🏛️ Taller: De la Queja a la Arquitectura

Objetivo: Como Arquitectos de Software, no construimos por construir. Construimos para resolver dolores reales. Hoy vamos a identificar un problema validado con datos y a traducirlo en requerimientos técnicos.

🟢 PASO 1: Creación de la Sonda

Abran **Microsoft Forms** (o Google Forms) y creen un formulario nuevo llamado: **"Encuesta de Optimización de Procesos"**.

Copien exactamente estas 4 preguntas. Están diseñadas para detectar problemas que el software puede resolver (automatización, gestión de información, notificaciones).

Pregunta 1 (Selección Múltiple)

Título: ¿En qué área sientes que pierdes más tiempo o control actualmente?

- **Trámites y Burocracia:** (Filas, llenar formularios a mano, fotocopias).
- **Gestión de Dinero:** (No sé en qué gasté, olvido pagar facturas, pierdo recibos).
- **Búsqueda de Información:** (No encuentro archivos, correos o datos cuando los necesito).
- **Coordinación:** (Ponerse de acuerdo con gente, reservar espacios, horarios).
- **Salud y Bienestar:** (Citas médicas, seguimiento de rutinas, dietas).
- **Otro:** (Especificar).

Pregunta 2 (Texto Abierto - El Dolor)

Título: Describe brevemente la última vez que tuviste un problema con el área que marcaste arriba. **Subtítulo/Descripción:** *Ejemplo: "Ayer tuve que ir a 3 oficinas distintas para que me firmaran un papel y perdí 2 horas".*

Pregunta 3 (Escala de 1 a 5 - La Frecuencia)

Título: ¿Qué tan seguido te ocurre este problema?

- 1 = Casi nunca (Una vez al año).
- 5 = Todo el tiempo (Diario o Semanal).

Pregunta 4 (Texto Abierto - La Solución Ideal)

Título: Si existiera una solución mágica (App o Web) para esto, ¿qué es lo PRIMERO que debería hacer por ti? **Subtítulo:** *Ejemplo: "Que me envíe una alerta al celular antes de que se venza el plazo".*

🟡 **PASO 2: Recolección de Datos**

El objetivo es conseguir **mínimo 10 respuestas** reales (Por lo menos 4 de ellas deben ser entrevistas físicas).

- Generen el QR o Link.
- Van a salir a entrevistar a personas, también pueden enviarlo a grupos de la U (que no sean compañeros de la clase actual), amigos, familia o compañeros de trabajo.
- **Ejemplo:** *"Ayúdame con 30 segundos para una tarea de arquitectura, necesito saber qué procesos te quitan tiempo".*

🔴 **PASO 3: Traducción a Arquitectura (40 Minutos)**

Revisen sus respuestas en Forms. Elijan **el problema más recurrente o el más grave**. Ahora, vamos a traducir esa "queja humana" a "especificación técnica" en su repositorio.

1. Vayan a su carpeta del proyecto (VS Code).
2. Abran (o creen) el archivo README.md.
3. Copien, peguen y completen la siguiente plantilla con **sus datos reales**:

📐 Proyecto: [Nombre de tu Solución]

1. El Problema (Discovery)

- ¿Qué duele?:** [Copia aquí la respuesta más impactante de la Pregunta 2]
- Frecuencia:** [Pon el promedio de la Pregunta 3. Ej: Es un problema recurrente (4/5)]
- La Solución Soñada:** [Copia aquí la respuesta de la Pregunta 4]

2. Definición Funcional

Historia de Usuario Principal (User Story)

- La fórmula es: Como [ROL], quiero [ACCIÓN], para [BENEFICIO].

Ejemplo: "Como estudiante ocupado, quiero recibir una notificación push cuando un libro de la biblioteca esté disponible, para no tener que ir físicamente a revisar."

Criterios de Aceptación (Definition of Done)

Son las condiciones SI O SI para que el usuario acepte el software.

- El sistema debe funcionar en móviles (Responsive).
- El tiempo de carga debe ser menor a 2 segundos.
- El usuario debe recibir confirmación por correo electrónico.

Requisitos Funcionales (Draft Técnico)

Son las funciones que el sistema debe programar.

RF-01: Módulo de Registro y Login (Autenticación).

RF-02: Conexión con Base de Datos para guardar [Lo que sea que guarden].

RF-03: [Alguna función específica, ej: Generación de PDF, Escaneo de QR, etc.]

🏁 **Cierre: Publicación**

Para finalizar el taller, guarden su definición en la nube:

1. Abran la terminal en VS Code.
2. Ejecuten (En Git Bash):

git add README.md

git commit -m "Definición del problema validada con encuesta y documentación inicial"

git push origin main

3. ¡Listo! Ya tienen la base fundamentada para empezar a diseñar la arquitectura.

📑 **Requisitos de Entrega:**

Para que este taller sea evaluado, tu repositorio de GitHub debe contener la documentación completa de tu proceso de "Arquitectura Forense". La estructura del archivo README.md debe seguir este orden estricto:

1. Documentación del Proyecto (En el README.md)

Sección A: Evidencia de Campo (Discovery)

- **El Problema:** Descripción del dolor detectado en la encuesta.
- **Resultados de la Sonda:** Enlace al formulario (Forms) y un resumen de los datos obtenidos (ej: "El 70% de los encuestados sufre por X motivo").

Sección B: Definición de Requisitos (Definition)

- **Historia de Usuario:** Aplicando la fórmula profesional (Como/Quiero/Para).
- **Criterios de Aceptación:** Mínimo 3 reglas que validen la solución.
- **Requisitos Funcionales:** Mínimo 3 especificaciones técnicas (iniciando con "El sistema debe...").

---

2. Demostración de uso de Git (Evaluación Técnica)

No basta con tener el archivo; el **historial de cambios (Logs)** debe contar la historia de cómo construiste el proyecto. Para la nota técnica, se revisará lo siguiente:

A. El Historial de Commits

Debes demostrar que trabajaste de forma organizada. Se esperan al menos **3 commits** con mensajes descriptivos:

1. git commit -m "docs: estructura inicial del proyecto"
2. git commit -m "feat: agregando resultados de encuesta y dolor detectado"
3. git commit -m "feat: definicion de historias y requisitos tecnicos"

B. Uso de Ramas (Branches)

El trabajo no debe estar solo en main. Debes demostrar la creación de una rama de trabajo:

1. Crear la rama: git checkout -b feature/definicion-requisitos
2. Realizar los cambios y commits en esa rama.
3. Fusionar (Merge) hacia la rama principal:
    - git checkout main
    - git merge feature/definicion-requisitos

Checklist de Calificación:

- ¿El repositorio es público en GitHub?
- ¿El README tiene las Historias, Criterios y Requisitos funcionales claros?
- ¿Los mensajes de los commits son profesionales y descriptivos?

🧠 Guía de Traducción: Del Dolor a la Ingeniería de Software

Como arquitectos, nuestra misión es convertir quejas humanas en instrucciones técnicas precisas. Para lograrlo, usaremos una **cascada lógica** de 4 pasos:

1. El Dolor (Input)

Es la queja cruda que obtuviste en la encuesta. No tiene orden ni términos técnicos.

2. Historia de Usuario (Visión)

Define quién tiene el problema y qué quiere lograr.

- **Fórmula:** Como **[ROL]**, quiero **[ACCIÓN]**, para **[BENEFICIO]**.

3. Criterios de Aceptación (Reglas)

Son las condiciones que el usuario pone para aceptar que la solución sirve. Es el "Contrato de Satisfacción".

4. Requisitos Funcionales (Instrucción Técnica)

Es lo que el **SISTEMA** debe hacer internamente para cumplir lo anterior.

- **Fórmula:** "El sistema debe..." + **Verbo Técnico** (Validar, Registrar, Calcular, Notificar).

📂 **Ejemplo 1: El Problema de las Filas (Alimentación/Servicios)**

- **El Dolor:** "Siempre pierdo 20 minutos haciendo fila para pagar un café y llego tarde a mi clase de las 7:00 AM".
- **Historia de Usuario:** Como **estudiante con prisa**, quiero **pedir y pagar mi café desde el celular antes de llegar**, para **solo pasar a recogerlo sin hacer fila**.
- **Criterios de Aceptación:**
    - El pedido solo se procesa si el pago es confirmado por la plataforma.
    - El sistema debe generar un código QR único para reclamar el producto.
    - El usuario debe recibir una notificación cuando el café esté listo.
- **Requisitos Funcionales (Lo que vamos a programar):**
    - **RF-01 (Pagos):** El sistema debe integrarse con una pasarela de pagos externa (PSE/Nequi) para procesar transacciones en tiempo real.
    - **RF-02 (Gestión):** El sistema debe permitir al administrador del local cambiar el estado del pedido de "Pendiente" a "Listo para Recoger".

- **RF-03 (Seguridad):** El sistema debe validar que el código QR escaneado corresponda al ID del pedido y al usuario que pagó.

📂 **Ejemplo 2: El Problema de las Finanzas (Gestión de Dinero)**

- **El Dolor:** "Nunca sé en qué me gasto la plata del semestre y siempre termino pidiendo prestado al final del mes".
- **Historia de Usuario:** Como **estudiante desorganizado**, quiero **registrar mis gastos diarios por categorías (comida, transporte, fiesta)**, para **ver gráficamente en qué estoy malgastando mi dinero**.
- **Criterios de Aceptación:**
  - No debe tomar más de 10 segundos registrar un gasto.
  - El sistema debe mostrar un color rojo si el gasto supera el presupuesto semanal.
  - Los datos deben estar disponibles aunque no tenga internet en el momento.
- **Requisitos Funcionales (Lo que vamos a programar):**
  - **RF-01 (Persistencia):** El sistema debe almacenar los registros en una base de datos local y sincronizarlos con la nube cuando haya conexión.
  - **RF-02 (Cálculo):** El sistema debe realizar la sumatoria de gastos por etiquetas (tags) y compararlos contra la variable "Presupuesto Mensual" definida por el usuario.
  - **RF-03 (Visualización):** El sistema debe generar reportes dinámicos (gráficos de torta o barras) basados en la tabla de transacciones.

📂 **Ejemplo 3: El Problema de los Objetos Perdidos (Logística/Gestión)**

- **El Dolor:** "Perdí mi carnet en la universidad y nadie me da razón de él; no sé si alguien lo encontró o si sigue tirado en el pasto".
- **Historia de Usuario:** Como **usuario despistado**, quiero **publicar un aviso del objeto que perdí y recibir una alerta si alguien lo encuentra**, para **recuperar mis pertenencias rápido**.
- **Criterios de Aceptación:**
  - El anuncio debe incluir una foto y el lugar donde se vio por última vez.
  - Por seguridad, el sistema no debe mostrar mi número de teléfono públicamente, solo un chat interno.

- **Requisitos Funcionales (Lo que vamos a programar):**
    - **RF-01 (Multimedia):** El sistema debe permitir la carga y almacenamiento de imágenes en formato comprimido (JPG/PNG).
    - **RF-02 (Búsqueda):** El sistema debe implementar filtros de búsqueda por palabras clave y fecha de publicación.
    - **RF-03 (Privacidad):** El sistema debe enmascarar los datos de contacto del usuario creador y habilitar un servicio de mensajería interna basado en WebSockets.

💡 **"Diccionario" de verbos para tus Requisitos Funcionales:**

Si te quedas bloqueado escribiendo los **RF**, usa estos verbos iniciales:

- **Validar** (Para login o datos correctos).
- **Registrar/Guardar** (Para enviar datos a la Base de Datos).
- **Consultar/Mostrar** (Para traer datos a la pantalla).
- **Notificar** (Para correos o alertas push).
- **Calcular** (Para operaciones matemáticas).
- **Integrar** (Para conectar con servicios como Google Maps, WhatsApp o Bancos).