UT4

Interfaz de usuario

- Es el medio por el cual una persona controla una aplicación de software o dispositivo de hardware
- Características
    - Atractivo visual
    - Claridad
    - Coherencia
    - Flexibilidad
- Tipos de UI
    - **Lenguaje natural:** la forma en la que el programa se comunica con la persona, por ejemplo Alexa se comunica como si fuera una persona y se le entiende fácilmente
    - **Preguntas y respuestas:** el programa muestra una pregunta y hay respuestas, dependiendo de la respuesta lo que hace. Ejemplo un instalador
    - **Interfaz gráfica del usuario:** usa imágenes, íconos y menús para mostrar las acciones disponibles, ejemplo el escritorio de Windows
    - **Realidad virtual:** dan más libertad que los medios normales, usan recursos innovadores como lentes de VR
    - **Realidad aumentada:** propone nuevas soluciones creativas y nuevas posibilidades. Ejemplo Pokemon Go
    - **Tangible:** interfaz física con palancas, botones, reguladores, etc. Por ejemplo la consola de un DJ
    - **Por voz:** dar indicaciones por voz, Siri, Alexa
- Ventajas de UI optimizada
    - Disminución de costos de desarrollo y capacitación
    - Optimización del área de atención al cliente
    - Mejora de la fidelidad y compromiso de los clientes
    - Aumento en la adquisición de clientes
    - Fomento de la lealtad a la marca
    - Publicidad de boca en boca debido a la experiencia satisfactoria
- Para optimizar
    - **Diseño web adecuado:** optimizar landing pages (páginas fáciles de navegar y con tiempos de carga rápidos). No llenar las páginas de cosas y mantener coherencia
    - **Llamados a la acción (CTA):** forma más efectiva y directa de pedirle a un usuario que dé el siguiente paso
    - **Comprende a tu cliente:** optimizar sitio para todos los canales y dispositivos
    - **Adopta estrategias omnicanal:** adoptar perspectiva omnicanal y crear una experiencia fluida en todos los canales
    - **Formularios:** que los formularios no sean largos y tengan una comprensión y respuesta fáciles

- **Brinda apoyo al usuario:** ofrecer ayuda en los momentos complicados de la página

10 usability heuristics for user interface design by Jakob Nielsen

1) **Visibilidad del estado del sistema:** mantener a los usuarios informados sobre lo que está pasando. Por ejemplo marcar la posición del usuario en un mapa
2) **Relación entre el sistema y el mundo real:** utilizar palabras, frases y conceptos familiares al usuario, así como convenciones. Por ejemplo en una cocina hay un cartel que indica qué hornalla prende cada perilla
3) **Control y libertad del usuario:** los usuarios cometen errores, hay que tener una salida fácil para abandonar la acción equivocada sin pasar por un largo proceso. Por ejemplo botón de cancelar, ctrl z
4) **Consistencia y estándares:** el usuario no tiene que estar adivinando palabras, situaciones acciones, etc, hay que usar las convenciones existentes en plataformas e industrias
5) **Prevención de errores:** tratar de que el usuario no cometa errores. Por ejemplo el autocompletar al escribir en el teléfono
6) **Reconocer antes que recordar:** el usuario tiene que ser capaz de reconocer las cosas (íconos, acciones, opciones) en vez de tener que acordarse de ellas
7) **Flexibilidad y eficiencia de uso:** atajos en teclado para los usuarios experimentados. Para los que no tienen experiencia, que puedan hacer lo mismo sin atajos
8) **Diseño estético y minimalista:** no poner información innecesaria, el diseño no tiene que estar sobrecargado
9) **Ayudar a reconocer, diagnosticar y corregir errores:** que los errores estén en un lenguaje entendido por todos. Si pongo error 402 la mayoría no sabe lo que es, poner algo más amigable
10) **Ayuda y documentación:** ofrecer manual fácil de localizar, que defina los pasos claramente y no sea muy largo

Severity ratings for usability problems

La severidad de un problema de usabilidad consiste en:

- Frecuencia con la que ocurre
- Impacto (fácil o difícil de superar para un usuario)
- Persistencia (se resuelve una vez o si aparece muchas veces)

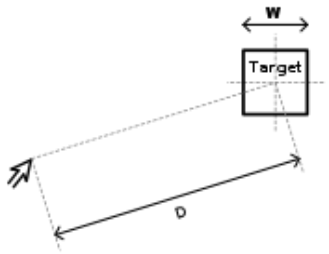
Se debe evaluar el impacto en el mercado. Para esto, se usa una escala entre 0 (no es un problema de usabilidad) y 4 (arreglar el problema antes de lanzar el producto)

16 reglas heurísticas de Tognazzini y cómo aplicarlas

1) **Anticipación:** darle toda la información y herramientas que pueda precisar el usuario

2) **Autonomía:** quien interactúa con el sistema debe sentir que tiene el control para tomar decisiones oportunas. Por ejemplo cambiar tipo de fuente, tamaño, ordenar íconos
3) **Daltonismo:** tener en cuenta colores de la interfaz y ofrecer alternativas
4) **Consistencia:** mantenerla con los estándares existentes y entre diseño de interfaces (tipografía, colores, etc)
5) **Valores por defecto:** valores por defecto deben poder sustituirse fácilmente por lo que quiera escribir el usuario
6) **Eficiencia del usuario:** diseño enfocado en la productividad del usuario, no del sistema. Sistema debe ayudar a cumplir el objetivo del usuario de forma rápida, clara y directa
7) **Interfaces explorables:** flujos de navegación bien delimitados para que el usuario alcance sus objetivos, ofreciendo alternativas para explorar
8) **Ley de Fitts:** velocidad y precisión del movimiento muscular humano para apuntar a un objetivo

$$T = a + b\log_2\left(\frac{D}{W} + 1\right)$$



   a. T es tiempo necesario para hacer el movimiento
   b. a y b son constantes empíricas
   c. D es distancia entre punto inicial y centro del objetivo
   d. W es anchura del objetivo
9) **Objetos de interfaz humana:** objetos de la interfaz deben corresponderse con elementos del mundo real (papelera de reciclaje, disquette para símbolo de guardar)
10) **Reducción de demora:** bajar tiempos de interacción entre sistema e interfaz. Ofrecer feedback de que está cargando (cambiar mouse por ruedita de carga, indicar tiempo de espera)
11) **Aprendizaje:** los productos no deberían tener curvas de aprendizaje
12) **Uso de metáforas:** diseñar usando metáforas que permitan al usuario entender rápidamente un modelo conceptual, conectar mundo real con interfaces digitales
13) **Protección del trabajo del usuario:** tratar de que nunca pierda su trabajo si ocurren problemas
14) **Legibilidad:** textos deben leerse fácilmente, tamaño de letra adecuado, usar palabras clave
15) **Registro del estado:** guardar información que mejore la UX, por ejemplo si es la primera vez que usa el sistema, dónde está el usuario, dónde quiere ir, dónde ha estado en la sesión, dónde abandonó la última vez
16) **Navegación visible:** ofrecer la mínima navegación indispensable de forma clara y natural

UT6

Arquitectura cliente-servidor

- Tareas repartidas entre servidor y cliente
- Cliente realiza petición, servidor responde

Arquitectura monolítica vs microservicios

- **Arquitectura monolítica:** código base para realizar varias funciones empresariales, todos los componentes son interdependientes debido a los mecanismos de intercambio de datos dentro del sistema. Cambiar algo lleva mucho tiempo porque se afecta a gran parte del sistema
- **Microservicios:** dividir el software en pequeños componentes o servicios independientes. Cada servicio realiza una única función y se comunica con otros servicios a través de una interfaz bien definida. Se pueden actualizar, modificar, implementar o escalar cada servicio independientemente

Arquitectura orientada a eventos (eda) y arquitectura orientada a servicios (soa)

- **Evento:** cambio significativo en un estado (por ejemplo comprar algo y que el estado pase de "en venta" a "vendido")
- **Arquitectura orientada a eventos:** diseño a partir de servicios pequeños y desacoplados que publican, consumen o enrutan eventos. Útil cuando hay asincronía e independencia entre componentes. En la arquitectura hay productores de eventos (por ejemplo un formulario que emite un evento cuando se completa) y consumidores de eventos (reciben y procesan eventos). Pueden haber middlewares que se encarguen de transportar eventos
- **Arquitectura orientada a servicios:** se utilizan componentes llamados servicios, cada uno ofrece una capacidad empresarial y pueden comunicarse con otros servicios mediante diferentes plataformas y lenguajes. Los servicios pueden ser reutilizados a lo largo de la aplicación

Caché y Redis

- **Caché:** memoria que guarda datos para ser utilizados en el futuro, agilizando el proceso de su uso
- **Redis** es una base de datos en memoria que guarda datos en la RAM, reduciendo la carga en una base de datos primaria mientras que acelera las lecturas. Redis se implementa como caché entre una aplicación y una base de datos, guardando datos frecuentes

GraphQL y gRPC

- **GraphQL** es un lenguaje de queries para la API. Permite definir la estructura de datos requerida y esta será la estructura devuelta por el servidor, evitando que se manden cantidades excesivas de datos. Soporta lectura, escritura y subscripción a cambios de información
- **gRPC** es un framework de llamadas a procedimientos remotos (un programa puede ejecutar funciones en otro espacio como si fuera una llamada local, generalmente llamando en otro equipo de una red comaprtida) usado para construir servicios distribuidos y aplicaciones que pueden comunicarse de manera eficiente y de alto rendimiento entre sí. Permite que clientes y servidores se comuniquen en diferentes lenguajes, y el cliente puede invocar métodos del servidor

Websocket y webhook

- **Websocket:** protocolo que permite que una aplicación y un servidor utilicen una conexión dúplex para comunicarse. En vez de ser pedido y respuesta, es una comunicación continua y en tiempo real. Un ejemplo es la página del banco con el cambio que se actualiza constantemente
- **Webhook:** permite a una aplicación enviar notificaciones o datos en tiempo real a otra aplicación cuando ocurre un evento. Estos datos se envían automáticamente cuando ocurren. El webhook tiene una URL a la que se le hace un POST cuando se le notifica que ocurrió un evento

MQTT y AMQP

- **MQTT:** protocolo que sirve para conexiones poco fiables o con ancho de banda limitado. Corre sobre un protocolo de transporte que asegure que los datos no se pierden y van ordenados. Hay una aplicación que notifica a todos los suscriptores, el suscriptor se suscribe a uno o más temas y recibe notificaciones cuando hay un mensaje publicado en estos temas. Es ideal en IoT o conexiones con dispositivos en lugares lejanos como el campo
- **AMQP:** comunicación asíncrona segura, confiable e interoperable entre aplicaciones. Existe una cola donde se guardan los mensajes, que serán enviados a los clientes luego.

MVVM y MVC

- **Model-View-ViewModel:** patrón de arquitectura que desacopla lo máximo posible la interfaz de usuario de la lógica de la aplicación.
    - El modelo es la capa de datos y lógica de negocio, tiene la información pero no las acciones o servicios que los manipulan
    - La vista representa la información a través de elementos visuales, lo que ve el usuario

    - El modelo de vista es un intermediario entre el modelo y la vista, la vista puede enlazar datos y avisa a la vista de cambios de estado mediante eventos
- **Modelo-vista-controlador:** patrón de arquitectura que separa los datos y lógica del negocio de la representación
    - El modelo es la representación de la información, gestiona consultas y actualizaciones de datos. Envía datos a la vista para que los muestre
    - El controlador es intermediario entre el modelo y vista, responde a eventos e invoca peticiones al modelo cuando se hace una solicitud sobre información. Puede enviar comandos a la vista si hay cambios en la forma en que se presenta el modelo
    - La vista presenta el modelo al usuario

API Gateway, forward proxy, reverse proxy

- **Forward proxy (lado del cliente):** intermediario entre cliente y servidor, toma las solicitudes del cliente y las deriva al servidor objetivo. Ofrece anonimato, seguridad, caché
- **Reverse proxy (lado del servidor):** intermediario entre servidor y cliente, los clientes se comunican con el proxy y este deriva la consulta al servidor adecuado. Ofrece balanceo de carga, seguridad, caché, disponibilidad (si cae un servidor manda a otro)
- **Api Gateway:** api que redirecciona solicitudes a otras api. Ofrece balanceo de carga, caché, transformación de datos, seguridad

PaaS, IaaS, SaaS

- **PaaS (platform as a service):** modelo de servicio de cloud computing que ofrece una plataforma en la nube flexible y escalable para desarrollar, desplegar, ejecutar y gestionar aplicaciones. No es necesario instalar ni manejar software
- **IaaS (infrastructure as a service):** modelo de servicio en la nube que ofrece recursos de infraestructura bajo demanda como computación, almacenamiento, redes y virtualización
- **SaaS (software as a service):** modelo basado en la nube que ofrece aplicaciones a través de navegadores que se pueden acceder bajo demanda