Informe de Implementación: Servicio de Gestión de agendamiento de cita (Paciente ↔ Cita ↔ Médico/Consultorio

Este documento describe la estructura, el flujo de trabajo y la justificación arquitectónica de los componentes desarrollados para el servicio de **agendamiento de citas médicas** en el sistema de gestión clínica.

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 para facilitar la legibilidad y mantenimiento.

2. Componentes y Flujo de Procesamiento

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

2.1. Adaptadores de Entrada (Infraestructura / HTTP)

> Este módulo define el contrato de datos que debe cumplir toda solicitud de creación de citas médicas antes de ser procesada por la capa de aplicación.
>
> Se utiliza la librería Zod para garantizar que los datos recibidos desde el cliente cumplan con el formato, tipo y reglas de negocio mínimas establecidas.

Fundamento:

La validación en esta etapa asegura la integridad de la información y evita que datos inválidos o malformados lleguen al dominio o la base de datos, manteniendo la coherencia del sistema y reduciendo posibles errores en tiempo de ejecución.

Principales Validaciones Implementadas:

- **Medico:**
  - Debe ser un identificador alfanumérico entre 5 y 15 caracteres.
  - Evita valores vacíos o con símbolos no permitidos.
  - **Ejemplo válido**: "MP12345".

- **tipoDocPaciente:**
  - Acepta solo valores enteros positivos entre 1 y 4, correspondientes a los tipos de documento definidos en la tabla tipo_documentos.
  - Garantiza la consistencia con la base de datos.

- **numeroDocPaciente:**
  - Campo alfanumérico obligatorio, entre 6 y 15 caracteres.
  - Permite validar la identificación del paciente según las reglas del sistema.

- **Fecha:**
  - Debe cumplir el formato ISO corto YYYY-MM-DD.
  - La expresión regular aplicada impide valores malformados (ej. “2025-13-40”).

- **horaInicio:**
  - Valida el formato exacto HH:MM (5 caracteres).
  - Evita valores inválidos o sin delimitador “:”.

Tipo Derivado (citaMedicaDTO):

A partir del esquema, se infiere automáticamente el tipo TypeScript citaMedicaDTO, el cual representa la estructura tipada de los datos válidos dentro del sistema. Esto permite mantener sincronización entre la validación y la definición de tipos en tiempo de compilación.

CitasControlador.ts – Adaptador HTTP (Fastify):

1. Recibe la solicitud HTTP POST con los datos del agendamiento.

2. Ejecuta la validación del esquema con Zod.

3. Llama al **Caso de Uso (AgendamientoCitaCasosUso.ts)** con los datos ya validados inicialmente con zod.

4. Gestiona errores:

    - Errores de formato (Zod): HTTP **400 – Petición inválida**.

    - Error al agendar intentar agendar una cita: HTTP **500 – Internal Server Error**.

2.2. Capa de Aplicación (Casos de Uso)

Esta capa orquesta toda la lógica de negocio relacionada con la creación de una cita médica.

AgendamientoCitaCasosUso.ts:

1. **Verificación de existencia (Puertos):**
   Llama a los repositorios de **Paciente** y **Médico** para asegurar que las entidades referenciadas existan antes de continuar.

    - Si alguna no existe, se lanza un error de negocio: *"Recurso inexistente"*.

2. **Validación de traslapes de horario, fecha cita y turnos medicos:**

    - Llama a `citasMedicasRepositorio.disponibilidadMedico()` para verificar que el médico no tenga otra cita en el mismo horario.
    - Llama a `citasMedicasRepositorio.validarCitasPaciente()` para confirmar que el paciente no tenga otra cita superpuesta.
      Estas validaciones aplican las **reglas de negocio clínicas** antes de intentar persistir la información.
    - Usa el método privado de `validarFechaVigente()` para validar que no se cree una cita en el pasado.
    - Usa el método privado de `validarTurnoMedico()` para validar que no se cree una cita con un médico específico que no tenga un turno válido ese día.

3. **Creación de la cita:**
   Si todas las verificaciones son exitosas, el caso de uso llama a `citasMedicasRepositorio.agendarCita()` para registrar la nueva cita con estado inicial “agendada”
   El caso de uso devuelve un objeto con los datos esenciales de la cita confirmada (paciente, médico, fecha, hora y estado).

2.3. Adaptadores de Salida (Infraestructura / Persistencia)

Esta capa se encarga de realizar la operación final sobre la base de datos.

ICitasRepositorio.ts – Puerto de Persistencia:
Define el contrato de dominio que debe cumplir cualquier implementación de persistencia, en este caso solo resaltamos los metodos utilizados para crear la nueva cita:

- `agendarCita()`
- `disponibilidadMedico()`
- `validarCitasPaciente()`
- `validarTurnoMedico()`

Este puerto actúa como intermediario entre la lógica de aplicación y la base de datos, permitiendo que el caso de uso no dependa directamente de SQL ni del tipo de persistencia.

CitasRepositorio.ts – Implementación PostgreSQL:

1. Contiene la lógica SQL (SELECT, INSERT) necesaria para realizar las verificaciones de disponibilidad y registrar las citas.

2. Implementa consultas de verificación de solapamiento usando condiciones en SQL sobre **fecha**, **hora_inicio** y **hora_fin**.

3. PostgreSQL genera automáticamente el identificador mediante **uuid_generate_v4().**

4. Devuelve el registro confirmado para su posterior respuesta al cliente.

Este adaptador es el único punto de acoplamiento con la tecnología de persistencia, manteniendo el resto del sistema agnóstico al tipo de base de datos.

3. Justificación y Beneficios

- **Reutilización y cohesión:** Cada capa cumple una única responsabilidad clara, permitiendo pruebas unitarias independientes.

- **Escalabilidad:** Nuevas reglas (por ejemplo, límite máximo de citas diarias por médico) pueden agregarse en los Casos de Uso sin afectar la infraestructura.

- **Seguridad de datos:** Las validaciones tempranas (Zod) y las verificaciones de disponibilidad (repositorio) evitan inconsistencias en la base de datos.

- **Extensibilidad:** Si se decide implementar recordatorios automáticos o una cola de notificaciones, puede añadirse como puerto adicional sin modificar el núcleo del dominio.

4. Conclusión

El **Servicio de Agendamiento de Citas** implementa una solución robusta y modular que cumple con los principios de la Arquitectura Hexagonal, garantizando la independencia de la lógica clínica respecto a la infraestructura.
El diseño asegura la **consistencia de los recursos** (pacientes, médicos y consultorios), evitando traslapes y entidades inexistentes. Además, deja el sistema preparado para futuras integraciones como recordatorios automáticos, cancelaciones o actualizaciones de estado de cita sin alterar el núcleo del dominio.

5. Video

https://www.youtube.com/watch?v=QY1FOk_g-8w