Technical Design Document

Template Intelligence Engine

AdvisoryAI Internal Platform

Document Purpose

This document defines the technical architecture and implementation blueprint for the Template Intelligence Engine. It translates the approved Product Requirements Document into concrete engineering decisions and system design.

1 System Overview

The Template Intelligence Engine ingests Word document templates, automatically understands their structure, identifies static and dynamic sections, and generates formatted Word documents using AI.

The system is designed to be auditable, versioned, and scalable while remaining hackathon-appropriate.

2 Technology Stack

2.1 Backend

- Language: Python 3.11+
- Framework: FastAPI
- ASGI Server: Uvicorn
- Background Tasks: Task Queue with Workers
- Broker and Result Backend: Redis

2.2 Frontend

- Framework: React
- Language: TypeScript
- Build Tool: Vite or Next.js
- Scope: Internal dashboard

2.3 Persistence

- Relational Database: PostgreSQL
- Object Storage: S3-compatible storage

2.4 AI Usage

- Document structure inference
- Section classification
- Content generation

3 Architectural Style

Modular, domain-driven backend architecture with event-driven background processing and hybrid persistence.

4 Domain Model

4.1 Core Domains

- Template Domain
- Section Domain
- Document Domain
- Job Domain
- Audit Domain

Each domain owns its models, business logic, and persistence rules.

5 Data Model

5.1 Templates

Field	**Type**	**Notes**
id	UUID	Primary key
name	text	Human-readable name
created_at	timestamp	Creation time
updated_at	timestamp	Last update

5.2 Template Versions

id	UUID	Primary key
template_id	UUID	Foreign key
version_number	int	Immutable version
source_doc_path	text	S3 location
parsed_representation_path	text	S3 location
created_at	timestamp	Creation time

5.3 Sections

id	serial	Auto-increment per version
template_version_id	UUID	Foreign key
section_type	enum	STATIC or DYNAMIC
structural_path	text	Word structure locator
prompt_config	jsonb	Generation metadata
created_at	timestamp	Creation time

5.4 Documents and Versions

id	UUID	Primary key
template_version_id	UUID	Foreign key
current_version	int	Active version
created_at	timestamp	Creation time

5.5 Document Versions

id	UUID	Primary key
document_id	UUID	Foreign key
version_number	int	Immutable
output_doc_path	text	S3 location
generation_metadata	jsonb	Inputs and prompts
created_at	timestamp	Creation time

5.6 Jobs

id	UUID	Primary key
job_type	enum	PARSE, CLASSIFY, GENERATE
status	enum	PENDING, RUNNING, FAILED, COMPLETED
payload	jsonb	Job input
error	text	Nullable
created_at	timestamp	Creation time
updated_at	timestamp	Last update

5.7 Audit Logs

id	UUID	Primary key
entity_type	text	Template, Document
entity_id	UUID	Target entity
action	text	Created, Updated
metadata	jsonb	Context data
timestamp	timestamp	Event time

6 Object Storage Layout

```
templates/{template_id}/{version}/source.docx
templates/{template_id}/{version}/parsed.json
documents/{document_id}/{version}/output.docx
```

7 Background Processing Pipeline

- Template upload triggers PARSE job
- PARSE emits CLASSIFY event
- CLASSIFY prepares generation readiness
- GENERATE jobs create documents

Each step persists state, retries on failure, and logs errors.

8 API Design

8.1 Base Path

`/api/v1`

8.2 Template APIs

- POST /templates/upload
- GET /templates/{id}
- GET /templates/{id}/versions

8.3 Document APIs

- POST /documents/generate
- POST /documents/{id}/regenerate
- GET /documents/{id}
- GET /documents/{id}/versions

8.4 Job APIs

- GET /jobs/{id}

9 Error Handling

- Automatic retries for recoverable errors
- Failures persisted in job records
- Errors surfaced through API

10 Logging

Logs are written in structured format to a dedicated directory.

- logs/api.log
- logs/worker.log
- logs/errors.log

11 Project Directory Structure

```
backend/
 app/
    main.py
    api/
    domains/
    infrastructure/
    workers/
    config/
    utils/
 logs/
 .env
 requirements.txt
```

12 Non-Goals

- Authentication and user management
- Real financial advice validation
- CRM or portfolio integrations
- Perfect Word edge-case handling

13 Conclusion

This technical design provides a complete, senior-grade blueprint for implementing the Template Intelligence Engine. All architectural decisions align with the approved PRD and are suitable for both hackathon delivery and future expansion.