Arquitectura de Software
Taller #2

Realizado por:
Miguel Uribe,
Juan Buitrago,
Frank Hernández.

Pontificia Universidad Javeriana
23-09-24

Tabla de Contenido

Temas Asignados

I. Arquitectura en Capas

Definición

- La Arquitectura por Capas es un patrón de diseño de software que organiza un sistema en capas jerárquicas, donde cada capa tiene una responsabilidad clara y específica. Cada capa interactúa únicamente con las capas adyacentes: una capa superior solicita servicios de la capa inmediatamente inferior, y una capa inferior devuelve los resultados a la superior. El objetivo principal de este enfoque es separar las preocupaciones, permitiendo que las distintas funcionalidades estén agrupadas en módulos que son más fáciles de gestionar, mantener y escalar.

Características

- **Modularidad:** Cada capa tiene una responsabilidad claramente definida y opera de manera independiente. Esto permite una mayor facilidad en el desarrollo y mantenimiento, ya que es posible hacer cambios en una capa sin afectar las otras.
- **Abstracción:** La arquitectura por capas promueve la separación de la implementación y la interfaz. Las capas superiores no necesitan conocer los detalles internos de las capas inferiores, solo se comunican a través de interfaces públicas.

- **Flexibilidad:** Cambios en una capa específica se pueden realizar sin afectar las demás, lo que permite sustituir tecnologías o modificar funcionalidades de forma relativamente sencilla.
- **Encapsulamiento:** Las capas ocultan sus detalles internos a las demás, proporcionando una interfaz clara para la interacción con otras capas.
- **Reutilización de componentes:** Las capas, al estar separadas, pueden reutilizarse en otros proyectos o aplicaciones si se diseñan de manera adecuada.
- **Interacción jerárquica:** Las capas superiores interactúan con las capas inferiores a través de APIs o interfaces públicas. La comunicación generalmente sigue un flujo lineal descendente.
- **Escalabilidad:** Se puede mejorar la escalabilidad del sistema modificando una capa sin necesidad de rediseñar toda la aplicación.

Historia y evolución

- El patrón de arquitectura por capas tiene su origen en la organización de sistemas complejos en áreas como las telecomunicaciones y la informática. La separación en capas se utilizaba en los sistemas operativos para gestionar las distintas funciones, como el manejo de la memoria, el procesamiento de tareas y la gestión de dispositivos.
- En la década de 1970, la arquitectura de red de protocolo OSI (Open Systems Interconnection) sentó las bases para la adopción de modelos por capas en el diseño de software. Este modelo consistía en 7 capas que gestionaban diferentes aspectos de la comunicación en redes, desde el nivel físico hasta la capa de aplicación.
- Durante los años 80 y 90, con la evolución del software empresarial, la arquitectura por capas se integró más en el desarrollo de aplicaciones corporativas y distribuidas. Empresas grandes comenzaron a utilizar este patrón para organizar sus sistemas, especialmente en el diseño de sistemas de bases de datos y aplicaciones web.
- A medida que el software evolucionó hacia aplicaciones distribuidas y sistemas en la nube, la arquitectura por capas se adaptó para formar la base de muchas arquitecturas modernas, como los microservicios y arquitecturas orientadas a servicios (SOA). Hoy en día, la arquitectura por capas sigue siendo un patrón común, especialmente en el desarrollo de sistemas empresariales y aplicaciones web.

Ventajas y desventajas

- **Ventajas:**
    - **Mantenibilidad:** Al dividir la funcionalidad del sistema en módulos separados, el código se vuelve más fácil de entender y mantener. Si una capa necesita actualizarse o corregirse, los cambios se pueden realizar sin interferir con otras capas.

    - **Escalabilidad:** Es posible escalar individualmente las capas, lo que significa que, si la lógica de negocio o la capa de datos requieren más recursos, se pueden optimizar de manera aislada.
    - **Reutilización de código:** Las capas pueden diseñarse para ser reutilizables en otros proyectos, especialmente si las interfaces están bien definidas. Por ejemplo, una capa de datos diseñada correctamente podría usarse en múltiples aplicaciones que acceden a la misma base de datos.
    - **Separación de responsabilidades:** La división clara de responsabilidades entre las capas mejora la organización del proyecto y permite que diferentes equipos se centren en aspectos específicos del desarrollo.
    - **Seguridad:** Se pueden agregar capas de seguridad específicas que protejan la interacción entre diferentes capas, garantizando el aislamiento de datos sensibles.
- **Desventajas:**
    - **Sobrecomplejidad:** Si se utilizan demasiadas capas, la arquitectura puede volverse demasiado compleja, haciendo que el mantenimiento y la comprensión del sistema sea más difícil de gestionar.
    - **Impacto en el rendimiento:** Cada capa añade un nivel de procesamiento adicional, lo que puede aumentar la latencia en el sistema. En sistemas con muchos niveles de comunicación, esto puede ser un problema.
    - **Rigidez:** Al estar limitado a la interacción entre capas adyacentes, puede ser difícil implementar ciertas funcionalidades que requieran comunicación directa entre capas no contiguas.
    - **Tiempo de desarrollo inicial:** La creación y organización de las capas requiere un diseño cuidadoso, lo que puede alargar el tiempo de desarrollo en las primeras fases del proyecto.

Casos de Uso

- **Aplicaciones empresariales grandes:** Por ejemplo, un sistema de planificación de recursos empresariales (ERP), donde hay una clara necesidad de separar la lógica de negocio, la interfaz de usuario y el acceso a datos. Estas aplicaciones suelen tener módulos que se pueden distribuir en diferentes capas para mantener el código organizado y evitar la duplicación de funcionalidades.
- **Aplicaciones de e-commerce:** En plataformas de comercio electrónico, es necesario gestionar múltiples capas como la interfaz de usuario (catálogo de productos, carritos de compras), lógica de negocio (gestión de inventario, precios dinámicos) y acceso a datos (bases de datos de usuarios, productos y transacciones).
- **Sistemas de información bancaria:** Donde las capas de seguridad, lógica de negocio y acceso a datos necesitan estar fuertemente definidas para garantizar que los datos financieros sensibles estén protegidos y que los sistemas puedan escalar según sea necesario.

- **Aplicaciones de atención médica:** Estos sistemas suelen requerir una arquitectura por capas para separar la lógica de negocio, como la gestión de registros médicos electrónicos (EMR), de la capa de datos, que almacena la información del paciente, y de la capa de presentación que interactúa con los profesionales de la salud.

Casos de aplicación

- **Amazon:** Utiliza una arquitectura por capas para gestionar su plataforma de comercio electrónico. La capa de presentación gestiona las interfaces web y móviles, la capa de lógica de negocio gestiona el procesamiento de pedidos, inventario y recomendaciones, y la capa de datos maneja las bases de datos distribuidas y el almacenamiento en la nube. La separación en capas ha permitido que Amazon escale globalmente y maneje millones de transacciones por minuto.
- **Netflix:** También sigue una arquitectura por capas. La capa de presentación incluye la interfaz de usuario en dispositivos como televisores inteligentes, móviles y navegadores web. La capa de lógica de negocio gestiona recomendaciones, personalización y la reproducción de contenido. La capa de datos maneja el acceso a bases de datos distribuidas y al sistema de almacenamiento de video.
- **Microsoft:** Su plataforma empresarial Dynamics 365 utiliza una arquitectura por capas para separar la lógica de negocio (por ejemplo, la gestión de relaciones con clientes, CRM), la interfaz de usuario (para dispositivos móviles y web), y el acceso a datos a través de servicios en la nube.

Qué tan común es el stack designado

- La arquitectura por capas es ampliamente utilizada en diversas industrias, como la financiera, de salud y comercio electrónico, así como en aplicaciones web y empresariales. Es común porque facilita la modularidad, escalabilidad y mantenibilidad del software al separar la presentación, la lógica de negocio y el acceso a datos. Frameworks populares como Angular, Spring, .NET y Django promueven este enfoque, permitiendo una organización clara y eficiente del sistema. Aunque tecnologías modernas como los microservicios tienden a desacoplar componentes, muchos microservicios siguen una estructura interna por capas, manteniendo su relevancia.

II. Python/Django

Definición

- Django es un framework de desarrollo web de alto nivel, escrito en Python, que sigue el patrón de diseño MTV (Model-Template-View). Este framework está diseñado para acelerar el desarrollo de aplicaciones web, permitiendo a los desarrolladores enfocarse

en escribir código para la lógica de la aplicación mientras Django se encarga de aspectos comunes como la autenticación de usuarios, la gestión de formularios, la seguridad y la administración de bases de datos. Gracias a su filosofía de "baterías incluidas", Django ofrece una solución completa para construir sitios web robustos y escalables con menor esfuerzo de desarrollo.

Características

- **Completitud:** Incluye muchas herramientas listas para usar, como autenticación, administración, manejo de formularios, y enrutamiento, lo que reduce la necesidad de dependencias externas.
- **Escalabilidad:** Su estructura modular permite que una aplicación crezca fácilmente, ya sea añadiendo más funcionalidades o aumentando la capacidad de procesamiento para soportar más usuarios y tráfico.
- **Seguridad avanzada:** Django está diseñado con buenas prácticas de seguridad, como protección contra ataques comunes (CSRF, XSS, SQL Injection) incorporadas por defecto.
- **ORM (Object-Relational Mapping):** El ORM de Django permite a los desarrolladores interactuar con bases de datos usando código Python en lugar de SQL, lo que simplifica el desarrollo y la mantenibilidad del código.
- **Manejador de URLs:** El sistema de enrutamiento de Django facilita el mapeo de URLs a las vistas de la aplicación, proporcionando flexibilidad en la configuración de las rutas.
- **Vistas basadas en clases y plantillas reutilizables:** Fomenta la reutilización del código y la separación de responsabilidades, lo que contribuye a la mantenibilidad.

Historia y evolución

- 2003: Django fue creado inicialmente por Adrian Holovaty y Simon Willison mientras trabajaban en un proyecto en una organización de medios de comunicación. Se diseñó para agilizar el desarrollo de sitios web y aplicaciones dinámicas, permitiendo a los desarrolladores lanzar productos en semanas en lugar de meses.
- 2005: Django fue lanzado como código abierto, con el objetivo de compartir su estructura robusta y rápida con la comunidad de desarrolladores. Este marco ganó rápidamente popularidad debido a su integración de herramientas y su enfoque en la seguridad.
- Años 2010-2020: Django siguió evolucionando para adaptarse a las demandas modernas, con una adopción masiva en startups y empresas tecnológicas. Su popularidad creció especialmente gracias a su fácil integración con Python, un lenguaje conocido por su simplicidad y amplia comunidad.
- Actualmente: Django es uno de los frameworks más utilizados para desarrollo web empresarial y escalable, siendo adoptado por gigantes tecnológicos como Instagram,

Pinterest y Mozilla, manteniendo su enfoque en la seguridad, la eficiencia y la modularidad.

Ventajas y desventajas

- **Ventajas:**
  - **Desarrollo rápido:** La filosofía "baterías incluidas" permite a los desarrolladores crear aplicaciones completas en menos tiempo al no tener que buscar e integrar herramientas adicionales para aspectos comunes.
  - **Escalabilidad:** El diseño de Django permite manejar aplicaciones de alta demanda, que pueden crecer sin comprometer el rendimiento.
  - **Comunidad y soporte:** Django tiene una gran comunidad global que contribuye con soluciones, documentación extensa y paquetes reutilizables, como Django REST Framework.
  - **Seguridad predeterminada:** Proporciona mecanismos para proteger automáticamente las aplicaciones contra amenazas comunes, sin necesidad de configuración adicional.
- **Desventajas:**
  - **Curva de aprendizaje:** Aunque Python es un lenguaje sencillo, Django tiene una curva de aprendizaje debido a la cantidad de características y convenciones que maneja.
  - **Sobrecarga de funcionalidades:** En proyectos pequeños o sencillos, algunas características de Django pueden sentirse innecesarias, añadiendo complejidad sin requerirse.
  - **Rendimiento:** En comparación con frameworks más ligeros (como Flask), Django puede consumir más recursos, lo que puede afectar el rendimiento si no se optimiza correctamente.

Casos de Uso

- **Desarrollo de sitios web complejos:** Es ideal para plataformas web que requieren gestionar grandes volúmenes de datos o usuarios, como sistemas de e-commerce, redes sociales o sitios de noticias.
- **Aplicaciones que demandan una seguridad robusta:** Es común ver Django utilizado en aplicaciones financieras, bancarias o médicas donde la seguridad de los datos es crucial.
- **Proyectos con plazos ajustados:** Debido a su capacidad para permitir el desarrollo rápido, es adecuado para startups o empresas que necesitan lanzar un MVP (Producto Mínimo Viable) rápidamente al mercado.

Casos de aplicación

- **Instagram:** Django soporta la infraestructura que permite a Instagram manejar millones de usuarios activos diarios y gestionar una gran cantidad de contenido multimedia sin sacrificar la escalabilidad ni el rendimiento.
- **Pinterest:** Utiliza Django para gestionar su plataforma visual, proporcionando tanto una experiencia de usuario fluida como una interfaz backend eficiente para el manejo de millones de imágenes.
- **Mozilla:** Django es la base de varios sitios web y aplicaciones de Mozilla, demostrando su capacidad para manejar proyectos de gran envergadura, con alto tráfico y estrictas demandas de seguridad.

Qué tan común es el stack designado

- El stack Python/Django es extremadamente común en la industria tecnológica debido a varias razones, entra las que se encuentras las siguientes:
    - Python es uno de los lenguajes de programación más populares y versátiles, utilizado tanto en desarrollo web como en ciencia de datos, inteligencia artificial y automatización.
    - Django es el framework preferido por muchas empresas para aplicaciones web debido a su seguridad, robustez y la capacidad de escalar sin problemas. Esto lo convierte en una opción recurrente en startups, empresas medianas y grandes que buscan lanzar productos al mercado rápidamente y sin comprometer la calidad.

III. Flutter/Dart

Definición

- Flutter es un framework de desarrollo de aplicaciones multiplataforma creado por Google. Permite a los desarrolladores crear aplicaciones móviles, web y de escritorio a partir de una base de código única. Utiliza el lenguaje de programación Dart y proporciona una amplia colección de herramientas y widgets personalizables, facilitando el desarrollo de interfaces de usuario ricas y nativas. Flutter destaca por su enfoque en la experiencia del usuario, el rendimiento y la rapidez en el desarrollo.
- Dart es un lenguaje de programación orientado a objetos desarrollado por Google, diseñado para construir aplicaciones rápidas y eficientes. Dart es el lenguaje subyacente que impulsa a Flutter, optimizado tanto para el frontend (interfaces) como para backend (lógica).

Características

- **Multiplataforma:** Flutter permite desarrollar aplicaciones móviles (iOS, Android), web y de escritorio utilizando un solo código base.
- **Hot Reload:** Permite a los desarrolladores ver los cambios realizados en el código en tiempo real sin necesidad de recompilar toda la aplicación, acelerando el desarrollo y la depuración.
- **Rendimiento cercano al nativo:** Flutter se destaca por su capacidad para renderizar gráficos y manejar animaciones con un rendimiento casi indistinguible de aplicaciones nativas.
- **Gran cantidad de widgets:** Ofrece una vasta colección de widgets personalizables, lo que facilita la creación de interfaces de usuario que se ven y se sienten nativas, tanto en Android como en iOS.
- **Soporte para Material Design y Cupertino:** Facilita la creación de interfaces con los estilos visuales nativos de Android (Material Design) e iOS (Cupertino), lo que garantiza una experiencia coherente en ambas plataformas.
- **Backend opcional con Dart:** Aunque Flutter se centra en la interfaz, Dart también se puede usar para el backend, proporcionando una solución completa en ciertas aplicaciones.

Historia y evolución

- 2011: Google presentó por primera vez Dart como un lenguaje de programación diseñado para reemplazar JavaScript, enfocándose en mejorar el rendimiento en aplicaciones web.
- 2015: Se lanza Flutter (originalmente conocido como Sky), como un framework experimental que utilizaba Dart para construir interfaces de usuario altamente dinámicas para Android y iOS.
- 2017: Google lanzó la primera versión estable de Flutter durante su conferencia Google I/O, centrado principalmente en el desarrollo de aplicaciones móviles.
- 2019: Con el lanzamiento de Flutter 1.0, el framework fue oficialmente considerado estable y listo para producción, ganando tracción rápidamente debido a su facilidad de uso, rendimiento y compatibilidad multiplataforma.
- 2020: Se amplía la funcionalidad de Flutter con soporte para desarrollo web y de escritorio, consolidándose como una herramienta versátil para crear interfaces en una variedad de plataformas.
- 2023 en adelante: Flutter sigue evolucionando, con una gran comunidad de desarrolladores y la inclusión de más características, como Flutter Web y Flutter Desktop, lo que lo convierte en uno de los frameworks más populares para desarrollo multiplataforma.

Ventajas y desventajas

- **Ventajas:**
  - **Desarrollo multiplataforma:** Permite crear aplicaciones para diversas plataformas con una única base de código, reduciendo significativamente los tiempos y costos de desarrollo.
  - **Hot Reload:** Acelera el proceso de desarrollo al permitir iterar rápidamente sobre el diseño y la lógica de la aplicación.
  - **Rendimiento similar al nativo:** Las aplicaciones Flutter son rápidas y tienen un rendimiento cercano al de las aplicaciones nativas, gracias a su motor de gráficos propio (Skia).
  - **Amplia personalización:** Los desarrolladores tienen control total sobre los widgets y las interfaces, permitiendo una personalización profunda del diseño.
  - **Soporte de Google:** Google está detrás del desarrollo de Flutter, lo que garantiza su actualización constante y soporte a largo plazo.
- **Desventajas:**
  - **Tamaño de las aplicaciones:** Las aplicaciones Flutter suelen ser más grandes en tamaño que las aplicaciones nativas debido a la inclusión del motor de renderizado y otros recursos.
  - **Curva de aprendizaje:** Aunque Dart es sencillo de aprender, los desarrolladores que vienen de lenguajes como JavaScript o Swift pueden tardar en adaptarse.
  - **Menos plugins que nativo:** Aunque la comunidad de Flutter está creciendo rápidamente, algunos plugins o integraciones nativas específicas pueden no estar disponibles, comparado con el desarrollo nativo en iOS o Android.

Casos de Uso

- **Desarrollo de aplicaciones móviles multiplataforma:** Ideal para empresas que buscan desarrollar una aplicación que funcione tanto en Android como en iOS con un solo código base.
- **Proyectos con plazos ajustados:** Flutter es adecuado para startups o equipos de desarrollo que necesitan lanzar aplicaciones rápidamente gracias a su Hot Reload y facilidad para desarrollar interfaces complejas.
- **Aplicaciones que requieren interfaces altamente personalizables:** Flutter es una excelente opción para aplicaciones que priorizan la estética y la interacción del usuario, como aplicaciones de comercio electrónico, redes sociales, o juegos.
- **Desarrollo web y de escritorio:** Aunque originalmente estaba enfocado en móviles, Flutter también permite crear aplicaciones web y de escritorio, haciendo que sea ideal para proyectos que necesitan alcanzar múltiples plataformas.

Casos de aplicación

- **Google Ads:** Google utiliza Flutter para su propia aplicación de Google Ads, que permite a los usuarios gestionar sus campañas publicitarias en múltiples plataformas, mostrando el poder de Flutter en aplicaciones comerciales y de alto tráfico.
- **Alibaba:** La compañía china utiliza Flutter para su aplicación Xianyu, una plataforma de comercio de segunda mano que maneja millones de usuarios activos, demostrando la capacidad de Flutter para manejar aplicaciones escalables.
- **BMW:** BMW adoptó Flutter para su aplicación My BMW, que ofrece servicios conectados a sus vehículos. Esta es una muestra de cómo Flutter puede ser utilizado en sectores industriales y aplicaciones IoT.
- **Reflectly:** Una aplicación de bienestar y journaling desarrollada completamente en Flutter. Reflectly es un ejemplo de cómo Flutter puede manejar tanto la estética como la funcionalidad en una aplicación con una excelente experiencia de usuario.

Qué tan común es el stack designado

- El stack Flutter/Dart ha ganado una gran popularidad en los últimos años y es cada vez más común en la industria. Se utiliza en empresas de tecnología, startups y proyectos de desarrollo móvil que requieren una rápida salida al mercado sin sacrificar la calidad ni el rendimiento. El soporte de Google ha fortalecido su crecimiento y adopción, y su capacidad de desarrollo multiplataforma lo ha convertido en una de las soluciones más buscadas para aplicaciones móviles y, recientemente, aplicaciones web y de escritorio.

IV. WebSocket

Definición

- WebSocket es un protocolo de comunicación bidireccional que permite la transmisión de datos en tiempo real entre un cliente (generalmente un navegador web) y un servidor. A diferencia del protocolo HTTP estándar, WebSocket establece una conexión persistente, lo que permite la comunicación continua sin la necesidad de realizar múltiples solicitudes y respuestas. Esto es ideal para aplicaciones que requieren actualizaciones en tiempo real, como chats, videojuegos multijugador y aplicaciones financieras.

Características

- **Comunicación bidireccional:** Tanto el cliente como el servidor pueden enviar mensajes entre sí en cualquier momento, sin tener que esperar a una solicitud.

- **Conexión persistente:** Una vez establecida, la conexión WebSocket permanece abierta, lo que reduce la latencia asociada con la creación de nuevas conexiones.
- **Bajo consumo de recursos:** Al eliminar la sobrecarga de solicitudes repetitivas, WebSocket es más eficiente en el uso de ancho de banda y recursos.
- **Soporte para mensajes en tiempo real:** Permite la entrega instantánea de mensajes, lo que es crucial para aplicaciones que requieren sincronización en tiempo real.
- **Protocolo ligero:** WebSocket es menos costoso en términos de procesamiento de datos en comparación con HTTP, debido a la reducción de encabezados.
- **Compatibilidad:** Es compatible con la mayoría de los navegadores modernos y se puede integrar fácilmente con diversos lenguajes y plataformas del servidor.

Historia y evolución

- 2008: Comienzan las primeras discusiones en torno a la creación de un protocolo que permita una mejor comunicación en tiempo real, como parte del desarrollo del estándar HTML5.
- 2009: Se lanzó el primer borrador de la especificación WebSocket API bajo el W3C, y se propuso el protocolo en el IETF como un mecanismo para reemplazar las ineficiencias de HTTP para aplicaciones interactivas.
- 2011: La especificación de WebSocket fue finalizada y estandarizada por la IETF (RFC 6455), y su API fue estandarizada por el W3C.
- 2012 en adelante: WebSocket empezó a ser adoptado en aplicaciones comerciales y se popularizó rápidamente, reemplazando soluciones de tiempo real más costosas como long polling.
- 2020 en adelante: Con la evolución de tecnologías como WebRTC, WebSocket ha mantenido su relevancia en el manejo de aplicaciones en tiempo real, especialmente en aplicaciones de gran escala como juegos multijugador, videollamadas, y transacciones bursátiles.

Ventajas y desventajas

- **Ventajas:**
    - **Baja latencia:** Las conexiones persistentes y bidireccionales permiten la transmisión de datos casi instantánea, eliminando la latencia típica del ciclo de solicitud/respuesta de HTTP.
    - **Menos sobrecarga:** A diferencia de HTTP, que requiere múltiples encabezados en cada solicitud, WebSocket solo requiere un encabezado al establecer la conexión, reduciendo el uso de ancho de banda.
    - **Ideal para aplicaciones en tiempo real:** WebSocket es la solución perfecta para aplicaciones que requieren actualizaciones continuas y sincronización en tiempo real.

    - **Escalabilidad:** Los servidores pueden gestionar más conexiones simultáneas con WebSocket, ya que no es necesario establecer una nueva conexión para cada interacción.
- **Desventajas:**
    - **Compatibilidad con proxies:** Algunas redes corporativas o proxies pueden no manejar adecuadamente las conexiones WebSocket, lo que puede causar problemas de conectividad.
    - **Seguridad:** Aunque existen versiones seguras de WebSocket (wss://), las conexiones no cifradas (ws://) pueden ser vulnerables a ataques como la interceptación de tráfico.
    - **Mayor complejidad:** El manejo de conexiones persistentes puede agregar complejidad en la implementación del servidor, ya que requiere gestionar conexiones a largo plazo.
    - **Control limitado de la conexión:** A diferencia de HTTP, que abre y cierra conexiones en cada interacción, WebSocket necesita un manejo adecuado para detectar cierres o fallos en las conexiones.

Casos de Uso

- **Chats en tiempo real:** WebSocket es ideal para aplicaciones de mensajería donde los usuarios necesitan recibir y enviar mensajes instantáneamente.
- **Juegos multijugador:** Permite la sincronización en tiempo real entre los jugadores, enviando actualizaciones de estado y acciones de forma instantánea.
- **Aplicaciones financieras:** WebSocket es común en plataformas bursátiles donde es crucial recibir actualizaciones de precios y transacciones en tiempo real.
- **Notificaciones en tiempo real:** Aplicaciones web que necesitan enviar notificaciones push a los usuarios, como actualizaciones en redes sociales o correos electrónicos.
- **Control de dispositivos IoT:** Se utiliza para la interacción con dispositivos de Internet de las Cosas (IoT) en tiempo real, permitiendo la monitorización y control continuo.

Casos de aplicación

- **Slack:** Esta popular herramienta de colaboración utiliza WebSocket para permitir la comunicación en tiempo real entre los usuarios, facilitando el envío y recepción de mensajes instantáneos.
- **Binance:** La plataforma de intercambio de criptomonedas utiliza WebSocket para transmitir información de precios en tiempo real, garantizando que los usuarios siempre tengan acceso a los datos más recientes.
- **Trello:** El sistema de gestión de proyectos Trello usa WebSocket para actualizar automáticamente los tableros y tareas, permitiendo que los usuarios vean los cambios en tiempo real sin necesidad de recargar la página.

- **Uber:** Uber utiliza WebSocket para enviar actualizaciones en tiempo real a sus conductores y pasajeros sobre la ubicación del vehículo, cambios en las rutas y el estado del viaje.
- **Facebook Messenger:** Utiliza WebSocket para enviar y recibir mensajes instantáneamente entre usuarios, asegurando que las conversaciones estén sincronizadas en todos los dispositivos.

Qué tan común es el stack designado

- WebSocket es muy común en aplicaciones que requieren comunicación en tiempo real. Con el crecimiento de aplicaciones interactivas, como juegos multijugador, videoconferencias y plataformas de comercio financiero, WebSocket se ha convertido en una herramienta estándar en la industria tecnológica. Si bien puede no ser necesario en todas las aplicaciones, es ampliamente utilizado en sectores que dependen de la baja latencia y la sincronización en tiempo real, lo que lo convierte en una opción popular para muchas soluciones modernas.

V. MySQL

Definición

- MySQL es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto. Utiliza el lenguaje SQL (Structured Query Language) para gestionar y organizar los datos. Es ampliamente utilizado en aplicaciones web, gracias a su robustez, velocidad y flexibilidad. MySQL es conocido por su capacidad para manejar grandes cantidades de datos y es una de las bases de datos más populares en el mundo.

Características

- **Relacional:** Organiza los datos en tablas que pueden relacionarse entre sí mediante claves.
- **SQL (Structured Query Language):** MySQL utiliza SQL como su lenguaje principal para consultar y manipular datos.
- **Alta performance:** Es eficiente y rápido, especialmente en operaciones de lectura.
- **Código abierto:** Está disponible bajo licencia GPL, lo que significa que es gratuito y puede ser modificado por los desarrolladores.
- **Multiplataforma:** Funciona en múltiples sistemas operativos, como Linux, Windows y macOS.
- **Alta escalabilidad:** Puede manejar bases de datos pequeñas y grandes, desde aplicaciones personales hasta sistemas empresariales.

- **Compatibilidad con replicación y clustering:** Permite la replicación de datos entre servidores para mejorar el rendimiento y la disponibilidad.
- **Transacciones seguras:** Soporta transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), asegurando integridad de datos.

Historia y evolución

- 1995: MySQL fue desarrollado por Michael Widenius, Allan Larsson, y David Axmark en Suecia. Originalmente creado como un sistema rápido y sencillo para manejar bases de datos pequeñas.
- 2000: MySQL se convierte en open source bajo la licencia GPL, lo que impulsa su popularidad en la comunidad de desarrolladores.
- 2005: MySQL AB (la empresa detrás de MySQL) fue adquirida por Sun Microsystems, lo que permitió una inversión más significativa en su desarrollo.
- 2010: Oracle Corporation adquirió Sun Microsystems, y MySQL pasó a ser parte del portafolio de Oracle. Aunque surgieron dudas sobre su futuro, MySQL siguió evolucionando con nuevas características y mejoras en rendimiento.
- 2015: Se lanzó MySQL 5.7, una versión que introdujo mejoras significativas en el rendimiento y las capacidades de JSON.
- 2021: Con la versión 8.0, MySQL amplió su soporte para funciones modernas, como expresiones de ventana, CTEs (Common Table Expressions) y mejoras en la seguridad.

Ventajas y desventajas

- **Ventajas:**
    - **Código abierto y gratuito:** Al ser de código abierto, los desarrolladores pueden utilizarlo sin costo y personalizarlo según sus necesidades.
    - **Velocidad y eficiencia:** MySQL es conocido por su rendimiento rápido en operaciones de lectura y su capacidad para manejar grandes cantidades de datos.
    - **Fácil de usar:** La simplicidad de su sintaxis SQL lo hace accesible para nuevos usuarios y desarrolladores experimentados.
    - **Amplia comunidad:** Tiene una comunidad global activa que contribuye a su desarrollo y proporciona soporte.
    - **Seguridad:** Proporciona sólidas características de seguridad para el acceso a los datos, incluidos cifrado y autenticación robusta.
- **Desventajas:**
    - **Limitaciones en operaciones avanzadas:** Aunque es muy rápido en operaciones de lectura, puede ser menos eficiente en transacciones complejas y escrituras masivas en comparación con otros RDBMS.

    - **Soporte limitado para grandes sistemas transaccionales:** Aunque MySQL ha mejorado su capacidad transaccional, no siempre es la mejor opción para aplicaciones extremadamente grandes o críticas.
    - **Falta de características específicas de bases de datos empresariales:** En comparación con sistemas como Oracle o SQL Server, MySQL puede carecer de algunas características avanzadas que son críticas para grandes empresas.
    - **Escalabilidad vertical limitada:** Aunque es escalable horizontalmente mediante replicación, su capacidad para escalar verticalmente tiene limitaciones en comparación con otros sistemas de bases de datos.

Casos de Uso

- **Sistemas de gestión de contenido (CMS):** MySQL es el motor de base de datos de muchas plataformas CMS populares como WordPress, Drupal y Joomla.
- **Aplicaciones web:** Es ampliamente utilizado en el desarrollo de aplicaciones web debido a su integración con lenguajes de programación como PHP y Python, y su capacidad para manejar grandes volúmenes de tráfico y datos.
- **E-commerce:** Las plataformas de comercio electrónico, como Magento y WooCommerce, utilizan MySQL para gestionar catálogos de productos, datos de clientes y transacciones.
- **Aplicaciones empresariales:** MySQL se utiliza en muchas aplicaciones empresariales para manejar datos de clientes, inventarios, facturación y más.
- **Sistemas de seguimiento de bugs:** Herramientas como Bugzilla y Mantis utilizan MySQL para almacenar y gestionar información relacionada con problemas y seguimiento de errores.

Casos de aplicación

- **Facebook:** Originalmente, la infraestructura de Facebook fue construida sobre MySQL, lo que permitió a la empresa escalar y gestionar el enorme volumen de datos generado por sus usuarios.
- **Twitter:** Durante sus primeros años, Twitter utilizó MySQL como su base de datos principal para gestionar la información relacionada con los tweets y usuarios.
- **GitHub:** La plataforma de alojamiento de código fuente utiliza MySQL para manejar sus repositorios y datos relacionados con el control de versiones y los usuarios.
- **YouTube:** El famoso sitio de videos también utiliza MySQL para gestionar información de usuarios, videos y comentarios, permitiendo una operación eficiente a escala masiva.
- **Netflix:** MySQL es parte de la infraestructura de datos de Netflix, manejando el almacenamiento y la recuperación de grandes volúmenes de información relacionada con su catálogo de contenido y datos de usuarios.

Qué tan común es el stack designado

- MySQL es extremadamente común, especialmente en entornos de desarrollo LAMP (Linux, Apache, MySQL, PHP/Python). Es uno de los sistemas de gestión de bases de datos más utilizados en el mundo, especialmente en aplicaciones web. Es popular tanto para proyectos pequeños como para aplicaciones empresariales grandes, debido a su simplicidad, eficiencia, y la amplia gama de soporte y documentación disponible. MySQL es también el motor predeterminado en muchos proveedores de servicios en la nube, lo que refuerza su uso en la industria moderna.

Matrices vs Temas

Para las matrices se provee un link a un excel con las mismas para su lectura y análisis: "https://livejaverianaedu-my.sharepoint.com/:x:/g/personal/buitrago_je_javeriana_edu_co/EZsZLnhIOY1Igt-hghgND5YBND9qO0fPn3gw0NkL_GCT4A?e=aeCWza".

Diagramas

Alto nivel

Para la representación de la arquitectura de alto nivel se usaron tres capas principales (Presentación, Lógica y Persistencia) y las describiré a continuación.

- Capa de Presentación (Flutter):
  - Es la capa encargada de la interacción con el usuario final. Utiliza el framework Flutter para proporcionar una interfaz gráfica, a través de la cual se envían solicitudes (requests) y mensajes WebSocket hacia la capa lógica.
- Capa de Lógica (Django):
  - Aquí se gestiona toda la lógica de negocio del sistema. Django, con su ORM (Object-Relational Mapping), facilita la conexión entre la aplicación y la base de datos. Esta capa recibe las solicitudes desde Flutter, procesa la información y responde en función de la lógica del negocio.
- Capa de Persistencia (MySQL):
  - En esta capa se encuentra la base de datos MySQL, que almacena de manera persistente toda la información del sistema. Django se comunica directamente con MySQL para realizar operaciones de lectura y escritura.

Las tres capas se comunican entre sí de manera estructurada, lo que asegura un flujo de datos eficiente y escalable en la aplicación. A continuación se muestra el diagrama de Alto nivel realizado.

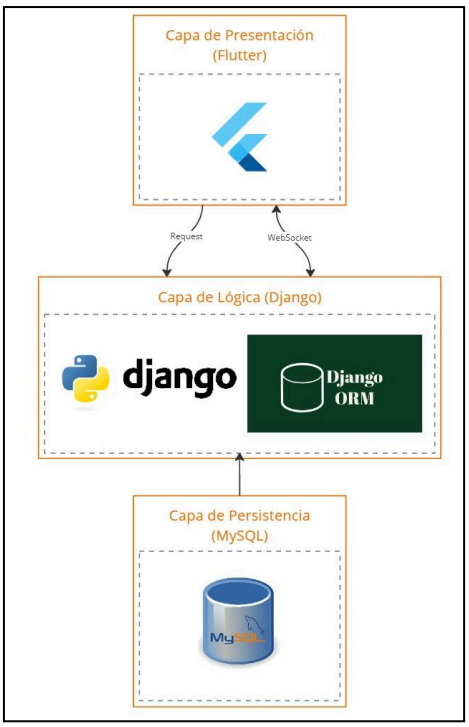

Imagen I. Diagrama Alto Nivel

C4 Model

Para la realización del diagrama de C4 Model se cuenta con tres niveles, los cuales son los de Contexto, Contenedores y Componentes, proporcionando una visión clara de cómo los usuarios interactúan con el sistema y cómo este se organiza internamente.

- Nivel I: Contexto
    - Este nivel muestra cómo el Usuario Final interactúa con el sistema a través de la Aplicación Flutter, que gestiona todas las operaciones que realiza el usuario.
- Nivel II: Contenedores (Acá básicamente se desglosan las principales partes del sistema:)
    - Capa de Presentación (Flutter): Los usuarios interactúan con la interfaz gráfica.
    - Capa de Lógica (API REST - Django): Maneja la lógica del sistema y las interacciones con la base de datos.
    - Base de Datos (MySQL): Almacena la información generada por la aplicación.
- Nivel III: Componentes (En esta ocasión se profundiza en los componentes internos del sistema)
    - Flutter: Interfaz que permite las interacciones del usuario.
    - Serializadores (Django REST Framework): Convierte datos entre formatos.
    - Componente de Seguridad (Django): Gestiona la seguridad en el acceso a la base de datos.
    - Base de Datos (MySQL): Gestiona las operaciones de lectura y escritura de datos.

A continuación se muestra el diagrama de C4 Model.

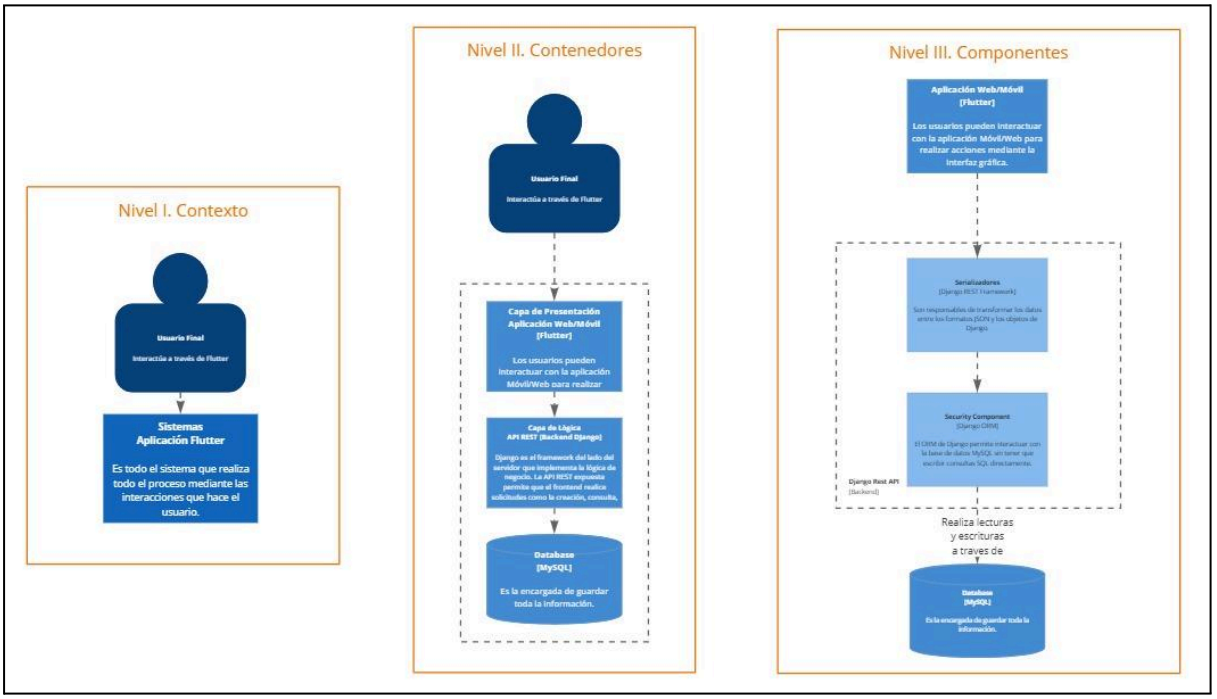

Imagen II. Diagrama C4 Model

Diagrama Despliegue C4

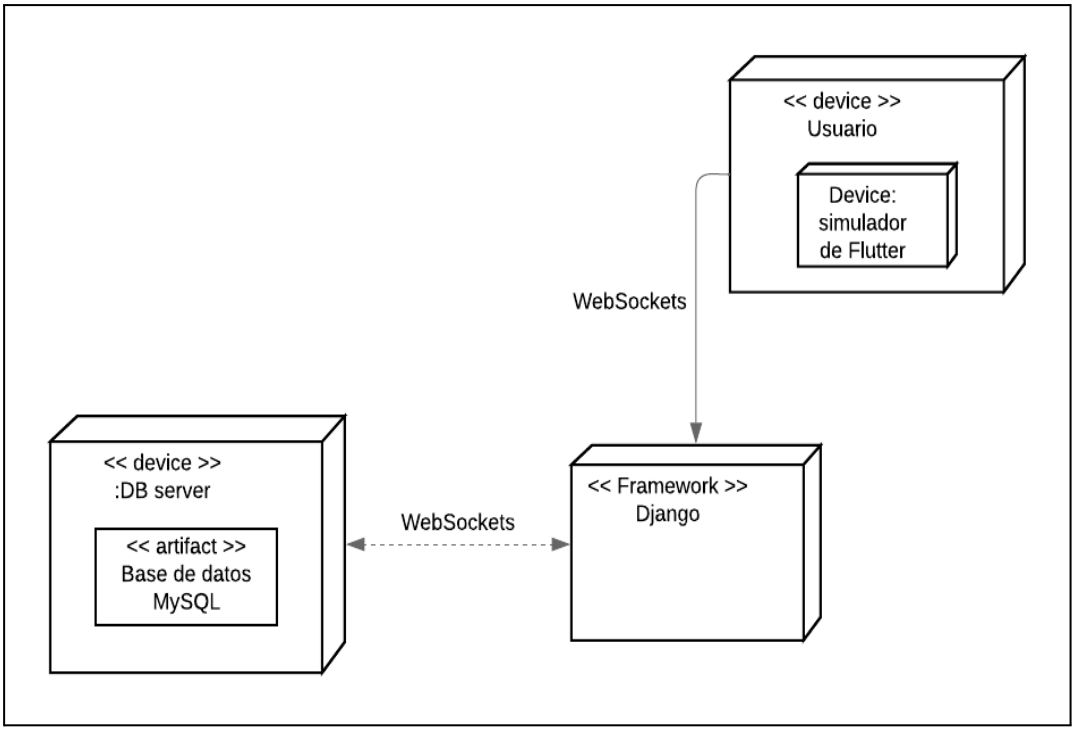

Imagen III. Diagrama C4 Despliegue

Diagrama de paquetes UML

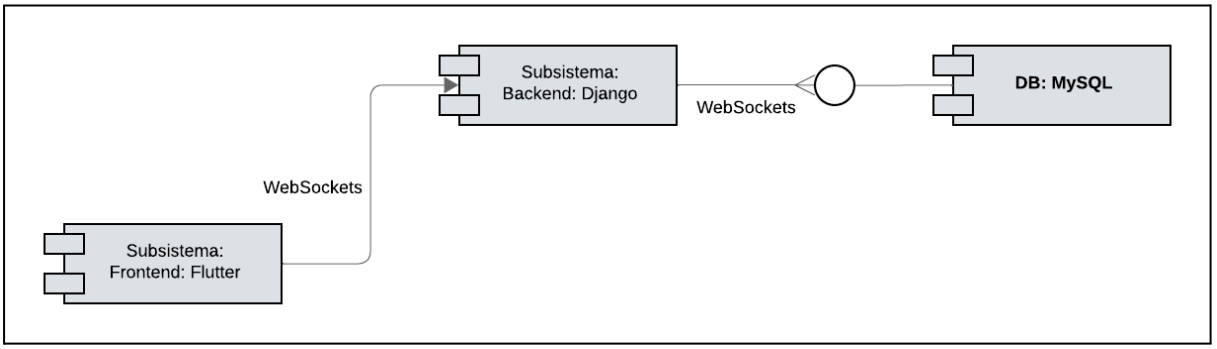

Imagen IV. Diagrama C4 de paquetes

Código Fuente en repositorio/s públicos git

- A continuación se encuentra el link del repositorio en Git: "https://github.com/Frank5005/Taller2ArquitecturaSoftware".

Uso obligatorio de Docker/Postman

- Se realizó la implementación principalmente buscando cumplir con los requerimientos como Docker entre otros, sin embargo, tuvimos problemas para el correcto funcionamiento de la solución propuesta. Todo esto se encuentra detallado dentro del repositorio de GitHub.