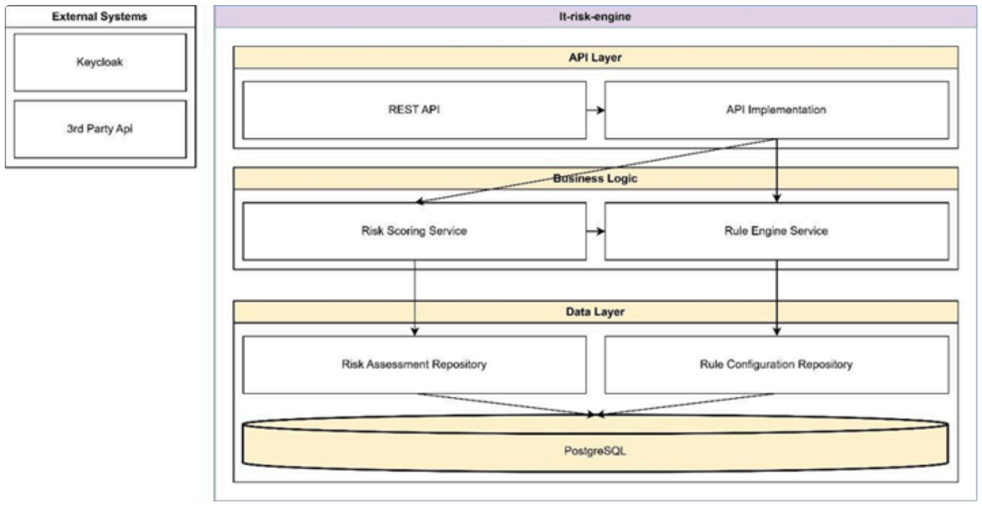

Figure 1: LT Risk Engine — Layered Architecture

Data Flow Diagram

The sequence diagram below details the complete request lifecycle from API client submission through rule loading, parallel external validations, risk scoring, persistence, and response delivery.

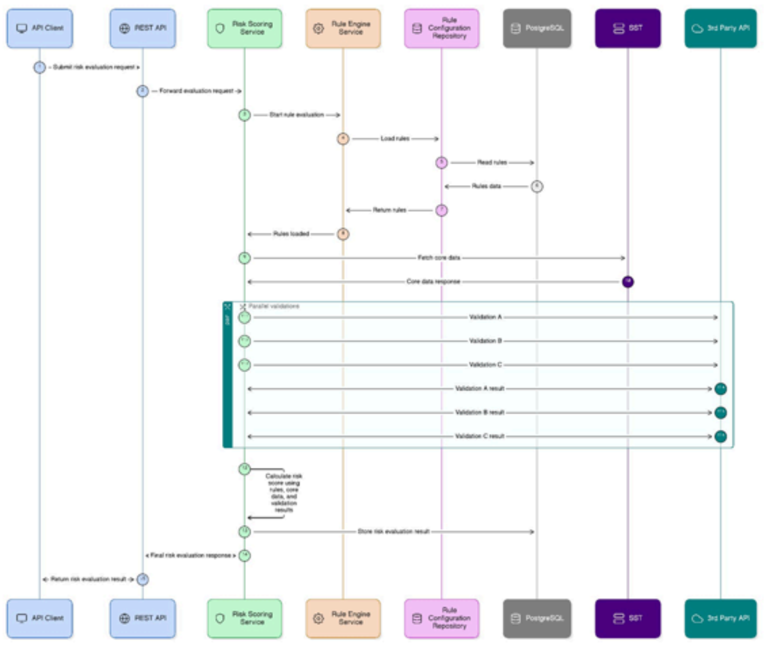

Figure 2: LT Risk Engine — Evaluation Sequence Flow

Module Structure

The service follows a layered modular architecture with clear separation of concerns. Each module has a well-defined responsibility boundary.

File Structure

```
lt-risk-engine

│

├── api

│   └── routes           # REST API Definitions (Routes) │

├── bootstrap

│   ├── config.py         # Application Configuration (Env Variables)

│   └── main.py           # Application Entry Point │

├── common

│   ├── auth              #  Authentication

│   └── utils             # Shared Helper Functions │

├── services

│   ├── entity            # Database Models

│   ├── model             # Data Transfer Objects

│   ├── repository        # Data Access Layer

│   └── service           # Core Business Logic  │

├── messaging

│   ├── consumers         # Kafka Consumers

│   └── producers         # Kafka Producers

│

├── schedulers

│   └── tasks             # Scheduled Jobs

│
```

```
└── tests                # Unit and Integration Tests
```

Core Components

Application Bootstrap

The FastAPI application instance manages service initialization and startup configuration. It enables OpenAPI documentation, authentication middleware for validating OAuth2/JWT tokens via Keycloak, database session management for transactions and persistence, an outbound HTTP client abstraction for inter-service communication, and optional background task scheduling.

Core Services

Service	Description
RiskEvaluationService	Orchestrates the overall risk assessment workflow. Aggregates data from the incoming request, local persistence layer, and external services. Calculates the final risk score and decision outcome.
RuleEngineService	Executes individual risk rules against the evaluation context. Manages rule activation, evaluation behavior, and weighted score contribution.
ExternalDataProviderClient	Handles integrations with external and internal services when additional data is required. Standardizes outbound communication, error handling, and response normalization.

Repositories

Repository	Description

RiskAssessmentRepository	Stores and retrieves calculated risk assessments. Persists audit details for each risk decision including input snapshots and rule execution logs.
RuleConfigurationRepository	Manages creation and maintenance of risk rules and thresholds. Provides efficient retrieval of active rules filtered by evaluation context.

Domain Entities

Entity	Description
RiskAssessment	Represents the result of a single risk evaluation. Contains identifiers, calculated score, final decision (APPROVED / REJECTED / MANUAL_REVIEW), and timestamp information.
RiskRule	Represents a configurable rule used during risk evaluation. Defines rule logic, weighting, activation status, and scope filters.
RiskThreshold	Defines score ranges and corresponding decision outcomes. Determines final classification based on the aggregated risk score.

Data Model

Database Type: PostgreSQL **Schema:** lt_risk_engine

Key Query Patterns

Query	Description

Find Active Rules	Retrieves enabled rules filtered by context (country, product_type). Uses inmemory caching to minimize database load.
Duplicate Detection	Checks for existing evaluations (subject_id + request_hash) within a configurable window (e.g., 24h) to prevent redundant external API calls.
Retrieve Audit Trail	Fetches complete history including decision, score, and rule logs for a given trace_id.

Table: risk_assessment

Central table storing the result of every risk evaluation request.

Column	Type	Description
id	UUID (PK)	Unique identifier of the risk assessment
created_at	TIMESTAMP	Evaluation timestamp
trace_id	UUID	Distributed tracing ID for correlation
subject_id	VARCHAR	Identifier of the evaluated entity (customer/company)
source_system	VARCHAR	Calling service (e.g., lt-kyc-service)
request_hash	VARCHAR	Hash of inputs used for duplicate detection
status	VARCHAR	IN_PROGRESS, COMPLETED, FAILED
total_score	INT	Aggregated risk score

decision	VARCHAR	APPROVED, REJECTED, MANUAL_REVIEW
input_data_snapshot	JSONB	Full snapshot of input data for replay and audit
rule_execution_log	JSONB	Breakdown of triggered rules and individual scores

Table: risk_rule

Stores configurable risk logic. Rules can be modified without code deployment.

Column	Type	Description
id	UUID (PK)	Primary key
rule_code	VARCHAR (UQ)	Human-readable ID (e.g., RULE_GEO_HighRisk)
category	VARCHAR	Classification: FRAUD, CREDIT, COMPLIANCE
weight	INT	Score impact if triggered
logic_config	JSONB	Rule parameters (e.g., banned countries list)
scope	JSONB	Context filters (e.g., applicable product types)
is_active	BOOLEAN	Toggle for enabling/disabling the rule

Table: external_data_log

Logs all calls to external providers (SST, Zefix, CRIF) for billing, debugging, and performance monitoring.

Column	Type	Description
id	UUID (PK)	Primary key
assessment_id	UUID (FK)	Links to risk_assessment
provider_name	VARCHAR	Name of external system (SST, CRIF)
status	VARCHAR	SUCCESS, FAILURE, TIMEOUT
latency_ms	INT	Execution time in milliseconds
raw_response	JSONB	Raw payload received from the provider

API Specification

Protocol: REST / JSON **Security:** OAuth2 Bearer Token (JWT)

Documentation: OpenAPI 3.0 / Swagger UI		**Design:** Synchronous Request-Response
Risk Evaluation Endpoint		
POST	**/v1/risk/evaluate**	

Performs a full synchronous risk evaluation for a given subject.

Request Structure

```
{
  "trace_id": "uuid",
  "subject_id": "string",
  "source_system": "string",
  "country": "string",
  "product_type": "string",
  "risk_category": "string",
  "context_data": { "key": "value" }
}
```

Response Structure

```
{
  "assessment_id": "uuid",
  "trace_id": "uuid",
  "decision": "APPROVED",
  "risk_score": 45,
  "reason_codes": [
    "RULE_GEO_HIGH_RISK_COUNTRY",
    "RULE_HIGH_DEBT_RATIO"
  ],
  "processing_time_ms": 320
}
```

Rule Management Endpoints (Internal)

Full **CRUD operations** for managing **risk rules**. Rules can be **created, updated, toggled, and deleted** without code deployment. Requires ***RISK_ADMIN*** role.

Method	Endpoint	Description
GET	**/v1/rules**	List **all configured risk rules**. Supports filtering by ***category***,

		is_active, and ***scope***.

GET	**/v1/rules/{id}**	Retrieve a **single rule** by its **UUID**, including full **logic_config** and **scope** details.
POST	**/v1/rules**	**Create** a **new risk rule**. Requires ***rule_code***, ***category***, ***weight***, ***logic_config***, and ***scope***.
PUT	**/v1/rules/{id}**	**Full update** of an **existing rule**. Replaces all fields. The ***rule_code*** must remain *unique*.
PATCH	**/v1/rules/{id}/status**	**Toggle** a rule's **is_active** flag. Use to **enable or disable** a rule without modifying its configuration.
DELETE	**/v1/rules/{id}**	Permanently **delete** a risk rule. Returns **409 Conflict** if the rule has *active assessment references*.

Security Model

Authentication

The LT Risk Engine is secured as an OAuth2 Resource Server and integrates with Keycloak for identity and access management. All incoming requests must include a valid signed JWT access token in the Authorization HTTP header.

Protocol: OAuth2 Resource Server	**Identity Provider:** Keycloak
Token Type: Bearer Token (JWT)	

Authorization (RBAC)

Authorization is enforced using Role-Based Access Control. Access control is validated before any business logic is executed.

Role	Permission	Assigned To
RISK_EVALUATOR	POST /v1/risk/evaluate	Internal service accounts
RISK_ADMIN	POST /v1/rules, PATCH /v1/rules/{id}/status	Authorized administrators

JWT Validation and Key Management

Aspect	Implementation
Signature Verification	Stateless JWT signature verification against Keycloak public keys
Token Validation	Token expiration and audience validation enforced on every request
Key Retrieval	Public keys dynamically retrieved from Keycloak JWKS endpoint
Key Storage	No signing keys stored locally; key rotation supported without service restart

Ecosystem Alignment

The LT Risk Engine follows the same security model as lt-kyc-service: OAuth2 Resource Server pattern, Keycloak-based identity management, JWT-based stateless authentication, role-based authorization, and dynamic JWKS key validation. This ensures consistency across the ecosystem while allowing independent implementation in Python.

Evaluation Workflow

The LT Risk Engine executes a fully synchronous risk evaluation workflow. All steps are completed within a single request/response cycle, optimized for real-time decision-making and low latency.

Step	Phase	Details
1	**Request Reception**	Client sends a risk evaluation request to the REST API. The request is authenticated and authorized. A trace_id is assigned or validated for correlation.
2	**Rule Loading**	Active rules are loaded from the database via the Rule Configuration Repository, filtered by request context (country, product type, risk category). Rule configuration may be cached to reduce database load.
3	**Data Collection**	Core data is retrieved from internal systems if required. External validations are triggered in parallel where possible. The service waits for all required responses before proceeding.
4	**Risk Calculation**	The Rule Engine Service evaluates all loaded rules.

		Each triggered rule contributes to the total risk score. A final decision is derived: APPROVED, REJECTED, or MANUAL_REVIEW.
5	**Persistence**	The final result is stored in PostgreSQL. Input snapshot, rule execution details, and validation results are recorded. Duplicate detection logic may reuse a recent result if applicable.
6	**Response**	The final decision and risk score are returned to the caller. The entire workflow completes synchronously within the original HTTP request.

Design Patterns and Principles

Pattern	Description
Synchronous RequestResponse	Risk evaluation is executed within a single API call. The service loads rules, collects required data, computes the score/decision, persists the result, and returns the response immediately.
Rule-Driven Decision Engine	Risk decisions are driven by configurable rules stored in the database. This allows changes to risk logic without redeploying the service. Rules are selected by request context and applied consistently.
Repository Pattern	Database access is isolated behind repositories for rule configuration and assessment persistence. This keeps business logic clean, reduces coupling, and improves testability.

Parallel External Validations	External validations are executed concurrently whenever possible to reduce total response time. The Risk Scoring Service aggregates all validation responses before calculating the final score.
Auditability by Design	Each evaluation is persisted with a trace/correlation identifier, input snapshot, rule execution details, and external validation status. This provides complete decision traceability.
Caching and Idempotency	Rule lookups are cached in-memory to reduce database load. Duplicate evaluations are detected within a configurable time window using subject_id and request hash to prevent redundant external provider calls

Revision History

Version	Date	Author	Description
1.0	2026-02-16	Engineering Team	Initial architecture document