Informe de Implementación: Servicio de Creación de Asignaciones

Este documento describe la estructura, el flujo de trabajo y la justificación arquitectónica de los componentes desarrollados para el servicio de creación de asignaciones (Médico-Consultorio) en el sistema.

1. Fundamento Arquitectónico

La implementación de este servicio sigue el patrón de diseño de **Arquitectura Hexagonal** (Puertos y Adaptadores).

Razón para esta elección:

1. **Aislamiento de Lógica (Dominio):** Permite que la lógica central del sistema (validaciones, Casos de Uso) sea completamente independiente de detalles externos como la base de datos (PostgreSQL), el *framework* web (Fastify) o el sistema de validación (Zod).
2. **Mantenibilidad y Escalabilidad:** Si el sistema decide cambiar de PostgreSQL a MongoDB o de Fastify a Express, solo se necesita modificar la capa de **Adaptadores** (Repositorios o Controladores), manteniendo intactos los **Casos de Uso** (Aplicación) y las **Entidades** (Dominio).
3. **Aislamiento del resto del sistema:** Esta decisión permite que la lógica y modelos en general de este servicio en particular (Que son bastantes), no queden mezclados con los del resto del sistema previamente implementado par facilitar la legibilidad y mantenimiento.

2. Componentes y Flujo de Procesamiento

El flujo de procesamiento de una solicitud **POST/asignaciones** atraviesa las siguientes capas, de fuera hacia adentro:

2.1. Adaptadores de Entrada (Infraestructura / HTTP)

Esta capa se encarga de recibir la petición HTTP y garantizar que los datos cumplan con el formato esperado:

- **asignacionesEsquema.ts - Esquema de Validación (Zod):**
  Define la estructura exacta (tipos de datos, longitudes, formato de hora HH:MM) que el cliente debe enviar, para realizar la validación de formato lo antes posible y rechazar peticiones mal formadas sin sobrecargar la lógica de negocio.
  Se ha usado
  Usé *"shape"* de ZOD para reutilizar la estructura existente de los esquemas de Médico y Consultorio existente para esas entidades, realizadas en el primer sprint.

- **AsignacionControlador.ts - Adaptador HTTP (Fastify):**
  1. Recibe la petición.

2. Aplica la validación de Zod.

3. Llama al Caso de Uso *(crearAsignacion)*. Se implementó el manejo de errores Zod (HTTP 400) y se redefinió el error de negocio (HTTP 409) a "Fallo en las Condiciones de Uso" para mayor claridad.

2.2. Capa de Aplicación (Casos de Uso)

Esta capa orquesta la lógica de negocio y las verificaciones de estado del sistema:

- **AsignacionCasosUso.ts:**

  1. Verificación de Existencia (Puertos): Utiliza los repositorios de Médico y Consultorio para confirmar que ambas entidades referenciadas existen. (Evita que se asigne a entidades fantasma).

  2.Verificación de solapamiento: Llama a *asignacionRepositorio.consultorioOcupado* para asegurar que la nueva asignación no se superponga con ninguna ya existente en el mismo consultorio y fecha, sin importar el médico asignado. Esto aplica la regla de negocio de **no doble-reserva** del recurso Consultorio, centralizando la lógica y manejando antes de persistir desde el repositorio.

  3. Verificación de Duplicidad: Llama a *asignacionRepositorio.existeAsignacion* para asegurar que el mismo médico no tenga un horario idéntico en el mismo lugar y fecha. Razón: Aplica las Reglas de Negocio (el porqué se crea o rechaza una asignación) de forma centralizada y agnóstica a la base de datos.

2.3. Adaptadores de Salida (Infraestructura / Persistencia)

Esta capa se encarga de realizar la operación final de persistencia de datos:

- **IRepositorioAsignacion.ts - Puerto de Persistencia:** Es la interfaz de dominio que define el contrato: *crearAsignacion(), existeAsignacion() y consultorioOcupado().* Lo que le permite actuar como el Puerto de la arquitectura hexagonal, permitiendo al Caso de Uso "hablar" con el Repositorio sin saber que utiliza PostgreSQL.

- **AsignacionRepositorio.ts - Implementación PostgreSQL:**

  1. Contiene la lógica SQL *(SELECT, INSERT).*

  2. Implementa la corrección para la creación de la asignación eliminando la clave *idAsignacion* antes del *INSERT*, lo que permite que la base de datos autogenere correctamente la clave primaria *(SERIAL)*. Este es el único lugar donde existe acoplamiento con la tecnología de persistencia *(PostgreSQL)*.