Instrucciones para la Tarea 1

Introducción: El Contexto de Xpendit

Xpendit es una plataforma SaaS que automatiza la gestión de gastos empresariales. Nuestro objetivo es eliminar la fricción de reportar gastos y dar a los equipos de finanzas un control total y visibilidad en tiempo real.

Una de las funcionalidades *core* de Xpendit es nuestro **Motor de Reglas de Gastos**. Este motor permite a las empresas definir políticas complejas (ej. "el centro de costo de Ventas no puede gastar más de $80 USD en cenas") y las aplica automáticamente a cada gasto que se reporta.

El Desafío: El Motor de Reglas

En este desafío, construirás un prototipo de este motor. La tarea está dividida en 3 partes que se construyen una sobre la otra:

1. **Parte 1: El núcleo de lógica pura** (el validador).
2. **Parte 2: La integración con una API externa** (tasas de cambio).
3. **Parte 3: Un script de análisis de lotes** (procesamiento de datos).

Evaluaremos tu solución no solo por su correctitud, sino también por su diseño, separación de responsabilidades, calidad de pruebas, y buen manejo de errores y secretos.

Filosofía sobre IA

Te incentivamos activamente a usar herramientas de IA (como ChatGPT, GitHub Copilot, etc.) a lo largo de esta tarea. En Xpendit, creemos que la habilidad clave no es memorizar, sino modelar, aplicar pensamiento crítico y usar las herramientas disponibles para crear soluciones eficientes.

Pre-requisito: API Key de Tasas de Cambio

Para este desafío, necesitarás una API Key de un servicio de tasas de cambio.

1. Crea una cuenta gratuita en Open Exchange Rates.
2. Obtén tu `app_id` (API Key) desde tu Dashboard.

Parte 1 - El Motor de Reglas (Lógica Pura)

Objetivo: Construir el núcleo del validador de gastos, demostrando un buen diseño de software y una lógica de negocio flexible.

El motor debe poder devolver 3 estados:

- APROBADO
- PENDIENTE (para revisión humana)
- RECHAZADO

En esta parte, toda la información externa (como las tasas de cambio) debe ser "mockeada" o inyectada. El motor no debe realizar ninguna llamada de red.

Tu Tarea:

1. **Diseña tus Modelos:** Crea las clases, structs o tipos para:
    - **Gasto:** Debe incluir id, monto (Number), moneda (String, ej. "CLP"), fecha (String ISO, ej. "2025-10-20"), categoria (String, ej. "food", "transport", "software").
    - **Empleado:** Debe incluir id, nombre (String), apellido (String), cost_center (String, ej. "sales_team", "core_engineering").
    - **Politica:** Un objeto que define las reglas. Por ejemplo:

```JSON
{
  {"moneda_base": "USD",
    "limite_antiguedad": {
      "pendiente_dias": 30,
      "rechazado_dias": 60
  },
  "limites_por_categoria": {
    "food": {
      "aprobado_hasta": 100,
      "pendiente_hasta": 150
    },
    "transport": {
      "aprobado_hasta": 200,
      "pendiente_hasta": 200
```

```
    }
  },
  "reglas_centro_costo": [
    {
      "cost_center": "core_engineering",
      "categoria_prohibida": "food"
    }
  ]}
```

2. **Crea una clase o módulo que contenga la lógica de validación.**

3. **Implementa las Siguientes Reglas** (basadas en la `Politica`):

Regla	ID	Condición (campo y rango)	Acción	Alerta/Nota
Antigüedad	1	0 ≤ días ≤ 30	APROBADO	
Antigüedad	1	31 ≤ días ≤ 60	PENDIENTE	
Antigüedad	1	días > 60	RECHAZADO	
Límite 'food'	2	category = "food" AND monto ≤ 100 USD	APROBADO	
Límite 'food'	2	category = "food" AND 100 < monto ≤ 150 USD	PENDIENTE	Requiere revisión
Límite 'food'	2	category = "food" AND monto > 150 USD	RECHAZADO	Excede límite aprobado
Regla cruzada	3	cost_center = "core_engineering" AND category = "food"	RECHAZADO	Rendición prohibida

Resolución de Estado Final	Criterio	Estado final	Alerta
Prioridad 1	Si cualquier regla gatilla RECHAZADO	RECHAZADO	
Prioridad 2	Si ninguna es RECHAZADO y al menos una es PENDIENTE	PENDIENTE	
Prioridad 3	Si ninguna es RECHAZADO ni PENDIENTE y al menos una es APROBADO	APROBADO	
Por defecto	Si no aplica ninguna regla anterior	PENDIENTE	Sin alertas

4. **Devuelve un Resultado:** La función de validación debe devolver un objeto estructurado que incluya el estado y las alertas (si las hay).

```JSON
// EJEMPLO APROBADO
{ "gasto_id": "g_125", "status": "APROBADO", "alertas": [] }

// Ejemplo PENDIENTE (con alerta)
{
  "gasto_id": "g_123",
  "status": "PENDIENTE",
  "alertas": [
    { "codigo": "LIMITE_ANTIGUEDAD", "mensaje": "Gasto excede los 30 días. Requiere revisión." }
  ]
}

// Ejemplo RECHAZADO (con alerta)
{
  "gasto_id": "g_124",
  "status": "RECHAZADO",
  "alertas": [
    { "codigo": "POLITICA_CENTRO_COSTO", "mensaje": "El C.C. 'core_engineering' no puede reportar 'food'." }
  ]
```

Entregables (Parte 1):

- El código y los modelos de datos.
- **Un conjunto de Pruebas Unitarias (Unit Tests).** Este es el entregable más importante de esta parte. Debes probar exhaustivamente cada regla y cada estado (APROBADO, PENDIENTE, RECHAZADO).

Parte 2 - El Cliente de Tasas de Cambio (Integración I/O)

Objetivo: Reemplazar el mock de la Parte 1 con una integración real a la API de Open Exchange Rates.

Tu Tarea: Extender tu solución de la Parte 1 para que obtenga las tasas de cambio reales desde la API, en lugar de usar el mock.

Entregables (Parte 2):

- El código de tu módulo o clase de integración (Cliente API).

Parte 3 - El Analizador de Lotes (Algoritmos y Aplicación)

Objetivo: Ensamblar las Partes 1 y 2 para procesar un lote de datos "sucios" del mundo real, aplicando no solo las reglas de política sino también la lógica de detección de anomalías.

Contexto: El equipo de Finanzas nos ha entregado un archivo *gastos_historicos.csv* que contiene datos migrados de un sistema antiguo.

[NOTA: Este archivo *gastos_historicos.csv* será adjuntado en el correo junto a este PDF].

Además de las reglas de política, implementa lógica para detectar **2 de las siguientes** anomalías "sospechosas" con sus alertas:

    - **Duplicados Exactos:** Gastos donde monto, moneda, y fecha son idénticos.
    - **Montos Negativos:** Gastos con montos erróneos.

2. **Optimización (Bonus):**
    - Una solución *naive* hará una llamada a la API de tasas por cada fila del CSV (problema N+1). Una solución optimizada agrupará los gastos por fecha y hará solo **una llamada a la API por fecha única**.

Entregables (Parte 3):

- El código de tu script.
- Un archivo ANALISIS.md y un video de no más de 1 minuto que resuma tus hallazgos, respondiendo:
    - ¿Cuál es el desglose de gastos por estado? (ej. 35 APROBADOS, 10 PENDIENTES, 5 RECHAZADOS).
    - ¿Qué anomalías (duplicados, negativos, etc.) encontraste? Muestra ejemplos.
    - **(Bonus)** Una breve explicación de cómo optimizaste (o cómo optimizarías) las llamadas a la API de Open Exchange Rates para evitar N+1 requests.

Resumen de Requisitos y Entrega

- **Stack: Python** o **JavaScript** (puedes usar cualquier framework o librería de estos ecosistemas, ej. FastAPI, Express, Pandas, Pydantic, Zod, etc.).
- **Plazo:** Tienes **5 días hábiles** desde la recepción de este documento.
- **Entrega:** Responde al correo con un **link a un repositorio Git** (GitHub, GitLab, etc.) o un **archivo .zip**.
- **Entregables Finales:** Tu entrega debe contener:
    1. Todo el código fuente de las Partes 1, 2 y 3.
    2. Los tests unitarios de la Parte 1.
    3. El archivo `ANALISIS.md` de la Parte 3.
    4. Un video que resuma tus hallazgos de no más de 1 minuto.
    5. Un archivo `README.md` que debe incluir:
        - Instrucciones claras de **instalación** (ej. `pip install -r requirements.txt` o `npm install`).
        - Instrucciones de **configuración** (ej. cómo proveer la API Key de Open Exchange Rates, por ejemplo, a través de un archivo `.env`).
        - Instrucciones para **ejecutar las pruebas unitarias** (ej. `pytest` o `npm test`).
        - Instrucciones para **ejecutar el analizador de lotes** (ej. `python analizar.py` o `node main.js`).