ARQUITECTURA TALLER 2

Integrantes

Martin Zhaohong Chen He
Santiago Mejía
Valentina Hernández Bayona

Grupo: 5

Asignatura

Arquitectura

Profesor

Andrés Sánchez

Contenido

ANÁLISIS DE LAS TECNOLOGÍAS

1. ARQUITECTURA HEXAGONAL

QUÉ ES LA ARQUITECTURA HEXAGONAL

La Arquitectura Hexagonal es un patrón de diseño de software que busca la separación clara de las responsabilidades de cada componente de un sistema. Se visualiza como un hexágono, donde el núcleo del sistema se encuentra en el centro, rodeado de puertos y adaptadores. Los puertos representan las interfaces con el mundo exterior, mientras que los adaptadores conectan estos puertos con el núcleo del sistema.

COMPONENTES

1. Núcleo o dominio de la aplicación (Domain Layer):
Contiene la lógica de negocio y las reglas que definen el comportamiento de la aplicación.
Es independiente de la tecnología subyacente y se comunica con el mundo exterior a través de interfaces definidas.
Define modelos de dominio y reglas de negocio, separándolos de la implementación técnica.
2. Puertos (Ports):
Son interfaces que definen cómo el núcleo de la aplicación se comunica con el mundo exterior.
Independientes de la tecnología subyacente y se implementan en adaptadores.
Divididos en puertos de entrada (para recibir datos) y puertos de salida (para enviar datos).
3. Adaptadores (Adapters):
Implementan los puertos definidos en el dominio, permitiendo que la aplicación se comunique con el mundo exterior.
Transforman los datos en un formato adecuado para su uso por el núcleo de la aplicación.
Se dividen en adaptadores de entrada (para recibir datos) y adaptadores de salida (para enviar datos).
4. Capa de infraestructura (Infrastructure Layer):
Proporciona soporte técnico al resto del sistema, como servicios de mensajería o de autenticación.
Es específica de la tecnología subyacente y puede ser reemplazada sin afectar al resto del sistema.
Contiene adaptadores y otros componentes no relacionados directamente con la lógica de negocio.

VENTAJAS DE LA ARQUITECTURA HEXAGONAL:

- ***Separación de Responsabilidades:*** La arquitectura hexagonal claramente separa las responsabilidades de la lógica de negocio de las relacionadas con la infraestructura. Esto significa que el desarrollo y mantenimiento de la lógica de negocio se realiza de manera independiente de los detalles de implementación técnica.
- ***Facilidad de Prueba:*** La separación entre la lógica de negocio y la infraestructura facilita la escritura de pruebas automatizadas. Las pruebas pueden enfocarse exclusivamente en validar la lógica de negocio sin la necesidad de involucrar la complejidad de la infraestructura, lo que mejora la eficiencia de las pruebas y la calidad del software.
- ***Flexibilidad:*** La arquitectura hexagonal permite una mayor flexibilidad en la elección de tecnologías y herramientas. Dado que la lógica de negocio está desacoplada de la

infraestructura, es posible cambiar la tecnología subyacente sin afectar la lógica de negocio, lo que facilita la adaptación a nuevas tecnologías o requerimientos.

- *Escalabilidad:* La separación clara entre las capas y la utilización de puertos y adaptadores permiten escalar la aplicación en diferentes niveles. Esto significa que se pueden escalar los componentes de la lógica de negocio o la infraestructura de manera independiente, lo que mejora la capacidad de respuesta y el rendimiento del sistema.
- *Mantenibilidad:* La arquitectura hexagonal facilita el mantenimiento del software a lo largo del tiempo. La separación de responsabilidades hace que sea más fácil entender y actualizar diferentes componentes del sistema. Además, la flexibilidad en la elección de tecnologías y herramientas simplifica las tareas de mantenimiento.

DESVENTAJAS DE LA ARQUITECTURA HEXAGONAL:

- *Complejidad:* La arquitectura hexagonal puede ser más compleja que otros enfoques, ya que requiere la creación de múltiples componentes como puertos y adaptadores. La comprensión y mantenimiento de estos componentes adicionales pueden aumentar la complejidad del desarrollo.
- *Sobrecarga de Código:* La implementación de la arquitectura hexagonal puede requerir más código que otros enfoques, lo que puede aumentar la complejidad y dificultar la comprensión del código. La necesidad de definir interfaces y adaptadores puede llevar a una mayor cantidad de código en comparación con enfoques más simples.
- *Dificultades en el Diseño Inicial:* La arquitectura hexagonal puede ser difícil de diseñar inicialmente, ya que requiere una comprensión profunda de la lógica de negocio y la infraestructura. El diseño de puertos y adaptadores que sean flexibles y mantenibles puede requerir un análisis exhaustivo de los requisitos y restricciones del sistema.
- *Potencial de Sobreingeniería:* Si no se diseña e implementa correctamente, la arquitectura hexagonal puede conducir a una sobreingeniería. Esto significa que se puede crear una solución excesivamente compleja y costosa en términos de tiempo y recursos. Es importante equilibrar la complejidad de la arquitectura con los requisitos y objetivos del proyecto para evitar la sobreingeniería.

SITUACIONES/PROBLEMAS DONDE USAR:

- Proyectos complejos que requieren una gestión clara de las responsabilidades y una mayor flexibilidad en la implementación.
- Aplicaciones con reglas de negocio complicadas que necesitan ser separadas de la infraestructura técnica.
- Situaciones donde la escalabilidad futura es una consideración importante.
- Necesidad de realizar pruebas automatizadas eficientes.
- Proyectos que utilizan diversas tecnologías y herramientas.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Eficiencia de Desempeño:*
    - Comportamiento temporal: La Arquitectura Hexagonal puede influir en el comportamiento temporal del sistema al permitir la optimización del tiempo de respuesta mediante la separación de la lógica de negocio y la infraestructura.

- Utilización de recursos: Al definir interfaces claras, la Arquitectura Hexagonal ayuda a gestionar eficientemente los recursos del sistema al evitar el exceso de uso de CPU, memoria y almacenamiento.

2. *Fiabilidad:*
    - Ausencia de fallos: La modularidad inherente a la Arquitectura Hexagonal facilita la identificación y corrección de fallos al aislar componentes individuales, reduciendo así la probabilidad de fallos en cascada.

3. *Mantenibilidad:*
    - Modularidad: La Arquitectura Hexagonal fomenta la modificación efectiva del sistema al permitir cambios en componentes individuales sin afectar a otros, lo que facilita la evolución del sistema con el tiempo.
    - Reusabilidad: Al definir componentes claramente separados, la Arquitectura Hexagonal promueve la reutilización de código en diferentes contextos, mejorando así la mantenibilidad del sistema.

4. *Flexibilidad:*
    - Adaptabilidad: La Arquitectura Hexagonal facilita la adaptación del sistema a cambios en los requisitos o el entorno al permitir la sustitución o actualización de componentes sin afectar al funcionamiento global del sistema.

CASOS DE ESTUDIO

1. *CASO 1 – NETFLIX*
    - Descripción:
      Con el crecimiento de la producción de Originales de Netflix, surgió la necesidad de aplicaciones eficientes para facilitar el proceso creativo. El equipo de Ingeniería de Estudio de Netflix desarrolló diversas aplicaciones para abarcar desde la adquisición de contenido hasta la reproducción final. Sin embargo, la integración de datos desde múltiples fuentes y la necesidad de mantener la flexibilidad en el desarrollo representaban desafíos significativos.

    - Solución:
      Para abordar estos desafíos, se implementó la Arquitectura Hexagonal. Esta arquitectura permitió la integración de datos desde múltiples fuentes sin afectar la lógica de negocio. Se definieron entidades, repositorios y interactores para aislar la lógica central de la aplicación de las preocupaciones externas. Además, se estableció una estrategia de pruebas detallada para garantizar la fiabilidad del sistema en cada etapa del desarrollo.

    - Impacto:
      La implementación de la Arquitectura Hexagonal ha tenido un impacto significativo en Netflix. Ha permitido desarrollar una aplicación altamente integrada y flexible, capaz de adaptarse a los cambios en los sistemas existentes y a las necesidades futuras del negocio. La capacidad de intercambiar fácilmente fuentes de datos sin afectar la lógica de negocio ha facilitado el desarrollo y la evolución continua de la aplicación, mejorando la eficiencia en todo el proceso creativo. Como resultado, Netflix ha logrado una mayor

eficiencia en la producción de contenido original y una mejora en la experiencia del usuario final.

2. POSTMAN

¿QUÉ ES POSTMAN?

Postman es una plataforma integral para la construcción y utilización de APIs. Simplifica cada etapa del ciclo de vida de las APIs y fomenta la colaboración para que puedas crear APIs mejores y más rápidamente.

HISTORIA DE POSTMAN

Postman, una plataforma fundamental para el desarrollo y la gestión de APIs, ha recorrido un viaje notable desde su concepción hasta convertirse en una herramienta esencial para millones de desarrolladores en todo el mundo. La historia de Postman comienza en 2009, cuando durante una pasantía de programación en Yahoo Bangalore, el futuro cofundador de Postman, Ankit Sobti, junto con otro pasante, se encontraron enfrentando repetidamente los mismos desafíos al trabajar con APIs. Esta experiencia les llevó a idear una solución para simplificar estas tareas.

En su primer emprendimiento, TeliportMe, uno de los cofundadores de Postman, Abhijit Kane, se unió al equipo. Trabajando en un API complejo, la necesidad de herramientas efectivas de gestión y prueba de APIs se hizo evidente. Esto llevó al equipo a concebir la idea de Postman.

La primera versión de Postman fue lanzada como un cliente HTTP básico para Chrome, y la respuesta fue abrumadora, con cientos de miles de usuarios adoptando rápidamente la herramienta. Esto impulsó al equipo a considerar seriamente la idea de crear una empresa.

Comprometidos con escuchar a sus usuarios, el equipo se enfocó en recopilar sus comentarios para mejorar constantemente el producto. Esta mentalidad centrada en el usuario ha sido fundamental para el desarrollo de Postman.

A medida que Postman creció, el equipo se centró en tomar decisiones basadas en datos y en mantener una infraestructura tecnológica simple y eficiente. Esto les permitió escalar y expandirse sin comprometer la calidad del producto.

La comunidad de Postman creció orgánicamente, con usuarios que compartían sus experiencias y se convertían en defensores de la plataforma. Esto llevó al equipo a expandir su enfoque más allá de los desarrolladores, reconociendo el potencial de Postman para resolver problemas en diversas áreas.

En 2020, Postman lanzó una versión web completa, cerrando el ciclo desde sus humildes comienzos en Chrome. Este hito marcó una nueva fase en la evolución de Postman.

Hoy en día, con una comunidad de más de 17 millones de desarrolladores y un crecimiento continuo, Postman sigue comprometido con su misión de simplificar el desarrollo y la gestión de APIs para todos.

La historia de Postman, desde una idea hasta una comunidad global, es un testimonio de la importancia de escuchar a los usuarios, tomar decisiones basadas en datos y mantener un enfoque simple y centrado en el usuario en todo momento.

VENTAJAS;

- Interfaz intuitiva y fácil de usar.
- Amplia gama de características y herramientas.
- Soporte para APIs REST y SOAP.
- Comunidad activa de usuarios.

DESVENTAJAS:

- Enfoque limitado en documentación y diseño de marcos.
- Dificultad en la colaboración con flujos de trabajo en equipo.
- Curva de aprendizaje pronunciada para principiantes.
- Problemas para mantener y actualizar la documentación de la API.

SITUACIONES/PROBLEMAS DONDE USAR:

- Desarrollo y pruebas de APIs.
- Colaboración en equipos de desarrollo de APIs.
- Gestión de documentación de APIs.
- Automatización de pruebas y monitoreo de APIs.

ASPECTOS RELEVANTES DE POSTMAN

- Repositorio de API: Permite almacenar, catalogar y colaborar en torno a todos los artefactos de API en una plataforma centralizada.
- Herramientas: Ofrece un conjunto completo de herramientas para acelerar el ciclo de vida de la API, desde diseño y pruebas hasta documentación y compartición.
- Gobernanza: Adopta un enfoque de ciclo de vida completo para la gobernanza de APIs, fomentando la colaboración entre equipos de desarrollo y diseño de APIs.
- Espacios de Trabajo: Facilita la organización del trabajo de API y la colaboración en toda la organización o en todo el mundo.
- Integraciones: Se integra con las herramientas más importantes en el pipeline de desarrollo de software para habilitar prácticas API-first.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Usabilidad – Capacidad de interacción:*
    - Operabilidad: Postman proporciona una interfaz intuitiva y fácil de usar que permite a los usuarios operar y controlar la plataforma con facilidad. Esto se refleja en su capacidad para realizar pruebas, gestionar documentación y colaborar en equipos de desarrollo de APIs.

2. *Mantenibilidad:*
    - Capacidad para ser probado: Postman ofrece una facilidad significativa para establecer criterios de prueba para sistemas o componentes y llevar a cabo las pruebas necesarias para determinar si se cumplen esos criterios. Esto es esencial para el desarrollo y la gestión de APIs, ya que permite a los desarrolladores garantizar la calidad y la fiabilidad de sus APIs mediante pruebas automatizadas.

3. *Flexibilidad:*
    - Adaptabilidad: Postman es altamente adaptable y puede integrarse con una variedad de herramientas y servicios en el pipeline de desarrollo de software. Esto permite a los usuarios adaptar Postman a diferentes entornos y necesidades específicas del proyecto, lo que mejora la flexibilidad en el desarrollo y la gestión de APIs.

4. *Seguridad:*
    - Confidencialidad: Postman asegura que los datos de los usuarios y los detalles de las APIs sean confidenciales y estén protegidos contra accesos no autorizados. Esto es fundamental para mantener la integridad y la seguridad de los datos sensibles utilizados en el desarrollo y la gestión de APIs.

CASOS DE ESTUDIO

1. *CASO 1: WHATSAPP*
    - Desafío:
      El equipo de desarrollo de WhatsApp enfrentaba el desafío de garantizar una rápida integración de nuevas APIs con el fin de reducir los tiempos de incorporación y minimizar la complejidad para los desarrolladores. Sin un proceso de incorporación eficiente, el inicio de proyectos podría demorarse considerablemente, afectando el desarrollo y la implementación de la API en la nube.

    - Solución:
      WhatsApp decidió asociarse con la Plataforma API de Postman debido a su reconocimiento como líder en la industria y su interfaz accesible. Postman permitió generar solicitudes en varios lenguajes de programación, simplificando así el proceso de integración y minimizando la curva de aprendizaje para los desarrolladores. Esta colaboración también proporcionó un entorno unificado para la gestión de comunicaciones, eliminando la necesidad de utilizar múltiples herramientas.

    - Impacto:
      La colaboración con Postman permitió a WhatsApp agilizar la comunicación con los clientes, reduciendo significativamente el tiempo necesario para iniciar interacciones. Los desarrolladores pudieron integrar WhatsApp en sus stacks tecnológicos existentes y gestionar comunicaciones sin problemas dentro de un entorno centralizado. Esta optimización del flujo de trabajo permitió a WhatsApp ofrecer una experiencia de usuario mejorada y aumentar la eficiencia en el desarrollo y la implementación de la API en la nube.

2. *CASO 2: PAYPAL*

- Desafío:
  PayPal se enfrentaba al desafío de optimizar la experiencia del desarrollador y reducir el tiempo de inicio de llamadas a las APIs, conocido como TTFC (Tiempo para la primera llamada), para facilitar una integración rápida y eficiente. El objetivo era proporcionar una plataforma centralizada que evitara la duplicación de trabajo, fomentara la colaboración y mejorara la documentación y la gobernanza de las APIs, tanto para uso interno como externo.

- Solución:
  PayPal decidió implementar la Plataforma API de Postman, que permitió a sus ingenieros definir, diseñar y probar APIs públicas, privadas y de socios en un único lugar. Postman se convirtió en una herramienta esencial para pruebas de extremo a extremo, solución de problemas, monitoreo y simulación. Esta solución automatizada optimizó los flujos de trabajo de PayPal, proporcionando una única fuente de verdad para todos los activos de API ejecutables, lo que mejoró tanto la experiencia de los consumidores como de los productores de API.

- Impacto:
  La implementación de Postman redujo drásticamente el TTFC de PayPal a solo 1 minuto, lo que permitió a los desarrolladores comenzar a utilizar las APIs de manera casi instantánea. El tiempo de prueba de las APIs se redujo de horas a minutos gracias a las colecciones de Postman, lo que aumentó la productividad de los ingenieros de PayPal. La popularidad de las APIs de PayPal creció exponencialmente, con más de 30,000 forks de la Colección Pública de Postman de PayPal. Esta colaboración también ha permitido a PayPal acceder a millones de desarrolladores a través de la plataforma de Postman, generando una mayor adopción y colaboración en el ecosistema de APIs. Además, la retroalimentación continua de la comunidad de Postman ha ayudado a PayPal a mantener sus artefactos oficiales a un alto nivel de calidad y a comprender mejor las necesidades y preferencias de los desarrolladores.

3. FLUTTER

¿QUÉ ES FLUTTER?

Flutter es un versátil conjunto de herramientas para crear aplicaciones nativas visualmente atractivas en diversas plataformas. Se jacta de ser compatible con bases de código existentes, lo que lo convierte en una opción accesible para desarrolladores de diversos niveles de habilidad. Con su avanzado motor de renderizado, Flutter facilita la realización de experiencias de usuario de alta gama mientras permite la prototiparían y desarrollos rápidos.

¿PARA QUIÉN ES FLUTTER?

Flutter atiende a un amplio espectro de usuarios:

- Para los usuarios, Flutter ofrece experiencias de aplicación cautivadoras y fluidas.

- Para los desarrolladores, simplifica el proceso de desarrollo de aplicaciones, reduciendo costos y complejidades asociadas con el desarrollo multiplataforma.
- Para los diseñadores, proporciona un lienzo para realizar interfaces de usuario intrincadas, fomentando la creatividad e innovación.
- Para los gerentes de ingeniería y empresas, agiliza los esfuerzos de desarrollo de aplicaciones y sincroniza los horarios de lanzamiento en múltiples plataformas.

PRINCIPALES VENTAJAS DE FLUTTER:

- ***Único Código Base:*** Desarrolle aplicaciones para múltiples plataformas desde un solo código base, reduciendo el tiempo de desarrollo y los costos.
- ***Desarrollo Rápido:*** La función de recarga en caliente de Flutter permite cambios de código en tiempo real, facilitando la iteración y prueba rápidas.
- ***Rendimiento Nativo:*** El motor de renderizado de alta performance de Flutter garantiza experiencias de aplicación suaves y receptivas.
- ***Widgets Personalizables:*** Flutter ofrece un rico conjunto de widgets personalizables, permitiendo a los desarrolladores crear interfaces de usuario a medida.

PRINCIPALES DESVENTAJAS DE FLUTTER:

- ***Curva de Aprendizaje:*** Aunque accesible, Flutter puede requerir algún tiempo para que los desarrolladores comprendan sus conceptos y funcionalidades.
- ***Funcionalidades Nativas Limitadas:*** A pesar del amplio soporte de paquetes de terceros, Flutter puede carecer de ciertas funcionalidades o integraciones nativas.
- ***Dependencias de Plataforma:*** La dependencia de Flutter en su motor de renderizado puede plantear desafíos en la integración de funcionalidades o actualizaciones específicas de la plataforma.

SITUACIONES/PROBLEMAS ADECUADOS PARA FLUTTER:

- ***Desarrollo Multiplataforma:*** Proyectos que requieren despliegue simultáneo en múltiples plataformas se benefician del enfoque unificado de Flutter.
- ***UI/UX de Alta Fidelidad:*** Flutter sobresale en la creación de interfaces de usuario visualmente impresionantes e interactivas, ideales para aplicaciones centradas en la marca.
- ***Prototipado Rápido:*** La función de recarga en caliente de Flutter lo convierte en una excelente opción para iterar y probar rápidamente ideas de aplicaciones.

HISTORIA DE FLUTTER

Flutter, un proyecto de código abierto concebido por Google, ha surgido como un destacado kit de desarrollo de software de interfaz de usuario, transformando la manera en que se construyen aplicaciones móviles y web. Desde su introducción en 2015, Flutter ha experimentado un crecimiento notable, alcanzando hitos significativos y expandiendo constantemente sus capacidades.

La génesis de Flutter se remonta a la observación de Google sobre la necesidad de una experiencia de desarrollo unificada para plataformas que abarcan desde dispositivos móviles hasta navegadores web. Esta visión condujo al nacimiento de Flutter en 2015, con el objetivo de permitir a los desarrolladores escribir un único código base que pudiera ejecutarse en Android e iOS, lo que aumentaría la eficiencia y reduciría los costos de desarrollo de aplicaciones.

El verdadero reconocimiento de Flutter llegó en 2017 con el lanzamiento de su primera aplicación comercial. Este evento marcó un punto de inflexión en la industria del desarrollo de aplicaciones móviles, demostrando la capacidad de Flutter para permitir un desarrollo rápido y eficiente en ambas plataformas, iOS y Android. A partir de entonces, Flutter ha seguido evolucionando y mejorando, recibiendo actualizaciones regulares y ampliando su alcance más allá de los dispositivos móviles.

Uno de los momentos más significativos en la historia de Flutter fue el lanzamiento de Flutter Live '18, donde se presentó oficialmente la versión 1.0 de Flutter. Este hito resaltó el rápido crecimiento y la popularidad del marco de desarrollo, así como su potencial para su uso en una variedad de plataformas, más allá de los dispositivos móviles.

En los años siguientes, Flutter continuó expandiéndose, con importantes avances que incluyeron la expansión del marco hacia la web y el escritorio. El proyecto "Hummingbird" se lanzó con el objetivo de llevar aplicaciones completas de Flutter al navegador web, mientras que también se exploraron las posibilidades de crear aplicaciones de escritorio con Flutter.

En eventos como Google I/O '19 y Flutter Engage, se destacaron los avances y las mejoras continuas en Flutter, reforzando su posición como una opción líder para el desarrollo de aplicaciones multiplataforma. La introducción de Flutter 2.0 en Google I/O '21, con un mayor soporte para la web y otras plataformas, marcó otro hito importante en la evolución de Flutter.

Hoy en día, Flutter sigue siendo una opción atractiva para los desarrolladores debido a su capacidad para crear aplicaciones visualmente atractivas y de alto rendimiento en múltiples plataformas. Con un fuerte respaldo de Google, actualizaciones regulares y una creciente comunidad de desarrolladores, el futuro de Flutter parece brillante, con un potencial continuo para la innovación y la expansión hacia nuevas áreas de desarrollo de aplicaciones.

ASPECTOS TECNOLÓGICOS CLAVE DE FLUTTER:

- ***Motor de Renderizado:*** Flutter emplea un motor de renderizado 2D de alto rendimiento para renderizar widgets y elementos de la interfaz de usuario.
- ***Marco de Trabajo:*** Flutter se entrega con un marco de trabajo reactivo moderno, facilitando el desarrollo eficiente de aplicaciones.
- ***Biblioteca de Widgets:*** Flutter proporciona un rico conjunto de widgets personalizables, incluidos widgets de Material Design y Cupertino.
- ***Pruebas y Depuración:*** Flutter ofrece herramientas integrales de pruebas y depuración, asegurando la calidad y confiabilidad de la aplicación.
- ***Lenguaje y Compatibilidad:*** Flutter está construido con Dart, ofreciendo compatibilidad con varias plataformas y dispositivos.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Usabilidad:*
   - Operabilidad: Flutter simplifica el proceso de desarrollo de aplicaciones, reduciendo costos y complejidades asociadas con el desarrollo multiplataforma. Esto hace que la plataforma sea más accesible y fácil de operar para los desarrolladores, lo que mejora su usabilidad.

2. *Mantenibilidad:*
   - Capacidad para ser probado: Flutter ofrece una funcionalidad de pruebas y depuración integradas, lo que facilita la evaluación del impacto de los cambios en el código y la identificación de posibles errores. Esto mejora la mantenibilidad al permitir cambios efectivos y eficientes en el código.

3. *Flexibilidad:*
   - Desarrollo Rápido: La función de recarga en caliente de Flutter permite cambios de código en tiempo real, lo que facilita la iteración y prueba rápidas. Esto proporciona flexibilidad en el desarrollo al permitir ajustes rápidos según sea necesario.

4. *Fiabilidad:*
   - Rendimiento Nativo: El motor de renderizado de alta performance de Flutter garantiza experiencias de aplicación suaves y receptivas. Esto contribuye a la fiabilidad al proporcionar un rendimiento consistente y de alta calidad en diferentes plataformas.

CASOS DE ESTUDIO

1. *Caso 1: Google Ads*

   - Descripción: La aplicación móvil de Google Ads, desarrollada con Flutter, ofrece a los usuarios la capacidad de recibir alertas, hacer seguimiento de campañas publicitarias, realizar modificaciones y obtener resultados desde sus dispositivos móviles. Esta aplicación complementa la versión web de Google Ads, proporcionando a los usuarios una experiencia ágil y conveniente mientras están en movimiento.

   - Beneficios de Flutter:

     - Diseño Atractivo: La aplicación destaca por su excelente diseño, utilizando los principios del Material Design de Google para dispositivos Android y componentes clásicos de iOS, lo que garantiza una apariencia nativa en ambas plataformas.
     - Desempeño Optimizado: Flutter ofrece un rendimiento fluido y rápido, asegurando una experiencia de usuario sin interrupciones, incluso en dispositivos móviles menos potentes.
     - Facilidad de Desarrollo: Los desarrolladores encuentran en Flutter una herramienta amigable y eficiente, lo que les permite implementar nuevas funcionalidades y realizar ajustes con rapidez.

- Reconocimiento en el Portal Web de Flutter: La aplicación de Google Ads se destaca como un ejemplo destacado en el portal web de Flutter, lo que subraya su calidad y eficacia en el desarrollo móvil.

2. *Caso 2: Alibaba*

- Descripción: Alibaba, uno de los mayores portales de comercio electrónico del mundo, eligió Flutter para desarrollar su aplicación móvil. Con millones de usuarios diarios y la necesidad de gestionar numerosas transacciones en tiempo real, Alibaba confió en Flutter debido a su excelente rendimiento y capacidad para manejar cargas de trabajo intensivas.

- Beneficios de Flutter:
  - Rendimiento y Escalabilidad: Flutter ofrece un rendimiento excepcional y escalabilidad, lo que permite a Alibaba manejar eficientemente la gran cantidad de transacciones y usuarios que utilizan su aplicación diariamente.
  - Desarrollo Eficiente: La facilidad de desarrollo de Flutter permitió al equipo de Alibaba crear y mantener su aplicación de manera eficiente, evitando problemas de desempeño que podrían haber surgido con otras tecnologías de desarrollo móvil.
  - Gestión en Tiempo Real: La capacidad de Flutter para manejar datos en tiempo real facilitó a Alibaba gestionar transacciones y actualizaciones en tiempo real, garantizando una experiencia de usuario fluida y sin retrasos.

Alternativa Superior: En comparación con otras tecnologías como React Native, Flutter se destacó como la mejor opción para Alibaba, ofreciendo un rendimiento superior y una experiencia de usuario excepcional para su aplicación móvil.

4. REST/JSON

¿QUÉ ES REST/JSON?

REST/JSON es un estilo arquitectónico para diseñar APIs web que utiliza JSON como formato de intercambio de datos. Este enfoque se basa en los principios de REST (Transferencia de Estado Representacional), que promueve una arquitectura de software orientada a recursos y utiliza métodos estándar de HTTP para manipular estos recursos.

HISTORIA

- ***Orígenes de REST:*** En el año 2000, Roy Fielding propuso Representational State Transfer (REST) en su tesis doctoral como un enfoque para diseñar arquitecturas de sistemas distribuidos en la World Wide Web. REST se basa en los principios fundamentales de la web, como el uso de URIs, HTTP y el manejo de recursos.
- ***Evolución y adopción:*** REST ganó popularidad rápidamente debido a su simplicidad y su enfoque en estándares web existentes. Empresas como Google, Amazon y Twitter comenzaron a adoptar REST para diseñar sus APIs, lo que contribuyó a su rápida expansión.

- ***Aparición de JSON:*** JSON (JavaScript Object Notation) se desarrolló como un formato ligero de intercambio de datos basado en JavaScript. Aunque su uso inicial estaba relacionado con JavaScript, su simplicidad y legibilidad lo hicieron popular más allá del ámbito de JavaScript.
- ***Combinación con REST:*** JSON se convirtió en un formato de elección para el intercambio de datos en aplicaciones web y móviles debido a su facilidad de uso y su capacidad para representar datos de manera estructurada. La combinación de JSON con las prácticas RESTful permitió la creación de APIs más simples y flexibles.
- ***Adopción generalizada:*** La combinación de REST y JSON se convirtió en un estándar de facto para el diseño de APIs web. Su simplicidad y su facilidad de implementación hicieron que fuera ampliamente adoptado por empresas y desarrolladores en todo el mundo.
- ***Continua evolución:*** A medida que las tecnologías web y las necesidades de los desarrolladores evolucionan, REST y JSON continúan adaptándose. Se desarrollan nuevas prácticas y herramientas para optimizar el rendimiento, la seguridad y la escalabilidad de las APIs basadas en REST/JSON.

VENTAJAS DE USAR REST/JSON

- ***Estandarización:*** Proporciona un protocolo claro que elimina inconsistencias en el diseño de la API.
- ***Eficiencia:*** Reduce la cantidad de datos transferidos mediante respuestas estructuradas y claras.
- ***Facilidad de Uso:*** Simplifica el trabajo de los desarrolladores al seguir un estándar uniforme y predecible.
- ***Flexibilidad:*** Permite respuestas detalladas incluyendo relaciones y enlaces.

DESVENTAJAS DE USAR REST/JSON

- ***Complejidad para operaciones complejas***: Para operaciones que requieren transacciones complejas o manipulación de datos atómica, REST/JSON puede resultar menos adecuado en comparación con otros protocolos como GraphQL.
- ***Over-fetching y under-fetching de datos:*** En algunas situaciones, REST/JSON puede conducir a la transferencia de datos innecesarios (over-fetching) o a la necesidad de realizar múltiples solicitudes para obtener toda la información necesaria (under-fetching).
- ***Falta de soporte para operaciones específicas:*** Aunque REST/JSON es muy versátil, puede carecer de soporte nativo para algunas operaciones específicas, como transacciones atómicas o actualizaciones parciales de datos.

COMPARATIVA CON OTROS ESTÁNDARES

REST/JSON se compara favorablemente con otros estándares utilizados en el diseño de APIs web. A diferencia de algunos enfoques más complejos, como SOAP (Simple Object Access Protocol), REST/JSON ofrece una arquitectura simple y liviana que se adapta bien a las necesidades de las aplicaciones web modernas.

SITUACIONES/PROBLEMAS DONDE USAR REST/JSON

- Desarrollo y pruebas de APIs.
- Colaboración en equipos de desarrollo de APIs.
- Gestión de documentación de APIs.
- Automatización de pruebas y monitoreo de APIs.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Compatibilidad:*
   - Coexistencia e interoperabilidad: REST/JSON permite la interacción entre diferentes sistemas y aplicaciones, lo que favorece la coexistencia y la interoperabilidad al utilizar un formato de intercambio de datos ampliamente compatible como JSON.

2. *Capacidad de Interacción:*
   - Operabilidad: REST/JSON facilita la operación y control de la API con facilidad, ya que sigue principios claros y predecibles, lo que mejora la capacidad de interacción de los usuarios y desarrolladores con la API.

3. *Fiabilidad:*
   - Disponibilidad: REST/JSON, al ofrecer respuestas estructuradas y claras, contribuye a la disponibilidad de la API al garantizar que los recursos estén accesibles para su uso cuando sea necesario.

4. *Seguridad:*
   - Confidencialidad e integridad: REST/JSON, al utilizar HTTPS y permitir la implementación de mecanismos de autenticación y autorización, contribuye a garantizar la confidencialidad y la integridad de los datos transferidos a través de la API.

5. WEBHOOK

¿QUÉ ES UN WEBHOOK?

Un webhook es una función de devolución de llamada que utiliza el protocolo HTTP para permitir la comunicación ligera entre dos interfaces de programación de aplicaciones (API) a través de eventos. A diferencia de las API tradicionales, que requieren que la aplicación cliente realice solicitudes periódicas para obtener datos, los webhooks permiten que la aplicación servidor envíe automáticamente datos o notificaciones a la aplicación cliente cuando se produce un evento específico.

HISTORIA DE LOS WEBHOOKS:

El concepto de webhooks se popularizó en 2007 con una publicación en el blog de Jeff Lindsay titulada "Web hooks to revolutionize the web". Desde entonces, los webhooks han sido adoptados por numerosas plataformas y servicios en línea debido a su capacidad para simplificar y automatizar la comunicación entre aplicaciones.

Inicialmente, los webhooks se utilizaron principalmente en el ámbito del desarrollo web y la integración de sistemas. Sin embargo, su versatilidad y eficacia los han llevado a ser ampliamente utilizados en diversas áreas, incluyendo el marketing digital, las ventas en línea, la gestión de infraestructura, entre otros.

VENTAJAS:

- ***Automatización de procesos:*** Permiten automatizar tareas y flujos de trabajo al activar eventos específicos.
- ***Entrega de información en tiempo real:*** Envían datos de manera instantánea cuando se produce un evento, asegurando que la información sea recibida de manera oportuna.
- ***Aumento de la eficiencia:*** Eliminan la necesidad de intervención manual y reducen los tiempos de espera para la transferencia de datos.
- ***Facilidad de configuración:*** Requieren una configuración mínima y utilizan HTTP para la transferencia de datos, lo que facilita su implementación.

DESVENTAJAS:

- ***Compatibilidad limitada:*** No todas las aplicaciones admiten webhooks, lo que puede limitar su utilidad en ciertos entornos.
- ***Menos funcionalidad que las API:*** Los webhooks solo permiten la comunicación unidireccional, lo que puede ser insuficiente para integraciones complejas que requieren comunicación bidireccional.
- ***Posibilidad de pérdida de datos:*** En caso de fallos o errores, los webhooks pueden perder datos si no se implementan mecanismos de reintentos o alertas.

SITUACIONES/PROBLEMAS DONDE USAR:

- ***Marketing digital:*** Para activar eventos en campañas de marketing y automatizar procesos de seguimiento y segmentación de clientes.
- ***Gestión de infraestructura:*** Para automatizar la implementación y gestión de la infraestructura mediante prácticas de GitOps e IaC.
- ***Automatización de procesos empresariales:*** Para integrar aplicaciones internas y centralizar la información, mejorando la eficiencia operativa.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Capacidad de Interacción:*
   - Automatización de procesos: Los Webhooks permiten la automatización de tareas y flujos de trabajo al activar eventos específicos, lo que mejora la capacidad de interacción al facilitar la comunicación entre sistemas y la ejecución de acciones automatizadas.

2. *Fiabilidad:*
   - Entrega de información en tiempo real: Los Webhooks envían datos de manera instantánea cuando se produce un evento, lo que garantiza la entrega oportuna de la información y contribuye a la fiabilidad de la comunicación entre sistemas.

3. *Eficiencia de Desempeño:*
    - Aumento de la eficiencia: Al eliminar la necesidad de intervención manual y reducir los tiempos de espera para la transferencia de datos, los Webhooks aumentan la eficiencia en la comunicación entre sistemas, lo que mejora el rendimiento general del sistema.

4. *Mantenibilidad:*
    - Facilidad de configuración: Los Webhooks requieren una configuración mínima y utilizan HTTP para la transferencia de datos, lo que facilita su implementación y mantenimiento, lo que contribuye a la mantenibilidad del sistema al reducir la complejidad y el esfuerzo requerido para su uso.

6. FLASK/PYTHON

¿QUÉ ES FLASK?

Flask es un marco web ligero desarrollado en Python, adecuado tanto para principiantes como para profesionales. Flask se limita a incluir solo lo esencial, pero los usuarios pueden implementar bibliotecas externas para expandir su funcionalidad.

HISTORIA:

Flask fue lanzado por el desarrollador austriaco Armin Ronacher en 2004 como una alternativa minimalista a otros marcos web más completos como Django. Ronacher diseñó Flask con la idea de ofrecer flexibilidad y simplicidad, permitiendo a los desarrolladores crear aplicaciones web de manera rápida y eficiente.

A lo largo de los años, Flask ha ganado una gran comunidad de seguidores debido a su facilidad de uso y su enfoque modular. La filosofía de "hacer una cosa y hacerla bien" ha resonado con muchos desarrolladores que prefieren un marco web que no imponga demasiadas restricciones y permita la máxima libertad creativa.

VENTAJAS DE FLASK:

- ***Alcance:*** Flask es extremadamente ligero y rápido de instalar, lo que lo hace ideal para proyectos pequeños y medianos.
- ***Flexibilidad:*** Flask ofrece una flexibilidad excepcional, permitiendo a los desarrolladores resolver problemas y utilizar las bibliotecas que necesiten de manera individualizada.
- ***Curva de aprendizaje:*** Flask es rápido de aprender gracias a su diseño simple, lo que lo convierte en una excelente opción tanto para principiantes como para profesionales.
- ***Código abierto:*** Flask es de código abierto y está disponible de forma gratuita, lo que permite a los desarrolladores probarlo y decidir si es la herramienta adecuada para sus necesidades.
- ***Comunidad:*** Flask cuenta con una gran comunidad que brinda consejos y soporte tanto a nuevos como a desarrolladores más experimentados.

DESVENTAJAS:

- *Alcance:* Dependiendo del uso previsto, el alcance minimalista de Flask puede ser una desventaja, ya que todas las herramientas requieren instalación individual.
- *Dependencia de proveedores externos:* El uso de bibliotecas externas siempre es una posible fuente de errores, y Flask depende de ellas.
- *Mantenimiento:* Mientras que otros marcos se mantienen automáticamente, Flask traslada esa responsabilidad al usuario, lo que significa más control, pero también más trabajo.

SITUACIONES DE USO

- Desarrollo de aplicaciones web simples y rápidas.
- Prototipado rápido de ideas de proyectos.
- Proyectos que requieren flexibilidad y libertad para seleccionar las bibliotecas y herramientas específicas que se necesiten.
- Desarrollo de API RESTful.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Compatibilidad:*
    - Coexistencia: Flask tiene la capacidad de coexistir con otras bibliotecas y herramientas en un entorno común, compartiendo recursos sin detrimento.
    - Interoperabilidad: Flask puede interoperar con otros sistemas o componentes al permitir el intercambio de información a través de APIs y servicios web.

2. *Fiabilidad:*
    - Ausencia de Fallos: Flask es conocido por su estabilidad y confiabilidad, permitiendo que los sistemas desarrollados con él funcionen sin fallos bajo condiciones normales de operación.
    - Disponibilidad: Flask tiene la capacidad de estar operativo y accesible para su uso cuando se requiere, asegurando que las aplicaciones web desarrolladas con él estén disponibles para los usuarios.

3. *Seguridad:*
    - Confidencialidad: Flask protege la información confidencial asegurando que solo los usuarios autorizados puedan acceder a ella, garantizando la privacidad de los datos.

CASOS DE ESTUDIO

1. *Descripción:*
   Pinterest y LinkedIn son dos plataformas sociales líderes que utilizan Flask como parte de su infraestructura de desarrollo. Pinterest es una plataforma de redes sociales y búsqueda visual que permite a los usuarios descubrir y guardar ideas para proyectos, hobbies e intereses. LinkedIn, por otro lado, es una plataforma de redes sociales profesionales que facilita la

conexión entre colegas, la búsqueda de oportunidades laborales y el aprendizaje de nuevas habilidades.

2. *Solución:*
   Tanto Pinterest como LinkedIn han optado por utilizar Flask, un framework de desarrollo web ligero y flexible en Python. Flask proporciona a estas plataformas la capacidad de desarrollar aplicaciones web robustas y escalables. Con Flask, ambas empresas pueden implementar rápidamente características y funcionalidades nuevas, mientras mantienen un rendimiento óptimo y una experiencia de usuario fluida.

3. *Impacto:*
   La elección de Flask ha tenido un impacto significativo en Pinterest y LinkedIn. Les ha permitido desarrollar y mantener plataformas sociales altamente eficientes y confiables. La flexibilidad de Flask ha facilitado la adaptación de estas plataformas a medida que han crecido y evolucionado con el tiempo. En última instancia, Flask ha contribuido al éxito continuo de Pinterest y LinkedIn al proporcionar una base sólida para su desarrollo y crecimiento futuro.

7. ORACLE

¿QUÉ ES ORACLE DATABASE?

Oracle Database es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado y comercializado por Oracle Corporation. Es uno de los sistemas de gestión de bases de datos más populares y ampliamente utilizados en el mundo empresarial. Oracle Database permite a las empresas almacenar, gestionar y recuperar grandes volúmenes de datos de manera eficiente y segura. Utiliza el lenguaje de consulta estructurado (SQL) como estándar para interactuar con la base de datos y ofrece un conjunto de herramientas y funciones avanzadas para el desarrollo y la administración de bases de datos.

HISTORIA

La historia de Oracle Database abarca más de 35 años de desarrollo innovador, con hitos importantes en su evolución:

- ***Fundación de Oracle Corporation:*** En 1977, Larry Ellison, Bob Miner y Ed Oates fundaron Software Development Laboratories, que más tarde se convirtió en Oracle Corporation.
- ***Primer RDBMS disponible comercialmente:*** En 1979, Oracle lanzó Oracle V2, el primer sistema de gestión de bases de datos relacional (RDBMS) disponible comercialmente basado en SQL, marcando un hito en la historia de las bases de datos relacionales.
- ***Versión portátil de Oracle Database:*** En 1983, Oracle Version 3 se convirtió en la primera base de datos relacional en ejecutarse en mainframes, minicomputadoras y computadoras personales, gracias a estar escrita en C, lo que permitió su portabilidad a múltiples plataformas.
- ***Mejoras en control de concurrencia, distribución de datos y escalabilidad:*** Versiones posteriores introdujeron avances significativos, como la consistencia de lectura multiversión, el soporte para computación cliente/servidor y sistemas de bases de datos distribuidas, mejoras en E/S de disco, bloqueo de filas, escalabilidad y recuperación de respaldo, así como la introducción de PL/SQL.
- ***Almacenamiento de programas PL/SQL:*** Oracle7, lanzado en 1992, introdujo procedimientos almacenados y desencadenadores en PL/SQL.

- *Objetos y particionado:* Oracle8, lanzado en 1997, fue una base de datos objeto-relacional que admitía nuevos tipos de datos y el particionado de tablas grandes.
- *Computación en internet:* Oracle8i Database, lanzado en 1999, proporcionó soporte nativo para protocolos de internet y soporte para Java en el lado del servidor, diseñado específicamente para la computación en internet.
- *Oracle Real Application Clusters (Oracle RAC*): Oracle9i Database introdujo Oracle RAC en 2001, permitiendo que múltiples instancias accedan simultáneamente a una sola base de datos.
- *Computación en grid:* Oracle Database 10g, lanzado en 2003, introdujo la computación en grid, permitiendo a las organizaciones virtualizar recursos informáticos y hacer que la base de datos sea autogestionada y autoajustable.
- *Adaptabilidad y automatización:* Oracle Database 11g, lanzado en 2007, introdujo nuevas características que simplificaron la infraestructura de información mediante la consolidación de información y la automatización.
- *Conexión con la nube:* Oracle Database 12c, lanzado en 2013, fue diseñado para la nube, presentando una nueva arquitectura multitenant, almacenamiento en memoria y soporte para documentos JSON.
- *Rendimiento de integración y memoria:* Oracle Database 18c simplificó la integración con servicios de directorio y mejoró el rendimiento de la memoria.
- *Estabilidad mejorada:* Oracle Database 19c, lanzado en 2019, se enfocó en la estabilidad y presentó mejoras significativas en características como JSON y Active Data Guard.
- *Oracle Database 21c: Lanzada* en 2021, Oracle Database 21c es la versión de innovación, con características como la compatibilidad con JSON nativo y mejoras en el rendimiento y la gestión.
- *Oracle Database 23c:* La próxima versión de soporte a largo plazo de Oracle Database, lanzada en 2023, presenta una suma de todas las características de la versión de innovación 21c y más de 300 nuevas características y mejoras. Se centra en áreas clave como JSON, gráficos, microservicios y productividad para desarrolladores.

VENTAJAS:

- *Alta compatibilidad:* Oracle Database es compatible con una amplia gama de plataformas y aplicaciones, lo que facilita su integración en diversos entornos informáticos.
- *Soporte de grandes fabricantes:* Es respaldado por importantes fabricantes de software y hardware, lo que garantiza un alto nivel de soporte y desarrollo continuo.
- *Diversidad de ediciones:* Desde la versión gratuita hasta la Enterprise Edition, Oracle Database ofrece diferentes ediciones para adaptarse a las necesidades de empresas de todos los tamaños.
- *Popularidad entre empresas:* Es ampliamente utilizado en empresas de todo el mundo debido a su fiabilidad y rendimiento.
- *Opción de bases de datos en la nube:* Oracle ofrece servicios en la nube que permiten externalizar y automatizar la gestión de bases de datos, lo que optimiza el uso de recursos y simplifica la administración.
- *Gran comunidad y soporte:* Cuenta con una gran comunidad de desarrolladores y un soporte de calidad por parte de Oracle, lo que facilita la resolución de problemas y la adopción de mejores prácticas.

- *Funciones de seguridad avanzadas:* Ofrece funciones robustas de protección de datos y seguridad, como autenticación y autorización estrictas, cifrado de datos y redes, garantizando la integridad y confidencialidad de la información.

DESVENTAJAS:

- *Requisitos de conocimientos:* Para utilizar la versión de entorno local de Oracle Database, se requiere un amplio conocimiento de SQL y experiencia administrativa en la gestión de bases de datos, lo que puede representar una barrera para algunos usuarios.
- *Costo de licencias:* Las licencias de Oracle Database pueden ser costosas, especialmente las ediciones de nivel empresarial, lo que puede ser una limitación para empresas con presupuestos ajustados.
- *Exigencias de hardware:* La versión de entorno local de Oracle Database puede tener altas exigencias de hardware, lo que puede requerir inversiones adicionales en infraestructura.

SITUACIONES/PROBLEMAS DONDE USAR:

- *Empresas de todos los tamaños:* Oracle Database es adecuado para empresas de todos los tamaños, desde pequeñas y medianas empresas hasta grandes corporaciones, gracias a sus diversas ediciones y capacidades de escalabilidad.
- *Aplicaciones empresariales críticas:* Se utiliza en entornos donde se requiere un alto rendimiento, confiabilidad y seguridad para aplicaciones empresariales críticas, como sistemas de gestión de recursos empresariales (ERP), sistemas de gestión de relaciones con clientes (CRM) y sistemas de análisis de datos.
- *Entornos de desarrollo:* Es una opción popular para entornos de desarrollo debido a su compatibilidad con una amplia gama de lenguajes de programación y herramientas de desarrollo, así como a su robusto conjunto de funciones y capacidades de gestión de bases de datos.
- *Proyectos de almacenamiento de datos grandes:* Se utiliza en proyectos que requieren el almacenamiento y procesamiento de grandes volúmenes de datos, como data warehouses y sistemas de análisis de big data.

ATRIBUTOS DE CALIDAD ASOCIADOS

1. *Fiabilidad:*
   - Oracle Database se destaca por su fiabilidad y estabilidad, lo que asegura que el sistema pueda desempeñar sus funciones especificadas durante largos periodos de tiempo sin interrupciones ni fallos. Además, cuenta con características de tolerancia a fallos y capacidad de recuperación para garantizar la disponibilidad continua de los datos.

2. *Seguridad:*
   - La seguridad es una prioridad en Oracle Database, que ofrece funciones avanzadas para proteger la confidencialidad, integridad y disponibilidad de los datos almacenados en la base de datos. Esto incluye autenticación y autorización estrictas, cifrado de datos, control de acceso basado en roles y auditoría de actividades.

3. *Mantenibilidad:*

- Oracle Database está diseñado para ser fácilmente mantenible, lo que permite a los administradores de bases de datos realizar modificaciones efectivas y eficientes en la estructura y configuración de la base de datos. Esto incluye la capacidad de realizar actualizaciones, parches y ajustes de configuración sin interrupciones en el servicio.

4. *Compatibilidad:*
   - Oracle Database ofrece una alta compatibilidad con una amplia gama de plataformas y aplicaciones, lo que facilita su integración en entornos heterogéneos. Esto garantiza que la base de datos pueda intercambiar información de manera efectiva con otros sistemas y aplicaciones dentro del entorno empresarial.

5. *Capacidad de Interacción:*
   - Oracle Database proporciona interfaces de usuario intuitivas y herramientas de administración que facilitan la interacción con la base de datos. Esto incluye capacidades de administración remota, monitoreo en tiempo real y generación de informes personalizados para satisfacer las necesidades de los usuarios y administradores de bases de datos.

CASOS DE ESTUDIO

- Problema:
  Royal Flying Doctor Service (RFDS), una organización médica que brinda atención en áreas rurales y remotas de Australia, enfrentaba desafíos significativos en la gestión de datos y la prestación de servicios de salud. Con más de 380,000 interacciones de pacientes al año, RFDS operaba en entornos con conectividad limitada o nula a Internet, lo que dificultaba el acceso a información crucial para la atención médica. Históricamente, el personal registraba los datos de los pacientes en formularios en papel, lo que resultaba en una entrada manual tediosa y en la falta de coordinación entre las bases de datos de las siete entidades operativas de RFDS. Esta falta de conectividad y eficiencia comprometía la calidad de los servicios médicos prestados.

- Solución:
  RFDS optó por una solución en la nube que le permitiera establecer un sistema de registro de pacientes nacional que facilitara el intercambio de información en tiempo real entre todas sus operaciones. Después de evaluar varias opciones, RFDS eligió Oracle Cloud Infrastructure (OCI) debido a su experiencia en soluciones de salud y su capacidad para resolver problemas de conectividad remota. Implementaron Oracle Autonomous Database para procesar transacciones y cargas de trabajo mixtas, junto con OCI GoldenGate para replicar datos de diversas fuentes dentro de la organización. Esta arquitectura permitió que los datos capturados offline se sincronizaran de manera confiable con otros sistemas de salud cuando se restablecía la conectividad.

- Resultados:
  La implementación de la solución de Oracle permitió a RFDS mejorar significativamente la eficiencia y la calidad de sus servicios de salud en entornos remotos. Ahora, los equipos médicos pueden acceder a información médica crítica en tiempo real, incluso en áreas sin conectividad a Internet. La eliminación casi total de los formularios en papel ha simplificado la recopilación y el acceso a los datos de los pacientes. Además, la capacidad de ajuste automático de Oracle Autonomous Database ha reducido los tiempos de

computación máximos de 12 horas diarias a 4 horas o menos, mejorando la velocidad y la eficiencia del procesamiento de datos. En general, RFDS ha logrado una arquitectura de datos tolerante a fallos y sin pérdida de datos que ha modernizado sus operaciones, mejorando la atención médica en las zonas más remotas de Australia.

RELACIONES ENTRE LAS TECNOLOGÍAS IMPLEMENTADAS

En el contexto del desarrollo de aplicaciones móviles, Flutter emerge como un framework de código abierto desarrollado por Google, que permite la creación de aplicaciones nativas para múltiples plataformas, incluyendo Android y iOS, desde una única base de código. Los desarrolladores encuentran en herramientas como Postman un aliado invaluable para probar, depurar y trabajar con APIs, que son esenciales para la interacción de las aplicaciones con los servicios web y la obtención de datos.

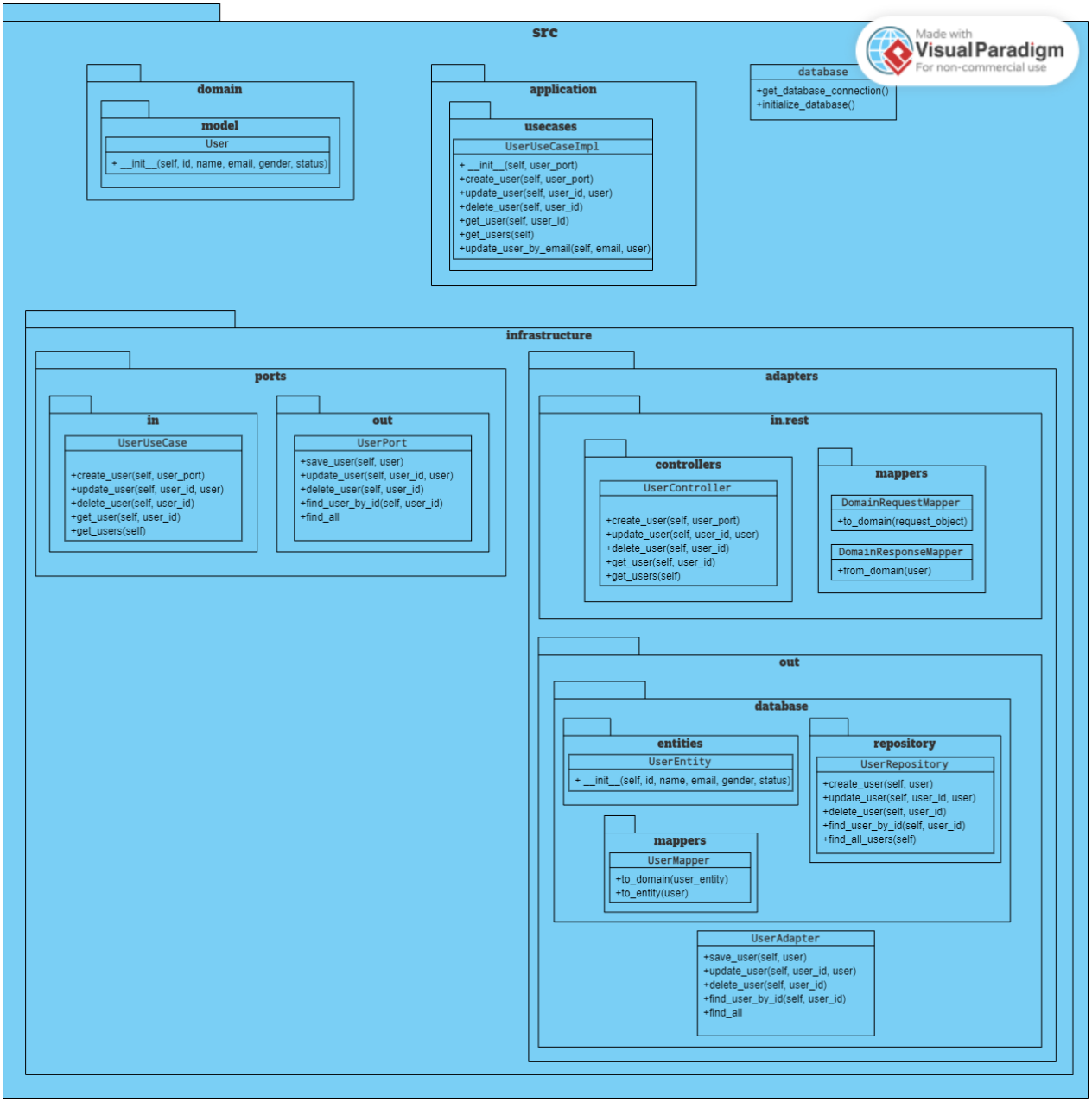
Los WebHooks se presentan como una solución para facilitar la comunicación en tiempo real entre aplicaciones. En el desarrollo de aplicaciones móviles con Flutter, los WebHooks pueden emplearse para notificar eventos importantes a la aplicación, como actualizaciones de datos en un servidor, lo que permite mantener la aplicación móvil sincronizada con la información más reciente de manera eficiente.

La comunicación entre la aplicación móvil y los servicios web se lleva a cabo típicamente mediante el uso del estilo arquitectónico REST (Transferencia de Estado Representacional), complementado con el formato de intercambio de datos JSON (Notación de Objetos de JavaScript). Los servicios web, a menudo construidos con frameworks como Flask en Python, proporcionan datos y funcionalidades específicas a las aplicaciones móviles a través de APIs RESTful.

Python, como lenguaje de programación de alto nivel, juega un papel fundamental en el desarrollo web y móvil debido a su versatilidad y facilidad de uso. En el contexto del desarrollo de aplicaciones móviles con Flutter, Python se utiliza tanto para la construcción de servicios web como para la interacción con bases de datos, como Oracle. Esto implica la utilización de bibliotecas y herramientas específicas de Python para conectarse, almacenar, recuperar y manipular datos importantes utilizados por la aplicación móvil.

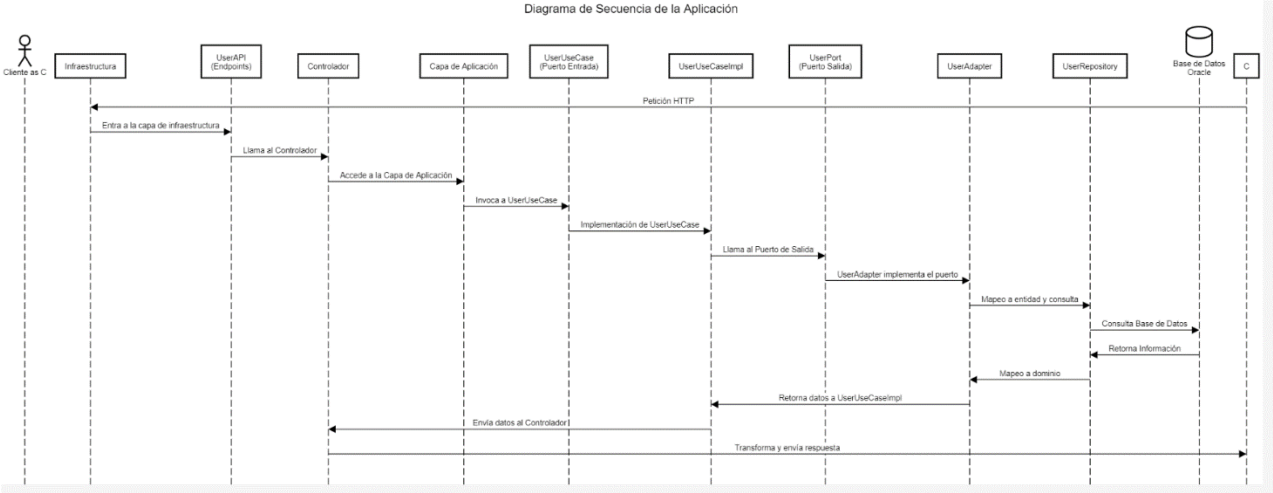
En resumen, todas estas tecnologías están intrínsecamente relacionadas en el proceso de desarrollo de aplicaciones móviles con Flutter, desde la comunicación con servicios web y APIs, hasta el almacenamiento y manipulación de datos en bases de datos. Cada una desempeña un papel crucial en diferentes etapas del ciclo de vida de desarrollo, colaborando para lograr el funcionamiento completo y eficiente de una aplicación móvil.

DIAGRAMA DE CLASES



Link para visualizarlo de mejor manera (Toca hacer login en Visual Paradigm Online)

DIAGRAMA DE SECUENCIA



[Link para visualizarlo de mejor manera]

PRINCIPIOS SOLID DE LA ARQUITECTURA HEXAGONAL

Single Responsibility Principle (SRP):

- En una arquitectura hexagonal, la separación entre la lógica de negocio y la infraestructura ayuda a asegurar que cada componente tenga una única responsabilidad. Por ejemplo, los adaptadores de la interfaz de usuario se centran exclusivamente en la interacción con el usuario, mientras que la lógica de negocio se concentra en las reglas y operaciones del dominio. Esto cumple con el SRP al mantener las responsabilidades bien definidas y separadas.

Open/Closed Principle (OCP):

- La arquitectura hexagonal permite que la aplicación sea extensible sin necesidad de modificar el código existente. Por ejemplo, si se desea añadir una nueva forma de interactuar con la aplicación (como un nuevo tipo de interfaz de usuario o un nuevo sistema de persistencia), se puede hacer implementando nuevos adaptadores para los puertos correspondientes sin modificar la lógica de negocio central ni otros adaptadores. Esto se alinea con el OCP, facilitando la extensión de la funcionalidad de la aplicación sin alterar su núcleo.

Liskov Substitution Principle (LSP):

- Los puertos definidos en una arquitectura hexagonal actúan como contratos entre la lógica de negocio y los adaptadores. Cualquier adaptador que cumpla con el contrato de un puerto puede ser intercambiado con otro sin afectar el funcionamiento de la aplicación. Esto significa que los adaptadores pueden ser sustituidos por sus subtipos de manera transparente, lo cual es una manifestación directa del LSP.

Interface Segregation Principle (ISP):

- La arquitectura hexagonal fomenta la creación de interfaces específicas (puertos) que los adaptadores deben implementar. Esto significa que los componentes de la aplicación no se ven obligados a depender de interfaces que contengan métodos que no utilizan. Cada puerto define exactamente las operaciones necesarias que un adaptador debe realizar, siguiendo así el ISP.

Dependency Inversion Principle (DIP):

- En la arquitectura hexagonal, tanto la lógica de negocio (alto nivel) como los adaptadores (bajo nivel) dependen de abstracciones (los puertos). Los adaptadores implementan estas abstracciones para interactuar con la lógica de negocio, invirtiendo la dependencia tradicional de que la lógica de negocio conozca los detalles de los mecanismos de entrada/salida o persistencia. Esto cumple con el DIP al enfocar las dependencias en las abstracciones en lugar de en los detalles concretos.

REFERENCIAS

1. https://docs.flutter.dev/resources/faq
2. https://www.postman.com/product/what-is-postman/
3. https://www.techaheadcorp.com/knowledge-center/history-of-flutter/#:~:text=It%20was%20first%20introduced%20in,macOS%2C%20Windows%2C%20and%20more.
4. https://www.redhat.com/es/topics/automation/what-is-a-webhook
5. https://www.ibm.com/docs/es/integration-bus/9.0.0?topic=SSMKHH_9.0.0/com.ibm.etools.mft.samples.jsonrest.doc/doc/introduction.html
6. https://jsonapi.org/
7. https://www.ionos.com/digitalguide/websites/web-development/flask-python/
8. https://flask.palletsprojects.com/en/3.0.x/
9. https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/introduction-to-oracle-database.html#GUID-43F9DD5C-8D8C-4E61-A2B4-5C05907D3CEC
10. https://docs.oracle.com/en/database/oracle/oracle-database/23/nfcoa/#Oracle%C2%AE-Database
11. https://blog.hubspot.es/website/que-es-arquitectura-hexagonal
12. https://www.happycoders.eu/software-craftsmanship/hexagonal-architecture/
13. https://tsh.io/blog/hexagonal-architecture/