Challenge Tekne — Full Stack DEV AI

- Tiempo estimado: 3 horas
- Objetivo: construir una mini solución end-to-end (datos + API + UI) con reglas de negocio, trazabilidad y una feature de IA.
- Fecha límite: **Viernes 06 de Febrero, 23:59 horas.**

Entregable

Un repo con:

- Backend: Node.js + Express (TypeScript recomendado)
- DB: PostgreSQL
- Frontend: React (obligatorio)
- Docs: README.md, DECISIONS.md, DEPLOY.md

Dataset (CSV input)

Campos:

policy_number,customer,policy_type,start_date,end_date,premium_usd,status,insured_value_usd

Ejemplo:

POL-001,Acme Corp,Property,2025-01-01,2025-12-31,1200,active,3000
POL-002,Globex,Auto,2025-02-01,2026-01-31,800,active,20000
POL-003,Initech,Property,2024-06-01,2025-05-31,1500,active,6000

1) Upload + Validación (Backend)

POST /upload (CSV)

- Recibe CSV
- Procesa filas
- Guarda solo las válidas
- Devuelve métricas y errores por fila

Validaciones mínimas (técnicas)

- policy_number obligatorio
- start_date < end_date
- status ∈ {active, expired, cancelled}

Reglas de negocio (obligatorias)

- Si policy_type = Property ⇒ insured_value_usd >= 5000
  - code: PROPERTY_VALUE_TOO_LOW
- Si policy_type = Auto ⇒ insured_value_usd >= 10000
  - code: AUTO_VALUE_TOO_LOW

Si una fila falla reglas: **no se inserta** y se reporta.

Respuesta requerida

```
{
 "operation_id": "uuid",
 "correlation_id": "uuid",
 "inserted_count": 2,
 "rejected_count": 1,
 "errors": [
  { "row_number": 1, "field": "insured_value_usd", "code": "PROPERTY_VALUE_TOO_LOW" }
 ]
}
```

2) Persistencia (PostgreSQL)

Tabla policies (mínimo):

- policy_number (ideal UNIQUE)
- customer
- policy_type
- start_date
- end_date
- premium_usd
- status
- insured_value_usd
- created_at

3) API de consulta (UI-friendly)

GET /policies (obligatorio, para UI)

Debe soportar paginado:

- limit (default 25, max 100)
- offset (default 0)

Filtros (obligatorios):

- status (opcional)
- policy_type (opcional)
- q búsqueda por policy_number o customer (opcional)

Respuesta:

```
{
  "items": [ ... ],
  "pagination": { "limit": 25, "offset": 0, "total": 120 }
}
```

GET /policies/summary (obligatorio)

Devuelve:

- total_policies
- total_premium_usd
- count_by_status
- premium_by_type

4) OOP (obligatorio) — Motor de reglas de negocio

Queremos ver herencia/polimorfismo aplicado al **dominio** (no a “tipos de datos”).

Requerido

- Clase base abstracta: BusinessRule
- Al menos 2 reglas concretas:
    - PropertyMinInsuredValueRule
    - AutoMinInsuredValueRule
- RuleEngine o PolicyValidator que aplique reglas sin conocer sus detalles (polimorfismo)

Errores de reglas:

- code
- field
- message

5) Trazabilidad (obligatorio) — Tabla de operaciones

Cada request a /upload registra una operación en DB.

Tabla operations (mínimo)

- id (UUID) → operation_id
- created_at
- endpoint
- status (RECEIVED, PROCESSING, COMPLETED, FAILED)
- correlation_id
- rows_inserted
- rows_rejected
- duration_ms
- error_summary (nullable)

Correlation ID

- Si viene x-correlation-id, usarlo
- Si no viene, generar UUID
- Incluirlo en logs, respuesta y tabla

6) UI (React) — Obligatoria

La UI debe permitir operar el sistema.

Pantallas mínimas

A) Upload

- Selector de archivo CSV + botón “Upload”
- Mostrar resultado:
    - inserted/rejected
    - operation_id
    - tabla/lista de errores por fila

B) Policies (Listado)

- Tabla con columnas mínimas:
    - policy_number, customer, policy_type, premium_usd, insured_value_usd, status
- Paginado:
    - Next/Prev (usando limit/offset)
- Filtros:
    - status
    - policy_type
    - búsqueda q

C) Summary

- Cards simples con GET /policies/summary:
    - total_policies
    - total_premium_usd
    - count_by_status
    - premium_by_type (puede ser lista simple)

No evaluamos diseño visual. Evaluamos que sea usable y conecte bien con la API.

7) Feature de IA (obligatoria) — usando la info de los endpoints

Queremos una funcionalidad de “asistente” que use los datos ya expuestos por la API.

Requerimiento mínimo (simple y potente)

En la UI de **Policies**, agregar un botón:

“Generate Insights”

Que llame a un endpoint backend:

POST /ai/insights (obligatorio)

Acá podes hacer lo que quieras, pero se da como ejemplo

Input sugerido:

```
{
  "filters": { "status": "active", "policy_type": "Property", "q": "" }
}
```

El backend:

1. consulta internamente policies y summary (o reutiliza la lógica de esos endpoints)
2. genera un texto corto (5–10 líneas) con:

- riesgos / anomalías (ej. muchos rechazos, valores asegurados cerca del mínimo, concentración por tipo)
- 2–3 recomendaciones accionables (ej. revisar umbrales, pedir más data, alertas)

Salida:

```
{
  "insights": [
    "Hay alta concentración de pólizas Property con insured_value cercano al mínimo...",
    "Recomendación: alertar cuando insured_value < 1.1x del mínimo..."
  ],
  "highlights": {
    "total_policies": 120,
    "risk_flags": 3
  }
}
```

8) Observabilidad & Deploy (diseño)

Logs estructurados (mínimo)

- correlation_id
- operation_id
- endpoint
- duration_ms
- inserted/rejected

DEPLOY.md (obligatorio)

Explicar cómo desplegarías en Azure:

- Azure Functions (preferido) o App Service
- Secrets en Key Vault
- App Insights para logs/métricas
- PostgreSQL managed
- CI/CD high-level

9) Documentación obligatoria

DECISIONS.md (10–20 líneas)

- por qué ese diseño OOP
- por qué ese paginado (UI)
- cómo evitarías duplicados / idempotencia
- cómo escalarías
- tradeoffs