Entrega 2
Proyecto Sistemas Transaccionales
ISIS 2304

1. Jeronimo Vasquez Ponce – 202223824
2. Paola Catherine Jimenez Jaque - 202116886
3. Yefran David Cespedes Cortes – 20231625

1. **Documentación de las clases de Java Spring para los RF y RFC de la entrega 1 completados**
- **Creación base de datos para los requerimientos de la entrega anterior, scripts de la creación de cada tabla de la BD.**

```
main > sql_info > Esquema.sql
CREATE TABLE Afiliado(
    idAfiliado NUMBER(20),
    tipoDocumento VARCHAR(40) NOT NULL,
    nombre VARCHAR(50)NOT NULL,
    fechaNacimiento DATE NOT NULL,
    direccion VARCHAR(50) NOT NULL,
    CONSTRAINT afiliadoPK PRIMARY KEY (idAfiliado)
);
```

Tabla Afiliado

La tabla Afiliado almacena la información de los usuarios afiliados al sistema de salud. Cada afiliado está identificado de manera única mediante el campo idAfiliado, que es la clave primaria de la tabla. Además, se registran datos esenciales como el tipoDocumento que especifica el tipo de identificación usado, el nombre del afiliado, su fechaNacimiento, y su dirección de residencia.

```
CREATE TABLE Contribuyente(
    afiliado NUMBER,
    CONSTRAINT AFILIADO_CONTRIBUYENTE_PK PRIMARY KEY (afiliado),
    CONSTRAINT AFILIADO_CONTRIBUYENTE_PK_FK FOREIGN KEY (afiliado) REFERENCES afiliado(idAfiliado)
);
```

Tabla Contribuyente

La tabla Contribuyente registra a los afiliados que también tienen el rol de contribuyentes dentro del sistema. Cada contribuyente está identificado de manera única por el campo afiliado, que a su vez es clave primaria y clave foránea que referencia al campo idAfiliado de la tabla Afiliado, garantizando que solo personas previamente registradas como afiliados puedan ser contribuyentes.

```
CREATE TABLE Beneficiario(
    afiliado NUMBER,
    contribuyente NUMBER,
    parentesco VARCHAR(50) NOT NULL,
    CONSTRAINT AFILIADO_BENEFICIARIO_PK PRIMARY KEY (afiliado, contribuyente),
    CONSTRAINT AFILIADO_BENEFICIARIO_A_PK_FK FOREIGN KEY (AFILIADO) REFERENCES AFILIADO(idAfiliado),
    CONSTRAINT AFILIADO_BENEFICIARIO_C_PK_FK FOREIGN KEY (CONTRIBUYENTE) REFERENCES AFILIADO(idAfiliado)
);
```

Tabla Beneficiario

La tabla Beneficiario registra la relación entre afiliados y contribuyentes, donde un afiliado es beneficiario de un contribuyente. Cada relación está identificada de manera única por la combinación de los campos afiliado y contribuyente, que en conjunto conforman la clave primaria. Además, se almacena el tipo de vínculo familiar a través del campo parentesco. Tanto afiliado como contribuyente son claves foráneas que referencian al campo idAfiliado de la tabla Afiliado, garantizando la integridad de las relaciones establecidas.

```sql
CREATE TABLE IPSs(
    nit NUMBER GENERATED BY DEFAULT AS IDENTITY,
    nombre VARCHAR(100) NOT NULL,
    direccion VARCHAR(255) NOT NULL,
    telefono VARCHAR(20) NOT NULL,
    horario VARCHAR(50) NOT NULL,
    CONSTRAINT pk_ips PRIMARY KEY (nit)
);
```

Tabla IPSs

Esta tabla representa las Instituciones Prestadoras de Servicios de Salud, identificadas por un NIT único. Se guardan atributos como nombre, dirección, teléfono y horario, lo que permite identificar y gestionar cada IPS de forma independiente.

```sql
CREATE TABLE Medicos(
    idMedico NUMBER GENERATED BY DEFAULT AS IDENTITY,
    identificacion VARCHAR(50) NOT NULL UNIQUE,
    nombre VARCHAR(100) NOT NULL,
    numRegistro VARCHAR(50) NOT NULL UNIQUE,
    especialidad VARCHAR(100) NOT NULL,
    CONSTRAINT pk_medico PRIMARY KEY(idMedico)
);
```

Tabla Medicos

Esta tabla unifica la información de los médicos, almacenando datos como identificación, nombre, número de registro y especialidad. Las restricciones de unicidad en ciertos campos aseguran la integridad y evitan duplicaciones.

```sql
CREATE TABLE OrdenServicios(
    idOrden NUMBER GENERATED BY DEFAULT AS IDENTITY,
    tipoOrden VARCHAR(250) NOT NULL,
    receta VARCHAR(1000) NOT NULL,
    estado VARCHAR(150) NOT NULL CHECK(estado IN ('VIGENTE','CANCELADA','COMPLETADA')),
    fecha DATE NOT NULL,
    idMedico NUMBER NOT NULL,
    idAfiliado NUMBER NOT NULL,
    CONSTRAINT pk_ordenServicio PRIMARY KEY(idOrden),
    FOREIGN KEY (idMedico) REFERENCES Medicos(idMedico),
    FOREIGN KEY (idAfiliado) REFERENCES Afiliados(idAfiliado)
);
```

Tabla OrdenServicios

Esta tabla registra las órdenes médicas, incluyendo detalles como tipo de orden, receta, estado (con valores restringidos a 'VIGENTE', 'CANCELADA' o 'COMPLETADA') y fecha. Además, asocia cada orden con un médico y un afiliado mediante claves foráneas.

```sql
CREATE TABLE ServiciosSalud (
    idServicio NUMBER GENERATED BY DEFAULT AS IDENTITY,
    fecha DATE NOT NULL,
    descripcion VARCHAR2(255) NOT NULL,
    CONSTRAINT pk_servicio PRIMARY KEY(idServicio)
);
```

Tabla ServiciosSalud

Esta tabla actúa como la entidad central para los servicios que ofrece la EPS. Contiene la fecha y una descripción de cada servicio, permitiendo centralizar y gestionar de forma uniforme los distintos tipos de servicios.

```
CREATE TABLE ExamenesDiagnosticos(
    idExamen NUMBER GENERATED BY DEFAULT AS IDENTITY,
    resultados VARCHAR(2000) NOT NULL,
    muestras VARCHAR(2000) NOT NULL,
    idServicio NUMBER NOT NULL,
    CONSTRAINT pk_examen PRIMARY KEY (idExamen),
    FOREIGN KEY(idServicio) REFERENCES ServiciosSalud(idServicio) ON DELETE CASCADE
);
```

Tabla ExamenesDiagnosticos

Esta tabla detalla los exámenes diagnósticos, almacenando resultados y muestras. Se relaciona con un servicio de salud, de forma que la eliminación de un servicio también elimina sus exámenes asociados.

```
CREATE TABLE Consultas(
    idConsulta NUMBER GENERATED BY DEFAULT AS IDENTITY,
    tipoConsulta VARCHAR(100) NOT NULL CHECK (tipoConsulta IN ('GENERAL','ESPECIALISTA','URGENCIA','CONTROL')),
    idAfiliado NUMBER NOT NULL,
    idMedico NUMBER NULL,
    idServicio NUMBER NOT NULL,
    CONSTRAINT pk_consulta_ PRIMARY KEY (idConsulta),
    FOREIGN KEY (idMedico) REFERENCES Medicos(idMedico) ON DELETE CASCADE,
    FOREIGN KEY (idAfiliado) REFERENCES Afiliados(idAfiliado) ON DELETE CASCADE,
    FOREIGN KEY(idServicio) REFERENCES ServiciosSalud(idServicio) ON DELETE CASCADE
);
```

Tabla Consultas

Esta tabla unifica la información de las consultas médicas, distinguiendo el tipo de consulta mediante un campo que admite valores como 'GENERAL', 'ESPECIALISTA', 'URGENCIA' o 'CONTROL'. Relaciona cada consulta con un afiliado, y opcionalmente con un médico y un servicio de salud.

```
CREATE TABLE Terapias(
    idTerapia NUMBER GENERATED BY DEFAULT AS IDENTITY,
    tipo VARCHAR(250) NOT NULL,
    cantidadSesiones NUMBER NOT NULL,
    idServicio NUMBER NOT NULL,
    CONSTRAINT pk_terapia PRIMARY KEY (idTerapia),
    FOREIGN KEY(idServicio) REFERENCES ServiciosSalud(idServicio) ON DELETE CASCADE
);
```

Tabla Terapias

Esta tabla almacena información sobre los servicios de terapia, indicando el tipo de terapia y la cantidad de sesiones programadas. Cada terapia se asocia a un servicio de salud para su correcta gestión.

```
CREATE TABLE ProcedimientosMedicos(
    idProcedimiento NUMBER GENERATED BY DEFAULT AS IDENTITY,
    tipo VARCHAR(250) NOT NULL,
    idServicio NUMBER NOT NULL,
    CONSTRAINT pk_procedimiento PRIMARY KEY (idProcedimiento),
    FOREIGN KEY(idServicio) REFERENCES ServiciosSalud(idServicio) ON DELETE CASCADE
);
```

Tabla ProcedimientosMedicos

Esta tabla contiene los datos de los procedimientos médicos especializados, como el tipo de procedimiento. Se vincula a un servicio de salud, permitiendo una gestión específica y detallada de estos procedimientos

```
CREATE TABLE Hospitalizaciones(
    idHospitalizacion NUMBER GENERATED BY DEFAULT AS IDENTITY,
    estado VARCHAR(255) NOT NULL,
    tratamiento VARCHAR(2000) NOT NULL,
    idServicio NUMBER NOT NULL,
    CONSTRAINT pk_hospitalizacion PRIMARY KEY (idHospitalizacion),
    FOREIGN KEY(idServicio) REFERENCES ServiciosSalud(idServicio) ON DELETE CASCADE
);
```

Tabla Hospitalizaciones

Esta tabla registra los eventos de hospitalización, incluyendo información sobre el estado y tratamiento. Se asocia a un servicio de salud, lo que permite gestionar y rastrear cada hospitalización de manera individual.

```
CREATE TABLE Afiliado_IPS(
    idAfiliado NUMBER NOT NULL,
    nit NUMBER NOT NULL,
    PRIMARY KEY (idAfiliado,nit),
    FOREIGN KEY (idAfiliado) REFERENCES Afiliados(idAfiliado) ON DELETE CASCADE,
    FOREIGN KEY (nit) REFERENCES IPSs(nit) ON DELETE CASCADE
);
```

Tabla Afiliado_IPS

Es una tabla intermedia que gestiona la relación muchos a muchos entre afiliados e IPSs. Permite asociar a cada afiliado con una o más IPS, garantizando la integridad referencial mediante una clave compuesta.

```
CREATE TABLE Medico_IPS(
    idMedico NUMBER NOT NULL,
    nit NUMBER NOT NULL,
    PRIMARY KEY (idMedico,nit),
    FOREIGN KEY (idMedico) REFERENCES Medicos(idMedico) ON DELETE CASCADE,
    FOREIGN KEY (nit) REFERENCES IPSs(nit) ON DELETE CASCADE
);
```

Tabla Medico_IPS

Esta tabla gestiona la relación muchos a muchos entre médicos e IPSs, permitiendo que un mismo médico se asocie a varias IPS. La clave compuesta asegura que cada vínculo es único y consistente.

```
CREATE TABLE Consulta_Examen(

    idConsulta NUMBER NOT NULL,
    idExamen NUMBER NOT NULL,
    PRIMARY KEY (idConsulta,idExamen),
    FOREIGN KEY (idConsulta) REFERENCES Consultas(idConsulta) ON DELETE CASCADE,
    FOREIGN KEY (idExamen) REFERENCES ExamenesDiagnosticos(idExamen) ON DELETE CASCADE
);
```

Tabla Consulta_Examen

Esta tabla vincula las consultas con los exámenes diagnósticos realizados, permitiendo que una consulta se asocie a uno o varios exámenes. La tabla utiliza una clave compuesta para garantizar la unicidad de cada asociación.

```
CREATE TABLE Hospitalizacion_Medico(
    idHospitalizacion NUMBER NOT NULL,
    idMedico NUMBER NOT NULL,
    PRIMARY KEY (idHospitalizacion, idMedico),
    FOREIGN KEY (idMedico) REFERENCES Medicos(idMedico) ON DELETE CASCADE,
    FOREIGN KEY (idHospitalizacion) REFERENCES Hospitalizaciones(idHospitalizacion) ON DELETE CASCADE
);
```

Tabla Hospitalizacion_Medico

Esta tabla asocia cada hospitalización con los médicos que participan en su atención. Utiliza una clave compuesta para asegurar que cada relación es única y se mantiene la integridad referencial.

```
CREATE TABLE Hospitalizacion_Afiliado(
    idHospitalizacion NUMBER NOT NULL,
    idAfiliado NUMBER NOT NULL,
    PRIMARY KEY (idHospitalizacion, idAfiliado),
    FOREIGN KEY (idAfiliado) REFERENCES Afiliados(idAfiliado) ON DELETE CASCADE,
    FOREIGN KEY (idHospitalizacion) REFERENCES Hospitalizaciones(idHospitalizacion) ON DELETE CASCADE
);
```

Tabla Hospitalizacion_Afiliado

Esta tabla registra la relación entre hospitalizaciones y afiliados, permitiendo identificar qué afiliados están involucrados en cada evento de hospitalización. La clave compuesta garantiza la unicidad de cada vínculo.

```
CREATE TABLE Ips_Servicio (
    IPSs NUMBER,
    idServicio NUMBER,
    CONSTRAINT IPS_SERVICIO_PK PRIMARY KEY (IPSs, idServicio),
    CONSTRAINT IPS_SERVICIO_IPS_FK FOREIGN KEY (IPSs) REFERENCES IPSs(NIT),
    CONSTRAINT IPS_SERVICIO_SERVICIO_FK FOREIGN KEY (idServicio) REFERENCES ServiciosSalud(idServicio)
);
```

Tabla Ips_Servicio

Esta tabla vincula los servicios de salud ofrecidos con las IPS que los proporcionan. La clave compuesta permite asociar cada servicio a una o más IPS, facilitando la gestión de estas relaciones.

```
CREATE TABLE Orden_Servicio(

    idOrden NUMBER NOT NULL,
    idServicio NUMBER NOT NULL,
    PRIMARY KEY(idOrden,idServicio),
    FOREIGN KEY (idOrden) REFERENCES OrdenServicios(idOrden) ON DELETE CASCADE,
    FOREIGN KEY (idServicio) REFERENCES ServiciosSalud(idServicio) ON DELETE CASCADE
);
```

Tabla Orden_Servicio

Esta tabla asocia cada orden médica con los servicios de salud prescritos. La tabla utiliza una clave compuesta para asegurar que la relación entre una orden y sus servicios es única y consistente.

```
CREATE TABLE Medico_Afiliado(
    idMedico NUMBER NOT NULL,
    idAfiliado NUMBER NOT NULL,
    PRIMARY KEY (idMedico,idAfiliado),
    FOREIGN KEY (idMedico) REFERENCES Medicos(idMedico) ON DELETE CASCADE,
    FOREIGN KEY (idAfiliado) REFERENCES Afiliados(idAfiliado) ON DELETE CASCADE

);
```

Tabla Medico_Afiliado

Esta tabla gestiona la relación muchos a muchos entre médicos y afiliados, permitiendo que se registre qué médicos atienden a qué afiliados. La clave compuesta garantiza la integridad y unicidad de cada asignación.

```
CREATE TABLE Disponibilidad (
    idDisponibilidad NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    idServicio       NUMBER NOT NULL REFERENCES ServiciosSalud(idServicio),
    nitIps           NUMBER NOT NULL REFERENCES IPSs(nit),
    idMedico         NUMBER NOT NULL REFERENCES Medicos(idMedico),
    fechaHoraInicio  TIMESTAMP NOT NULL,
    fechaHoraFin     TIMESTAMP NOT NULL,
    estado           VARCHAR2(20) NOT NULL CHECK (estado IN ('LIBRE','OCUPADO')),
    CONSTRAINT FK_Medico_IPS FOREIGN KEY (idMedico, nitIps) REFERENCES Medico_IPS(idMedico, nit),
    CONSTRAINT FK_Ips_Servicio FOREIGN KEY (nitIps, idServicio) REFERENCES Ips_Servicio(IPSs, idServicio)
);
```

Tabla Disponibilidad

La tabla Disponibilidad registra los espacios de tiempo disponibles u ocupados para un servicio de salud en una IPS, asociados a un médico. Cada disponibilidad tiene un rango de fechas y un estado ('LIBRE' o 'OCUPADO').

```
CREATE TABLE Citas (
    idCita           NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    idDisponibilidad NUMBER NOT NULL
                     REFERENCES Disponibilidad(idDisponibilidad)
                     ON DELETE CASCADE,
    idAfiliado       NUMBER NOT NULL
                     REFERENCES Afiliado(idAfiliado)
                     ON DELETE CASCADE,
    fechaReserva     TIMESTAMP NOT NULL
);
```

Tabla Citas

La tabla Citas almacena las reservas realizadas por los afiliados sobre las disponibilidades de los servicios de salud. Cada cita está identificada por idCita y guarda la fecha de reserva (fechaReserva). Además, mantiene integridad referencial con Disponibilidad y Afiliado, eliminando la cita automáticamente si alguno de estos registros es eliminado.

- **Clases Java descritas para los requerimientos anteriores**

Paquete controller

El paquete controller contiene las clases que exponen los servicios del sistema de salud a través de endpoints REST. Cada clase gestiona una entidad o proceso específico, como afiliados, citas, disponibilidades, órdenes de servicio y procedimientos médicos. Los controladores reciben solicitudes HTTP, coordinan la lógica de negocio básica y delegan las operaciones principales a los servicios y repositorios.

Paquete modelo

El paquete modelo contiene las clases que representan las entidades del sistema de salud. Cada clase mapea una tabla de la base de datos y define los atributos, llaves primarias, llaves foráneas y relaciones entre entidades. Incluye tanto entidades principales como AfiliadoEntity, CitaEntity y ServicioSaludEntity, como clases para gestionar claves compuestas (PK) y enumeraciones que modelan estados (EstadoDisponibilidad, EstadoOrden).

Paquete repositories

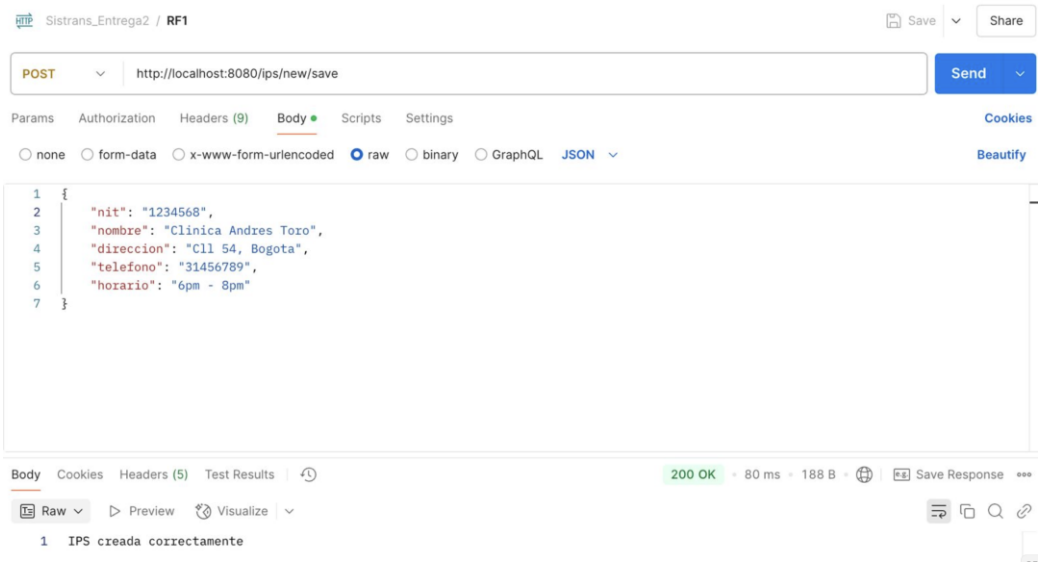
El paquete repositories contiene las interfaces que gestionan el acceso a la base de datos mediante Spring Data JPA. Cada repositorio corresponde a una entidad del sistema y define métodos para realizar operaciones CRUD y consultas personalizadas. Estos componentes permiten interactuar con las tablas de afiliados, citas, disponibilidades, servicios, médicos, hospitalizaciones, órdenes y procedimientos médicos de forma estructurada y eficiente.

- Requerimientos funcionales completados

RF1: El RF1 se encarga de registrar una nueva IPS en el sistema, lo que implica insertar un registro en la tabla correspondiente con los datos esenciales de la institución de salud, como nombre, dirección, teléfono y horario. Este proceso se inicia al enviar una solicitud desde Postman al endpoint designado, que recibe la información en formato JSON y la remite al repositorio encargado de persistirla en la base de datos.

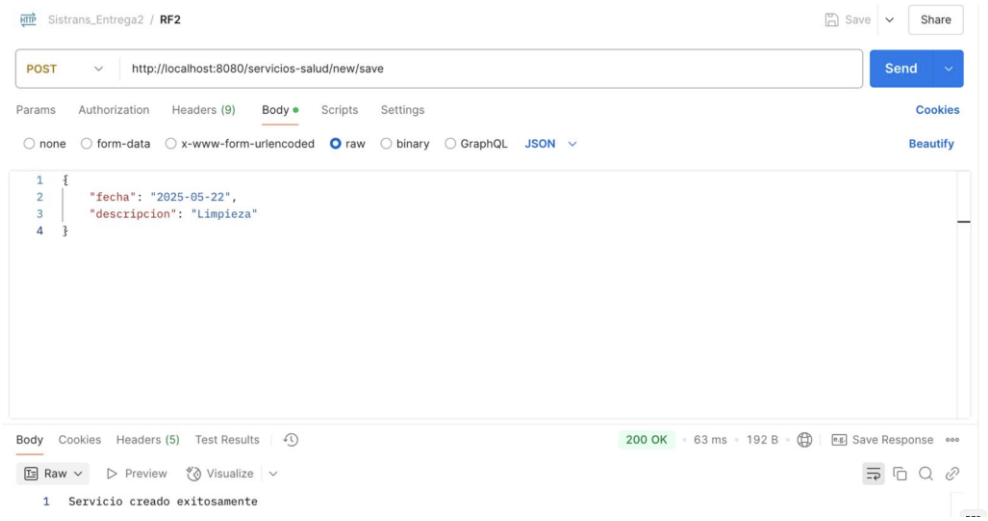
El repositorio para IPS, definido en la clase IpsRepository, extiende JpaRepository y utiliza consultas nativas para realizar operaciones específicas. En este código, el método insertarIp ejecuta un INSERT en la tabla IPSS, aprovechando una secuencia para generar automáticamente el identificador único (nit) de la IPS. Además, el repositorio define métodos como darIps para obtener todas las IPS, darIp para consultar una IPS en particular por su nit, y métodos para actualizar y eliminar registros. Las anotaciones @Modifying y @Transactional garantizan que las operaciones de inserción, actualización y eliminación se ejecuten de forma atómica y segura, permitiendo manejar los cambios en la base de datos de manera consistente.

Para probar este RF en Postman, se envía una solicitud POST al endpoint asociado con la creación de IPS, incluyendo en el cuerpo de la petición los campos requeridos (nombre, dirección, teléfono y horario). El controlador recibe esta información y llama al método insertarIp del repositorio, lo que resulta en la inserción del nuevo registro en la base de datos. El mensaje de respuesta confirma que la IPS se ha registrado correctamente, y los datos pueden verse reflejados en la base de datos, como se ilustra en la imagen:



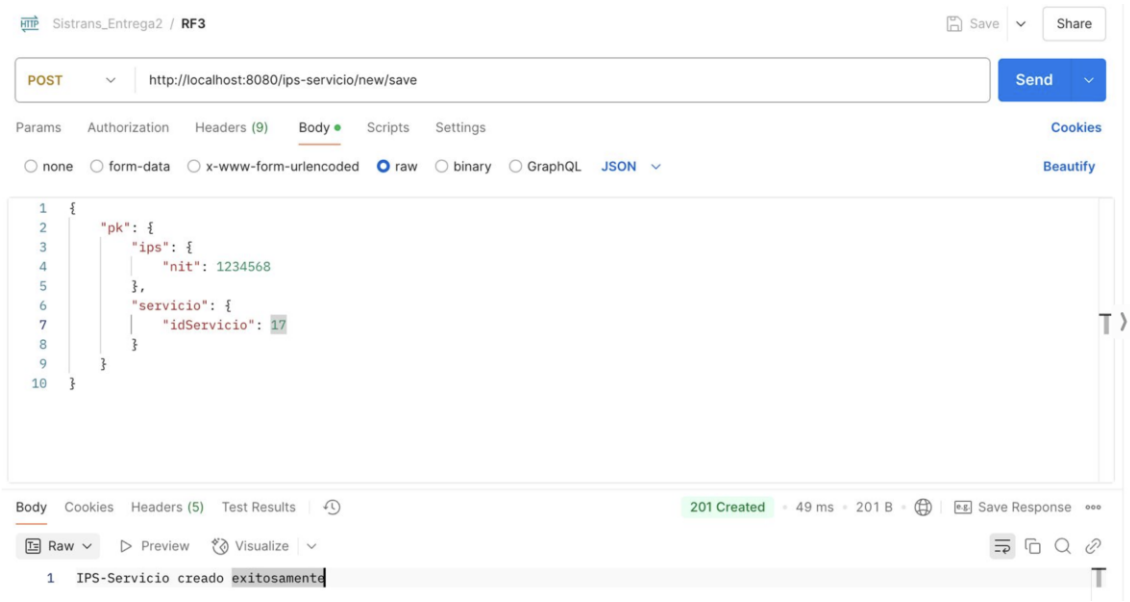
RF2: El RF2 se encarga de registrar un servicio de salud que la EPS ofrece a sus afiliados, lo que implica insertar un nuevo registro en la tabla correspondiente con datos esenciales como la fecha en la que se brinda el servicio y una descripción detallada del mismo. Este proceso se activa al enviar una solicitud POST desde Postman al endpoint designado para registrar servicios, donde se recibe la información en formato JSON. El controlador valida que los campos obligatorios estén presentes y realiza las conversiones necesarias, en particular transformando la fecha al formato requerido, para luego remitir los datos al repositorio que se encargará de persistir la información en la base de datos.

El repositorio de ServicioSalud, definido en la clase ServicioSaludRepository, extiende de JpaRepository y utiliza consultas nativas para realizar las operaciones de inserción, actualización, eliminación y consulta sobre la tabla ServiciosSalud. En este caso, el método insertarServicioSalud ejecuta una sentencia INSERT que utiliza la secuencia SERVICIO_SEQ.nextval para generar automáticamente el identificador único (idServicio) del servicio, y emplea la función TO_DATE para convertir la fecha al formato 'YYYY-MM-DD'. Las anotaciones @Modifying y @Transactional garantizan que la operación se ejecute de forma atómica y segura. Al probar este RF en Postman se envía una solicitud con la fecha y la descripción, el controlador invoca el método correspondiente del repositorio y se inserta el registro en la base de datos, tal como se observa en la imagen:



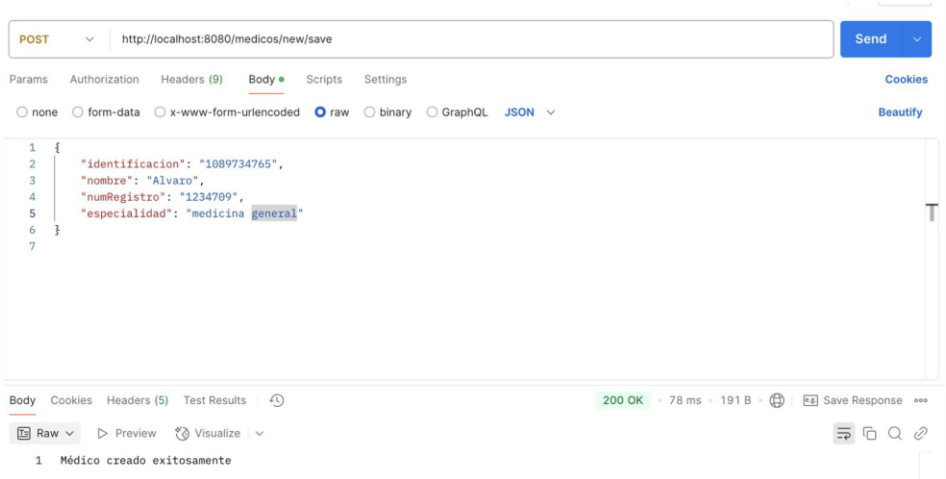
RF3: En el RF3 se busca asignar un servicio de salud a una IPS, lo que implica crear y registrar la relación entre un servicio ya existente y la institución que lo ofrece. Esta operación permite determinar qué servicios se ofrecen en cada IPS, lo que es fundamental para la gestión y consulta posterior de los servicios disponibles. La acción se realiza registrando una nueva asociación en la tabla intermedia correspondiente, lo que garantiza que cada asignación se realice de manera controlada y única.

El repositorio que actúa en el RF3 es el IpsServicioRepository, el cual extiende JpaRepository y utiliza consultas nativas para insertar, actualizar, eliminar y consultar las asociaciones entre servicios y IPSs. Mediante el uso de anotaciones como @Modifying y @Transactional, este repositorio asegura que la operación de asignación se ejecute de forma atómica y consistente, respetando las claves foráneas y las restricciones definidas en la base de datos. La integración de este componente permite que, al recibir los datos del servicio y la IPS a través del controlador, la relación se establezca de manera fiable en el sistema.



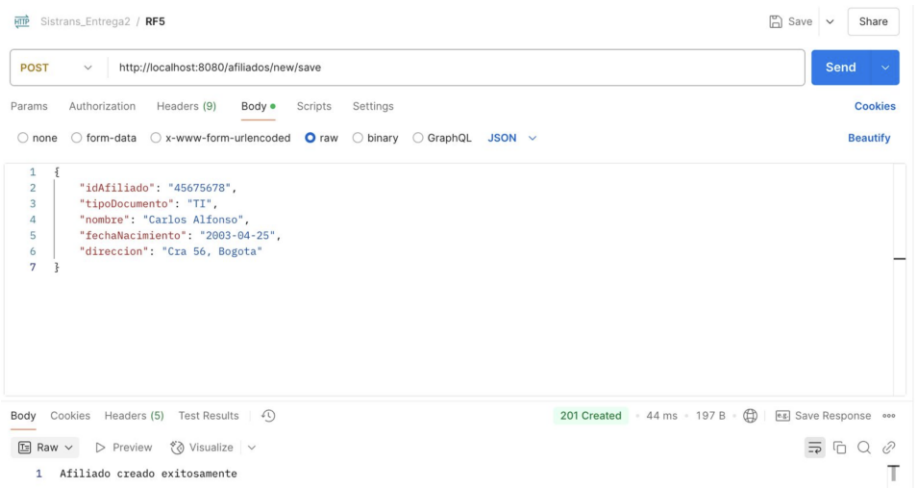
RF4: El RF4 se encarga de registrar un nuevo médico en el sistema, permitiendo que la EPS asocie a un profesional de la salud con los servicios que presta. Este proceso se activa mediante una solicitud POST al endpoint correspondiente, donde se envían los datos del médico en formato JSON. La información incluye la identificación, nombre, número de registro y especialidad, la cual es validada en el controlador para asegurarse de que todos los campos obligatorios están presentes antes de proceder a su inserción.

El repositorio de médicos, definido en la clase MedicoRepository, gestiona el acceso a la tabla Medicos a través de consultas nativas. En particular, el método insertarMedico realiza un INSERT utilizando la secuencia MEDICO_SEQ.nextval para generar automáticamente el identificador único del médico, y asegura que los datos se inserten de forma consistente en la base de datos. El controlador de médicos (MedicosController) invoca este método al recibir la solicitud de registro, y se encarga de enviar una respuesta confirmando que el médico se ha creado exitosamente. De esta forma, al probar el RF4 con Postman se observa que el registro se realiza correctamente en la base de datos, como se ilustra en la imagen:

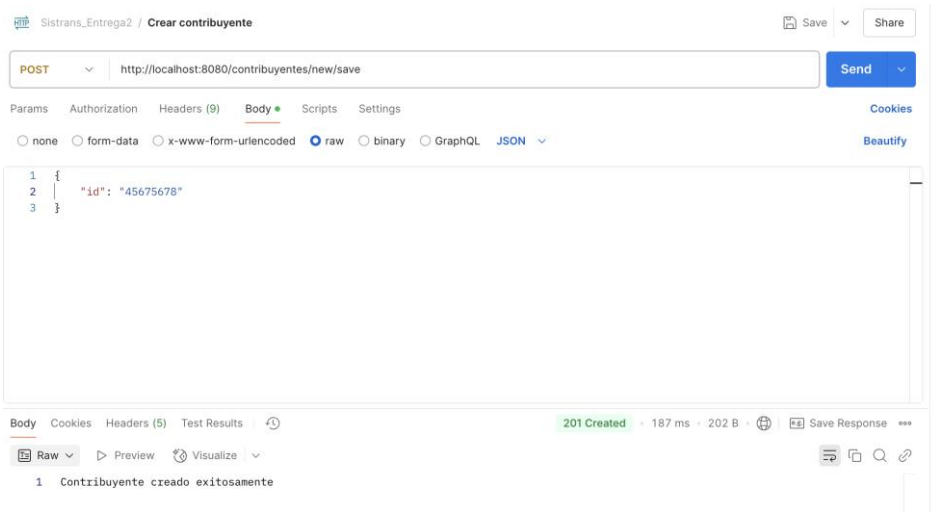


RF5: El RF5 se encarga de registrar un nuevo afiliado en el sistema, capturando toda la información personal necesaria, como el tipo de documento, nombre, fecha de nacimiento, dirección, teléfono, parentesco y el tipo de afiliación (CONTRIBUYENTE o BENEFICIARIO). Este proceso es esencial para garantizar que cada usuario quede debidamente identificado y para establecer las relaciones correspondientes con otros registros, como los beneficiarios que dependen de un contribuyente. Así, la inserción de un nuevo afiliado alimenta la base de datos con registros fundamentales para el funcionamiento de la aplicación.

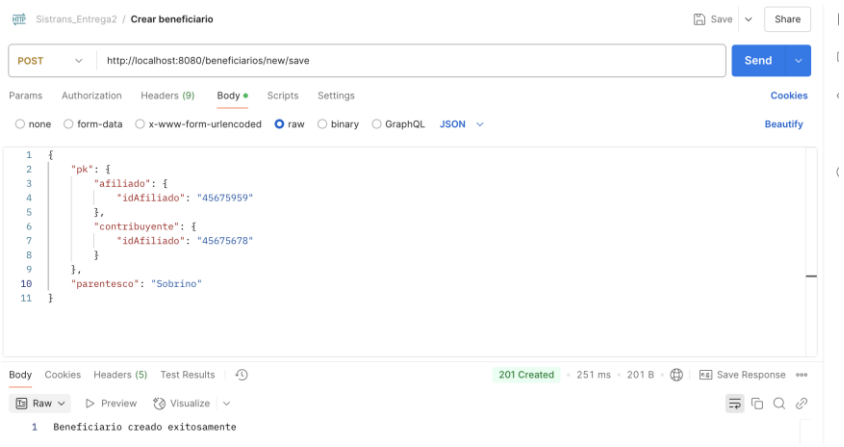
Para este RF, el AfiliadoRepository es el encargado de gestionar la persistencia de los datos del afiliado. Este repositorio utiliza consultas nativas a través de la anotación @Query para insertar, actualizar, eliminar y recuperar registros de la tabla Afiliados. Con el soporte de anotaciones @Modifying y @Transactional, el AfiliadoRepository garantiza que las operaciones se realicen de forma segura y en el contexto de una transacción, cumpliendo con las restricciones de integridad y las reglas de negocio establecidas en el modelo.



Creación del afiliado



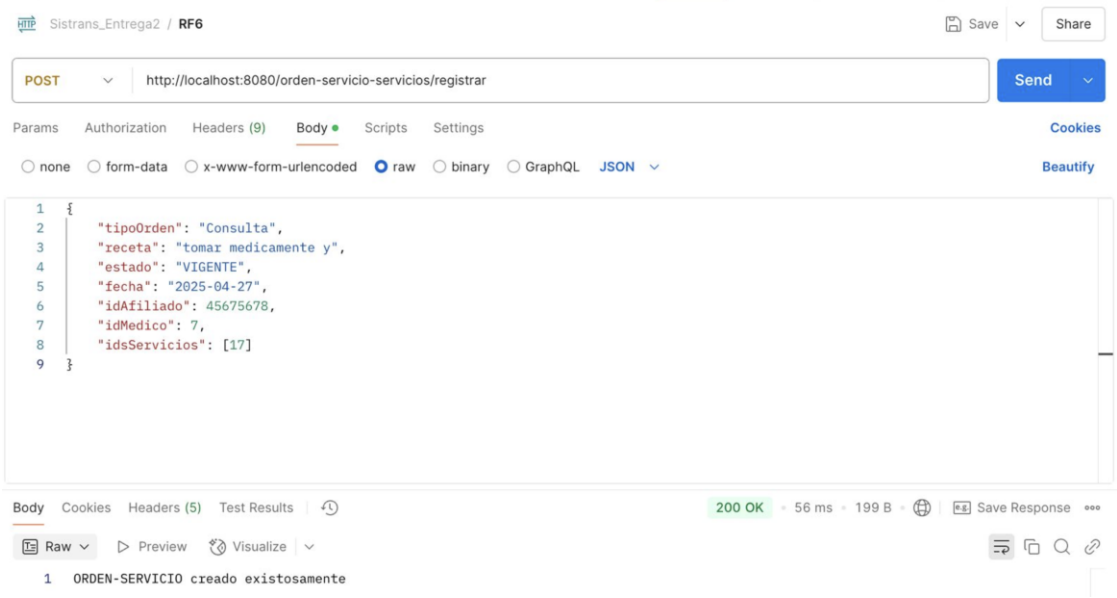
Creación contribuyente



Creación beneficiario

RF6: El RF6 se centra en registrar una orden de servicio de salud emitida por un médico para un afiliado. Este proceso implica la creación de un nuevo registro en la tabla de órdenes (OrdenServicios), donde se capturan datos críticos como el tipo de orden, la receta, el estado de la orden, la fecha y las referencias tanto al médico como al afiliado. La operación formaliza la prescripción de un servicio y establece las bases para posteriores procesos, como la asignación y ejecución del servicio.

El OrdenServicioRepository es el componente responsable en este RF, al proporcionar métodos para insertar, actualizar, eliminar y consultar órdenes de servicio mediante consultas nativas. Además, el OrdenServicioServiciosRepository se utiliza para vincular cada orden con uno o más servicios específicos. Juntos, estos repositorios se aseguran de que cada orden se registre cumpliendo las restricciones de integridad referencial y que las relaciones con los registros de médicos, afiliados y servicios se establezcan de forma coherente y transaccional.

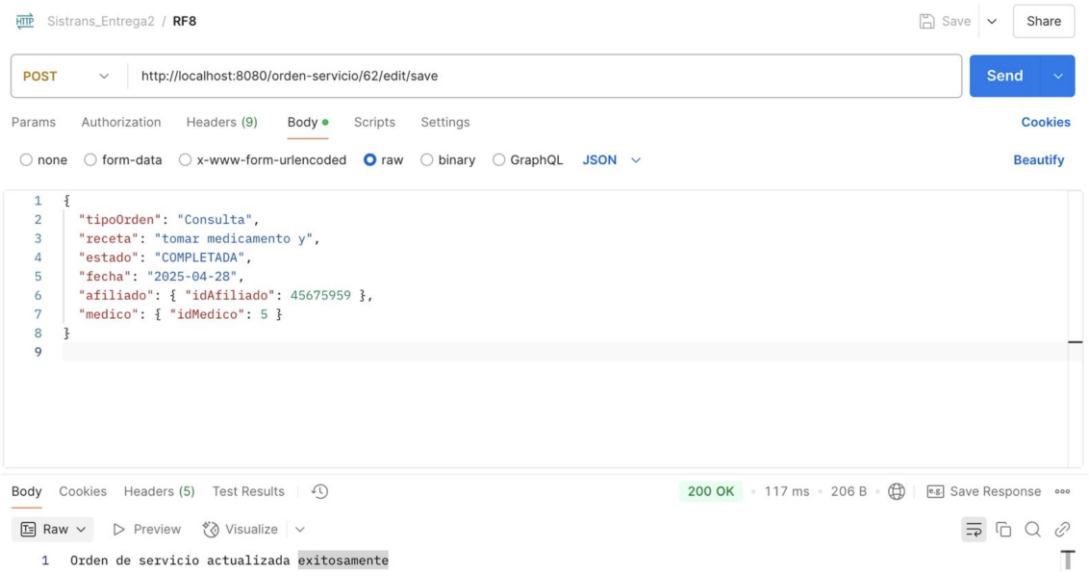


RF7: El RF7 tiene por objetivo agendar un servicio de salud para un afiliado, lo que implica reservar una cita o generar una consulta en el sistema. Este proceso consiste en registrar en la base de datos la intención de recibir el servicio en una fecha determinada, permitiendo coordinar la disponibilidad de los profesionales y las IPS, así como gestionar la planificación de los recursos necesarios. De esta forma, se habilita la operación de reserva o agendamiento, que es vital para la organización y el flujo de atención de la EPS.

El componente encargado de esta operación es el ConsultaRepository, que administra los registros de consultas en la tabla Consultas. Este repositorio define métodos mediante consultas nativas para insertar, actualizar, eliminar y consultar las reservas de servicio (consultas) realizadas por los afiliados. Al trabajar en conjunto con otros repositorios que gestionan datos de afiliados y médicos, el ConsultaRepository garantiza que el agendamiento se realice en un contexto transaccional, respetando las relaciones y restricciones definidas en el modelo.

RF8: Finalmente, el RF8 se encarga de registrar la prestación efectiva de un servicio de salud a un afiliado por parte de una IPS. Esta operación documenta la ejecución del servicio, incluyendo aspectos como la fecha, el estado y el tratamiento o procedimiento realizado, y establece la relación entre la prestación del servicio y las entidades involucradas (afiliados, IPS, médicos, etc.). Este registro es esencial para el seguimiento y la evaluación de la atención prestada, permitiendo auditar y controlar el uso de los recursos del sistema.

Para cumplir con el RF8 se utilizan varios repositorios coordinados: el OrdenServicioRepository registra la orden de servicio a utilizar, y así bien, aquella que se debe actualizar.



2. Documentación RF9, RFC5 Y RFC6

RF9

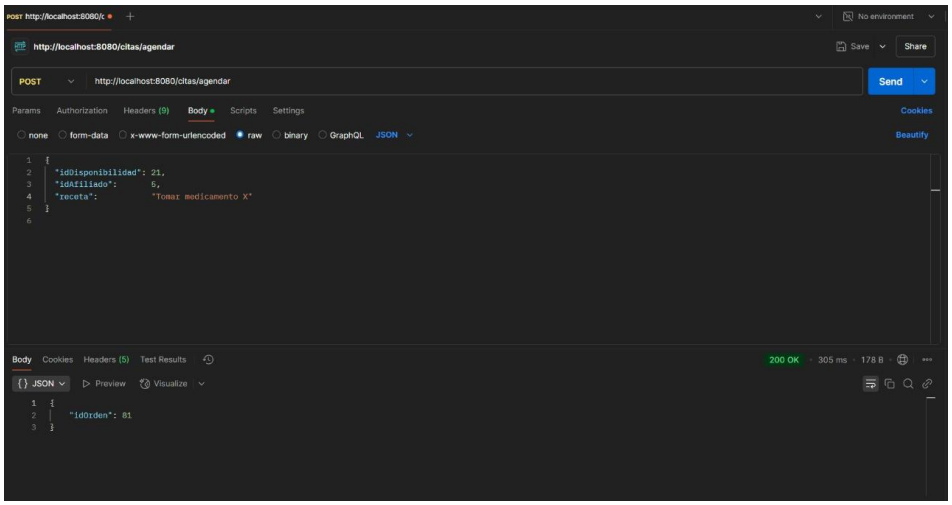
El RF9 permite a un afiliado reservar una franja horaria de un servicio de salud y generar al mismo tiempo una orden de servicio, garantizando que ambas operaciones se ejecuten de forma atómica. El afiliado primero consulta las disponibilidades disponibles mediante un endpoint que devuelve en formato JSON todas las franjas con su identificador, horario, médico e IPS asociado. Una vez elige una franja libre, envía un único POST al endpoint de agendamiento incluyendo el identificador de la disponibilidad, su propio identificador de afiliado y la receta correspondiente.

Al recibir la solicitud, el controlador bloquea la fila de disponibilidad seleccionada para evitar conflictos de concurrencia y comprueba que su estado sea “LIBRE”. Si la franja está disponible, el sistema crea una nueva entrada en la tabla de órdenes de servicio con estado “VIGENTE”, registra la relación entre esa orden y el servicio elegido, y actualiza la disponibilidad a “OCUPADO”. Todo este proceso se encapsula dentro de una transacción para asegurar que, en caso de cualquier fallo, no quede inconsistencia ni mitad de operación en la base de datos.

En caso de que la disponibilidad solicitada no exista o ya haya sido ocupada por otra solicitud, el procedimiento realiza un rollback completo y devuelve un código HTTP adecuado (404 si la franja no se encontró, 409 si ya estaba ocupada), junto a un mensaje de error en JSON que explica la razón. De este modo se evita que el cliente reciba información contradictoria o que la base de datos entre en un estado inconsistente.

Cuando la operación es satisfactoria, el sistema confirma al cliente la creación de la orden devolviendo un código HTTP 200 junto con el identificador único de la nueva orden (idOrden). Este valor permite al

afiliado o a la aplicación cliente realizar posteriores consultas, modificaciones o añadidos a esa orden concreta, completando así el ciclo de gestión de la cita.

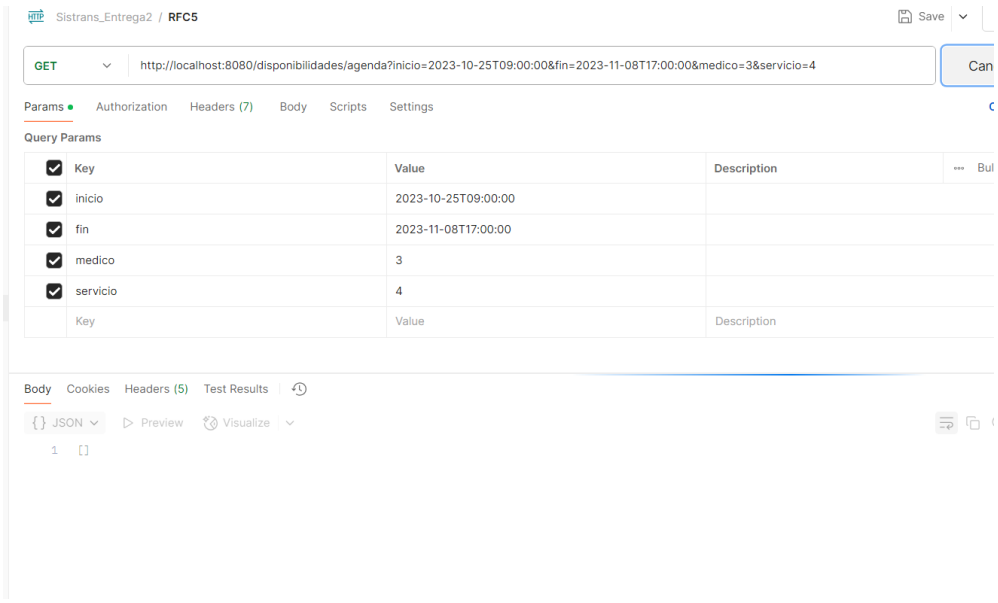


Por otro lado, el RF9 en su sentencia SQL se materializa mediante el procedimiento almacenado SP_AGENDAR_SERVICIO, que orquesta toda la lógica de reserva en una única transacción. Al invocarlo, se le pasan seis parámetros de entrada: el identificador del servicio (p_idServicio), de la IPS (p_nitIps), del médico (p_idMedico), de la franja horaria (p_fechaHoraInicio), del afiliado solicitante (p_idAfiliado) y el texto de la receta (p_receta). Internamente se declaran dos variables: v_estado para capturar el estado actual de la disponibilidad y v_idOrden para almacenar el identificador de la orden recién creada.

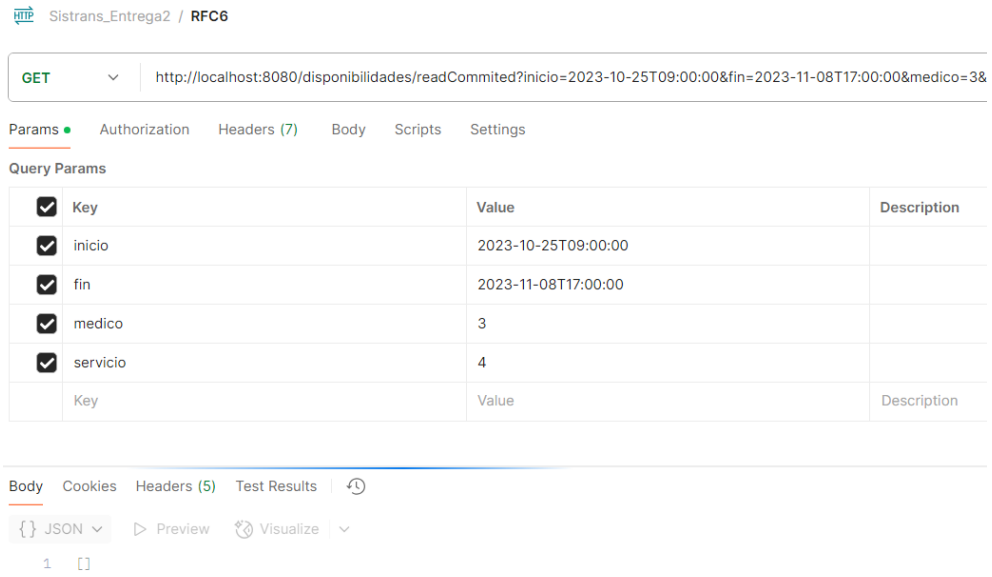
El procedimiento comienza con un SELECT … FOR UPDATE sobre la tabla Disponibilidad, bloqueando la fila que coincide con los cuatro criterios (idServicio, nitIps, idMedico y fechaHoraInicio) y cargando su columna estado en v_estado. Este bloqueo pesimista garantiza que ninguna otra sesión pueda reservar la misma franja al mismo tiempo. Acto seguido, evalúa si v_estado = 'LIBRE'; de ser así, continúa con las operaciones de inserción y actualización.
Si la franja está libre, primero inserta un nuevo registro en OrdenServicios con tipoOrden = 'Agendamiento Servicio', la receta, estado = 'VIGENTE', la fecha actual y las referencias al médico y afiliado; la cláusula RETURNING obtiene el idOrden generado automáticamente. A continuación, genera el vínculo en la tabla puente Orden_Servicio, relacionando esa orden con el servicio seleccionado. Finalmente, actualiza la misma fila de Disponibilidad para marcarla como 'OCUPADO' y ejecuta un COMMIT, confirmando todas las operaciones de forma atómica y evitando inconsistencias.

En caso de que la franja no esté libre (ramas del ELSE) o directamente no exista ninguna fila que coincida (capturado por la excepción NO_DATA_FOUND), el procedimiento realiza un ROLLBACK y emite un mensaje con DBMS_OUTPUT.PUT_LINE indicando “No hay disponibilidad en esta franja horaria” o “Franja horaria no encontrada”, respectivamente. Así se asegura que ninguna operación parcial persista en la base de datos, cumpliendo el principio de todo o nada que define el RF9.

RFC5 y RFC6



RFC5



RFC6

Para los requerimientos de consulta se decidió hacer la implementación de dos nuevas clases nombradas cita y disponibilidad, con el fin de poder llevar un registro tanto de tiempo como de los afiliados quienes hacen uso de servicios médicos dados por órdenes médicas

El requerimiento 5 es igual al 6, tan solo cambia la forma de iniciar la transacción ya que uno es de forma SERIALIZABLE el otro es de forma READCOMMITED, para lograr esto en java se creó un archivo Service que permite la integración de niveles de aislamiento y también la simulación de concurrencia con tiempo.

```java
@Service
public class DisponibilidadServicio {

    private final DisponibilidadRepository repo;
    public DisponibilidadServicio(DisponibilidadRepository repo
        this.repo = repo;
    }
    @Transactional(
    isolation   = Isolation.SERIALIZABLE,
    readOnly    = true,
    rollbackFor = Exception.class
)
public List<AgendaDisponibilidad> consultarAgenda(
        LocalDateTime inicio,
        LocalDateTime fin,
        Integer medicoId,
        Integer servicioId) {
    try {
        // Pausa de 30 segundos (30,000 ms) antes de lanzar la
        System.out.println("Iniciando consulta con parámetros:
```

```java
@Transactional(
    isolation   = Isolation.READ_COMMITTED,
    readOnly    = true,
    rollbackFor = Exception.class
)
public List<AgendaDisponibilidad> consultarAgendaReadCommitted(
        LocalDateTime inicio,
        LocalDateTime fin,
        Integer medicoId,
        Integer servicioId) {
    try {
        // Pausa de 30 segundos (30,000 ms) antes de lanzar la consulta
        System.out.println("Iniciando consulta con parámetros: " +
            "inicio=" + inicio + ", fin=" + fin +
            ", medicoId=" + medicoId + ", servicioId=" + servicioId);
        Thread.sleep(millis:30_000); // 30 segundos como especifica el r
```

En la capa de datos definimos una consulta SQL común para ambos requerimientos: un SELECT sobre la tabla Disponibilidad que une con ServiciosSalud y Medicos, y que aplica filtros opcionales por código de servicio, médico y rango de fechas. Este SQL devuelve el nombre del servicio, la fecha y hora del turno, y el médico responsable, ordenado cronológicamente. La única diferencia entre RFC5 y RFC6 no está en el SQL, sino en el nivel de aislamiento con el que ejecutamos esa consulta dentro de una transacción.

Para RFC5 encapsulamos la llamada en un método de servicio anotado con @Transactional(isolation = Isolation.SERIALIZABLE). De este modo, la base de datos impone el nivel más estricto de aislamiento, previniendo lecturas sucias, no repetibles y fenómenos de filas fantasmas. Justo antes de invocar el repositorio o el EntityManager, introducimos Thread.sleep(30000) para mantener la transacción abierta 30 segundos, lo que permite probar simultaneidad en Postman y verificar que, incluso bajo carga concurrente, no se producen inconsistencias de lectura.

En RFC6, el esquema es idéntico —mismo SQL, misma lógica de parámetros, mismo Thread.sleep(30000)— salvo que cambiamos la anotación a @Transactional(isolation = Isolation.READ_COMMITTED). Con este nivel de aislamiento garantizamos que no leeremos datos no confirmados (es decir, evitan “lecturas sucias”), pero permitimos lecturas no repetibles y fantasmas. Así, durante los 30 segundos de retardo podríamos observar que dos peticiones en paralelo ven diferentes conjuntos de filas si otra transacción compromete nuevos registros intermedios.

Para exponer ambos comportamientos al exterior, creamos dos endpoints REST en Spring Boot:

GET /api/agenda invoca el método serializable.

GET /api/agenda/readCommitted invoca el método con aislamiento Read Committed.

Ambos reciben como parámetros query idServicio, idMedico, fechaInicio y fechaFin en formato ISO-8601. En Postman basta con lanzar peticiones GET a esas URLs y comprobar que en ambos casos la respuesta tarda 30 segundos (por el sleep), pero que sólo con Serializable se elimina por completo cualquier anomalía de concurrencia al probar múltiples solicitudes superpuestas.

3. **Documentación de los escenarios de prueba**
   → **Escenarios de Prueba de Concurrencia 1:**

Ejecutar RFC5 (consulta de disponibilidades) en nivel SERIALIZABLE y dejarla "esperando" 30 segundos. Durante esos 30 segundos, en paralelo, ejecutar RF9 (agendar servicio).

- **Población de tablas:**

```
INSERT INTO Afiliado (idAfiliado, tipoDocumento, nombre, fechaNacimiento, direccion)
VALUES (1001, 'CC', 'Juan Pérez', TO_DATE('1990-01-01', 'YYYY-MM-DD'), 'Calle 123');

INSERT INTO IPSs (nit, nombre, direccion, telefono, horario)
VALUES (2001, 'Clínica ABC', 'Carrera 45', '1234567', '7AM-7PM');

INSERT INTO Medicos (identificacion, nombre, numRegistro, especialidad)
VALUES ('1111', 'Dr. House', 'REG-123', 'Medicina General');

INSERT INTO ServiciosSalud (fecha, descripcion)
VALUES (SYSDATE, 'Consulta General');

INSERT INTO Medico_IPS (idMedico, nit)
VALUES (1, 2001);

INSERT INTO Ips_Servicio (IPSs, idServicio)
VALUES (2001, 1);
```

- **Script para RFC5 (Consultar disponibiliodad):**

```
-- RFC5 - SERIALIZABLE SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN -- Simular sleep de 30 segundos antes de leer dbms_lock.sleep(30);

-- Ahora sí hacer la consulta de disponibilidades SELECT s.descripcion, d.fechaHoraInicio, m.nombre
FROM Disponibilidad d JOIN ServiciosSalud s ON s.idServicio = d.idServicio JOIN Medicos m ON
m.idMedico = d.idMedico WHERE d.idServicio = 1 -- ID del servicio creado AND d.idMedico = 1 -- ID del
médico creado AND d.fechaHoraInicio BETWEEN TIMESTAMP '2025-05-01 00:00:00' AND
TIMESTAMP '2025-05-02 23:59:59'; END; /
```

- **Script para RF9 (Agendar Servicio de Salud)**

```
-- RF9 - AGENDAR SERVICIO SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN SP_AGENDAR_SERVICIO( p_idServicio => 1, p_nitIps => 2001, p_idMedico => 1,
p_fechaHoraInicio => TO_TIMESTAMP('2025-05-01 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
p_idAfiliado => 1001, p_receta => 'Tomar medicamento cada 8 horas' ); END; /
```

- **Resultados**

t0	Se lanza RFC5 (consulta disponibilidad SERIALIZABLE). Hace sleep(30s) antes de leer la tabla Disponibilidad.
t1	Mientras RFC5 aún está durmiendo, se lanza RF9 (agendar servicio) para reservar esa disponibilidad.
t2	RF9 trata de hacer SELECT estado FROM Disponibilidad ... FOR UPDATE. Al ser SERIALIZABLE y estar bloqueada la disponibilidad por RFC5, se queda esperando.
t3	RFC5 despierta y termina su consulta.
t4	RF9 puede continuar, toma la disponibilidad, crea la orden de servicio y cambia el estado a OCUPADO.

Durante la ejecución concurrente, el componente que implementa RF9 (agendar servicio) sí debió esperar a que terminara la ejecución de la consulta de RFC5 para poder registrar la orden de servicio.
Esto se debe a que la transacción de RFC5, configurada con nivel de aislamiento SERIALIZABLE, bloqueó las filas de la tabla Disponibilidad que consultaba, impidiendo que RF9 pudiera realizar su operación SELECT ... FOR UPDATE sobre las mismas filas hasta que RFC5 completara su ejecución.

En resumen: RF9 quedó bloqueado esperando que RFC5 liberara los recursos.

En general, el resultado de la consulta de RFC5 no mostró la orden de servicio ingresada por RF9, porque:

- La consulta de RFC5 se ejecutó antes de que RF9 realizara su commit.
- En el nivel SERIALIZABLE, RFC5 trabaja como si toda la base de datos fuera "un estado congelado" desde el inicio de su transacción.
- Por tanto, RFC5 únicamente vio las disponibilidades existentes antes de que RF9 modificara el estado.

En conclusión, la orden de servicio creada por RF9 no apareció en los resultados de RFC5, ya que RFC5 no observó ningún cambio concurrente.