**Documento de Arquitectura**

PROYECTO: “AI Assistant”

Integrantes – Año 2024

|  |  |
| --- | --- |
| Nombre | E-Mail |
| Dimeglio Saade Tadeo | alumno24.dimeglio.tadeo@ipm.edu.ar |
| Ruckj Aragona Tiago | alumno24.ruckj.tiago@ipm.edu.ar |
| Sgro Leonel Brian | alumno24.sgro.leonel@ipm.edu.ar |

Contenido

[Historial de Revisión 3](#_Toc445483355)

[1. Objetivo 4](#_Toc445483356)

[2. Diagrama de Contexto 4](#_Toc445483357)

[3. Diagrama de Arquitectura 5](#_Toc445483358)

[**3.1.** **Arquitectura General** 5](#_Toc445483359)

[**3.2.** **Arquitectura de Infraestructura** 5](#_Toc445483360)

[4. Atributos de calidad – Requisitos No Funcionales 6](#_Toc445483361)

[5. Tácticas para garantizar RNF 7](#_Toc445483362)

[6. Mecanismos de integración 8](#_Toc445483363)

[7. Patrones de diseño y de Arquitectura 9](#_Toc445483364)

Historial de Revisión

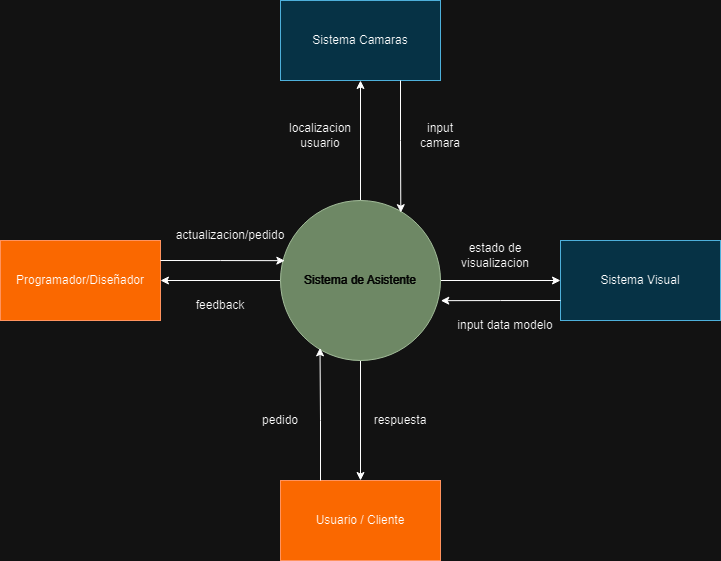
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fecha | Versión | Descripción | Rol | Autor |
| 28/05/24 | 1.0 | Añado Diagrama Contexto | Líder de Equipo | Dimeglio Saade Tadeo / N/A |
| 29/05/24 | 1.1 | Añado Diagramas Arquitectura, Atributos de Calidad, RN | Lider de Equipo | Dimeglio Saade Tadeo / Puntos 3.1 y 3.2 |

# Objetivo

Este documento describe la arquitectura general de la solución implementada en el proyecto, los componentes individuales que componen la solución y la forma en que interactúan. Esto se describe en el diagrama de contexto y en el diagrama de la arquitectura de una manera abierta a un público más amplio y en la sección posterior del documento de una manera más técnica y detallada.

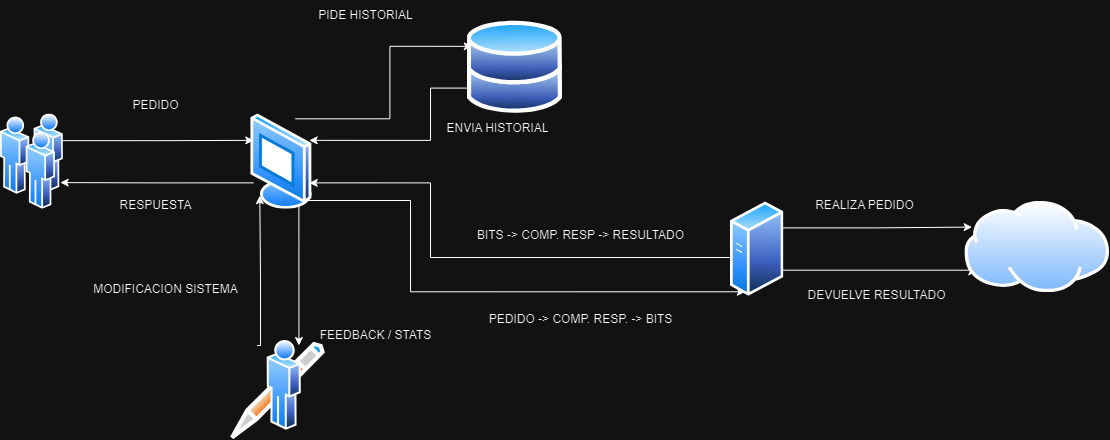
# Diagrama de Contexto

Este diagrama muestra los entes y entidades que interactúan con el sistema determinando los límites y el ambiente de este.

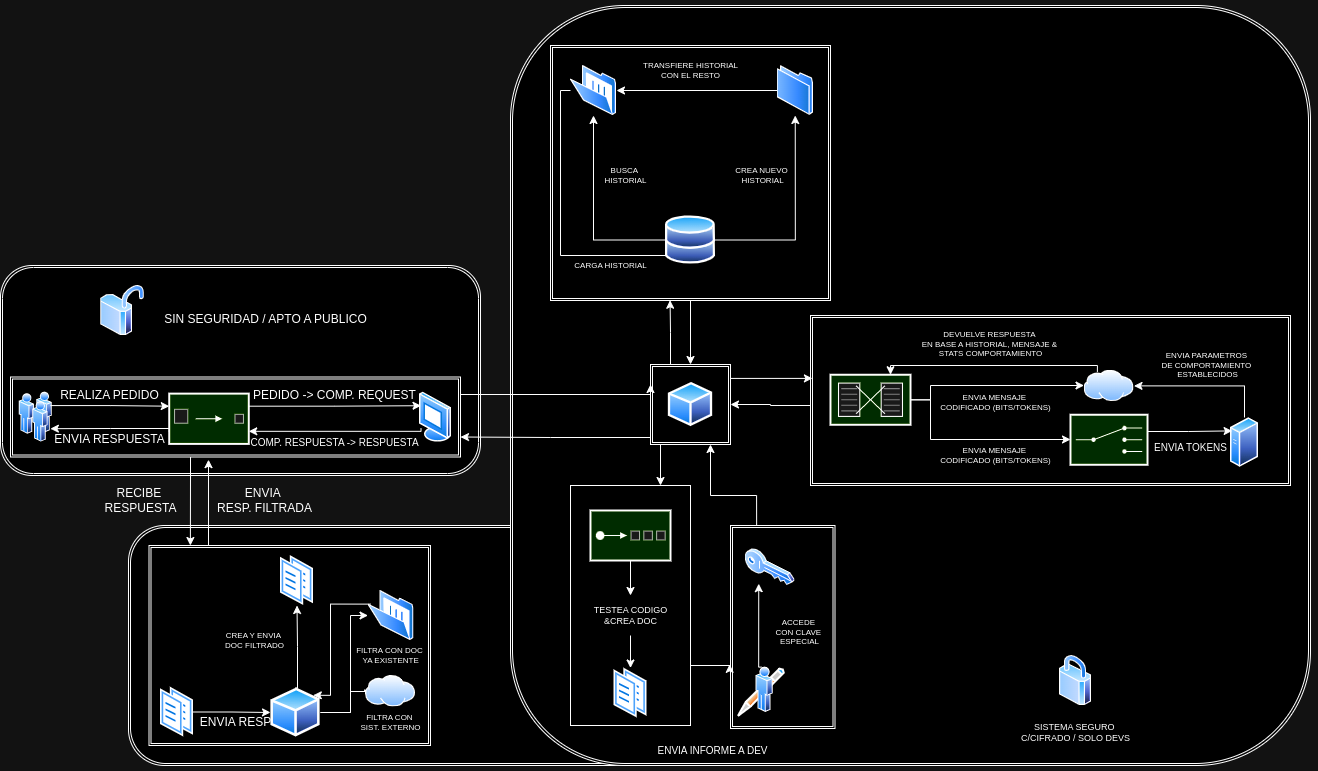


# Diagrama de Arquitectura

## **Arquitectura General**



## **Arquitectura de Infraestructura**



# Atributos de calidad – Requisitos No Funcionales

Son los aspectos del sistema también llamados requerimientos No funcionales, que en general no afectan directamente a la funcionalidad requerida, sino que definen la calidad y las características que el sistema debe soportar.

|  |  |
| --- | --- |
| ***Disponibilidad*** | |
| ***Estímulo*** | *Origen del Estímulo* | | *Usuario* |
| *Estímulo* | | *Intento de acceso al sistema* |
| ***Ambiente*** | *Ambiente* | | *Sistema en normal operación* |
| *Componente* | | *Servidor & Servicios backend* |
| ***Respuesta*** | *Respuesta* | | *Sistema disponible; accesible* |
| *Medida de Respuesta* | | *Disponible el 95,0%, no más de 30m de inactividad* |

|  |  |
| --- | --- |
| **Funcionabilidad** | |
| **Estímulo** | Origen del Estímulo | | Usuario |
| Estímulo | | Solicitud de reconocimiento y autenticación |
| **Ambiente** | Ambiente | | Sistema en operación normal |
| Componente | | Módulo de reconocimiento de objetos y autenticación |
| **Respuesta** | Respuesta | | Identificar y autenticar al usuario correctamente |
| Medida de Respuesta | | 97,5% de precisión de reconocimiento |

|  |  |
| --- | --- |
| **Eficiencia** | |
| **Estímulo** | Origen del Estímulo | | Usuario |
| Estímulo | | Consulta de datos |
| **Ambiente** | Ambiente | | Sist. Bajo carga normal de usuarios |
| Componente | | Procesador de IA; Base de datos |
| **Respuesta** | Respuesta | | Procesar la solicitud de reconocimiento & consulta  rápida |
| Medida de Respuesta | | T° respuesta menor a 1 segundo |

|  |  |
| --- | --- |
| **Compatibilidad** | |
| **Estímulo** | Origen del Estímulo | | Integrador de sistema |
| Estímulo | | Integración con otros sistemas o databases |
| **Ambiente** | Ambiente | | Fase de Integración |
| Componente | | Módulo de Integración |
| **Respuesta** | Respuesta | | Integración sin problemas con otros sistemas |
| Medida de Respuesta | | 100% de conformidad con la vinculación |

|  |  |
| --- | --- |
| **Seguridad** | |
| **Estímulo** | Origen del Estímulo | | Usuario mal intencionado |
| Estímulo | | Intento de acceso no autorizado |
| **Ambiente** | Ambiente | | Sistema en operación normal |
| Componente | | Módulo de Seguridad |
| **Respuesta** | Respuesta | | Bloquear intento de acceso y notificar devs |
| Medida de Respuesta | | 100% de intentos de acceso no autorizados |

|  |  |
| --- | --- |
| **Mantenibilidad** | |
| **Estímulo** | Origen del Estímulo | | Developer |
| Estímulo | | Actualizar/Modificar sistema |
| **Ambiente** | Ambiente | | Fase de Mantenimiento |
| Componente | | Códigos fuente, módulos, funciones |
| **Respuesta** | Respuesta | | Facilitar modificación/actualización sin problemas |
| Medida de Respuesta | | 95% de actualizaciones/modificaciones con éxito |

|  |  |
| --- | --- |
| **Usabilidad** | |
| **Estímulo** | Origen del Estímulo | | Usuario final |
| Estímulo | | Interacción con interfaz de usuario |
| **Ambiente** | Ambiente | | Sistema operando normalmente |
| Componente | | Interfaz de usuario |
| **Respuesta** | Respuesta | | Proporcionar actitudes y comportamientos naturales |
| Medida de Respuesta | | Satisfacción de usuario 85% + |

|  |  |
| --- | --- |
| **Fiabilidad** | |
| **Estímulo** | Origen del Estímulo | | Usuario |
| Estímulo | | Uso continuo |
| **Ambiente** | Ambiente | | Sistema operando bajo uso continuo |
| Componente | | Todos los módulos, funciones |
| **Respuesta** | Respuesta | | Funcionamiento sin fallos |
| Medida de Respuesta | | MTBF superior a 1000hs + |

# Tácticas para garantizar RNF

**Funcionalidad**

Táctica para garantizar funcionalidad: Validación de entradas y salidas

* + Se implementarán mecanismos de validación para garantizar que todos los datos ingresados por los usuarios sean correctos y que las respuestas generadas por el sistema sean precisas. Esto se realizará mediante la verificación de formatos y tipos de datos en cada punto de entrada y salida del sistema.

**Eficiencia**

Táctica para garantizar eficiencia: Uso de historiales

* + Se utilizará un sistema de caché (txts, jsons) para almacenar temporalmente los resultados de consultas frecuentes, lo que reducirá el tiempo de respuesta del sistema al evitar consultas repetitivas a la base de datos y mejorará la eficiencia del sistema en general.

**Compatibilidad**

Táctica para garantizar compatibilidad: APIs

* + Se desarrollarán APIs siguiendo protocolos de comunicación personalizados, facilitando la integración con otros sistemas y aplicaciones. Esto asegurará que el sistema pueda interactuar de manera efectiva y sin problemas con una variedad de otros servicios y plataformas.

**Seguridad**

Táctica para garantizar seguridad: Autenticación y autorización

* + Se implementarán mecanismos de autenticación y autorización para asegurar que solo los usuarios autorizados puedan acceder al sistema y sus datos. Esto incluirá métodos seguros para verificar la identidad de los usuarios y controlar sus permisos de acceso.

**Mantenibilidad**

Táctica para garantizar mantenibilidad: Modularización del código

* + Se adoptará una arquitectura modular, dividiendo el código en componentes independientes con funcionalidades específicas. Esto facilitará las actualizaciones y modificaciones del sistema, ya que cada módulo podrá ser gestionado y mantenido de forma aislada sin afectar a los demás.

**Usabilidad**

Táctica para garantizar usabilidad: Diseño centrado en el usuario

* + Se adoptarán principios de diseño centrado en el usuario, realizando estudios de usabilidad y pruebas con usuarios reales para asegurar que la interfaz del sistema sea intuitiva y fácil de relacionar. Esto garantizará que los usuarios puedan interactuar con el sistema de manera efectiva y satisfactoria.

**Fiabilidad**

Táctica para garantizar fiabilidad: Redundancia y replicación

Se implementarán mecanismos de redundancia y replicación de datos para asegurar la continuidad del servicio en caso de fallos. Esto significa que los datos se replicarán en múltiples ubicaciones y el sistema será capaz de recuperarse rápidamente de cualquier interrupción.

**Disponibilidad**

Táctica para garantizar disponibilidad: Implementación de balanceo

* + Se utilizará un balanceador para distribuir las solicitudes de los usuarios entre varios servidores. Esto asegurará que el sistema permanezca disponible y funcione de manera eficiente incluso durante períodos de alta demanda, evitando la sobrecarga de cualquier servidor individual.

# Mecanismos de integración

**Base de Datos:** Almacena y gestiona los datos de los usuarios, las consultas, y los resultados del reconocimiento de objetos.

* Almacenamiento centralizado: Todos los datos relevantes se almacenan en una base de datos central, garantizando integridad y consistencia.
* Acceso eficiente: La capa de acceso a datos utiliza consultas optimizadas para obtener y almacenar datos rápidamente.

**Colas MQ:** Gestiona el procesamiento asíncrono de tareas, mejorando la disponibilidad y eficiencia del sistema.

* Gestión de solicitudes: Durante picos de alta demanda, las solicitudes se encolan para ser procesadas en orden, evitando sobrecarga del sistema.
* Procesamiento de tareas en segundo plano: Tareas como el análisis de comportamiento y procesamiento de imágenes se manejan de manera asíncrona.

**Sockets TCP:** Permiten la comunicación directa y en tiempo real entre componentes del sistema.

* Comunicación en tiempo real: Utilizado para la transmisión de datos en tiempo real, como el feed de la cámara para la detección de objetos.

# Patrones y estilos Arquitectura

En esta sección se detallan los patrones y estilos de arquitectura escogidos para implementar la solución.

* **Cliente-Servidor:** El estilo Cliente-Servidor divide el sistema en dos tipos principales de componentes: clientes y servidores. Los clientes solicitan servicios y los servidores responden a esas solicitudes.

**Clientes:** Los dispositivos de los usuarios que interactúan con la IA.

**Servidor:** El backend que procesa las consultas a la base de datos y la detección de objetos.

* **Capas (Layers):** En este estilo, el sistema se organiza en capas jerárquicas, donde cada capa ofrece servicios a la capa superior y utiliza los servicios de la capa inferior.

**Capa de Presentación:** Interfaz de usuario para interactuar con la IA.

**Capa de Lógica de Negocio**: Procesa los parámetros de comportamiento y las consultas a la base de datos.

**Capa de Acceso a Datos**: Maneja las interacciones con la base de datos de usuarios.

* **Modelo-Vista-Controlador (MVC):** MVC separa la aplicación en tres componentes interrelacionados: el modelo (datos), la vista (UI) y el controlador (lógica de negocio).

**Modelo**: Representación de los datos de los usuarios.

**Vista**: Interfaz que muestra los datos y resultados de la IA.

**Controlador**: Maneja la lógica de la aplicación y las interacciones del usuario.

* **Observer**: Define una dependencia uno-a-muchos entre objetos para que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.

**Sujeto**: Los parámetros de comportamiento que cambian.

**Observadores**: Módulos que necesitan actuar en base a esos cambios, como la base de datos o el sistema de reconocimiento.

* **Microservicios**: Separa las funcionalidades en servicios pequeños y autónomos que se comunican entre sí a través de API.

**Servicios Independientes:** Un servicio para el reconocimiento de objetos, otro para las consultas a la base de datos, y otro para el manejo de la lógica de usuario.