Extractor Automatizado de Informes Financieros

Solución Técnica Detallada para la Tarea 1 del GeoAI Engineer Test

Implementación con Arquitectura Hexagonal Simplificada e Integración de IA (Gemini)

Desarrollado por: José Armando Son Rojas

29 de mayo de 2025

Resumen

Resumen Ejecutivo: Este documento presenta una descripción técnica exhaustiva de la solución implementada para la Tarea 1 del GeoAI Engineer Technical Test. El objetivo es la construcción de un proceso automatizado para la extracción de datos financieros clave a partir de informes corporativos públicos. Se detalla la arquitectura del sistema, el diseño de sus componentes, el flujo de procesamiento de datos, y cómo se abordan los cinco puntos específicos del requerimiento. Se enfatiza la claridad, modularidad y efectividad del enfoque, que se basa en una arquitectura hexagonal simplificada y la integración con la API de Google Gemini para el procesamiento inteligente de documentos PDF. El documento también cubre los mecanismos de fallback, la estandarización de la salida de datos y el sistema de validación mediante un puntaje de precisión.

Índice

1. Introducción y Visión General del Sistema

El presente documento técnico describe la concepción, diseño e implementación de un sistema automatizado para la extracción de datos financieros, desarrollado como respuesta a la Tarea 1 del GeoAI Engineer Technical Test. El objetivo principal es construir un proceso robusto y eficiente capaz de interactuar con fuentes de información pública (específicamente el sitio web de Torex Gold), obtener informes financieros y extraer métricas clave de manera estructurada y automatizada.

1.1. Planteamiento del Problema (Tarea 1)

La Tarea 1 del test técnico plantea el desafío de automatizar la recolección y el procesamiento de datos financieros a partir de informes empresariales. Los puntos clave del problema, que esta solución busca abordar integralmente, son:

- **Navegación y Descarga:** Acceder al sitio web de Torex Gold y obtener todos los informes financieros (en formato PDF) correspondientes al período comprendido entre 2021 y 2024.
- **Extracción de Métricas:** Procesar los informes descargados para extraer las métricas específicas que permitan construir dos tablas definidas: "Operational Highlights" y "Financial Highlights". Estos datos deben presentarse con una base trimestral y deben incluir resúmenes anuales.
- **Manejo de Variabilidad:** Considerar que los informes pueden variar en su formato y estructura. La solución debe ser lo suficientemente flexible para adaptarse a estas diferencias.
- **Mecanismo de Respaldo (Fallback):** Implementar una estrategia para manejar datos faltantes o inconsistentes. Indicar "No encontrado" es una respuesta aceptable para métricas no localizadas.
- **Estandarización de Salida:** Asegurar que la estructura de los datos extraídos sea uniforme y consistente, independientemente del formato del informe de origen.
- **Validación y Precisión:** Desarrollar un mecanismo de validación que muestre un puntaje de precisión porcentual (%), indicando el grado de completitud y fiabilidad de los datos extraídos (donde 100 % representa una extracción completa y correcta de los datos esperados).

1.2. Arquitectura de la Solución Propuesta

Para abordar estos requerimientos de manera efectiva, se ha optado por una **Arquitectura Hexagonal (también conocida como Puertos y Adaptadores) simplificada**. Este paradigma arquitectónico permite un diseño modular y desacoplado, aislando la lógica de negocio central (el "hexágono") de las dependencias externas como la interfaz de usuario (API REST), la base de datos, o los servicios de terceros (como la API de Gemini). La inteligencia para el procesamiento de documentos y la extracción de métricas se delega a la API de Google Gemini, que actúa como un "Agente de IA" especializado.

1.3. Objetivos Clave de la Solución para la Tarea 1

La solución se enfoca en cumplir con los cinco puntos de la Tarea 1, prestando especial atención a los siguientes aspectos, alineados con los criterios de evaluación:

- **Robustez:** Diseñar un sistema capaz de adaptarse a diferentes formatos de PDF y manejar la ausencia de datos de manera controlada.
- **Precisión y Validación:** Implementar una lógica de validación clara y un puntaje de precisión cuantificable para evaluar la calidad de la extracción.
- **Claridad y Modularidad del Código:** Organizar el código en componentes bien definidos y desacoplados, facilitando su comprensión, mantenimiento y extensión.
- **Efectividad del Enfoque:** Utilizar herramientas y técnicas de IA de manera efectiva para lograr una extracción de datos superior a los métodos tradicionales basados en reglas.
- **Automatización Completa:** Establecer un flujo de trabajo completamente automatizado desde la obtención de los informes hasta el almacenamiento de los datos estructurados.

2. Arquitectura del Sistema Detallada

La arquitectura del sistema se fundamenta en los principios de la Arquitectura Hexagonal, lo que proporciona una estructura flexible y mantenible.

2.1. Modelo de Arquitectura Hexagonal Implementado

El núcleo de la aplicación, que contiene la lógica de negocio y los casos de uso, se comunica con el mundo exterior a través de "puertos" (interfaces abstractas). Los "adaptadores" son las implementaciones concretas de estos puertos y se encargan de la interacción con tecnologías o sistemas específicos.

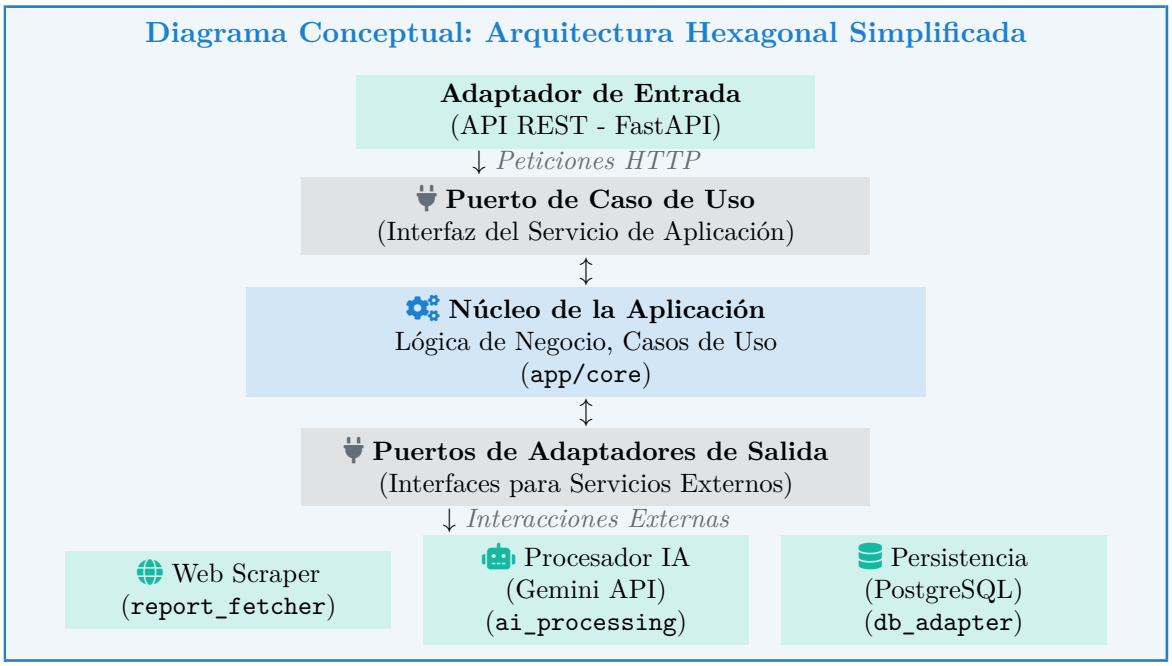

Figura 1: Representación conceptual de la Arquitectura Hexagonal implementada para el extractor de informes.

2.2. Componentes Principales y sus Responsabilidades

2.2.1. Núcleo de la Aplicación (`app/core`)

Este es el corazón del sistema, independiente de frameworks y herramientas externas.

- **`schemas.py`:** Define los Data Transfer Objects (DTOs) y modelos de datos utilizando Pydantic. Estos esquemas (ej. `ReportDetails`, `MetricItem`, `LLMFullExtractionResult`, `StandardizedExtractionOutput`) son cruciales para la validación de datos, la estandarización y la comunicación entre capas.
- **`use_cases.py`:** Contiene la lógica de orquestación de los procesos de negocio. La función principal es `orchestrate_report_extraction_process`, que coordina las interacciones entre los diferentes adaptadores para cumplir con el flujo de extracción. También incluye funciones auxiliares como `_calculate_accuracy_score` para la validación y `_standardize_llm_output` para asegurar la consistencia de los datos.
- **`ai_prompts.py`:** Almacena las plantillas de los prompts que se envían a la API de Gemini. Estos prompts son cuidadosamente diseñados para guiar al LLM en la extracción de las métricas deseadas (contenidas en `EXPECTED_OPERATIONAL_METRICS` y `EXPECTED_FINANCIAL_METRICS`) y en el manejo de casos especiales como datos faltantes.

2.2.2. Adaptadores de Entrada (*Driving Adapters*)

- **API REST con FastAPI (`app/main.py`):** Actúa como el principal punto de entrada al sistema. Expone endpoints HTTP que permiten a los clientes (como el frontend React o herramientas como Postman) interactuar con la aplicación.

2.2.3. Adaptadores de Salida (*Driven Adapters*)

Ubicados en `app/adapters/`, estos componentes implementan la lógica de interacción con los servicios e infraestructuras externas.

- **Adaptador de Obtención de Informes (`report_fetcher_adapter.py`):** Realiza el web scraping, filtra y descarga los PDFs.
- **Adaptador de Procesamiento con IA (`ai_processing_adapter.py`):** Extrae texto de PDFs y se comunica con Gemini API para la extracción de métricas.
- **Adaptador de Base de Datos (`db_adapter.py`):** Gestiona la persistencia de datos en PostgreSQL con SQLAlchemy.

2.2.4. Integración con IA (API de Google Gemini)

Definición 2.1. *Agente de IA Simplificado Funcionalidad dentro del `ai_processing_adapter.py` que formula prompts y procesa respuestas del LLM de Gemini para interpretar PDFs, extraer métricas, identificar períodos y formatear salidas.*

2.3. Stack Tecnológico Empleado

Cuadro 1: Resumen del Stack Tecnológico Utilizado

Categoría	**Tecnología/Librería**
Lenguaje	Python 3.9+
Framework API	FastAPI
ORM	SQLAlchemy
Base de Datos	PostgreSQL
PDF (Texto)	PyPDF2
IA (Extracción)	Google Gemini API (`google-generativeai`)
Contenerización	Docker, Docker Compose
Validación/Esquemas	Pydantic
HTTP	Requests
Parseo HTML	BeautifulSoup4
Configuración	`python-dotenv`, Pydantic Settings

2.4. Endpoints de la API Expuestos

La API REST, implementada con FastAPI en `app/main.py`, proporciona los siguientes puntos de interacción principales:

POST `/api/v1/reports/trigger-extraction`

- **Propósito:** Inicia el proceso completo de búsqueda, descarga y extracción de datos de los informes financieros.
- **Cuerpo de la Petición (Opcional):** JSON con `start_year` y `end_year` para especificar un rango. Si no se provee, usa los valores por defecto de la configuración.
- **Respuesta Exitosa (200 OK):** Un mensaje JSON indicando que el proceso se ha iniciado en segundo plano, el número de informes identificados inicialmente y una lista de sus URLs.
- **Funcionamiento:** Delega la orquestación a una tarea de fondo (`BackgroundTasks`) para no bloquear la respuesta HTTP.

GET `/api/v1/reports/extracted-data/{report_id}`

- **Propósito:** Obtiene los datos detallados extraídos para un informe financiero específico, identificado por su `report_id` numérico.
- **Parámetros de Ruta:** `report_id` (entero).
- **Respuesta Exitosa (200 OK):** Un objeto JSON (`schemas.ExtractedReportDataResponse`) que contiene el nombre del informe, URL, año, trimestre, estado de extracción, puntaje de precisión, y las listas de "Operational Highlights" y "Financial Highlights" con sus valores. También incluye los datos crudos devueltos por el LLM y cualquier mensaje de error.
- **Respuesta de Error (404 Not Found):** Si no se encuentra un informe con el ID proporcionado.

GET `/api/v1/reports/summary`

- **Propósito:** Proporciona un resumen general de todos los informes financieros que han sido procesados o están en la base de datos.

- **Respuesta Exitosa (200 OK):** Un objeto JSON (`schemas.AllReportsSummaryResponse`) que incluye estadísticas como el total de informes en la base de datos, cuántos se procesaron con éxito, cuántos fallaron, el puntaje de precisión promedio, y una lista de resúmenes de cada informe (ID, nombre, año, trimestre, estado, precisión).

GET `/api/v1/reports/annual-table/{year}`

- **Propósito:** (Conceptual y parcialmente implementado en backend) Intenta devolver los datos financieros y operativos consolidados para un año específico, presentados en un formato de tabla con columnas trimestrales (Q1, Q2, Q3, Q4) y un total/promedio anual.
- **Parámetros de Ruta:** `year` (entero).
- **Respuesta Exitosa (200 OK):** Un objeto JSON (`schemas.AnnualReportTableResponse`) con las tablas "Operational Highlights" y "Financial Highlights" formateadas para el año.
- **Respuesta de Error (404 Not Found):** Si no hay suficientes datos procesados con éxito para construir la tabla para el año solicitado.
- **Nota:** La lógica de agregación en el backend para este endpoint es un borrador y requiere un desarrollo más profundo para una precisión completa en la consolidación de datos.

GET `/`

- **Propósito:** Ruta raíz simple para verificar que la API está funcionando.
- **Respuesta Exitosa (200 OK):** Un mensaje JSON de bienvenida.

3. Solución Detallada de los Puntos de la Tarea 1

A continuación, se detalla cómo la solución implementada aborda cada uno de los cinco puntos requeridos por la Tarea 1.

3.1. Punto 1: Acceso al Sitio Web de Torex Gold

Requerimiento: "Go to the Torex Gold website: https://torexgold.com/investors/financial-reports/"

Solución Implementada: El adaptador `app/adapters/report_fetcher_adapter.py` es el responsable de esta tarea.

- La función `fetch_financial_report_urls()` utiliza la librería `requests` para realizar una petición HTTP GET a la URL especificada, obteniendo el contenido HTML de la página.
- La URL base está configurada en `app/config.py` para fácil modificación.

3.2. Punto 2: Obtención de Informes Financieros (2021-2024)

Requerimiento: "Fetch all financial reports from 2021 to 2024."

Solución Implementada: Continuando en `app/adapters/report_fetcher_adapter.py`:

- La función `fetch_financial_report_urls()` utiliza `BeautifulSoup4` para parsear el HTML e identificar elementos `<a>` que enlacen a archivos ``.pdf''.
- Se aplica un filtrado por rango de años (2021-2024).

- La función auxiliar `_parse_report_name_details()` intenta determinar año, trimestre y tipo de documento (FS, MD&A, Presentación) para una selección precisa. Se priorizan MD&A y FS, complementados por Presentaciones.
- Las URLs seleccionadas se devuelven como objetos `schemas.ReportDetails`.
- `download_pdf_content()` descarga el contenido binario de cada PDF.

3.3. Punto 3: Extracción de Métricas para Tablas Objetivo

Requerimiento: "Extract the metrics that allow you to build this two tables in a Quaterly basis with anual summaries."

Solución Implementada: Gestionado por `app/adapters/ai_processing_adapter.py` y orquestado por `app/core/use_cases.py`:

3.3.1. Procesamiento de PDFs y Extracción de Texto

- `_extract_text_from_pdf_bytes()` en `ai_processing_adapter.py` usa `PyPDF2` para extraer texto.

3.3.2. Extracción de Métricas con IA (Gemini API)

- `process_pdf_with_ai()` en `ai_processing_adapter.py` envía el texto a Gemini con un prompt de `ai_prompts.get_metric_extraction_prompt()`.
- El prompt instruye a Gemini para identificar año/trimestre, buscar métricas de `EXPECTED_OPERATIONAL_` y `EXPECTED_FINANCIAL_METRICS`, extraer valores trimestrales y YTD, manejar unidades/escalas, devolver "Not found" para datos faltantes, y formatear la salida como JSON (`schemas.LLMFullExtractionResult`).

3.3.3. Tablas Objetivo

Las métricas extraídas están diseñadas para poblar las tablas "Operational Highlights" y "Financial Highlights". El frontend puede usar el endpoint `/api/v1/reports/annual-table/{year}` para obtener estos datos de forma consolidada.

3.4. Punto 4: Manejo de Formatos, Fallbacks y Estandarización

Requerimiento: "Reports may vary in format... Include a fallback mechanism... Standardize the output structure."

Solución Implementada:

3.4.1. Mecanismo de Fallback ("Not found")

- El prompt a Gemini instruye devolver ``Not found'' para métricas no localizadas.
- El backend maneja y almacena estos valores. El frontend los muestra, indicando datos no extraídos.

3.4.2. Estandarización de la Salida

- **Modelos Pydantic (`schemas.py`):** Definen estructuras de datos consistentes.
- **Respuesta Estructurada del LLM:** Se solicita JSON al LLM.
- **Proceso de Estandarización en Backend (`_standardize_llm_output` en `use_cases.py`):** Mapea la salida del LLM a esquemas Pydantic fijos.
- **Esquema de Base de Datos:** Refuerza la estructura estandarizada.

3.4.3. Manejo de Formatos Variables

- El uso de un LLM (Gemini) es la estrategia principal, ya que puede comprender y extraer de texto con diferentes layouts y fraseologías.
- El prompt está diseñado para ser robusto a variaciones menores en la presentación de métricas.

3.5. Punto 5: Mecanismo de Validación y Puntaje de Precisión

Requerimiento: "Implement a validation mechanism that shows a % accuracy score..."
Solución Implementada:

- **Lógica de Cálculo (`_calculate_accuracy_score` en `use_cases.py`):**
  - Compara los datos del LLM con las métricas esperadas (`EXPECTED_OPERATIONAL_METRICS`, `EXPECTED_FINANCIAL_METRICS`).
  - Cuenta campos esperados (trimestral, YTD, año/trimestre del informe).
  - Cuenta campos encontrados y no ``` ``Not found'' ```.
  - Puntaje $= \left(\frac{\text{Campos\_Encontrados\_Y\_Validos}}{\text{Total\_Campos\_Esperados}} \right) \times 100$.
- **Interpretación del Puntaje:** Mide la **completitud** de la extracción contra métricas objetivo. No valida la exactitud numérica del valor (requeriría ground truth).
- **Almacenamiento y Visualización:** El `accuracy_score` se guarda en la DB y se expone vía API (`/summary`, `/extracted-data/{id}`). El frontend muestra estos puntajes.

4. Flujo del Proceso de Extracción

El proceso sigue un flujo automatizado y orquestado.

4.1. Diagrama de Secuencia Conceptual

Flujo de Secuencia Principal

Actor/Componente		Interacción	Componente	Acción/Resultado
Usuario/API Client	POST →	FastAPI Endpoint (`/trigger`)	Núcleo (`use_cases`)	Inicia tarea de fondo
FastAPI	Resp →	Usuario/API Client		Msg: "Proceso iniciado"
Tarea de Fondo (`use_cases`)	call →	`ReportFetcher`		Obtener URLs
`ReportFetcher`	scrape →	Torex Gold Web		Lista de URLs PDF
`ReportFetcher`	return →	Tarea de Fondo		URLs obtenidas
(Bucle por cada URL de informe identificado)				
Tarea de Fondo	call →	`ReportFetcher`		Descargar PDF
Tarea de Fondo	call →	`AIProcessing`		Extraer Texto + Métricas
`AIProcessing`	API call →	Gemini API		Respuesta JSON
`AIProcessing`	return →	Tarea de Fondo		`LLMFullExtractionResult`
Tarea de Fondo	calc →	Lógica Interna		Estandarizar, Score
Tarea de Fondo	call →	`DBAdapter`		Guardar Datos
`DBAdapter`	write →	PostgreSQL DB		Datos persistidos

Figura 2: Diagrama de secuencia conceptual simplificado del flujo de extracción.

5. Diseño del Sistema (Diagramas C4 Simplificados)

5.1. Nivel 1: Diagrama de Contexto del Sistema

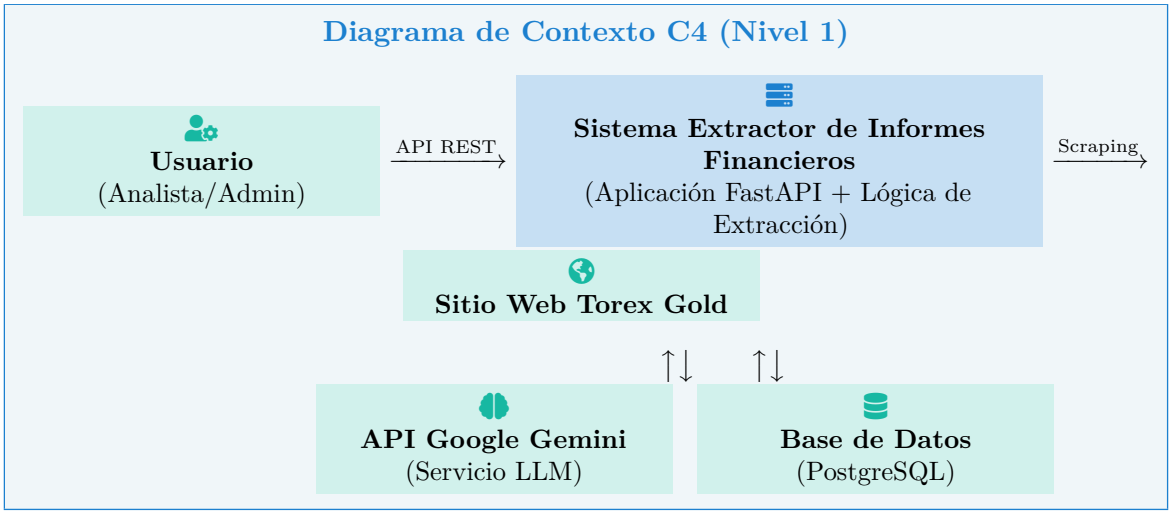

Figura 3: Diagrama de Contexto C4 simplificado del sistema y sus interacciones externas.

5.2. Nivel 2: Diagrama de Contenedores

Figura 4: Diagrama de Contenedores C4 simplificado mostrando los principales componentes Dockerizados.

6. Características y Etapas del Desarrollo

6.1. Características Principales Implementadas

- Obtención automatizada y selectiva de URLs de informes financieros (PDFs).
- Filtrado avanzado por rango de años y priorización de tipo de documento (MD&A, FS, Presentación).
- Descarga eficiente de los PDFs identificados.
- Extracción de texto robusta de los PDFs mediante `PyPDF2`.
- Integración sofisticada con la API de Google Gemini para la extracción inteligente de métricas.
- "Prompt Engineering" detallado para guiar al LLM.
- Estandarización rigurosa de la estructura de datos con modelos Pydantic.
- Mecanismo de fallback claro para datos no encontrados (reportando ``Not found''`).
- Cálculo transparente de un puntaje de precisión (%) basado en la completitud.
- Persistencia de datos en PostgreSQL, con un esquema relacional bien definido.
- API RESTful desarrollada con FastAPI, incluyendo documentación automática.
- Entorno de desarrollo y ejecución contenerizado con Docker y Docker Compose.
- Procesamiento de informes en segundo plano mediante `BackgroundTasks` de FastAPI.

6.2. Etapas del Desarrollo (Simplificadas)

El desarrollo siguió un proceso iterativo:

1) **Análisis y Diseño Inicial:** Comprensión de requisitos, elección de arquitectura y tecnologías.
2) **Configuración del Entorno:** Docker, PostgreSQL.
3) **Implementación de Adaptadores Base:** Obtención de informes y persistencia.
4) **Integración de IA:** Diseño de prompts, conexión con Gemini API.
5) **Desarrollo de Lógica de Negocio:** Casos de uso, validación, estandarización.
6) **Creación de API REST:** Endpoints con FastAPI.
7) **Pruebas y Refinamiento Continuo:** Ajustes basados en resultados y errores.
8) **Documentación Técnica.**

7. Cumplimiento de Criterios de Evaluación (Tarea 1)

- 👍 **Claridad y Modularidad del Código:** La Arquitectura Hexagonal y la organización en `core` y `adapters` aseguran una alta modularidad y separación de conceptos. El uso de Pydantic para esquemas de datos mejora la claridad.
- 👍 **Creatividad y Efectividad del Enfoque:** La delegación de la extracción de datos a un LLM (Gemini) es un enfoque moderno y efectivo para manejar la complejidad y variabilidad de los PDFs financieros, superando las limitaciones de los parsers tradicionales.
- 👍 **Robustez de los Mecanismos de Respaldo:** El sistema maneja explícitamente los datos no encontrados (``Not found''), errores de descarga y fallos en la API de IA, registrando el estado y permitiendo la reanudación o el análisis posterior.
- 👍 **Precisión y Lógica de Validación:** Se implementa un puntaje de precisión cuantitativo basado en la completitud de la extracción contra un conjunto definido de métricas objetivo. La estandarización de salida también contribuye a la validación estructural.
- 👍 **Claridad en la Comunicación de la Presentación:** Este documento, junto con el `README.md` y la documentación de la API, busca comunicar de forma clara la solución. (La presentación PPT sería un componente adicional).

8. Puntos Clave para el Éxito (Tarea 1)

8.1. Abordando la Robustez ante Formatos Variables de PDF

Característica 8.1. *Delegación a LLM La estrategia central para la robustez es el uso de Gemini. En lugar de parsers frágiles, se extrae texto y se confía en la capacidad del LLM para entender y extraer información de formatos diversos.*

Característica 8.2. *Prompt Engineering Detallado Los prompts son cruciales. Guían al LLM sobre qué buscar, cómo manejar ambigüedades (ej. datos faltantes) y cómo estructurar la respuesta. Esto permite una adaptación flexible a las variaciones.*

Característica 8.3. *Estandarización Post-Extracción La salida del LLM, aunque solicitada en un formato JSON, se valida y se mapea a modelos Pydantic rigurosos. Esto asegura que los datos almacenados y servidos por la API sean consistentes.*

Característica 8.4. *Extracción de Texto Generalizada Se usa `PyPDF2`. Para PDFs puramente escaneados (sin capa de texto), se necesitaría OCR, lo cual es una posible mejora futura pero no se implementó para mantener la simplicidad inicial enfocada en informes financieros típicos.*

Este enfoque prioriza la **adaptabilidad semántica** sobre el parsing estructural rígido, lo cual es fundamental para la naturaleza variable de los documentos financieros.

9. Conclusión y Direcciones Futuras

La solución desarrollada para la Tarea 1 del GeoAI Engineer Technical Test aborda de manera integral los requisitos para la extracción automatizada de datos financieros. La arquitectura hexagonal simplificada, combinada con la potencia de la API de Google Gemini, proporciona una plataforma robusta, modular y efectiva. El sistema es capaz de navegar fuentes web, descargar informes, extraer métricas clave, manejar la variabilidad inherente a los documentos PDF, y ofrecer un mecanismo de validación transparente.

El cumplimiento de los cinco puntos de la tarea se ha logrado mediante un diseño cuidadoso de componentes, una estrategia de "prompt engineering" efectiva, y una lógica de procesamiento de datos bien definida. La solución actual sienta una base sólida que puede ser extendida y optimizada.

9.1. Posibles Mejoras Futuras

→ **Integración de OCR Avanzado:** Para manejar PDFs basados en imágenes o con tablas complejas que `PyPDF2` no pueda procesar adecuadamente.

→ **Validación Numérica Cruzada:** Comparar los valores extraídos con otras fuentes o rangos esperados para mejorar la fiabilidad.

→ **Mecanismo de Feedback y Reentrenamiento (Conceptual):** Permitir correcciones manuales de datos y usar este feedback para refinar prompts o, en un sistema más avanzado, reentrenar modelos especializados.

→ **Escalabilidad Mejorada:** Para un volumen muy alto de informes, migrar de `BackgroundTasks` a un sistema de colas de mensajes más robusto como Celery con RabbitMQ/Redis.

→ **Procesamiento de Otros Formatos:** Extender la capacidad para procesar informes en formatos HTML, DOCX, etc.

→ **Interfaz de Usuario Más Rica:** Desarrollar un frontend más completo para la gestión, visualización y análisis de los datos extraídos.

En conclusión, el sistema actual representa una implementación efectiva y bien arquitecturada para el desafío propuesto, demostrando la aplicación de principios de ingeniería de software modernos y el uso inteligente de herramientas de IA.