PROGRAMACIÓN MÓVIL

UN ENFOQUE DESDE LA INGENIERÍA DE SISTEMAS

ISBN (Digital): 978-628-7693-63-0 1a. edicón

HERIBERTO FERNANDO VARGAS LOSADA
Coordinador

Universidad de la Amazonia
Vigilada MinEducación

INSTITUCIÓN ACREDITADA EN ALTA CALIDAD

PROGRAMACIÓN MÓVIL

UN ENFOQUE DESDE LA INGENIERÍA DE SISTEMAS

ISBN (Digital): 978-628-7693-63-0
Primera edición 2026

ASIGNATURA PROGRAMACIÓN MÓVIL

Programa de Ingeniería de Sistemas
Facultad de Ingeniería
Universidad de la Amazonia

Esta obra deberá ser citada de la siguiente manera:
Vargas-Losada H.F. (Coordinador) (2026). Programación Móvil: un enfoque desde la Ingeniería de Sistemas. Editorial Universidad de la Amazonia. p.p. 160.
Tamaño (21,59 x 27,94 cm).

Incluye bibliografía.

ISBN (Digital): 978-628-7693-63-0

Palabras claves: Movil; Diseño; Desarrollo Movil

Depósito Legal: Biblioteca Nacional de Colombia.

Diagramación y diseño de cubierta
Los autores

Número y año de edición
Primera edición, 2026.
Tiraje: Online
CDD:

Vicerrectoría de Investigación e Innovación
Editorial Universidad de la Amazonia
Campus Porvenir: Calle 17 Diagonal 17 con Carrera 3F - Barrio Porvenir
Contacto: vrinvestigaciones@udla.edu.co - editorial@uniamazonia.edu.co

Florencia, Caquetá 2026.

Presentación

Este libro, diseñado como parte del plan de estudios de Ingeniería de Sistemas, ofrece una comprensión integral del desarrollo de aplicaciones móviles, abordando tanto los fundamentos como las técnicas avanzadas necesarias en el ámbito actual de la programación. Comienza ofreciendo una visión clara de los principios básicos de la programación móvil, diferenciándola de la programación tradicional. Se exploran las características que hacen únicos a los dispositivos móviles, como la necesidad de interfaces adaptativas, el manejo de recursos limitados y la importancia de la experiencia del usuario en este contexto. Esta primera sección proporciona el trasfondo necesario para entender la evolución y el papel fundamental que desempeñan las aplicaciones móviles en la sociedad.

A continuación, el libro profundiza en las diferentes arquitecturas utilizadas en el desarrollo de aplicaciones móviles, desde las más simples hasta las más complejas. Se explica cómo estructurar aplicaciones robustas y escalables utilizando las herramientas y entornos de desarrollo más relevantes, como Android Studio y Xcode. La configuración de un entorno adecuado para el desarrollo es clave para que los estudiantes puedan construir aplicaciones eficientes y funcionales desde el primer momento. Seguidamente, se desarrolla el proceso de construcción de aplicaciones, guiando al lector a través de todas las etapas necesarias para crear aplicaciones móviles. Esto incluye la creación de interfaces de usuario, la gestión de componentes y el manejo de eventos. Se proporciona un enfoque práctico que permite a los estudiantes aplicar lo aprendido para desarrollar

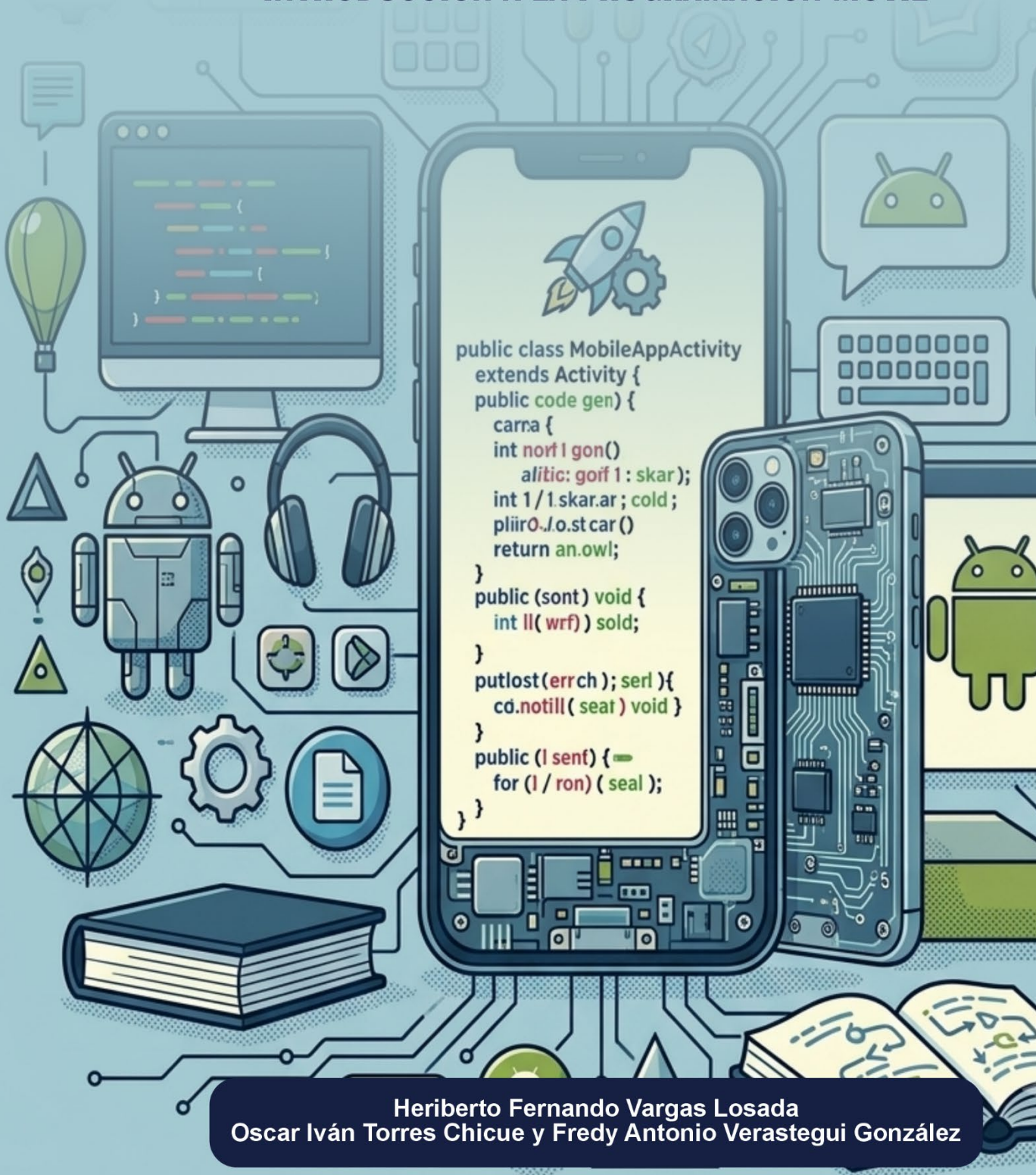
sus propias aplicaciones, tanto nativas como híbridas, comparando sus ventajas y limitaciones.

Finalmente, el texto aborda dos áreas esenciales: la persistencia de datos y la comunicación en tiempo real. Los estudiantes aprenderán a almacenar y recuperar información tanto localmente como en la nube, utilizando bases de datos y archivos, así como a integrar sus aplicaciones con servicios externos mediante el uso de APIs. Se cubren protocolos de comunicación que aseguran que las aplicaciones puedan interactuar con servidores remotos de manera efectiva, mejorando así la funcionalidad y la experiencia del usuario. En resumen, este libro es una herramienta clave para los estudiantes de Ingeniería de Sistemas, ya que no solo proporciona una sólida base teórica, sino también una serie de prácticas que preparan al lector para enfrentar los desafíos del desarrollo de software móvil.

Tabla de contenido

CAPÍTULO 1.

INTRODUCCIÓN A LA PROGRAMACIÓN MÓVIL



Heriberto Fernando Vargas Losada
Oscar Iván Torres Chicue y Fredy Antonio Verastegui González

CAPÍTULO 1.
Introducción a la programación móvil

1. Esquema
2. Competencias y Resultado de Aprendizaje
3. Metodología o Actividades Formativas de Enseñanza
4. Desarrollo Temático.
5. Actividades de Evaluación
6. Referencias Bibliográficas

Autores.

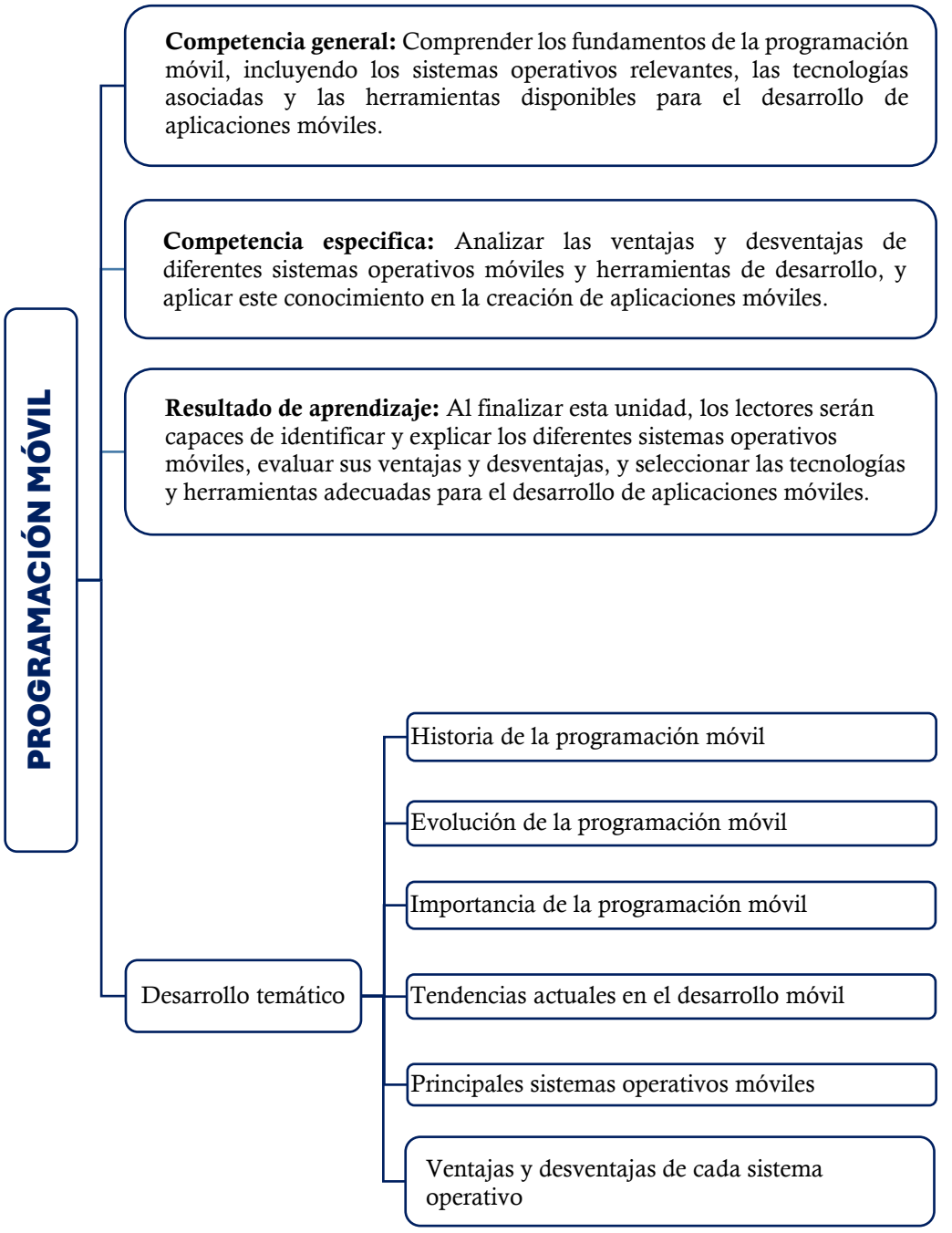
Heriberto Fernando Vargas Losada
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: heri.vargas@udla.edu.co.

Oscar Iván Torres Chicue
Estudiante del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: os.torres@udla.edu.co

Fredy Antonio Verastegui González
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: f.verastegui@udla.edu.co

Como Referenciar el Capítulo de libro.
Vargas Losada, H. F., Torres Chicue, O. I. & Verastegui González, F. A. (2026). Capítulo 1. Introducción a la Programación Móvil. En *Programación Móvil: un enfoque desde la Ingeniería de Sistemas* (págs. 7-53). Florencia: Universidad de la Amazonia.

1.1. ESQUEMA DEL CAPÍTULO 1: INTRODUCCIÓN A LA PROGRAMACIÓN MÓVIL



1.2. COMPETENCIAS Y RESULTADOS DE APRENDIZAJE

El objetivo principal de este Capítulo es proporcionar al lector una comprensión sólida de los fundamentos de la programación móvil. Esto incluye familiarizarse con la historia, evolución, importancia y las tendencias actuales de la programación, así como los sistemas operativos móviles relevantes. Al completar este capítulo, el lector estará equipado con la capacidad de identificar las características únicas de los dispositivos móviles, comprender la interacción entre los sistemas operativos y el hardware subyacente, y tener una visión integral del ecosistema de desarrollo móvil en su totalidad. Además, se espera que los lectores puedan adaptarse ágilmente a los rápidos cambios en la tecnología móvil, manteniéndose actualizados y relevantes en un campo en constante evolución en el ámbito de la programación móvil.

En este capítulo, se enfoca en desarrollar una competencia específica fundamental para el éxito en el desarrollo de aplicaciones móviles. El objetivo principal es capacitar al lector en el análisis exhaustivo de las ventajas y desventajas de diferentes sistemas operativos (SO) móviles y las herramientas de desarrollo asociadas. Esta habilidad es crucial para tomar decisiones informadas y efectivas durante el proceso de creación de aplicaciones móviles. Para lograr este objetivo, primero se explorarán las características clave de cada sistema operativo móvil, incluyendo su arquitectura, interfaz de usuario, seguridad y rendimiento. Este análisis detallado permitirá comprender cómo estas características impactan en el desarrollo de aplicaciones móviles y en la experiencia del usuario final. Además, se enfatizará la importancia de mantenerse actualizado con las últimas tendencias y avances en tecnología móvil, lo cual permitirá no solo comprender el estado actual del campo, sino también anticipar y adaptarse a los cambios futuros. Al adquirir este conocimiento, se estará mejor preparado para tomar decisiones informadas y estratégicas en el desarrollo de software móvil, asegurando que las aplicaciones sean relevantes, eficientes y seguras en un mercado en constante evolución.

Al concluir este capítulo, el lector habrá adquirido habilidades fundamentales para destacarse en el desarrollo de aplicaciones móviles. Será capaz de identificar y explicar los diferentes sistemas operativos (SO) móviles, comprendiendo sus arquitecturas, modelos de distribución y características distintivas. Además, podrá evaluar de manera crítica las ventajas y desventajas de cada SO en términos de desarrollo y distribución de aplicaciones móviles, considerando diversos aspectos como la popularidad del SO, la compatibilidad con hardware y software, la seguridad y la flexibilidad de personalización.

Finalmente, estará preparado para adaptarse a los cambios en el contexto tecnológico móvil, manteniéndose actualizado y continuando su aprendizaje en este campo en constante evolución. A continuación, en la Tabla 1.1, se definen los niveles de desempeño del resultado de aprendizaje.

Tabla 1.1
Niveles de desempeño del resultado de aprendizaje.

Avanzado (4.6 – 5)	Los estudiantes que alcanzan este nivel demuestran una comprensión excepcional de los sistemas operativos móviles. Son capaces de identificar con precisión y detalle los diferentes sistemas operativos, incluyendo sus características, arquitecturas y modelos de distribución. Además, realizan un análisis exhaustivo de las ventajas y desventajas de cada sistema operativo en términos de desarrollo y distribución de aplicaciones móviles, mostrando una comprensión profunda y crítica de los factores relevantes. Estos estudiantes son capaces de seleccionar y justificar de manera convincente las tecnologías y herramientas adecuadas para el desarrollo de aplicaciones móviles en una variedad de contextos, evidenciando una comprensión integral del contexto tecnológico móvil y su evolución.
Intermedio (4 – 4.5)	Los estudiantes en este nivel poseen una comprensión sólida de los sistemas operativos móviles. Son capaces de identificar y explicar correctamente los diferentes sistemas operativos móviles, junto con algunas de sus características y diferencias clave. Además, realizan un análisis básico de las ventajas y desventajas de cada sistema operativo en relación con el desarrollo y la distribución de aplicaciones móviles, demostrando una comprensión general de los principales factores a considerar. Estos estudiantes pueden seleccionar adecuadamente las tecnologías y herramientas para el desarrollo de aplicaciones móviles en situaciones simples o conocidas, aunque podrían requerir orientación adicional para tomar decisiones en contextos más complejos.
Básico (3 – 3.9)	Los estudiantes en este nivel tienen una comprensión básica de los sistemas operativos móviles. Son capaces de identificar los sistemas operativos móviles y algunas de sus características fundamentales, aunque pueden presentar algunas imprecisiones o confusiones. Su análisis de las ventajas y desventajas de los sistemas operativos es limitado, lo que refleja una comprensión superficial de los factores relevantes. Estos estudiantes pueden seleccionar tecnologías y herramientas para el desarrollo de aplicaciones móviles en contextos simples o familiares, pero es probable que necesiten apoyo adicional para enfrentar situaciones más complejas o desconocidas.

Bajo (0 – 2.9)	Los estudiantes en este nivel enfrentan dificultades para comprender los sistemas operativos móviles. Les resulta complicado identificar correctamente los diferentes sistemas operativos móviles y entender sus características básicas. Además, muestran dificultades para realizar un análisis significativo de las ventajas y desventajas de estos sistemas operativos y para seleccionar las tecnologías y herramientas adecuadas para el desarrollo de aplicaciones móviles. Es probable que estos estudiantes requieran un apoyo significativo y una revisión adicional para mejorar su comprensión y habilidades en este ámbito, así como para desarrollar una base más sólida en el conocimiento de los sistemas operativos móviles.

Nota. Los niveles de desempeño se basan en una escala de 5 puntos. Fuente: elaboración propia.

Al final de cada capítulo, se presenta una serie de actividades diseñadas para evaluar la comprensión de los conceptos clave abordados en la unidad. Estas actividades permiten aplicar los conocimientos adquiridos sobre sistemas operativos móviles, tecnologías de desarrollo y la selección de herramientas en situaciones prácticas, facilitando así la integración y el fortalecimiento del aprendizaje.

1.3. METODOLOGÍA DE ENSEÑANZA

El objetivo de esta metodología es proporcionar a los estudiantes una comprensión integral de la programación móvil, abarcando su historia, evolución, importancia, tendencias actuales, y una comparación de los principales sistemas operativos móviles, detallando sus ventajas y desventajas. Para lograr este objetivo, se utilizarán diversos materiales didácticos que faciliten el aprendizaje de los conceptos clave. Entre los materiales se incluyen presentaciones en formato PowerPoint para exponer de manera visual y organizada los aspectos históricos y técnicos de la programación móvil. También se proporcionarán artículos y lecturas complementarias en formato PDF, junto con enlaces a artículos actualizados sobre tendencias en el desarrollo móvil, para que los estudiantes tengan acceso a información relevante y reciente.

Además, se incorporarán videos educativos que consisten en clases grabadas y tutoriales sobre la evolución y la importancia de la programación móvil, disponibles en plataformas como YouTube y Coursera. Estos recursos visuales servirán para reforzar el contenido teórico de manera dinámica. Para complementar,

se utilizarán infografías y gráficos comparativos sobre los principales sistemas operativos móviles, como iOS y Android, resaltando sus ventajas y desventajas de una manera visual y fácil de comprender. Adicionalmente, se presentarán casos prácticos de aplicaciones móviles exitosas que ilustran la evolución y la importancia del desarrollo móvil en la industria actual. En cuanto a las técnicas de aprendizaje, se aplicarán estrategias variadas para facilitar la comprensión y la retención de los temas. Se promoverá el aprendizaje colaborativo mediante actividades en grupo donde los estudiantes investigarán y presentarán sobre la historia y evolución de la programación móvil, fomentando así el trabajo en equipo y el intercambio de ideas. El uso de mapas conceptuales permitirá a los estudiantes visualizar y conectar los conceptos clave de la evolución y las tendencias actuales en programación móvil. Asimismo, se llevarán a cabo sesiones de discusión guiada, donde los estudiantes podrán debatir sobre las ventajas y desventajas de los diferentes sistemas operativos móviles, desarrollando su pensamiento crítico.

Se incluirán talleres prácticos en los que los estudiantes realizarán ejercicios de programación básica en plataformas móviles para entender de forma práctica las tendencias actuales y la importancia de estas habilidades. El estudio de casos reales también será fundamental para ilustrar la evolución de la programación móvil y su impacto en la industria, permitiendo a los estudiantes relacionar la teoría con situaciones prácticas del mundo laboral. Para evaluar la comprensión y la aplicación de los conocimientos adquiridos, se utilizarán diversas herramientas de evaluación. Se realizarán cuestionarios de opción múltiple y respuesta corta para medir el conocimiento sobre la historia, evolución, y la importancia de la programación móvil. Además, los estudiantes trabajarán en proyectos grupales donde desarrollarán una pequeña aplicación móvil o un prototipo que incorpore las tendencias actuales del mercado, justificando la elección del sistema operativo. Las presentaciones orales permitirán a cada grupo exponer sus hallazgos sobre un sistema operativo móvil específico, destacando sus ventajas y desventajas. Asimismo, se asignarán reflexiones escritas en las que los estudiantes deberán elaborar ensayos cortos sobre la importancia de la programación móvil en la actualidad y su proyección futura. Se incluirán también actividades de

autoevaluación y coevaluación, para que los estudiantes puedan evaluar tanto su propio aprendizaje como el de sus compañeros, fomentando la reflexión y la autocrítica.

El desarrollo de estos contenidos se llevará a cabo en un periodo de seis semanas, dedicando cada semana a un tema específico: la primera semana se centrará en la historia de la programación móvil; la segunda, en su evolución; la tercera, en su importancia; la cuarta semana abordará las tendencias actuales en el desarrollo móvil; la quinta se enfocará en los principales sistemas operativos móviles, y la sexta semana se dedicará a analizar las ventajas y desventajas de cada sistema operativo, concluyendo con las evaluaciones finales.

En conclusión, esta metodología combina una variedad de materiales didácticos y técnicas de aprendizaje activo para ofrecer a los estudiantes una visión completa de la programación móvil. Las evaluaciones continuas están diseñadas para asegurar que los estudiantes no solo memoricen los conceptos, sino que realmente los comprendan y sean capaces de aplicarlos en contextos reales.

1.4. DESARROLLO TEMÁTICO

El contenido del capítulo aborda aspectos fundamentales y tendencias actuales en el desarrollo de aplicaciones móviles. En el Capítulo 1: Introducción, se establece una base sólida para comprender la programación móvil desde sus inicios hasta las tendencias contemporáneas. A través de un enfoque histórico, se examina la evolución de la programación móvil, destacando hitos significativos y avances tecnológicos clave en el desarrollo de aplicaciones para dispositivos móviles. Se analiza detalladamente la importancia de la programación móvil en el panorama actual, resaltando su impacto en la sociedad, la economía y la vida cotidiana. Además, se exploran las tendencias actuales en el desarrollo móvil, incluyendo el auge de las aplicaciones móviles, el crecimiento del mercado de dispositivos móviles y el impacto de la tecnología móvil en diversos sectores.

Un aspecto central del capítulo 1 es el estudio de los principales sistemas operativos móviles. Se realiza un análisis detallado de los sistemas operativos líderes en el mercado, como Android y iOS, examinando sus características, arquitecturas y modelos de distribución. Se discuten las ventajas y desventajas de cada sistema operativo en términos de mercado, desarrollo y distribución de aplicaciones, proporcionando una visión completa de las opciones disponibles para los desarrolladores. Para ilustrar los conceptos discutidos y mostrar su aplicación en situaciones del mundo real, se incluyen ejemplos prácticos y casos de estudio.

1.4.1. Historia de la programación móvil

La historia siempre ha sido un tema relevante en cualquier ámbito de la evolución del ser humano, y el desarrollo o programación móvil no es la excepción. Antes de adentrarse en la historia de la codificación de aplicaciones móviles, es importante definir el concepto de programación móvil. La programación móvil se define como el campo de la informática dedicado al desarrollo de software específicamente diseñado para dispositivos móviles, tales como smartphones y tabletas, que operan en sistemas operativos móviles como Android e iOS (Bautista-Sánchez, 2020; Patta et al., 2023). En esencia, este proceso implica la creación de aplicaciones móviles que brindan una amplia gama de servicios y funcionalidades para satisfacer las necesidades de los usuarios en movimiento.

La historia de la programación móvil se compone de una serie de eventos que abarcan desde los primeros dispositivos móviles hasta el ecosistema actual de aplicaciones móviles. Los primeros dispositivos móviles eran relativamente simples y contaban con capacidades limitadas, como el IBM Simón Personal Communicator, lanzado en 1994, que combinaba funciones de teléfono y PDA (Personal Digital Assistant) e incluía capacidades de envío de correos electrónicos y faxes, además de aplicaciones básicas como una agenda y una calculadora (Jin, 2017). Estos dispositivos marcaron los primeros pasos hacia la movilidad digital, aunque la programación en este contexto se limitaba principalmente a aplicaciones básicas.

A partir del año 2000, se integraron capacidades de procesamiento y conexión a redes, transformando los teléfonos móviles en los primeros dispositivos inteligentes, capaces de cumplir diversas funciones además de realizar llamadas (Caballé, 2013). Sin embargo, la verdadera explosión en el desarrollo de aplicaciones móviles ocurrió con el lanzamiento del iOS del iPhone y la App Store de Apple en 2007, conocido entonces como iPhone OS (Rojas-Lizarazo et al., 2011; Bermúdez-Moreno & López-Hincapié, 2011), y posteriormente con el sistema operativo Android en 2008 (Andreu-Ropero, 2011; Montes-León et al., 2014). Estos sistemas ofrecieron a los desarrolladores un entorno más robusto y flexible para crear aplicaciones, abriendo un nuevo mundo de posibilidades para llegar a una audiencia masiva de consumidores.

Este avance marcó el comienzo de una era en la que los desarrolladores podían acceder a un mercado amplio de consumidores ávidos de nuevas aplicaciones. Desde entonces, el desarrollo de aplicaciones móviles ha crecido exponencialmente, acompañado por la creación de nuevas tecnologías y herramientas que facilitan el proceso. Una de las tecnologías más importantes en el desarrollo de aplicaciones móviles es el lenguaje de programación Java, ampliamente utilizado en sistemas operativos como Android (Muyan-Özçelik, 2017; Syaifudin et al., 2021). Con el tiempo, surgieron debates sobre si desarrollar aplicaciones nativas específicas para cada plataforma (iOS, Android) o utilizar herramientas de desarrollo multiplataforma como Xamarin, React Native o Flutter (Masaad-Alsaid et al., 2021), que ofrecen mayor versatilidad y economía para los desarrolladores. Esta dicotomía sigue siendo relevante hoy en día, con ventajas y desventajas para ambos enfoques; la elección depende de las necesidades y objetivos del proyecto de desarrollo de la aplicación móvil. En la actualidad, las aplicaciones móviles son una parte fundamental de nuestras vidas, y su importancia solo seguirá creciendo en el futuro.

1.4.2. Evolución de la programación móvil

La evolución del desarrollo de la programación móvil ha sido un proceso constante y continuo, impulsado por la creciente demanda de los usuarios y los avances tecnológicos. En sus inicios, las aplicaciones móviles solo podían realizar funciones básicas como hacer llamadas y enviar mensajes, pero con el tiempo se han vuelto cada vez más sofisticadas, teniendo un impacto significativo en la sociedad. Con el aumento del uso de dispositivos móviles en la vida diaria, la demanda de aplicaciones móviles ha crecido exponencialmente, lo que ha llevado a una evolución continua de las tecnologías y metodologías utilizadas en su desarrollo. Las aplicaciones móviles se han convertido en una parte integral de la vida cotidiana, proporcionando servicios que abarcan desde el entretenimiento hasta la productividad, y desde la salud hasta la educación (Phongtraychack & Dolgaya, 2018). La programación móvil abarca diversos aspectos del desarrollo de aplicaciones, incluyendo el diseño de interfaces de usuario, la implementación de funcionalidades específicas utilizando lenguajes de programación como Java, Kotlin, Swift y Objective-C, así como la optimización del rendimiento y la seguridad de las aplicaciones (Syaifudin et al., 2022).

Desde el lanzamiento del primer sistema operativo móvil en 1994 hasta la popularización de las aplicaciones móviles en la década de 2000, la programación móvil ha experimentado importantes avances y transformaciones. Con la llegada de tecnologías como Java, iOS y Android, los desarrolladores han tenido acceso a nuevas herramientas y plataformas para crear aplicaciones más complejas y diversificadas. A medida que la tecnología sigue avanzando, se espera que la evolución del desarrollo de aplicaciones móviles continúe en el futuro. Una de las tendencias emergentes en este campo es el desarrollo de aplicaciones híbridas, que combinan características de las aplicaciones nativas y multiplataforma. Las aplicaciones híbridas se desarrollan utilizando tecnologías web como HTML, CSS y JavaScript (Bautista-Sánchez, 2020), pero pueden ser empaquetadas como aplicaciones nativas y acceder a funciones del dispositivo como la cámara y el GPS, lo que permite una mayor flexibilidad y una experiencia de usuario más cercana a la de las aplicaciones nativas.

Otra tendencia en evolución es la programación en la nube, donde las aplicaciones se ejecutan principalmente en servidores remotos y son accesibles a través de dispositivos móviles (Amazon Web Services, 2023). Esto proporciona mayor flexibilidad y escalabilidad, además de una mejor integración con otras tecnologías emergentes como el Internet de las Cosas (IoT).

Además, los desarrolladores de aplicaciones móviles deben adaptar sus creaciones a las características únicas de los dispositivos móviles, como el tamaño de pantalla, la capacidad de procesamiento y el consumo de energía, así como a las limitaciones impuestas por los diferentes sistemas operativos móviles. Este enfoque garantiza la compatibilidad y el funcionamiento adecuado de las aplicaciones en una amplia variedad de dispositivos y entornos (Bautista-Sánchez, 2020). La evolución del desarrollo y la programación móvil ha sido guiada por la demanda del mercado, lo que ha llevado a la creación de nuevas tecnologías y metodologías para satisfacer las necesidades cambiantes de los usuarios.

1.4.3. Importancia de la programación móvil

La programación móvil se refiere al proceso de creación de aplicaciones y software específicamente diseñados para dispositivos móviles como teléfonos inteligentes y tabletas. Este tipo de desarrollo ha adquirido una gran relevancia en la última década debido al aumento en el uso de estos dispositivos y a la creciente demanda de aplicaciones móviles por parte de los usuarios. La importancia del desarrollo móvil en la actualidad es indudable, ya que los dispositivos móviles se han convertido en herramientas fundamentales en la vida diaria de las personas. Una de las principales ventajas del desarrollo móvil es su capacidad para brindar una experiencia de usuario más personalizada y adaptada a las necesidades y preferencias individuales. Además, las aplicaciones móviles pueden mejorar la eficiencia y productividad en diferentes áreas, como el trabajo, la educación y la salud. Según un estudio del Pew Research Center, el 85% de los estadounidenses poseen un teléfono móvil y el 77% de ellos lo utilizan para acceder a internet. Además, el 91% de los adultos informan tener al menos una de estas tecnologías (Perrin, 2021).

El desarrollo móvil es una disciplina en constante evolución, y su importancia en la sociedad moderna es innegable. Su potencial para mejorar la vida de las personas y la eficiencia en diferentes ámbitos la convierte en un área de estudio y trabajo altamente demandada en la actualidad. Además, la programación móvil tiene un impacto significativo en la economía global. Según un informe de la GSMA, en 2023 "las tecnologías y servicios móviles generaron el 5,4% del PIB mundial, una contribución que ascendió a 5,7 billones de dólares de valor económico añadido", y la industria de aplicaciones móviles generó ingresos de 350 mil millones de dólares (GSMA, 2024). El desarrollo móvil es una industria en constante crecimiento que ofrece oportunidades de empleo a profesionales de diversos campos, como la informática, el diseño y el marketing. Esto se debe a que el proceso de creación de una aplicación móvil requiere diversas habilidades y conocimientos, lo que hace necesario un equipo multidisciplinario para su desarrollo exitoso.

La importancia del desarrollo móvil radica en su impacto en la vida diaria de las personas, su contribución a la economía global y su constante evolución e innovación en diferentes campos. Sin duda, esta disciplina seguirá siendo relevante en el futuro y es fundamental para el progreso tecnológico y social.

1.4.4. Tendencias actuales en el desarrollo móvil

En la actualidad, la programación móvil es un campo en constante evolución, marcado por la aparición de nuevas tendencias como el desarrollo de aplicaciones híbridas, la inteligencia artificial y la integración de tecnologías emergentes como la realidad aumentada y la realidad virtual en aplicaciones móviles. Estas innovaciones permiten a los desarrolladores crear experiencias más interactivas y personalizadas para los usuarios.

Además, con el avance de la tecnología 5G, las aplicaciones móviles tienen el potencial de volverse aún más sofisticadas, ofreciendo velocidades de conexión más rápidas y menor latencia, lo que facilita la implementación de funcionalidades avanzadas y la mejora de la experiencia del usuario. Esto incrementa la influencia

de las aplicaciones móviles en la vida cotidiana de las personas, ampliando su uso en áreas como la telemedicina, la educación a distancia, y las experiencias de entretenimiento inmersivas, consolidando aún más su relevancia en la sociedad moderna.

1.4.4.1. Aplicaciones hibridas

Antes de abordar la temática de las aplicaciones híbridas, es esencial considerar varios factores para determinar si estas tecnologías son adecuadas para el desarrollo de una aplicación, como los requisitos específicos de la aplicación, el presupuesto del cliente, el tiempo asignado para el proyecto y otros aspectos relevantes. No obstante, esta decisión a menudo se toma basándose en la experiencia de los desarrolladores. Las aplicaciones híbridas combinan características de aplicaciones nativas y aplicaciones web: pueden ser distribuidas a través de tiendas de aplicaciones y aprovechar algunas funcionalidades nativas del dispositivo (Zohud & Zein, 2021). A la vez, como también son aplicaciones web, dependen del HTML que se presenta en un navegador. Aunque pueden parecer y operar como aplicaciones nativas, en esencia siguen siendo aplicaciones web simples que se ejecutan en un navegador, de ahí el término "aplicación híbrida" (C. G. & Devi, 2021).

El desarrollo de aplicaciones móviles híbridas ofrece diversas ventajas, como un desarrollo más rápido y económico en comparación con las aplicaciones nativas, la capacidad de proporcionar una experiencia de usuario consistente en diferentes plataformas o navegadores, y la facilidad de mantener una única base de código que puede ser actualizada simultáneamente en todas las plataformas objetivo. Además, las aplicaciones híbridas pueden estar disponibles sin conexión, pueden utilizar las funciones del dispositivo móvil como si fueran aplicaciones nativas, y se basan en tecnologías web, permitiendo que estas aplicaciones se ejecuten en varios navegadores como cualquier otro sitio web o incluso como aplicaciones web progresivas (PWA).

Sin embargo, también presentan algunas desventajas. Las aplicaciones híbridas no son adecuadas para desarrollar aplicaciones que requieran un alto rendimiento, como los juegos que utilizan tecnologías 3D, ya que su rendimiento suele ser inferior al de las aplicaciones nativas debido a las limitaciones impuestas por la vista web. Además, la interfaz de usuario puede no ser óptima, ya que la experiencia de usuario no se puede personalizar tanto como en las aplicaciones nativas, lo cual puede afectar la percepción del usuario sobre la calidad y la fluidez de la aplicación (Janani, 2020).

1.4.4.2. Inteligencia artificial

La inteligencia artificial (IA, por sus siglas en inglés) ha experimentado un notable crecimiento en los últimos años y ha impactado significativamente el desarrollo móvil, integrándose en aplicaciones móviles de distintos sistemas operativos a medida que más personas las utilizan para mejorar su vida diaria. Al incorporar IA en aplicaciones móviles, los desarrolladores pueden introducir características cognitivas y lógicas que ofrecen una amplia variedad de funcionalidades para los teléfonos inteligentes (Mir & Llueca, 2020). Entre las características destacadas de la IA se encuentra el razonamiento automatizado, que permite proteger los softwares mediante el uso de matemáticas y lógica avanzadas para demostrar que los sistemas funcionan según lo previsto y de manera fiable, asegurando así un software verificado tanto en el presente como en el futuro. Un ejemplo de razonamiento automatizado es la capacidad de encontrar la ruta más corta para los vehículos cuando se encuentran atrapados en el tráfico. Además, la IA aplicada al desarrollo móvil permite etiquetar imágenes para la detección de objetos, segmentación semántica de píxeles e incluso para la clasificación de imágenes de escenas. Igualmente, se está avanzando en la detección de rostros humanos en imágenes digitales, una tecnología que se utiliza en numerosas aplicaciones con el objetivo de rastrear a una persona o un objeto (Narh & Ranjan, 2022). Esta integración de IA en el desarrollo móvil no solo mejora la funcionalidad de las aplicaciones, sino que también expande las posibilidades de interacción y personalización para los usuarios.

1.4.4.3. Realidad aumentada

La Realidad Aumentada (RA) está revolucionando las experiencias de los usuarios al elevarlas a nuevas alturas mediante la integración de superposiciones virtuales que se fusionan perfectamente con el mundo real, ofreciendo encuentros interactivos e inmersivos (Chatterjee, 2023). En esencia, la RA consiste en añadir contenido digital al entorno real y analizarlo para ofrecer una experiencia más enriquecida. Desde una perspectiva de RA, actualmente se tienen capacidades como colocar un objeto virtual en el mundo real y mantenerlo en su posición, escalar, rotar, mover y animar objetos virtuales, interactuar con ellos, medir objetos reales mediante anclajes virtuales, y generar efectos de sonido basados en la proximidad de la fuente (Appssemble, 2023). Para maximizar la efectividad de estas capacidades, es crucial diseñar aplicaciones que mejoren la experiencia de usuario y aumenten la inmersión (Anderies et al., 2023).

Un ejemplo destacado de una aplicación de Realidad Aumentada es Pokémon Go, lanzada por Niantic en 2016 para dispositivos Android y iOS. Esta aplicación utiliza el GPS del dispositivo para permitir a los usuarios localizar, capturar y combatir a personajes virtuales llamados Pokémon, que aparecen en el entorno físico del usuario. Aunque para los desarrolladores nuevos en RA puede ser un desafío crear una capa completamente nueva sobre el mundo real, los diseñadores y desarrolladores expertos no están limitados por los límites tradicionales ni por las leyes de la física. Aunque los objetos de RA no existen en el mundo real, tienen el poder de transformar la experiencia del usuario como si realmente existieran (Totla, 2021). La Realidad Aumentada, por lo tanto, no solo amplía las posibilidades creativas de los desarrolladores, sino que también redefine la manera en que los usuarios interactúan con su entorno, haciendo que lo digital y lo físico se entrelacen de formas innovadoras y sorprendentes.

1.4.4.4. Realidad virtual

El desarrollo de aplicaciones móviles y la Realidad Virtual (RV) son áreas tecnológicas distintas, pero su combinación permite la creación de experiencias inmersivas e interactivas en dispositivos móviles. En el ámbito de la RV, empresas como Google y Facebook han desarrollado gafas y dispositivos especiales que se conectan a teléfonos móviles, permitiendo a los usuarios explorar mundos virtuales. A pesar de estos avances, los escenarios más complejos aún requieren una gran potencia de procesamiento, lo que limita su uso principalmente a juegos que demandan altos niveles de rendimiento (Appssemble, 2023).

La Realidad Virtual ha transformado radicalmente la percepción y la interacción con los entornos digitales, expandiéndose ampliamente en diversas aplicaciones. En el desarrollo de aplicaciones móviles, la RV ofrece numerosos beneficios, especialmente en términos de rendimiento y experiencia del usuario. La RV proporciona una experiencia de usuario fluida y de alta calidad, lo que otorga a las empresas una ventaja competitiva significativa. Además, en muchas industrias que requieren monitoreo y seguimiento constante de sistemas y procesos, la RV puede ser especialmente útil. Las aplicaciones móviles basadas en RV pueden rastrear diversas ubicaciones de manera eficiente, facilitando la supervisión de múltiples elementos desde un teléfono inteligente (Singh, 2023).

Los desarrolladores pueden aprovechar la RV en aplicaciones móviles utilizando kits de desarrollo de software (SDK) que están diseñados para ofrecer experiencias inmersivas en dispositivos móviles. Esta integración de la RV en aplicaciones móviles no solo amplía las capacidades de los dispositivos, sino que también redefine la forma en que los usuarios interactúan con la tecnología, ofreciendo nuevas y mejoradas formas de entretenimiento, aprendizaje y gestión de tareas en la vida cotidiana.

1.4.4.5. Otras tendencias

Otras tendencias aplicadas en la programación móvil incluyen el Internet de las Cosas (IoT), que se centra en el desarrollo de aplicaciones para la automatización del hogar, permitiendo controlar dispositivos como parlantes, televisores, cámaras y aires acondicionados. A medida que avanza este campo, también se observa el desarrollo de aplicaciones no solo para el hogar y el entretenimiento, sino también para controlar equipos en fábricas, oficinas e incluso vehículos (C. G. & Devi, 2021).

Por otro lado, la tecnología 5G promete una reproducción a alta velocidad y con latencia cero para aplicaciones de gran tamaño, como películas, videos, animaciones y juegos, permitiendo el desarrollo de aplicaciones con grandes cantidades de datos que antes eran difíciles de manejar (Insights Success, 2019).

Asimismo, se está trabajando en la computación en la nube móvil, que combina la computación en la nube con la computación móvil a través de una red inalámbrica. Esta tecnología innovadora permite la entrega y ejecución de aplicaciones de alta calidad en dispositivos móviles, independientemente del sistema operativo, la capacidad de almacenamiento y las capacidades de procesamiento de estos. La computación en la nube móvil ofrece una mayor velocidad y flexibilidad tanto para los usuarios finales como para los desarrolladores de aplicaciones. Esta tecnología abarca datos basados en la nube y servicios diseñados específicamente para dispositivos móviles, integrando el desarrollo de aplicaciones móviles con servicios basados en la nube y facilitando la entrega de aplicaciones y servicios en la nube a los usuarios móviles. Los datos relevantes y la ejecución de aplicaciones se gestionan en centros de datos remotos (Simplilearn, 2023).

Las tendencias actuales en el desarrollo y programación móvil están marcando el ritmo de la innovación en la industria. Desde la creación de experiencias multiplataforma hasta la integración de tecnologías emergentes como la inteligencia artificial y la realidad aumentada, los desarrolladores están impulsando la evolución de las aplicaciones móviles para satisfacer las crecientes demandas de los usuarios y aprovechar las oportunidades del mercado. La

capacidad de adaptarse a las cambiantes expectativas y necesidades de los consumidores seguirá siendo fundamental para el éxito en este campo en constante evolución.

1.4.5. Principales sistemas operativos

Los sistemas operativos desempeñan un papel crucial en la determinación de la experiencia del usuario y las capacidades de los dispositivos móviles. En esta sección, se explorarán en detalle los principales sistemas operativos móviles que dominan el mercado actual, proporcionando a los lectores una comprensión sólida para tomar decisiones informadas tanto como usuarios como desarrolladores de aplicaciones móviles. Los sistemas operativos móviles han transformado la forma en que interactuamos con la tecnología, convirtiendo nuestros dispositivos móviles en herramientas potentes y versátiles que abarcan desde la comunicación hasta la productividad y el entretenimiento. Desde la icónica simplicidad de iOS de Apple hasta la amplia capacidad de personalización de Android de Google, y los esfuerzos de otras empresas por establecer su propio espacio en este competitivo mercado, la variedad y la innovación en los sistemas operativos móviles son amplias y dinámicas.

En este contexto, se examinarán los principales sistemas operativos móviles que impulsan nuestros teléfonos inteligentes y dispositivos móviles, analizando sus características distintivas, su impacto en la experiencia del usuario y su papel en la evolución continua de la tecnología móvil. Esta exploración permitirá a los lectores comprender mejor cómo cada sistema operativo contribuye a la usabilidad y funcionalidad de los dispositivos, y cómo influyen en la dirección futura del desarrollo móvil.

iOS

Es el sistema operativo desarrollado por Apple y es exclusivo de dispositivos como el iPhone, iPad y iPod Touch. Es reconocido por su estabilidad, seguridad y un ecosistema cerrado que integra de manera armoniosa el hardware y software de Apple. La primera versión de iOS fue presentada por Apple en 2007 bajo el nombre de iPhone OS, corriendo en la primera generación de iPhone. Desde entonces, Apple ha lanzado una nueva versión principal de este sistema operativo aproximadamente cada año, haciéndolo disponible para los productos compatibles de la marca (Universidad del Pacífico, 2020).

Desde su lanzamiento, Apple ha puesto un énfasis considerable en el diseño y la usabilidad de iOS, como señala Vieira-Jófili (2023). Este enfoque se manifiesta en una interfaz gráfica caracterizada por su estética limpia, organización elegante y simplicidad en la presentación de las aplicaciones. La interfaz de iOS se distingue por su minimalismo, con predominancia de colores planos y líneas simples, lo que proporciona una experiencia visual agradable y ordenada, facilitando la identificación y comprensión de los elementos por parte del usuario.

La usabilidad de iOS se configura como uno de sus pilares fundamentales. Su diseño cuidado, la interacción intuitiva y la simplicidad en la presentación de las aplicaciones se traducen en una experiencia de usuario fluida y familiar, convirtiendo a iOS en un sistema operativo accesible tanto para usuarios principiantes como para aquellos con mayor experiencia. Esta combinación de diseño y funcionalidad ha consolidado a iOS como una de las opciones preferidas en el mercado de sistemas operativos móviles.

iOS 1.0

En junio de 2007, Apple marcó un hito en la historia de la tecnología móvil con el lanzamiento del iPhone y su sistema operativo, inicialmente conocido como iPhone OS 1.0. Esta versión, que fue la precursora del actual iOS, estableció las bases para la experiencia de usuario que se conoce hoy en día. Es importante señalar que

iPhone OS 1.0 era compatible con un número limitado de dispositivos, específicamente la primera generación del iPhone y el iPod Touch. En esa época, la App Store aún no existía, lo que restringía a los usuarios a utilizar únicamente las aplicaciones preinstaladas y a gestionar su contenido multimedia a través de iTunes (Rubio, 2023).

iOS 2.0

Según Rubio (2023), el año 2008 marcó un antes y un después en la historia de los dispositivos móviles con la llegada de la App Store. Integrada en la segunda versión del sistema operativo, iPhone OS 2.0, y disponible en el iPhone 3G, la tienda de aplicaciones revolucionó la manera en que los usuarios interactuaban con sus dispositivos. En sus inicios, la App Store contaba con un número limitado de aplicaciones, apenas 500, pero su introducción permitió a los usuarios descargar e instalar aplicaciones de terceros, ampliando significativamente la funcionalidad y personalización de sus dispositivos. Esta innovación sentó las bases para el ecosistema de aplicaciones móviles que conocemos hoy en día.

iOS 3.0

En junio de 2009, Apple presentó iOS 3.0, una versión del sistema operativo que marcó un punto de inflexión en la convergencia entre el iPhone y el iPad. Lanzada junto con el iPhone 3GS, esta versión introdujo nuevas funcionalidades que ampliaron significativamente las capacidades de los dispositivos móviles de Apple. Entre las características más destacadas se incluyó el soporte para mensajes multimedia (MMS), lo que permitió a los usuarios enviar y recibir mensajes con imágenes, videos y audio, mejorando así la experiencia de comunicación. Además, iOS 3.0 incorporó la función de copiar y pegar, y amplió su compatibilidad con el iPad, sentando las bases para una mayor integración y usabilidad entre los diferentes dispositivos de la marca (Rubio, 2023).

iOS 4

El lanzamiento de iOS 4.0 en 2010 representó un hito en la evolución del sistema operativo, al introducir funcionalidades que permitieron a los usuarios personalizar y organizar mejor la pantalla de inicio de sus dispositivos. Una de las novedades más destacadas fue la posibilidad de utilizar imágenes personalizadas como fondos de pantalla, lo que ofreció un mayor nivel de personalización visual. Además, iOS 4.0 introdujo la función de carpetas de iconos, que facilitó la organización de las aplicaciones en la pantalla de inicio. Esta funcionalidad permitió a los usuarios agrupar aplicaciones por categorías o temas, optimizando el espacio y mejorando la accesibilidad (López, 2024).

iOS 5

Según López (2024), iOS 5 representó un salto significativo en la experiencia del usuario al introducir funcionalidades que optimizaron la fluidez y la integración del sistema operativo. Entre las novedades más destacadas se presentó el Centro de Notificaciones, que centralizaba alertas y notificaciones en un solo lugar, mejorando la gestión de información por parte del usuario. También se introdujeron las actualizaciones inalámbricas, eliminando la necesidad de conectar el dispositivo a un computador para instalar nuevas versiones del software. Además, iOS 5 integró iCloud, permitiendo la sincronización automática de datos entre dispositivos, y Siri, el asistente virtual de Apple, que añadió capacidades de control por voz y asistencia inteligente, entre otras funcionalidades importantes.

iOS 6

El lanzamiento de iOS 6 en 2012 representó un punto de inflexión en la estrategia de Apple para su sistema operativo móvil. Una de las novedades más destacadas fue la introducción de Apple Maps, que sustituyó a Google Maps como la aplicación de mapas predeterminada en los iPhone (López, 2024).

iOS 7

El lanzamiento de iOS 7 en 2013 significó un cambio significativo en la estética y funcionalidad del sistema operativo de Apple. Esta versión presentó un rediseño completo de la interfaz, estableciendo las bases del estilo visual que perdura hasta iOS 17. Además, iOS 7 incluyó la innovadora función AirDrop (CertiDeal, 2023).

iOS 8

Lanzado en septiembre de 2014, iOS 8 representó un avance significativo en la integración y expansión del ecosistema de Apple. Esta versión introdujo una serie de nuevas funciones que ampliaron las capacidades de los dispositivos iOS y fortalecieron la interconexión entre los diferentes productos y servicios de la compañía. iOS 8 permitió a los usuarios elegir entre una variedad de teclados de terceros, diversificando las opciones de escritura y personalización. Además, facilitó responder a las notificaciones sin necesidad de abrir la aplicación correspondiente, e integró la posibilidad de realizar llamadas de voz a través de redes Wi-Fi. También se añadieron nuevas funciones y servicios como Apple Music, HomeKit y Apple Pay (CertiDeal, 2023).

iOS 9

En septiembre de 2015, iOS 9.0 marcó un hito significativo al incorporar una serie de mejoras sustanciales en el rendimiento y la usabilidad del sistema operativo. Entre las innovaciones más destacadas se incluyeron la optimización del rendimiento, la búsqueda contextual, un mayor control sobre la publicidad, una multitarea mejorada y funciones para el cuidado de la salud visual, entre otras (CertiDeal, 2023). En conjunto, las mejoras introducidas en iOS 9.0 representaron un avance considerable en la experiencia del usuario.

iOS 10

La versión iOS 10, lanzada en 2016, introdujo una serie de cambios y mejoras significativas en el sistema operativo móvil de Apple. Entre las modificaciones más relevantes se destacaron el rediseño de las notificaciones, una navegación web optimizada, la renovación de la aplicación Fotos y la función de localización de dispositivos (Doonamis, 2022).

iOS 11

En el año 2017, la introducción de iOS 11 marcó un hito significativo en el sistema operativo de Apple al presentar una renovación integral del centro de control, así como una modificación en la visualización de las notificaciones en la pantalla de bloqueo. Es importante destacar que esta actualización solo es compatible con "dispositivos equipados con procesadores de 64 bits, lo que implica la exclusión de aquellos dispositivos con procesadores de 32 bits, tales como el iPhone 5, el iPhone 5c y el iPad de cuarta generación" (Doonamis, 2022).

iOS 12

Doonamis (2022) informa que, en septiembre de 2018, Apple presentó el lanzamiento del nuevo iOS 12, el cual debutó junto a los dispositivos iPhone XS y iPhone XS Max. Esta versión introdujo notables mejoras en el rendimiento y nuevas funcionalidades. Entre las novedades destacadas se encuentra la incorporación de Memoji, un sistema de Animoji altamente personalizable que enriquece la expresividad y el entretenimiento en los mensajes. Asimismo, los atajos de Siri han sido introducidos como una forma más eficiente de realizar diversas tareas, permitiendo la integración de cualquier aplicación con Siri, entre otras características añadidas.

iOS 13

Mora-Antoja (2019), informa que el 19 de septiembre de 2019 marcó el lanzamiento de iOS 13, una actualización que promete mejorar significativamente el rendimiento de los dispositivos compatibles. Según afirmaciones de Apple, el desbloqueo facial mediante Face ID ahora es un 30 % más rápido. Además, se han implementado medidas para reducir el espacio ocupado por las descargas de aplicaciones en un 50 %, y las actualizaciones ahora pesan un 60 % menos. La velocidad de apertura de aplicaciones se ha duplicado, lo que resulta en una experiencia más fluida para el usuario. Una de las características destacadas presentadas durante la WWDC fue la introducción del Modo Oscuro en iOS, ofreciendo una nueva estética a la interfaz. Asimismo, Apple ha desarrollado una alternativa para los inicios de sesión automáticos a través de Facebook o Twitter, permitiendo ahora autenticarse mediante Face ID y elegir si se desea compartir la dirección de correo electrónico personal o permitir que Apple genere una aleatoria para cada aplicación utilizada.

iOS 14

Apple (2024) comunica que iOS 14 representa una mejora significativa en la experiencia fundamental del iPhone, destacando por la introducción de widgets rediseñados para la pantalla de inicio, la implementación de una biblioteca de aplicaciones que reorganiza automáticamente las apps, y un diseño compacto completamente renovado tanto para las llamadas telefónicas como para el asistente virtual Siri. Entre las mejoras adicionales, la aplicación Mensajes ahora permite anclar conversaciones y presenta mejoras en la funcionalidad de grupos y Memojis. Por su parte, la aplicación Mapas ha sido enriquecida con indicaciones específicas para ciclistas y la inclusión de guías para descubrir nuevos lugares de interés. Asimismo, la incorporación de App Clips proporciona una manera rápida de explorar y utilizar funciones específicas de una aplicación, mientras que las nuevas características de privacidad aumentan la transparencia del usuario y brindan un mayor control sobre cómo las aplicaciones acceden a la ubicación, fotos, micrófono y cámara del dispositivo.

iOS 15

iOS 15 introduce una serie de mejoras significativas en diversas áreas, incluyendo avances en las capacidades de audio y video de FaceTime, como el audio espacial y el modo Retrato. Se presenta la sección "Se compartió contigo", que destaca en las aplicaciones correspondientes los elementos recibidos en conversaciones de Mensajes, incluyendo artículos, fotografías y otros contenidos relevantes. Los nuevos enfoques ayudan a mitigar distracciones al filtrar las notificaciones según la actividad en curso del usuario. Las notificaciones han sido rediseñadas y se ha implementado un nuevo resumen de notificaciones, permitiendo a los usuarios ponerse al día en su propio tiempo. La aplicación Mapas cuenta con un nuevo diseño estético, ofreciendo una experiencia tridimensional mejorada de las ciudades y la capacidad de visualizar rutas a pie mediante realidad aumentada. La función "Texto en vivo" utiliza la inteligencia del dispositivo para identificar texto en fotos, aplicándose a todo el sistema y la web. Finalmente, se han añadido nuevos controles de privacidad en Siri, Mail y otras áreas, con el objetivo de proporcionar mayor transparencia y otorgar al usuario un control más amplio sobre sus datos personales (Apple, 2024).

iOS 16

Según la página oficial de Apple (2024), iOS 16 introduce una renovada pantalla de bloqueo con nuevas opciones de personalización y widgets diseñados para proporcionar información instantánea con solo un vistazo. Esta actualización permite vincular la pantalla de bloqueo a un enfoque específico y utilizar filtros para reducir distracciones provenientes de las aplicaciones. En la aplicación Mensajes, se han añadido mejoras que permiten editar o cancelar el envío de un mensaje recién enviado. La función de Consulta Visual facilita aislar un sujeto del fondo de una imagen, permitiendo su copiado y pegado en aplicaciones como Mail y Mensajes. Además de estas innovaciones, iOS 16 incluye actualizaciones adicionales para aplicaciones clave como Mail, Mapas, Wallet, Salud, entre otras.

iOS 17

Según Apple (2024), iOS 17 incorpora impresionantes actualizaciones en las funciones de Teléfono, Mensajes y FaceTime, brindando nuevas formas de expresión durante las conversaciones. La introducción del modo En espera, disponible al cargar el dispositivo en posición horizontal, ofrece una experiencia de pantalla completa con información relevante accesible de manera rápida y remota. AirDrop facilita el intercambio de contenido y la comunicación con personas cercanas, mientras que NameDrop simplifica el intercambio de información de contacto. Las mejoras en el teclado permiten una entrada de texto más rápida y eficiente. Además, iOS 17 incluye actualizaciones para los widgets, Safari, Música, AirPlay, entre otras funciones destacadas.

Android

Desarrollado por Google, Android es el sistema operativo móvil más utilizado en el mundo, reconocido por su flexibilidad, capacidad de personalización y amplia disponibilidad en una variedad de dispositivos de diferentes fabricantes (Android, 2024). Android tiene sus raíces en los esfuerzos de Android Inc., una compañía fundada en 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White, que inicialmente se enfocó en desarrollar un sistema operativo para cámaras digitales. Sin embargo, pronto comprendieron que el mercado de cámaras no ofrecía el alcance necesario para sus ambiciones. A principios de la década de 2000, el panorama tecnológico estaba en plena transformación, con los teléfonos inteligentes, equipados con pantallas táctiles y acceso a internet, comenzando a dominar el mercado y desplazando gradualmente a los teléfonos tradicionales.

En 2005, "Google adquirió Android Inc. con el propósito de ingresar al mercado de dispositivos móviles, reconociendo en los smartphones una oportunidad para ampliar su presencia en el sector tecnológico. Esta compra fue una decisión clave para lograr ese objetivo" (Ridge, 2023). Antes de 2007, Android se mantuvo en secreto, pero ese año, la Open Handset Alliance, un consorcio tecnológico que incluía a empresas como HTC, Samsung, T-Mobile, Qualcomm y Google, hizo el anuncio oficial del lanzamiento de Android. Tras su presentación, el desarrollo del sistema operativo progresó rápidamente, y en 2008 se lanzó el HTC Dream, el primer smartphone con Android. Desde entonces, Google ha continuado apoyando el sistema operativo, expandiéndolo a múltiples plataformas y actualizándolo con frecuencia, con versiones que llevan nombres de postres y dulces (Ibertrónica, 2013).

Android 1.6 Donut

El 15 de septiembre de 2009, Google lanzó Android 1.6 Donut, una actualización significativa que introdujo varias mejoras y nuevas funcionalidades, según Ramírez (2022). Uno de los cambios más destacados fue la introducción del cuadro de búsqueda rápida, que permitía realizar búsquedas directamente dentro de la aplicación, en lugar de redirigir al navegador web como lo hacía anteriormente el

widget de búsqueda de Google. Además de las mejoras en la interfaz de usuario, Android 1.6 Donut incorporó cambios importantes en el sistema operativo subyacente, permitiendo que este se adaptara dinámicamente al tamaño y la resolución de la pantalla, lo que facilitó la compatibilidad con una amplia variedad de configuraciones de dispositivos.

Entre otras novedades, Android 1.6 Donut incluyó un sintetizador de voz en múltiples idiomas, mejoras en la cámara y la galería, como la capacidad de seleccionar y eliminar varias fotos simultáneamente, soporte para redes CDMA y conexiones VPN. Estas mejoras contribuyeron a expandir la funcionalidad y la versatilidad del sistema operativo en diversos dispositivos y regiones.

Android 2.3 Gingerbread

El 6 de diciembre de 2010, Google lanzó Android 2.3 Gingerbread, de acuerdo con Ramírez (2022), esta versión de Android se estrenó junto con el nuevo Nexus S, fabricado por Samsung. Esta actualización marcó un punto de madurez en el desarrollo de Android, centrándose en mejoras y refinamientos en lugar de introducir nuevas funcionalidades importantes. Gingerbread introdujo pequeños ajustes en la interfaz de usuario, adoptando iconos de acento de color verde Android. También se mejoró la compatibilidad con pantallas de mayor resolución, como WXGA y superiores. Igualmente se añadieron APIs para juegos, soporte para NFC y soporte para múltiples cámaras en un dispositivo.

Android 4.4 KitKat

Ramírez (2022) indica que Android 4.4 KitKat, fue lanzado en 2013, y se convirtió en una de las versiones más emblemáticas de Android. Su nombre pegadizo, resultado de una colaboración con Nestlé, y su amplia adopción la convirtieron en una versión muy utilizada durante muchos años. KitKat introdujo el modo inmersivo, que ocultaba la barra de estado y la barra de navegación para proporcionar una experiencia más inmersiva en las aplicaciones. También se eliminó el botón físico de menú y lo reemplazó con un menú de desbordamiento (el botón de tres puntos verticales) para mostrar opciones adicionales que no cabían en

la barra de acción. KitKat introdujo ART (Android Runtime) como un reemplazo experimental para la máquina virtual Dalvik. ART mejoró el rendimiento de las aplicaciones y la eficiencia de la memoria.

Android 5.0 Lollipop

Android Lollipop, lanzado en 2014, marcó un punto de inflexión en el desarrollo de Android con la introducción de Material Design y varias mejoras técnicas significativas. ART reemplazó oficialmente a la máquina virtual Dalvik, lo que mejoró el rendimiento y la eficiencia de la memoria. Este conjunto de mejoras se centró en la optimización de la batería y el rendimiento, incluyendo un modo de ahorro de energía y la programación de tareas para que se ejecuten solo con Wi-Fi. Se agrego al protección antirrobo tras el reinicio de fábrica y el soporte oficial para múltiples tarjetas SIM (Ramírez, 2023).

Android 6.0 Marshmallow

Android 6.0 Marshmallow, lanzado en octubre de 2015, Ramírez (2023) indica que esta versión se centró en optimizar y cohesionar el sistema operativo Android después de casi una década de desarrollo. Marshmallow introdujo permisos en tiempo de ejecución, lo que permitió a las aplicaciones solicitar permisos específicos (como acceso a la cámara o al micrófono) solo cuando fuera necesario, en lugar de exigir permisos generales al instalar la aplicación. Marshmallow añadió soporte para USB-C, modo 4K para aplicaciones y multiventana (experimental). Por último, se añadió el soporte nativo para lectores de huellas dactilares entre otras funciones.

Android 7.0 Nougat

Android 7.0 Nougat, lanzado en 2016, continuó el enfoque de Android 6.0 Marshmallow en la optimización y la refinación del sistema operativo. Nougat mejoró la función Doze de Marshmallow para que fuera efectiva incluso cuando el dispositivo estaba en movimiento. También se introdujo un nuevo compilador JIT (Just-In-Time) que redujo el tiempo de instalación de las aplicaciones en un 75% y requirió menos espacio de almacenamiento. Por último, se añadió soporte para la

API de gráficos Vulkan 3D, que proporcionó mejoras en el rendimiento gráfico para los juegos entre otras funcionalidades que mejoraron la experiencia del usuario (Ramírez, 2023).

Android 8.0 Oreo

Android 8.0 Oreo, lanzado en 2017, introdujo mejoras significativas en las áreas de seguridad, rendimiento y fragmentación. Oreo introdujo Google Play Protect, un servicio que escanea regularmente las aplicaciones en busca de programas malignos. También se eliminó la opción "Orígenes desconocidos", lo que obligó a los usuarios a especificar qué aplicaciones podían instalar archivos APK, entre otras mejora de rendimiento y funcionalidades (Universidad Politécnica de Valencia, 2021).

Android 9.0 Pie

Android 9.0 Pie, lanzado en 2018, UPV (2021), indica que se introdujeron mejoras significativas en las áreas de navegación por gestos, inteligencia artificial y bienestar digital. Pie reemplazó los tres botones de navegación en pantalla con dos gestos: gesto de retroceso y gesto de inicio. También se implementó la inteligencia artificial para mejorar varios aspectos del sistema operativo: aprendizaje de hábitos de uso de aplicaciones y predicción de carga de aplicaciones. Igualmente se introdujeron varias funciones para promover el uso responsable y saludable del dispositivo móvil. En general Android 9.0 Pie introdujo una serie de mejoras significativas que mejoraron la experiencia del usuario y promovieron el uso responsable del dispositivo móvil.

Android 10

Android 10, lanzado en 2019, marcó un cambio significativo en la marca y las características de Android. Google abandonó los nombres de postre para las versiones de Android, adoptando un sistema de numeración simple. El logotipo de Android se simplificó para incluir solo la cabeza del robot en un tono verde más claro. Android 10 eliminó los botones de navegación en pantalla y adoptó un sistema de navegación por gestos similar al de iOS. Se incluyó un modo oscuro nativo para

pantallas OLED, lo que resultó en un ahorro significativo de batería. Por último, se añadió el soporte para 5G, Wi-Fi 6, WPA3, teléfonos plegables y presión en pantallas táctiles, entre otras funciones importantes (Universidad Politécnica de Valencia, 2021).

Android 11

Android 11, lanzado en 2020, introdujo mejoras significativas en las áreas de compatibilidad, seguridad y privacidad. Se mejoró la compatibilidad con dispositivos plegables y conectividad 5G. Igualmente, se introdujo Project Mainline, que permitió a Google actualizar componentes del sistema operativo a través de Google Play, independientemente de las actualizaciones del fabricante del dispositivo. Se incluyó soporte para la autenticación de llamadas STIR/SHAKEN, que ayudó a prevenir las llamadas fraudulentas. Además de otras funcionalidades y mejoras extras, se introdujo una nueva opción de permiso de ubicación que permitió a los usuarios otorgar acceso de ubicación a las aplicaciones solo una vez (Seidor, 2023).

Android 12

Seidor (2023) indica que a partir de noviembre de 2021 se produjo la distribución de Android 12, una nueva versión del sistema operativo móvil desarrollado por Google que introduce diversas mejoras y funcionalidades. Entre las más relevantes se encuentra la incorporación de un menú flotante específico para videojuegos, el cual facilita el acceso a funciones como la grabación de la pantalla o la captura de imágenes. Asimismo, Android 12 presenta la función App Pairs, que permite abrir dos aplicaciones de forma simultánea en una pantalla dividida, y la posibilidad de utilizar el teléfono como llave digital para acceder a vehículos compatibles.

Android 13

En el mes de marzo de 2022 se produjo el lanzamiento oficial de Android 13. Esta actualización trae consigo una serie de mejoras y funcionalidades que buscan optimizar la experiencia del usuario. Android 13 incluye nuevas funciones para

ayudar a los usuarios a gestionar su tiempo frente a la pantalla y mejorar su bienestar digital. El portapapeles se ha visto renovado para ser más seguro y privado. Ahora, los usuarios pueden controlar qué aplicaciones tienen acceso al historial del portapapeles. Las aplicaciones ahora deben solicitar permiso para acceder a cada tipo de dato de forma individualizada, lo que brinda a los usuarios un mayor control sobre su información personal (Seidor, 2023).

Android 14

De acuerdo con la página oficial de Android (2024), desde octubre de 2023, la versión final y estable de Android 14 se encuentra disponible para los usuarios, brindando una experiencia más personalizada, segura y accesible. Esta actualización del sistema operativo introduce una serie de mejoras y funcionalidades que optimizan el uso del dispositivo móvil. Los usuarios tienen la posibilidad de crear temas personalizados que se ajustan a sus preferencias, incluyendo nuevos fondos de pantalla y estilos de iconos. Android 14 incorpora la generación de fondos de pantalla personalizados mediante inteligencia artificial, adaptándose al estilo del usuario y a las condiciones ambientales. El sistema operativo proporciona información clara y comprensible sobre los permisos que solicitan las aplicaciones, permitiendo al usuario tomar decisiones informadas sobre el acceso a su información personal. Android 14 se actualiza de forma regular con parches de seguridad para proteger al usuario contra programas malignos, virus y otras amenazas. Se han implementado nuevas funciones para mejorar la experiencia de uso para personas con discapacidades visuales, auditivas o motoras.

Se ha llevado a cabo un exhaustivo estudio del sistema operativo Android y iOS, así como de sus diversas versiones, con el objetivo de ofrecer un análisis detallado y comprensible para el lector. Si bien nos centraremos en Android, es importante destacar que los lectores son libres de investigar y explorar también el sistema operativo iOS y sus variantes. Además, en unidades posteriores se abordarán otras temáticas relevantes como las tecnologías y herramientas de desarrollo asociadas, incluyendo los SDK, IDEs y otras herramientas fundamentales para el desarrollo de aplicaciones móviles.

1.4.6. Ventajas y desventajas de cada sistema operativo

En el ámbito de los dispositivos móviles, la elección del sistema operativo puede resultar crucial tanto para los usuarios como para los desarrolladores de aplicaciones. En esta sección, exploraremos en detalle las ventajas y desventajas de los dos principales sistemas operativos móviles: Android y iOS. Proporcionando una visión completa para aquellos que buscan comprender y tomar decisiones informadas sobre el desarrollo y el uso de aplicaciones móviles en Android y iOS.

Android, desarrollado por Google, es conocido por su flexibilidad y personalización. Su naturaleza de código abierto permite a los fabricantes de dispositivos adaptarlo a una amplia gama de hardware, lo que conduce a una diversidad de opciones para los usuarios. Además, la tienda de aplicaciones Google Play Store ofrece una gran variedad de aplicaciones, muchas de ellas gratuitas. Sin embargo, esta fragmentación puede llevar a problemas de compatibilidad y seguridad en algunos dispositivos. Por otro lado, iOS, el sistema operativo de Apple, se destaca por su estabilidad y optimización. La integración vertical entre hardware y software permite un rendimiento fluido y una experiencia de usuario coherente en todos los dispositivos de la marca. Además, la App Store de Apple ofrece un ecosistema cuidadosamente controlado con altos estándares de calidad y seguridad. Sin embargo, está cerrada política puede limitar la libertad de personalización y elección para algunos usuarios y desarrolladores.

1.4.6.1. Ventajas de Android

Ahora, se explorarán algunas ventajas que puede ofrecer el sistema operativo Android. De acuerdo con BLU Radio (2023), Android ofrece una gran cantidad de opciones para los usuarios, ya que está disponible en una amplia gama de dispositivos de diferentes marcas y precios. Por otro lado, el sistema operativo Android se distingue por su mayor flexibilidad y capacidad de personalización. Esta característica permite a los usuarios adaptar sus dispositivos a sus necesidades y preferencias individuales, brindándoles un mayor control sobre la experiencia de usuario (Universidad Tecnológica del Perú, 2024). Además, la naturaleza de código

abierto del sistema operativo Android constituye una de sus principales características distintivas. Esta particularidad implica que el código fuente de Android se encuentra disponible públicamente, permitiendo que desarrolladores de todo el mundo puedan acceder, modificar y distribuir libremente el mismo (Reclu IT, 2021). Por último, la integración con los servicios de Google constituye una de las ventajas competitivas del sistema operativo Android. Esta característica facilita la sincronización automática de las cuentas de usuario en diferentes dispositivos Android, incluyendo Gmail, Google Calendar, Google Drive y otros servicios de Google. Esta integración se traduce en una experiencia de usuario fluida y eficiente. Los usuarios pueden acceder sin problemas a sus correos electrónicos, calendarios, documentos, fotos y demás información personal desde cualquier dispositivo Android, sin necesidad de configuraciones adicionales (Wabimovil, 2023).

Las diversas ventajas que ofrece Android son verdaderamente destacables y juegan un papel crucial en el desarrollo móvil. Su naturaleza de código abierto y su amplia disponibilidad en una variedad de dispositivos permiten a los desarrolladores llegar a una audiencia más amplia y diversa. La flexibilidad y personalización que ofrece Android proporcionan a los creadores de aplicaciones la libertad de innovar y experimentar con nuevas ideas, adaptándose a las necesidades específicas de los usuarios. Además, la comunidad de desarrolladores de Android proporciona recursos, herramientas y soporte para ayudar en el proceso de desarrollo. La integración con los servicios de Google facilita la creación de aplicaciones potentes y conectadas. En conjunto, estas ventajas brindan un entorno propicio para el desarrollo móvil, permitiendo a los desarrolladores crear experiencias únicas y satisfactorias para los usuarios en una plataforma dinámica y en constante evolución.

1.4.6.2. Desventajas de Android

Campos-Gavilánez (2023), indica que es innegable que la capacidad de multitarea puede representar una ventaja considerable en términos de productividad y eficiencia al permitir la realización simultánea de múltiples tareas. No obstante, es importante reconocer que esta práctica también conlleva ciertos inconvenientes,

particularmente en dispositivos móviles, donde puede resultar en un consumo considerable de la batería. Además, la multitarea puede incentivar la instalación de aplicaciones superfluas, las cuales, al ocupar espacio en la memoria del dispositivo, pueden contribuir a un rápido agotamiento de esta y afectar adversamente la velocidad y el rendimiento general del dispositivo. Asimismo, es importante señalar que la falta de soporte de actualizaciones constituye otro factor a considerar, dado que los dispositivos más recientes tienden a recibir actualizaciones durante varios años luego de su lanzamiento, mientras que los dispositivos más antiguos pueden quedar limitados a su versión original sin recibir actualizaciones posteriores (Torres, 2020). La versatilidad de Android presenta un dilema de doble filo. Debido a la menor cantidad de restricciones en la creación de aplicaciones para la plataforma PlayStore, es posible encontrar una amplia gama de aplicaciones de calidad cuestionable, e incluso, en casos extremos, estas pueden constituir un riesgo de fraude o contener virus. Como contrapartida, los especialistas en control de calidad enfrentan el desafío de dedicar más tiempo a realizar pruebas exhaustivas para evaluar el rendimiento de las aplicaciones en una diversidad de modelos de teléfonos Android. Sin embargo, es importante destacar que esta desventaja no excluye la existencia de aplicaciones Android de alta calidad que satisfacen, al menos, las necesidades y requisitos básicos de los usuarios (Bleger, 2022).

Si bien Android presenta numerosas ventajas para el desarrollo móvil, también hay desafíos que los desarrolladores deben tener en cuenta. La fragmentación del mercado de dispositivos Android, con una amplia variedad de fabricantes y modelos, puede resultar en problemas de compatibilidad y optimización para ciertas aplicaciones. Esto requiere un esfuerzo adicional por parte de los desarrolladores para garantizar que sus aplicaciones funcionen de manera consistente en una variedad de dispositivos. Además, la diversidad de versiones de Android en el mercado puede dificultar la implementación de nuevas características y tecnologías, ya que los desarrolladores deben considerar la compatibilidad con versiones más antiguas del sistema operativo. Esto puede ralentizar el proceso de desarrollo y limitar la adopción de nuevas funcionalidades. Debido a la naturaleza abierta de Android y la diversidad de dispositivos, el sistema puede ser más

vulnerable a *malware* y ataques de seguridad en comparación con iOS. Esto requiere que los desarrolladores implementen medidas de seguridad adicionales en sus aplicaciones para proteger la privacidad y la seguridad de los usuarios.

1.4.6.3. Ventajas de iOS

Una de las principales fortalezas inherentes a iOS radica en su ecosistema integrado, caracterizado por la interconexión sin fisuras entre los diversos dispositivos de Apple, tales como iPhones, iPads y Macs. Esta cohesión facilita una experiencia de usuario altamente fluida, donde la transferencia de datos se ejecuta de manera eficiente y las transiciones entre dispositivos se realizan de manera impecable, promoviendo así un entorno de operatividad sin interrupciones para el usuario (KeepCoding, 2023). Apple es reconocida por su estrategia de actualizaciones sincronizadas en todos los dispositivos iOS compatibles, lo cual implica que los usuarios tienen acceso simultáneo a las últimas funcionalidades y mejoras en seguridad, sin importar el dispositivo que utilicen. Esta consistencia en las actualizaciones asegura una experiencia homogénea en todo el ecosistema de la marca (Wabimovil, 2023). Apple brinda un sólido respaldo a largo plazo para sus dispositivos iOS al proporcionar a los usuarios actualizaciones continuas durante varios años tras la adquisición de sus dispositivos. Este compromiso asegura que inclusive los modelos más veteranos puedan beneficiarse de las últimas funcionalidades y medidas de seguridad mejoradas (Rubio-Mazas, 2021). Por último, Apple ejerce un control integral sobre tanto el hardware como el software, lo que posibilita una integración fluida entre ambos componentes. Esta integración se traduce en un rendimiento óptimo y una eficiencia energética, dado que el sistema operativo se encuentra diseñado de manera específica para los dispositivos de la marca Apple (De la Calle, 2024).

Las ventajas ofrecidas por iOS son fundamentales para el desarrollo móvil y contribuyen significativamente a la excelencia en la experiencia del usuario. La estabilidad, optimización y cohesión del ecosistema iOS proporcionan a los desarrolladores un entorno sólido para crear aplicaciones de alta calidad y rendimiento excepcional. La integración vertical entre el hardware y el software en

los dispositivos de Apple garantiza una experiencia consistente y fluida para los usuarios, lo que facilita la creación de aplicaciones que aprovechan al máximo el potencial de los dispositivos iOS. Además, la rigurosa selección y curación de aplicaciones en la App Store de Apple asegura a los usuarios una amplia gama de aplicaciones seguras y de alta calidad. Para los desarrolladores, esto significa una mayor visibilidad y confianza en sus aplicaciones, lo que puede traducirse en mayores oportunidades de éxito en el mercado.

1.4.6.4. Desventajas de iOS

La Universidad Tecnológica del Perú (2024), indica que el costo de adquisición de los dispositivos móviles de Apple tiende a ser considerablemente superior al de sus contrapartes basadas en el sistema operativo Android, lo que puede representar una limitación económica significativa para numerosos usuarios. En términos de personalización, iOS se caracteriza por ofrecer opciones restringidas en comparación con Android. Los usuarios de dispositivos iOS se ven obligados a conformarse con la interfaz predeterminada de Apple, una restricción que puede resultar desfavorable para aquellos que buscan una mayor flexibilidad para adaptar su experiencia móvil a sus preferencias individuales. Además, el conjunto de herramientas de desarrollo de este sistema operativo solo está disponible a través de los dispositivos fabricados por la empresa, lo cual posiblemente incrementa los gastos asociados al desarrollo al demandar la adquisición de hardware particularmente diseñado para este propósito (Montatixe-Granada, 2022). iOS ha sido concebido con una orientación hacia una mayor sinergia con otros productos y servicios ofrecidos por Apple, lo cual podría restringir la libertad de los usuarios en la utilización de servicios proporcionados por terceros (BLU Radio, 2023).

Si bien iOS ofrece una experiencia de usuario excepcional y un entorno de desarrollo robusto, también presenta desafíos para los desarrolladores móviles. Una de las principales limitaciones es la restricción de la plataforma, que puede limitar la libertad de personalización y la flexibilidad para los desarrolladores. Apple mantiene un estricto control sobre su ecosistema, lo que puede dificultar la implementación de ciertas características o tecnologías en las aplicaciones. Además, la exclusividad

de iOS para los dispositivos de Apple puede limitar el alcance del mercado de una aplicación, ya que solo está disponible para un segmento específico de usuarios. Esto puede ser un obstáculo para los desarrolladores que buscan llegar a una audiencia más amplia y diversa. Otro desafío importante es el proceso de aprobación de la App Store, que puede ser largo y exigente. Si bien esto garantiza altos estándares de calidad y seguridad para los usuarios, puede retrasar el lanzamiento de nuevas aplicaciones y generar incertidumbre para los desarrolladores.

1.5. ACTIVIDADES DE EVALUACIÓN

Se comienzan las actividades de evaluación teóricas destinadas a profundizar en el contenido temático abordado hasta el momento. Estas actividades permitirán evaluar la comprensión y el dominio de los conceptos fundamentales relacionados con la programación móvil, así como identificar áreas de mejora. Seleccione la respuesta correcta.

¿Cuál de las siguientes opciones describe correctamente uno de los hitos importantes en la historia de la programación móvil?

a) El lanzamiento del iPhone en 2007.
b) El desarrollo del primer teléfono inteligente en 1992.
c) La introducción del primer sistema operativo móvil, Android, en 2008.
d) La creación de la primera aplicación móvil en 1994.

¿Qué tecnología marcó un punto de inflexión importante en la evolución de la programación móvil?

a) La introducción del lenguaje de programación Java para el desarrollo de aplicaciones móviles.
b) La adopción masiva de dispositivos con pantalla táctil.
c) La aparición de las tiendas de aplicaciones, como App Store y Google Play.
d) El lanzamiento de la primera versión de Android por Google.

¿Por qué es importante la programación móvil en la actualidad?

a) Porque permite a las empresas llegar a una audiencia más amplia a través de dispositivos portátiles.

b) Porque reduce los costos de desarrollo de software en comparación con otras plataformas.

c) Porque facilita la integración de tecnologías emergentes como la inteligencia artificial y la realidad aumentada.

d) Todas las anteriores.

¿Qué tecnología emergente se está integrando cada vez más en el desarrollo de aplicaciones móviles para ofrecer experiencias más inmersivas?

a) Aplicaciones híbridas.

b) Realidad aumentada.

c) Inteligencia artificial.

d) Todas las anteriores.

¿Cuál de las siguientes afirmaciones describe correctamente una característica principal del sistema operativo iOS de Apple?

a) Es un sistema operativo de código abierto.

b) Solo está disponible en dispositivos fabricados por Apple.

c) Permite una alta personalización por parte del usuario.

d) Es conocido por su amplia variedad de fabricantes de dispositivos.

¿Cuál de las siguientes afirmaciones describe una ventaja común de Android sobre iOS?

a) Mayor optimización de la duración de la batería.

b) Mayor control de seguridad y privacidad.

c) Mayor variedad de opciones de personalización.

d) Menor disponibilidad de aplicaciones en la tienda oficial.

¿Cuál de los siguientes es un sistema operativo de código abierto utilizado en dispositivos móviles?

a) iOS

b) Android

c) Windows Mobile

d) BlackBerry OS

¿Cuál es una ventaja típica de iOS en comparación con Android en términos de seguridad?

a) Mayor variedad de aplicaciones disponibles

b) Actualizaciones más frecuentes del sistema operativo

c) Mayor control sobre los permisos de las aplicaciones

d) Sistema operativo de código abierto

1.6. REFERENCIAS BIBLIOGRÁFICAS

Amazon Web Services. (2023). *¿Qué es la computación en la nube para móviles? - Explicación de la computación para móviles - AWS*. Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/mobile-cloud-computing/

Anderies, Marvella, M., Hakim, N. A., Seciawanto, P. A., & Chowanda, A. (2023). Implementation of Augmented Reality in Android-based Application to Promote Indonesian Tourism. *Procedia Computer Science*, *227*, 573-581. https://doi.org/10.1016/j.procs.2023.10.560

Andreu-Ropero, A. (2011). *Estudio del desarrollo de aplicaciones RA para Android* [Bachelor thesis, Universitat Politècnica de Catalunya]. https://upcommons.upc.edu/handle/2099.1/11284

Android. (2024a). *Android 14 | Personalización, accesibilidad y protección | Android*. Android. https://www.android.com/intl/es_us/android-14/

Android. (2024b). *Qué es Android*. Android. https://www.android.com/intl/es_es/what-is-android/

Apple. (2024a, febrero 26). *Acerca de las actualizaciones de iOS 14—Soporte técnico de Apple (CO)*. Apple Support. https://support.apple.com/es-co/118390

Apple. (2024b, marzo 6). *Acerca de las actualizaciones de iOS 15—Soporte técnico de Apple (CO)*. Apple Support. https://support.apple.com/es-co/108051

Apple. (2024c, marzo 6). *Acerca de las actualizaciones de iOS 16—Soporte técnico de Apple (CO)*. Apple Support. https://support.apple.com/es-co/101566

Apple. (2024d, marzo 6). *Acerca de las actualizaciones de iOS 17—Soporte técnico de Apple (CO)*. Apple Support. https://support.apple.com/es-co/109043

Appssemble. (2023, octubre 13). Augmented & Virtual Reality (AR/VR) in mobile apps. *Appssenble - Augmented & Virtual Reality (AR/VR) in mobile apps*. https://appssemble.com/

Bautista-Sánchez, A. (2020). Desarrollo de aplicaciones móviles. *MoleQla: revista de Ciencias de la Universidad Pablo de Olavide*, *36*, 3. https://dialnet.unirioja.es/servlet/articulo?codigo=7220419

Bermúdez-Moreno, Y. M., & López-Hincapié, J. G. (2011). *Análisis comparativo entre sistemas operativos de dispositivos móviles Android, iPhone y BlackBerry* [Bachelor thesis, Universidad Tecnológica de Pereira]. https://hdl.handle.net/11059/2474

Bleger, M. (2022, marzo 25). *Android vs iOS: Ventajas y desventajas*. https://www.crehana.com. https://www.crehana.com/blog/transformacion-digital/android-vs-ios/

BLU Radio. (2023, abril 3). *Android vs. iOS: Ventajas y desventajas de ambos sistemas operativos* (world) [Text]. Blu Radio; Blu Radio. https://www.bluradio.com/tecnologia/android-vs-ios-ventajas-y-desventajas-de-ambos-sistemas-operativos-so35

C. G., T., & Devi, A. J. (2021). A Study and Overview of the Mobile App Development Industry. *International Journal of Applied Engineering and Management Letters*, *5*(1), 115-130. https://doi.org/10.47992/IJAEML.2581.7000.0097

Caballé, S. (2013, octubre 21). *Aprendizaje móvil (I): Un poco de historia* [Educativa]. Tecnología++ El blog de los Estudios de Informática, Multimedia y Telecomunicación de la UOC. https://blogs.uoc.edu/informatica/es/aprendizaje-movil-i-un-poco-de-historia/

Campos-Gavilánez, D. M. (2023). *Análisis comparativo entre sistemas operativos de dispositivos móviles ANDROID, IPHONE OS y MIUI* [bachelorThesis, Universidad Técnica de Babahoyo]. http://dspace.utb.edu.ec/handle/49000/13969

CertiDeal. (2023). *iOS: Historia, resumen de todas las versiones de iOS ya lanzadas, rumores y fecha de salida del iOS 17 - CERTIDEAL*. CERTIDEAL. https://www.certideal.es/content/208-ios

Chatterjee, S. (2023, agosto 11). What is Full-Stack Web Development? The Top 8 Frameworks Guide. *Emeritus Online Courses*. https://emeritus.org/blog/mobile-app-development/

De la Calle, R. (2024, febrero 8). *Pasa de Android a iPhone y viceversa. ¿Qué ganas y qué pierdes?* MovilZona. https://www.movilzona.es/tutoriales/software/cambiar-android-iphone-ventajas-inconvenientes/

Doonamis. (2022, marzo 29). Evolución de iOS y iOS 15. *Doonamis*. https://www.doonamis.com/evolucion-de-ios-y-ios-15/

Garrison, D. R. (1997). Self-Directed Learning: Toward a Comprehensive Model. *Adult Education Quarterly*, *48*(1), 18-33. https://doi.org/10.1177/074171369704800103

GSMA. (2024). *The Mobile Economy 2024* (9; p. 49). GSMA Intelligence. https://www.gsma.com/solutions-and-impact/connectivity-for-good/mobile-economy/

Ibertrónica. (2013, marzo 4). *Android, el sistema operativo de Google*. Ibertrónica - Blog de tecnología. https://ibertronica.es/blog/tutoriales/android-sistema-operativo/

Insights Success. (2019, noviembre 26). App Development Industry in India & its Dynamics- Insights Success. *Insights Success*. https://www.insightssuccess.in/app-development-industry-india-dynamics/

Janani. (2020, agosto 14). Different Types of Mobile Application Development- Every Business Owner Must Know! *Zuan Technologies*. https://www.zuantechnologies.com/blog/different-types-mobile-application-development-every-business-owner-must-know/

Jin, D. Y. (2017). *Smartland Korea: Mobile Communication, Culture, and Society*. University of Michigan Press. https://doi.org/10.3998/mpub.9332315

KeepCoding, R. (2023, diciembre 27). *Ventajas y desventajas de iOS | KeepCoding Bootcamps*. https://keepcoding.io/blog/ventajas-y-desventajas-de-ios/

Knowles, M. S. (1975). *Self-directed Learning: A Guide for Learners and Teachers* (Vol. 2). Cambridge Adult Education.

López, M. (2024, febrero 14). *Apple iOS: Todas las versiones hasta la fecha, cómo saber la que estás usando en tu iPhone y cómo actualizar el sistema operativo*. Applesfera. https://www.applesfera.com/nuevo/apple-ios-todas-versiones-fecha-como-saber-que-estas-usando-tu-iphone-como-actualizar-sistema-operativo

Masaad-Alsaid, M. A. M., Ahmed, T. M., Jan, S., Khan, F. Q., Mohammad, & Khattak, A. U. (2021). A Comparative Analysis of Mobile Application Development Approaches: Mobile Application Development Approaches. *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences*, *58*(1), 35-45. https://doi.org/10.53560/PPASA(58-1)717

Mir, S. B., & Llueca, G. F. (2020). Introduction to Programming Using Mobile Phones and MIT App Inventor. *Revista Iberoamericana de Tecnologias Del Aprendizaje*, *15*(3), 192-201. Scopus. https://doi.org/10.1109/RITA.2020.3008110

Montatixe-Granada, J. P. (2022). *Desarrollo de una aplicación móvil de promoción de las principales obras pictóricas de los artistas cotopaxenses empleando la metodología "mobile-d en la Casa de la Cultura Benjamín Carrión núcleo de Cotopaxi* [bachelorThesis, Universidad Técnica de Cotopaxi (UTC).]. http://localhost/handle/27000/9768

Montes-León, S. R., Ramos-Benalcazar, J. D., & Montes-León, H. (2014). Estudio preliminar de las empresas que más contribuyen al desarrollo de Android. *Revista Eletrônica de Sistemas de Informação*, *13*(2), Article 2. https://doi.org/10.21529/RESI.2014.1302007

Mora-Antoja, A. (2019, septiembre 19). *Todas las novedades del iOS 13*. Macworld. https://www.macworld.com/article/1453221/ios-13-todas-las-novedades-de-la-ultima-actualizacion-de-ios.html

Muyan-Özçelik, P. (2017). *A hands-on cross-platform mobile programming approach to teaching OOP concepts and design patterns*. 33-39. Scopus. https://doi.org/10.1109/SECM.2017.12

Narh, P. G., & Ranjan, R. (2022). *Integration Of Ai With Mobile Application Development*. 2605-2611. Scopus. https://doi.org/10.1109/ICACITE53722.2022.9823641

Patta, A. R., Funabiki, N., Lu, X., & Syaifudin, Y. W. (2023). *A Study of Grammar-concept Understanding Problem for Flutter Cross-platform Mobile Programming Learning*. 255-260. Scopus. https://doi.org/10.1109/ICVEE59738.2023.10348237

Perrin, A. (2021, junio 3). Mobile Technology and Home Broadband 2021 [Pew Research Center]. *Pew Research Center: Internet, Science & Tech*. https://www.pewresearch.org/internet/2021/06/03/mobile-technology-and-home-broadband-2021/

Phongtraychack, A., & Dolgaya, D. (2018). Evolution of Mobile Applications. *MATEC Web of Conferences*, *155*(01027), 7. https://doi.org/10.1051/matecconf/201815501027

Ramírez, I. (2022, enero 9). *Historia y evolución de Android: Cómo un sistema operativo para cámaras digitales acabó conquistando los móviles*. Xataka Android. https://www.xatakandroid.com/sistema-operativo/historia-evolucion-android-como-sistema-operativo-para-camaras-digitales-acabo-conquistando-moviles

Ramírez, I. (2023, julio 25). *Todas las versiones de Android de la historia*. Xataka Android. https://www.xatakandroid.com/sistema-operativo/todas-versiones-android-historia

Reclu IT. (2021, diciembre 16). *Beneficios del desarrollo con Android*. https://recluit.com/beneficios-del-desarrollo-con-android/

Ridge, B. V. (2023, diciembre 23). El Origen de Android: Una Mirada Detallada al Sistema Operativo Móvil. *MEDIUM Multimedia Agencia de Marketing Digital*. https://www.mediummultimedia.com/apps/cual-es-el-origen-de-android/

Rojas-Lizarazo, K. M., Roa-Castañeda, J. E., & Alarcón-Aldana, A. C. (2011). Desarrollo de aplicaciones móviles bajo la plataforma de Iphone. *Facultad de Ingeniería*, *20*(31), 77-91. https://www.redalyc.org/articulo.oa?id=413940770007

Rubio, R. (2023, marzo 15). *iOS: Características y versiones del sistema operativo de Apple*. ADSLZone. https://www.adslzone.net/reportajes/software/que-es-ios/

Rubio-Mazas, C. (2021, septiembre 19). *5 ventajas de iOS frente a Android*. Andro4all. https://www.lavanguardia.com/andro4all/apple/5-ventajas-de-ios-frente-a-android

Seidor. (2023, febrero 14). Todas las versiones del sistema operativo Android (13 incluida). *SEIDOR | Todas las versiones del sistema operativo Android (13 incluida)*. https://www.seidor.com/blog/versiones-android

Simplilearn. (2023, agosto 8). What Is Mobile Cloud Computing & Its Applications | Simplilearn. *Simplilearn*. https://www.simplilearn.com/what-is-mobile-cloud-computing-article

Singh, H. (2023, abril 28). Virtual Reality (VR) in Mobile App Development – A Comprehensive Guide. *Apptunix - VR Mobile App Development*. https://www.apptunix.com/blog/virtual-reality-vr-in-mobile-app-development-a-comprehensive-guide/

Syaifudin, Y. W., Funabiki, N., & Liem, I. (2021). *Comparisons of Student's Self-Learning Performances Using Java and Kotlin Languages in Android Programming Learning Assistance System*. 93-97. Scopus. https://doi.org/10.1109/OT4ME53559.2021.9638952

Syaifudin, Y. W., Hatjrianto, A. S., Funabiki, N., Liliana, D. Y., Kaswar, A. B., & Nurhasan, U. (2022). *An Implementation of Automatic Dart Code Verification for Mobile Application Programming Learning Assistance System Using Flutter*. 322-326. Scopus. https://doi.org/10.1109/IEIT56384.2022.9967902

Torres, A. (2020, abril 15). *Sistema Operativo Android: Ventajas y desventajas (2020)*. Portal Psicología y Mente. https://psicologiaymente.com/miscelanea/sistema-operativo-android-ventajas-desventajas

Totla, J. (2021, febrero 12). How Augmented Reality Is Transforming Mobile App Development? *SynergyTop*. https://synergytop.com/blog/how-augmented-reality-is-transforming-mobile-app-development/

Universidad del Pacífico. (2020). iOS – Edutic. *iOS*. https://edutic.up.edu.pe/catalogo-software/ios/

Universidad Politécnica de Valencia. (2021). *Las versiones de Android y niveles de API*. Universidad Politécnica de Valencia - UPV. http://www.androidcurso.com/index.php/recursos/31-unidad-1-vision-general-y-entorno-de-desarrollo/146-las-versiones-de-android-y-niveles-de-api

Universidad Tecnológica del Perú. (2024, febrero 23). *Android vs iOS ¿Cuál es el mejor? Ventajas y Dsventajas—Universidad Tecnológica del Perú*. https://www.utp.edu.pe/blog/android-vs-ios-ventajas-desventajas

Vieira-Jófili, F. (2023). *8 características de iOS para aprovechar tu iPhone*. Platzi. https://platzi.com/blog/caracteristicas-ios/

Wabimovil. (2023a, noviembre 20). El sistema operativo ANDROID: Ventajas e inconvenientes. *Wabimovil*. https://wabimovil.es/el-sistema-operativo-android-ventajas-e-inconvenientes/

Wabimovil. (2023b, diciembre 13). El sistema operativo IOS: Ventajas e inconvenientes. *Wabimovil*. https://wabimovil.es/el-sistema-operativo-ios-ventajas-e-inconvenientes/

Zohud, T., & Zein, S. (2021). Cross-Platform Mobile App Development in Industry: A Multiple Case-Study. *International Journal of Computing*, 46-54. https://doi.org/10.47839/ijc.20.1.2091

CAPÍTULO 2.

ARQUITECTURAS Y ENTORNO DE DESARROLLO

Johan Sebastián Lorza Pulido
Heriberto Fernando Vargas Losada y Edwin Eduardo Millán Rojas

Capítulo 2.
Arquitecturas y entorno de desarrollo

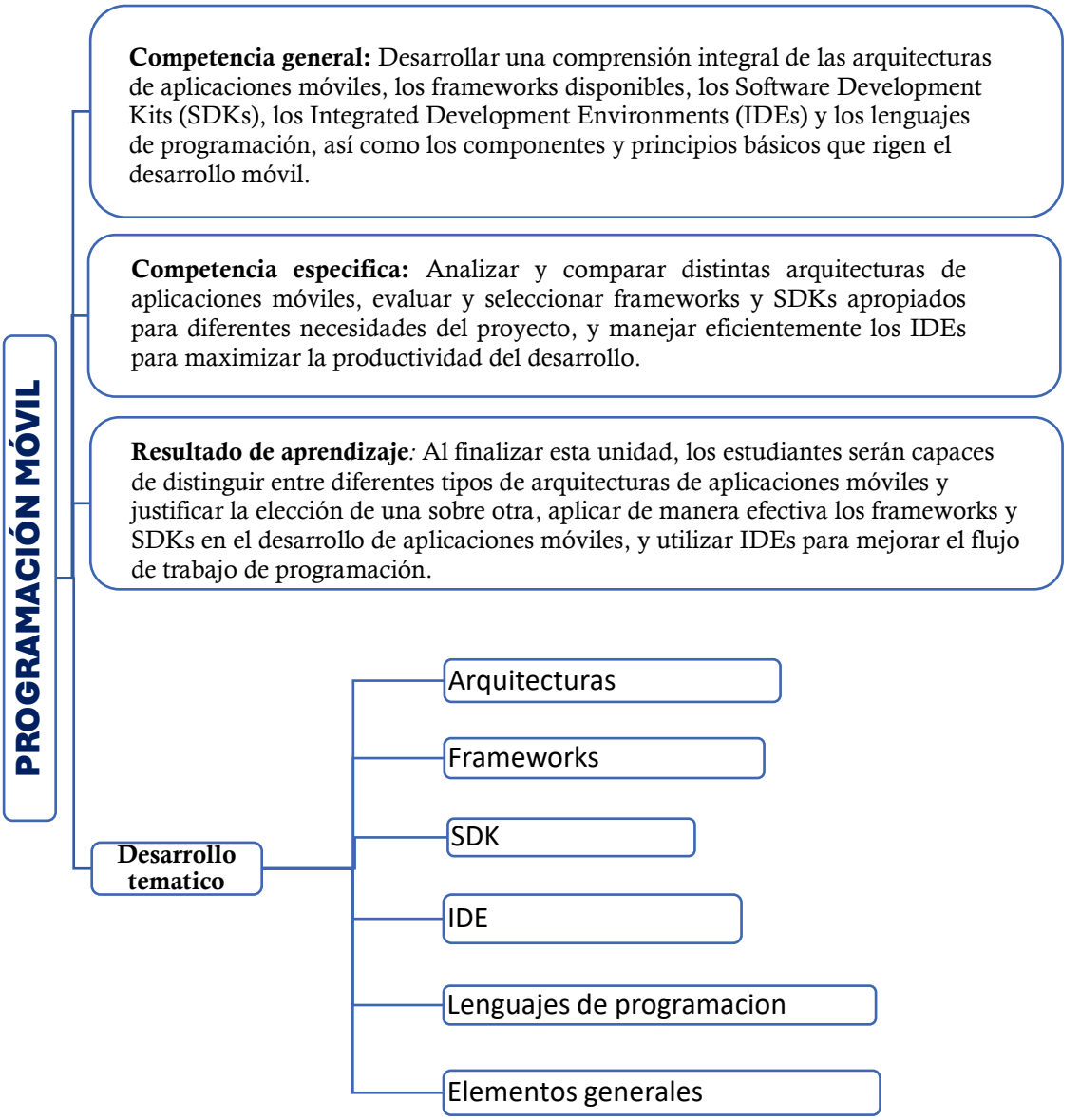
Autores
Johan Sebastián Lorza Pulido
Estudiante del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: j.lorza@udla.edu.co

Heriberto Fernando Vargas Losada
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: heri.vargas@udla.edu.co

Edwin Eduardo Millán Rojas
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: e.millan@udla.edu.co.

Como Referenciar el Capítulo de libro.
Lorza Pulido, J. S., Vargas Losada, H. F. & Millán Rojas, E. E. (2026). Capítulo 2. Arquitecturas y Entorno de Desarrollo. En *Programación Móvil: un enfoque desde la Ingeniería de Sistemas* (págs. 53-88). Florencia: Universidad de la Amazonia.

2.1. Esquema del Capítulo 2: Arquitecturas y Entorno de Desarrollo



2.2. COMPETENCIAS Y RESULTADOS DE APRENDIZAJE

La Unidad 2 abarca la temática de las arquitecturas de aplicaciones móviles y el entorno de desarrollo, constituyendo una exploración para cualquier aspirante a desarrollador de aplicaciones móviles. En la competencia general de esta unidad, tiene la finalidad de comprender los fundamentos de la programación móvil. Este recorrido abarcará las tecnologías asociadas hasta el manejo de las herramientas de desarrollo más actualizadas y efectivas que se ofrecen en el mercado. Nos adentraremos en las arquitecturas de software que sirven como la columna vertebral de las aplicaciones móviles, discerniendo cómo estructuran y dirigen el flujo de operaciones y datos. Con un enfoque práctico, examinaremos los frameworks más prominentes que permiten a los desarrolladores construir aplicaciones móviles con mayor rapidez y eficiencia, así como los SDKs que proporcionan los bloques de construcción esenciales para interactuar con los sistemas operativos móviles.

La competencia específica se centra en dotar a los estudiantes de la capacidad de analizar críticamente las ventajas y limitaciones de las diversas arquitecturas, frameworks y SDKs. Con un enfoque en la aplicación práctica, los estudiantes aprenderán a seleccionar el entorno de desarrollo más adecuado, incluyendo el IDE óptimo y el lenguaje de programación que mejor se adapte a las necesidades de sus proyectos. A través de esta lente, seremos testigos de cómo la elección de una arquitectura y las herramientas de desarrollo pueden impactar directamente en la funcionalidad, mantenibilidad y escalabilidad de las aplicaciones móviles.

Al final de la Unidad 2, se espera que los estudiantes sean capaces de seleccionar con criterio las herramientas y tecnologías adecuadas para el desarrollo de aplicaciones móviles. El resultado de aprendizaje es que, al tener un dominio de estos componentes, los estudiantes podrán no solo explicar los elementos que conforman el desarrollo de aplicaciones móviles, sino también aplicarlos eficazmente, creando aplicaciones que no solo funcionen bien, sino que también

ofrezcan una experiencia de usuario óptima y se alineen con las tendencias actuales y futuras del mercado de aplicaciones móviles.

En la siguiente sección, presentaremos los niveles de desempeño asociados al resultado de aprendizaje de nuestra Unidad 2. Estos niveles proporcionan un marco detallado que define qué se espera de los estudiantes al finalizar la unidad, variando desde un entendimiento básico hasta un dominio avanzado de las competencias.

Tabla 1
Niveles de desempeño del resultado de aprendizaje.

Avanzado (4.6 – 5)	El estudiante muestra un dominio excepcional de las arquitecturas de aplicaciones móviles, los frameworks, SDKs, IDEs y lenguajes de programación relevantes. Puede evaluar críticamente y elegir la herramienta más adecuada para cualquier escenario de desarrollo móvil con una comprensión profunda de las implicaciones de estas elecciones en la usabilidad, rendimiento y mantenimiento de las aplicaciones. Además, puede innovar y aplicar conocimientos de manera creativa en proyectos que establecen nuevos estándares de calidad en el desarrollo móvil.
Intermedio (4 – 4.5)	El estudiante tiene una buena comprensión de los componentes clave del desarrollo móvil y puede aplicar este conocimiento con competencia. Demuestra la capacidad de seleccionar y justificar arquitecturas y herramientas de desarrollo adecuadas para proyectos estándar y puede resolver problemas de desarrollo móvil con una perspectiva informada y métodos establecidos.
Básico (3 – 3.9)	El estudiante comprende los conceptos fundamentales de las arquitecturas y el entorno de desarrollo móvil y puede manejar tareas de desarrollo estándar con alguna orientación. Puede realizar selecciones de arquitecturas y herramientas basadas en criterios dados, pero todavía está desarrollando la capacidad de evaluar estas decisiones de forma crítica y puede requerir apoyo adicional para enfrentar desafíos más complejos.
Bajo (0 – 2.9)	El estudiante muestra una comprensión insuficiente de las arquitecturas y entornos de desarrollo en programación móvil y no puede aplicar eficazmente este conocimiento en la práctica. Tiene dificultades significativas para identificar las herramientas adecuadas y carece de la capacidad para justificar sus elecciones. Requiere una revisión fundamental de los conceptos y técnicas de desarrollo móvil y necesita una guía considerable para alcanzar los resultados de aprendizaje esperados.

Nota. Los niveles de desempeño se basan en una escala de 5 puntos. Fuente: elaboración propia.

Cada nivel refleja no solo la profundidad de comprensión del material del curso, sino también la habilidad práctica para aplicar dicho conocimiento en el diseño y desarrollo de aplicaciones móviles. Con estos criterios, esperamos fomentar un ambiente de aprendizaje que no solo aspire a alcanzar metas académicas, sino

que también prepare a los estudiantes para enfrentar retos reales del mundo profesional.

2.3. METODOLOGÍA DE ENSEÑANZA

El objetivo de esta metodología es proporcionar a los estudiantes una formación práctica y aplicada en el uso y comprensión de arquitecturas de software, frameworks, SDKs, IDEs, lenguajes de programación y los elementos generales necesarios para el desarrollo de aplicaciones. Esta metodología se enfoca en la aplicación práctica de los conocimientos mediante el uso de herramientas y técnicas que los estudiantes encontrarán en situaciones reales de desarrollo.

Para abordar estos temas de manera efectiva, se utilizarán diversos materiales didácticos que faciliten el aprendizaje práctico. Se incluirán guías técnicas y manuales detallados para la instalación y configuración de frameworks, SDKs e IDEs, así como ejemplos de código y proyectos base que los estudiantes podrán modificar y ampliar. Los tutoriales en video guiarán a los estudiantes a través de la construcción de proyectos prácticos desde cero, utilizando los lenguajes de programación y herramientas específicas discutidas en clase. Además, se proporcionarán plantillas de proyectos y ejemplos de código en GitHub para que los estudiantes puedan explorar y experimentar con diferentes arquitecturas y frameworks en un entorno controlado.

Las técnicas de aprendizaje se centrarán en la práctica y la experimentación. Se utilizará el aprendizaje basado en proyectos, donde los estudiantes desarrollarán aplicaciones completas o módulos específicos que implementen las arquitecturas, frameworks y lenguajes de programación estudiados. Los laboratorios prácticos permitirán a los estudiantes trabajar con SDKs e IDEs para desarrollar aplicaciones móviles, web o de escritorio, aplicando las mejores prácticas de desarrollo y depuración. El aprendizaje autodirigido también será clave, incentivando a los estudiantes a buscar y utilizar documentación oficial, comunidades en línea y foros de desarrolladores para resolver problemas y ampliar sus conocimientos más allá de

lo impartido en las clases. Además, se promoverá la programación en parejas, para que los estudiantes puedan colaborar y aprender mutuamente, compartiendo sus habilidades y conocimientos en tiempo real.

La evaluación del aprendizaje se realizará a través de proyectos prácticos y entregables que reflejen la aplicación de los conceptos enseñados. Los estudiantes deberán completar desafíos semanales, como la implementación de una función específica utilizando un framework o SDK particular, o la configuración de un entorno de desarrollo en un IDE determinado. Las entregas serán evaluadas por su funcionalidad, calidad del código, adherencia a las mejores prácticas de programación y uso efectivo de las herramientas enseñadas. Se realizarán presentaciones individuales o grupales donde los estudiantes expondrán los proyectos desarrollados, explicando las decisiones técnicas tomadas, las dificultades encontradas y las soluciones aplicadas. Además, se incluirán pruebas prácticas cortas, como mini-hackathons, donde los estudiantes deberán aplicar rápidamente sus conocimientos para resolver problemas específicos en un tiempo limitado, simulando situaciones reales de desarrollo.

El contenido se desarrollará en un lapso de 4 semanas, con sesiones semanales enfocadas en la integración de los temas:

Semana 1: Introducción a Arquitecturas, Frameworks y SDKs. En esta semana, se abordarán los fundamentos de las arquitecturas de software y se instalarán y configurarán frameworks y SDKs clave. Se realizará un proyecto inicial que combine estos elementos.

Semana 2: IDEs y Lenguajes de Programación. Se explorarán los principales IDEs utilizados en el desarrollo de software, junto con la configuración de entornos para varios lenguajes de programación. Los estudiantes comenzarán un proyecto práctico utilizando estas herramientas.

Semana 3: Desarrollo Integrado de Proyectos. Los estudiantes trabajarán en un proyecto más complejo que integre arquitecturas, frameworks, SDKs y

lenguajes de programación. Se enfocarán en la implementación de funcionalidades clave y la optimización del proyecto.

Semana 4: Pruebas, Debugging y Presentaciones Finales. Los estudiantes realizarán pruebas, depuración y mejoras finales en sus proyectos. La semana culminará con presentaciones de los proyectos, donde se expondrán los logros y aprendizajes, y se recibirá retroalimentación. Esta metodología combina un enfoque práctico con materiales didácticos variados y técnicas de aprendizaje activas para asegurar que los estudiantes no solo entiendan los conceptos, sino que también sean capaces de aplicarlos en contextos reales. A través de evaluaciones continuas y proyectos prácticos, los estudiantes desarrollarán las habilidades necesarias para desenvolverse en el mundo del desarrollo de software con confianza y competencia.

2.4. DESARROLLO TEMÁTICO

En la Unidad 2 de este libro, nos adentraremos en los elementos constitutivos de la programación móvil, una competencia indispensable en el panorama tecnológico contemporáneo. Esta unidad ha sido diseñada para proporcionar a los estudiantes una base sólida y conocimientos avanzados en arquitecturas de aplicaciones, frameworks de desarrollo, SDKs, IDEs y lenguajes de programación específicos para el ámbito móvil.

2.4.1. Arquitecturas

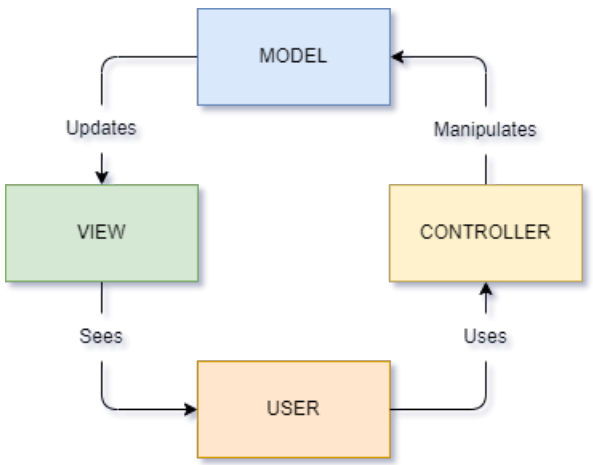
Cuando hablamos de arquitecturas en programación móvil, nos referimos al diseño estructural subyacente que define el modo en que las aplicaciones móviles se construyen y operan. Una arquitectura de aplicación móvil no es más que un plan que describe la disposición de cada elemento que compone la aplicación y cómo estos interactúan entre sí. Es esencial para establecer patrones claros de diseño y desarrollo, permitiendo la escalabilidad, el mantenimiento y la eficiencia en el proceso de construcción de la aplicación. Las arquitecturas guían el proceso de desarrollo desde la conceptualización hasta la implementación y el mantenimiento

posterior. En el desarrollo móvil, algunas de las arquitecturas más comunes incluyen:

2.4.1.1. Model-View-Controller (MVC)

Se divide en tres componentes principales: el Modelo que es responsable de la lógica de negocio y la gestión de datos, "la Vista que se ocupa de la interfaz de usuario, y el Controlador que actúa como un intermediario entre el Modelo y la Vista. Esta separación de responsabilidades facilita la prueba y mantenimiento de la aplicación" (Leff & Rayfield, 2001, p. 10).

Figura 2.1
Imagen del ciclo del modelo MVC.



Nota. Adaptado de MVC: Model, View, Controller [Diagrama], por Codecademy Team, 2024. Codecademy (https://www.codecademy.com/article/mvc)

En el diagrama, se pueden observar las siguientes partes y su interacción: En el Modelo, que se encuentra en la parte superior del diagrama, reside la lógica de negocio y los datos de la aplicación. Su responsabilidad principal es manejar las operaciones que involucran datos, procesar las reglas de negocio y responder a las solicitudes de información o cambios por parte del controlador. Cuando hay cambios en los datos, el modelo notifica a la vista para que pueda actualizar la presentación al usuario. La Vista, mostrada en el lado izquierdo del diagrama, es la

representación visual de los datos, es decir, lo que el usuario ve y con lo que interactúa. No contiene lógica de negocio y se limita a mostrar la información proporcionada por el modelo. Su actualización se produce cuando el modelo indica que ha habido un cambio en los datos que requiere una nueva presentación en la pantalla.

El Controlador, representado en el lado derecho, sirve como un intermediario entre la vista y el modelo. Responde a la entrada del usuario, procesada a través de acciones como clics o entradas de teclado, y manipula el modelo como resultado. Además, el controlador decide qué vista presentar, basándose en la interacción del usuario y los cambios en el modelo. En la parte inferior del diagrama, encontramos al Usuario, el actor final que interactúa con la aplicación. El usuario realiza acciones que el controlador interpreta, comienza el ciclo de procesamiento de datos en el modelo y, finalmente, ve los resultados reflejados a través de la vista. La disposición del diagrama destaca la naturaleza cíclica y reactiva del patrón MVC, con cada componente desempeñando un papel específico en la respuesta a la interacción del usuario y en la presentación de los datos. La dirección de las flechas indica el flujo de la información y las actualizaciones, asegurando que los cambios en el modelo se reflejen en la vista que el usuario ve, todo bajo el control del controlador que maneja la lógica de qué y cómo los datos deben ser presentados.

Model

El modelo se compone de diversas clases que simulan la información del entorno real necesaria para el procesamiento del sistema. Funciona como la entidad que encarna la lógica empresarial dentro de una aplicación. Teóricamente, el modelo debería ser ajeno a las vistas y al controlador, aunque en la práctica esto puede ser complicado. Esto se debe a que es necesario que existan interfaces para la comunicación entre módulos. Por esta razón, SmallTalk propone que el modelo se divida en dos submódulos: el modelo de dominio y el modelo de aplicación (Bascón-Pantoja, 2004).

View

Presenta el modelo (datos) en un formato específico. No realiza ningún procesamiento de los datos, su único trabajo es mostrar la información al usuario en un contexto particular. Una aplicación puede tener múltiples vistas que presentan diferentes elementos del mismo modelo. Las vistas se utilizan para administrar y preparar la interfaz de usuario (UI) de nuestra aplicación. Al utilizar esa interfaz de usuario, el usuario interactúa con nuestra aplicación (Ambani, 2020).

Controller

Actúa como un intermediario entre el modelo y la vista. Responde a la entrada del usuario, interactuando con los datos del modelo y finalmente seleccionando una vista para presentar. En esencia, toma decisiones, maneja el control de qué vista mostrar para una solicitud particular y con qué datos responder (Ambani, 2020).

Ventajas del MVC

El modelo Model-View-Controller (MVC) es ampliamente reconocido por sus múltiples ventajas en el desarrollo de software, especialmente en aplicaciones web y móviles donde la claridad de la estructura y la separación de responsabilidades son cruciales. Aquí se detallan algunas de las principales ventajas de usar la arquitectura MVC Gavilánez Álvarez et al., 2022:

- Una de las mayores ventajas de MVC es que divide la aplicación en tres componentes principales (Modelo, Vista y Controlador), cada uno con responsabilidades bien definidas. Esto no solo facilita el desarrollo y mantenimiento al permitir que los desarrolladores trabajen en componentes individuales sin afectar a otros, sino que también mejora la organización del código y hace que el sistema sea más manejable.
- Dado que MVC separa la lógica de negocio del usuario y la interfaz de usuario, permite que múltiples desarrolladores trabajen simultáneamente en el mismo proyecto con un mínimo de interferencia. Por ejemplo, mientras

un desarrollador trabaja en la vista, otro puede trabajar en la lógica de negocio en el modelo, lo cual es especialmente útil en equipos grandes

- MVC hace que sea fácil cambiar la interfaz de usuario de la aplicación sin tener que tocar la lógica de negocio, ya que las vistas son separadas y no dependen del modelo. Esto es especialmente valioso en el entorno actual de desarrollo rápido donde los requisitos del usuario y las interfaces gráficas cambian con frecuencia.
- El aislamiento de la lógica de negocio, las reglas de negocio y las tareas de interfaz de usuario en componentes separados significa que los modelos pueden reutilizarse sin ninguna modificación, porque no están atados a las vistas específicas.
- Cada componente puede ser probado de manera independiente (unit testing), lo cual es una gran ventaja. Además, el soporte para el testing es mejorado debido a que el MVC, al tener una clara separación de componentes, permite escribir diferentes pruebas específicas para cada componente sin necesidad de escribir pruebas para el resto del sistema.
- MVC permite que una aplicación tenga múltiples vistas de un modelo. Dado que la lógica de interfaz de usuario está separada del resto, es más fácil tener varias representaciones de la misma información fundamental, lo cual es útil para adaptarse a diferentes dispositivos o interfaces de usuario.

Desventajas del MVC

Aunque el modelo Model-View-Controller (MVC) ofrece numerosos beneficios que lo hacen popular en el desarrollo de software, también presenta ciertas desventajas que pueden complicar su implementación y uso, especialmente en aplicaciones más grandes o complejas. A continuación, se exploran algunas de estas limitaciones (Enríquez et al., 2023; Rosero-Guerrero, 2020):

- A medida que una aplicación crece en tamaño y complejidad, el modelo MVC puede volverse más difícil de manejar. La interdependencia entre los

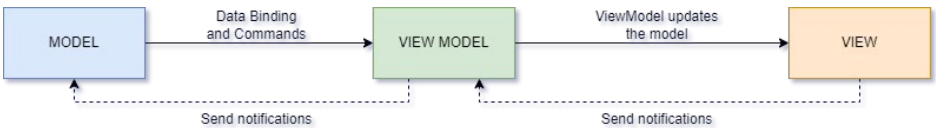
componentes, especialmente entre las vistas y los controladores, puede aumentar, lo que lleva a un acoplamiento más fuerte y a una menor mantenibilidad. Esto puede hacer que la aplicación sea más difícil de entender y de modificar.

- En MVC, los controladores pueden volverse sobrecargados con la lógica de enrutamiento, lo que los hace masivos y difíciles de mantener. Esto ocurre porque el controlador maneja todas las entradas de usuario, lo que puede conducir a una gran cantidad de lógica de control siendo incluida en el controlador en lugar de estar en el modelo o en otra parte.
- MVC puede introducir una sobrecarga en términos de rendimiento debido a la separación de datos y la forma en que se gestionan las actualizaciones de la vista. Por ejemplo, si los datos cambian con frecuencia, mantener la vista actualizada puede resultar costoso y afectar el rendimiento de la aplicación, especialmente si las vistas necesitan procesar grandes cantidades de datos en tiempo real.
- Para los nuevos desarrolladores, entender y aplicar correctamente el patrón MVC puede ser desafiante. Incorrectamente implementado, puede llevar a un pobre diseño de la aplicación, con problemas como demasiada lógica de negocio siendo colocada en los controladores o en las vistas, lo que contradice el propósito del patrón.

2.4.1.2. Model-View-ViewModel (MVVM)

La arquitectura MVVM, que se ha popularizado con frameworks modernos como Angular y Vue.js, Similar al MVC, pero con el ViewModel que proporciona enlaces de datos entre el Modelo y la Vista, permitiendo una mayor separación entre la lógica de negocio y la interfaz de usuario (Anderson, 2012). Esto es particularmente útil en aplicaciones que requieren una gran cantidad de interacciones dinámicas en la interfaz.

Figura 2.2
Imagen del modelo ciclo del MVVM.



Nota. Adaptado de El patrón Model-View-ViewModel [Diagrama], por Britch et al., 2023. Microsoft Learn.

El patrón Model-View-ViewModel (MVVM) se segmenta en tres partes esenciales: el Modelo, que encapsula la lógica de negocio y los datos; la Vista, que presenta la información y recibe la interacción del usuario; y el ViewModel, que vincula el Modelo y la Vista mediante enlaces de datos y comandos, facilitando así la separación de la lógica de presentación de la lógica empresarial. Este patrón es crucial para el desarrollo de aplicaciones con interfaces ricas y dinámicas, donde la experiencia del usuario y la capacidad de mantenimiento son prioritarias. De acuerdo con Gaudioso (2010), el patrón Model-View-ViewModel (MVVM) está compuesto por tres componentes clave que colaboran para crear aplicaciones robustas y fácilmente mantenibles:

Modelo (Model)

Constituye la base de datos y la lógica de negocio de la aplicación. Es la representación directa de los datos y las operaciones que se pueden realizar con ellos, encargado de la gestión y manipulación de estos datos, así como de las reglas de negocio, validaciones y operaciones de negocio.

Vista (View)

Corresponde a la interfaz de usuario, que se encarga de presentar la información al usuario y recoger sus interacciones. Define la estructura, el diseño y la apariencia de lo que los usuarios ven en pantalla. La vista está activamente

vinculada al ViewModel y reacciona a sus cambios para reflejar la información actualizada.

ViewModel

Es el intermediario entre el Modelo y la Vista, que expone públicamente los comandos y los datos que la Vista necesita. Contiene toda la lógica de presentación y los estados de la vista, y reacciona a las interacciones del usuario dentro de la vista, realizando cambios en el Modelo cuando es necesario. Cada uno de estos componentes juega un papel fundamental en la organización del código y en la eficiencia del desarrollo y mantenimiento de las aplicaciones. El MVVM es especialmente poderoso en entornos que ofrecen capacidades de enlace de datos avanzadas, lo que permite una sincronización eficiente entre la Vista y el ViewModel.

Ventajas del MVVM

El patrón Model-View-ViewModel (MVVM) proporciona varias ventajas significativas en el desarrollo de aplicaciones, particularmente aquellas con interfaces de usuario complejas y ricas en datos. De acuerdo con García, (2023) algunas ventajas son:

- MVVM fomenta una distribución bien definida de las responsabilidades en la aplicación, lo que facilita el mantenimiento y mejora la organización del código.
- Al separar la lógica de la interfaz de usuario de la lógica de negocios, el MVVM permite una mejor cobertura de prueba unitaria, especialmente para la lógica de presentación en el ViewModel.
- MVVM aprovecha el enlace de datos para minimizar el código de pegamento necesario para actualizar la vista cuando cambian los datos, y viceversa, lo que reduce errores y trabajo innecesario.
- Permite a los diseñadores y desarrolladores trabajar más independientemente y de forma paralela; los diseñadores pueden centrarse

en la capa de vista, mientras que los desarrolladores pueden trabajar en el ViewModel y el Modelo.

- Los ViewModels pueden reutilizarse en diferentes vistas si la aplicación requiere presentar los mismos tipos de datos en diferentes formas.

Estas ventajas hacen del MVVM un patrón atractivo para el desarrollo de aplicaciones modernas, especialmente aquellas que utilizan frameworks como WPF, Xamarin o Angular que tienen capacidades de enlace de datos integradas.

Desventajas del MVVM

De acuerdo con García, (2023) el patrón Model-View-ViewModel (MVVM) trae consigo desventajas que pueden afectar a ciertos proyectos de desarrollo de software, tales como:

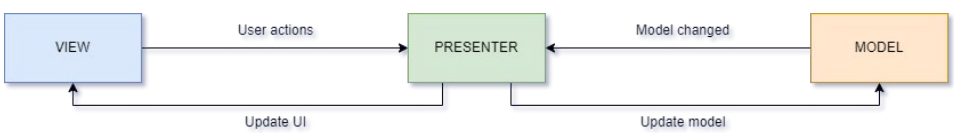
- Puede ser complejo de implementar, particularmente para equipos no familiarizados con el patrón o el enlace de datos. Esto puede resultar en una curva de aprendizaje pronunciada y una configuración inicial que consume tiempo.
- La abstracción entre la vista y el modelo puede conducir a escenarios en los que el código se vuelve demasiado indirecto, dificultando el seguimiento del flujo de la aplicación y la depuración.
- El enlace de datos automático puede afectar el rendimiento en aplicaciones grandes o complejas, ya que la gestión de las actualizaciones de estado y las dependencias puede volverse intensiva en recursos.
- Mantener y gestionar el estado dentro de los ViewModels puede ser desafiante, sobre todo cuando se trata de estados complejos o dependencias entre diferentes ViewModels.
- MVVM a menudo depende de capacidades específicas del framework utilizado, lo que puede limitar la portabilidad del código y la elección de tecnologías.

Las desventajas del MVVM subrayan la importancia de considerar los requisitos y capacidades del equipo antes de elegir este patrón de diseño para un proyecto.

2.4.1.3. Model-View-Presenter (MVP)

Esta arquitectura, es una evolución del MVC que proporciona una mejor separación entre la lógica de presentación y la lógica de negocio. El Presenter asume la función de intermediario, mejorando la testabilidad y el modularidad del código, especialmente en escenarios donde la lógica de interacción es intensa. A diferencia del MVC, tanto la Vista como el Modelo están completamente separados y toda la lógica de interacción se maneja a través del Presenter (Potel, 1996).

Figura 2.3
Imagen del modelo ciclo del MVP.



Nota. Adaptado de Model View Presenter [Diagrama], Fuente: Vyas, 2018. Medium.

El ciclo comienza con el usuario interactuando con la interfaz de usuario. La vista recoge esta interacción y la comunica al presentador. El presentador, luego, interactúa con el modelo según sea necesario y, al recibir la información o el resultado de la operación, actualiza la vista correspondientemente. Este ciclo asegura una clara separación entre la presentación, la lógica de aplicación y la lógica de negocios, permitiendo así un desarrollo y mantenimiento más sencillo y organizado. El patrón Model-View-Presenter (MVP) se estructura en tres componentes fundamentales que separan la lógica de la interfaz de usuario de la lógica de negocio, facilitando el desarrollo y el testing. A continuación, de acuerdo con Zhang & Luo (2010), se detalla cada componente:

Modelo (Model)

Representa la capa de datos y la lógica de negocio. Se encarga de consultar y manipular la información que la aplicación utiliza, actuando como la fuente de verdad para todos los datos que se presentan en la interfaz de usuario.

Vista (View)

Corresponde a la capa de presentación, donde los datos son mostrados al usuario. En MVP, la vista es generalmente pasiva, recibiendo instrucciones del presentador sobre qué mostrar, por lo que no contiene ninguna lógica de negocio y se centra exclusivamente en la representación visual de los datos.

Presentador (Presenter)

Actúa como un intermediario entre la vista y el modelo. Se encarga de toda la lógica de presentación. Escucha las acciones del usuario transmitidas por la vista y reacciona a ellas actualizando tanto la vista como el modelo. A diferencia de MVC, el presentador en MVP también toma decisiones sobre la lógica de navegación y la coordinación de las vistas.

Ventajas de MVP

El patrón Model-View-Presenter (MVP) tiene varias ventajas en el desarrollo de aplicaciones, entre las que de acuerdo con Castillo-Gomez (2022), destacan:

- Al desacoplar la lógica de la vista y el modelo, MVP facilita la escritura de pruebas unitarias y de integración, especialmente para la lógica de presentación en el presentador.
- Cada componente del MVP tiene responsabilidades bien definidas, lo cual simplifica la gestión del código y hace que las aplicaciones sean más fáciles de mantener y evolucionar.
- El desacoplamiento de la vista y el modelo permite que la lógica de negocio y la lógica de presentación sean reutilizadas en diferentes partes de la aplicación o incluso en diferentes aplicaciones.

- Al mantener una separación más estricta entre la interfaz de usuario y la lógica de negocios, MVP permite que los desarrolladores y diseñadores trabajen de forma más independiente y paralela

Desventajas de MVP

El patrón Model-View-Presenter (MVP), aunque beneficioso, de acuerdo con García, (2023a) conlleva algunas desventajas que deben ser consideradas:

- La implementación de MVP puede añadir complejidad adicional al código, ya que introduce más capas y abstracciones que pueden no ser necesarias para aplicaciones más simples.
- Para equipos no familiarizados con el patrón, puede haber una curva de aprendizaje significativa. Entender cómo dividir responsabilidades entre el presentador, la vista y el modelo puede llevar tiempo.
- MVP puede resultar en un desarrollo más lento debido a la necesidad de mantener una coordinación constante entre los componentes separados.
- Algunos desarrolladores encuentran que MVP puede ser rígido en términos de diseño, limitando la flexibilidad en cómo se pueden organizar los componentes de la aplicación.
- El presentador puede volverse muy grande y difícil de gestionar si no se tiene cuidado con la separación de la lógica y las responsabilidades, cayendo en el problema de "God Object".

Cada arquitectura tiene su propia estructura y método de funcionamiento, pero todas tienen el objetivo común de organizar el código de una manera que facilite la gestión y evolución de las aplicaciones móviles.

2.4.2. Frameworks

En el ámbito de la programación móvil, los frameworks son conjuntos de herramientas y bibliotecas de software que proporcionan una estructura subyacente para desarrollar aplicaciones móviles (Forcada-Sanz, 2020). Facilitan procesos

comunes y ofrecen una serie de componentes reutilizables que aceleran el desarrollo y ayudan a mantener la consistencia en diferentes proyectos. Su uso permite a los desarrolladores centrarse en las características únicas de sus aplicaciones en lugar de tener que construir cada elemento desde cero (De la Cruz-Ceron, 2021).

Los frameworks móviles vienen en varias formas y se especializan en distintos aspectos del desarrollo de aplicaciones. Pueden ser específicos para un sistema operativo, como Swift para iOS, o multiplataforma, como React Native o Flutter, que permiten a los desarrolladores escribir un único código base que funciona en múltiples sistemas operativos. Generalmente, los frameworks siguen una arquitectura específica que dicta cómo se estructura el código y cómo los distintos componentes de una aplicación interactúan. Proporcionan librerías de código prescrito para funciones estándar, como manejo de datos, navegación entre vistas y comunicaciones de red, entre otros. Además, muchos frameworks ofrecen herramientas de desarrollo (como compiladores, depuradores y editores de código) que facilitan aún más el proceso de desarrollo.

Los frameworks simplifican el proceso de desarrollo al manejar muchos de los detalles complicados que vienen con la programación de aplicaciones móviles. Proporcionan una serie de funcionalidades listas para usar, como la gestión de la interfaz de usuario, el acceso a bases de datos, la comunicación de red y más, lo que permite a los desarrolladores centrarse en la lógica y características específicas de sus aplicaciones. Funcionan al proporcionar un conjunto de componentes y herramientas que se adhieren a un diseño de arquitectura específico, lo cual puede variar entre MVC, MVP, MVVM, o cualquier otro patrón de diseño estructural que el framework adopte. Esta estructura ayuda a organizar el código y hace que las aplicaciones sean más fáciles de mantener y escalar.

Frameworks para Android

Para el desarrollo de aplicaciones Android, existen varios frameworks que los desarrolladores pueden utilizar para facilitar el proceso de construcción y mejorar la calidad de sus aplicaciones. Aquí se presentan algunos de los frameworks más destacados:

Flutter

Flutter es un framework de UI (interfaz de usuario) de código abierto creado por Google para el desarrollo de aplicaciones, usa el lenguaje de programación Dart, el cual está optimizado para la construcción de interfaces de usuario fluidas y de alto rendimiento. Se utiliza para desarrollar aplicaciones multiplataforma, lo que significa que con un único código base, los desarrolladores pueden crear aplicaciones nativas compiladas no solo para dispositivos móviles Android e iOS, sino también para web y escritorio (Osuna-Lizárraga, 2020).

Ventajas de Flutter

Sus ventajas han atraído a una creciente comunidad de desarrolladores, y muchas empresas están adoptando Flutter para sus desarrollos móviles, web y de escritorio. De acuerdo con Quisaguano et al. (2022), algunas de las ventajas clave de Flutter son:

- Flutter permite a los desarrolladores escribir un único código base que funciona tanto en Android como en iOS, y también se está expandiendo para incluir soporte web y de escritorio. Esto puede resultar en un ahorro significativo de tiempo y recursos, ya que elimina la necesidad de crear y mantener múltiples bases de código para diferentes plataformas.
- A diferencia de otros frameworks que dependen de "puentes" para comunicarse con los componentes nativos del dispositivo, Flutter compila directamente a código de máquina nativo, lo que elimina cualquier necesidad de interpretación en el camino y permite un rendimiento óptimo.
- Flutter viene con un rico conjunto de widgets que siguen las directrices de Material Design de Google y Cupertino de Apple, permitiendo crear

aplicaciones visualmente atractivas y altamente interactivas. Además, permite una personalización sin precedentes, lo que es ideal para crear una marca distintiva a través de la UI.

Desventajas de Flutter

Aunque Flutter es una herramienta poderosa y flexible para el desarrollo de aplicaciones multiplataforma, también presenta ciertas desventajas que pueden afectar la decisión de utilizarlo en ciertos proyectos. De acuerdo con Zurita-Núñez (2020), algunas de las desventajas principales son:

- Las aplicaciones desarrolladas en Flutter tienden a tener tamaños de archivo más grandes en comparación con las nativas. Esto se debe principalmente a que Flutter necesita incluir la máquina virtual Dart y los componentes de Flutter Engine en la aplicación. Los tamaños de archivo grandes pueden ser un factor limitante, especialmente para usuarios en regiones con conexiones a internet limitadas o dispositivos con almacenamiento reducido.
- Aunque Dart, el lenguaje utilizado en Flutter, es relativamente fácil de aprender, los desarrolladores que no están familiarizados con él o con los conceptos de desarrollo de aplicaciones multiplataforma pueden enfrentarse a una curva de aprendizaje.

Ionic.

onic es un framework de desarrollo de aplicaciones móviles híbridas de código abierto, diseñado para permitir a los desarrolladores construir aplicaciones multiplataforma utilizando tecnologías web como HTML5, CSS y JavaScript. Este framework se centra en la experiencia del usuario y la interacción con la interfaz, ofreciendo una amplia gama de componentes de UI optimizados para móviles y gestos (Osuna-Lizárraga, 2020).

Ventajas de Ionic

Esta plataforma ofrece varias ventajas clave que la hacen atractiva para los desarrolladores y empresas que buscan construir aplicaciones móviles eficientes y multiplataforma. Según (Tixi-Moya, 2024; Forcada-Sanz, 2020), algunas ventajas que ofrece el framework son:

- Ionic permite a los desarrolladores escribir una sola vez y ejecutar en cualquier lugar, facilitando la creación de aplicaciones multiplataforma con un único código base. Esto reduce significativamente el tiempo y los recursos necesarios para desarrollar y mantener múltiples versiones de una aplicación.
- Ionic se integra bien con frameworks populares como Angular y React, proporcionando una robusta estructura de desarrollo y acceso a un amplio conjunto de funcionalidades adicionales y optimizaciones de rendimiento.
- Ionic disfruta de una comunidad de desarrolladores muy activa y de un amplio soporte, lo que facilita encontrar soluciones a problemas, compartir mejores prácticas y acceder a una variedad de recursos y tutoriales.

Desventajas de Ionic

Aunque Ionic ofrece numerosas ventajas para el desarrollo de aplicaciones móviles híbridas, también presenta ciertas desventajas que pueden influir en la decisión de utilizarlo en proyectos específicos. Según Tixi-Moya (2024), algunas de las principales desventajas de usar Ionic son:

- A pesar de las mejoras significativas, las aplicaciones desarrolladas con Ionic pueden no alcanzar el mismo nivel de rendimiento que las aplicaciones nativas, especialmente en dispositivos más antiguos o con recursos limitados. Esto se debe a que Ionic se basa en una "WebView" que puede resultar en tiempos de respuesta más lentos y menor fluidez en las animaciones y transiciones.

- Ionic utiliza Cordova o Capacitor para acceder a funcionalidades nativas del dispositivo. La calidad y el mantenimiento de estos plugins varían, y algunos pueden no estar actualizados o carecer de las características necesarias, lo que puede limitar la funcionalidad de la aplicación o su compatibilidad con las últimas versiones de sistemas operativos.
- Para los desarrolladores que no están familiarizados con tecnologías web como HTML, CSS y JavaScript, puede haber una curva de aprendizaje para empezar a usar Ionic eficazmente, aunque para desarrolladores web puede ser más fácil adaptarse.
- A medida que las aplicaciones desarrolladas con Ionic se vuelven más grandes y complejas, gestionar el estado y el rendimiento puede ser más desafiante en comparación con las soluciones nativas que están diseñadas para manejar grandes volúmenes de datos y lógica compleja más eficientemente.

2.4.3. SDKs

Los SDK (Software Development Kits) en programación móvil son esenciales para facilitar y acelerar el desarrollo de aplicaciones. Estos kits de desarrollo proporcionan un conjunto de herramientas, guías, bibliotecas de código y APIs necesarias para desarrollar software en plataformas específicas o múltiples plataformas.

Un SDK es un conjunto de herramientas de software que los desarrolladores pueden utilizar para crear aplicaciones para un sistema operativo específico o múltiples sistemas operativos. Facilitan el acceso a plataformas de dispositivos y funciones sin necesidad de desarrollar desde cero cada aspecto del software. Los SDKs a menudo incluyen bibliotecas, documentación, código de ejemplo, procesos y guías que ayudan en el desarrollo de aplicaciones móviles eficientes y efectivas (Amazon Web Services, 2023a).

Los SDKs operan como un conjunto comprensivo de herramientas que permiten a los desarrolladores escribir menos código, aprovechando las bibliotecas

y funciones preconstruidas para tareas comunes como la autenticación de usuarios, el manejo de redes, la manipulación de gráficos y el acceso a bases de datos. También suelen incluir emuladores que facilitan el testing y la depuración de aplicaciones en diferentes entornos de hardware y software sin necesidad de dispositivos físicos.

Android SDK

De acuerdo con Developer Android (2024), "Android SDK (Software Development Kit) es un conjunto integral de herramientas de desarrollo proporcionado por Google para facilitar la creación de aplicaciones para el sistema operativo Android". Este SDK incluye todo lo necesario para empezar a desarrollar aplicaciones, desde herramientas y bibliotecas hasta documentación y ejemplos de código. Componentes Principales del Android SDK:

- Android Studio: Es el entorno de desarrollo integrado (IDE) oficial para Android, proporcionando una interfaz robusta y eficiente para la codificación, depuración, y pruebas de aplicaciones.
- APIs del Framework Android: Proporciona bibliotecas que los desarrolladores pueden utilizar para interactuar con el hardware del dispositivo, como la cámara y el GPS, así como con componentes de software como ventanas emergentes y vistas.
- Android Debug Bridge (ADB): Es una herramienta de línea de comandos que permite la comunicación con un dispositivo conectado para la transferencia de datos, instalación de aplicaciones y ejecución de comandos de shell.
- Herramientas de Compilación: Incluye compiladores y herramientas para convertir el código fuente en aplicaciones Android ejecutables, utilizando Gradle como el sistema de compilación automatizado.
- Documentación y Ejemplos de Código: El SDK proporciona una documentación exhaustiva y ejemplos de código que son esenciales para

aprender a desarrollar aplicaciones Android y para comprender cómo utilizar las diversas características del SDK.

- Soporte para Kotlin y Java: Android SDK soporta Kotlin, que es el lenguaje preferido por Google para el desarrollo de Android, así como Java, que ha sido el lenguaje tradicional para el desarrollo de Android.

2.4.4. IDEs

Los Entornos de Desarrollo Integrado (IDEs) son herramientas fundamentales en el desarrollo de software, incluido el desarrollo móvil. Un IDE combina varias herramientas de desarrollo en un único software de aplicación, proporcionando a los desarrolladores un entorno coherente y eficiente para escribir, modificar, probar y depurar sus programas.

Un IDE es una plataforma de software que integra herramientas comunes de desarrollo de software en un solo producto. Su función principal es simplificar el proceso de desarrollo de software proporcionando componentes como un editor de código, herramientas de compilación y depuración en una única interfaz. Esto no solo mejora la eficiencia, sino que también ayuda a mantener la coherencia entre diferentes etapas del desarrollo (Amazon Web Services, 2023b).

Los IDEs funcionan al encapsular herramientas de programación comunes en un entorno unificado que soporta el desarrollo del ciclo de vida completo de una aplicación. Esto incluye desde la escritura del código hasta su prueba y depuración. Los IDEs pueden también integrarse con sistemas de control de versiones, lo que facilita la colaboración entre desarrolladores y el manejo de cambios en el código fuente.

Android Studio

Según Developer Android (2024b), Android Studio es el entorno de desarrollo integrado (IDE) oficial para la creación de aplicaciones Android, lanzado por Google en 2013. Basado en el potente editor de código y las herramientas para desarrolladores de IntelliJ IDEA. Es el sucesor del Eclipse Android Developer

Tools (ADT) como principal IDE para el desarrollo nativo de Android. Características Principales de Android Studio:

- Editor de Código Inteligente: Proporciona sugerencias de código, autocompletado y análisis en tiempo real que facilitan la escritura de código robusto y optimizado.
- Emulador Integrado: Viene con un emulador de alto rendimiento para probar aplicaciones en diferentes dispositivos Android y versiones del sistema operativo sin necesidad de dispositivos físicos.
- Diseñador de Interfaz de Usuario (UI): Incluye un diseñador de UI que permite arrastrar y soltar componentes para construir la interfaz gráfica, lo que simplifica el proceso de diseño.
- Gradle-Based Build Support: Utiliza Gradle para su sistema de compilación, lo que proporciona un método de construcción personalizable y potente.
- Compatibilidad con Kotlin: Android Studio soporta Kotlin, un lenguaje de programación moderno que Google ha adoptado como lenguaje oficial para el desarrollo de Android, junto a Java.
- Herramientas de Análisis y Perfilado: Ofrece un conjunto de herramientas para el análisis de rendimiento y el perfilado de aplicaciones, lo que ayuda a identificar y solucionar cuellos de botella en el rendimiento.
- Control de Versiones Integrado: Es compatible con sistemas de control de versiones como Git, facilitando el manejo de las revisiones y colaboraciones en el código fuente.

Android Studio está diseñado para aumentar la productividad de los desarrolladores, proporcionando herramientas que aceleran la codificación y reducen la posibilidad de errores. Su amplia gama de funcionalidades y la integración nativa con el ecosistema de Android lo convierten en una elección preferida para muchos desarrolladores de aplicaciones móviles.

2.4.5. Lenguajes de programación

En el desarrollo de aplicaciones móviles, la elección del lenguaje de programación es crucial, ya que define cómo se construirá la aplicación, qué herramientas se pueden usar, y a menudo, qué dispositivos podrán ejecutar la aplicación. Los lenguajes de programación pueden ser específicos para una plataforma o diseñados para ser utilizados en múltiples plataformas.

Java

Java ha sido uno de los lenguajes de programación más prominentes para el desarrollo móvil, especialmente para aplicaciones Android. Desde el lanzamiento de Android, Java fue el lenguaje oficial recomendado por Google para este propósito hasta la introducción de Kotlin como una alternativa preferida en 2017. A pesar de esto, Java sigue siendo ampliamente utilizado en el desarrollo de aplicaciones Android debido a su madurez, estabilidad y la vasta comunidad de desarrolladores que lo respaldan (López-Corrales, 2021).

Kotlin

Kotlin es un lenguaje de programación moderno y versátil, diseñado por JetBrains, que ha ganado popularidad como una alternativa eficiente a Java para el desarrollo de aplicaciones Android (Osuna-Lizárraga, 2020). En 2017, Google lo anunció como lenguaje oficial para el desarrollo de Android, lo que ha impulsado su adopción entre los desarrolladores de aplicaciones móviles.

2.4.6. Elementos generales

Los elementos generales para el desarrollo de aplicaciones móviles constituyen los componentes básicos y las prácticas fundamentales que son esenciales en la creación de cualquier aplicación móvil, independientemente del sistema operativo o la plataforma. Estos elementos abarcan desde la estructura del código y las interfaces de usuario hasta la gestión de datos y la seguridad de la aplicación.

Estructura del Código y Patrones de Diseño

La organización del código en una aplicación móvil es crucial para su mantenimiento y escalabilidad. Usar patrones de diseño como MVC (Model-View-Controller), MVP (Model-View-Presenter) o MVVM (Model-View-ViewModel) ayuda a separar la lógica de la interfaz de usuario de la lógica de negocio, facilitando la gestión del código y los futuros cambios.

Interfaces de Usuario (UI) y Experiencia de Usuario (UX)

El diseño de interfaces de usuario atractivas y funcionales es fundamental para el éxito de una aplicación móvil. Esto incluye la consideración cuidadosa de la experiencia del usuario (UX), asegurando que la aplicación sea intuitiva, accesible y fácil de usar. Los desarrolladores deben prestar atención a los principios de diseño de material para Android y las pautas de diseño human-interfaz para iOS.

Gestión de Datos

Las aplicaciones móviles a menudo necesitan manejar grandes cantidades de datos, requiriendo sistemas eficientes para su almacenamiento, recuperación y sincronización. Las opciones incluyen bases de datos locales como SQLite o Realm y soluciones en la nube como Firebase o AWS, que también facilitan la sincronización de datos entre múltiples dispositivos y plataformas.

Redes y Comunicaciones

Manejar la comunicación de datos entre la aplicación y la web es otro elemento crucial. Esto puede incluir llamadas a APIs REST, consumo de servicios web y gestión de la conectividad de red, asegurando que la aplicación funcione bien incluso con conexiones de red pobres o intermitentes.

Pruebas y Depuración

Las pruebas son esenciales para garantizar la funcionalidad y la estabilidad de la aplicación antes de su lanzamiento. Esto incluye pruebas unitarias, pruebas de integración y pruebas de interfaz de usuario automatizadas.

Seguridad

Garantizar la seguridad de los datos del usuario es fundamental. Esto incluye la implementación de prácticas de codificación segura, encriptación de datos sensibles, y asegurar las comunicaciones de la aplicación con técnicas como HTTPS y SSL/TLS.

Accesibilidad

Las aplicaciones móviles deben ser accesibles para todos los usuarios, incluyendo aquellos con discapacidades. Esto implica seguir las mejores prácticas de accesibilidad, como etiquetas descriptivas para controles de UI y soporte para lectores de pantalla.

2.5. ACTIVIDADES DE EVALUACIÓN

Para evaluar eficazmente el aprendizaje en la Unidad 2, que cubre arquitecturas, frameworks, SDKs, IDEs, lenguajes de programación, y elementos generales del desarrollo móvil, es crucial diseñar actividades que no solo prueben el conocimiento teórico, sino que también evalúen la capacidad práctica de los estudiantes para aplicar lo aprendido.

Exámenes escritos serán implementados para evaluar la comprensión teórica de los estudiantes sobre arquitecturas, frameworks, SDKs, IDEs y lenguajes de programación. Estos exámenes incluirán preguntas de opción múltiple y verdadero/falso para probar el conocimiento básico, además de preguntas de desarrollo que requerirán que los estudiantes profundicen en la explicación de conceptos más complejos y resuelvan problemas específicos relacionados con la selección y uso de herramientas de desarrollo particulares.

Proyectos de programación ofrecerán a los estudiantes la oportunidad de demostrar su habilidad práctica. Se asignará el desarrollo de aplicaciones sencillas en plataformas Android, iOS o multiplataforma, utilizando las arquitecturas y herramientas estudiadas. Además, se realizarán ejercicios de revisión de código donde los estudiantes identificarán y corregirán fragmentos de código, aplicando las mejores prácticas aprendidas durante el capítulo. Estudios de caso permitirán a los estudiantes aplicar teorías en situaciones del mundo real. Se les pedirá que analicen y seleccionen la arquitectura y el stack tecnológico más adecuados para proyectos específicos, justificando sus decisiones. Además, deberán evaluar críticamente diversos frameworks para determinar cuál es más apropiado para un tipo de aplicación dado, discutiendo sus pros y contras.

Presentaciones servirán para que los estudiantes profundicen en herramientas de desarrollo específicas y compartan sus hallazgos con la clase. Cada estudiante o grupo presentará sobre un IDE o SDK en particular, explicando sus características,

ventajas, y desventajas en comparación con otras opciones en el mercado. Además, se organizarán debates donde los estudiantes argumentarán sobre la eficacia de diferentes lenguajes de programación para ciertos tipos de desarrollo móvil. Evaluación práctica incluirá pruebas de depuración donde los estudiantes enfrentarán aplicaciones con errores intencionales que deberán identificar y corregir. También se les desafiará a integrar múltiples sistemas como bases de datos, interfaces de usuario y APIs externas para crear aplicaciones funcionales.

Evaluación colaborativa se fomentará mediante trabajos en grupo, donde los estudiantes deberán colaborar para diseñar y desarrollar una aplicación, promoviendo la interacción y el aprendizaje entre pares. La revisión por pares también será parte de esta evaluación, donde los estudiantes podrán ofrecer y recibir retroalimentación constructiva sobre los proyectos de sus compañeros. Estas actividades están diseñadas para cubrir un espectro completo de evaluación, desde el conocimiento teórico hasta la aplicación práctica, asegurando que los estudiantes no solo entiendan los materiales de la unidad, sino que también sean capaces de utilizar estos conocimientos en el desarrollo de aplicaciones móviles reales y efectivas.

2.6. REFERENCIAS BIBLIOGRÁFICAS

Amazon Web Services. (2023a). ¿Qué es el SDK? - Explicación del SDK - AWS. Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/sdk/

Amazon Web Services. (2023b). ¿Qué es un IDE? - Explicación de los entornos de desarrollo integrado - AWS. Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/ide/

Ambani, D. (2020). Model View Controller (MVC): A Latest Mobile & Web Application Development Approaches. Vidhyayana - An International Multidisciplinary Peer-Reviewed E-Journal - ISSN 2454-8596, 6(3), Article 3. https://j.vidhyayanaejournal.org/index.php/journal/article/view/353

Anderson, C. (2012). The Model-View-ViewModel (MVVM) Design Pattern. En C. Anderson (Ed.), Pro Business Applications with Silverlight 5 (pp. 461-499). Apress. https://doi.org/10.1007/978-1-4302-3501-9_13

Bascón-Pantoja, E. (2004). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. Acta Nova, 2(4), 493-507. http://www.scielo.org.bo/scielo.php?script=sci_abstract&pid=S1683-07892004000100005&lng=es&nrm=iso&tlng=es

Britch, D., Schonning, N., Dunn, C., & Osborne, J. (2023, julio 13). El patrón Model-View-ViewModel. Microsoft Learn. https://learn.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

Castillo-Gomez, A. E. (2022). Arquitectura limpia con MVVM (Model View Viewmodel) para aplicaciones móviles Android. [Monografia, Universidad Autónoma del Estado de Quintana Roo]. http://risisbi.uqroo.mx/handle/20.500.12249/3225

Codecademy Team. (2024). MVC: Model, View, Controller. Codecademy. https://www.codecademy.com/article/mvc

De la Cruz-Ceron, R. C. (2021). Análisis comparativo entre frameworks de desarrollo para aplicaciones móviles híbridas [Tesis, Universidad Señor de Sipán]. http://repositorio.uss.edu.pe//handle/20.500.12802/9217

Developer Android. (2024a). Descarga Android Studio y App Tools—Android Developers. Andriod Studio. https://developer.android.com/studio?hl=es-419

Developer Android. (2024b). Introducción a Android Studio | Android Developers. Andriod Studio. https://developer.android.com/studio/intro?hl=es-419

Enríquez, F., Fierro, S., Flores, B., Esparza, D. I., & Michelena, J. (2023). Impacto del patrón modelo vista controlador (MVC) en la seguridad, interoperabilidad y usabilidad de un sistema informático durante su ciclo de vida. EASI: Ingeniería y Ciencias Aplicadas en la Industria, 2(1), Article 1. https://doi.org/10.53591/easi.v2i1.2043

Forcada-Sanz, J. (2020). Frameworks para desarrollo de aplicaciones móviles híbridas: Análisis comparativo y aplicación a servicios de emergencia [Tesis, Universidad Politécnica de Madrid]. https://oa.upm.es/64414/

García, R. F. (2023a). MVP: Model–View–Presenter. En R. F. García (Ed.), iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift (pp. 107-144). Apress. https://doi.org/10.1007/978-1-4842-9069-9_3

García, R. F. (2023b). MVVM: Model–View–ViewModel. En R. F. García (Ed.), iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift (pp. 145-224). Apress. https://doi.org/10.1007/978-1-4842-9069-9_4

Garrison, D. R. (1997). Self-Directed Learning: Toward a Comprehensive Model. Adult Education Quarterly, 48(1), 18-33. https://doi.org/10.1177/074171369704800103

Gaudioso, V. (2010). MVVM: Model-View-ViewModel. En V. Gaudioso, T. Brown, B. Renow-Clarke, C. Collins, C. Andres, S. Anglin, M. Beckner, E. Buckingham, G. Cornell, J. Gennick, J. Hassell, M. Lowman, M. Moodie, D. Parkes, J. Pepper, F. Pohlmann, D. Pundick, D. Shakeshaft, M. Wade, & T. Welsh (Eds.), Foundation Expression Blend 4 with Silverlight (pp. 341-367). Apress. https://doi.org/10.1007/978-1-4302-2974-2_15

Gavilánez Álvarez, O. D., Layedra Larrea, N. P., & Ramos Valencia, M. V. (2022). Análisis comparativo de Patrones de Diseño de Software. Polo del Conocimiento: Revista científico - profesional, 7(7 (JULIO 2022)), 2146-2165. https://doi.org/10.23857/pc.v7i7

Knowles, M. S. (1975). Self-directed Learning: A Guide for Learners and Teachers (Vol. 2). Cambridge Adult Education.

Leff, A., & Rayfield, J. T. (2001). Web-application development using the Model/View/Controller design pattern. Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, 118-127. https://doi.org/10.1109/EDOC.2001.950428

López-Corrales, M. A. (2021). Implementación de framework para el desarrollo de aplicaciones móviles multiplataforma basada en componentes JSON aplicando SOA [Tesis, Universidad Católica de Santa María]. https://repositorio.ucsm.edu.pe/handle/20.500.12920/11505

Osuna-Lizárraga, K. A. (2020). Implementación de Flutter para el desarrollo de aplicaciones móviles nativas en iOS y Android [Tesis, Universidad Politécnica de Sinaloa]. http://repositorio.upsin.edu.mx/Fragmentos/tesinas/A031LIZARRAGAOSUNAKEVINANTONIO8101.pdf

Potel, M. (1996). MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. 14.

Quisaguano, L. R., Camalle, T. G., & Toca, J. W. (2022). Análisis comparativo de entornos de desarrollo móvil. Ciencia Latina Revista Científica Multidisciplinar, 6(4), 4478-4498. https://doi.org/10.37811/cl_rcm.v6i4.2950

Rosero-Guerrero, D. F. (2020). Seguridad informática del modelo vista controlador en aplicaciones punto de pago-pos. [Monografia, Universidad Nacional Abierta y a Distancia - UNAD]. http://repository.unad.edu.co/handle/10596/39026

Tixi-Moya, D. J. (2024). Aplicación híbrida utilizando el Framework ionic para la comercialización de productos en la microempresa "Tu Despensa El Tío"

ubicada en la ciudad de Mocha. [Proyecto de Investigación, Universidad Técnica de Ambato]. https://repositorio.uta.edu.ec:8443/jspui/handle/123456789/40761

Vyas, A. (2018, octubre 7). Model View Presenter. Medium. https://anshul-vyas380.medium.com/model-view-presenter-b7ece803203c

Zhang, Y., & Luo, Y. (2010). An architecture and implement model for Model-View-Presenter pattern. 2010 3rd International Conference on Computer Science and Information Technology, 8, 532-536. https://doi.org/10.1109/ICCSIT.2010.5565090

Zurita-Núñez, K. S. (2020). Frameworks para el desarrollo de aplicaciones móviles multiplataforma compiladas de forma nativa – estudio comparativo y ejemplos prácticos [Tesis, Pontifica Universidad Católica del Ecuador]. https://repositorio.puce.edu.ec/handle/123456789/27391

CAPÍTULO 3.

DESARROLLO DE APLICACIONES MÓVILES

Oscar Iván Torres Chicue y Edwin Eduardo Millán Rojas

Capítulo 3
Desarrollo de aplicaciones móviles

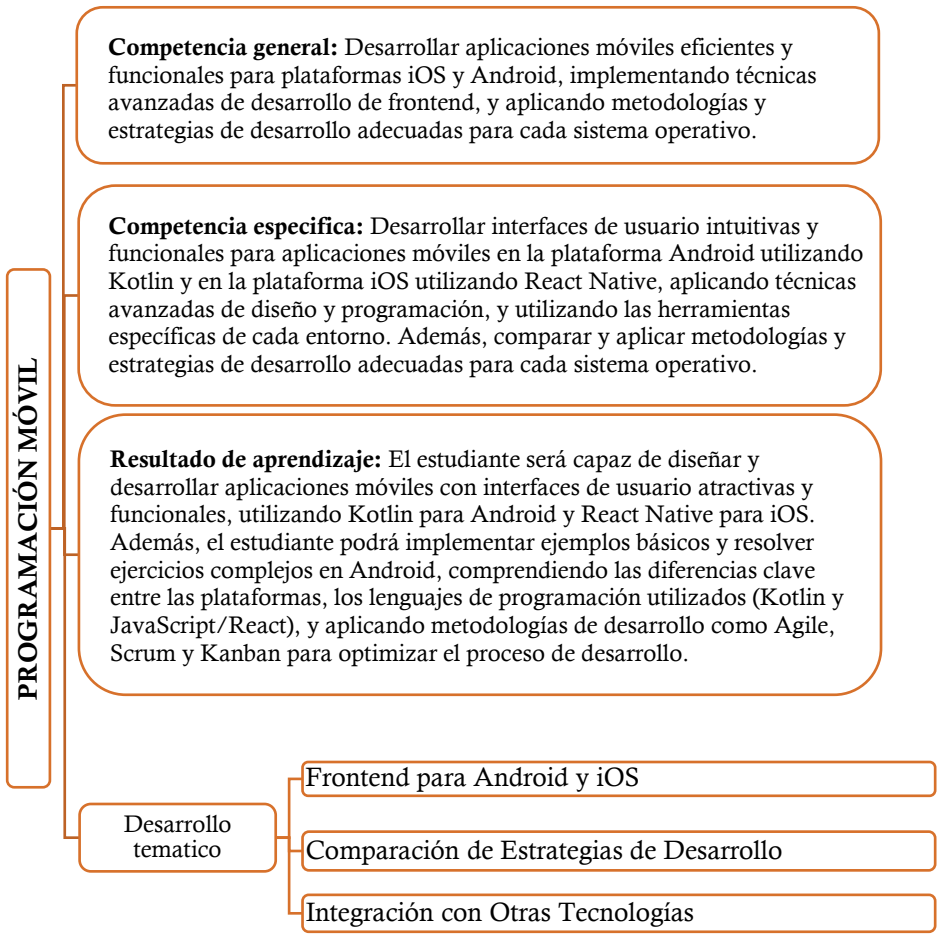
Autores

Oscar Iván Torres Chicue
Estudiante del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: os.torres@udla.edu.co.

Edwin Eduardo Millán Rojas
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: e.millan@udla.edu.co

Como Referenciar el Capítulo de libro.
Torres Chicue, O. I. & Millán Rojas, E. E. (2026). Capítulo 3. Desarrollo de Aplicaciones Móviles. En *Programación Móvil: un enfoque desde la Ingeniería de Sistemas* (págs. 89-). Florencia: Universidad de la Amazonia.

3.1. ESQUEMA CAPÍTULO 3. DESARROLLO DE APLICACIONES MÓVILES.



3.2 COMPETENCIAS Y RESULTADOS DE APRENDIZAJE

La competencia general de este capítulo se centra en capacitar a los estudiantes para desarrollar aplicaciones móviles eficientes y funcionales en las plataformas Android e iOS. Esto incluye el uso de Kotlin para Android y React Native para iOS, herramientas que facilitan la creación de interfaces de usuario intuitivas y atractivas. La competencia general abarca no solo la programación y diseño de estas aplicaciones, sino también la comprensión y aplicación de metodologías de desarrollo adecuadas para cada plataforma, como Agile, Scrum y Kanban. El objetivo es que los estudiantes sean capaces de crear aplicaciones móviles que sean técnicamente sólidas, altamente usables y visualmente atractivas para los usuarios finales.

La competencia específica de este capítulo se enfoca en el desarrollo de interfaces de usuario (UI) y experiencia de usuario (UX) para aplicaciones móviles, utilizando Kotlin para Android y React Native para iOS. Esta competencia incluye el aprendizaje de técnicas avanzadas de diseño y programación, la implementación de ejemplos prácticos, y la resolución de ejercicios complejos específicos de cada plataforma. Los estudiantes aprenderán a comparar y aplicar diferentes metodologías de desarrollo, comprendiendo las ventajas y desventajas de cada una y cómo se adaptan a los distintos entornos de desarrollo móvil.

El resultado de aprendizaje esperado de este capítulo es que los estudiantes sean capaces de diseñar y desarrollar aplicaciones móviles con interfaces de usuario atractivas y funcionales, utilizando Kotlin para Android y React Native para iOS. Se espera que los estudiantes implementen ejemplos básicos y resuelvan problemas complejos específicos de Android, comprendiendo las diferencias clave entre ambas plataformas y los lenguajes de programación utilizados, además de aplicar metodologías de desarrollo eficaces. Al finalizar la unidad, los estudiantes deberán demostrar habilidades avanzadas en la creación de aplicaciones móviles, optimizando tanto la experiencia del usuario como el rendimiento de las aplicaciones.

A continuación, se presentará una tabla de niveles de desempeño asociados al resultado de aprendizaje del Capítulo 3. Esta tabla servirá como guía para evaluar el progreso y la competencia de los estudiantes en el desarrollo de aplicaciones móviles, destacando los criterios clave y las expectativas en cada nivel de desempeño. Los niveles de desempeño permitirán a los estudiantes identificar áreas de mejora y consolidar sus habilidades en el desarrollo de frontend para aplicaciones móviles en Android e iOS.

Tabla 3.1
Niveles de desempeño del resultado de aprendizaje.

Avanzado (4.6 – 5)	Demuestra habilidades excepcionales en el diseño y desarrollo de interfaces de usuario móviles, utilizando Kotlin para Android y React Native para iOS. Resuelve ejercicios complejos con alta eficiencia y optimización.
Intermedio (4 – 4.5)	Muestra competencia sólida en el diseño y desarrollo de interfaces de usuario móviles, utilizando Kotlin para Android y React Native para iOS. Resuelve ejercicios con competencia, aunque con algunas áreas de mejora.
Básico (3 – 3.9)	Presenta habilidades básicas en el diseño y desarrollo de interfaces de usuario móviles. Puede completar ejemplos básicos y algunos ejercicios complejos con apoyo.
Bajo (0 – 2.9)	Tiene dificultades significativas en el diseño y desarrollo de interfaces de usuario móviles. Necesita mejorar en la aplicación de conceptos y técnicas fundamentales.

Nota. Los niveles de desempeño se basan en una escala de 5 puntos. Fuente: elaboración propia.

La tabla de niveles de desempeño proporciona una evaluación estructurada del progreso de los estudiantes en el desarrollo de aplicaciones móviles. La tabla se divide en cuatro niveles: Avanzado, Intermedio, Básico y Bajo, cada uno con un rango de puntuación correspondiente en una escala de 5 puntos. Estos niveles permiten identificar el grado de competencia de los estudiantes, desde aquellos que demuestran habilidades excepcionales y resuelven ejercicios complejos con alta eficiencia, hasta aquellos que necesitan mejorar en la aplicación de conceptos y técnicas fundamentales.

3.3. METODOLOGÍA DE APRENDIZAJE

La metodología para el estudio práctico de frontend para Android e iOS, comparación de estrategias de desarrollo e integración con otras tecnologías, está diseñada para capacitar a los estudiantes en la creación de aplicaciones móviles eficientes y funcionales. El objetivo es que los estudiantes adquieran habilidades prácticas y comprendan las diferencias entre las plataformas, permitiéndoles elegir y aplicar la estrategia más adecuada para cada proyecto. Para facilitar el aprendizaje, se utilizarán guías y manuales técnicos en formato PDF sobre el desarrollo frontend en Kotlin para Android y React Native para iOS, incluyendo ejemplos de código y mejores prácticas. Se complementará con tutoriales en video que mostrarán el proceso de diseño y desarrollo de interfaces, así como proyectos y ejemplos prácticos en GitHub que los estudiantes podrán modificar y extender. Además, se proporcionarán infografías y comparativas que destaquen las diferencias entre estrategias de desarrollo nativo y multiplataforma, y artículos sobre las últimas tendencias y casos de integración con otras tecnologías como APIs y servicios en la nube.

La metodología se basa en técnicas de aprendizaje prácticas como el aprendizaje basado en proyectos, donde los estudiantes desarrollarán aplicaciones completas que integren frontend con servicios externos, permitiéndoles aplicar los conceptos teóricos en escenarios reales. Se realizarán laboratorios prácticos para configurar entornos de desarrollo y resolver ejercicios específicos con retroalimentación inmediata. También se incluirán ejercicios de comparación de estrategias mediante la creación de prototipos en ambas plataformas, evaluando rendimiento, facilidad de uso y tiempo de desarrollo. Además, se llevarán a cabo actividades de resolución de problemas complejos, enfocadas en la integración con APIs externas, manejo de bases de datos y optimización de rendimiento, y estudios de casos que serán discutidos en grupo para reflexionar sobre las decisiones tomadas y sus resultados.

Las evaluaciones incluirán proyectos prácticos en los que los estudiantes deberán desarrollar aplicaciones móviles funcionales, justificando la elección de tecnologías y estrategias. Se realizarán presentaciones grupales para exponer los proyectos, destacar desafíos y soluciones, y recibir retroalimentación. Además, se aplicarán quiz y pruebas prácticas sobre los conceptos clave, así como reflexiones escritas donde los estudiantes analizarán las diferencias entre las plataformas y justificarán sus decisiones. Se fomentará la autoevaluación y coevaluación para promover la reflexión crítica y la autocrítica.

El cronograma se desarrollará en cuatro semanas: la primera semana estará dedicada a la introducción al frontend en Android e iOS, configuración de entornos y desarrollo de interfaces básicas; la segunda semana se centrará en la comparación de estrategias de desarrollo, con la creación de prototipos y análisis de diferencias entre desarrollo nativo y multiplataforma; la tercera semana abordará la integración con otras tecnologías, implementando APIs y servicios externos, y manejo de bases de datos; finalmente, en la cuarta semana, los estudiantes trabajarán en un proyecto integrador y presentarán sus aplicaciones completas que combinen frontend y servicios integrados. Esta metodología proporciona un enfoque práctico y aplicado, permitiendo a los estudiantes desarrollar habilidades avanzadas en la creación de aplicaciones móviles, optimizando tanto la experiencia del usuario como el rendimiento de las aplicaciones.

3.4. DESARROLLO TEMÁTICO

A continuación, se iniciará el desarrollo temático de la Unidad 3, en la cual se explorará en detalle el diseño y desarrollo de aplicaciones móviles para las plataformas Android e iOS. A lo largo de esta unidad, se abordarán aspectos esenciales como el uso de Kotlin y React Native para la creación de interfaces de usuario intuitivas y funcionales, la implementación de ejemplos prácticos y la resolución de ejercicios complejos. También se analizarán las diferencias clave entre ambas plataformas y se aplicarán metodologías de desarrollo adecuadas, con el

objetivo de optimizar tanto la experiencia del usuario como el rendimiento de las aplicaciones.

3.4.1. Frontend para Android y iOS

El frontend, o desarrollo del lado del cliente, es crucial para el éxito de las aplicaciones móviles por varias razones. En primer lugar, define la experiencia del usuario (UX) y la interfaz de usuario (UI), aspectos fundamentales para atraer y retener a los usuarios. Una interfaz bien diseñada no solo mejora la estética de la aplicación, sino que también facilita la navegación y la interacción, lo que puede aumentar la satisfacción y fidelización del usuario (Marques, 2020). Un buen diseño de frontend puede tener un impacto directo en la retención de usuarios. Las investigaciones indican que, si una aplicación no es intuitiva o presenta dificultades de uso, los usuarios tienden a abandonarla rápidamente en busca de alternativas más accesibles. Por ejemplo, aproximadamente el 79% de los usuarios abandonan una aplicación si no encuentran fácilmente lo que buscan (Munawar, 2022). Esta estadística destaca la importancia de un frontend eficiente y bien diseñado para mantener a los usuarios comprometidos y reducir las tasas de abandono.

El rendimiento de una aplicación móvil es otro aspecto crítico influenciado por el frontend. La rapidez con la que una aplicación se carga y responde a las acciones del usuario es vital. Los tiempos de carga largos pueden desanimar a los usuarios, mientras que una aplicación que responde rápidamente puede mejorar significativamente la experiencia del usuario. Además, la percepción del rendimiento puede ser tan importante como el rendimiento real; mostrar indicadores de progreso o mensajes informativos durante operaciones prolongadas puede mantener a los usuarios informados y satisfechos (Cloudinary, 2024).

El frontend también juega un papel crucial en la construcción y fortalecimiento de la marca. Una aplicación móvil con una UI coherente y alineada con la identidad de la marca ayuda a crear una imagen de marca sólida y reconocible. Además, una buena experiencia de usuario puede generar confianza y lealtad hacia la marca. Cuando los usuarios confían en la aplicación, es más

probable que la recomienden a otros, lo que puede aumentar el alcance de la marca y atraer nuevos usuarios (Munawar, 2022). Un ejemplo destacado de cómo un diseño de frontend simple y efectivo puede llevar al éxito es WhatsApp. La aplicación se destacó por su interfaz intuitiva y la eliminación de barreras de uso, como la publicidad. Esto llevó a una alta tasa de uso diario y un crecimiento explosivo en su base de usuarios, lo que eventualmente resultó en su adquisición por parte de Facebook por una suma récord (Marques, 2020).

La importancia del frontend en el desarrollo de aplicaciones móviles no puede ser subestimada. Desde mejorar la experiencia del usuario y la retención hasta construir una marca fuerte y confiable, un buen diseño de frontend es fundamental para el éxito de cualquier aplicación móvil. La inversión en un frontend bien diseñado y funcional no solo mejora la satisfacción del usuario, sino que también puede traducirse en beneficios comerciales significativos.

3.4.1.1. Diseño de Interfaces en Android con Kotlin

El diseño de interfaces en Android con Kotlin se centra en la creación de aplicaciones que sean no solo funcionales, sino también atractivas y fáciles de usar. Kotlin, el lenguaje preferido por Google para el desarrollo de Android ofrece una sintaxis concisa y moderna que facilita la creación de interfaces de usuario (UI) eficientes. Utilizando herramientas como XML para el diseño de la interfaz y Android Studio como entorno de desarrollo integrado (IDE), los desarrolladores pueden crear aplicaciones robustas y de alto rendimiento. Los siguientes son los Componentes Fundamentales:

Layouts: **LinearLayout**: Organiza los elementos en una sola dirección, ya sea vertical u horizontal. **RelativeLayout**: Permite posicionar elementos en relación unos con otros o con el contenedor. **ConstraintLayout**: Proporciona un control más avanzado sobre la disposición de los elementos, similar a un sistema de diseño basado en "constraints" o restricciones.

Views: **TextView**: Muestra texto en la interfaz. **ImageView**: Muestra imágenes. **Button**: Elemento interactivo que puede desencadenar acciones al ser presionado. **EditText**: Campo de entrada de texto.

3.4.1.2. Creación de una Interfaz Básica

Para ilustrar cómo crear una interfaz básica en Android con Kotlin, consideremos un ejemplo sencillo de una aplicación.

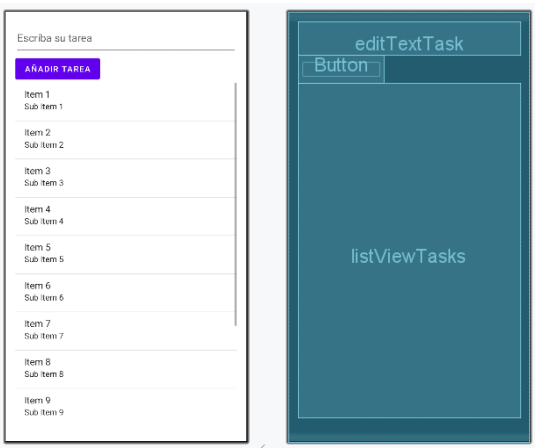
Figura 3.1
Componentes fundamentales de una interfaz en Android.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextTask"
        android:layout_width="match_parent"
        android:layout_height="57dp"
        android:hint="Escriba su tarea" />

    <Button
        android:id="@+id/buttonAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Añadir tarea" />

    <ListView
        android:id="@+id/listViewTasks"
        android:layout_width="match_parent"
        android:layout_height="568dp" />
</LinearLayout>
```



Nota. El diseño del archivo XML define la interfaz de usuario para una aplicación simple de lista de tareas. Fuente: elaboración propia.

A continuación, se explica cómo se vería la interfaz y qué componentes contiene:

LinearLayout:

- **Descripción**: El contenedor principal que organiza los elementos en una disposición vertical.
- **Atributos**:
  - android:layout_width="match_parent": El contenedor ocupa todo el ancho de la pantalla.
  - android:layout_height="match_parent": El contenedor ocupa todo el alto de la pantalla.
  - android:orientation="vertical": Los elementos hijos se organizan de manera vertical, uno debajo del otro.
  - android:padding="16dp": Añade un margen interior de 16dp alrededor de todo el contenedor.

EditText:

- **Descripción**: Un campo de entrada de texto donde los usuarios pueden ingresar una tarea.
- **Atributos**:
  - android:id="@+id/editTextTask": Identificador único del componente para ser referenciado en el código Kotlin.
  - android:layout_width="match_parent": El campo de texto ocupa todo el ancho disponible.
  - android:layout_height="wrap_content": La altura del campo de texto se ajusta al contenido.
  - android:hint="Enter a task": Muestra un texto de sugerencia en el campo de entrada.

Button:

- **Descripción**: Un botón que, al ser presionado, añade la tarea ingresada a la lista.
- **Atributos**:
  - android:id="@+id/buttonAdd": Identificador único del botón.
  - android:layout_width="wrap_content": El botón ocupa solo el espacio necesario para su contenido.
  - android:layout_height="wrap_content": La altura del botón se ajusta a su contenido.
  - android:text="Add Task": El texto que se muestra en el botón.

ListView:

- **Descripción**: Un componente de lista que muestra las tareas ingresadas por el usuario.
- **Atributos**:
  - android:id="@+id/listViewTasks": Identificador único de la lista.

- android:layout_width="match_parent": La lista ocupa todo el ancho disponible.
- android:layout_height="wrap_content": La altura de la lista se ajusta al contenido.

Visualización de la Interfaz

- **Campo de Entrada de Tareas (EditText)**: En la parte superior de la pantalla, hay un campo de texto donde los usuarios pueden escribir la tarea que desean agregar.
- **Botón para Agregar Tareas (Button)**: Debajo del campo de texto, hay un botón etiquetado como "AÑADIR TAREA". Al presionar este botón, la tarea ingresada se añadirá a la lista.
- **Lista de Tareas (ListView)**: Debajo del botón, se muestra una lista que inicialmente estará vacía. A medida que los usuarios ingresan tareas y las agregan usando el botón, las tareas aparecerán en esta lista.

Interacción del Usuario

- Los usuarios pueden ingresar texto en el campo EditText.
- Al presionar el botón Button, el texto ingresado se añade a la ListView y el campo de texto se vacía para permitir la entrada de nuevas tareas.
- La ListView mostrará todas las tareas ingresadas, permitiendo a los usuarios ver la lista completa de sus tareas.

Esta configuración proporciona una interfaz simple y funcional para la gestión de tareas, permitiendo a los usuarios agregar y visualizar sus tareas de manera efectiva.

Herramientas Avanzadas de Diseño

- **Jetpack Compose**: Una herramienta moderna y declarativa de UI que simplifica la creación de interfaces de usuario en Android. Permite a los desarrolladores definir componentes de UI directamente en Kotlin, eliminando la necesidad de XML.
- **Data Binding**: Facilita la vinculación de la lógica de la aplicación con los componentes de la UI, reduciendo el código repetitivo y mejorando la mantenibilidad.

Buenas Prácticas

- **Consistencia Visual**: Mantener una consistencia en el diseño visual utilizando temas y estilos.
- **Optimización del Rendimiento**: Evitar jerarquías de vista profundas y utilizar componentes ligeros.
- **Accesibilidad**: Asegurarse de que la aplicación sea accesible para todos los usuarios, incluyendo descripciones para tecnologías de asistencia y ajustes de contraste.

El diseño de interfaces en Android con Kotlin ofrece una combinación poderosa de herramientas y prácticas que permiten a los desarrolladores crear aplicaciones eficientes y atractivas. Al utilizar componentes fundamentales, herramientas avanzadas como Jetpack Compose y seguir buenas prácticas de diseño, es posible desarrollar aplicaciones que no solo cumplen con los requisitos funcionales, sino que también proporcionan una experiencia de usuario excelente.

3.4.2. Comparación de Estrategias de Desarrollo

En esta sección, comenzaremos a explorar la comparación de las estrategias de desarrollo de aplicaciones móviles, enfocándonos en las diferencias clave entre los lenguajes de programación Kotlin y JavaScript/React. Analizaremos cómo estas tecnologías se utilizan en el desarrollo de aplicaciones para Android e iOS, respectivamente, evaluando sus ventajas y desventajas en términos de rendimiento, facilidad de aprendizaje, y soporte comunitario. Esta comparación nos permitirá comprender mejor cuál de estas herramientas puede ser más adecuada para diferentes tipos de proyectos y equipos de desarrollo.

3.4.2.1. Comparación de Estrategias de Desarrollo: Kotlin vs. JavaScript/React

Kotlin es un lenguaje de programación de tipado estático creado por JetBrains, que se integra perfectamente con Java. Es ampliamente utilizado para desarrollar aplicaciones Android debido a sus múltiples beneficios, como una sintaxis más simplificada y un sistema de seguridad que ayuda a prevenir errores comunes, como las excepciones de puntero nulo. Kotlin permite a los desarrolladores reducir la cantidad de código necesario para lograr la misma funcionalidad que en Java, lo que facilita un desarrollo más eficiente y con menos errores (Sarkar, 2023; Katariya, 2023).

Por otro lado, React Native es un marco de desarrollo de aplicaciones móviles creado por Facebook que utiliza JavaScript y el patrón JSX, una extensión de JavaScript. React Native permite a los desarrolladores crear aplicaciones móviles

que se ejecutan en ambas plataformas, Android e iOS, a partir de una única base de código. Esto se traduce en una reutilización significativa del código y tiempos de desarrollo más rápidos. React Native también ofrece características como el "hot reloading", que permite ver los cambios en tiempo real sin necesidad de recompilar toda la aplicación (Mansour, 2023; Bilenkyi & Dobrytskyi, 2024).

En términos de desempeño, Kotlin tiene una ventaja al ser un lenguaje nativo para Android, lo que le permite acceder directamente a las APIs de Android y ofrecer un rendimiento más fluido y rápido. En contraste, las aplicaciones desarrolladas con React Native pueden enfrentar problemas de rendimiento, especialmente cuando se manejan animaciones complejas o grandes conjuntos de datos, debido a la capa de traducción entre JavaScript y el código nativo (Cohavy, 2024). Sin embargo, React Native brilla en la velocidad de desarrollo y la facilidad de uso para desarrolladores con experiencia en JavaScript. La capacidad de reutilizar el código entre plataformas y la extensa comunidad de soporte hacen que sea una opción atractiva para proyectos que requieren un desarrollo rápido y eficiente (Bilenkyi & Dobrytskyi, 2024).

La curva de aprendizaje para Kotlin puede ser más pronunciada para los desarrolladores que no están familiarizados con Java, mientras que React Native, basado en JavaScript, puede ser más fácil de aprender para aquellos con experiencia en desarrollo web. La comunidad de React Native es más grande y ofrece una mayor cantidad de recursos, bibliotecas de terceros y soporte, lo que facilita la resolución de problemas y la implementación de nuevas características (Mansour, 2023).

Ambos lenguajes tienen sus fortalezas y debilidades. Kotlin es ideal para aplicaciones que requieren un rendimiento óptimo y un control completo sobre los componentes nativos de Android, mientras que React Native es excelente para proyectos que buscan rapidez en el desarrollo y la capacidad de operar en múltiples plataformas con una sola base de código. La elección entre Kotlin y React Native dependerá de las necesidades específicas del proyecto y de la experiencia del equipo de desarrollo.

3.4.2.2. Comparación de Estrategias de Desarrollo: Herramientas de Desarrollo

Android Studio es el IDE oficial para desarrollar aplicaciones Android, lanzado por Google en 2013. Construido sobre IntelliJ IDEA, este entorno ofrece un sistema de compilación adaptable basado en Gradle, una interfaz gráfica de usuario (GUI) intuitiva, y herramientas integradas para depuración y pruebas. Es compatible con varios lenguajes de programación, como Kotlin, Java y C++, y permite a los desarrolladores manejar proyectos de cualquier tamaño de manera eficiente. Además, Android Studio se integra con Google Cloud Platform y GitHub, lo que facilita la colaboración en equipo y el despliegue en la nube. (Stewart, 2024).

Según (Stewart, 2024) algunas ventajas y desventajas de Android Studio son:

Ventajas de Android Studio

- **Integración Nativa**: Ofrece una integración completa con las APIs y herramientas de Android, proporcionando acceso directo a características nativas y soporte para múltiples tipos de dispositivos Android.
- **Rendimiento**: Las aplicaciones desarrolladas en Android Studio suelen tener un rendimiento superior debido al acceso directo a los recursos del sistema y la optimización nativa.
- **Herramientas Avanzadas**: Incluye herramientas robustas para depuración, perfiles de memoria y refactorización de código, lo que facilita el desarrollo de aplicaciones complejas y de alta calidad.

Desventajas de Android Studio

- **Requisitos de Sistema**: Consume muchos recursos del sistema, lo que puede ser un desafío para los desarrolladores con hardware menos potente.
- **Curva de Aprendizaje**: Puede ser difícil para los principiantes debido a la complejidad de sus herramientas y funcionalidades avanzadas.

React Native es un framework de desarrollo de aplicaciones móviles desarrollado por Facebook que permite crear aplicaciones nativas para Android e iOS utilizando JavaScript y el patrón JSX. Adopta la filosofía de "aprende una vez, escribe en todas partes", lo que permite a los desarrolladores utilizar un solo código base para ambas plataformas, reduciendo así considerablemente el tiempo de desarrollo. Además, facilita iteraciones rápidas gracias a funciones como "hot reloading", que permite visualizar los cambios en tiempo real sin necesidad de recompilar toda la aplicación(Paterska, 2022). Según (Paterska, 2022) algunas ventajas y desventajas de Android Studio son:

Ventajas de React Native

- **Código Reutilizable**: Permite compartir un solo código base entre Android e iOS, lo que reduce el tiempo y los costos de desarrollo.
- **Desarrollo Rápido**: Las características como "hot reloading" mejoran la velocidad de desarrollo al permitir cambios rápidos y visualización instantánea de estos cambios.
- **Ecosistema Amplio**: React Native tiene una comunidad grande y activa, con abundantes bibliotecas de terceros y recursos de soporte que facilitan la resolución de problemas y la implementación de nuevas características.

Desventajas de React Native

- **Rendimiento**: Las aplicaciones pueden enfrentar problemas de rendimiento, especialmente cuando se utilizan muchas animaciones o se manejan grandes volúmenes de datos, debido a la capa de traducción entre JavaScript y el código nativo.
- **Integración Nativa**: Puede requerir el uso de módulos nativos para acceder a algunas funcionalidades específicas, lo que puede complicar el desarrollo y la depuración.

Android Studio es ideal para proyectos que requieren un rendimiento óptimo y un control completo sobre los componentes nativos de Android, mientras que **React Native** es excelente para desarrollos rápidos y eficientes que requieren soporte multiplataforma con un solo código base. La elección entre estas herramientas dependerá de las necesidades específicas del proyecto y del equipo de desarrollo.

3.4.2.3. Comparación de Estrategias de Desarrollo: Metodologías de Desarrollo

Agile es un enfoque de desarrollo de software que enfatiza la flexibilidad, la colaboración y la entrega continua de software funcional. Se basa en un conjunto de principios definidos en el Manifiesto Agile, que incluyen la colaboración con el cliente, la adaptación al cambio y la entrega frecuente de software de valor. Agile no es una metodología específica, sino un marco que engloba diversas metodologías como Scrum y Kanban (Brereton, 2023).

Los principios clave de Agile incluyen:

- **Colaboración**: Enfatiza la importancia de la interacción continua entre los equipos de desarrollo y los clientes para asegurar que el producto final cumpla con las expectativas.
- **Adaptabilidad**: Fomenta la flexibilidad y la capacidad de responder rápidamente a los cambios en los requisitos del proyecto.
- **Entrega Continua**: Promueve la entrega regular de pequeñas partes del software que se pueden probar y mejorar iterativamente.

Scrum es una metodología Agile que estructura el trabajo en ciclos cortos llamados sprints, que generalmente duran entre dos y cuatro semanas. Durante cada sprint, un equipo multifuncional trabaja en un conjunto específico de tareas del backlog del producto, priorizado por el Product Owner. Scrum se enfoca en la transparencia, inspección y adaptación, facilitado por roles como el Scrum Master, que asegura la adherencia a los procesos de Scrum, y el Product Owner, que gestiona el backlog del producto (Hagman, 2023).

Los elementos clave de Scrum incluyen:

- **Sprints**: Períodos de tiempo definidos para completar un conjunto de tareas.
- **Reuniones de Scrum**: Incluyen planificaciones de sprint, reuniones diarias, revisiones de sprint y retrospectivas para evaluar y mejorar el proceso.
- **Roles Definidos**: Product Owner, Scrum Máster y Equipo de Desarrollo, cada uno con responsabilidades claras para facilitar el proceso y asegurar la entrega de valor.

Kanban es una metodología Lean que utiliza una visualización del flujo de trabajo para mejorar la eficiencia y reducir el desperdicio. A diferencia de Scrum, el método Kanban no emplea sprints, sino que facilita la entrega continua de tareas a medida que se completan. En Kanban se utiliza un tablero visual para gestionar las actividades, el cual se divide en columnas que reflejan las distintas fases del proceso de trabajo, como "Pendiente", "En curso" y "Completado". Su propósito principal es establecer límites en el trabajo en progreso (WIP) para prevenir la sobrecarga del equipo y garantizar un flujo de trabajo constante y optimizado (Jadhav, 2023).

Los principios clave de Kanban incluyen:

- **Visualización del Trabajo**: Utiliza tableros y tarjetas para representar visualmente las tareas y su estado.
- **Limitación del WIP**: Restringe la cantidad de trabajo en progreso para evitar la saturación del equipo.
- **Mejora Continua**: Enfocado en evaluar y mejorar continuamente el proceso para optimizar el flujo de trabajo.

Diferencias Principales:

- **Estructura y Flexibilidad**: Scrum es más prescriptivo con roles definidos y ceremonias, mientras que Kanban es más flexible y puede ser implementado en cualquier proceso existente sin necesidad de cambios significativos.
- **Ciclo de Trabajo**: Scrum trabaja en sprints definidos, mientras que Kanban permite un flujo continuo de trabajo.
- **Roles y Responsabilidades**: Scrum define roles específicos como el Scrum Master y el Product Owner, mientras que Kanban no prescribe roles específicos y se enfoca en la colaboración general del equipo.

Similitudes:

- Ambas metodologías promueven la transparencia y la mejora continua.
- Están diseñadas para adaptarse a los cambios y optimizar la eficiencia del equipo.
- Utilizan visualizaciones del trabajo para mejorar la gestión y priorización de tareas.

La elección entre Agile, Scrum, Kanban u otra metodología depende de las necesidades específicas del proyecto y del equipo. Scrum es ideal para equipos que necesitan una estructura clara y ciclos de trabajo definidos, mientras que Kanban es más adecuado para equipos que buscan flexibilidad y mejora continua en un flujo de trabajo existente. Agile, como marco general, ofrece los principios fundamentales que guían ambas metodologías para asegurar la entrega de productos de alta calidad y valor para el cliente.

3.4.2.4. Comparación de Estrategias de Desarrollo: Estrategias de Diseño y Usabilidad

El diseño y la usabilidad son componentes críticos en el desarrollo de aplicaciones móviles exitosas. Una buena estrategia de diseño no solo mejora la

estética de la aplicación, sino que también optimiza la experiencia del usuario (UX), lo que puede llevar a una mayor retención y satisfacción del usuario.

Principios de Diseño de UX y UI

Para asegurar una experiencia de usuario positiva, es esencial seguir principios de diseño centrados en el usuario. Esto implica entender y abordar las necesidades y expectativas del usuario desde la fase inicial de investigación hasta la implementación final del diseño. Los principios clave incluyen:

Diseño Centrado en el Usuario: Este enfoque se basa en entender profundamente a los usuarios, sus necesidades, comportamientos y preferencias. Mediante la realización de investigaciones de usuario, análisis de mercado y pruebas de usabilidad, los diseñadores pueden crear aplicaciones que realmente resuenen con los usuarios (Solution Analysts, 2023; King, 2024).

Consistencia y Familiaridad: Seguir las directrices de diseño de plataforma, como las Human Interfaz Guidelines de Apple y las Material Design de Google, ayuda a crear una experiencia de usuario coherente y familiar. Esto facilita la navegación y reduce la curva de aprendizaje, lo que es crucial para la aceptación del usuario (BairesDev, 2024).

Retroalimentación Visual: Proporcionar retroalimentación visual después de acciones significativas, como agregar un artículo al carrito o enviar un formulario, ayuda a los usuarios a entender que sus acciones han sido registradas y procesadas. Esto reduce la incertidumbre y mejora la confianza del usuario en la aplicación (Think with Google, 2016).

Accesibilidad: Asegurarse de que la aplicación sea accesible para todos los usuarios, incluidos aquellos con discapacidades, es esencial. Esto incluye el uso de tamaños de fuente adecuados, contrastes de color suficientes y soporte para tecnologías de asistencia (BairesDev, 2024).

Proceso de diseño de aplicaciones móviles

El proceso de diseño de aplicaciones móviles es iterativo y colaborativo, e incluye varias etapas clave:

Investigación y Conceptualización: Involucra la recopilación de datos sobre las necesidades y comportamientos de los usuarios, así como el análisis del mercado y la competencia. Con esta información, los diseñadores pueden definir los objetivos de la aplicación y conceptualizar sus características principales (King, 2024).

Wireframing y Prototipado: Los wireframes son representaciones visuales de baja fidelidad que muestran la estructura básica y el flujo de navegación de la aplicación. Los prototipos, por otro lado, son maquetas interactivas que simulan la interfaz y la funcionalidad de la aplicación. Estas herramientas permiten a los diseñadores probar y refinar las interacciones del usuario antes de pasar al desarrollo (BairesDev, 2024).

Pruebas e Iteración: Las pruebas de usabilidad, A/B testing y las pruebas beta son cruciales para identificar problemas y áreas de mejora. Basándose en la retroalimentación de los usuarios, los diseñadores pueden realizar ajustes y optimizaciones continuas para asegurar una experiencia de usuario óptima (Solution Analysts, 2023).

Entrega Continua y Actualizaciones: Mantener la aplicación actualizada con nuevas características y correcciones de errores es vital para mantener el compromiso del usuario. Monitorear el rendimiento de la aplicación y la retroalimentación de los usuarios ayuda a identificar oportunidades de mejora y a implementar actualizaciones regulares (Solution Analysts, 2023).

La aplicación de estrategias efectivas de diseño y usabilidad es fundamental para el éxito de las aplicaciones móviles. Al centrarse en las necesidades del usuario, seguir las directrices de diseño de plataforma, proporcionar retroalimentación visual adecuada y asegurar la accesibilidad, los desarrolladores pueden crear aplicaciones atractivas y funcionales que satisfagan y superen las expectativas de los usuarios.

3.4.3 Integración con Otras Tecnologías en Android

La integración de aplicaciones Android con otras tecnologías es crucial para crear aplicaciones robustas y funcionales que pueden interactuar con diversas fuentes de datos y servicios. Aquí se describen algunas de las integraciones más comunes y cómo implementarlas.

Integración con Bases de Datos Locales (Room)

Room es una biblioteca de persistencia de datos que forma parte del conjunto de componentes Jetpack de Android. Facilita la interacción con una base de datos SQLite mediante la abstracción de la complejidad del manejo directo de la base de datos.

Ventajas:

- Abstracción sobre SQLite.
- Integración con LiveData para observar cambios en los datos.
- Migraciones de esquemas manejadas automáticamente.

Integración con Servicios de Red (Retrofit)

Retrofit es una biblioteca de cliente HTTP para Android y Java que facilita la conexión a servicios web RESTful. Es ampliamente utilizada debido a su simplicidad y flexibilidad.

Ventajas:

- Fácil de usar y configurar.
- Soporte para conversión de JSON a objetos Kotlin/Java.
- Interceptores para manejo de peticiones y respuestas.

Integración con Firebase

Firebase proporciona una amplia gama de servicios backend como autenticación, bases de datos en tiempo real, almacenamiento, y más. Es especialmente útil para

aplicaciones que requieren sincronización de datos en tiempo real o autenticación de usuarios.

Ventajas:

- Amplia gama de servicios backend.
- Integración sencilla con aplicaciones Android.
- Sincronización en tiempo real y autenticación segura.

Integración con API Externas (GraphQL)

GraphQL es un lenguaje de consulta para APIs que permite a los clientes solicitar exactamente los datos que necesitan, mejorando la eficiencia en el consumo de datos.

Ventajas:

- Consultas precisas y eficientes.
- Reducción de la sobrecarga de datos.
- Flexibilidad en la obtención de datos.

La integración de aplicaciones Android con otras tecnologías como bases de datos locales, servicios de red, Firebase y GraphQL permite a los desarrolladores crear aplicaciones ricas en funcionalidades y eficientes. Cada tecnología tiene sus propias ventajas y se puede seleccionar según las necesidades específicas del proyecto.

Ejercicio: Creación de un Formulario en Android

En este ejercicio, aprenderemos a crear un formulario en Android utilizando XML para el diseño de la interfaz de usuario y Kotlin para la lógica de la aplicación. El formulario incluirá campos de entrada para el nombre, cédula y número de celular del usuario, además de un botón para cargar una imagen y otro para registrar al usuario. Finalmente, se mostrará un TextView para visualizar mensajes o resultados. A través de este ejercicio, comprenderás cómo utilizar LinearLayout para organizar los elementos, TextInputLayout y TextInputEditText para la entrada de datos, y cómo manejar eventos de clic en botones. Este ejercicio te permitirá

aplicar conceptos clave de diseño de interfaces y programación en Android, facilitando el desarrollo de aplicaciones funcionales y atractivas.

A continuación, se describe cada componente del diseño del formulario en Android utilizando XML y se explica su propósito y atributos:

Figura 3.2
Contenedor Principal: LinearLayout.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="40dp"
    android:layout_marginEnd="40dp"
    android:orientation="vertical">
```

Nota. LinearLayout es el contenedor principal que organiza los elementos de forma vertical. Fuente: elaboración propia a partir de Android Studio.

Atributos:

- ✓ android:layout_width="match_parent": El ancho del contenedor coincide con el ancho del padre.
- ✓ android:layout_height="wrap_content": La altura del contenedor se ajusta al contenido.
- ✓ android:layout_marginStart="40dp" y android:layout_marginEnd="40dp": Margen en los lados izquierdo y derecho.
- ✓ android:orientation="vertical": Los elementos se organizan verticalmente.

Figura 3.3
ImageView para la Imagen del Usuario.

```
<!-- ImageView con margen superior -->
<ImageView
    android:id="@+id/ImagenUsuario"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_margin="20dp"
    android:adjustViewBounds="true"
    android:scaleType="centerCrop"
    android:src="@android:mipmap/sym_def_app_icon" />
```

Nota. Muestra la imagen del usuario. Fuente: elaboración propia a partir de Android Studio.

Atributos:

- ✓ android:id="@+id/ImagenUsuario": Identificador único del ImageView.
- ✓ android:layout_width="match_parent": El ancho del ImageView coincide con el ancho del padre.
- ✓ android:layout_height="200dp": Altura fija de 200dp.
- ✓ android:layout_margin="20dp": Margen alrededor del ImageView.
- ✓ android:adjustViewBounds="true": Ajusta los límites de la vista.
- ✓ android:scaleType="centerCrop": Escala la imagen para llenar el ImageView centrando el contenido.
- ✓ android:src="@android:mipmap/sym_def_app_icon": Fuente de la imagen.

Figura 3.4
Primer Campo de Texto: TextInputLayout y TextInputEditText para Nombre

```
<!-- TextInputLayout 1 -->
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Nombre">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/TextNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="" />

</com.google.android.material.textfield.TextInputLayout>
```

Nota. Campo de entrada para el nombre del usuario. Fuente: elaboración propia a partir de Android Studio.

Atributos:

- ✓ android:layout_width="match_parent": El ancho del campo coincide con el ancho del padre.
- ✓ android:layout_height="wrap_content": La altura del campo se ajusta al contenido.
- ✓ android:hint="Nombre": Texto de sugerencia que se muestra cuando el campo está vacío.
- ✓ android:id="@+id/TextNombre": Identificador único del campo de texto.

Figura 3.5
Segundo Campo de Texto: TextInputLayout y TextInputEditText para Cédula

```
<!-- TextInputLayout 2 con margen superior -->
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:hint="Cedula">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/TextCedula"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint=""
        android:inputType="numberSigned" />

</com.google.android.material.textfield.TextInputLayout>
```

Nota. Campo de entrada para la cédula del usuario. Fuente: elaboración propia a partir de Android Studio.

Atributos:

- ✓ android:layout_width="match_parent": El ancho del campo coincide con el ancho del padre.
- ✓ android:layout_height="wrap_content": La altura del campo se ajusta al contenido.
- ✓ android:layout_marginTop="20dp": Margen superior de 20dp.
- ✓ android:hint="Cedula": Texto de sugerencia que se muestra cuando el campo está vacío.
- ✓ android:id="@+id/TextCedula": Identificador único del campo de texto.
- ✓ android:inputType="numberSigned": Tipo de entrada configurado para números.

Figura 3.6
Tercer Campo de Texto: TextInputLayout y TextInputEditText para Celular

```
<!-- TextInputLayout 3 con margen superior -->
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:hint="Celular">


    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/TextCelular"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="numberSigned" />

</com.google.android.material.textfield.TextInputLayout>
```

Nota. Campo de entrada para el número de celular del usuario. Fuente: elaboración propia a partir de Android Studio.

Atributos:

- ✓ android:layout_width="match_parent": El ancho del campo coincide con el ancho del padre.
- ✓ android:layout_height="wrap_content": La altura del campo se ajusta al contenido.
- ✓ android:layout_marginTop="20dp": Margen superior de 20dp.
- ✓ android:hint="Celular": Texto de sugerencia que se muestra cuando el campo está vacío.
- ✓ android:id="@+id/TextCelular": Identificador único del campo de texto.
- ✓ android:inputType="numberSigned": Tipo de entrada configurado para números.

Figura 3.7

LinearLayout Horizontal para Botón de Imagen y Texto

```
<!-- LinearLayout horizontal para TextView y Botón 1 -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:orientation="horizontal">

    <!-- TextView -->
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Ingresar Imagen"
        android:gravity="center"/>

    <!-- Botón 1 -->
    <Button
        android:id="@+id/BtnImagen"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Imagen"
        android:onClick="ImagePers"/>


</LinearLayout>
```

Nota. Contenedor horizontal que incluye un TextView y un Button. Fuente: elaboración propia a partir de Android Studio.

Atributos del LinearLayout:

- ✓ android:layout_width="match_parent": El ancho del contenedor coincide con el ancho del padre.
- ✓ android:layout_height="wrap_content": La altura del contenedor se ajusta al contenido.
- ✓ android:layout_marginTop="20dp": Margen superior de 20dp.
- ✓ android:orientation="horizontal": Los elementos se organizan horizontalmente.

Atributos del TextView:

✓ android:layout_width="0dp": El ancho del TextView se ajusta según el peso.

✓ android:layout_height="wrap_content": La altura del TextView se ajusta al contenido.

✓ android:layout_weight="1": Peso que determina el espacio ocupado.

✓ android:text="Ingresar Imagen": Texto que se muestra.

✓ android:gravity="center": Centra el texto dentro del TextView.

✓

Atributos del Button:

✓ android:id="@+id/BtnImagen": Identificador único del botón.

✓ android:layout_width="0dp": El ancho del botón se ajusta según el peso.

✓ android:layout_height="wrap_content": La altura del botón se ajusta al contenido.

✓ android:layout_weight="1": Peso que determina el espacio ocupado.

✓ android:text="Imagen": Texto que se muestra en el botón.

✓ android:onClick="ImagePers": Método a llamar cuando se hace clic en el botón.

Figura 3.8

Botón para Registrar Usuario.

```
<!-- Botón 2 -->
<Button
    android:id="@+id/BtnRegistrar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="Registrar Usuario"
    android:onClick="onClick"
    />
```

Nota. Botón para registrar al usuario. Fuente: elaboración propia a partir de Android Studio.

Atributos:

- android:id="@+id/BtnRegistrar": Identificador único del botón.
- android:layout_width="match_parent": El ancho del botón coincide con el ancho del padre.
- android:layout_height="wrap_content": La altura del botón se ajusta al contenido.
- android:layout_marginTop="20dp": Margen superior de 20dp.
- android:text="Registrar Usuario": Texto que se muestra en el botón.
- android:onClick="onClick": Método a llamar cuando se hace clic en el botón.

Figura 3.9
TextView para Mostrar Resultados

```
<!-- TextView 2 -->
<TextView
    android:id="@+id/LabelMostrar"
    android:layout_width="match_parent"
    android:layout_height="71dp"
    android:layout_marginTop="20dp"
    android:text="" />
```

Nota. TextView para mostrar resultados o mensajes. Fuente: elaboración propia a partir de Android Studio.

Atributos:

✓ android:id="@+id/LabelMostrar": Identificador único del TextView.

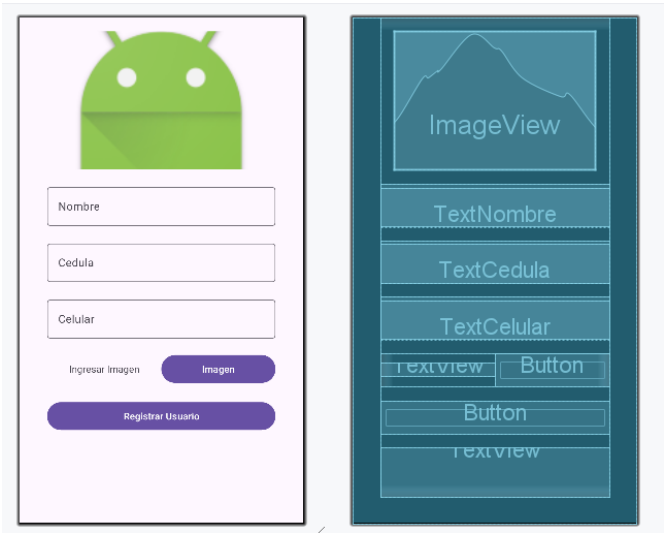
✓ android:layout_width="match_parent": El ancho del TextView coincide con el ancho del padre.

✓ android:layout_height="71dp": Altura fija de 71dp.

✓ android:layout_marginTop="20dp": Margen superior de 20dp.

✓ android:text="": Texto inicial vacío.

Figura 3.10
Interfaz.



Nota. Este diseño crea un formulario simple y funcional para una aplicación Android. Incluye campos de entrada para el nombre, cédula y celular del usuario, un botón para cargar una imagen, un botón para registrar al usuario y un TextView para mostrar resultados o mensajes. Cada componente tiene atributos específicos que definen su apariencia y comportamiento, asegurando una experiencia de usuario coherente y accesible. Fuente: elaboración propia.

3.5. ACTIVIDADES DE EVALUACIÓN

Para evaluar el aprendizaje y la aplicación de conceptos en el desarrollo de aplicaciones Android, se pueden utilizar una variedad de métodos. Aquí se proponen ejercicios prácticos, presentaciones, videos y ejercicios de programación específicos para Android.

Ejercicio 1: Crear una Calculadora Simple

Descripción: Desarrollar una aplicación Android que funcione como una calculadora básica capaz de realizar operaciones de suma, resta, multiplicación y división.

Objetivos:

- Diseñar una interfaz de usuario con botones para los números y las operaciones.
- Implementar la lógica para realizar las operaciones matemáticas.
- Mostrar el resultado en un TextView.

Criterios de Evaluación:

- Funcionalidad completa de la calculadora.
- Interfaz de usuario clara y fácil de usar.
- Manejo adecuado de las operaciones y visualización del resultado.

Instrucciones:

1. Crear un LinearLayout vertical que contenga un TextView para mostrar el resultado y varios Button para los números y las operaciones.
2. Implementar la lógica en Kotlin para manejar los clics de los botones y realizar las operaciones matemáticas.

Ejercicio 2: Crear una Aplicación de Notas

Descripción: Desarrollar una aplicación Android donde los usuarios puedan agregar, editar y eliminar notas.

Objetivos:

- Crear una interfaz de usuario para agregar y mostrar notas.
- Implementar la funcionalidad para agregar, editar y eliminar notas.
- Guardar las notas en la memoria interna del dispositivo.

Criterios de Evaluación:

- Funcionalidad completa para agregar, editar y eliminar notas.
- Interfaz de usuario intuitiva.
- Persistencia de datos utilizando la memoria interna del dispositivo.

Instrucciones:

1. Crear una actividad principal con un EditText y un Button para agregar notas.
2. Mostrar las notas en un RecyclerView.
3. Implementar la lógica para editar y eliminar notas.

Ejercicio 3: Crear una Aplicación de Conversión de La Moneda

Descripción: Desarrollar una aplicación Android que convierta una cantidad de una moneda a otra utilizando tasas de cambio predefinidas.

Objetivos:

- Crear una interfaz de usuario con EditText para ingresar la cantidad y Spinner para seleccionar las monedas.
- Implementar la lógica para convertir la cantidad ingresada a la moneda seleccionada.
- Mostrar el resultado en un TextView.

Criterios de Evaluación:

- Funcionalidad completa de la conversión de moneda.
- Interfaz de usuario clara y fácil de usar.
- Manejo adecuado de las tasas de cambio y visualización del resultado.

Instrucciones:

1. Crear una interfaz con EditText para ingresar la cantidad y Spinner para seleccionar las monedas.
2. Implementar la lógica en Kotlin para realizar la conversión de moneda y mostrar el resultado.

Ejercicios de Evaluación Adicionales Presentaciones

Descripción: Los estudiantes deberán preparar una presentación sobre un tema específico relacionado con el desarrollo de aplicaciones Android, como el uso de RecyclerView, la gestión de la actividad del ciclo de vida, o la implementación de temas y estilos.

Objetivos:

- Desarrollar habilidades de investigación y síntesis de información.
- Mejorar habilidades de presentación y comunicación.
- Profundizar en temas específicos del desarrollo de Android.

Criterios de Evaluación:

- Claridad y estructura de la presentación.
- Profundidad y precisión de la información.
- Uso de ejemplos prácticos y demostraciones.

Videos Tutoriales

Descripción: Los estudiantes deberán crear un video tutorial de 5-10 minutos donde expliquen y demuestren cómo implementar una funcionalidad específica en Android, como la creación de un RecyclerView simple o la gestión de preferencias compartidas (SharedPreferences).

Objetivos:

- Desarrollar habilidades de enseñanza y comunicación.
- Reforzar el conocimiento práctico mediante la demostración.
- Crear recursos útiles para otros estudiantes.

Criterios de Evaluación:

- Claridad y calidad del video.
- Precisión y utilidad del contenido.
- Efectividad en la explicación y demostración de la funcionalidad.

Estos ejercicios de programación y evaluación permiten a los estudiantes aplicar sus conocimientos en situaciones prácticas y demostrar su comprensión del desarrollo de aplicaciones Android. Al completar estos ejercicios, los estudiantes estarán mejor preparados para enfrentar desafíos reales en el desarrollo de aplicaciones móviles.

3.6. REFERENCIAS BIBLIOGRÁFICAS

BairesDev. (2024, abril 9). Mobile App Design: Key UI/UX Principles. BairesDev. https://www.bairesdev.com/blog/mobile-app-design/

Bilenkyi, S., & Dobrytskyi, D. (2024, marzo 28). Kotlin Multiplatform vs. React Native Comparison. Blog - Mind Studios. https://themindstudios.com/blog/kotlin-multiplatform-vs-react-native/

Brereton, J. (2023). Agile vs Scrum vs Kanban: A Comprehensive Comparison. https://www.launchnotes.com/blog/agile-vs-scrum-vs-kanban-a-comprehensive-comparison

Cloudinary. (2024). Front-End Development: The Complete Guide. Cloudinary. https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide

Cohavy, G. (2024, enero 2). Choosing the Best Mobile App Development Framework: React Native vs. Flutter vs. Swift & Kotlin. Medium. https://medium.com/@cohavygal/should-i-use-react-native-flutter-or-native-swift-kotlin-for-mobile-app-f482fd2cf172

Hagman, D. (2023). Lean, Agile, Scrum, and Kanban: A Comparison Guide | Toptal®. Toptal Projects Blog. https://www.toptal.com/project-managers/agile/project-management-blueprint-part-1-agile-scrum-kanban-lean

Jadhav, M. (2023, abril 13). Scrum vs Kanban: Key Differences in Project Management Methodologies. Atlassian Community. https://community.atlassian.com/t5/Agile-articles/Scrum-vs-Kanban-Key-Differences-in-Project-Management/ba-p/2331061

Katariya, L. (2023). React Native vs Kotlin: What Everything You Need To Know [Product Engineering]. https://www.brilworks.com/blog/react-native-vs-kotlin/

King, M. (2024). Mobile App Design Guidelines. Business of Apps. https://www.businessofapps.com/research/mobile-app-design-guidelines/

Mansour, N. (2023, diciembre 3). React Native vs Kotlin Mutliplatform: The 2024 Guide. https://www.instabug.com/blog/react-native-vs-kotlin-mutliplatform-guide

Marques, O. (2020, mayo 19). Front-end Best Practices for Mobile Apps. Oge Marques, PhD. https://www.ogemarques.com/2020/05/19/front-end-best-practices-for-mobile-apps/

Munawar, S. (2022, marzo 17). Top 8 Reasons Why Front End Development Is Important For Your Business Success. Novateus. https://novateus.com/blog/top-8-reasons-why-front-end-development-is-important-for-your-business-success/

Paterska, P. (2022). React Native vs Native App Development: Pros and Cons in 2023 | EL Passion. https://www.elpassion.com/blog/react-native-vs-native-app-development

Sarkar, S. (2023, diciembre 20). Kotlin Vs React Native. Medium. https://medium.com/@sumanta_93769/kotlin-vs-react-native-8916e7f76985

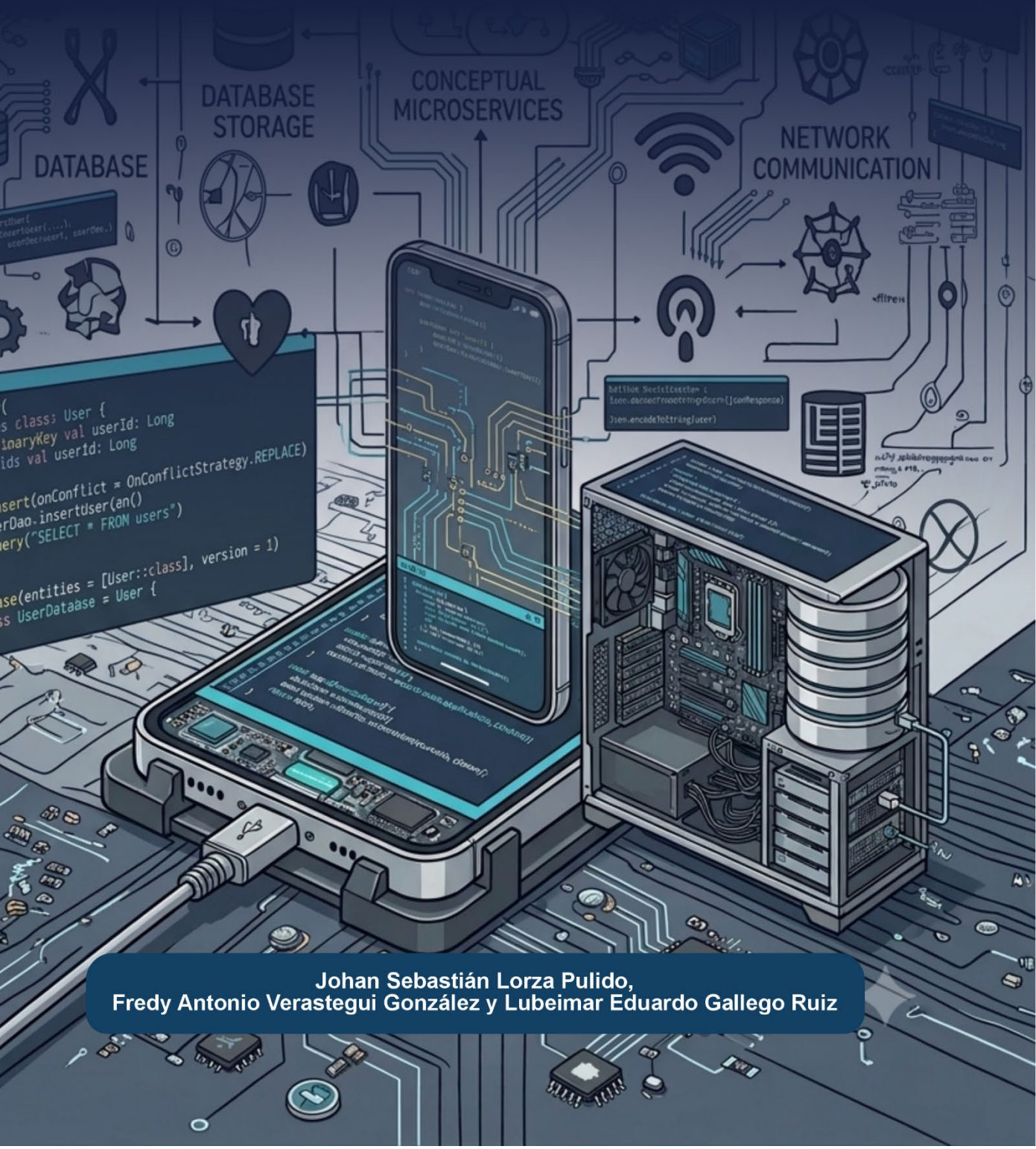
Solution Analysts. (2023, julio 11). 12 Effective Mobile App Development Strategies for 2023. Solution Analysts. https://www.solutionanalysts.com/blog/12-effective-mobile-app-development-strategies-for-2023/

Stewart, H. (2024, febrero 8). A Basic Breakdown of Android Studio vs React Native. Aircada Pro. https://aircada.com/android-studio-vs-react-native/

Think with Google. (2016). Chapter 6: Usability and Comprehension. Think with Google. https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/chapter-6-usability-and-comprehension/

CAPÍTULO 4.

PERSISTENCIA Y COMUNICACIÓN



Johan Sebastián Lorza Pulido,
Fredy Antonio Verastegui González y Lubeimar Eduardo Gallego Ruiz

Capítulo 4.
Persistencia y comunicación

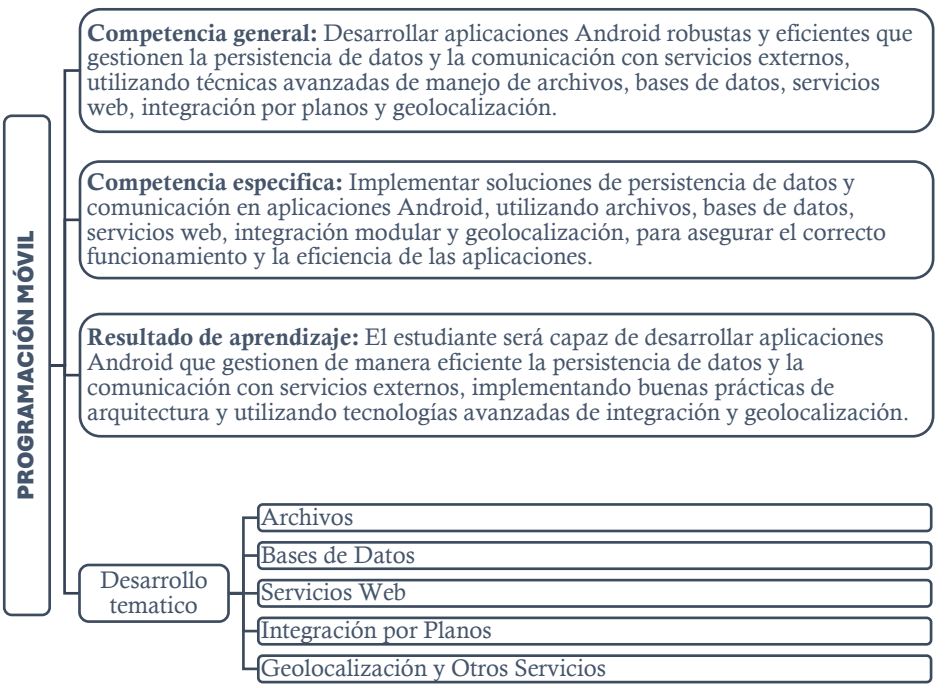
Autores
Johan Sebastián Lorza Pulido
Estudiante del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: j.lorza@udla.edu.co

Fredy Antonio Verastegui González
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: f.verastegui@udla.edu.co

Lubeimar Eduardo Gallego Ruiz
Docente de tiempo completo del programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de la Amazonia, correo: lu.gallego@udla.edu.co

Como Referenciar el Capítulo de libro.
Lorza Pulido, J.S., Verastegui González, F.A. & Gallego Ruiz, L. E. (2026). Capítulo 4. Persistencia y Comunicación. En *Programación Móvil: un enfoque desde la Ingeniería de Sistemas* (págs. 127-159). Florencia: Universidad de la Amazonia

4.1. ESQUEMA CAPÍTULO



4.2 COMPETENCIAS Y RESULTADOS DE APRENDIZAJE

La competencia general en el contexto de la Unidad 4: Persistencia y Comunicación para el desarrollo de aplicaciones Android se centra en la capacidad del estudiante para crear aplicaciones robustas y eficientes que manejen la persistencia de datos y la comunicación con servicios externos. Esta competencia incluye el dominio de técnicas avanzadas para el manejo de archivos, bases de datos, servicios web, integración modular y geolocalización, asegurando que las aplicaciones desarrolladas sean escalables, mantenibles y respondan adecuadamente a las necesidades del usuario final. La competencia específica se enfoca en la implementación de soluciones de persistencia de datos y comunicación dentro de aplicaciones Android. Esto implica que el estudiante debe ser capaz de utilizar archivos para el almacenamiento temporal de datos, gestionar bases de datos para un manejo estructurado y persistente de la información, consumir y manejar servicios web para la interacción con servidores remotos, aplicar principios de integración modular para mantener un código limpio y organizado, y utilizar servicios de geolocalización para agregar funcionalidades basadas en la ubicación. Al dominar esta competencia, el estudiante estará preparado para desarrollar aplicaciones móviles que sean funcionales y eficientes en un entorno real.

El resultado de aprendizaje esperado para esta unidad es que el estudiante sea capaz de desarrollar aplicaciones Android que gestionen de manera eficiente la persistencia de datos y la comunicación con servicios externos. El estudiante debe demostrar una comprensión profunda de las tecnologías y técnicas avanzadas de manejo de archivos, bases de datos, servicios web y geolocalización. Además, debe aplicar buenas prácticas de arquitectura de software y modularizarían del código para asegurar la mantenibilidad y escalabilidad de las aplicaciones. Este resultado refleja la habilidad del estudiante para integrar todos estos elementos en aplicaciones móviles que satisfagan las necesidades y expectativas de los usuarios. A continuación, se presentará una tabla de niveles de desempeño asociados al resultado de aprendizaje del capítulo 4. Esta tabla detallará los diferentes niveles de

competencia que el estudiante puede alcanzar, desde un nivel básico hasta un nivel avanzado, proporcionando una guía clara sobre las expectativas y criterios de evaluación para cada nivel. Esto ayudará a los estudiantes a comprender mejor los objetivos de aprendizaje y a identificar áreas para su desarrollo y mejora continua.

Tabla 4.1
Niveles de Desempeño del Resultado de Aprendizaje.

Avanzado (4.6 – 5)	Demuestra una competencia sobresaliente en la implementación de soluciones de persistencia y comunicación en aplicaciones Android. Utiliza de manera experta archivos, bases de datos, servicios web, integración modular y geolocalización, aplicando buenas prácticas de arquitectura y asegurando la eficiencia y mantenibilidad del código. Las aplicaciones desarrolladas son altamente funcionales, escalables y cumplen con las expectativas del usuario.
Intermedio (4 – 4.5)	Muestra una competencia sólida en la implementación de soluciones de persistencia y comunicación en aplicaciones Android. Utiliza adecuadamente archivos, bases de datos, servicios web, integración modular y geolocalización, con una comprensión clara de las buenas prácticas de arquitectura. Las aplicaciones desarrolladas son funcionales y mantenibles, aunque pueden requerir optimizaciones menores.
Básico (3 – 3.9)	Presenta una competencia básica en la implementación de soluciones de persistencia y comunicación en aplicaciones Android. Puede manejar archivos, bases de datos y servicios web con un conocimiento fundamental, pero necesita mejorar en la aplicación de buenas prácticas de arquitectura y en la integración modular. Las aplicaciones desarrolladas son funcionales, pero pueden tener limitaciones en términos de eficiencia y escalabilidad.
Bajo (0 – 2.9)	Muestra una competencia limitada en la implementación de soluciones de persistencia y comunicación en aplicaciones Android. Tiene dificultades para manejar archivos, bases de datos y servicios web, y carece de comprensión en la aplicación de buenas prácticas de arquitectura y en la integración modular. Las aplicaciones desarrolladas son básicas y a menudo no cumplen con los requisitos esperados en términos de funcionalidad y eficiencia.

Nota. Los niveles de desempeño se basan en una escala de 5 puntos. Fuente: elaboración propia.

Esta tabla proporciona una guía clara sobre las expectativas y criterios de evaluación para cada nivel de desempeño, ayudando a los estudiantes a comprender mejor los objetivos de aprendizaje y a identificar áreas para su desarrollo y mejora continua.

4.3. DESARROLLO TEMÁTICO

Se iniciará el desarrollo temático de la Unidad 4: Persistencia y Comunicación en el contexto del desarrollo de aplicaciones Android. Este capítulo tiene como objetivo proporcionar una comprensión detallada y práctica sobre cómo gestionar la persistencia de datos y la comunicación con servicios externos dentro de una aplicación móvil. A través de esta unidad, exploraremos en profundidad los aspectos fundamentales del manejo de archivos, bases de datos, servicios web, integración modular y geolocalización.

4.3.1. Tipos de archivos

En el desarrollo de aplicaciones Android, la gestión de archivos es fundamental para el almacenamiento y recuperación de datos persistentes. Los archivos pueden ser de diferentes tipos, y cada tipo tiene su uso y técnicas específicas para su manejo. Según Gómez-Jiménez (2022), algunos tipos de archivos son:

1. **Archivos de Texto:**
- Utilizados para almacenar datos en formato de texto plano.
- Son fáciles de leer y escribir, pero no adecuados para grandes volúmenes de datos estructurados.

2. **Archivos Binarios:**
- Almacenan datos en formato binario, lo cual es más eficiente en términos de espacio.
- Se utilizan para almacenar datos que no son fácilmente legibles como imágenes, audio, video y otros tipos de datos binarios.

3. **Archivos Multimedia:**
- Incluyen imágenes, audio y video.
- Requieren un manejo especializado para la reproducción y manipulación dentro de la aplicación.

La gestión adecuada de archivos es crucial para el desarrollo de aplicaciones Android que necesiten almacenar y recuperar datos de manera persistente. Entender los diferentes tipos de archivos y las operaciones básicas de manejo de archivos es esencial para crear aplicaciones robustas y eficientes. Además, SharedPreferences ofrece una solución conveniente para almacenar configuraciones y preferencias del usuario (Android, 2024).

Esta sección proporcionará al lector una comprensión completa de las técnicas de manejo de archivos en Android, preparándolos para implementar soluciones de persistencia de datos en sus propias aplicaciones.

4.3.2. Base de datos

La persistencia de datos es un componente esencial en el desarrollo de aplicaciones Android. Las bases de datos permiten almacenar y gestionar grandes volúmenes de datos estructurados de manera eficiente y segura. En Android, la opción más común para la gestión de bases de datos es SQLite, una base de datos ligera y potente que está integrada en la plataforma Android (Putra et al., 2020). Además, la biblioteca Room Persistence Library proporciona una abstracción sobre SQLite que simplifica la interacción con la base de datos (Android, 2024), ofreciendo una interfaz más amigable y reduciendo el riesgo de errores.

Según Gaffney et al. (2022), SQLite es una biblioteca de software que proporciona una base de datos relacional ligera. A diferencia de otras bases de datos, SQLite no necesita un servidor para funcionar, lo que la hace ideal para aplicaciones móviles donde la simplicidad y el rendimiento son cruciales.

Características y Ventajas de SQLite:

- **Integración nativa con Android:** SQLite está integrada en el SDK de Android, lo que facilita su uso.
- **Ligereza:** Es una base de datos ligera, con un bajo consumo de recursos.

- **Autocontenido:** SQLite no requiere configuración de servidor ni administración, ya que todo el contenido se almacena en un solo archivo.
- **Transacciones ACID:** Asegura que las transacciones sean atómicas, consistentes, aisladas y duraderas.

Gestión de Bases de Datos SQLite

Para gestionar una base de datos SQLite en Android, se utiliza la clase SQLiteOpenHelper, que facilita la creación y actualización de la base de datos (Android, 2024).

Creación y Gestión de la Base de Datos:

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "example.db";
    private static final int DATABASE_VERSION = 1;

    public MyDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, email TEXT)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS users");
        onCreate(db);
    }
}
```

La clase SQLiteOpenHelper es una utilidad proporcionada por Android para gestionar la creación y actualización de bases de datos SQLite.

Definición de la Clase MyDatabaseHelper: Aquí, estamos creando una clase MyDatabaseHelper que extiende SQLiteOpenHelper. Esta clase manejará la creación y gestión de la base de datos.

Definición de las Constantes para el Nombre y Versión de la Base de Datos:

DATABASE_NAME: Especifica el nombre del archivo de la base de datos. En este caso, se llama example.db.

DATABASE_VERSION: Especifica la versión de la base de datos. Es útil para manejar actualizaciones de la base de datos.

Constructor de la Clase MyDatabaseHelper: El constructor llama al constructor de la superclase SQLiteOpenHelper con el contexto de la aplicación, el nombre de la base de datos, un CursorFactory (que normalmente se pasa como null), y la versión de la base de datos.

Método onCreate(SQLiteDatabase db): Este método se llama automáticamente cuando la base de datos se crea por primera vez. Aquí, estamos ejecutando una sentencia SQL para crear una tabla llamada users con tres columnas:

- **id:** un entero que actúa como la clave primaria.
- **name:** un campo de texto para almacenar el nombre del usuario.
- **email:** un campo de texto para almacenar el correo electrónico del usuario.

Método onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion): Este método se llama cuando la versión de la base de datos cambia, es decir, cuando se incrementa DATABASE_VERSION. En este ejemplo, estamos eliminando la tabla users si existe y luego llamando a onCreate(db) para recrearla. Este enfoque simple es adecuado para cambios iniciales, pero en una aplicación real, querrías manejar las migraciones de datos de manera más sofisticada para no perder datos existentes.

Operaciones CRUD

Las operaciones CRUD (Create, Read, Update, Delete) son fundamentales para la gestión de bases de datos.

Crear (Insert):

La operación de creación inserta nuevos registros en la base de datos. Utilizamos el método insert para esto.

```
// Crear un nuevo registro
ContentValues values = new ContentValues();
values.put("name", "John Doe");
values.put("email", "john.doe@example.com");

// Insertar el nuevo registro
long newRowId = db.insert("users", null, values);
```

ContentValues: Se utiliza para almacenar un conjunto de valores que representan el nuevo registro a insertar. Las claves deben ser los nombres de las columnas y los valores deben ser los datos para insertar.

Insertar Registro: El método insert se llama en la instancia de la base de datos (db). El primer argumento es el nombre de la tabla (users), el segundo es un valor nulo opcional, y el tercero son los valores para insertar.

Leer (Query)

La operación de lectura consulta registros en la base de datos y devuelve un Cursor que puede usarse para iterar sobre los resultados.

```
// Consultar el registro
Cursor cursor = db.query(
        "users",                        // Nombre de la tabla
        new String[]{"id", "name", "email"}, // Columnas a devolver
        "id = ?",                       // Columnas para la cláusula WHERE
        new String[]{String.valueOf(newRowId)}, // Valores para la cláusula WHERE
        null,                           // Agrupamiento de filas
        null,                           // Filtrado por grupo de filas
        null                            // Orden de las filas
);

if (cursor.moveToFirst()) {
    String name = cursor.getString(cursor.getColumnIndexOrThrow("name"));
    String email = cursor.getString(cursor.getColumnIndexOrThrow("email"));
    System.out.println("User: " + name + ", Email: " + email);
}
cursor.close();
```

Consulta: El método query se utiliza para seleccionar registros de la base de datos. Los argumentos incluyen:

- Nombre de la tabla ("users").
- Array de columnas a devolver (new String[]{"id", "name", "email"}).
- Cláusula WHERE ("id = ?").
- Valores para la cláusula WHERE (new String[]{String.valueOf(newRowId)}).

Procesar Resultados: El cursor se mueve al primer resultado con moveToFirst() y se extraen los valores de las columnas con getString.

Actualizar (Update)

La operación de actualización modifica registros existentes en la base de datos. Utilizamos el método update para esto.

```
// Valores a actualizar
ContentValues values = new ContentValues();
values.put("name", "Jane Doe");

// Actualizar el registro
int count = db.update(
        "users",          // Nombre de la tabla
        values,           // Valores a actualizar
        "id = ?",         // Cláusula WHERE
        new String[]{String.valueOf(newRowId)} // Valores para la cláusula WHERE
);
```

Valores Por Actualizar: Similar a la inserción, usamos ContentValues para especificar los nuevos valores.

Actualizar Registro: El método update se llama en la instancia de la base de datos (db). Los argumentos incluyen:

- Nombre de la tabla ("users").
- Valores por actualizar (values).
- Cláusula WHERE ("id = ?").
- Valores para la cláusula WHERE (new String[]{String.valueOf(newRowId)}).

Eliminar (Delete)

La operación de eliminación borra registros existentes de la base de datos. Utilizamos el método delete para esto.

```
// Eliminar el registro
int deletedRows = db.delete(
        "users",          // Nombre de la tabla
        "id = ?",         // Cláusula WHERE
        new String[]{String.valueOf(newRowId)} // Valores para la cláusula WHERE
);
```

Eliminar Registro: El método delete se llama en la instancia de la base de datos (db). Los argumentos incluyen:

- Nombre de la tabla ("users").
- Cláusula WHERE ("id = ?").
- Valores para la cláusula WHERE (new String[]{String.valueOf(newRowId)}).

Las operaciones CRUD son fundamentales para la manipulación de datos en una base de datos. Utilizando SQLiteOpenHelper y los métodos insert, query, update, y delete, podemos gestionar eficazmente los datos en una aplicación Android. Cada una de estas operaciones juega un papel crucial en la gestión de la persistencia de datos, asegurando que las aplicaciones sean capaces de almacenar, recuperar, modificar y eliminar datos de manera eficiente y segura.

4.3.3. Servicio web

En el desarrollo de aplicaciones Android, la comunicación con servicios web es fundamental para crear aplicaciones dinámicas y conectadas que puedan interactuar con servidores remotos, obtener datos en tiempo real, y sincronizar información. Los servicios web permiten a las aplicaciones acceder a recursos externos, como bases de datos en la nube, servicios de autenticación, APIs de terceros, y mucho más (Sraïri, 2014; Gironés & Mauri, 2022). En este contexto, los servicios RESTful son ampliamente utilizados debido a su simplicidad y eficiencia.

Conceptos Básicos de APIs y Servicios Web

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de reglas que permiten que diferentes programas se comuniquen entre sí. En el contexto de servicios web, una API RESTful es una interfaz que sigue los principios de REST (Representational State Transfer). REST utiliza métodos HTTP estándar como GET, POST, PUT y DELETE para interactuar con recursos definidos por URLs (Ahmad et al., 2021; Ehsan et al., 2022).

Principios de REST

De acuerdo con Hernández et al. (2021) los principios de la arquitectura REST son los siguientes:

1. **Uniform Interface:** Utiliza una interfaz uniforme para interactuar con recursos.
2. **Stateless:** Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud.
3. **Client-Server:** Separa las responsabilidades del cliente y el servidor.
4. **Cacheable:** Las respuestas deben ser explícitamente marcadas como cacheables o no cacheables.
5. **Layered System:** La arquitectura debe ser organizada en capas.

Consumo de APIs RESTful con Retrofit

Retrofit es una biblioteca de clientes HTTP para Android y Java, desarrollada por Square, que facilita la conexión a servicios web. Simplifica el consumo de APIs RESTful al proporcionar una interfaz clara y concisa para realizar solicitudes HTTP y manejar respuestas (Android Developers, 2024).

Definir la Interfaz de la API:

Crea una interfaz que defina los endpoints de la API y los métodos HTTP correspondientes.

```
public interface ApiService {
    @GET("users/{id}")
    Call<User> getUserById(@Path("id") int userId);

    @POST("users")
    Call<User> createUser(@Body User user);

    @PUT("users/{id}")
    Call<User> updateUser(@Path("id") int userId, @Body User user);

    @DELETE("users/{id}")
    Call<Void> deleteUser(@Path("id") int userId);
}
```

Declaración de la Interfaz: Aquí estamos declarando una interfaz pública llamada ApiService. En Retrofit, esta interfaz definirá los endpoints y métodos HTTP.

Método GET para Obtener un Usuario por ID:

- @GET("users/{id}"): Anotación que indica que este método realizará una solicitud GET a la URL users/{id}, donde {id} es un valor de ruta que será reemplazado en tiempo de ejecución.
- Call<User>: El tipo de retorno del método es Call<User>, lo que indica que este método devolverá un objeto User envuelto en una llamada Retrofit.
- @Path("id") int userId: Anotación @Path que indica que el argumento userId será utilizado para reemplazar {id} en la URL.

Método POST para Crear un Usuario:

- @POST("users"): Anotación que indica que este método realizará una solicitud POST a la URL users.
- Call<User>: El tipo de retorno del método es Call<User>, lo que indica que este método devolverá el objeto User creado envuelto en una llamada Retrofit.
- @Body User user: Anotación @Body que indica que el objeto user será convertido a JSON y enviado en el cuerpo de la solicitud POST.

Método PUT para Actualizar un Usuario:

- @PUT("users/{id}"): Anotación que indica que este método realizará una solicitud PUT a la URL users/{id}, donde {id} es un valor de ruta.
- Call<User>: El tipo de retorno del método es Call<User>, indicando que este método devolverá el objeto User actualizado envuelto en una llamada Retrofit.
- @Path("id") int userId: Anotación @Path que indica que el argumento userId reemplazará {id} en la URL.
- @Body User user: Anotación @Body que indica que el objeto user será convertido a JSON y enviado en el cuerpo de la solicitud PUT.

Método DELETE para Eliminar un Usuario:

- @DELETE("users/{id}"): Anotación que indica que este método realizará una solicitud DELETE a la URL users/{id}, donde {id} es un valor de ruta.
- Call<Void>: El tipo de retorno del método es Call<Void>, indicando que este método devolverá una llamada Retrofit sin cuerpo de respuesta (indicado por Void).
- @Path("id") int userId: Anotación @Path que indica que el argumento userId reemplazará {id} en la URL.

Crear la Instancia de Retrofit:

Configura Retrofit para que use la interfaz de la API y el convertidor Gson para manejar las respuestas JSON.

```
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.example.com/")
        .addConverterFactory(GsonConverterFactory.create())
        .build();

ApiService apiService = retrofit.create(ApiService.class);
```

Después de definir la interfaz de la API, necesitamos crear una instancia de Retrofit y utilizar esta interfaz para realizar solicitudes HTTP.

- Retrofit.Builder(): Se utiliza para crear una instancia de Retrofit.
- baseUrl("https://api.example.com/"): Especifica la URL base de la API.
- addConverterFactory(GsonConverterFactory.create()): Agrega un convertidor Gson para manejar las respuestas JSON.
- build(): Construye la instancia de Retrofit.
- apiService = retrofit.create(ApiService.class): Crea una implementación de la interfaz ApiService utilizando Retrofit.

Definir la interfaz de la API es un paso esencial en el uso de Retrofit para la comunicación con servicios web en Android. La interfaz declara los endpoints de la API y los métodos HTTP, lo que permite a Retrofit generar automáticamente las implementaciones necesarias para realizar las solicitudes HTTP. Este enfoque proporciona una manera limpia y eficiente de interactuar con servicios web, reduciendo la cantidad de código boilerplate y facilitando el manejo de respuestas y errores.

Realizar Solicitudes HTTP:

Utiliza la instancia de Retrofit para realizar solicitudes HTTP. Cada método de la interfaz devuelve un objeto Call que puedes usar para ejecutar la solicitud de manera sincrónica o asincrónica.

Ejemplo de Solicitud GET:

```
Call<User> call = apiService.getUserById(1);
call.enqueue(new Callback<User>() {
    @Override
    public void onResponse(Call<User> call, Response<User> response) {
        if (response.isSuccessful()) {
            User user = response.body();
            System.out.println("User Name: " + user.getName());
        } else {
            System.out.println("Request failed with code: " + response.code());
        }
    }

    @Override
    public void onFailure(Call<User> call, Throwable t) {
        t.printStackTrace();
    }
});
```

apiService.getUserById(1): Llama al método getUserById definido en la interfaz ApiService con el userId de 1.
call.enqueue(new Callback<User>()): Encola la llamada de manera asincrónica y define el comportamiento en onResponse y onFailure.

Ejemplo de Solicitud POST:

```
User newUser = new User("Jane Doe", "jane.doe@example.com");
Call<User> call = apiService.createUser(newUser);
call.enqueue(new Callback<User>() {
    @Override
    public void onResponse(Call<User> call, Response<User> response) {
        if (response.isSuccessful()) {
            User user = response.body();
            System.out.println("Created User ID: " + user.getId());
        } else {
            System.out.println("Request failed with code: " + response.code());
        }
    }

    @Override
    public void onFailure(Call<User> call, Throwable t) {
        t.printStackTrace();
    }
});
```

- Se crea una instancia de User con los datos del nuevo usuario.
- call.enqueue(new Callback<User>()): Encola la solicitud para ejecutarse de manera asincrónica.
- onResponse: Se llama cuando la respuesta es recibida. Si la solicitud es exitosa (response.isSuccessful()), se procesa la respuesta.
- onFailure: Se llama si la solicitud falla debido a un error de red u otro problema.

Realizar solicitudes HTTP con Retrofit es sencillo y eficiente. Al definir una interfaz de API y configurar Retrofit, puedes realizar fácilmente solicitudes GET, POST, PUT, DELETE, y más. Retrofit maneja automáticamente la conversión de datos JSON a objetos Java y viceversa, y proporciona métodos asincrónicos para manejar respuestas de manera eficiente, mejorando la experiencia del desarrollador y la robustez de la aplicación.

Manejo de Respuestas JSON y XML

Retrofit facilita el manejo de respuestas en formato JSON y XML utilizando convertidores. El más común es el convertidor Gson, que convierte automáticamente las respuestas JSON a objetos Java.

Ejemplo de Conversión con Gson:

```
public class User {
    private int id;
    private String name;
    private String email;

    // Getters and setters
}
```

Retrofit utiliza el convertidor Gson para convertir la respuesta JSON a una instancia de la clase User.

Autenticación y Manejo de Sesiones

La autenticación es un aspecto crucial en la comunicación con servicios web. Los métodos comunes incluyen autenticación basada en tokens (JWT), OAuth, y autenticación básica HTTP. Retrofit puede manejar la autenticación a través de interceptores de OkHttp.

Ejemplo de Autenticación con Token:

```
OkHttpClient client = new OkHttpClient.Builder()
        .addInterceptor(chain -> {
            Request original = chain.request();
            Request request = original.newBuilder()
                    .header("Authorization", "Bearer " + token)
                    .build();
            return chain.proceed(request);
        })
        .build();

Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.example.com/")
        .client(client)
        .addConverterFactory(GsonConverterFactory.create())
        .build();

ApiService apiService = retrofit.create(ApiService.class);
```

Crear el Cliente OkHttp:

- OkHttpClient.Builder(): Se crea una nueva instancia del constructor de OkHttpClient.
- addInterceptor(chain -> { ... }): Se añade un interceptor al cliente. El interceptor es una función lambda que toma un Interceptor.Chain y devuelve una Response.
- Request original = chain.request(): Se obtiene la solicitud original.

- Request request = original.newBuilder(): Se crea una nueva solicitud basada en la original.
- .header("Authorization", "Bearer " + token): Se añade el encabezado Authorization con el valor Bearer <token> a la nueva solicitud. El token debe ser una cadena que has obtenido previamente (por ejemplo, al iniciar sesión).
- return chain.proceed(request): Se procede con la solicitud modificada.

Configurar Retrofit:

- Retrofit.Builder(): Se crea una nueva instancia del constructor de Retrofit.
- baseUrl("https://api.example.com/"): Se especifica la URL base de la API.
- .client(client): Se establece el cliente OkHttp configurado anteriormente.
- addConverterFactory(GsonConverterFactory.create()): Se añade un convertidor Gson para manejar las respuestas JSON.
- build(): Se construye la instancia de Retrofit.

La autenticación con token utilizando Retrofit y OkHttp es un enfoque eficaz para asegurar las solicitudes HTTP en aplicaciones Android. Al utilizar un interceptor de OkHttp, puedes añadir automáticamente el token de autenticación a cada solicitud, garantizando que todas las interacciones con la API sean seguras. Este método no solo simplifica el código, sino que también mejora la seguridad y la mantenibilidad de la aplicación.

Ejemplos de Integración con Servicios Web Populares

Retrofit se puede utilizar para integrar aplicaciones Android con una variedad de servicios web populares, como las APIs de Google, redes sociales (Facebook, Twitter), servicios de pago (Stripe, PayPal), y muchos más.

Ejemplo de Integración con la API de GitHub:

```
public interface GitHubService {
    @GET("users/{username}/repos")
    Call<List<Repo>> listRepos(@Path("username") String username);
}

// Configuración de Retrofit
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .addConverterFactory(GsonConverterFactory.create())
        .build();

GitHubService service = retrofit.create(GitHubService.class);

// Realizar solicitud
Call<List<Repo>> repos = service.listRepos("octocat");
repos.enqueue(new Callback<List<Repo>>() {
    @Override
    public void onResponse(Call<List<Repo>> call, Response<List<Repo>> response) {
        if (response.isSuccessful()) {
            List<Repo> reposList = response.body();
            for (Repo repo : reposList) {
                System.out.println("Repo: " + repo.getName());
            }
        }
    }

    @Override
    public void onFailure(Call<List<Repo>> call, Throwable t) {
        t.printStackTrace();
    }
});
```

La integración con servicios web es esencial para el desarrollo de aplicaciones Android conectadas y dinámicas. Retrofit simplifica significativamente el proceso de consumo de APIs RESTful, ofreciendo una interfaz fácil de usar y herramientas poderosas para manejar solicitudes y respuestas HTTP. Al dominar estas técnicas, los desarrolladores pueden crear aplicaciones que interactúan de manera eficiente con servidores remotos, ofreciendo una experiencia rica y dinámica a los usuarios.

4.3.4. Integración por planos

La integración por planos en el desarrollo de aplicaciones Android se refiere a la práctica de organizar y estructurar el código de manera modular y mantenible. Utilizando principios de arquitectura limpia y patrones de diseño, los desarrolladores pueden construir aplicaciones que sean fáciles de mantener, escalar y probar (González-Díez, 2016). Este enfoque es especialmente útil en proyectos grandes y complejos donde la separación de responsabilidades y la claridad del código son cruciales.

Principios de Arquitectura Limpia

La arquitectura limpia, o "Clean Architecture", propone una estructura de software que facilita el mantenimiento y la escalabilidad al separar claramente las responsabilidades de cada componente del sistema (Godoy et al., 2024). Este enfoque asegura que las dependencias entre los módulos estén bien definidas y minimiza el acoplamiento entre ellos.

Principios Clave

De acuerdo con Dellepiane Espinoza & Díaz Alvarado (2016) los principios claves de las arquitecturas limpias son:

1. **Separación de responsabilidades:** Cada capa tiene una responsabilidad específica y bien definida.
2. **Independencia de frameworks:** La arquitectura no debe depender de detalles de implementación de frameworks específicos.
3. **Testabilidad:** El diseño debe permitir pruebas unitarias fáciles y efectivas.
4. **Independencia de UI:** Los detalles de la interfaz de usuario (UI) deben estar separados de la lógica de negocio.

Patrones de Diseño

Varios patrones de diseño pueden ser utilizados para implementar la arquitectura limpia en Android. Los dos patrones más comunes son MVVM (Model-View-ViewModel) y MVP (Model-View-Presenter).

MVVM (Model-View-ViewModel)

Según Castillo-Gomez (2022), el MVVM es un patrón de diseño que facilita la separación entre la lógica de presentación y la lógica de negocio, permitiendo una mejor organización del código y una mayor testabilidad.

Componentes de MVVM:

- **Model:** Representa los datos y la lógica de negocio.
- **View:** Representa la UI y sus interacciones.
- **ViewModel:** Actúa como un intermediario entre el Model y la View, gestionando la lógica de presentación.

Ejemplo de MVVM:

Model:

```
public class User {
    private String name;
    private String email;

    // Getters y setters
}
```

View (Activity):

```
public class MainActivity extends AppCompatActivity {
    private UserViewModel userViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        userViewModel = new ViewModelProvider(this).get(UserViewModel.class);

        userViewModel.getUser().observe(this, user -> {
            // Actualizar la UI con los datos del usuario
        });
    }
}
```

ViewModel:

```
public class UserViewModel extends ViewModel {
    private MutableLiveData<User> user;

    public LiveData<User> getUser() {
        if (user == null) {
            user = new MutableLiveData<>();
            loadUser();
        }
        return user;
    }

    private void loadUser() {
        // Cargar datos del usuario (posiblemente desde una base de datos o una API)
        User userData = new User();
        userData.setName("John Doe");
        userData.setEmail("john.doe@example.com");
        user.setValue(userData);
    }
}
```

MVP (Model-View-Presenter)

Según O'g'li (2024), el MVP es un patrón de diseño popular que separa la lógica de presentación de la lógica de negocio. En MVP, el Presenter actúa como un intermediario entre la View y el Model.

Componentes de MVP:

- **Model:** Representa los datos y la lógica de negocio.
- **View:** Representa la UI y sus interacciones.
- **Presenter:** Gestiona la lógica de presentación y actúa como un intermediario entre la View y el Model.

Ejemplo de MVP:

Model:

```
public class User {
    private String name;
    private String email;

    // Getters y setters
}
```

View (Activity):

```
public class MainActivity extends AppCompatActivity implements UserView {
    private UserPresenter userPresenter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        userPresenter = new UserPresenter(this);
        userPresenter.loadUser();
    }

    @Override
    public void showUser(User user) {
        // Actualizar la UI con los datos del usuario
    }
}
```

Presenter:

```
public class UserPresenter {
    private UserView view;
    private User user;

    public UserPresenter(UserView view) {
        this.view = view;
    }

    public void loadUser() {
        // Cargar datos del usuario (posiblemente desde una base de datos o una API)
        user = new User();
        user.setName("John Doe");
        user.setEmail("john.doe@example.com");
        view.showUser(user);
    }
}
```

View Interface:

```
public interface UserView {
    void showUser(User user);
}
```

Separación de Responsabilidades y Organización del Código

La separación de responsabilidades y la organización del código son fundamentales para mantener la claridad y la mantenibilidad de un proyecto. En una arquitectura bien organizada, cada módulo o componente tiene una función específica y no depende innecesariamente de otros módulos.

Ejemplo de Organización de Código:

- **models:** Contiene las clases de datos.
- **views:** Contiene las actividades, fragmentos y vistas personalizadas.
- **viewmodels (o presenters):** Contiene las clases que gestionan la lógica de presentación.
- **repositories:** Contiene las clases que gestionan el acceso a los datos.
- **network:** Contiene las clases relacionadas con las solicitudes de red y APIs.
- **utils:** Contiene utilidades y clases auxiliares.

La integración por planos, mediante el uso de principios de arquitectura limpia y patrones de diseño como MVVM y MVP, es crucial para la construcción de aplicaciones Android mantenibles y escalables. Al estructurar el código de manera modular y seguir buenas prácticas de desarrollo, los desarrolladores pueden crear aplicaciones que no solo sean robustas y eficientes, sino también fáciles de mantener y expandir. Esta organización del código facilita la colaboración en equipos de desarrollo y mejora la calidad del software a largo plazo.

4.3.5. Geolocalización y otros servicios

La geolocalización y otros servicios externos son componentes clave en muchas aplicaciones móviles modernas. Estos servicios permiten a las aplicaciones proporcionar funcionalidades avanzadas basadas en la ubicación del usuario, notificaciones en tiempo real, almacenamiento en la nube, y más. En el contexto de aplicaciones Android, Google Play Services ofrece una variedad de APIs que facilitan la integración de estas funcionalidades.

Servicios de Geolocalización

La geolocalización permite a las aplicaciones determinar la ubicación física del dispositivo. Esto es útil para una variedad de aplicaciones, incluyendo navegación, servicios de transporte, entrega de productos, y aplicaciones de redes sociales.

Uso de Google Play Services para Geolocalización

Google Play Services proporciona una API de geolocalización que simplifica la obtención de la ubicación del dispositivo.

```
public class MainActivity extends AppCompatActivity {

    private FusedLocationProviderClient fusedLocationClient;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);

        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
                && ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
            return;
        }

        fusedLocationClient.getLastLocation()
            .addOnSuccessListener(this, location -> {
                if (location != null) {
                    double latitude = location.getLatitude();
                    double longitude = location.getLongitude();
                    System.out.println("Latitude: " + latitude + ", Longitude: " + longitude);
                }
            });
    }
}
```

En tu actividad principal, configura FusedLocationProviderClient para obtener la ubicación del dispositivo.

```
fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
```

FusedLocationProviderClient: Se utiliza para obtener la última ubicación conocida del dispositivo.

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
    return;
}
```

Comprobación de Permisos: Antes de acceder a la ubicación, debes comprobar si los permisos necesarios han sido otorgados. Si no, debes solicitarlos.

```
fusedLocationClient.getLastLocation()
    .addOnSuccessListener(this, location -> {
        if (location != null) {
            double latitude = location.getLatitude();
            double longitude = location.getLongitude();
            System.out.println("Latitude: " + latitude + ", Longitude: " + longitude);
        }
    });
```

Obtener la Última Ubicación: Una vez que se han otorgado los permisos, puedes obtener la última ubicación conocida del dispositivo.

Integración con Mapas

La integración con mapas es una funcionalidad común en aplicaciones que utilizan geolocalización. Google Maps API permite a los desarrolladores mostrar mapas interactivos en sus aplicaciones.

Paso 1: Obtener la Clave de API de Google Maps

Debes obtener una clave de API de Google Maps desde la consola de Google Cloud Platform.

Paso 2: Configurar Permisos y Clave de API en AndroidManifest.xml

Agrega los permisos necesarios y la clave de API en tu archivo AndroidManifest.xml.

Paso 3: Configurar el Mapa en la Actividad

Configura y muestra el mapa en tu actividad.

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtener el SupportMapFragment y ser notificado cuando el mapa esté listo para ser usado.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Añadir un marcador en una ubicación y mover la cámara
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

SupportMapFragment: Se utiliza para integrar el fragmento de mapa en la actividad.

OnMapReadyCallback: Implementa la interfaz OnMapReadyCallback para ser notificado cuando el mapa esté listo.

Configurar el Mapa: En el método onMapReady, configura el mapa, añade marcadores y mueve la cámara.

La geolocalización y otros servicios externos enriquecen significativamente las aplicaciones Android, permitiendo funcionalidades avanzadas y mejorando la experiencia del usuario. Utilizando las APIs proporcionadas por Google Play Services y Firebase, los desarrolladores pueden integrar fácilmente geolocalización, mapas, notificaciones push y almacenamiento en la nube en sus aplicaciones, creando soluciones más robustas y dinámicas.

4.4. ACTIVIDADES DE EVALUACIÓN

Para evaluar la comprensión y aplicación de los conceptos aprendidos en la Unidad 4: Persistencia y Comunicación, se pueden implementar una variedad de actividades que incluyan presentaciones, tutoriales, y tareas de programación. A continuación, se describen algunas actividades de evaluación propuestas:

1. Presentación de Proyecto

Objetivo: Evaluar la capacidad del estudiante para integrar y presentar una solución completa utilizando los conocimientos adquiridos sobre persistencia y comunicación en aplicaciones Android.

Descripción:

- Los estudiantes deberán desarrollar una aplicación Android que incluya funcionalidades de persistencia de datos (archivos y bases de datos), comunicación con servicios web, geolocalización, y otras APIs de Google Play Services o Firebase.
- Cada estudiante o grupo de estudiantes presentará su proyecto ante la clase, explicando las características principales, los desafíos encontrados y cómo los resolvieron.
- La presentación debe incluir una demostración en vivo de la aplicación.

Criterios de Evaluación:

- Complejidad y funcionalidad de la aplicación.
- Claridad y efectividad de la presentación.
- Innovación y creatividad en la solución propuesta.
- Resolución de problemas y manejo de desafíos técnicos.

2. Tutorial Práctico

Objetivo: Evaluar la capacidad del estudiante para enseñar y comunicar conceptos técnicos mediante la creación de un tutorial paso a paso.

Descripción:

- Los estudiantes deberán crear un tutorial práctico que explique cómo implementar una funcionalidad específica relacionada con la unidad, como la integración de geolocalización, el uso de Retrofit para consumir una API RESTful, o la implementación de Room para la persistencia de datos.
- El tutorial debe ser claro, detallado y seguir una estructura lógica, incluyendo fragmentos de código, capturas de pantalla, y explicaciones de cada paso.
- Los estudiantes deben entregar el tutorial en formato escrito (documento PDF) y realizar una breve presentación en clase.

Criterios de Evaluación:

- Claridad y organización del tutorial.
- Precisión técnica y corrección del contenido.
- Uso adecuado de ejemplos y explicaciones.
- Habilidad para comunicar conceptos complejos de manera accesible.

3. Tarea de Programación: CRUD con SQLite y Room

Objetivo: Evaluar la capacidad del estudiante para implementar operaciones CRUD utilizando SQLite y Room en una aplicación Android.

Descripción:

- Los estudiantes deberán desarrollar una aplicación Android sencilla que permita realizar operaciones CRUD (Create, Read, Update, Delete) en una base de datos utilizando SQLite y Room.
- La aplicación debe incluir una interfaz de usuario básica que permita al usuario interactuar con la base de datos (por ejemplo, una lista de contactos donde se puedan añadir, ver, editar y eliminar contactos).
- Los estudiantes deberán entregar el código fuente de la aplicación junto con un breve informe que explique la estructura del proyecto y las decisiones de diseño tomadas.

Criterios de Evaluación:

- Correcta implementación de operaciones CRUD.
- Uso adecuado de SQLite y Room.
- Calidad del código y buenas prácticas de programación.

- Funcionalidad y usabilidad de la interfaz de usuario.

4. Proyecto de Integración de Geolocalización y Google Maps

Objetivo: Evaluar la capacidad del estudiante para integrar servicios de geolocalización y Google Maps en una aplicación Android.

Descripción:

- Los estudiantes deberán desarrollar una aplicación Android que muestre la ubicación actual del usuario en un mapa utilizando Google Maps API.
- La aplicación debe incluir características adicionales como la marcación de ubicaciones específicas, la obtención de direcciones entre dos puntos, y la actualización en tiempo real de la ubicación del usuario.
- Los estudiantes deberán entregar el código fuente de la aplicación y realizar una breve presentación en clase demostrando las funcionalidades de la aplicación.

Criterios de Evaluación:

- Correcta integración de Google Maps y servicios de geolocalización.
- Funcionalidad y precisión de las características implementadas.
- Calidad del código y buenas prácticas de programación.
- Innovación y creatividad en las características adicionales.

5. Implementación de Notificaciones Push con Firebase

Objetivo: Evaluar la capacidad del estudiante para integrar y utilizar Firebase Cloud Messaging (FCM) para enviar notificaciones push en una aplicación Android.

Descripción:

- Los estudiantes deberán desarrollar una aplicación Android que reciba notificaciones push enviadas desde Firebase Cloud Messaging.
- La aplicación debe manejar diferentes tipos de notificaciones (por ejemplo, notificaciones de alta prioridad, notificaciones en segundo plano) y mostrar contenido relevante al usuario.

- Los estudiantes deberán configurar Firebase en su proyecto, implementar el servicio de mensajería, y demostrar la recepción de notificaciones en diferentes escenarios.
- Los estudiantes deberán entregar el código fuente de la aplicación y realizar una breve demostración en clase.

Criterios de Evaluación:

- Correcta configuración e integración de Firebase Cloud Messaging.
- Manejo efectivo de diferentes tipos de notificaciones.
- Calidad del código y buenas prácticas de programación.
- Funcionalidad y usabilidad de la aplicación.

Estas actividades de evaluación permiten a los estudiantes demostrar su comprensión y habilidad para aplicar los conceptos aprendidos en la Unidad 4. Al combinar presentaciones, tutoriales y tareas de programación, se fomenta una evaluación integral que abarca tanto la teoría como la práctica. Esto asegura que los estudiantes estén bien preparados para desarrollar aplicaciones Android robustas y eficientes que gestionen la persistencia de datos y la comunicación con servicios externos.

4.5 REFERENCIAS BIBLIOGRÁFICAS

Ahmad, I., Suwarni, E., Borman, R. I., Asmawati, Rossi, F., & Jusman, Y. (2021). Implementation of RESTful API Web Services Architecture in Takeaway Application Development. *2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS)*, 132-137. https://doi.org/10.1109/ICE3IS54102.2021.9649679

Android. (2024a). *Room | Jetpack*. Android Developers. https://developer.android.com/jetpack/androidx/releases/room?hl=es-419

Android. (2024b). *SharedPreferences*. Android Developers. https://developer.android.com/reference/android/content/SharedPreferences

Android. (2024c). *SQLiteOpenHelper*. Android Developers. https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper

Android Developers. (2024). *Cómo obtener datos de Internet*. Android Developers. https://developer.android.com/codelabs/basic-android-kotlin-compose-getting-data-internet?hl=es-419

Castillo-Gomez, A. E. (2022). *Arquitectura limpia con MVVM (Model View Viewmodel) para aplicaciones móviles Android.* [Monografia, Universidad Autónoma del Estado de Quintana Roo]. https://risisbi.uqroo.mx/handle/20.500.12249/3225

Dellepiane Espinoza, S. G., & Díaz Alvarado, H. A. (2016). *Mytraininggoal: Facilitar la adopción de un estilo de vida sano en los clientes del Gimnasio Personal Training, mediante la utilización de un sistema móvil con android, aplicando el concepto de Clean Architecture* [Tesis, Universidad Ricardo Palma]. https://repositorio.urp.edu.pe/handle/20.500.14138/2299

Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Applied Sciences*, *12*(9), Article 9. https://doi.org/10.3390/app12094369

Gaffney, K. P., Prammer, M., Brasfield, L., Hipp, D. R., Kennedy, D., & Patel, J. M. (2022). SQLite: Past, present, and future. *Proceedings of the VLDB Endowment*, *15*(12), 3535-3547. https://doi.org/10.14778/3554821.3554842

Gironés, J. T., & Mauri, J. L. (2022). *El gran libro de Android 9ed* (9.ª ed.). Marcombo.

Godoy, M. L., Lopez, A. del C., & Mariño, S. I. (2024). Implementación de Arquitectura Limpia en una aplicación móvil. Registros de ingresos y egresos de personas. *Memorias de las JAIIO*, *10*(5), Article 5. https://ojs.sadio.org.ar/index.php/JAIIO/article/view/935

Gómez-Jiménez, E. (2022). *Metodología De La Programación por Enrique Gómez Jiménez—9789587788426—Libros Técnicos Universitarios* (1.ª ed.). Alpha Editorial. https://www.alpha-editorial.com/Papel/9789587788426/Metodología+De+La+Programación

González-Díez, M. (2016). *Clean architecture y RxJava en Android* [Tesis]. Universitat de Barcelona.

Hernández, L. M. A., Romero, V. A. P., González, S. A. S., & Rodríguez, J. A. V. (2021). Arquitectura REST para el desarrollo de aplicaciones web empresariales. *Revista Electrónica sobre Tecnología, Educación y Sociedad*, *8*(15), Article 15. https://www.ctes.org.mx/index.php/ctes/article/view/748

O'g'li, M. S. S. (2024). MODEL VIEW PRESENTER (MVP) ARXITEKTURASI. *Ta'limning Zamonaviy Transformatsiyasi*, *8*(4), Article 4. https://pedagoglar.org/index.php/03/article/view/3813

Putra, R. B. D., Budi, E. S., & Kadafi, A. R. (2020). Perbandingan Antara SQLite, Room, dan RBDLiTe Dalam Pembuatan Basis Data pada Aplikasi Android. *JURIKOM (Jurnal Riset Komputer)*, *7*(3), Article 3. https://doi.org/10.30865/jurikom.v7i3.2161

Sraïri, M.-A. (2014). *Utilización de servicios Web en dispositivos móviles Android* [Tesis Doctoral, Universitat Politècnica de València]. http://polipapers.upv.es/index.php/IA/article/view/3293

Universidad de la Amazonia
Vigilada MinEducación

INSTITUCIÓN ACREDITADA EN ALTA CALIDAD