Investigacion para Diseño de Apps

Reyes Contreras Ramses

4 de Febrero del 2024

1. Fundamentos de Desarrollo de Software

En el amplio terreno del desarrollo de software, existen conceptos fundamentales que actúan como pilares sobre los cuales se construyen aplicaciones sólidas y efectivas. Desde el conjunto de ceros y unos, el lenguaje de las máquinas, hasta los paradigmas y arquitecturas que guían la creatividad humana y dan forma y estructura a nuestras creaciones. Pero muchas veces pasamos por alto estos conceptos o no siempre comprendemos del todo cuál es la relación entre cada uno de ellos y por qué son importantes.

El Codigo

En el corazón del desarrollo de software residen los lenguajes de programación, un tipo de lenguaje común que permite a los humanos comunicarse con las máquinas

Clasificaciones de Lenguajes de Programación

Por Niveles:

1. Lenguajes de Bajo Nivel: Ofrecen un control granular sobre el hardware, como el lenguaje ensamblador.

2. Lenguajes de Medio Nivel: Equilibran el control con la abstracción, como C y C++.

3. Lenguajes de Alto Nivel: Se centran en la legibilidad y la facilidad de programación, como Python y Ruby.

Por Compilación:

1. Lenguajes Compilados: Requieren ser traducidos por un compilador antes de la ejecución, como C++ y Rust.

2. Lenguajes Interpretados: Son ejecutados directamente por un intérprete, como Python y JavaScript.

Por Paradigma:

1. Lenguajes Orientados a Objetos: Se basan en objetos con atributos y métodos, ejemplificados por Java y C#.

2. Lenguajes Reactivos: Diseñados para sistemas que responden a estímulos, como Erlang y ReactiveX.

Paradigmas de Programacion

Los paradigmas de programación son enfoques estructurales que guían a los desarrolladores en la resolución de problemas de software. Los tres principales son:

- Programación Imperativa: Centrada en la secuencia de instrucciones y el cambio de estado.
- Programación Orientada a Objetos: Basada en la creación de objetos con atributos y métodos.
- Programación Funcional: Se enfoca en funciones puras y evita el cambio de estado.

Arquitectura de Software: Diseñando la Estructura

La arquitectura de software es la columna vertebral que sostiene una aplicación. Define cómo los componentes interactúan y se organizan para cumplir con los objetivos del sistema. La arquitectura garantiza la modularidad, escalabilidad y mantenibilidad de una aplicación.

Una buena arquitectura de software aborda varios problemas:

- Complejidad: Divide el sistema en módulos manejables.
- Mantenibilidad: Facilita actualizaciones y correcciones sin afectar a toda la aplicación.
- Escalabilidad: Permite un crecimiento sin problemas al agregar nuevos componentes.
- Reutilización: Fomenta la utilización de componentes probados en lugar de reinventar soluciones.

2. Aplicaciones nativas, no nativas y multiplataforma

1. App Nativa:
   Las aplicaciones nativas (Native App) son apps desarrolladas para un sistema operativo móvil concreto (iOS o Android normalmente), en el lenguaje de programación específico de cada plataforma. Esto quiere decir que una app nativa creada para Android no puede ser utilizada en un dispositivo iOS y viceversa.

   Ventajas:

   - Tienen el mejor rendimiento. Las aplicaciones nativas son las más rápidas y tienen un rendimiento superior a otros tipos de apps, ya que han sido optimizadas específicamente para el hardware y el sistema operativo del dispositivo.
   - Acceso completo e integración con las funciones hardware del dispositivo. Las apps nativas permiten

aprovechar al máximo las funcionalidades móviles: cámara, micrófono, lector biométrico de huella, sensores, redes inalámbricas (Wi-Fi, bluetooth...).

- Pueden funcionar sin acceso a Internet(funcionamiento offline) si han sido diseñadas para ello.

Desventajas

- Costes de desarrollo altos. Si queremos tener nuestra app disponible para los dos sistemas, necesitaremos dos líneas de desarrollo diferentes, ya que el código utilizado para un sistema no es reutilizable para otro.
- Complejidad de desarrollo. Necesitamos equipos expertos en el lenguaje específico de cada sistema. Por ejemplo, en Kotlin para Android y en Swift para iOS.
- Tiempo de desarrollo superior. De 4 a 6 meses.

Ejemplos

- Facebook
- Whatsapp
- Netflix
- Spotify

2. App No Nativa: En este caso, las Apps no nativas realmente son sitios webs especialmente diseñadas para navegadores móviles. A diferencia de las apps nativas o Multiplataforma, no necesitan ser descargadas, ya que se accede a ellas desde un navegador web.

   Emplea las mismas tecnologías de desarrollo que una web, como sería HTML, CSS o JavaScript. Así, estaríamos hablando de una web con apariencia de app, por lo que presentaría sus mismas limitaciones.

   Ventajas

   - Carácter multiplataforma, con una sola línea de desarrollo.
   - Fácil desarrollo, ya que se emplean tecnologías ampliamente conocidas.
   - Tiempo y coste de desarrollo bajo.

   Desventajas

   - Acceso limitado a las funciones del dispositivo.
   - No se pueden subir a las tiendas de aplicaciones.
   - Diferentes experiencias de usuario en función del navegador utilizado.
   - Necesidad de contar con conexión a Internet incluso si se cuenta con un modo pensado para ello. Esto es necesario para acceder a las posibles actualizaciones o para entrar por primera vez.

   Ejemplos

- Google Docs.
- Microsoft Office Online.
- Pixlr.

3. App Multiplataforma:
   Las aplicaciones multiplataforma combinan elementos de las aplicaciones nativas y las aplicaciones web. Estas aplicaciones se desarrollan utilizando tecnologías web como HTML, CSS y JavaScript, pero se empaquetan en un formato que puede ser instalado en un dispositivo móvil como cualquier otra aplicación nativa. Por tanto, podemos obtener una aplicación para varias plataformas con un único desarrollo.

   Ventajas

   - Menor coste gracias al uso de lenguajes de programación más conocidos, con una mayor disponibilidad de profesionales en el mercado.
   - Carácter multiplataforma, con una sola línea de desarrollo.
   - Acceso a algunas funcionalidades del móvil.
   - Reducción de los tiempos de desarrollo a 3 meses.
   - Se pueden subir a los markets de aplicaciones (App Store y Google Play), obteniendo así las ventajas que ello supone, como la opción de monetizar cobrando por descarga o la visibilidad y la accesibilidad.

   Desventajas

   - Rendimiento inferior a una app nativa. Suelen tener un tamaño considerable y, además, ser más lentas.
   - El acceso a las funciones del dispositivo es limitado.

   Ejemplos

   - Amazon.
   - Instagram.
   - Uber.
   - Gmail.

3. Patrones de diseño para moviles

En el contexto del desarrollo de aplicaciones móviles, los "patrones de diseño móvil"se refieren a soluciones generales y reutilizables que abordan problemas y desafíos de diseño comunes que se encuentran al diseñar aplicaciones móviles. Estos patrones de diseño constan de mejores prácticas y metodologías comprobadas que ayudan a los desarrolladores a crear aplicaciones móviles eficientes, estéticamente atractivas y fáciles de usar.

Los patrones de diseño móvil se pueden clasificar en varios tipos, incluyendo navegación, entrada e interacción, arquitectura

visual y de información.
A continuación, se presentan algunos de los patrones de diseño en interfaces móviles más comunes:

- Barra de acciones. La barra de acciones es una característica clave en muchas aplicaciones móviles. Se encuentra en la parte superior de la pantalla y generalmente contiene iconos o botones para acciones comunes, como «Volver» o «Compartir». Este patrón facilita la navegación y la interacción del usuario.
- Navegación por pestañas. La navegación por pestañas le permite a los usuarios cambiar entre diferentes secciones de una aplicación deslizando el dedo horizontalmente. Es especialmente útil cuando una aplicación tiene múltiples vistas o funciones principales.
- Tarjetas. Las tarjetas son elementos de diseño que contienen información o contenido, como imágenes y texto. Se utilizan en aplicaciones de redes sociales y noticias para mostrar publicaciones o artículos de una manera visualmente atractiva.
- Menú deslizante. El menú deslizante es un patrón de diseño que le permite a los usuarios acceder a diferentes secciones de la aplicación deslizando un panel desde el lado de la pantalla. Es una forma eficaz de organizar el contenido y las opciones de navegación.
- Pantallas de inicio. Las pantallas de inicio son la primera impresión que los usuarios tienen de una aplicación. Deben ser atractivas y proporcionar una visión general de lo que ofrece la aplicación.

Cómo utilizar patrones de diseño en interfaces móviles de manera efectiva

- Consistencia: mantén la consistencia en todo tu diseño. Usa los mismos patrones de diseño en toda la aplicación para que los usuarios se sientan cómodos y familiarizados.
- Pruebas de usuario: realiza pruebas de usuario para obtener retroalimentación sobre la usabilidad de tu diseño. Esto te ayudará a identificar áreas de mejora.
- Adaptabilidad: ten en cuenta la diversidad de dispositivos móviles y tamaños de pantalla. Asegúrate de que tu diseño sea adaptable y se vea bien en diferentes dispositivos.
- Simplicidad: mantén el diseño lo más simple posible. Evita la sobrecarga de información y opciones, lo que puede abrumar a los usuarios.

Bibliografias

1. Zottola, R. (2023, 24 agosto). Fundamentos del desarrollo de software: lenguajes, paradigmas y arquitectura. Medium. https://dr-zottola.medium.com/fundamentos-del-desarrollo-de-software-lenguajes-paradigmas-y-arquitectura-e39dbfd8f000

2. Nunez, L. (2023, 31 enero). Tipos de aplicaciones, características, ejemplos y comparativa — EMMA. — EMMA. https://emma.io/blog/tipos-aplicaciones-caracteristicas-ejemplos/

3. KeepCoding, R. (2023, 27 septiembre). Patrones de diseño en interfaces móviles. KeepCoding Bootcamps. https://keepcoding.io/blog/patrones-de-diseno-en-interfaces-moviles/