ACTIVIDADES DEL LABORATORIO | UNIDAD III

Actividad 1 — Diseño de casos con Partición de Equivalencias + Valores Límite

Objetivo: convertir requisitos en casos de prueba mínimos pero representativos.

Escenario (ejemplo): Campo **Edad** válido entre 0 y 120 (incl.).

Pasos:

1. **Lee** el requisito y escribe su regla en una línea (ej.: "0 ≤ edad ≤ 120").
2. **Identifica** clases de equivalencia:
    - Inválida A: edad < 0
    - Válida B: 0 ≤ edad ≤ 120
    - Inválida C: edad > 120
3. **Elige** 1 caso representativo por clase (A: −1, B: 35, C: 121).
4. **Añade límites:** 0 y 120.
5. **Registra** en una tabla: *ID caso, datos, clase, resultado esperado, prioridad*.
6. **Entrega:** tabla en Excel/Google Sheets + breve justificación (2–3 líneas) de por qué cada caso cubre su clase.

Extensión (opcional): repite con otro campo (p. ej., "monto de transferencia 10–10,000").

Actividad 2 — Pruebas Unitarias (caja blanca) con cobertura básica

Objetivo: crear y ejecutar pruebas unitarias sobre una función simple.

Escenario (ejemplo): calcular_descuento(monto, categoria)
Reglas:

- categoría "A": 10% si monto ≥ 1000; si no, 0%
- categoría "B": 5% siempre
- otras: 0%

Pasos (propuesta en Python, puedes usar Java/JUnit si prefieres):

1. **Crea** carpeta de proyecto: validacion-unidad/.

2. **Implementa** la función (archivo descuentos.py).
3. **Instala** framework de pruebas: pip install pytest (o usa JUnit/NUnit).
4. **Escribe** tests en test_descuentos.py cubriendo:
    - A con 999, 1000 y 2000 (incluye límites)
    - B con 500
    - categoría inválida
5. **Ejecuta** pytest (o tu runner de JUnit) y **corrige** la función si falla algún test.
6. **Entrega:** archivos de código + salida de ejecución (captura o reporte).

Extensión (opcional): mide cobertura (pytest-cov/JaCoCo) y reporta el %.

Actividad 3 — Pruebas de Integración (API) con Postman

Objetivo: validar integración entre cliente y servicio (contrato, estados, payload).

Escenario (ejemplo simple): endpoint REST /login y /saldo (usa un **mock** o un servicio de prueba; si no hay API disponible, crea **colección** con ejemplos y "Tests" en Postman que validen estructura y códigos).

Pasos:

1. **Crea** una **colección** en Postman: "Validación-Integración".
2. **Añade** solicitud **POST /login** con body de prueba.
    - En **Tests**, valida: status 200, campo token existe y no vacío.
3. **Guarda** el token en variable de colección.
4. **Añade** solicitud **GET /saldo** usando el token en header.
    - En **Tests**, valida: status 200, saldo ≥ 0, tipo numérico.
5. **Ejecuta** la colección y **exporta** el reporte (o captura de resultados).
6. **Entrega:** colección exportada .json + evidencia de ejecución.

Extensión (opcional): agrega casos de error (401 sin token / 400 payload inválido).

Actividad 4 — Pruebas de Sistema (E2E) grabadas con Playwright

Objetivo: automatizar un flujo de negocio end-to-end básico.

Escenario (sitio demo o app de práctica del curso): flujo "login → acción simple → logout".

Pasos (Node + Playwright):

1. **Instala** Node.js si no lo tienes.
2. **Inicializa** proyecto:
    - npm init -y
    - npm i -D @playwright/test
    - npx playwright install
3. **Graba** el flujo con **Codegen** (si lo permites):
    - npx playwright codegen <URL_DEMO>
4. **Guarda** el script como e2e.spec.ts (o .js) y **ajusta** aserciones (ej.: que el texto "Bienvenido" exista tras login).
5. **Ejecuta**: npx playwright test (headless) y, si falla, depura con --headed.
6. **Entrega:** script + captura del reporte HTML (playwright-report).

Extensión (opcional): parametriza credenciales vía variables de entorno.

Actividad 5 — Pruebas de Rendimiento (carga suave) con JMeter *o* k6

Objetivo: observar tiempo de respuesta y errores bajo pequeña concurrencia.

Pasos (JMeter GUI, básico):

1. **Abre** JMeter → **Test Plan** → **Thread Group** (p. ej., 10 usuarios, ramp-up 10 s).
2. **Agrega HTTP Request** apuntando a una URL pública o del aula.
3. **Añade View Results in Table/Tree** y **Summary Report**.
4. **Ejecuta** y **observa**: avg/95th percentile, errores, throughput.
5. **Entrega:** captura de Summary Report + breve interpretación (2–3 líneas).

(Si prefieres k6 CLI, puedes usar un script mínimo http.get() con 10 VUs/30s).

Actividad 6 — Trazabilidad y Reporte de Defectos

Objetivo: asegurar cobertura de requisitos y documentar hallazgos.

Pasos:

1. **Construye** una **matriz** (hoja de cálculo) con columnas:
   ID Req | Descripción Req | ID Caso | Nivel (Unidad/Int/Sistema) | Resultado | Evidencia | Observaciones.
2. **Vincula** al menos 1 caso a cada requisito (de Actividades 1–4).
3. **Registra** resultados (Pass/Fail) y **enlaza** evidencias (capturas/reportes).
4. **Si hay fallos**, completa una **ficha de defecto** con:
   ID | Título | Severidad | Prioridad | Paso a paso | Resultado esperado/obtenido | Evidencia | Estado.
5. **Entrega:** matriz de trazabilidad + 1 ejemplo de reporte de defecto real (si hubo).

Rúbrica Evaluación

Criterio	Excelente	Bueno	Básico	Insuficiente
Diseño de pruebas (técnicas aplicadas: equivalencias, límites, niveles, escenarios)	Casos completos, mínimos y bien justificados; cubren válidos/ inválidos y bordes con claridad.	Casos adecuados con pocos huecos; límites presentes pero no en todos los campos.	Casos suficientes pero redundantes o con omisiones notorias; poca justificación.	Casos escasos o no alineados al requisito; sin criterio técnico.
Ejecución y evidencia (rigurosidad, reproducibilidad)	Evidencias claras, ordenadas y trazables; scripts/colecciones ejecutan sin errores; resultados replicables.	Evidencias correctas con leves faltas de orden; ejecución con mínimas incidencias resueltas.	Evidencias parciales; ejecución con fallos no explicados o sin repetibilidad.	Sin evidencias o no verificables; ejecución incompleta.
Análisis de resultados (interpretación, hallazgos)	Interpreta métricas y fallos con precisión; propone acciones/hipótesis; prioriza por riesgo.	Describe resultados y fallos con claridad; propuestas generales.	Menciona resultados sin interpretación; no prioriza.	No analiza; solo pega pantallas.
Buenas prácticas (automatización, cobertura, trazabilidad, reporte)	Uso apropiado de herramientas; cobertura/casos justificadas; matriz de trazabilidad consistente; reporte de defectos completo.	Buen uso general; trazabilidad con pequeños vacíos; reporte aceptable.	Uso limitado de herramientas; trazabilidad básica; reporte incompleto.	Sin método; sin trazabilidad; sin reporte.

Autonomía y presentación	Entrega pulcra, estructurada y autosuficiente; README/guía breve.	Entrega ordenada; mínimos detalles faltantes.	Entrega funcional pero desordenada; requiere guía del docente.	Entrega confusa/incompleta; depende totalmente de guía externa.