Evaluación Arquitectónica ATAM

Concerns del Cliente

1. **Modificabilidad:** El cliente requiere un sistema que permita agregar nuevas herramientas de manipulación de imágenes de manera rápida y sencilla. También quiere que el sistema sea fácil de mantener a medida que crezca.
2. **Rendimiento:** El sistema debe ser capaz de cargar imágenes DICOM de manera rápida, y las herramientas de manipulación (contraste, negativo, mapas de colores) deben aplicarse sin afectar la experiencia del usuario.
3. **Disponibilidad:** La aplicación debe estar disponible en todo momento para los médicos, ya que trabajarán con ella en el día a día. Un tiempo de inactividad no es aceptable en entornos médicos.

Decisiones Arquitectónicas

1. **Uso de Cornerstone para la Visualización de Imágenes DICOM:**

- **Contexto del problema**: El sistema debe visualizar imágenes médicas en formato DICOM en tiempo real, permitiendo a los usuarios interactuar con las imágenes, cambiar el contraste, brillo, y aplicar otras manipulaciones sin perder tiempo. Es importante seleccionar una librería que soporte estas funcionalidades de manera eficiente y sencilla.
- **Alternativas consideradas**:
  - **VTK.js**: Otra opción de visualización médica con soporte DICOM, pero más compleja en su integración y configuración.
  - **Papaya.js**: Proporciona una visualización simple de imágenes médicas, pero no ofrece la profundidad de manipulación de imágenes que se necesita (por ejemplo, ajuste de contraste en tiempo real o inversión a negativo).
  - **Cornerstone**: Ofrece una solución optimizada para DICOM y es fácil de integrar con herramientas de manipulación de imágenes como contraste y brillo.
- **Razón por la que se eligió Cornerstone**: Se eligió **Cornerstone** porque está optimizado específicamente para la visualización de imágenes médicas y tiene un buen soporte para la manipulación de parámetros importantes como el contraste y el brillo en tiempo real. Su documentación y comunidad de usuarios son amplias, lo que facilita la resolución de

problemas y el soporte en su implementación. Además, Cornerstone es modular, lo que permite agregar nuevas funcionalidades sin grandes cambios en la arquitectura del proyecto.

- **Impacto en los atributos de calidad**:
    - **Rendimiento**: Cornerstone es eficiente y está diseñado para manejar grandes volúmenes de datos de imágenes médicas, lo que mejora el rendimiento en la visualización y manipulación de imágenes.
    - **Modificabilidad**: Gracias a su arquitectura modular, Cornerstone permite agregar nuevas herramientas o mejoras en las herramientas de visualización sin tener que rehacer la base del sistema.
    - **Riesgo**: La configuración inicial de Cornerstone puede ser compleja, y algunos de sus componentes requieren un conocimiento avanzado para aprovechar al máximo sus capacidades. Si no se implementa correctamente, podría generar problemas de compatibilidad con otros componentes del sistema.

2. **Arquitectura Modular**:

Contexto del problema: El proyecto requiere ser escalable y fácil de mantener. Dado que el sistema incluye varias funcionalidades (búsqueda de imágenes, visualización en diferentes planos, manipulación de imágenes, etc.), es necesario que estas funcionalidades estén desacopladas para permitir futuras mejoras sin afectar al sistema completo.

Alternativas consideradas:

- **Arquitectura monolítica**: Una opción era implementar todas las funcionalidades dentro de un solo bloque de código (monolito). Esta solución podría ser más sencilla en términos de desarrollo inicial, pero generaría problemas a largo plazo en términos de mantenibilidad y escalabilidad.
- **Arquitectura modular**: Desacoplar las funcionalidades principales (búsqueda, visualización y manipulación) en módulos separados permite realizar cambios en una parte del sistema sin afectar las otras. Esto facilita futuras mejoras y la adición de nuevas herramientas sin afectar la estabilidad del sistema.

Razón por la que se eligió una arquitectura modular: Se eligió la **arquitectura modular** para garantizar la escalabilidad del sistema y facilitar su mantenimiento. Al separar las funcionalidades en módulos independientes, es más sencillo agregar nuevas características o mejorar las existentes sin causar

efectos colaterales en el resto del sistema. También facilita la prueba y el despliegue de cada módulo por separado.

Impacto en los atributos de calidad:

- **Modificabilidad**: La modularidad asegura que el sistema sea fácil de modificar o ampliar, ya que cada componente es independiente del otro.
- **Rendimiento**: La arquitectura modular puede ayudar a optimizar el rendimiento, ya que cada módulo puede ser optimizado o escalado de manera independiente según las necesidades.
- **Riesgo**: La modularidad añade complejidad en la **coordinación** entre los módulos. Hay que asegurar que los módulos trabajen de manera cohesiva requiere un esfuerzo adicional en la integración y pruebas.

4. **API Restful para la búsqueda y recuperación de imágenes DICOM**:

Contexto del problema: El sistema necesita una forma eficiente de buscar y recuperar imágenes DICOM. Los usuarios (médicos) necesitan buscar imágenes basadas en criterios como el ID del paciente o el nombre del paciente, y la respuesta debe ser rápida y eficiente para que las imágenes se carguen y procesen sin demoras.

Alternativas consideradas:

- **Comunicación directa sin API**: Una opción sería realizar la búsqueda directamente a través del cliente, interactuando con la base de datos desde el frontend. Esto puede resultar en problemas de seguridad y escalabilidad, ya que no habría un middleware que gestione las peticiones.
- **API RESTful**: Se decidió implementar una API RESTful para gestionar las búsquedas de imágenes y la comunicación entre el cliente (frontend) y el servidor (backend). Esta arquitectura permite gestionar múltiples solicitudes de manera eficiente y es fácil de escalar si se necesita.

Razón por la que se eligió una API RESTful: Se eligió una **API RESTful** por su capacidad para gestionar múltiples solicitudes simultáneas, su facilidad para ser escalada y su flexibilidad para agregar nuevos puntos de acceso (endpoints) en el futuro. Además, es una solución ampliamente adoptada, lo que facilita la integración con otros servicios o sistemas en caso de que el proyecto se expanda.

Impacto en los atributos de calidad:

- **Rendimiento**: La API RESTful optimiza el manejo de solicitudes, lo que mejora el tiempo de respuesta de las búsquedas y la recuperación de imágenes.
- **Disponibilidad**: La API puede ser escalada fácilmente utilizando balanceadores de carga o servidores redundantes, asegurando que el sistema esté disponible incluso durante picos de demanda.

- **Riesgo**: Si la API no está correctamente diseñada o implementada, podría convertirse en un cuello de botella en el rendimiento, afectando el tiempo de respuesta de las solicitudes y la experiencia del usuario.

El backend está diseñado utilizando una API Restful que permite buscar y recuperar imágenes DICOM basadas en el ID o nombre del paciente**.**

Identificación de Riesgos

1. Carga de Imágenes Lentas: La carga lenta de imágenes podría ser el resultado de archivos DICOM grandes o complejos que demandan un mayor procesamiento. Esto puede agravarse si el sistema maneja múltiples usuarios accediendo simultáneamente. Si el backend no está optimizado para manejar grandes volúmenes de datos, la velocidad de respuesta del sistema puede verse afectada.

Mitigación: Implementar mecanismos de **caching**, mejorar el procesamiento en el backend y optimizar la transferencia de datos para reducir el tiempo de carga. También, utilizar tecnologías como **WebSockets** para mantener una conexión activa y evitar recargas innecesarias.

2. Escalabilidad: La escalabilidad podría verse comprometida si el sistema modular no maneja bien la sincronización de los datos entre módulos. Por ejemplo, si el módulo de manipulación de imágenes necesita constantemente actualizar el estado de las imágenes con el módulo de visualización, podría haber problemas de rendimiento o de coherencia de datos.

Mitigación: Implementar estrategias de escalabilidad como **microservicios** independientes que puedan escalar de manera horizontal, y usar **balanceo de carga** para distribuir mejor el tráfico entre los servidores. Además, aplicar patrones de diseño como **event-driven architecture** para mejorar la comunicación entre módulos.

3. Compatibilidad de formatos DICOM:

- **Riesgo**: Existen diversas variantes del formato DICOM y no todas las imágenes pueden tener los mismos metadatos o estructuras. Esto podría generar problemas de compatibilidad en el sistema si algunas imágenes no se procesan correctamente o si los metadatos faltan.
- **Mitigación**: Implementar validaciones para garantizar que las imágenes DICOM cargadas cumplen con los estándares necesarios para la visualización. Además, desarrollar una estrategia de manejo de errores que notifique al usuario cuando una imagen no pueda procesarse.