Pontificia Universidad Javeriana

Presentación 1

Arquitectura de Software

Integrantes:

Camilo Esteban Mora Gómez

Santiago Barajas

Santiago Guerrero

Presentado a:

Andrés Armando Sánchez

21/04/2025

Bogotá, Colombia

Contenido

1. Stack usado

1.1. SOA

1.1.1. Definición

La Arquitectura Orientada a Servicios (SOA) es un estilo arquitectónico que organiza un sistema de software como un conjunto de servicios interoperables, independientes y reutilizables. Cada servicio es una unidad funcional autónoma que se comunica con otros mediante interfaces bien definidas y estándares abiertos, comúnmente usando protocolos como SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) y XML. En este modelo, los servicios pueden ser desarrollados en distintos lenguajes o plataformas y luego integrados sin problemas dentro del ecosistema general del sistema. Esta independencia permite una alta escalabilidad, mantenibilidad y adaptabilidad.

1.1.2. Características

- Interoperabilidad tecnológica: SOA facilita que aplicaciones construidas con tecnologías distintas (por ejemplo, frontend en Svelte y backend en Jakarta EE) puedan comunicarse de manera estandarizada a través de servicios SOAP. Esto es crucial en entornos donde conviven múltiples tecnologías.
- Bajo acoplamiento: Los servicios están desacoplados entre sí, lo que significa que pueden evolucionar o cambiar sin afectar directamente a otros módulos. Esto favorece la evolución y mantenimiento continuo del sistema.
- Reusabilidad: Un mismo servicio, como el de agendamiento de citas o verificación de disponibilidad de un médico, puede ser utilizado en diferentes aplicaciones: desde una web administrativa hasta una aplicación móvil de pacientes.
- Composición de servicios: SOA permite crear procesos más complejos a través de la orquestación o coreografía de múltiples servicios.
- Escalabilidad organizacional: La modularidad favorece el trabajo en equipos distribuidos, donde cada equipo puede gestionar un servicio específico.

1.1.3. Historia y evolución

SOA surgió a principios de los años 2000 como respuesta a la necesidad de integrar diversos sistemas legados en grandes corporaciones que utilizaban tecnologías

heterogéneas. Su auge se debió al fuerte impulso de empresas como IBM, Oracle, SAP y Microsoft, quienes promovieron el uso de servicios estándar como medio de integración.

Durante años fue el enfoque dominante en sectores críticos como banca, salud y gobierno, especialmente por su robustez, capacidad de integración y alineación con estándares. Sin embargo, con el crecimiento del desarrollo ágil y de arquitecturas basadas en contenedores, surgieron alternativas más livianas como los microservicios basados en REST. Aun así, SOA sigue vigente y es preferido en entornos donde la estabilidad, la trazabilidad y la interoperabilidad formal son más importantes que la velocidad de desarrollo.

1.1.4. Ventajas y desventajas

Ventajas	**Desventajas**
Modularidad del backend: Al separar el sistema en servicios, se puede actualizar o escalar partes específicas sin afectar el todo.	Complejidad en el despliegue: Requiere infraestructura adicional (servicios de descubrimiento, brokers de mensajes, políticas de seguridad).
Interoperabilidad estandarizada: Facilita la integración con sistemas de terceros (EPS, hospitales, aseguradoras), incluso si usan tecnologías distintas.	Overhead de comunicación: El uso de XML y protocolos como SOAP puede ser más pesado que alternativas como REST+JSON.
Reutilización de servicios: Reduce duplicación de lógica y facilita futuras integraciones (por ejemplo, con una aplicación móvil o nuevos portales).	No ideal para proyectos pequeños o de rápido desarrollo: La configuración y diseño de servicios puede ralentizar el inicio de un proyecto si no se justifica.

1.1.5. Casos de uso

- Integración con sistemas legados, **e**mpresas con sistemas antiguos (ERP, mainframes, etc.) que necesitan exponer o consumir servicios.

- Orquestación compleja de servicios (ESB)**, f**lujos de negocio que requieren enrutamiento, transformación de datos y coordinación entre múltiples servicios.
- Servicios compartidos como autenticación, mensajería de recordatorios, validación de identidad o generación de reportes médicos.

1.1.6. Casos de aplicación

- NHS Spine (Reino Unido): Plataforma nacional que soporta la interoperabilidad de millones de historiales médicos a través de servicios SOA, permitiendo integración entre hospitales, farmacias, laboratorios y servicios de emergencia.
- Ministerio de Salud de Colombia: Implementa servicios SOAP para asegurar la interoperabilidad entre las EPS (Entidades Promotoras de Salud) y los IPS (Instituciones Prestadoras de Salud), usando WS-Security y estándares HL7.
- SUNAT (Perú): Arquitectura SOA para declaración electrónica de impuestos e integración de diversos sistemas financieros nacionales.

1.2. Svelte

1.2.1. Definición

Svelte es un framework de desarrollo frontend moderno, diseñado para construir interfaces de usuario (UI) altamente eficientes y reactivas. A diferencia de otros frameworks como React o Vue, Svelte no implementa un DOM virtual ni ejecuta una librería pesada en tiempo de ejecución. En cambio, compila los componentes a JavaScript puro durante el proceso de build (tiempo de compilación), lo cual reduce significativamente el tamaño del bundle final, mejora el rendimiento en el navegador y simplifica el código. Esta característica hace que Svelte sea una solución ideal para aplicaciones web ligeras, interactivas y que requieren tiempos de respuesta inmediatos.

1.2.2. Características

- Svelte es un framework de desarrollo frontend moderno, diseñado para construir interfaces de usuario (UI) altamente eficientes y reactivas. A diferencia de otros frameworks como React o Vue, Svelte no implementa un DOM virtual ni ejecuta una librería pesada en tiempo de ejecución. En cambio, compila los componentes a JavaScript puro durante el proceso de build (tiempo de compilación), lo cual reduce significativamente el tamaño del bundle final, mejora el rendimiento en el

navegador y simplifica el código. Esta característica hace que Svelte sea una solución ideal para aplicaciones web ligeras, interactivas y que requieren tiempos de respuesta inmediatos.

- Para el sistema de citas médicas, Svelte se convierte en una herramienta estratégica para ofrecer una interfaz de usuario moderna, ágil y fácilmente mantenible, que permita a pacientes y personal médico interactuar con los distintos servicios del backend de manera intuitiva y eficiente.
- Ligereza y eficiencia: Svelte produce archivos de salida mucho más pequeños que otros frameworks, lo cual mejora la velocidad de carga en navegadores, especialmente en dispositivos móviles o con conectividad limitada. Esto es crucial para usuarios que accedan al sistema de citas desde zonas con infraestructura limitada.
- Reactividad integrada: La gestión de estado y la actualización de la interfaz se realizan automáticamente sin necesidad de herramientas externas o conceptos complejos como "hooks". Por ejemplo, una vez que se actualiza la lista de citas disponibles desde el backend, la interfaz se actualiza inmediatamente.
- Sintaxis amigable y mantenimiento sencillo: La estructura de los componentes Svelte se basa en HTML, CSS y JavaScript estándar, reduciendo la curva de aprendizaje y facilitando su adopción por nuevos desarrolladores.
- Integración sencilla con servicios externos: Aunque Svelte no tiene una solución propia para consumir servicios SOAP, puede integrar fácilmente herramientas como fetch() o axios, así como bibliotecas XML para interpretar respuestas SOAP, facilitando su uso en entornos basados en SOA como este proyecto.

1.2.3. Historia y evolución

Svelte fue creado por el periodista y desarrollador Rich Harris en 2016 como una respuesta a las complejidades de frameworks modernos que requerían mucha configuración y conocimiento profundo de sus mecanismos internos. Su propuesta revolucionaria consistía en trasladar el trabajo del framework al momento de la compilación, dejando al navegador un código mucho más simple y directo.

A partir de Svelte 3 (lanzado en 2019), el framework ganó reconocimiento por su enfoque minimalista, su rendimiento y su excelente experiencia de desarrollo. Actualmente cuenta

con una comunidad muy activa y crece rápidamente como alternativa lightweight para interfaces modernas.

1.2.4. Ventajas y desventajas

Ventajas	Desventajas
Rendimiento optimizado: Svelte ofrece cargas rápidas, ideal para una interfaz centrada en citas en tiempo real.	Ecosistema más pequeño: Existen menos plugins o integraciones listas para usar, lo cual puede requerir desarrollo manual de ciertas funcionalidades.
Curva de aprendizaje accesible: Permite que el equipo de desarrollo construya y mantenga la interfaz sin requerir años de experiencia en frameworks complejos.	Documentación aún en expansión: Aunque clara, la documentación aún no cubre tantos casos avanzados como otros frameworks.
Alta reactividad: Las actualizaciones de estado (como seleccionar un médico, cambiar fecha o ver disponibilidad) se reflejan de inmediato en la UI.	Integración con servicios SOAP no nativa: Requiere trabajo adicional para consumir servicios SOAP, ya que no tiene soporte directo como REST.

1.2.5. Casos de uso

- Aplicaciones médicas ligeras: Una interfaz web para pacientes y médicos que deseen agendar o visualizar citas desde un navegador, incluso en dispositivos móviles.
- Dashboards administrativos: Paneles de control para supervisores de clínicas que necesitan consultar rápidamente el estado de agenda, disponibilidad de profesionales o rendimiento del sistema.
- Formularios interactivos: Registro de pacientes, validación en tiempo real y búsqueda de disponibilidad por fechas.

1.2.6. Casos de aplicación

- Storyblok: Un sistema CMS moderno y modular que utiliza Svelte para construir interfaces rápidas y personalizables, con una experiencia UX fluida.
- Square (empresa de pagos): Emplea Svelte en herramientas internas para gestión de finanzas, donde la ligereza del framework permite interfaces rápidas y sin distracciones.
- Routify y SvelteKit: Plataformas construidas sobre Svelte que están siendo utilizadas en producción por múltiples empresas emergentes enfocadas en velocidad y escalabilidad.

1.3. Jakarta EE

1.3.1. Definición

Jakarta EE (Enterprise Edition) es un conjunto de especificaciones y tecnologías open source para el desarrollo de aplicaciones empresariales robustas, seguras, escalables y modulares utilizando el lenguaje de programación Java. Surge como la evolución directa de Java EE (Java Platform, Enterprise Edition), tras su transferencia por parte de Oracle a la Eclipse Foundation en 2017. Desde entonces, Jakarta EE ha sido impulsado por una comunidad activa y se mantiene como un estándar moderno para aplicaciones backend críticas.

Jakarta EE provee un ecosistema completo de APIs y servicios para construir aplicaciones distribuidas, entre ellos: JPA (Java Persistence API) para persistencia de datos, CDI (Contexts and Dependency Injection) para inyección de dependencias, JAX-WS para servicios web SOAP, JAX-RS para servicios REST, EJB (Enterprise JavaBeans), Servlets, entre otros.

1.3.2. Características

- Estandarización y mantenibilidad: Al seguir estándares oficiales de la industria, Jakarta EE promueve una arquitectura clara y mantenible, compatible con buenas prácticas de ingeniería de software. Esto permite que otros desarrolladores puedan comprender, mantener o escalar el sistema fácilmente.
- Escalabilidad empresarial: Las aplicaciones Jakarta EE pueden funcionar sin problemas desde entornos pequeños hasta servidores de alta concurrencia, gracias a

su capacidad para manejar múltiples hilos, conexiones concurrentes y gestión de sesiones.

- Ecosistema Java maduro: Existe una amplia comunidad global de desarrolladores Java, además de una gran variedad de herramientas compatibles (IDE como IntelliJ o Eclipse, frameworks, bibliotecas de seguridad, testing, monitoreo, etc.). Esto facilita la integración con tecnologías empresariales, como Oracle Database o sistemas de autenticación externa.
- Modularidad por capas: Promueve una arquitectura en capas (modelo-vista-controlador o servicio/repositorio/controlador), donde cada clase tiene una responsabilidad definida.

1.3.3. Historia y evolución

Jakarta EE nació de la transición de Java EE, luego de que Oracle transfiriera el control de la especificación a la Eclipse Foundation en 2017. Este cambio dio paso a una evolución más abierta, colaborativa y orientada a la nube.

Antes de esto, Java EE fue durante más de 15 años la plataforma estándar para construir aplicaciones empresariales en Java, especialmente en sectores donde la confiabilidad y la escalabilidad eran esenciales (banca, seguros, salud, telecomunicaciones, etc.).

Desde su renombramiento a Jakarta EE, se han introducido nuevas versiones con mejoras significativas: soporte a contenedores como Docker, integración con microservicios, despliegue en la nube, y optimización del ciclo de vida de las aplicaciones.

1.3.4. Ventajas y desventajas

Ventajas	Desventajas
Robustez y confiabilidad: Ideal para construir servicios críticos, como los requeridos en un sistema de agendamiento médico que debe operar con precisión y sin errores.	Curva de aprendizaje inicial: El uso de múltiples capas y configuraciones puede ser intimidante al principio para desarrolladores sin experiencia en backend empresarial.
Integración directa con estándares: APIs como JPA para persistencia y JAX-WS para SOAP facilitan la construcción de	Mayor complejidad inicial: La configuración de un entorno Jakarta EE

servicios bien estructurados, conectados a Oracle Database.	requiere mayor conocimiento y puede ser más pesada que frameworks más livianos.
Escalable y seguro: Proporciona mecanismos de seguridad empresarial (control de acceso, encriptación, manejo de sesiones) y soporte para despliegue en clústeres o servidores distribuidos.	Mayor configuración inicial: Aunque Jakarta EE automatiza muchas tareas, se necesita definir estructuras y convenciones antes de obtener un resultado funcional.
Soporte empresarial: Muchas empresas del sector salud ya operan sobre Java EE, lo que permite integrar o migrar sus sistemas de forma más sencilla.	Requiere servidores compatibles: Su despliegue depende de servidores de aplicaciones como GlassFish, Payara, JBoss o WildFly, lo que puede ser más complejo que frameworks embebidos.

1.3.5. Casos de uso

- Construcción del backend que gestiona usuarios, citas, historial médico, autenticación y notificaciones.
- Implementación de servicios SOAP consumidos por otros sistemas (hospitales, EPS, aseguradoras).
- Módulo administrativo para controlar los registros del sistema y el acceso de usuarios.
- Servicios independientes con potencial de escalamiento, como horarios, especialidades médicas, consultorios.

1.3.6. Casos de aplicación

- ADP (Automatic Data Processing): Empresa global de gestión de talento humano que utiliza Java EE/Jakarta EE en su infraestructura backend por su fiabilidad y capacidad transaccional.
- Gobiernos de Brasil y Colombia: Varios portales institucionales en salud, educación y justicia están construidos sobre Java EE por su robustez, estandarización y capacidad de integración con plataformas SOAP/XML.

- Servicios financieros y de salud en países como Chile, España y México, donde Jakarta EE es usado para implementar soluciones interoperables con fuerte control de seguridad y trazabilidad.

1.4. Glassfish

1.4.1. Definición

GlassFish es un servidor de aplicaciones open source diseñado para ejecutar aplicaciones empresariales basadas en Jakarta EE. Actúa como un contenedor de aplicaciones que permite desplegar y gestionar componentes empresariales como servicios web SOAP y REST, Enterprise JavaBeans (EJB), entidades JPA (Java Persistence API), Servlets y otros artefactos propios de arquitecturas Java EE/Jakarta EE. Su diseño modular y su enfoque en la compatibilidad lo convierten en una opción ideal para entornos de desarrollo, pruebas, educación y prototipado de aplicaciones empresariales.

1.4.2. Características

- Compatibilidad nativa con Jakarta EE: GlassFish es uno de los primeros servidores certificados para Jakarta EE. Soporta tecnologías como JAX-WS (para servicios SOAP), JPA (para acceso a base de datos), CDI (para inyección de dependencias) y EJB, lo que garantiza un entorno estable y conforme a los estándares.
- Herramientas administrativas integradas: Cuenta con una consola web intuitiva que permite administrar el servidor, desplegar aplicaciones .war o .ear, configurar recursos como fuentes de datos (DataSources) y ajustar parámetros de rendimiento o seguridad de manera visual.
- Integración rápida para pruebas y pilotos: Su facilidad de instalación, documentación y soporte para herramientas de desarrollo como NetBeans y Eclipse lo convierten en una excelente opción para entornos académicos y escenarios de demostración o validación inicial.
- Soporte para servicios SOAP: GlassFish puede exponer y consumir servicios web SOAP mediante JAX-WS, facilitando la interoperabilidad del sistema con otras instituciones, clínicas o entidades aseguradoras.

1.4.3. Historia y evolución

GlassFish fue desarrollado inicialmente por Sun Microsystems como el servidor de referencia para Java EE. Con la compra de Sun por parte de Oracle, fue mantenido como una opción gratuita para desarrolladores, aunque Oracle concentró sus esfuerzos comerciales en WebLogic. En 2017, Oracle transfirió la responsabilidad del proyecto GlassFish a la Eclipse Foundation, junto con la migración de Java EE a Jakarta EE.

Actualmente, GlassFish continúa existiendo como una implementación oficial y open source de Jakarta EE, aunque ha sido superado en rendimiento y estabilidad por alternativas como Payara Server (una bifurcación de GlassFish con soporte profesional) o WildFly. A pesar de ello, su facilidad de uso lo mantiene vigente en entornos educativos, pruebas de concepto y proyectos académicos como este.

1.4.4. Ventajas y desventajas

Ventajas	**Desventajas**
Integración directa con Jakarta EE: Permite ejecutar directamente las aplicaciones construidas con Jakarta EE sin requerir adaptaciones.	Estabilidad limitada en producción: GlassFish no está optimizado para cargas pesadas o aplicaciones críticas con alta concurrencia.
Despliegue ágil de aplicaciones WAR/EAR: Gracias a su consola web o interfaz de línea de comandos, se pueden desplegar y probar aplicaciones de forma casi inmediata.	Menor rendimiento comparado con otros servidores como WildFly o Payara, especialmente en ambientes con múltiples usuarios concurrentes.
Ideal para servicios SOAP y persistencia con JPA: Es compatible de forma nativa con JAX-WS y JPA, herramientas fundamentales para este proyecto.	Comunidad técnica menos activa: Aunque existen foros y documentación, su comunidad ha disminuido desde que la mayoría de los desarrolladores empresariales migraron a alternativas más modernas.
Bajo costo y open source: No requiere licencias ni configuraciones avanzadas para ambientes educativos.	Ciclo de actualizaciones más lento: Al no estar respaldado por un modelo comercial

	fuerte, sus mejoras y actualizaciones pueden tardar más en llegar.

1.4.5. Casos de uso

- Despliegue del backend desarrollado en Jakarta EE para exponer servicios SOAP que permiten registrar, consultar y modificar citas médicas.
- Plataforma de pruebas para validar las operaciones de los servicios web SOAP y su integración con el frontend (Svelte).
- Entorno académico ideal para evaluar la implementación de principios SOLID, patrones arquitectónicos y separación de responsabilidades.

1.4.6. Casos de aplicación

- Universidades y centros de formación técnica: GlassFish ha sido ampliamente utilizado como plataforma de enseñanza en materias de arquitectura de software, ingeniería de software, programación distribuida y servicios web.
- Proyectos prototipo en instituciones de salud pública: Algunas secretarías de salud en Latinoamérica han utilizado GlassFish en proyectos piloto o fases de prueba antes de pasar a entornos de producción en servidores como JBoss o Payara.
- Casos históricos en Oracle: Durante años, Oracle utilizó GlassFish como servidor de pruebas para aplicaciones Java EE y para desarrollar las implementaciones de referencia de muchas especificaciones Java.

1.5. SOAP

1.5.1. Definición

SOAP (Simple Object Access Protocol) es un protocolo estándar para el intercambio de mensajes estructurados entre aplicaciones a través de redes, comúnmente sobre el protocolo HTTP o SMTP. Basado en XML, SOAP define cómo deben formatearse y transmitirse los mensajes para invocar métodos de servicios web, garantizar interoperabilidad y asegurar confiabilidad. A través del uso de WSDL (Web Services Description Language), se establecen contratos formales entre consumidor y proveedor, lo que garantiza que ambas partes conozcan de antemano la estructura exacta de los mensajes que serán intercambiados.

1.5.2. Características

- Estructura XML validada y estandarizada: Todos los mensajes SOAP están formateados en XML, lo que permite validarlos fácilmente contra esquemas (XSD). Esto evita errores de interpretación entre el consumidor (por ejemplo, el frontend Svelte) y el proveedor (el backend Jakarta EE).
- Uso de WSDL (Web Services Description Language): Permite definir contratos de servicios de forma explícita, especificando los métodos disponibles, los tipos de datos, las operaciones y sus parámetros. Esto es crucial para garantizar la interoperabilidad entre sistemas heterogéneos.
- Seguridad WS-Security: SOAP incluye especificaciones para cifrado de mensajes, firmas digitales y control de acceso basado en tokens, lo que resulta útil si el sistema evoluciona hacia entornos más regulados o si se conecta con entidades externas que exigen seguridad avanzada.
- Transaccionalidad y confiabilidad: SOAP puede funcionar con mecanismos de entrega garantizada, transacciones distribuidas y almacenamiento persistente de mensajes, asegurando la integridad de los procesos críticos.

1.5.3. Historia y evolución

SOAP fue introducido oficialmente en 1999 por Microsoft, IBM y otros actores de la industria como parte de la estrategia de integración de sistemas heterogéneos sobre internet. Durante más de una década, fue el estándar dominante para servicios web empresariales, especialmente en sectores como banca, salud, telecomunicaciones y administración pública.

Con el auge del desarrollo ágil, REST (Representational State Transfer) y formatos livianos como JSON comenzaron a desplazar a SOAP en aplicaciones móviles y web modernas. Sin embargo, SOAP se mantiene vigente en sectores donde la robustez, trazabilidad, compatibilidad con sistemas legados y cumplimiento de estándares son prioridades.

1.5.4. Ventajas y desventajas

Ventajas	**Desventajas**
Contratos formales (WSDL): Permiten definir y documentar los servicios de manera estructurada y validable, lo que es	Uso de lenguaje: El uso de XML incrementa el tamaño de los mensajes y puede hacer más lentas las respuestas en

ideal en entornos donde se requiere precisión en la comunicación.	comparación con alternativas más ligeras como REST+JSON.
Seguridad y transaccionalidad: SOAP permite aplicar WS-Security, integridad de mensajes, autenticación y firmas digitales, lo cual es indispensable si el sistema se extiende a interoperar con EPS, clínicas o entidades gubernamentales.	Complejidad de integración con frontend moderno: El consumo de servicios SOAP en clientes frontend requiere el manejo de XML y a menudo bibliotecas adicionales, lo que puede ser menos directo que trabajar con APIs REST.
Ideal para entornos regulados: SOAP es ampliamente utilizado en entornos que exigen cumplimiento de normas como HL7, ISO/IEC 27001, o leyes de protección de datos como la Ley 1581 en Colombia o el GDPR en Europa.	Requiere herramientas de desarrollo y pruebas especializadas: Para probar y consumir servicios SOAP, se requieren herramientas como SoapUI, WSDL parsers o clientes personalizados, lo que aumenta la complejidad del desarrollo.
Independencia de plataforma: Puede ser implementado en cualquier lenguaje (Java, .NET, Python) y soportado por múltiples herramientas (Postman, SoapUI, etc.).	Menor popularidad en proyectos modernos: La mayoría de frameworks frontend y backend actuales favorecen REST, lo que puede limitar la oferta de ejemplos, documentación y recursos actualizados.

1.5.5. Casos de uso

- Servicios clínicos que deben cumplir con normativas de interoperabilidad, trazabilidad y seguridad de la información médica.
- Integración con sistemas externos como EPS o registros nacionales de pacientes, donde se requiere el envío estructurado de información (XML) con validación estricta.
- Comunicación entre módulos internos del sistema que requieren precisión transaccional, como la asignación de citas en horarios restringidos o autorizaciones por parte de entidades externas.

1.5.6. Casos de aplicación

Sistemas internos de Oracle: SOAP sigue siendo ampliamente usado en herramientas empresariales como Oracle Financials, PeopleSoft y Siebel, que requieren integraciones precisas y robustas.

EPS y aseguradoras en Colombia y España: Muchas entidades utilizan servicios SOAP para el intercambio de información clínica, facturación de servicios de salud y autorización de tratamientos, ya que cumplen con los estándares de interoperabilidad definidos por los ministerios de salud.

Red Nacional de Salud de España (SNS): Intercambia datos clínicos y administrativos entre comunidades autónomas mediante servicios SOAP bajo el estándar HL7 v3.

1.6. Oracle Database

1.6.1. Definición

Definición
Oracle Database es un sistema gestor de bases de datos relacional (RDBMS) de nivel empresarial, desarrollado por Oracle Corporation, que permite el almacenamiento, gestión, consulta y protección de grandes volúmenes de datos con alta eficiencia, seguridad y disponibilidad. Se trata de una de las plataformas de bases de datos más potentes y utilizadas a nivel mundial, ampliamente reconocida por su capacidad para operar en entornos críticos y de alto rendimiento.
Basado en el lenguaje SQL y su extensión propietaria PL/SQL, Oracle Database soporta operaciones transaccionales, procesamiento analítico, replicación de datos, auditoría, cifrado, clustering, y gestión avanzada de usuarios y permisos, lo que lo convierte en una solución robusta para sistemas que manejan información sensible o altamente estructurada.

1.6.2. Características

- Alta disponibilidad y tolerancia a fallos: Oracle cuenta con tecnologías como Real Application Clusters (RAC) y Data Guard que permiten que el sistema esté disponible incluso frente a fallos de hardware o software, una capacidad indispensable cuando se gestionan citas médicas y datos de salud críticos.
- Seguridad y auditoría: El sistema ofrece controles avanzados para la autenticación, el cifrado de datos, la gestión de roles y privilegios, así como el registro detallado

de auditorías (quién accedió, cuándo y a qué datos). Esto es especialmente relevante en entornos donde se debe cumplir con normativas de protección de datos, como la Ley 1581 en Colombia o el GDPR en Europa.

- Optimización avanzada de consultas: El motor de consultas SQL de Oracle permite índices, particiones, vistas materializadas y otras optimizaciones automáticas o personalizadas que mejoran la eficiencia en la recuperación de datos, vital cuando se realizan búsquedas complejas o filtros por fecha, especialidad médica o disponibilidad.
- Escalabilidad horizontal y vertical: Oracle está diseñado para funcionar tanto en arquitecturas monolíticas como en entornos distribuidos, facilitando el crecimiento del sistema a medida que aumenta el número de usuarios, transacciones y datos almacenados.
- Integración con Jakarta EE: Existe una compatibilidad directa entre Oracle Database y los entornos Java empresariales, permitiendo que los servicios backend desarrollados con Jakarta EE utilicen JPA (Java Persistence API) para mapear las entidades del sistema con la base de datos de forma eficiente y segura.

1.6.3. Historia y evolución

Oracle Database fue lanzado en 1979, posicionándose como uno de los primeros sistemas de bases de datos comerciales en implementar el modelo relacional propuesto por Edgar F. Codd. Desde entonces, ha evolucionado hacia un sistema híbrido que combina procesamiento transaccional (OLTP), analítico (OLAP) y capacidades de inteligencia artificial.

Con el auge de la computación en la nube, Oracle ha desarrollado soluciones como Oracle Autonomous Database, que automatiza la administración, optimización y seguridad de la base de datos usando inteligencia artificial. También ofrece Oracle Cloud Infrastructure (OCI), que permite desplegar bases de datos en entornos híbridos, multicloud o 100% en la nube.

1.6.4. Ventajas y desventajas

Ventajas	**Desventajas**

Alta confiabilidad y consistencia: Ideal para sistemas donde los datos no pueden perderse ni duplicarse, como la gestión de pacientes y citas médicas.	Costo de licenciamiento: Oracle Database requiere licencias comerciales que pueden representar una inversión considerable, especialmente para instituciones pequeñas.
Seguridad robusta: Incluye mecanismos de control de acceso, cifrado en reposo y en tránsito, y monitoreo de eventos sospechosos.	Requiere hardware robusto: Aunque existen versiones gratuitas como Oracle XE, su uso empresarial óptimo exige infraestructura especializada o soluciones en la nube.
Rendimiento superior en consultas: El sistema puede ser optimizado para consultas complejas, como buscar la disponibilidad de un médico en rangos de fechas, calcular el número de citas por especialidad, o emitir reportes administrativos.	Curva de aprendizaje para PL/SQL: Aunque es un lenguaje muy potente, PL/SQL tiene particularidades que exigen conocimiento específico y experiencia previa para su dominio efectivo.
Escalabilidad: Capaz de crecer con el sistema sin sacrificar el rendimiento, lo cual es crucial en proyectos con proyección nacional o institucional.	

1.6.5. Casos de uso

- Gestión de usuarios, pacientes, citas médicas, horarios y agendas con control de transacciones, validación de datos y persistencia confiable.
- Registro de auditoría médica: Quién consultó, modificó o eliminó información clínica, con trazabilidad completa.
- Almacenamiento de datos clínicos sensibles: Historial médico, resultados de laboratorio, documentos adjuntos (consentimientos informados, órdenes médicas).
- Generación de reportes administrativos: Estadísticas de uso del sistema, número de citas por día, disponibilidad de especialistas, tiempo promedio de espera, entre otros.

1.6.6. Casos de aplicación

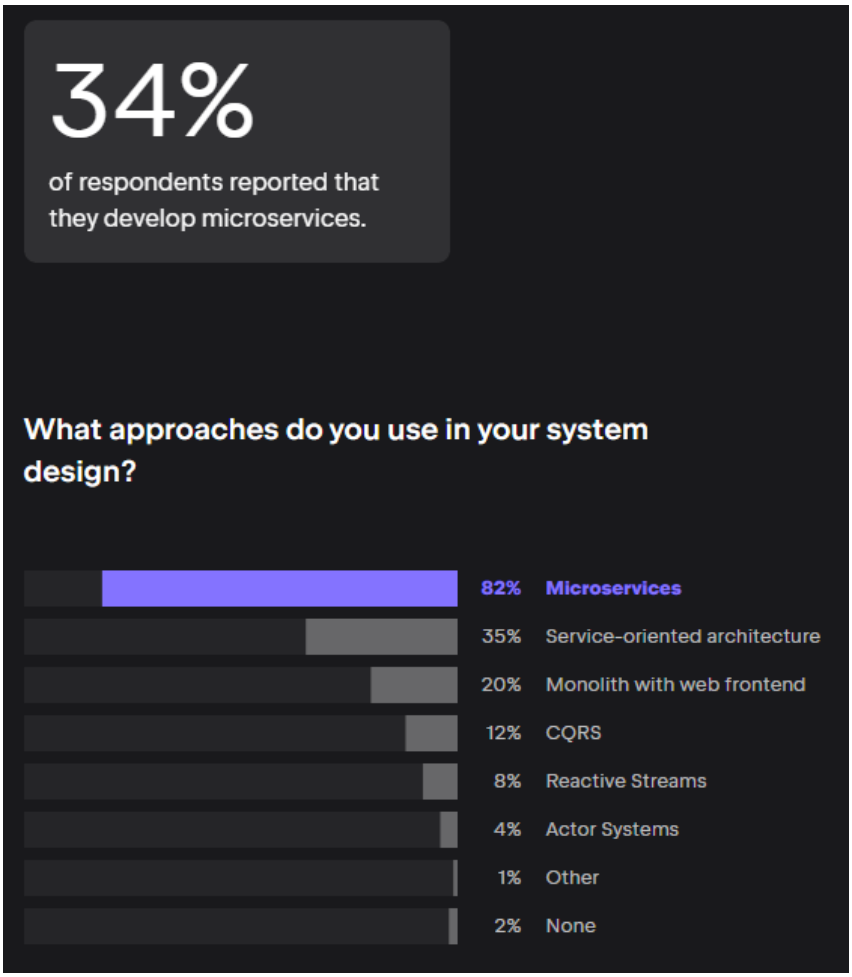
- Amazon, PayPal, Boeing: Empresas que manejan millones de transacciones por segundo confían en Oracle Database para operaciones críticas, gracias a su seguridad y confiabilidad.
- Hospitales y clínicas de alto nivel en Estados Unidos y Europa: Utilizan Oracle como base para sus sistemas HIS (Hospital Information Systems) y EHR (Electronic Health Records).
- EPS y aseguradoras en Colombia: Algunas entidades utilizan Oracle para almacenar información de afiliados, gestionar autorizaciones, auditorías médicas y reportes de facturación.

2. Relación entre los temas asignados

2.1. Que tan común es el stack

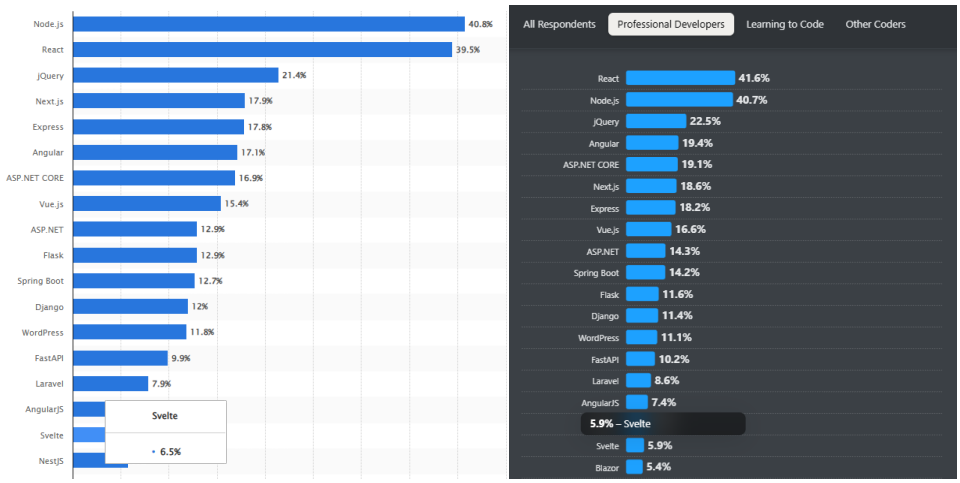
2.1.1. Arquitectura Orientada a Servicios (SOA)

Aunque la Arquitectura Orientada a Servicios (SOA) ya no domina el panorama como lo hacía entre 2005 y 2015, aún mantiene cierta presencia. Según datos recientes, el 35% de los encuestados indican que utilizan SOA en sus diseños de sistema, lo que demuestra que, si bien ha sido ampliamente superada por enfoques como los microservicios (82%), no ha desaparecido del todo (JetBrains, 2023). Su uso persiste especialmente en sistemas legacy o entornos donde la interoperabilidad entre servicios sigue siendo una prioridad.

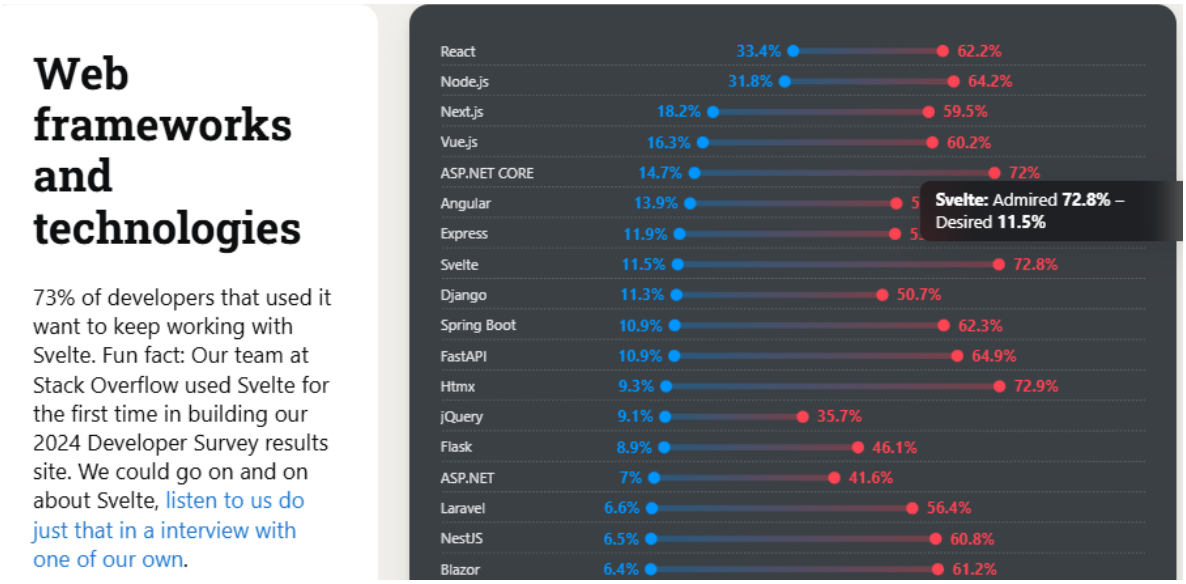


2.1.2. Svelte

Svelte ha ganado notable atención en la comunidad de desarrollo frontend. Aunque su adopción actual es moderada (alrededor del 5.9% al 6.5% según encuestas de Stack Overflow y Statista 2024), su nivel de satisfacción es excepcionalmente alto: el 72.8% de quienes lo usan desean seguir trabajando con él, posicionándolo entre los frameworks más admirados. Esto refleja una percepción positiva y un potencial de crecimiento considerable, aunque aún no alcanza el volumen de uso de tecnologías como React, Node.js o Vue.js.



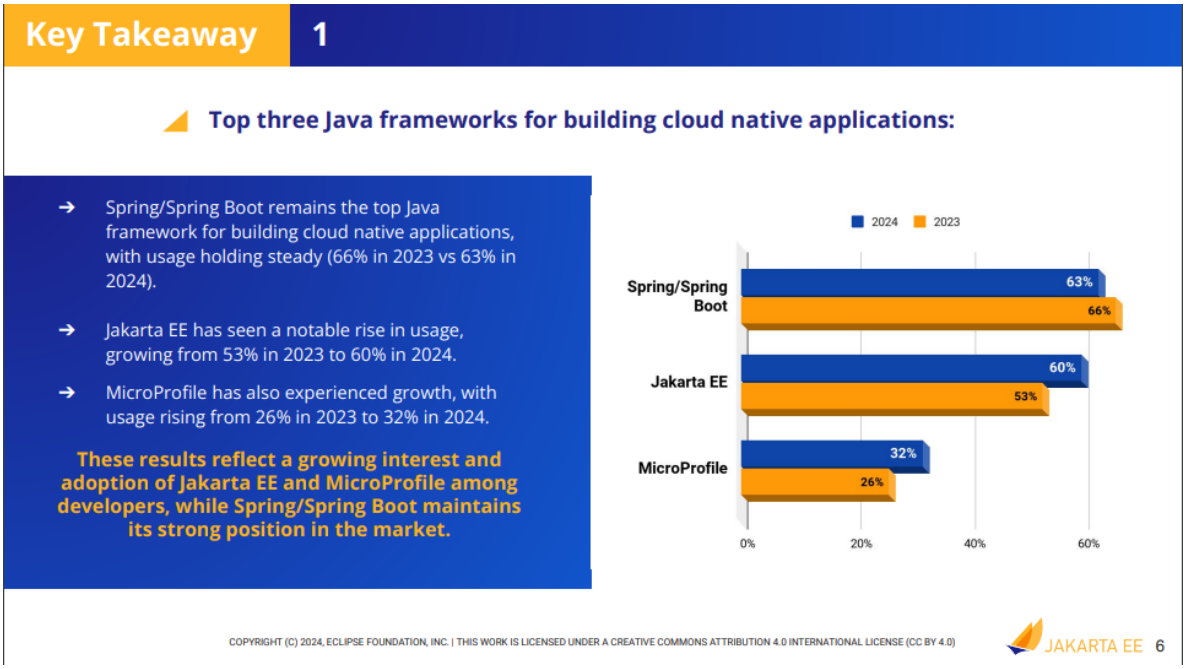
A pesar de que Svelte todavía no figura entre los 10 lenguajes o frameworks con mayor número de contribuciones, reflejando que aunque su crecimiento es fuerte, todavía es una tecnología en expansión.



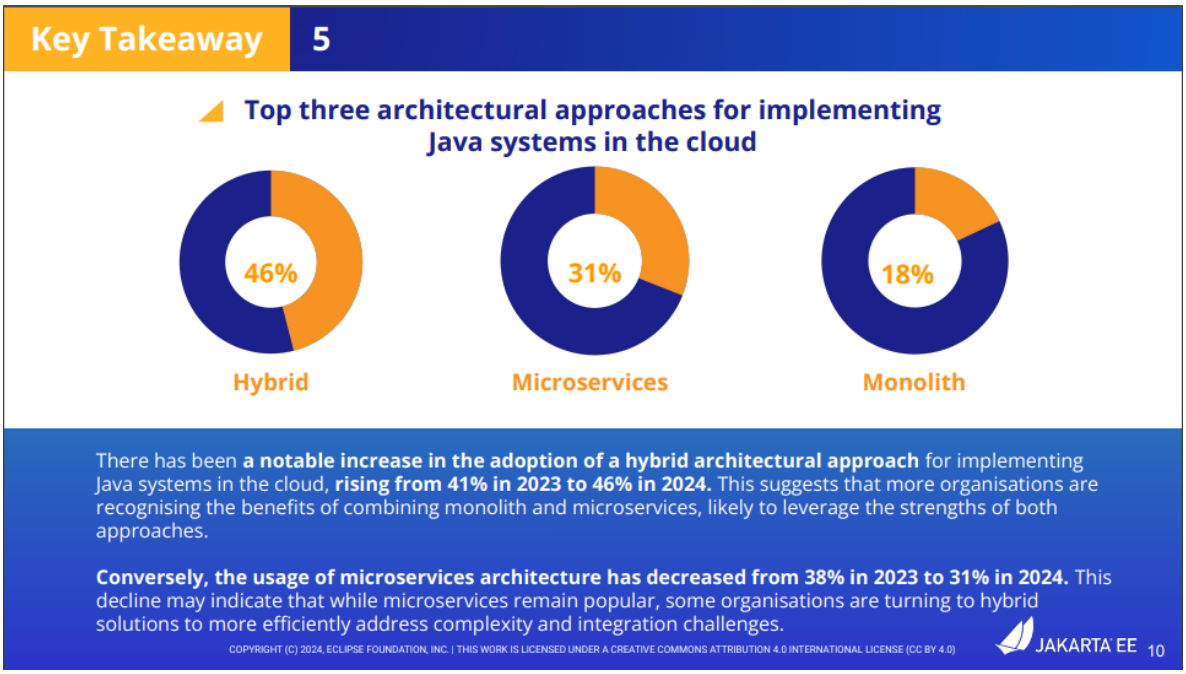
2.1.3. Jakarta EE

Jakarta EE (previamente Java EE) continúa siendo una tecnología muy relevante, especialmente en contextos modernos de aplicaciones cloud-native. Según la encuesta oficial de la Fundación Eclipse (2024 Jakarta EE Developer Survey Report), su adopción ha aumentado notablemente, pasando del 53% en 2023 al 60% en 2024. Además, se ha duplicado la adopción de su versión más reciente, Jakarta EE 10, indicando claramente una

tendencia hacia la actualización tecnológica. Esto refleja un interés creciente por aprovechar las innovaciones recientes de Java SE, como Records y Virtual Threads.



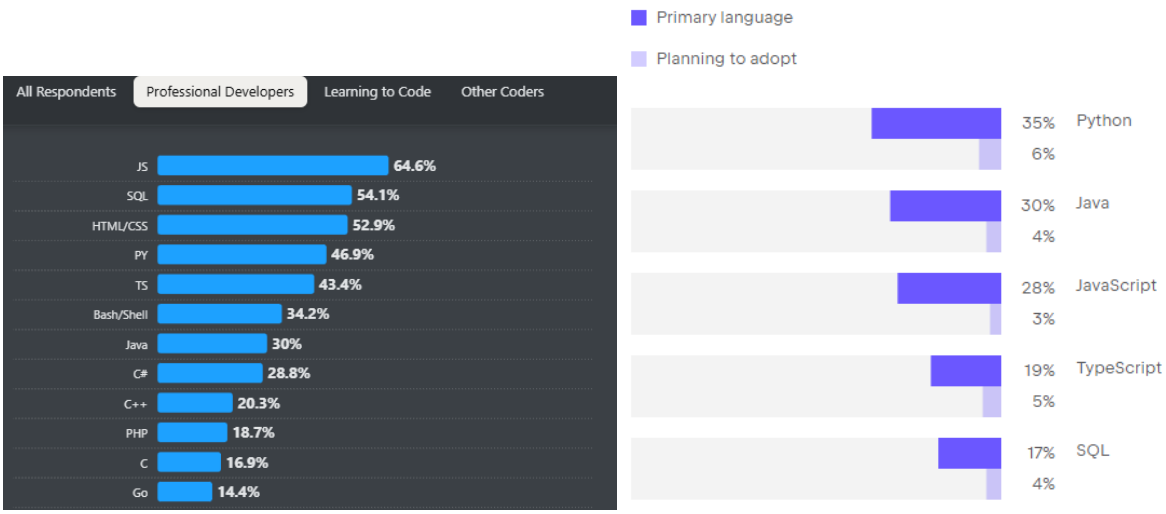
La adopción de enfoques híbridos para implementar sistemas Java en la nube ha crecido del 41% en 2023 al 46% en 2024. En cambio, el uso de arquitecturas basadas únicamente en microservicios ha disminuido del 38% al 31%, sugiriendo una preferencia por soluciones que combinan ambos modelos para manejar mejor la complejidad y los desafíos de integración.



Jakarta EE actualmente atraviesa un momento de consolidación y crecimiento significativo, destacándose especialmente en entornos cloud-native. Su adopción ha aumentado considerablemente, impulsada por la modernización constante y su capacidad para integrarse con innovaciones recientes de Java. Mirando hacia el futuro, la tendencia indica que Jakarta EE seguirá siendo relevante para los desarrolladores y empresas.

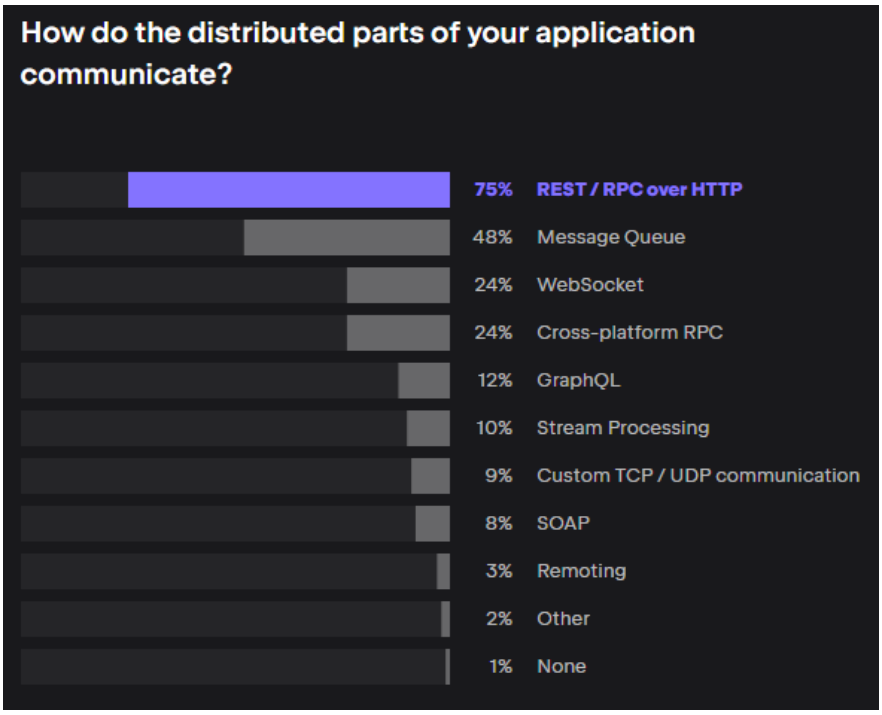
Java continúa siendo un lenguaje sólido y ampliamente utilizado entre los desarrolladores profesionales, con un 30% de adopción (Stack Overflow, 2024) y (Jetbrains, 2024). Además, es el segundo lenguaje más mencionado como primario en entornos empresariales, y un 4% adicional planea adoptarlo próximamente, lo que demuestra que su vigencia no ha decaído.

Este uso sostenido impulsa también la relevancia de tecnologías basadas en Java, como Jakarta EE. El fuerte respaldo que aún mantiene Java garantiza un ecosistema activo y en evolución para frameworks y plataformas empresariales, consolidando a Jakarta EE como una opción robusta y preparada para enfrentar los desafíos actuales del desarrollo cloud-native y la modernización de aplicaciones legacy.



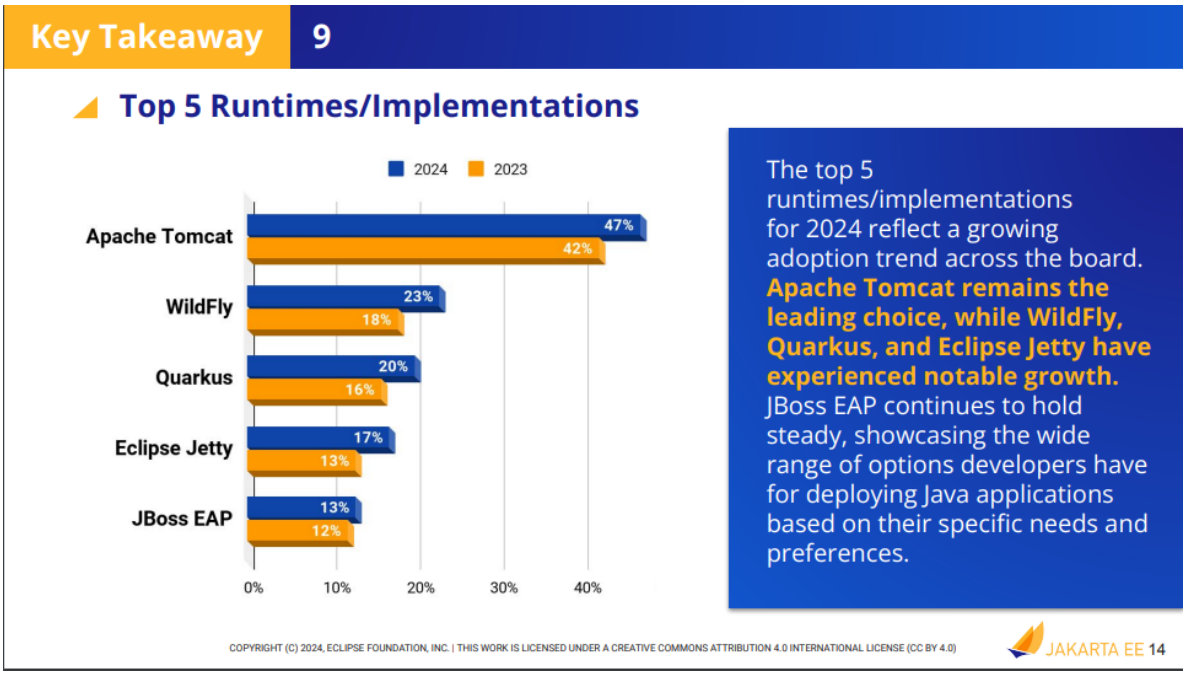
2.1.4. SOAP

SSOAP, aunque fue el estándar dominante para servicios web empresariales durante muchos años, ha quedado relegado en la actualidad. Solo el 8% de los encuestados reporta usarlo como método de comunicación entre partes distribuidas de sus aplicaciones, frente al abrumador 75% que utiliza REST/RPC sobre HTTP (JetBrains, 2023). Esta diferencia refleja una clara preferencia por soluciones más ligeras y flexibles. SOAP persiste principalmente en sistemas legacy, especialmente en sectores regulados como el gubernamental o financiero, donde aún se valora su robustez y soporte para transacciones complejas.



2.1.5. GlassFish

GlassFish, un servidor de aplicaciones compatible con Jakarta EE, tuvo una alta popularidad alrededor del 2010-2015. Hoy en día, su relevancia ha disminuido considerablemente, no figura en las encuestas recientes ni en las tendencias actuales de GitHub, JetBrains o Stack Overflow.



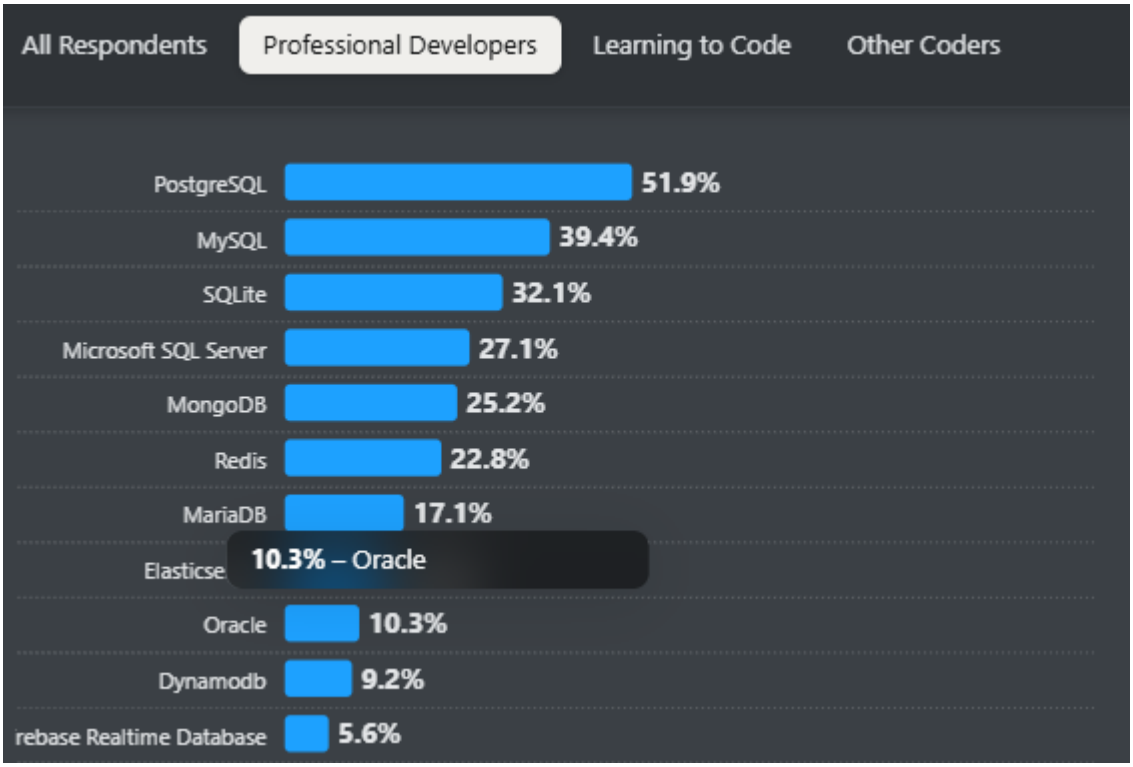
En el panorama actual de runtimes compatibles con Jakarta EE, Apache Tomcat lidera con amplia ventaja, seguido por implementaciones como WildFly, Quarkus y Eclipse Jetty, todas con un crecimiento destacado en 2024. Aunque JBoss EAP se mantiene estable, refleja la diversidad de opciones disponibles para los desarrolladores según sus necesidades específicas.

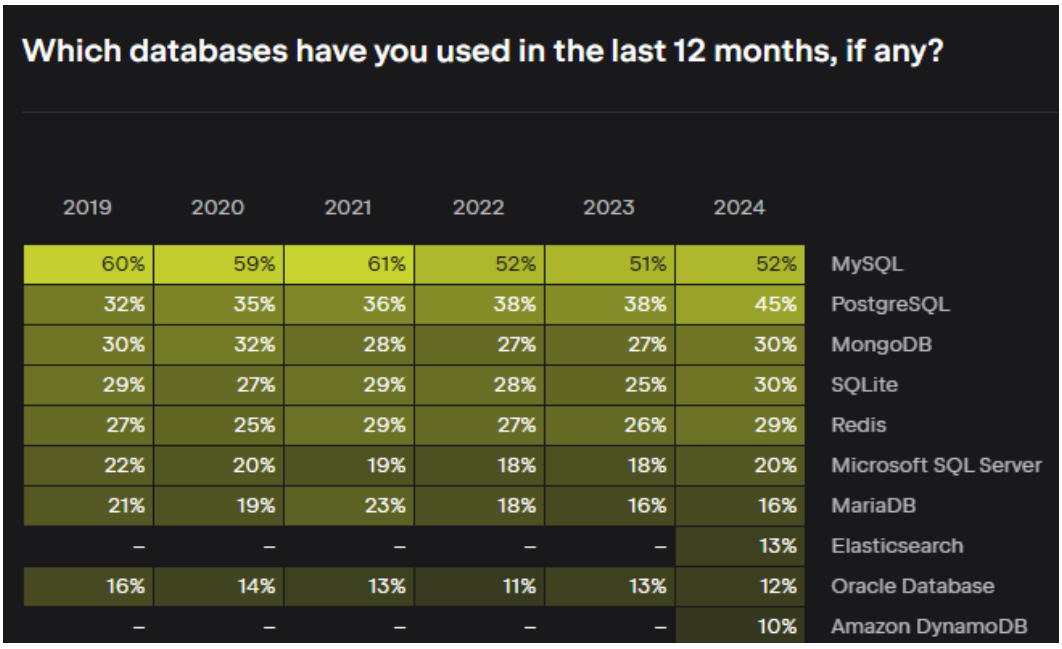
En contraste con el declive de GlassFish, Payara, un fork de este, se ha consolidado como su sucesor natural, manteniéndose relevante especialmente en entornos empresariales que buscan continuidad con tecnologías Jakarta EE. Aunque no aparece entre los cinco más usados según esta encuesta, su adopción sigue firme gracias a su enfoque en soporte a largo plazo, estabilidad y actualizaciones frecuentes, posicionándose como una alternativa moderna y confiable dentro del ecosistema.

2.1.6. Oracle DB

Oracle Database continúa siendo una opción válida en entornos empresariales que priorizan rendimiento, seguridad y disponibilidad. Sin embargo, su presencia en el ecosistema de desarrollo ha disminuido frente al crecimiento sostenido de alternativas open-source como PostgreSQL y MySQL.

Según la encuesta de Stack Overflow 2024, solo un 10.3% de los desarrolladores profesionales reporta usar Oracle, en comparación con un 51.9% que utiliza PostgreSQL y un 39.4% que usa MySQL. De forma similar, los datos de JetBrains muestran a Oracle en décimo lugar con un 12% de uso. Esto confirma que, aunque su adopción es más limitada, Oracle mantiene una posición estable en nichos específicos donde sus características empresariales siguen siendo valoradas.





2019	2020	2021	2022	2023	2024	
60%	59%	61%	52%	51%	52%	MySQL
32%	35%	36%	38%	38%	45%	PostgreSQL
30%	32%	28%	27%	27%	30%	MongoDB
29%	27%	29%	28%	25%	30%	SQLite
27%	25%	29%	27%	26%	29%	Redis
22%	20%	19%	18%	18%	20%	Microsoft SQL Server
21%	19%	23%	18%	16%	16%	MariaDB
–	–	–	–	–	13%	Elasticsearch
16%	14%	13%	11%	13%	12%	Oracle Database
–	–	–	–	–	10%	Amazon DynamoDB

2.2. Matriz de análisis de Principios SOLID vs Temas

Tecnología / Principio	**S** (Responsabilidad Única)	**O** (Abierto/Cerrado)	**L** (Sustitución de Liskov)	**I** (Segregación de Interfaces)	**D** (Inversión de Dependencias)
SOA	Alta separación entre servicios (usuarios, citas). Cada uno tiene una sola responsabilidad clara.	Servicios pueden extenderse con nuevas funciones sin alterar los existentes.	Las interfaces de servicio pueden extenderse sin romper la compatibilidad.	Cada servicio ofrece interfaces específicas (WSDL) según su función.	Servicios consumen y exponen interfaces sin conocer su implementación concreta.
Svelte	Componentes UI por vista/página; cada uno con responsabilidad única.	Componentes pueden ser reutilizados/extensibles.	Props y eventos pueden ser redefinidos sin romper estructura.	Alta cohesión: cada componente gestiona una función visual específica.	Se apoya en stores o contextos para invertir dependencias visuales.
Jakarta EE	Beans y servicios separados (UserService, AppointmentService).	Permite heredar servicios y decorarlos con lógica adicional.	Beans y repositorios siguen interfaces que pueden intercambiarse	Interfaces JAX-WS/JPA bien definidas, cada una con su rol.	Usa CDI para invertir dependencias entre capas.
GlassFish	Servidor gestiona cada servicio por separado.	Puede desplegar múltiples aplicaciones sin interferencias.	Soporta sustitución de EAR/WAR que respeten contratos.	Gestiona módulos separados de forma aislada.	Permite inyección de dependencias por Jakarta EE.
SOAP	Cada endpoint responde a una función (getCita, getUsuario).	Nuevas operaciones pueden añadirse sin afectar las existentes.	Contratos WSDL permiten compatibilidad futura.	Cada operación es independiente; bien segregadas.	Contratos y bindings se consumen sin conocer su lógica interna.
Oracle DB	Tablas bien diseñadas reflejan	Se puede extender con nuevas tablas y	Vistas e índices pueden sustituirse sin	Las consultas están	JDBC/JPA abstraen la lógica de

	responsabilidades únicas (usuarios, citas).	relaciones sin afectar las actuales.	romper consultas.	optimizadas por entidad.	acceso; la lógica no depende directamente de SQL

2.3. Matriz de análisis de Atributos de Calidad vs Temas

Tecnología / Principio	AF	ED	CO	US	FI	SE	MA	PO
SOA	Alta (Servicios bien definidos por función)	Media (Overhead SOAP)	Alta (interoperabilidad)	N/A (backend)	Alta	Alta (con WS-Security)	Alta (servicios modulares)	Alta
Svelte	Alta (interfaces reactivas y funcionales)	Alta (compila a JS nativo)	Media (requiere adaptar consumo SOAP)	Alta (UX moderna)	Alta (fallos aislados a componentes)	Media	Alta (componentes simples)	Alta (SPA)
Jakarta EE	Alta (modularización por capas y servicios)	Media (ligeramente pesado)	Alta (soporte de múltiples APIs)	N/A	Alta	Alta (soporte JAAS, SSL)	Alta (estructura por beans/repos)	Media
Glass Fish	Alta (implementa full Jakarta EE)	Media-baja (no óptimo en producción)	Alta	N/A	Media (requiere buena configuración)	Alta	Media (menos soporte que otros servidores)	Baja
SOAP	Alta (contratos	Media-baja	Alta (ampliamente	N/A	Alta (transmisión	Alta (WS-Security,	Alta (servicios aislados)	Alta

	WSDL precisos)	(verbose, lento)	soportado)		estructur ada)	XMLSi g)		
Oracl e DB	Alta (integrida d, ACID)	Alta (procesami ento optimizado)	Media (uso de drivers específic os)	N/A	Alta (resisten cia a fallos)	Alta (control de roles, auditorí a)	Media (configura ción compleja)	Media

- Adecuación Funcional (AF): El sistema cumple correctamente con los requisitos funcionales.
- Eficiencia de Desempeño (ED): Tiempo de respuesta, uso de recursos.
- Compatibilidad (CO): Capacidad de trabajar con otros sistemas.
- Usabilidad (US): Facilidad de uso para los usuarios finales.
- Fiabilidad (FI): Comportamiento estable y sin fallos.
- Seguridad (SE): Protección contra acceso no autorizado.
- Mantenibilidad (MA): Facilidad para corregir errores o mejorar.
- Portabilidad (PO): Capacidad para adaptarse a distintos entornos.

2.4. Matriz de análisis de Tácticas vs Temas

Tecnolo gía / Principi o	**RA**	**SR**	**MD**	**GE**	**CA**	**BE**	**TF**	**IO**
SOA	Alta (servicios independi entes)	Alta (servicios modulares por función)	Media (requiere herramie ntas externas)	Alta (gestión de errores por servicio)	Alta (seguri dad a nivel de servicio)	Media (a través de orquestac ión)	Alta (si falla un servicio , no cae todo)	Alta
Svelte	Alta (compone ntes desacopla dos)	Alta (UI por módulos)	Media (requiere integració n con herramie	Alta (catch de errores UI)	Baja (seguri dad delegad a al	Media (SPA optimiza carga)	Media	Media

			ntas de log)		backend)			
Jakarta EE	Alta (inyección de dependencias CDI)	Alta (beans separados por capas)	Alta (APIs de logging y auditoría)	Alta (control por excepción)	Alta (JAAS, roles, filtros)	Alta (clusterización posible)	Alta (soporte a múltiples instancias)	Alta
Glass Fish	Media (configuración modular)	Alta (soporta módulos WAR)	Media (consola de administración básica)	Media (requiere configuración)	Alta (configuración de seguridad)	Media-baja (no recomendado para producción masiva)	Media	Alta
SOAP	Alta (cada operaciones independiente)	Alta (WSDL por funcionalidad)	Baja (complejo de monitorear)	Alta (manejo de errores estándar)	Alta (WS-Security)	Baja	Alta (respuesta predecible)	Alta
Oracle DB	Alta (modelo entidad-relación claro)	Alta (consultas por entidad)	Alta (auditorías, triggers)	Alta (integridad referencial, transacciones)	Alta (roles, privilegios)	Alta (RAC, particionamiento)	Alta (backup, recuperación)	Media (requiere drivers específicos)

- Reducción del acoplamiento (RA)
- Separación de responsabilidades (SR)
- Monitoreo y diagnósticos (MD)
- Gestión de errores (GE)
- Control de acceso (CA)
- Balanceo de carga y escalabilidad (BE)
- Tolerancia a fallos (TF)
- Interoperabilidad (IO)

2.5. Matriz de análisis de Patrones vs Temas

Tecnología / Principio	MVC	DAO	DTO	Service Layer	Proxy/ Gateway	Observer/Reactive	Facade	Orquestación
SOA	N/A	N/A	Alta (envío de datos estructurados)	Alta (cada servicio actúa como capa intermedia)	Alta (a través de WSDL y bindings)	N/A	Alta (servicios simplifican procesos)	Alta (orquestación de múltiples servicios)
Svelte	Alta (estructura UI clara)	N/A	Media (usa objetos de datos desde el backend)	N/A	Media (puede implementar fetch wrappers)	Alta (reactividad, bindings)	N/A	N/A
Jakarta EE	Alta (Controladores, Servicios, DAOs)	Alta (repositorios JPA)	Alta (uso de DTOs entre capas)	Alta (Servicios entre endpoints y lógica)	Media (endpoints SOAP actúan como gateway)	N/A	Alta (Beans centralizan lógica)	Media (posible con EJB y Jakarta Batch)
Glass Fish	Alta (facilita despliegue de MVC en Jakarta EE)	Alta	Alta	Alta	Media	N/A	Media	Media
SOAP	N/A	N/A	Alta (estructura XML definida)	Alta (cada operación expone una función)	Alta (definición formal de interfaces)	N/A	Alta (expone solo lo necesario al consumidor)	

Oracle DB	N/A	Alta (acceso a datos con DAOs)	Media (requiere mapeo externo)	N/A	N/A	N/A	N/A	N/A

- MVC (Modelo-Vista-Controlador)
- DAO (Data Access Object)
- DTO (Data Transfer Object)
- Service Layer
- Proxy o Gateway
- Observer/Reactive UI
- Facade
- Orquestación de Servicios

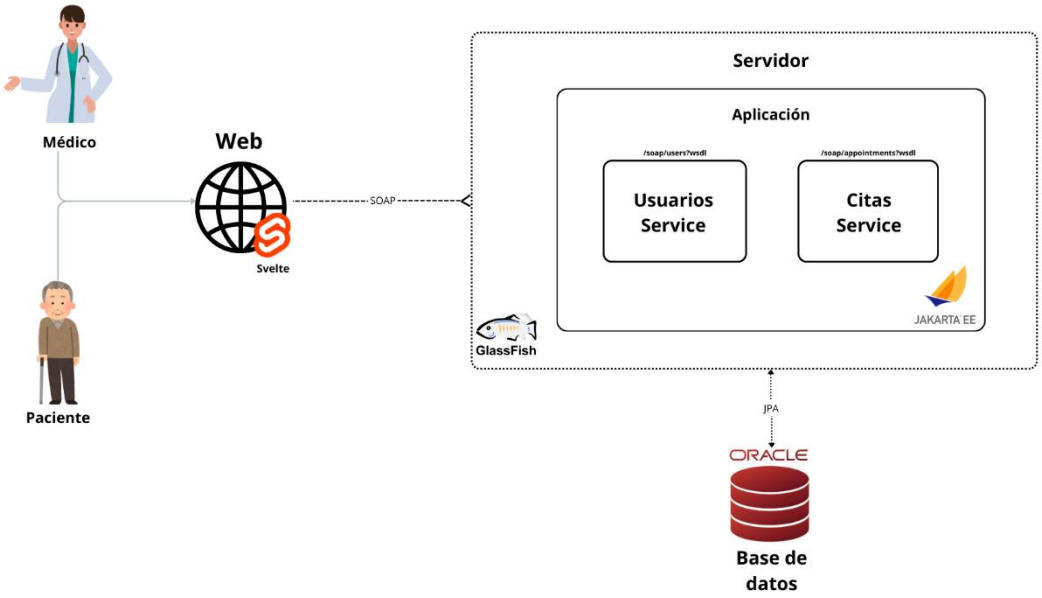
2.6. Matriz de análisis de Mercado Laboral vs Temas

Tecnología / Principio	**Demanda Laboral**	**Adopción en la Industria**	**Madurez**	**Proyección de crecimiento**	**Comunidad / Soporte**
SOA	Media (presente en grandes empresas, especialmente legacy)	Media-Alta (entornos bancarios, salud, gobierno)	Alta (consolidada)	Baja (reemplazada en muchos casos por microservicios)	Media (documentación formal, pero en declive activo)
Svelte	Media-Baja (nicho creciente, aún no tan demandado como React)	Baja-Media (usado en startups o proyectos personales)	Media (aún en expansión)	Alta (rápido crecimiento y satisfacción de desarrolladores)	Alta (comunidad muy activa y entusiasta)

Jakarta EE	Alta (muy usado en empresas grandes que usan Java)	Alta (corporativo, financiero, gubernamental)	Alta (evolución de Java EE)	Media (se mantiene vigente, pero no disruptiva)	Alta (respaldo de Eclipse Foundation)
GlassFish	Baja (no recomendado para producción moderna)	Baja (usado en educación y pruebas)	Media (estable pero superado por otros)	Baja (reemplazado por Payara, WildFly, etc.)	Media (existe documentación, pero comunidad limitada)
SOAP	Media (activo en industrias reguladas)	Alta (sector salud, banca, aseguradoras)	Alta (protocolo robusto y estandarizado)	Baja (en desuso frente a REST/GraphQL)	Alta (soporte continuo en plataformas empresariales)
Oracle DB	Alta (presente en grandes corporativos, gobiernos)	Alta (infraestructura crítica, ERP, CRM)	Alta (consolidada desde hace décadas)	Media (mantiene cuota pero compite con PostgreSQL)	Alta (documentación, soporte empresarial, certificaciones)

3. Ejemplo práctico

3.1. Diagrama Alto Nivel

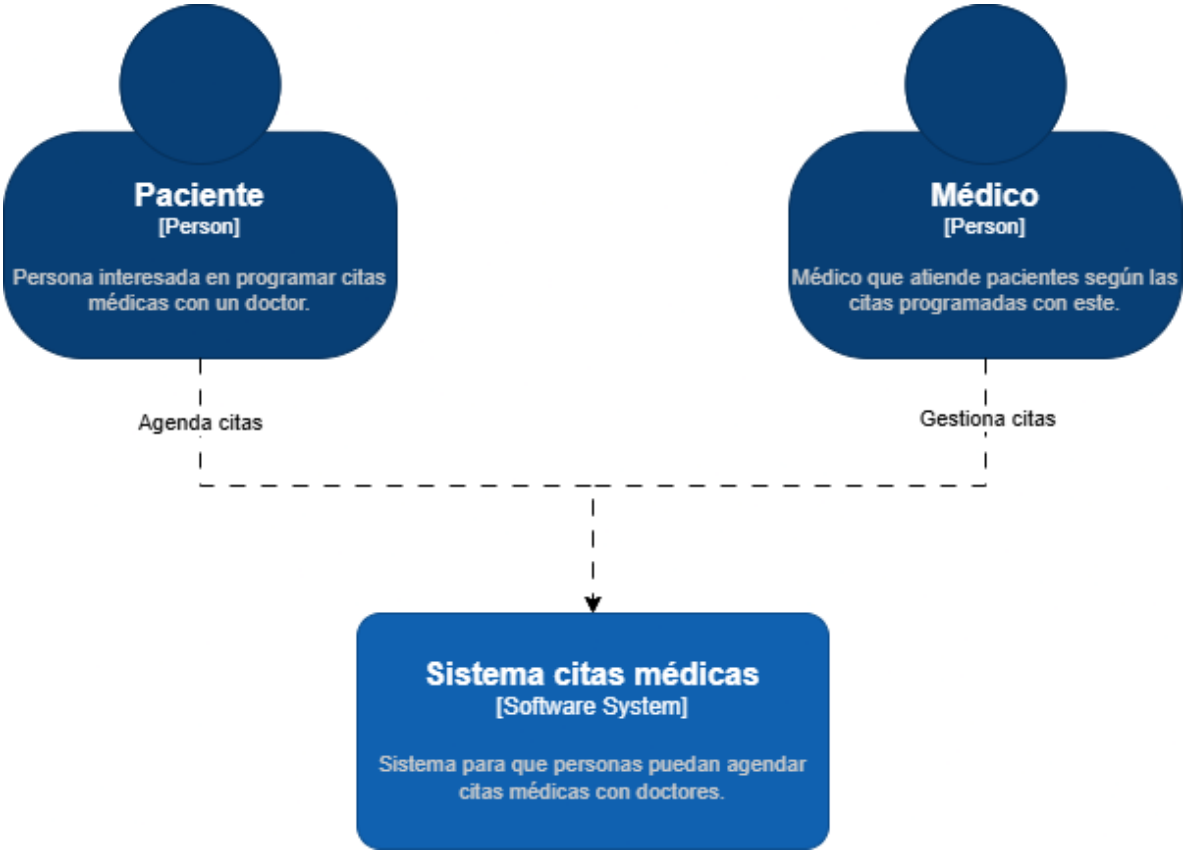


Esta es la visión general de la arquitectura del sistema, representando la interacción entre los distintos elementos involucrados. Los usuarios del sistema, representados por los **pacientes** y los **médicos**, acceden a una **aplicación web desarrollada en Svelte**, que actúa como interfaz de usuario. Esta aplicación se comunica mediante el protocolo **SOAP** con una aplicación del lado servidor construida sobre **Jakarta EE** y desplegada en un servidor **GlassFish**.

Dentro del servidor se encuentran definidos dos servicios principales: el **Usuarios Service** y el **Citas Service**, cada uno accesible a través de un endpoint SOAP. Ambos servicios manejan las operaciones correspondientes a la gestión de usuarios y citas médicas. Finalmente, toda la información se almacena de forma persistente en una **base de datos Oracle**, a la cual se accede mediante **JPA (Jakarta Persistence API)** para el manejo de entidades persistentes. El diseño refleja una arquitectura distribuida y que sigue el estilo SOA (Service-Oriented Architecture).

3.2. Vistas C4

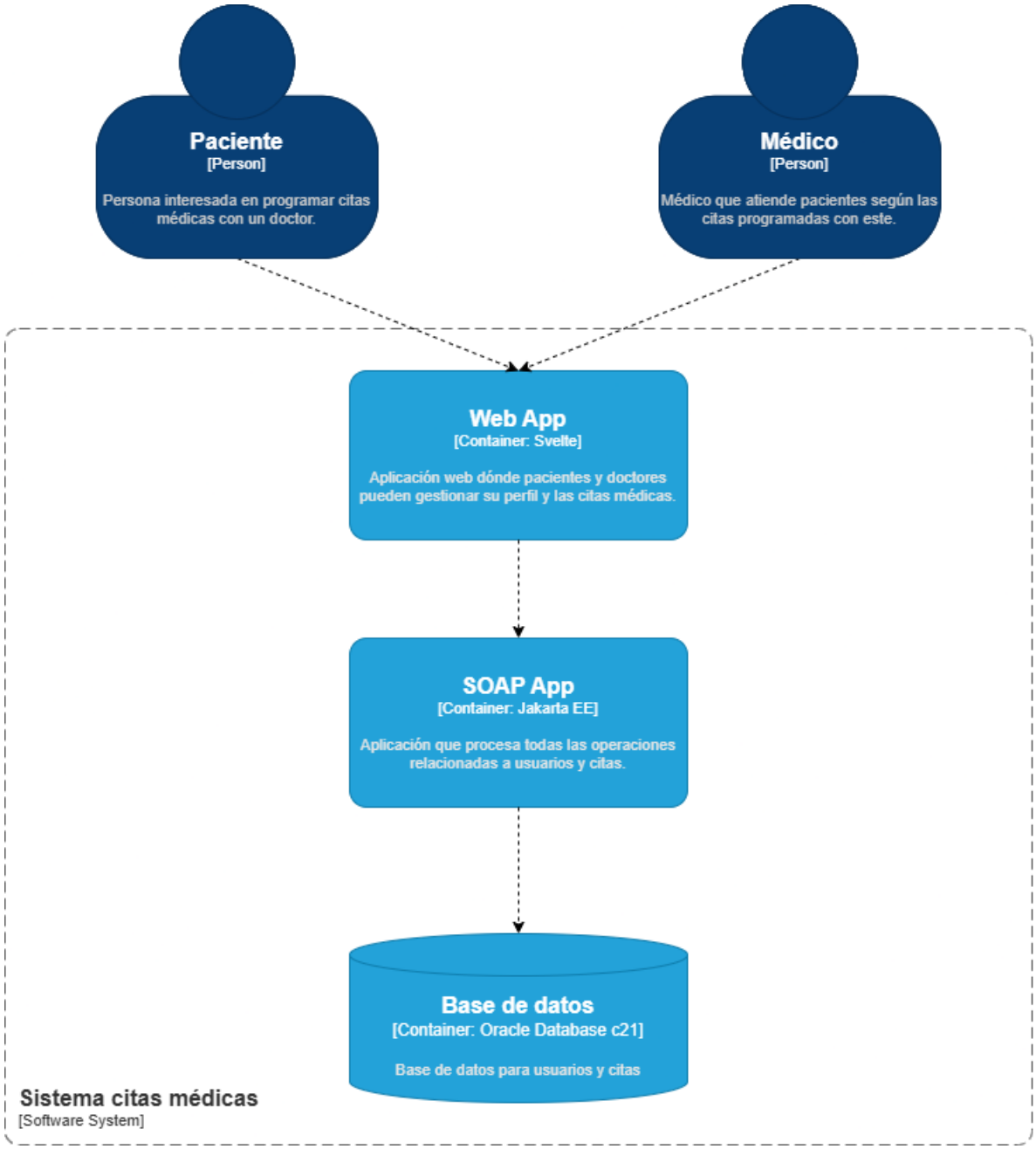
3.2.1. Vista de contexto



Esta vista contextualiza el sistema dentro de su entorno de uso, identificando a los **actores humanos** que interactúan con él. Los dos usuarios principales son el **paciente**, quien busca agendar citas médicas con un doctor, y el **médico**, encargado de atender a los pacientes según las citas previamente programadas.

Ambos actores se relacionan con el **sistema de citas médicas**, que se define como una solución digital diseñada para facilitar la interacción.

3.2.2. Vista de contenedores

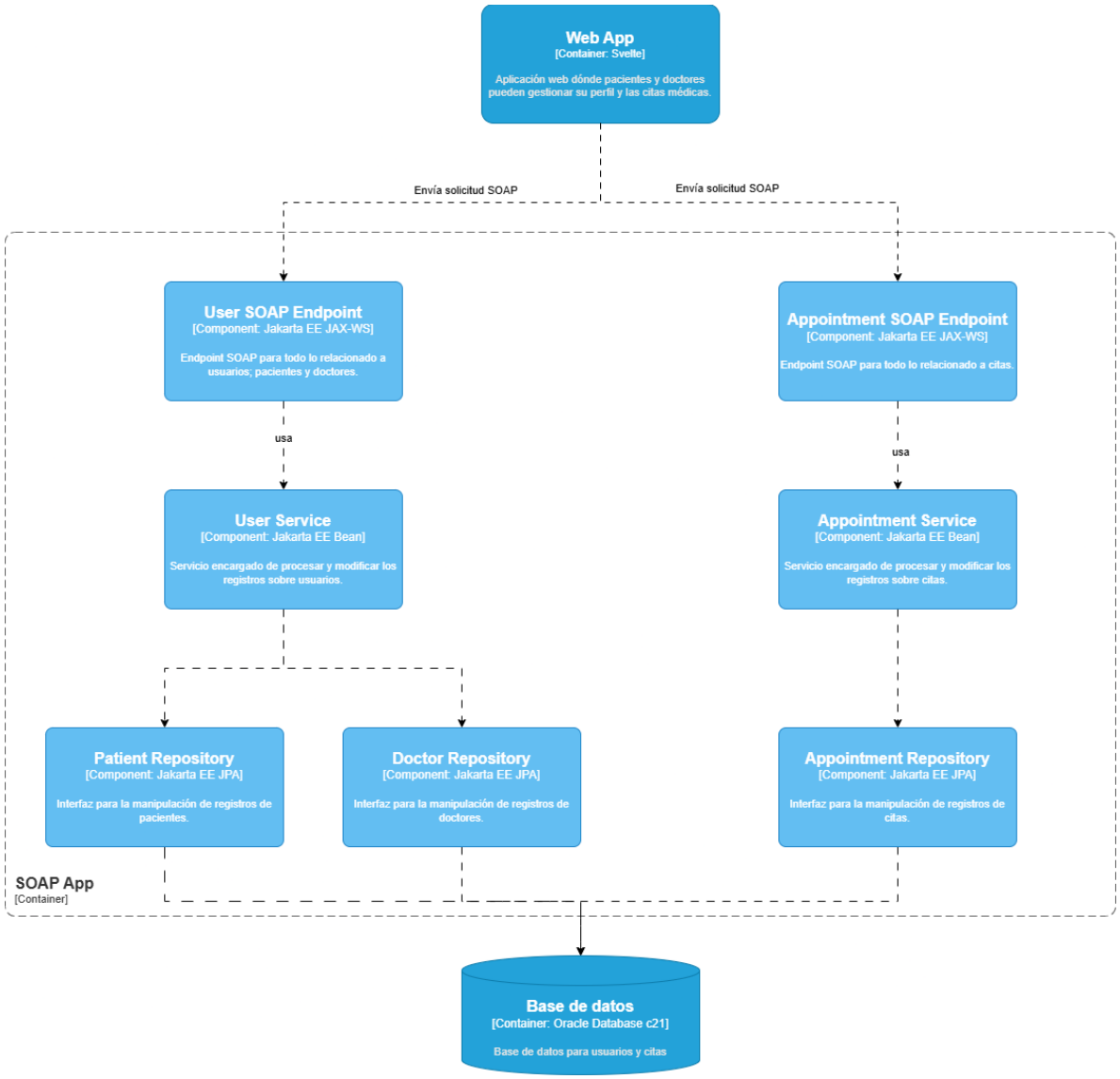


Esta vista ofrece una visión más detallada del sistema, mostrando cómo está estructurado en distintos **contenedores lógicos**, cada uno cumpliendo una función específica dentro de la arquitectura.

- La **Web App**, desarrollada en **Svelte**, permite que tanto médicos como pacientes gestionen sus perfiles y programen o visualicen citas médicas desde el navegador.

- La **SOAP App**, construida con **Jakarta EE**, contiene toda la lógica de negocio del sistema. Aquí se procesan las operaciones relacionadas con la gestión de usuarios y citas, respondiendo a las solicitudes SOAP realizadas por la aplicación web.
- La **base de datos**, una instancia de **Oracle Database 21c**, actúa como el repositorio central donde se almacena toda la información de usuarios y citas médicas.

3.2.3. Vista de componentes

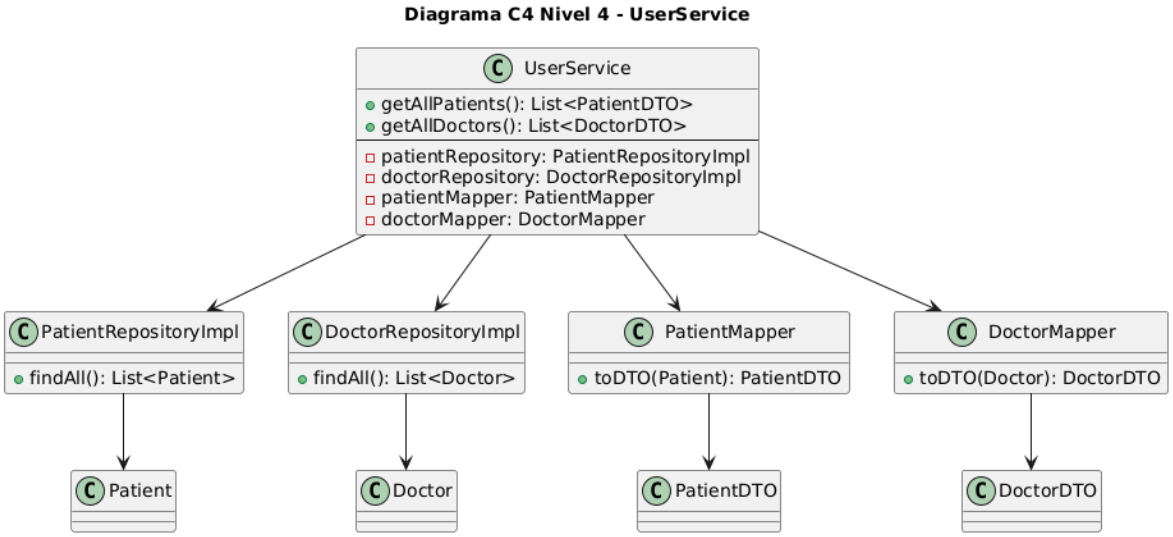


Esta vista descompone el contenedor **SOAP App** para mostrar los distintos **componentes** que lo conforman y sus relaciones. Se distinguen dos grupos principales de funcionalidades, cada uno expuesto mediante un endpoint específico:

- El **User SOAP Endpoint** maneja todas las solicitudes relacionadas con la gestión de usuarios (pacientes y doctores). Utiliza internamente el **User Service**, el cual interactúa con los repositorios correspondientes para acceder y modificar datos, además de convertir entidades a objetos DTO mediante mappers especializados.
- El **Appointment SOAP Endpoint** se encarga de las operaciones relacionadas con citas médicas. Delega su lógica al **Appointment Service**, el cual también se apoya en su repositorio y mapeadores respectivos.

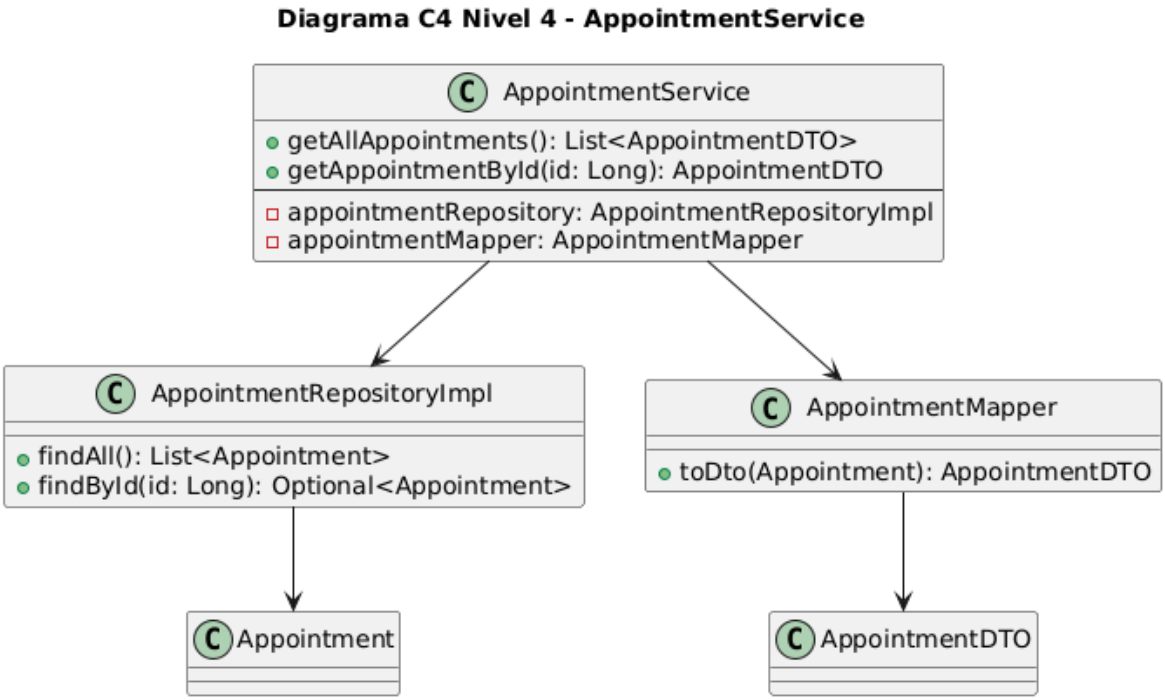
Esta perspectiva presenta de forma más detallada como se implementa SOA en el ejemplo práctico. A su vez se muestra el estilo por capas que se usa en la SOAP App.

3.2.4. Vista de código



Define cómo el componente UserService interactúa con sus dependencias:

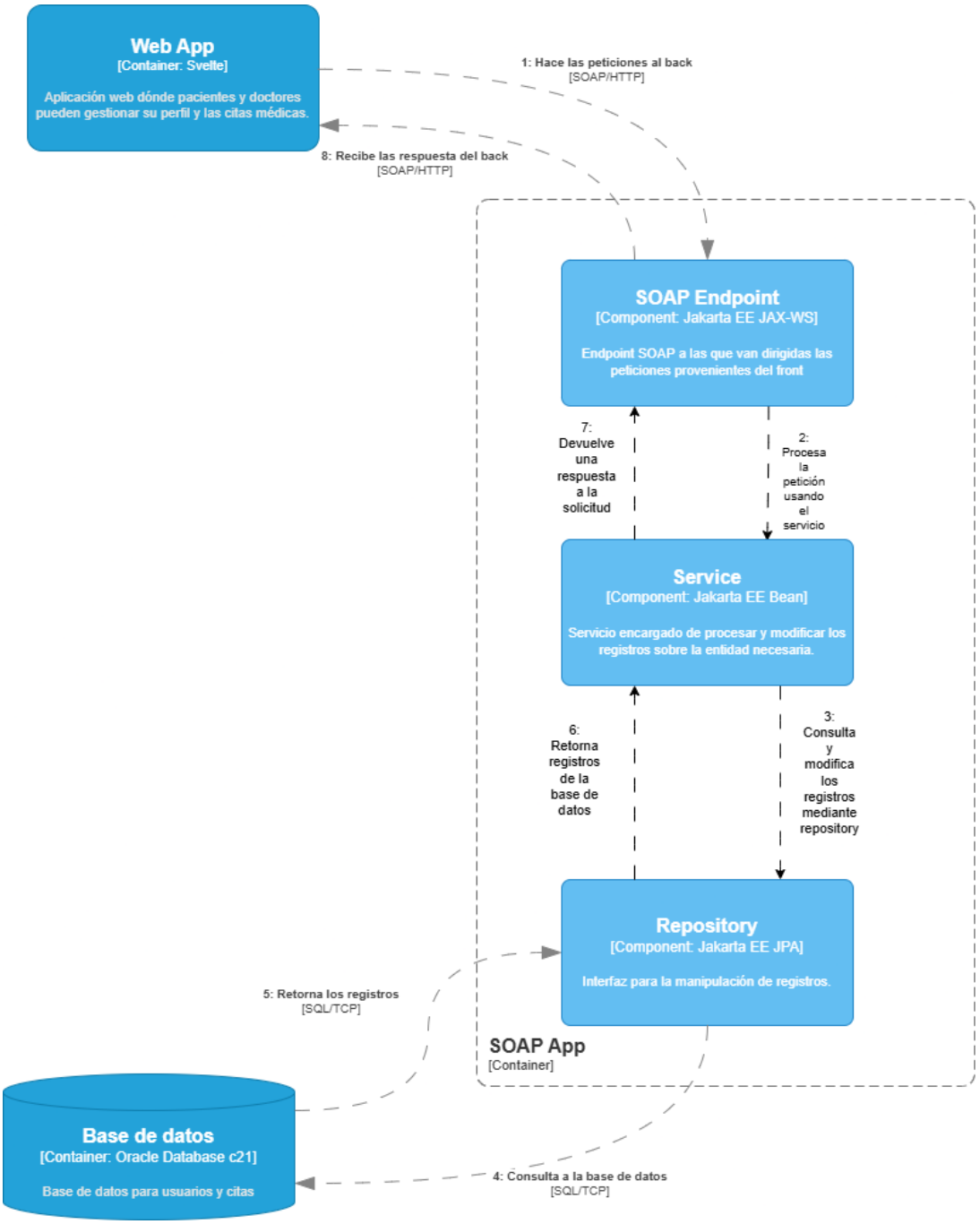
- Repositorios: PatientRepositoryImpl y DoctorRepositoryImpl, para obtener los datos.
- Mappers: PatientMapper y DoctorMapper, para convertir las entidades a DTOs (PatientDTO, DoctorDTO).



Similar al anterior, pero enfocado en el AppointmentService:

- Usa AppointmentRepositoryImpl para recuperar datos de citas.
- Aplica AppointmentMapper para transformar los datos a objetos DTO (AppointmentDTO).

3.3. Diagrama Dynamic C4



Este diagrama representa el flujo dinámico de información en el sistema de citas médicas, desde que un usuario realiza una solicitud en la interfaz web hasta que se devuelve una respuesta con los datos solicitados desde la base de datos. A través de ocho pasos secuenciales, se muestra cómo se coordinan los distintos componentes para cumplir una petición:

1. **Solicitud desde la Web App (Svelte)**
   El proceso comienza cuando un **paciente o médico**, usando la **aplicación web**, realiza una petición (por ejemplo, consultar citas o actualizar un perfil). Esta solicitud se realiza mediante el protocolo **SOAP sobre HTTP** hacia el backend.

2. **Recepción en el SOAP Endpoint (Jakarta EE JAX-WS)**
   La solicitud es recibida por el componente **SOAP Endpoint**, que actúa como punto de entrada del sistema backend. Aquí se identifica el tipo de operación requerida y se delega su procesamiento al servicio correspondiente.

3. **Procesamiento en el Service (Jakarta EE Bean)**
   El **Service** es responsable de la lógica de negocio. Según el tipo de operación, este componente determina qué acciones deben realizarse sobre los datos. Utiliza el **Repository** para acceder a la información necesaria.

4. **Consulta a la Base de Datos desde el Repository (JPA)**
   El **Repository**, implementado mediante **Jakarta EE JPA**, se encarga de consultar o modificar los registros en la base de datos. Este paso representa la interacción directa con el almacenamiento persistente.

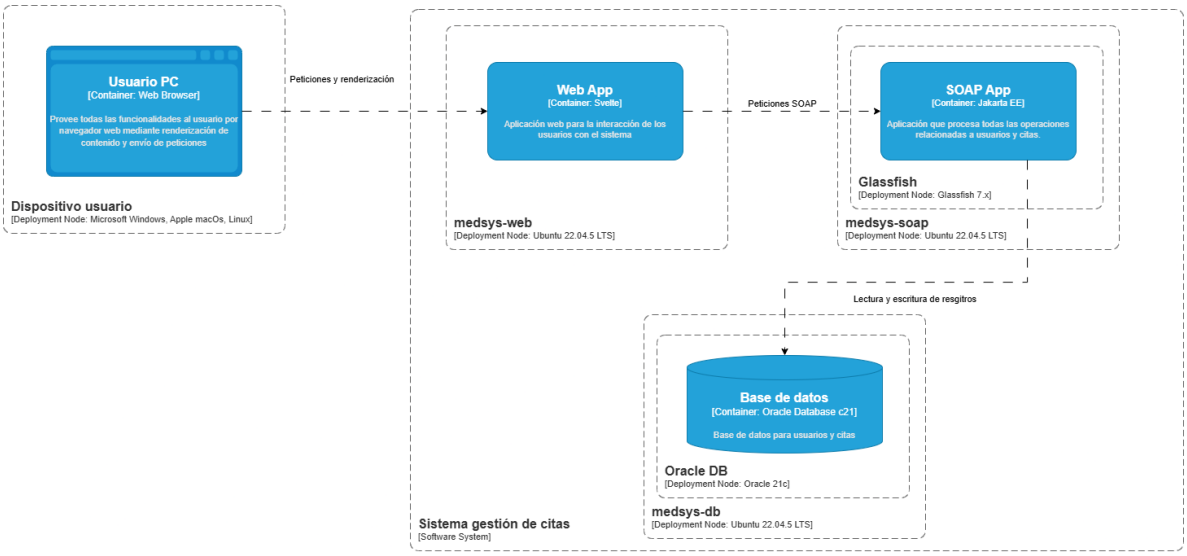
5. **Obtención de datos desde Oracle Database**
   La base de datos **Oracle c21** responde a la solicitud con los registros requeridos (usuarios, citas, etc.). Esta comunicación se realiza mediante **SQL/TCP**.

6. **Retorno de los registros al Service**
   Los datos obtenidos son devueltos al **Service**, el cual puede transformarlos si es necesario (por ejemplo, aplicando lógica adicional o formateo).

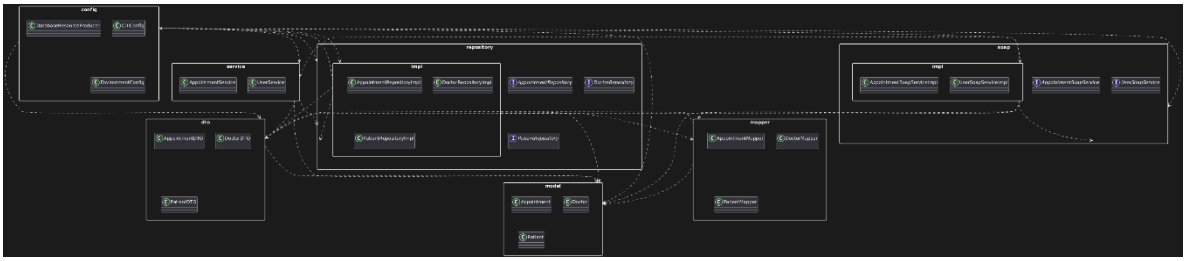
7. **Respuesta generada por el Endpoint**
   El **SOAP Endpoint** genera una respuesta SOAP con los datos procesados y la envía de regreso al cliente (la aplicación web).

8. **Respuesta recibida en la Web App**
   Finalmente, la **Web App** recibe la respuesta del backend. Los datos son mostrados al usuario final, completando así el ciclo de la solicitud.

3.4. Diagrama Despliegue C4



3.5. Diagrama de paquetes UML



3.6. Código Fuente

El ejemplo práctico se divide en tres repositorios:

3.6.1. P1-Infrastructure

Aquí se coordina todo el despliegue del sistema en una máquina gracias a Docker compose, es la forma más fácil y rápida de desplegar todo el sistema y ya esté integrado. Además, despliega la base de datos con todas las configuraciones necesarias.

https://github.com/Actividades-Arqui-2510/P1-Infrastructure

3.6.2. P1-Backend

Como su nombre indica contiene todo el backend hecho en Jakarta EE. También viene con configuraciones que facilitan el uso de glassfish considerando que es una tecnología en desuso.

https://github.com/Actividades-Arqui-2510/P1-Backend

3.6.3. P1-Front

Aquí se ubica todo el lado del front-end desarrollado en SvelteKit, ya cuenta con configuraciones para detectar el despliegue en docker y cambiar la url a la que manda solicitudes de acuerdo con esto.

https://github.com/Actividades-Arqui-2510/P1-Front

3.7. Muestra de funcionalidad

https://youtu.be/TvfyMZYYoAU

4. Referencias

2024 Jakarta EE Developer Survey Report. (2024, octubre).

https://outreach.eclipse.foundation/jakarta-ee-developer-survey-2024

JetBrains. (2023). The State of Developer Ecosystem in 2023 Infographic. JetBrains: Developer Tools For Professionals And Teams. https://www.jetbrains.com/lp/devecosystem-2023/

JetBrains. (2024). Software Developers Statistics 2024 - State of Developer Ecosystem Report. JetBrains: Developer Tools For Professionals And Teams. https://www.jetbrains.com/lp/devecosystem-2024

Statista. (2024, julio). *Most used web frameworks among developers worldwide 2024*. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Stack Overflow. (2024). *2024 Stack Overflow Developer Survey*. https://survey.stackoverflow.co/2024