PROMETEO

Unidad 1: Introducción y tecnologías de interfaces

Una Interfaz Gráfica de Usuario (GUI) es la capa visual y manipulable con la que el usuario interactúa: botones, menús, cuadros de texto, listas, diálogos, atajos de teclado y gestos

Sesión 1 – Concepto de GUI, importancia en la experiencia de usuario, overview de tecnologías (JavaFX, .NET MAUI, Flutter, React Native). Configuración de IntelliJ y primer proyecto GUI.

Una **Interfaz Gráfica de Usuario (GUI)** es la capa visual y manipulable con la que el usuario interactúa: botones, menús, cuadros de texto, listas, diálogos, atajos de teclado y gestos. Es la **punta del iceberg** que condensa la complejidad de un sistema en acciones comprensibles. Su objetivo no es "verse bonita", sino **hacer comprensibles y ejecutables las tareas** del usuario con la mínima fricción posible. En términos de negocio, una GUI bien diseñada **reduce el tiempo de aprendizaje**, **minimiza errores**, **incrementa la satisfacción** y **mejora las conversiones** (que en aplicaciones corporativas se traducen en productividad).

Desde la perspectiva de ingeniería, una GUI efectiva nace de tres pilares:

Modelo mental del usuario

Qué espera encontrar, cómo nombra las cosas, qué flujo le parece natural

Consistencia visual y conductual

Mismos patrones para la misma acción, respuestas predecibles a interacciones

Rendimiento percibido

La interfaz *responde* al instante con microfeedback: estados de carga, validaciones, transiciones suaves

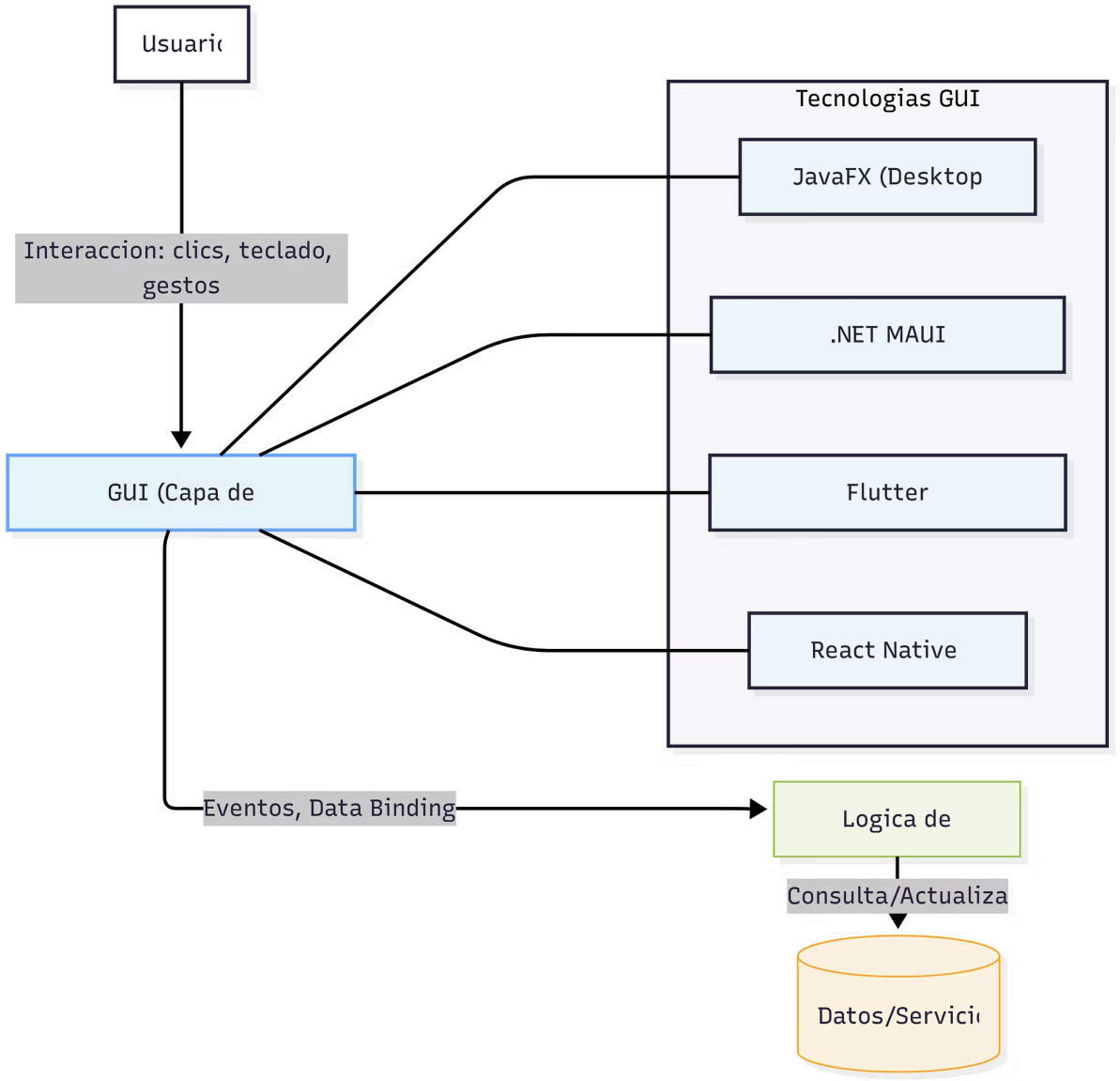
En el ecosistema actual conviven tecnologías con distintos enfoques:

- **JavaFX**: sólido para **escritorio** (Windows, macOS, Linux) con un modelo de escena, FXML declarativo y CSS para estilos. Destaca por su madurez en entornos **Java** y su integración empresarial.
- **.NET MAUI**: apuesta de Microsoft para **multiplataforma nativa** (Windows, macOS, iOS, Android) con **C#** y XAML; comparte lógica y componentes nativos.
- **Flutter**: framework de Google que compila a **nativo** (iOS/Android, web y desktop) usando **Dart**, con widgets propios y gran rendimiento gracias a su motor de render.
- **React Native**: capa JavaScript/TypeScript que **puentea a componentes nativos**; excelentemente posicionado para equipos web que migran a mobile.

La elección depende de **contexto, equipo y dispositivos objetivo**: si tu empresa ya está en Java y necesita escritorio robusto, **JavaFX** encaja; si el core es C# y buscas multiplataforma nativa, **.NET MAUI**; si priorizas time-to-market y UI altamente personalizada, **Flutter**; si tu equipo domina React/web, **React Native** acelera la curva. Esta sesión combina visión estratégica con un aterrizaje práctico: **configurar IntelliJ IDEA** y crear tu **primer proyecto JavaFX** funcional para experimentar el ciclo completo diseño–lógica–ejecución.

Esquema Visual

El siguiente diagrama conceptualiza la relación **Usuario–GUI–Aplicación** y ubica las tecnologías mencionadas según su ámbito principal:



Qué representa cada elemento:

- Usuario (U): Origen de toda acción; su modelo mental define la estructura de la GUI.
- GUI: Capa visual y de interacción; traduce gestos/teclado a eventos y muestra estados/resultados (feedback).
- Lógica de Negocio (BL): Valida, procesa, aplica reglas; debe estar desacoplada para testabilidad y mantenimiento.
- Datos/Servicios (DB): Persistencia local/remota, APIs, ficheros.
- Tecnologías GUI: Herramientas con las que implementas la capa GUI según dispositivos/meta del proyecto.

Caso de Estudio – Slack: claridad visual para complejidad funcional

Contexto

Slack irrumpió en el entorno corporativo como un **hub de comunicación** que integra canales, mensajes, hilos, menciones, buscador, archivos y cientos de integraciones. La complejidad funcional era alta: si la interfaz no organizaba bien, el valor del producto se diluía.

Estrategia

- **Jerarquía y partición visual:** Barra lateral izquierda para **espacios, canales y DMs**, zona central para **conversación principal**, panel lateral derecho para **detalles, hilos o apps**. La distribución reduce la carga cognitiva y acorta el *time-to-task*.
- **Consistencia y patrones:** Atajos, menús contextuales y microinteracciones coherentes (mismo patrón para responder, reaccionar, citar).
- **Búsqueda omnipresente:** Un **search** potente, con sugerencias y filtros, siempre accesible desde el encabezado.
- **Feedback inmediato:** Estados de envío/entrega, indicadores de escritura, recuento de mensajes no leídos, badges discretos.
- **Temas y accesibilidad:** Opciones de **tema oscuro/alto contraste**; tipografías legibles con jerarquías claras.

Resultado

Adopción y retención

La curva de aprendizaje baja al apoyarse en patrones consistentes.

Productividad

Menos tiempo para encontrar hilos o archivos; más foco en tareas.

Escalabilidad

La GUI permite sumar integraciones sin "romper" el mapa mental del usuario gracias a una arquitectura visual estable.

> **Conclusión** Slack demuestra que la GUI no es cosmética: **habilita el valor del producto**. Cuando la información se organiza con una jerarquía clara y se suministra feedback contextual, **tareas complejas parecen simples** y la herramienta se vuelve imprescindible.

Herramientas y Consejos

1

IntelliJ IDEA + JDK 17+ + JavaFX

- **Instala JDK 17 o superior** (soporte LTS y mejor rendimiento).
- En **IntelliJ IDEA**, crea un proyecto **Java con Gradle/Maven** y añade dependencias de **JavaFX** (módulos javafx-controls, javafx-fxml).
- **Scene Builder** (Gluon) para construir vistas **FXML** mediante drag&drop y enlazar controladores.

2

Plantillas y arranque rápido

- Usa **Archetypes/Plantillas**: en Maven (arquetipos) o Gradle (build scripts preconfigurados) para no pelearte con configuración inicial.
- Repositorios de ejemplo: proyectos minimalistas con Main, App, Controller y un sample.fxml te ahorran horas.

3

Diseño escalable desde el inicio

- **Separa capas**: FXML (vista), Controlador (orquestación), Servicio/Modelo (reglas y datos). Evita lógica de negocio en controladores.
- **CSS para estilos**: centraliza colores, tamaños y fuentes; así podrás aplicar **tema claro/oscuro** sin tocar código.

4

Comparativa estratégica de frameworks

- **.NET MAUI**: si tu equipo domina C# y quieres **nativo multiplataforma** con un único proyecto.
- **Flutter**: para UIs personalizadas, animaciones fluidas y despliegue a **móvil/web/desktop** desde un mismo stack.
- **React Native**: si vienes del mundo web/React y quieres aprovechar ecosistema JS/TS en **mobile**.

5

Prácticas de calidad

- **Atajos y productividad** en IntelliJ (Refactor, Navigate to Symbol, Live Templates).
- **Pruebas**: separa la lógica en servicios testeables con **JUnit5**.
- **Accesibilidad mínima**: tamaños de fuente legibles, contraste en colores, foco visible al tabular.

Mitos y Realidades

❌ **MITO**: "La interfaz es solo la piel; lo importante es el back-end."

→ **FALSO**: Para el usuario, **la interfaz *es* el producto**. Una GUI lenta o confusa invalida la mejor arquitectura de servidor. La UX impacta directamente en **adopción, satisfacción y objetivos de negocio**. Un back-end excelente necesita una GUI que lo haga utilizable.

❌**MITO:** "Aprendo una tecnología de GUI y me sirve para siempre."

→ **FALSO:** El sector evoluciona. Dominar fundamentos (eventos, data binding, layouts, estados, accesibilidad) te habilita para migrar entre JavaFX, .NET MAUI, Flutter o React Native. La adaptabilidad es la ventaja profesional real.

Resumen Final

- **GUI**: capa visual que traduce objetivos del usuario en acciones; clave para **UX y negocio**.
- **Tecnologías**: JavaFX (desktop), .NET MAUI (nativo multiplataforma), Flutter (render propio), React Native (puente nativo).
- **Buenas prácticas**: separar vista/logic, usar FXML+CSS, controladores “delgados”, pruebas de servicios.
- **IntelliJ + JDK 17 + JavaFX**: stack base para el **primer proyecto GUI**; Scene Builder acelera el diseño.