

Desarrollo Microservicios Gestion de citas medicas

Inmersión .NET
LEIDY STEPHANIA HERNANDEZ VARON



Contexto del Proyecto

El sistema de gestión de citas está diseñado para facilitar la administración de citas médicas, pacientes y recetas en un ecosistema distribuido. Se implementa en .NET Framework 4.8 utilizando la arquitectura Domain-Driven Design (DDD) para garantizar modularidad, mantenibilidad y escalabilidad.

Introducción

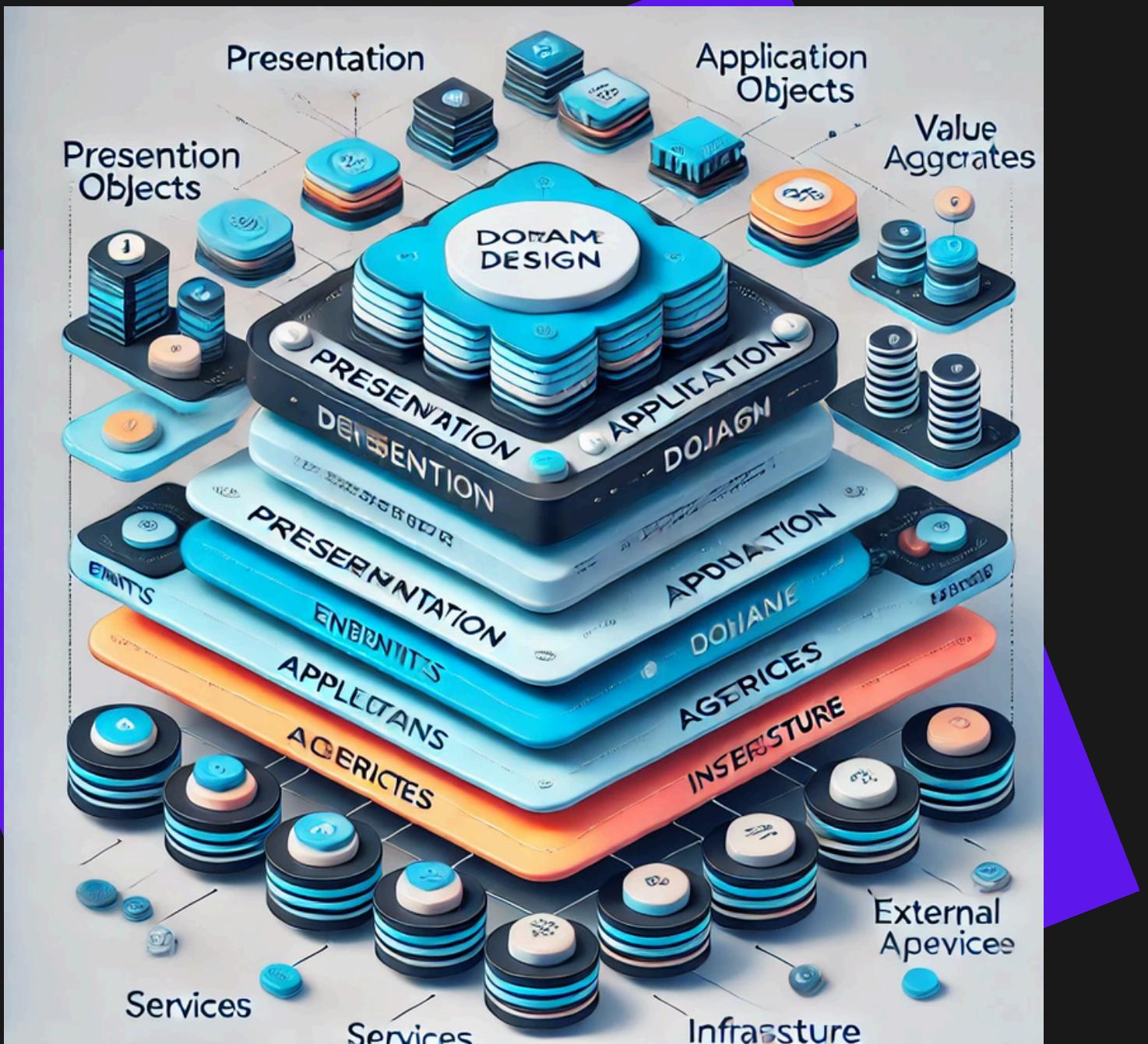
Microservicios Gestión de citas medicas

Objetivos del Proyecto

- Desarrollar un sistema basado en microservicios para la gestión de citas médicas.
- Aplicar la arquitectura DDD para una mejor organización del código.
- Implementar RabbitMQ como sistema de mensajería entre los microservicios.

Arquitectura del Sistema

Microservicios Gestión de citas medicas



Estructura de Microservicios
El sistema se compone de los siguientes microservicios independientes:

- Microservicio de Personas (gestión de pacientes y médicos).
- Microservicio de Citas (manejo de citas médicas).
- Microservicio de Recetas (emisión y administración de recetas médicas).

En este proyecto se implementa comunicación asíncrona (RabbitMQ) y síncrona (API Web) entre microservicios.

Aplicación de DDD

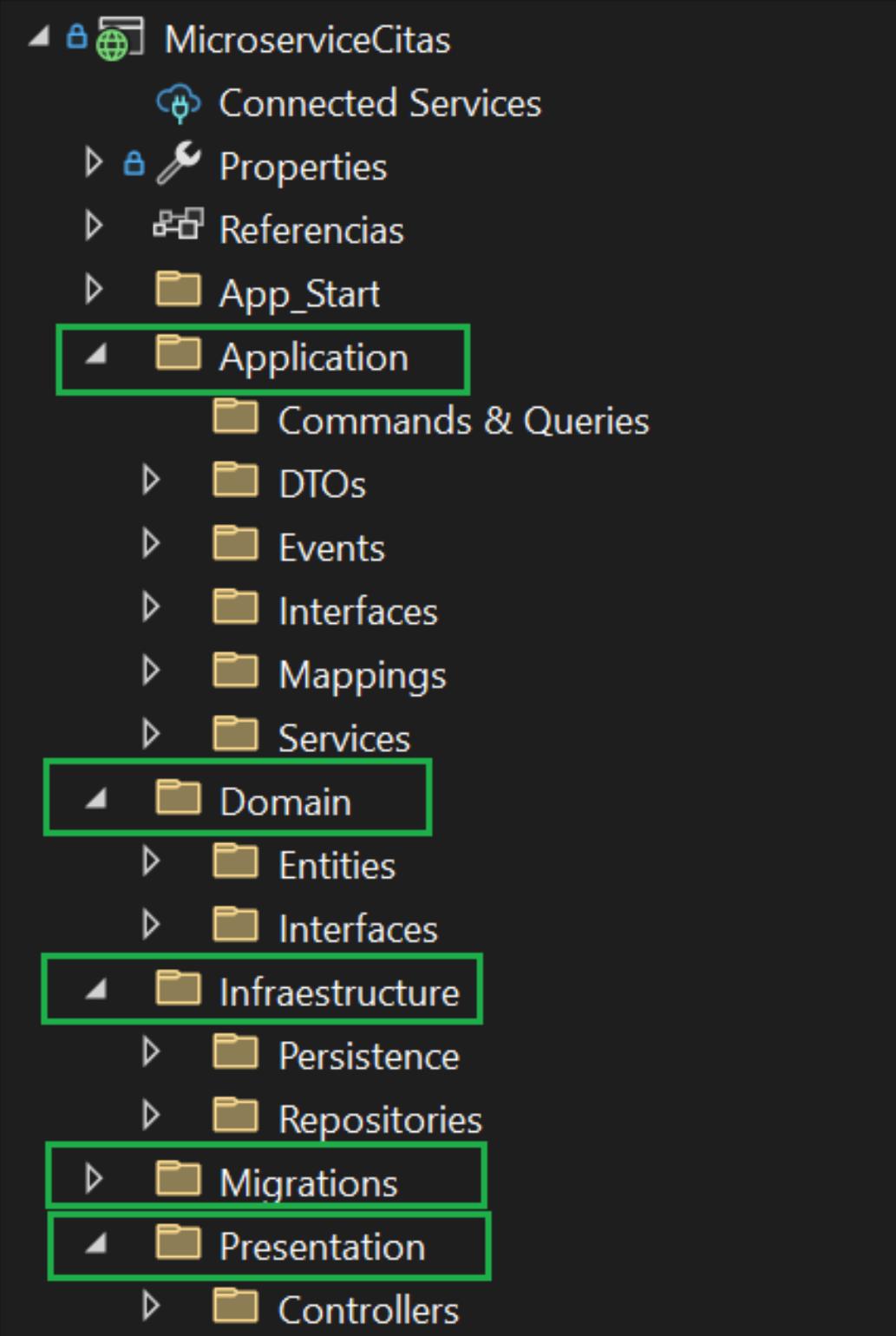
La arquitectura DDD se implementa a través de los siguientes niveles:

- Capa de Dominio: Contiene entidades, agregados, repositorios y servicios de dominio.
- Capa de Aplicación: Define los casos de uso y la lógica de negocio.
- Capa de Infraestructura: Maneja la persistencia de datos y la mensajería con RabbitMQ.
- Capa de Presentación: Expone APIs RESTful para interacción con clientes externos.

Arquitectura del Sistema

Arquitectura basada en Domain-Driven Design (DDD)

Microservicios Gestión de citas medicas



El proyecto sigue los principios de Domain-Driven Design (DDD) organizando el código en capas bien definidas para lograr un diseño modular, mantenible y escalable. La estructura utilizada en cada microservicio es la siguiente:

Application: Contiene la lógica de aplicación, incluyendo DTOs, mapeos, servicios, interfaces y eventos. Aquí se definen las reglas de negocio y las operaciones de cada servicio.

Domain: Representa el núcleo del dominio, incluyendo las entidades y sus interfaces. Define el modelo de negocio sin depender de frameworks externos.

Infrastructure: Maneja la persistencia de datos, implementaciones de repositorios y la configuración de la base de datos mediante Entity Framework.

Presentation: Contiene los controladores (Controllers) que exponen los endpoints para la comunicación con otros servicios o clientes.

Migrations: Administra las migraciones de bases de datos, permitiendo mantener la estructura de datos en los distintos entornos.

Desarrollo Microservicios Gestión de citas medicas

Inmersión .NET



Migraciones y Creación de Bases de Datos con Entity Framework

Para la gestión de bases de datos en cada microservicio se utilizó Entity Framework con Code First Migrations.

Ejemplo de Migración en el Microservicio de Citas

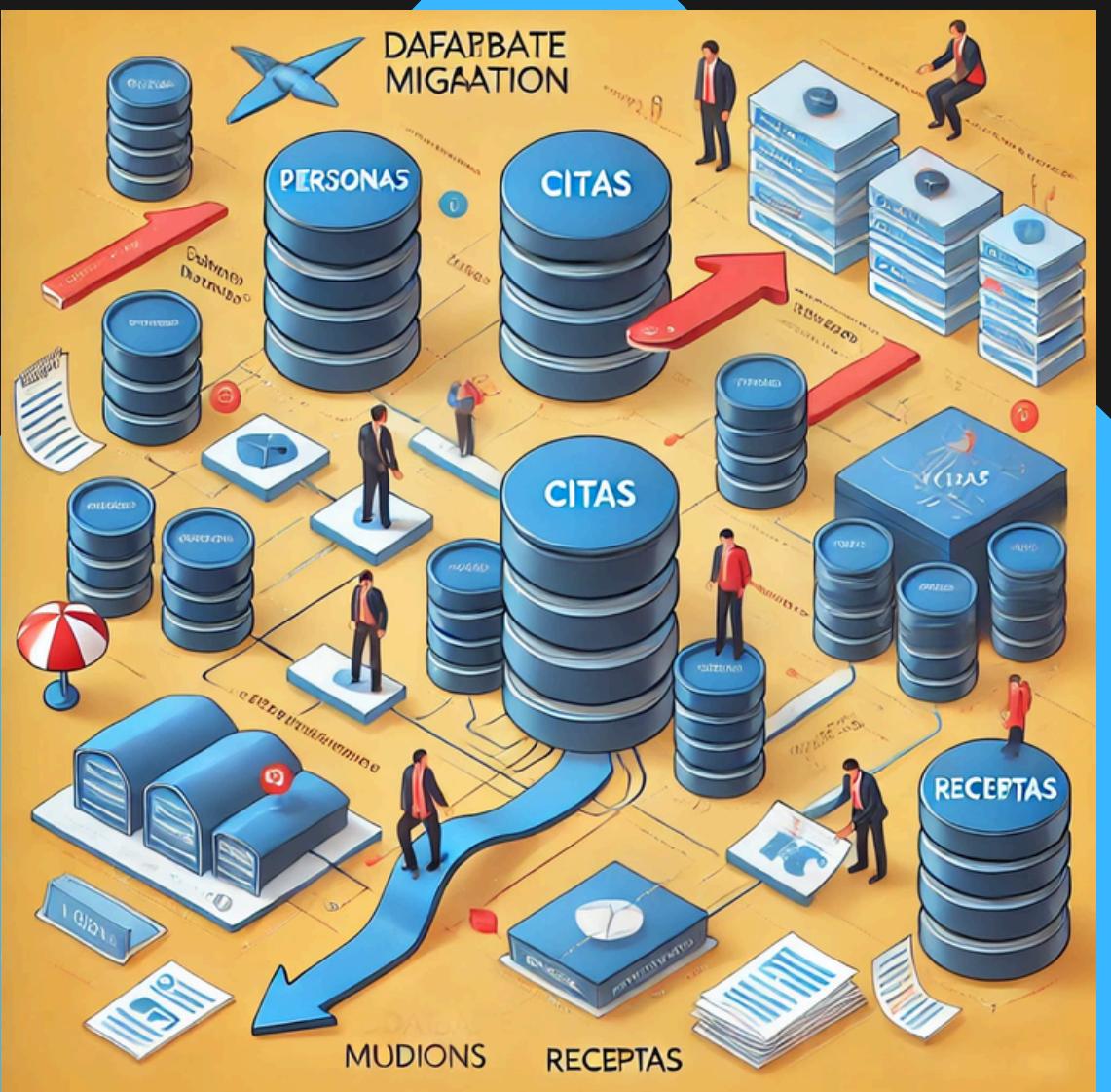
The screenshot shows the Visual Studio IDE interface. The main window displays the code for the `InitialCreate` migration class in the `MicroserviceCitas` project. The code defines a table named `Cita` with columns for Id, Paciente, IdMedico, Motivo, Fecha_Hora, Lugar, and Estado. The `Up()` method creates the table, and the `Down()` method drops it. The solution explorer on the right shows the `GestionCitasMedicas` solution with three projects: `MicroserviceCitas`, `MicroservicePersonas`, and `MicroserviceRecetas`. The `MicroserviceCitas` project is selected, and the `Migrations` folder is expanded, showing the `InitialCreate.cs` file highlighted.

```
public partial class InitialCreate : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Cita",
            c => new
            {
                Id = c.Int(nullable: false, identity: true),
                IdPaciente = c.Int(nullable: false),
                Paciente = c.String(),
                IdMedico = c.Int(nullable: false),
                Medico = c.String(),
                Fecha_Hora = c.DateTime(nullable: false),
                Motivo = c.String(),
                Lugar = c.String(),
                Estado = c.String(),
            })
            .PrimaryKey(t => t.Id);
    }

    public override void Down()
    {
        DropTable("dbo.Cita");
    }
}
```

Desarrollo Microservicios Gestion de citas medicas

Inmersión .NET



Migraciones y Creación de Bases de Datos con Entity Framework

Ejemplo de Contexto en el Microservicio de Personas

Screenshot of the Visual Studio IDE showing the code for the `PersonasDbContext` class. The code defines a database context for the 'Personas' service, inheriting from `DbContext` and overriding the `OnModelCreating` method to map entities to tables.

```
namespace MicroservicePersonas.Infrastructure.Persistence
{
    public class PersonasDbContext : DbContext
    {
        public PersonasDbContext() : base("name=PersonasDbContext")
        {
            this.Configuration.LazyLoadingEnabled = false;
        }

        public DbSet<Persona> Personas { get; set; }
        public DbSet<TipoPersona> TipoPersonas { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Persona>().
                ToTable("Persona");
            modelBuilder.Entity<TipoPersona>()
                .ToTable("TipoPersona");
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

The Visual Studio interface also shows the solution structure on the right, with the `PersonasDbContext.cs` file highlighted in red.

Conexión entre Servicios: Comunicación Síncrona mediante API Web

Escenario: Creación de una Cita

Cuando se crea una cita en el Microservicio de Citas, se necesita validar que el médico y el paciente existen en el Microservicio de Personas. Esto se realiza a través de una API Web síncrona.

Código de Comunicación en Citas:

Microservicios Gestión de citas médicas

```
public async Task<PersonaDTO> GetByIdentificacion(int tipoPersona, string identificacion)
{
    if (tipoPersona <= 0 || (tipoPersona != 1 && tipoPersona != 2))
    {
        throw new System.ArgumentException("El tipo de persona debe ser 1 (Médico) o 2 (Paciente).");
    }

    if (string.IsNullOrEmpty(identificacion))
    {
        throw new System.ArgumentException("La Identificación de la persona no puede ser nula o vacía.");
    }

    var request = new RestRequest(_uri, Method.Get);
    request.AddParameter("TipoPersona", tipoPersona);
    request.AddParameter("Identificacion", identificacion);

    var response = await _client.ExecuteAsync<PersonaDTO>(request);

    if (response.isSuccessful && response.Data != null)
    {
        return response.Data;
    }

    return null;
}
```

Conexión entre Servicios: Comunicación Síncrona mediante API Web

Código de Validación en la Creación de Cita:

Microservicios Gestión de citas médicas

```
public async Task<string> Create(CitaDTO citaDto)
{
    citaDto.Estado = "Pendiente";

    PersonaDTO medico = await _personaService.GetByIdentificacion(1, citaDto.Medico);
    if (medico == null)
    {
        return "El médico no existe.";
    }

    PersonaDTO paciente = await _personaService.GetByIdentificacion(2, citaDto.Paciente);
    if (paciente == null)
    {
        return "El paciente no existe.";
    }

    var cita = _mapper.Map<Cita>(citaDto);

    cita.IdMedico = medico.Id;
    cita.Medico = $"{medico.Nombre} {medico.Apellido}";
    cita.IdPaciente = paciente.Id;
    cita.Paciente = $"{paciente.Nombre} {paciente.Apellido}";

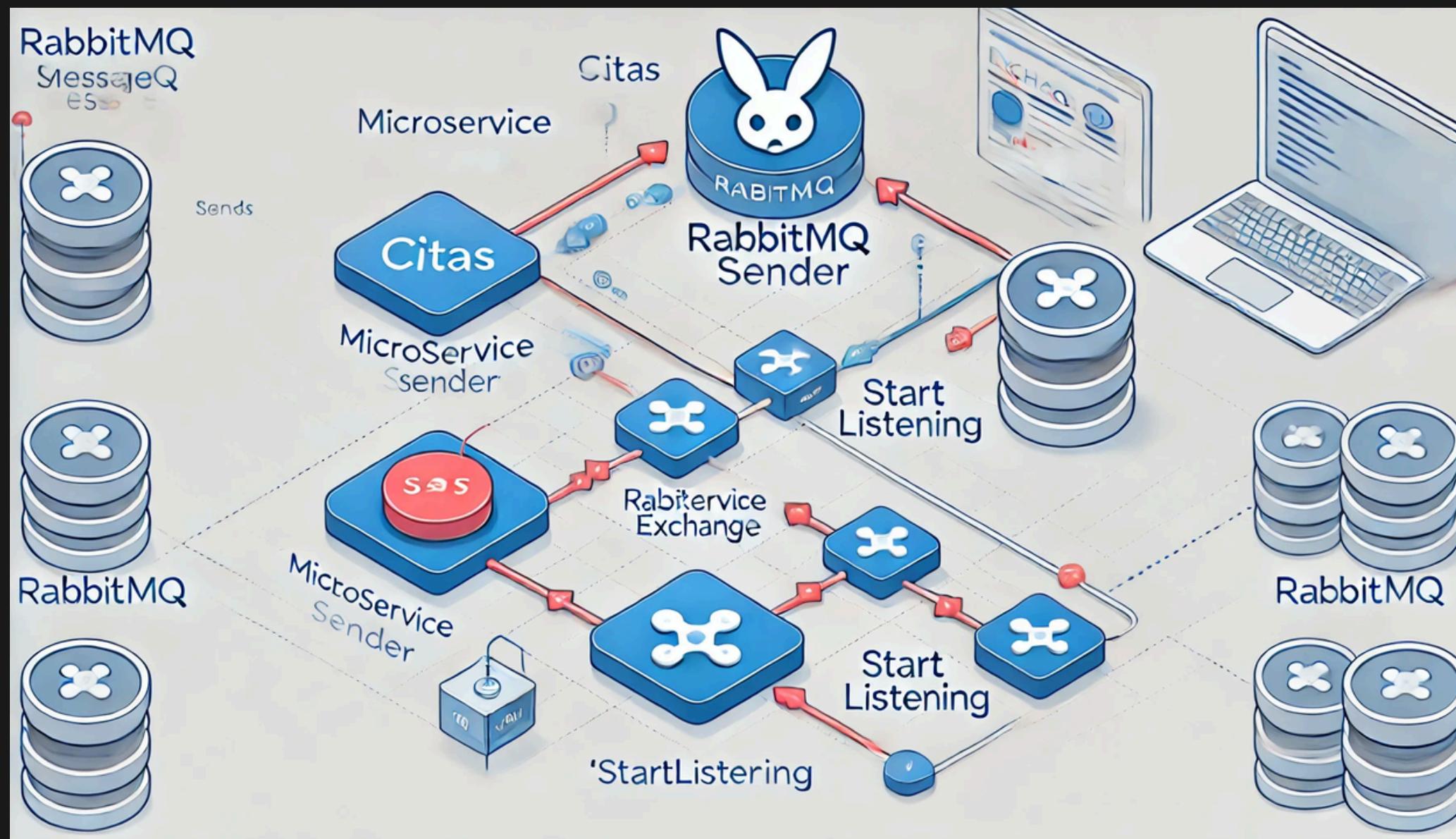
    return await _citaRepository.Create(cita);
}
```

Conexión entre Servicios: Comunicación Asíncrona con RabbitMQ

Después de finalizar una cita, se debe crear automáticamente una receta en el Microservicio de Recetas mediante RabbitMQ:

Microservicios Gestión de citas medicas

1. El Microservicio de Citas actúa como emisor (sender) al publicar un mensaje en la cola de RabbitMQ cuando se finaliza una cita.
2. El Microservicio de Recetas actúa como receptor (listener) para recibir los datos de la receta y crearla automáticamente.



Conexión entre Servicios: Comunicación Asíncrona con RabbitMQ

Microservicios Gestión de citas médicas

Implementación en el Emisor (Microservicio de Citas):

```
2 referencias | IBLINETUM\leidy.hernandez, Hace 1 hora | 1 autor, 1 cambio
public async Task<string> Finish(int id, RecetaDTO recetaDto)
{
    var cita = await _citaRepository.GetById(id);
    if (cita == null)
    {
        return "Cita no encontrada.";
    }
    // Enviar mensaje a RabbitMQ
    _rabbitMqSender.SendMessage(recetaDto);
    var finish = await _citaRepository.Finish(cita);
    return finish;
}
```

Implementación Controller Microservice Cita:

```
[HttpPost]
[Route("Finish/{id}")]
0 referencias | IBLINETUM\leidy.hernandez, Hace 1 hora | 1 autor, 1 cambio
public async Task<IHttpActionResult> Finish(int id, [FromBody] RecetaDTO recetaDto)
{
    if (recetaDto == null)
    {
        return BadRequest("Los datos de la cita no puede ser nula/vacio.");
    }

    try
    {
        string msj = await _citaService.Finish(id, recetaDto);
        return Ok(msj);
    }
    catch (Exception ex)
    {
        return InternalServerError(ex);
    }
}
```

Conexión entre Servicios: Comunicación Asíncrona con RabbitMQ

Implementación servicio SendMessage

Microservicios Gestión de citas medicas

```
public void SendMessage(object message)
{
    var factory = new ConnectionFactory()
    {
        HostName = _hostName,
        Port = _port,
        UserName = _userName,
        Password = _password,
        VirtualHost = _virtualHost
    };

    using (var connection = factory.CreateConnection())
    using (var channel = connection.CreateModel())
    {
        channel.QueueDeclare(queue: _queueName,
            durable: false,
            exclusive: false,
            autoDelete: false,
            arguments: null);

        var body = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(message));

        channel.BasicPublish(exchange: "",
            routingKey: _queueName,
            basicProperties: null,
            body: body);
    }
}
```

Conexión entre Servicios: Comunicación Asíncrona con RabbitMQ

Implementación servicio StartListening

Microservicios Gestión de citas médicas

```
public void StartListening()
{
    var factory = new ConnectionFactory()
    {
        HostName = _hostName,
        Port = _port,
        UserName = _userName,
        Password = _password,
        VirtualHost = _virtualHost
    };

    using (var connection = factory.CreateConnection())
    using (var channel = connection.CreateModel())
    {
        channel.QueueDeclare(queue: _queueName,
            durable: false,
            exclusive: false,
            autoDelete: false,
            arguments: null);

        var consumer = new EventingBasicConsumer(channel);
        consumer.Received += async (model, ea) =>
        {
            var body = ea.Body.ToArray();
            var message = Encoding.UTF8.GetString(body);
            var receta = JsonConvert.DeserializeObject<RecetaDTO>(message);
```

Desarrollo Microservicios Gestión de citas medicas

Inmersión .NET

Conclusiones:

- Se implementó la arquitectura DDD en los microservicios.
- Se realizó la comunicación síncrona entre los microservicios Personas y Citas, así como Recetas y Citas, mediante APIs Web.
- Se implementó comunicación asíncrona con RabbitMQ para la creación automática de recetas.
- Se aseguraron las validaciones necesarias en cada microservicio para evitar inconsistencias en los datos.