# Prueba Técnica – Desarrollador/a de Software Jr (CRUD + SOLID + Consumo de API)

**Nombre del candidato:** David Castillo
**ID de usuario para el API (user_id):** J54GF1

**Duración estimada:** 2–3 horas
**Tecnologías:** PHP o Python (libre elegir). DB: MySQL / PostgreSQL / SQLite.
**Entrega:** Repositorio (GitHub/GitLab) o .zip con código + README.

---

## 1) Objetivo

Construir una aplicación **CRUD** con enfoque **SOLID** que consuma el API suministrada y ejecute un proceso de "mejora" sobre resultados **bad** hasta dejar **todos** los registros en **medium** o **good**.

---

## 2) Descripción funcional

1. **Carga inicial (100 llamadas)**
   - Consumir `testapi.php?user_id=<TU_ID>` hasta obtener **100** respuestas.
   - Guardar **cada** respuesta en la base de datos.
2. **Barridos de mejora**
   - Identificar los registros con categoría **bad**.
   - Reintentar (hacer nuevos llamados al API) y **reemplazar** esos registros siempre que el nuevo resultado sea **medium** o **good**.
   - Repetir barridos **(n)** hasta que **no existan** registros **bad**.
3. **CRUD**
   - Implementar endpoints o pantallas para **crear**, **listar**, **editar** y **eliminar** registros.
   - Mantener una separación por capas (controlador/servicio/repositorio) y aplicar principios **SOLID**.
4. **Reporte final**
   - Generar un **informe breve** con:
     - Resumen de ejecuciones (cuántos intentos totales, cuántos barridos, evolución por categoría).

        - Enfoque técnico (cómo se aplicó SOLID, arquitectura, decisiones).
        - Consultas SQL utilizadas para obtener las métricas.
        - Tablas finales o vistas empleadas.

---

# 3) API suministrada (resumen)

- **Método:** `GET`
- **Parámetro:** `user_id` (alfanumérico)
- **Respuesta:**
    - `value`: 0–100
    - `category`: `bad` (0–60), `medium` (61–85), `good` (86–100)
- **Uso de ejemplo:**
  `https://4advance.co/testapi/get.php?user_id=<TU_ID>`

Nota: planea tu cliente HTTP con manejo de errores y reintentos controlados. Evita ráfagas excesivas.

---

# 4) Requisitos no funcionales

- **Diseño SOLID:**
    - **SRP:** separar responsabilidades (servicio de API, servicio de dominio, repositorio).
    - **OCP/DIP:** inyectar dependencias; facilidad para cambiar DB o cliente HTTP.
    - **ISP/LSP:** interfaces enfocadas y clases sustituibles.
- **Pruebas:** al menos pruebas unitarias básicas.
- **Documentación:** README.

---

# 5) Informe solicitado (entregable)

Incluye un documento breve con:

1. **Resumen de resultados**
    - de llamadas iniciales: 100
    - de barridos realizados
    - de llamadas totales (initial + improvement)
    - Distribución final por categoría
2. **Descripción técnica**
    - Arquitectura y carpetas
    - Cómo aplicaste **SOLID**

    - Patrones/decisiones (inyección de dependencias, repositorios, DTOs, etc.)
    - Manejo de errores y reintentos
3. **Datos y consultas**
    - Esquema de tablas (DDL si corresponde)
    - Consultas SQL usadas para el reporte
    - Capturas o resultados de ejemplo (si aplica)

---

# 6) Entrega

- **Código** + **README** con:
    - Requisitos y pasos para ejecutar (incluye variables de entorno si usas).
    - Cómo correr el proceso de carga inicial y los barridos.
    - Endpoint(s) CRUD y cómo probarlos (cURL o Postman).
- **Informe PDF/MD** con lo del punto 5.

---

# 7) Criterios de evaluación

- Cumplimiento funcional (100 cargas, barridos hasta 0 *bad*).
- Calidad del diseño y aplicación de SOLID.
- Claridad del código y de la documentación.
- Correcto consumo del API y manejo de errores.
- Consultas/reportes reproducibles.

---

### Extra (opcional si te queda tiempo)

- Pruebas unitarias del servicio que decide cuándo actualizar `results`.
- Pequeña interfaz (HTML/JSON) para disparar un barrido y ver estado.
- Métrica del promedio de intentos para “convertir” un `bad`.