Anaís Castro - 202322450

Andrea Davila - 2023

Daniel Vergara - 202320392

27 de abril del 2025

DOCUMENTACION PROYECTO 2

CAMBIOS EN IMPLEMENTACION

1. Correccion pruebas de Postman

En la iteración pasada, hicimos pruebas erróneas de postman, evaluando los métodos POST y GET para cada entidad que creamos. Para esta iteración ajustamos las pruebas para que fueran más específicas, teniendo en cuenta los requerimientos funcionales. Además, incluimos también las pruebas de los requerimientos funcionales de consulta, que no fueron implementados para la última entrega. Tenemos pruebas para RF1 – RF9 y RFC1 – RFC6 (RF9, RFC5 y RFC6 son los requerimientos nuevos para esta iteración del proyecto).

Requerimientos Funcionales Mejorados

Durante esta entrega se corrigieron errores de implementación en los siguientes requerimientos funcionales:

RF1 - REGISTRAR IPS

Se solucionaron errores de conexión entre el formulario y la base de datos. Ahora se puede registrar una IPS correctamente desde Postman usando form-urlencoded, guardando todos los datos necesarios en la tabla correspondiente.

RF2 - REGISTRAR UN SERVICIO DE SALUD

Se ajustó el registro de servicios de salud para que los datos se inserten de forma correcta en la base de datos. Antes había problemas con los campos, pero ahora el registro funciona bien y pasa la prueba en Postman.

RF3 - ASIGNAR UN SERVICIO DE SALUD A UNA IPS

Se arregló el problema de la clave compuesta entre servicio e IPS. Se modificó el controlador para recibir los datos por RequestParam y se solucionó el error que causaba null en la clave primaria. Ahora la asignación de servicios a IPS funciona.

RF4 - REGISTRAR MÉDICO

Se corrigieron errores en el método de inserción y se ajustó el controlador para recibir correctamente los datos del médico. Ahora se puede registrar un médico desde Postman sin errores.

RF5 - REGISTRAR AFILIADO

Se solucionaron errores de mapeo que impedían registrar un afiliado. También se ajustó el controlador para recibir correctamente la información enviada. El registro de afiliados ahora pasa la prueba en Postman.

RF6 - REGISTRAR UNA ORDEN DE SERVICIO DE SALUD PARA UN AFILIADO POR PARTE DE UN MÉDICO

Se solucionaron los problemas relacionados con el envío de la fecha y las referencias a afiliado, médico y servicio. Ahora desde Postman se puede crear una nueva orden de servicio, asociándola correctamente al afiliado y al médico que la prescribe, cumpliendo así con la creación de órdenes.

RF7 - AGENDAR UN SERVICIO DE SALUD POR PARTE DE UN AFILIADO

Se habilitó correctamente la funcionalidad para que un afiliado pueda agendar un servicio de salud. En esta entrega se ajustaron los problemas que impedían crear una cita desde Postman, garantizando que ahora se registren la fecha, hora, el afiliado que agenda y el servicio solicitado. Ya se puede probar y confirmar que las citas quedan almacenadas correctamente en la base de datos.

RF8 - REGISTRAR LA PRESTACIÓN DE UN SERVICIO DE SALUD A UN AFILIADO POR PARTE DE UNA IPS

Se corrigió el registro de la prestación de un servicio de salud. Ahora es posible asociar correctamente la prestación a un afiliado, un médico y una IPS, indicando la fecha y la hora

del servicio. El proceso ya funciona en Postman y la información queda almacenada en la BD.

RF9 – AGENDAR UN SERVICIO DE SALUD POR PARTE DE UN AFILIADO – TRANSACCIONAL

Para este requerimiento, implementamos una transacción que permite a un afiliado agendar un servicio de salud. Lo que se debe cumplir para que el commit de a lugar es: confirmar que el servicio de salud seleccionado por el usuario esté disponible en la fecha y hora indicadas y agendar formalmente el servicio, asociándolo a la orden de servicio correspondiente. Si cualquiera de estos dos pasos falla, la transacción realiza un rollback automático para asegurar la consistencia de los datos. Esta funcionalidad se prueba a través de un POST a él url: http://localhost:8080/agenda/agendar y pasa correctamente la prueba en Postman.

La descripción de las nuevas clases y métodos creados para soportar esta funcionalidad, como la clase Agenda, se encuentra documentada por separado más adelante en este mismo documento.

Requerimientos Funcionales de Consulta Implementados:

RFC1 - Consultar la Agenda de Disponibilidad de un Servicio de Salud en las Siguientes 4 Semanas

Se implementó correctamente la consulta de disponibilidad para un servicio de salud ingresado por el usuario. El endpoint recibe el nombre del servicio y devuelve los horarios disponibles para las próximas cuatro semanas, incluyendo el nombre del servicio, la fecha y hora, la IPS que lo ofrece y el médico asignado. La prueba en Postman fue exitosa y se obtuvieron los resultados esperados. La consulta se hace mediante un GET al siguiente url: http://localhost:8080/citas/disponibilidad/{{nombreServicio}}.

RFC2 - MOSTRAR LOS 20 SERVICIOS MÁS SOLICITADOS

Se desarrolló el requerimiento para consultar los 20 servicios de salud más solicitados dentro de un período de tiempo dado. El usuario envía las fechas de inicio y fin, y se retorna un listado con los nombres de los servicios ordenados de mayor a menor según la cantidad

de solicitudes. El requerimiento fue probado en Postman enviando mediante un get al siguiente url: http://localhost:8080/ordenes/20masSolicitados, y la respuesta fue correcta, mostrando la lista como se espera.

RFC3 - Mostrar el Índice de Uso de Cada Uno de los Servicios Provistos

Se implementó el requerimiento que calcula el índice de uso para cada servicio ofrecido. Se calcula a partir de los datos de citas agendadas disponibles y cuantas de esas fueron usadas. Esta funcionalidad fue probada satisfactoriamente en Postman, mediante un GET a este url: http://localhost:8080/serviciosSalud/indiceUso. La respuesta muestra correctamente el nombre de cada servicio y su índice de uso dentro del período consultado.

RFC4 - MOSTRAR LA UTILIZACIÓN DE SERVICIOS DE EPSANDES POR UN AFILIADO DADO, EN UN PERIODO DADO

Se implementó un servicio que permite consultar todos los servicios utilizados por un afiliado en un rango de fechas específico. El usuario debe proporcionar el tipo de documento, el número de documento del afiliado, la fecha de inicio y la fecha de fin. La consulta se realiza mediante un GET a el url: http://localhost:8080/citas/disponibilidad/{{tipoDoc}}/{{numDoc}}/{{fechaInicio}}/{{fechaFin}}.

Como respuesta, el sistema devuelve una lista donde cada registro incluye el nombre del servicio, la fecha en que se utilizó, el médico que prestó la atención y la IPS que ofreció el servicio. La funcionalidad fue probada en Postman y pasa correctamente, entregando la información de manera clara y precisa, de acuerdo con los datos suministrados en la solicitud.

RFC5 – CONSULTAR LA AGENDA DE DISPONIBILIDAD DE UN SERVICIO DE SALUD - SERIALIZABLE

Se creó una consulta que retorna todos los horarios disponibles de los servicios de salud en las próximas semanas. Esta operación garantiza el nivel de aislamiento serializable para evitar inconsistencias si hay modificaciones concurrentes en la disponibilidad. Además, se implementó un temporizador que limita la ejecución de la transacción a 30 segundos, como se necesitaba.

La consulta se realiza haciendo un GET a este url: http://localhost:8080/agenda/disponibilidad y retorna la lista de servicios, IPS y horarios disponibles.

RFC6 – CONSULTAR LA AGENDA DE DISPONIBILIDAD DE UN SERVICIO DE SALUD - READ COMMITED

Es muy similar al que tiene un nivel de aislamiento serializable, solo que si se podrían ver los datos inconsistentes, evitando lecturas sucias, pero permitiendo fantasmas y no repetibles. El resultado es el listado de los servicios cuya fecha está en el rango dicho, prestados por el médico indicado.

2. **Creacion de AgendarServicio**

   En esta clase se encuentran los métodos de consultar disponibilidad serializable y read commited y el de agendar cita. Aca se encuentra la lógica para poder agendar una cita para un servicio de salud en una fecha y hora en la que el medico remitente tenga disponibilidad.

   Métodos:

   1. AgendarServicio

      Este método verifica que haya disponibilidad, y si la hay agenda la cita. Tiene excepciones en el caso de que no exista cupo en el horario solicitado. Hace un llamado a una función que hace la consulta de la disponibilidad para poder agendar o no la cita. Se usa la anotación de @Transactional.
   2. ConsultarDispSerializable

      Este método hace una consulta en la base de datos para verificar la disponibilidad en la agenda, teniendo en cuenta un servicio de salud, un médico y las fechas de inicio y fin sobre las cuales se desea hacer la búsqueda. Hace un llamado a la función de find disponibilidad en el repositorio de citas. Este retorna una lista con la disponibilidad si existen cupos dentro de las fechas solicitadas y lanza un error si no se puede consultar correctamente la agenda. Usa la anotación @Transactional. Tiene un temporizador de 30 segundos y el nivel de aislamiento es serializable.
   3. ConsultarDispReadCommited

Este método, al igual que el anterior, hace una consulta en la agenda para encontrar la disponibilidad de citas dentro de un rango de fechas. Sin embargo, para este método el nivel de aislamiento es distinto, usando read commited en vez de serializable.

3. **Creacion de AgendaController**

   En esta clase se manejan las peticiones HTTP para la ruta base /agenda. Acá podemos hacer las consultas de disponibilidad de citas y agendar las citas que correspondan. Este controlador no tiene la lógica de consulta y creacion, para esto hace un llamado a la clase AgendaServicio, la cual hace también un llamado al repositorio de citas.

   Métodos

   1. ConsultarDisponibilidad

      Este método permite consultar la disponibilidad de servicios médicos en la agenda. Es de tipo GET y la ruta que se usa es /agenda/disponibilidad
   2. Agendar

      Este método permite reservar una cita para un afiliado con un médico, en una hora especifica dada. Es de tipo POST y la ruta que se usa es /agenda/agendar. Si la cita se reserva exitosamente retorna “Servicio agendado exitosamente”, de no ser el caso retorna “Error al agendar servicio: [mensaje correspondiente al error]”.

4. **Cambios a CitaRepository y CitaController**

   En CitaRepository se agregaron unas funciones para poder probar los requerimientos funcionales de consulta, entre estas se encuentran findDisponibilidadServicioEnProximas4Semanas, findServiciosUsadosPorAfiliadoEnRango y findDisponibilidadAgenda. Además, se agregó la función que se encarga de agendar una cita y una función secundaria que ayuda a contar la disponibilidad de un médico que ofrece un servicio teniendo en cuenta una hora y una fecha, para poder agendar una cita con base a esto.

ESCENARIO DE PRUEBA DE CONCURRENCIA 1

El escenario de prueba 1 asegura que se lleve a cabo de manera concurrente el RF9 y el RFC5. El RF9 agenda una cita exitosamente y el RFC5 hace la consulta para encontrar la disponibilidad de

un servicio de salud. Para este escenario, la consulta se hace con un nivel de aislamiento serializable.

Se supone que esta prueba debería funcionar de tal manera que se ejecuta RFC5, el cual tiene un temporizador de 30 segundos y antes de que termine este tiempo se ejecute RF9 para que reserve una cita. Se haría la reserva después de que termine el temporizador, ya que esta consulta tiene un nivel de aislamiento serializable y RF9 no puede operar sobre las filas de la consulta durante ese tiempo.

ESCENARIO DE PRUEBA DE CONCURRENCIA 2

El escenario de prueba 2 es similar al anterior; este sigue el mismo proceso de ejecutar RFC6 y luego RF9 mientras corre el temporizador de 30 segundos del primero. En este caso, RFC6 tiene un nivel de aislamiento read commited, lo que implica que apenas se ejecute RF9 va a hacer la reserva inmediatamente, modificando la agenda.

Este escenario resulta en una consulta errónea, ya que RF9 hace un cambio a la agenda de disponibilidad que no se ve reflejada en la consulta final hecha por RFC6. Debido al nivel de aislamiento, los cambios hechos por RF9 durante la ejecución de RFC6 resultan en que esta consulta muestre como disponible un horario que ya ha sido agendado. Este nivel de aislamiento no garantiza que los resultados finales sean coherentes. Igualmente, al anterior, las pruebas no nos corren correctamente, entonces