UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Trabalho prático nº2

Servidor de cobertura de Wholesaler

Licenciatura em Engenharia Informática

Sistemas Distribuídos 2022/2023

Trabalho elaborado por:
Bruna Silva al74141
Andre Pereira al74066
Francisco al71506

Data: 31/05/2023

Docentes:
Manuel Cunha
Hugo Paredes
Denis Paulino

Protocolo

O modelo **cliente-servidor** é uma estrutura de aplicação distribuída que distribui as tarefas e cargas de trabalho entre os servidores, que fornecem recursos ou serviços, e os clientes, que fazem o pedido desses recursos. Normalmente os clientes e servidores comunicam através de uma rede de computadores, em computadores distintos apesar de estes poderem operar no mesmo computador.
O gRPC é um sistema de código aberto que usa o HTTP/2 para transporte e o Protocol Buffers como linguagem de descrição da interface. Fornece recursos como, por exemplo, autenticação e controlo de fluxo, e gera ligações do tipo cliente-servidor. Os cenários de uso mais comuns incluem serviços de conexão numa arquitetura de micros serviços ou conexão de clientes de dispositivos móveis a serviços de back-end.

1. Servidor e Serviços RPC

O RPC é um protocolo de comunicação entre sistemas que permite a um programa de computador chamar um procedimento noutro computador, ou seja, permite a execução de métodos e procedimentos entre processos remotos. A essência de um RPC é que um serviço é implementado por meio de um procedimento, cujo corpo é executado num servidor. O cliente apenas faz uma chamada RPC ao servidor que recebe e aciona o procedimento propriamente dito, retornando os resultados. O RPC oferece facilidades de comunicação síncrona, uma vez que o cliente é bloqueado até que o servidor envia uma resposta. Para o nosso servidor desenvolvemos os serviços necessários para o total funcionamento da aplicação, utilizamos uma base de dados em sql server e usamos a entity framework para fazer iterações com a base de dados. Desenvolvemos o serviço ProvisioningService que trata de todos os pedidos relacionados com o utilizador. Para o funcionamento destes serviços foram criados os protos necessários.

2. Cliente-Operador externo:

- O cliente pode Registar ou fazer Login.
- Dados: o username, a password, a operadora e isAdmin.
- Permite a visualização do menu do cliente que tem acesso aos processos Reserva, Ativação, Desativação e Terminação.
- Permite a realização das ações dos serviços Reserva, Ativação, Desativação e Terminação.
- Ao fazer a reserva é-lhe dado um número administrativo que lhe será pedido para a realização dos restantes processos.

3. Cliente- Operador/Administrador

- O cliente pode Registar ou fazer Login.
- Dados: o username, a password, a operadora e isAdmin.
- Permite a visualização do menu do Administrador que tem acesso aos processos de Listagem de coberturas disponíveis e Listagem de Operações realizadas.

Implementação

Criamos um sistema cliente/servidor que permite simular que os operadores externos possam aceder ao sistema criado para proceder à Reserva, Ativação, Desativação e Terminação de uma linha de fibra associada a um domicílio, de acordo com uma modalidade definida. Considera-se que o processo de reserva é síncrono e, ao pedido do cliente, o servidor devolve a informação de se foi possível, ou não, realizar a reserva do recurso no domicílio enviado para a modalidade em questão, indicando o número de reserva associado. Os restantes processos (Ativação, Desativação e Terminação) são assíncronos. Assim, quando um operador externo desejar activar, desactivar ou terminar uma linha de fibra irá enviar um pedido ao servidor que será executado a priori e o cliente receberá uma notificação num sistema de subscrição de mensagens ao qual se deverá subscrever anteriormente. Criamos então os models User, Cobertura e operações. De seguida criamos a base de dados com as tabelas e relações correspondentes. Criamos também a proto do provisionamento e os serviços do mesmo, criamos também os serviços de ligação ao RabbitMQ (serviço de gerenciamento de filas para recebimento e entrega de mensagens entre aplicações. Utilizamos também o Docker que é necessário para executar o RabbitMQ num container). Todos os códigos fonte poderão ser consultados em anexo, por isso nesta parte da implementação iremos apenas apresentar a parte executável do projeto.

Temos o menu inicial que nos permite registar um novo user, fazer o login e sair.

```
[Menu Inicial]
[1] - Registar
[2] - Login
[0] - Sair
Escolha uma opção:
```

No menu registo conseguimos registar um novo utilizador e escolher se é administrador ou não, se for necessário.

```
[Registar Utilizador]
Nome de Utilizador:exemploCliente
Palavra-Passe:123
Operador:MEO
É Administrador? (S/N):N
Utilizador registado com sucesso!
Pressione qualquer tecla para continuar...
```

No menu cliente permite-nos fazer as 4 operações previstas: Reserva, Ativação, Desativação e Terminação, bem como sair.

```
[Menu Cliente]
[1] - Reserva
[2] - Ativação
[3] - Desativação
[4] - Terminação
[0] - Sair
Escreva a opção desejada:
```

No menu de reserva, depois de introduzir a Rua e o número a reserva é efetuada e dá-nos o número administrativo.

```
[Reserva]

Rua:rua 4
Número:4
Reserva efetuada com sucesso.
Número Administrativo: 4
```

No menu de Ativação, se introduzirmos o número Administrativo é-nos possível Ativar e o servidor retorna uma mensagem de sucesso.

```
[Ativação]

Número administrativo:4
Ativação iniciada com sucesso
Tempo estimado: 5 segundos

Mensagem recebida: [Server] Sent Servico foi ativado para o exemploCliente com o Numero Administrativo 4.
```

No menu de Desativação, com a introdução do número Administrativo conseguimos Desativar o Serviço e o servidor devolve uma mensagem de sucesso.

```
[Desativação]

Número Administrativo:4

Mensagem recebida: [Server] Sent Servico foi ativado para o exemploCliente com o Numero Administrativo 4.

Mensagem recebida: [Server] Sent Servico foi ativado para o exemploCliente com o Numero Administrativo 4.
Desativação iniciada com sucesso
Tempo estimado: 5 segundos
```

No menu de terminação conseguimos terminar o serviço e retorna-nos a mensagem de sucesso!

```
[Terminação]

Número Administrativo:4

Mensagem recebida: [Server] Sent Servico foi ativado para o exemploCliente com o Numero Administrativo 4.

Mensagem recebida: [Server] Sent Servico foi ativado para o exemploCliente com o Numero Administrativo 4.

Mensagem recebida: [Server] Sent Servico foi ativado para o exemploCliente com o Numero Administrativo 4.
Término iniciado com sucesso
Tempo estimado: 5 segundos
```

Agora registamos um Utilizador do tipo Admin.

```
[Registar Utilizador]
Nome de Utilizador:exemploAdmin
Palavra-Passe:321
Operador:NOS
É Administrador? (S/N):S
Utilizador registado com sucesso!
Pressione qualquer tecla para continuar...
```

A janela de Login, que é igual para todos (Cliente e Admin).

```
[Login]
Username:exemploAdmin
Palavra-Passe:321
```

O menu de Administrador mostra-nos as opções disponíveis para o Administrador, dos Domicílios Disponíveis e as operações realizadas, bem como a opção de sair.

```
[Menu Administrador]
[5] - Domicílios disponíveis
[6] - Operações realizadas
[7] - Sair
Escreva a opção desejada:
```

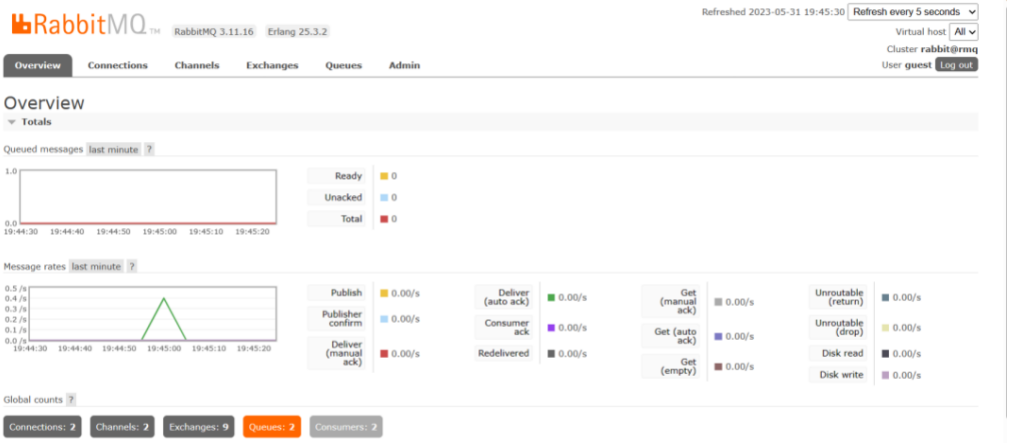
A operação dos domicílios disponíveis, mostra-nos em lista os domicílios que estão com o estado 'FREE'.

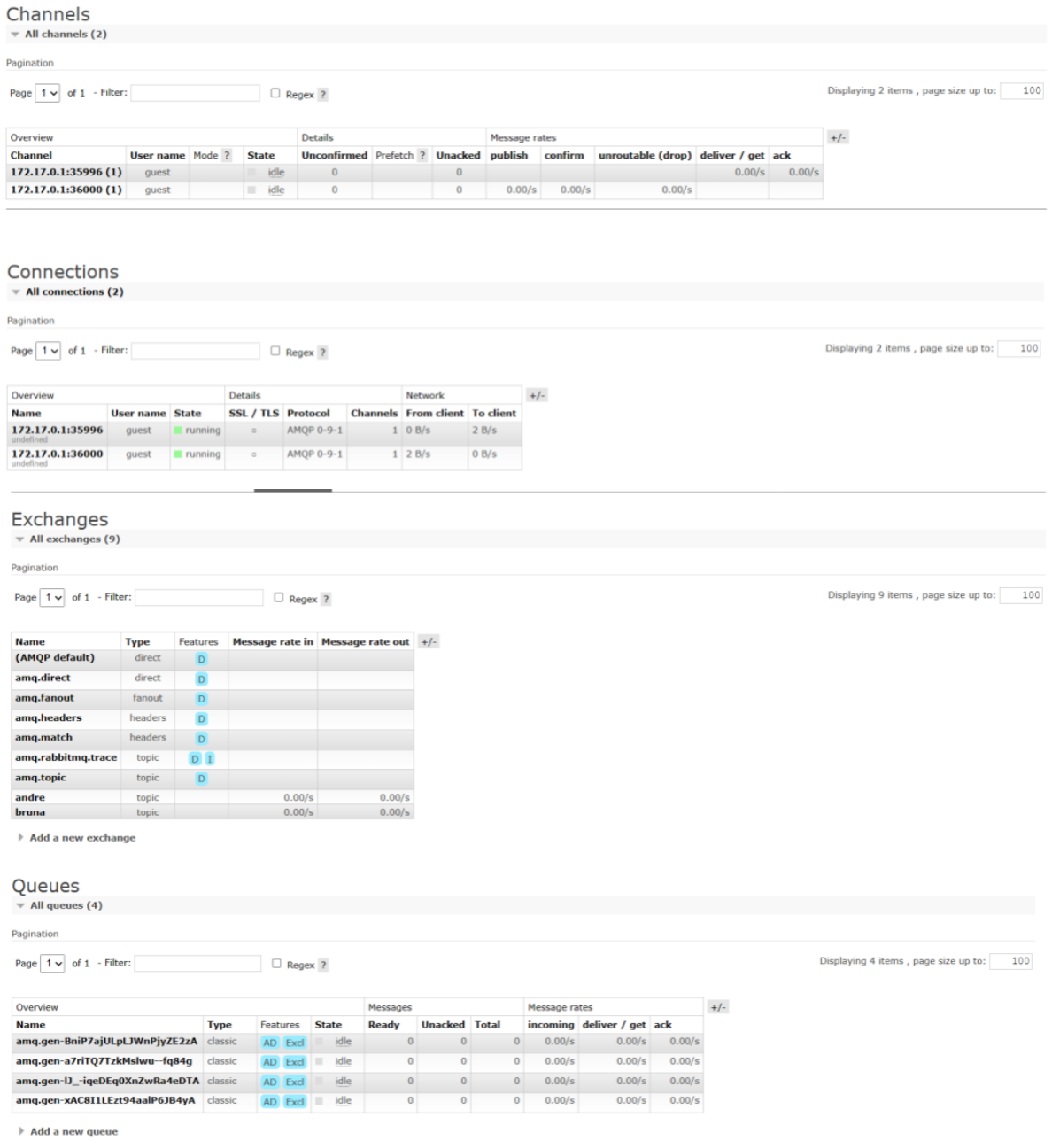
```
[Domicílios disponíveis]

Número administrativo: 3,Rua:z,Número:3, Estado: FREE
Número administrativo: 4,Rua:a,Número:4, Estado: FREE
Número administrativo: 5,Rua:b,Número:5, Estado: FREE
Número administrativo: 6,Rua:c,Número:6, Estado: FREE
```

A operação 'Todas as Operações', mostra-nos em lista todas as operações realizadas por todos os clientes.

```
[Todas as operações]

Sucesso
Operacao ID:2, Operator:Vodafone, Operacao:RESERVATION
Operacao ID:3, Operator:Vodafone, Operacao:RESERVATION
Operacao ID:4, Operator:Vodafone, Operacao:ACTIVATION
Operacao ID:5, Operator:Vodafone, Operacao:ACTIVATION
Operacao ID:6, Operator:Vodafone, Operacao:DEACTIVATION
Operacao ID:7, Operator:Vodafone, Operacao:TERMINATION
Operacao ID:1003, Operator:Nos, Operacao:RESERVATION
Operacao ID:1004, Operator:Nos, Operacao:ACTIVATION
Operacao ID:1005, Operator:Nos, Operacao:DEACTIVATION
Operacao ID:1006, Operator:Nos, Operacao:TERMINATION
Operacao ID:1007, Operator:Meo, Operacao:RESERVATION
Operacao ID:1008, Operator:Meo, Operacao:ACTIVATION
Operacao ID:1009, Operator:Meo, Operacao:DEACTIVATION
Operacao ID:1010, Operator:Meo, Operacao:TERMINATION
Operacao ID:1011, Operator:Vodafone, Operacao:RESERVATION
```

No RabbitMQ temos os seguintes resultados:





Channels

All channels (2)

Pagination

Page 1 of 1 - Filter: Regex ? Displaying 2 items , page size up to: 100

Overview				Details			Message rates				
Channel	User name	Mode ?	State	Unconfirmed	Prefetch ?	Unacked	publish	confirm	unroutable (drop)	deliver / get	ack
172.17.0.1:35996 (1)	guest		idle	0		0				0.00/s	0.00/s
172.17.0.1:36000 (1)	guest		idle	0		0	0.00/s	0.00/s	0.00/s		

Connections

All connections (2)

Pagination

Page 1 of 1 - Filter: Regex ? Displaying 2 items , page size up to: 100

Overview			Details			Network	
Name	User name	State	SSL / TLS	Protocol	Channels	From client	To client
172.17.0.1:35996 undefined	guest	running	○	AMQP 0-9-1	1	0 B/s	2 B/s
172.17.0.1:36000 undefined	guest	running	○	AMQP 0-9-1	1	2 B/s	0 B/s

Exchanges

All exchanges (9)

Pagination

Page 1 of 1 - Filter: Regex ? Displaying 9 items , page size up to: 100

Name	Type	Features	Message rate in	Message rate out
(AMQP default)	direct	D		
amq.direct	direct	D		
amq.fanout	fanout	D		
amq.headers	headers	D		
amq.match	headers	D		
amq.rabbitmq.trace	topic	D I		
amq.topic	topic	D		
andre	topic		0.00/s	0.00/s
bruna	topic		0.00/s	0.00/s

Add a new exchange

Queues

All queues (4)

Pagination

Page 1 of 1 - Filter: Regex ? Displaying 4 items , page size up to: 100

Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
amq.gen-BniP7ajULpLJWnPjyZE2zA	classic	AD Excl	idle	0	0	0	0.00/s	0.00/s	0.00/s
amq.gen-a7riTQ7TzkMslwu--fq84g	classic	AD Excl	idle	0	0	0	0.00/s	0.00/s	0.00/s
amq.gen-IJ_-iqeDEq0XnZwRa4eDTA	classic	AD Excl	idle	0	0	0	0.00/s	0.00/s	0.00/s
amq.gen-xAC8I1LEzt94aalP6JB4yA	classic	AD Excl	idle	0	0	0	0.00/s	0.00/s	0.00/s

Add a new queue

Com isto apresentamos a parte executável do projeto. Em anexo poderá ser consultado todo o código fonte do projeto para um maior entendimento do mesmo.

Anexo - Código Fonte

Link para repositório GitLab:https://gitlab.com/sistemas-distribuidos-pl4/server-client_sd/-/tree/TP2?ref_type=heads

GrpcClient

provisioning.proto:

```
syntax = "proto3";

// Esta opção permite especificar o namespace a ser usado pelo o código
option csharp_namespace = "GrpcServer.Protos";

// Nome do pacote onde todas as mensagens e serviços declarados neste ficheiro estão contidos
package provisioning;

// Definição do serviço Provisioning que declara várias operações RPC
service Provisioning{

  // Define uma função RPC chamada Reserve que recebe uma mensagem do tipo
  //ReserveRequest e retorna uma mensagem do tipo ReserveReply.
  rpc Reserve (ReserveRequest) returns (ReserveReply) {}

  // Similar à função Reserve, mas para a operação Activate.
  rpc Activate (ActivateRequest) returns (ActivateReply) {}

  // Similar à função Reserve, mas para a operação Deactivate.
  rpc Deactivate (DeactivateRequest) returns (DeactivateReply) {}
  rpc Terminate (TerminateRequest) returns (TerminateReply) {}
  rpc MyInfo (MyInfoRequest) returns (MyInfoReply) {}
  rpc FreeCobertura (FreeCoberturaRequest) returns (FreeCoberturaReply) {}
  rpc AllInfo (AllInfoRequest) returns (AllInfoReply) {}
  rpc GetUser (GetUserRequest) returns (GetUserReply){}
  rpc RegisterUser(RegisterUserRequest) returns (RegisterUserReply) {}
}

// Definição da mensagem ReserveRequest.
message ReserveRequest {
  string rua = 1;
  int32 num=2;
  string userID = 3;
  string pass = 4;
}

// Definição da mensagem ReserveReply.
message ReserveReply {
  string num_administrativo=1;
  string result = 2;
}
```

```
message ActivateRequest {
  string num_administrativo = 1;
  string userID = 2;
  string pass = 3;
}

message ActivateReply{
  bool canActivate = 1;
  int32 expectedActivationTime = 2;
  string result =3;

}

message DeactivateRequest {
  string num_administrativo = 1;
  string userID = 2;
  string pass = 3;
}

message DeactivateReply{
  bool canActivate = 1;
  int32 expectedActivationTime = 2;
  string result =3;

}

message TerminateRequest {
    string num_administrativo = 1;
    string userID = 2;
    string pass = 3;
}

message TerminateReply{
    bool canActivate = 1;
    int32 expectedActivationTime = 2;
    string result =3;
}
```

```
message MyInfoRequest{
    string userID = 1;
    string pass = 2;
}

message MyInfoReply {
    string result = 1;
    repeated OperacoesMessage operacoes = 2;
}

message FreeCoberturaRequest{
    string userID = 1;
    string pass = 2;
}

message FreeCoberturaReply{
    string result = 1;
    repeated CoberturaMessage coberturas = 2;
}

message AllInfoRequest{
    string userID = 1;
    string pass = 2;
}

message AllInfoReply{
    string result = 1;
    repeated OperacoesMessage operacoes = 2;
}

message OperacoesMessage {
    int32 id = 1;
    string operator = 2; // Operador envolvido na operação.
    string operacao = 3; // Tipo de operação realizada.
}

message CoberturaMessage {
    string num_admin = 1;   // Número administrativo da cobertura.
    string rua = 2;         // Rua da cobertura.
    int32 nume = 3;         // Número da cobertura.
    string state = 4;       // Estado da cobertura.
}

message GetUserRequest{
    string userID = 1;
    string pass = 2;
}

message GetUserReply{
    string userID = 1;
    string pass = 2;
    bool isAdmin = 3;
    string operator = 4;
}

message RegisterUserRequest{
    string userID = 1;
    string pass = 2;
    bool isAdmin = 3;
    string operator = 4;
}

message RegisterUserReply{
    string userID = 1;
    string pass = 2;
    bool isAdmin = 3;
    string operator = 4;
    string message = 5;
}
```

Program.cs:

```
0 references
public static class Program
{

    20 references
    private static User? User { get; set; } = null; // Estado do utilizador atual, utilizado para manter o estado de login

    0 references
    public static async Task Main(string[] args)...

    // Método assíncrono para registar um novo utilizador
    1 reference
    private static async Task RegisterUser(Provisioning.ProvisioningClient client)...

    1 reference
    private static async Task LoginUser(Provisioning.ProvisioningClient client)...

    1 reference
    private static void ServicoReservar(Provisioning.ProvisioningClient client)
    {

        Console.Write("Rua:");
        string rua = Console.ReadLine();

        Console.Write("Número:");
        string numeroInput = Console.ReadLine();
        // Converte o número da casa para um inteiro
        int numero = int.Parse(numeroInput);

        var request = new ReserveRequest
        {
            Rua = rua,
            Num = numero,
            UserID = User?.Username,
            Pass = User?.Password
        };

        // Envia a solicitação para o servidor e recebe a resposta
        var response = client.Reserve(request);

        Console.WriteLine(response.Result);
        Console.WriteLine("Número Administrativo: " + response.NumAdministrativo);

        Console.ReadLine();
    }
```

```
        private static void ServicoAtivar(Provisioning.ProvisioningClient client)
        {
            Console.Write("Número administrativo:");
            string numInput = Console.ReadLine();
            // Converte o número administrativo para um inteiro
            int num_administrativo = int.Parse(numInput);

            var request = new ActivateRequest
            {
                NumAdministrativo = Convert.ToString(num_administrativo),
                UserID = User?.Username,
                Pass = User?.Password
            };

            // Envia a solicitação para o servidor e recebe a resposta
            var response = client.Activate(request);

            Console.WriteLine(response.Result);
            Console.WriteLine("Tempo estimado: " + response.ExpectedActivationTime + " segundos");
            Console.ReadLine();
        }

        1 reference
        private static void ServicoDesativar(Provisioning.ProvisioningClient client)...

        1 reference
        private static void ServicoTerminar(Provisioning.ProvisioningClient client)...

        1 reference
        private static void AllInfo(Provisioning.ProvisioningClient client)...

        1 reference
        private static void FreeCobertura(Provisioning.ProvisioningClient client)...

    }
}
```

RabbitService.cs:

```
namespace GrpcClient
{
    // Classe interna responsável pela conexão e manipulação do RabbitMQ
    2 references
    internal class RabbitService
    {

        // Criação da fábrica de conexões e da conexão com o RabbitMQ. ...
        public static ConnectionFactory factory = new ConnectionFactory() { HostName = "localhost" };
        public static IConnection connection = factory.CreateConnection();

        // Criação do canal para a comunicação com o RabbitMQ.
        public static IModel channelRabbit = connection.CreateModel();

        // Método que estabelece a conexão com o RabbitMQ e configura a troca e a fila.
        2 references
        public static void ConnectRabitMQ(string topic)
        {
            // Declaração da exchange com o nome do tópico e tipo "Topic".
            channelRabbit.ExchangeDeclare(exchange: topic, type: ExchangeType.Topic);


            // Declaração de uma fila exclusiva.
            var queueName = channelRabbit.QueueDeclare().QueueName;
            // Vinculação da fila à exchange com uma chave de roteamento vazia.
            channelRabbit.QueueBind(queueName, topic, "");

            // Configuração do consumidor para receber as mensagens da fila.
            var consumer = new EventingBasicConsumer(channelRabbit);
            // Criação do evento que é acionado quando uma mensagem é recebida.
            consumer.Received += (model, ea) =>
            {
                var body = ea.Body.ToArray();
                var message = Encoding.UTF8.GetString(body);
                Console.WriteLine("\nMensagem recebida: {0}", message);
            };
            // Inicio das mensagens.
            channelRabbit.BasicConsume(queueName, true, consumer);
        }

    }
}
```

GrpcClient

Program.cs:

```
using GrpcServer.Data;
using GrpcServer.Services;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<SistD2Context>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("SistD2Context")));
// Additional configuration is required to successfully run gRPC on macOS.
// For instructions on how to configure Kestrel and gRPC clients on macOS, visit https://go.microsoft.com/fwlink/?linkid=2099682

// Add services to the container.
builder.Services.AddGrpc();

var app = builder.Build();


// Configure the HTTP request pipeline.
app.MapGrpcService<GreeterService>();
app.MapGrpcService<ProvisioningService>();
app.MapGet("/", () => "Communication with gRPC endpoints must be made through a gRPC" +
" client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");

app.Run();
```

RabbitService.cs:

```
namespace GrpcService.Services
{
    3 references
    public static class RabbitService
    {
        public static readonly dataContext dB = new dataContext();
        public static ConnectionFactory factory = new ConnectionFactory()
        {
            HostName = "localhost"
        };

        public static IConnection connection = factory.CreateConnection();
        // Criação do canal para a comunicação com o RabbitMQ.
        public static IModel channel = connection.CreateModel();


        /// Função para gerar os topicos para o Rabbit MQ
        0 references
        public static void CreateTopics()
        {
            // Listagem das operadoras na base de dados.
            List<string> operadoras = dB.Coberturas.Select(x => x.Operator).Distinct().ToList();

            foreach (var op in operadoras)
            {
                // Declaração da exchange do tipo "topic" para cada operadora.
                channel.ExchangeDeclare(op, ExchangeType.Topic);
            }
            //Declaração da queue
            channel.QueueDeclare(queue: "",
                                 durable: false,
                                 exclusive: false,
                                 autoDelete: false,
                                 arguments: null);
        }
        // Método para enviar uma mensagem para um tópico específico.
        3 references
        public static void SendMessage(string topic, string? message)
        {
            var newMessage = $"[Server] Sent {message}";
            var body = Encoding.UTF8.GetBytes(newMessage);

            // Publicação da mensagem na exchange especificada pelo tópico.
            channel.BasicPublish(exchange: topic,
                                 routingKey: "",
                                 basicProperties: null,
                                 body: body);
            Console.WriteLine(newMessage);
        }

    }
}
```

ProvisioningService.cs:

```
namespace GrpcServer.Services
{
    4 references
    public class ProvisioningService : Provisioning.ProvisioningBase
    {
        private readonly SistD2Context _dbContext;
        private readonly ILogger<ProvisioningService> _logger;

        0 references
        public ProvisioningService(ILogger<ProvisioningService> logger, SistD2Context dBContext)
        {
            _dbContext = dBContext;
            _logger = logger;
        }


        //SERVIÇO DE RESERVA
        3 references
        public override Task<ReserveReply> Reserve(ReserveRequest request, ServerCallContext context)
        {
            try
            {
                var domicilio = _dbContext.Coberturas.FirstOrDefault(d => d.Rua == request.Rua && d.Numero == request.Num);
                if (domicilio == null || domicilio.Estado != "FREE")
                {
                    return Task.FromResult(new ReserveReply { Result = "Não pode reservar este domicilio." });
                }

                var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);

                domicilio.Operator = user.Operator;
                domicilio.Estado = "RESERVED";
                _dbContext.Coberturas.Update(domicilio);
                _dbContext.SaveChanges();

                var operacao = new Operacoes
                {
                    Operacao = "RESERVATION",
                    Operador = user.Operator,
                    NumAdministrativo = domicilio.NumAdmin,
                    Dataatual = DateTime.UtcNow
                };
                _dbContext.Operacoes.Add(operacao);

                _dbContext.SaveChanges();

                return Task.FromResult(new ReserveReply { Result = "Reserva efetuada com sucesso.", NumAdministrativo = domicilio.NumAdmin });
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ocorreu um erro: {ex.Message}");

                throw new RpcException(new Status(StatusCode.Internal, "Ocorreu um erro"));
            }
        }
```

```
//SERVIÇO DE ATIVAÇÃO
3 references
public override async Task<ActivateReply> Activate(ActivateRequest request, ServerCallContext context)
{
    try
    {
        var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);

        var domicilio = _dbContext.Coberturas.FirstOrDefault(d => d.NumAdmin == request.NumAdministrativo);
        if (domicilio == null || (domicilio.Estado != "RESERVED" && domicilio.Estado != "DEACTIVATED") || domicilio.Operator != user.Operator)
        {
            return new ActivateReply { Result = "Não pode ativar este domicilio.", CanActivate = false};
        }
        int estimatedTime = 5; // Tempo estimado em segundos

        await ActivateServiceAsync(user, domicilio, estimatedTime);
        return new ActivateReply { Result = "Ativação iniciada com sucesso", ExpectedActivationTime = estimatedTime, CanActivate = true};
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ocorreu um erro: {ex.Message}");

        throw new RpcException(new Status(StatusCode.Internal, "Ocorreu um erro"));
    }
}
1 reference
private async Task ActivateServiceAsync(User user, Cobertura cobertura, int estimatedTime)
{
    // Atualizar estado da cobertura
    cobertura.Estado = "ACTIVATED";
    _dbContext.Coberturas.Update(cobertura);
    await _dbContext.SaveChangesAsync();

    // Adicionar linha à tabela Operacoes
    var operacao = new Operacoes
    {
        Operacao = "ACTIVATION",
        Operador = user.Operator,
        NumAdministrativo = cobertura.NumAdmin,
        Dataatual = DateTime.UtcNow
    };
    _dbContext.Operacoes.Add(operacao);
    await _dbContext.SaveChangesAsync();

     // Publicar mensagem no tópico EVENTS do RabbitMQ
     string activationMessage = $"Servico foi ativado para o {user.Username} com o Numero Administrativo {cobertura.NumAdmin}.";
     RabbitService.SendMessage(user.Username, activationMessage);

}
```

```
//SERVIÇO DE DESATIVAÇÃO
3 references
public override async Task<DeactivateReply> Deactivate(DeactivateRequest request, ServerCallContext context)
{
    try
    {
        var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);

        var domicilio = _dbContext.Coberturas.FirstOrDefault(d => d.NumAdmin == request.NumAdministrativo);
        if (domicilio == null || domicilio.Estado != "ACTIVATED" || domicilio.Operator != user.Operator)
        {
            return new DeactivateReply { Result = "Não pode desativar este domicilio." };

        }
        int estimatedTime = 5; // Tempo estimado em segundos

        await DeactivateServiceAsync(user, domicilio, estimatedTime);
        return new DeactivateReply { Result = "Desativação iniciada com sucesso", ExpectedActivationTime = estimatedTime };
    }
    catch (Exception ex)
    {
        // Log the exception or return an appropriate error message to the client
        Console.WriteLine($"Ocorreu um erro: {ex.Message}");

        throw new RpcException(new Status(StatusCode.Internal, "Ocorreu um erro"));
    }
}
1 reference
private async Task DeactivateServiceAsync(User user, Cobertura cobertura, int estimatedTime)
{
    // Simular tempo de ativação
    await Task.Delay(TimeSpan.FromSeconds(estimatedTime));

    // Atualizar estado da cobertura
    cobertura.Estado = "DEACTIVATED";
    _dbContext.Coberturas.Update(cobertura);
    await _dbContext.SaveChangesAsync();

    // Adicionar linha à tabela Operacoes
    var operacao = new Operacoes
    {
        Operacao = "DEACTIVATION",
        Operador = user.Operator,
        NumAdministrativo = cobertura.NumAdmin,
        Dataatual = DateTime.UtcNow
    };
    _dbContext.Operacoes.Add(operacao);
    await _dbContext.SaveChangesAsync();

    // Publicar mensagem no tópico EVENTS do RabbitMQ
    string activationMessage = $"Servico foi ativado para o {user.Username} com o Numero Administrativo {cobertura.NumAdmin}.";
    RabbitService.SendMessage(user.Username, activationMessage);
}
```

```
//SERVIÇO DE TERMINO
3 references
public override async Task<TerminateReply> Terminate(TerminateRequest request, ServerCallContext context)
{
    try
    {
        var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);

        var domicilio = _dbContext.Coberturas.FirstOrDefault(d => d.NumAdmin == request.NumAdministrativo);
        if (domicilio == null || domicilio.Estado != "DEACTIVATED" || domicilio.Operator != user.Operator)
        {
            return new TerminateReply { Result = "Não pode terminar este domicilio." };

        }
        int estimatedTime = 5; // Tempo estimado em segundos

        await TerminationServiceAsync(user, domicilio, estimatedTime);
        return new TerminateReply { Result = "Término iniciado com sucesso", ExpectedActivationTime = estimatedTime };
    }
    catch (Exception ex)
    {
        // Log the exception or return an appropriate error message to the client
        Console.WriteLine($"Ocorreu um erro: {ex.Message}");

        throw new RpcException(new Status(StatusCode.Internal, "Ocorreu um erro"));
    }
}
1 reference
private async Task TerminationServiceAsync(User user, Cobertura cobertura, int estimatedTime)
{
    // Simular tempo de ativação
    await Task.Delay(TimeSpan.FromSeconds(estimatedTime));

    // Atualizar estado da cobertura
    cobertura.Estado = "FREE";
    cobertura.Operator = null;
    cobertura.Modalidade = null;

    _dbContext.Coberturas.Update(cobertura);
    await _dbContext.SaveChangesAsync();

    // Adicionar linha à tabela Operacoes
    var operacao = new Operacoes
    {
        Operacao = "TERMINATION",
        Operador = user.Operator,
        NumAdministrativo = cobertura.NumAdmin,
        Dataatual = DateTime.UtcNow
    };
    _dbContext.Operacoes.Add(operacao);
    await _dbContext.SaveChangesAsync();

    // Publicar mensagem no tópico EVENTS do RabbitMQ
    string activationMessage = $"Servico foi ativado para o {user.Username} com o Numero Administrativo {cobertura.NumAdmin}.";
    RabbitService.SendMessage(user.Username, activationMessage);
}
```

```
//Mostrar toda a informação das operaçoes para o admin
3 references
public override async Task<AllInfoReply> AllInfo(AllInfoRequest request, ServerCallContext context)
{
    var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);

    // Get all the Operacoes records
    var operacoes = await _dbContext.Operacoes.ToListAsync();

    // Convert the Operacoes records to OperacoMessage objects
    var operacoesMessages = operacoes.Select(o => new OperacoesMessage
    {
        Id = o.Id,
        Operator = o.Operador,
        Operacao = o.Operacao
        // Map other fields from the Operaco class as needed
    }).ToList();

    // Return the result
    return new AllInfoReply
    {
        Result = "Sucesso",
        Operacoes = { operacoesMessages }
    };

}

//Mostrar os domicilios que estao disponiveis/FREE para o admin
3 references
public override async Task<FreeCoberturaReply> FreeCobertura(FreeCoberturaRequest request, ServerCallContext context)
{
    var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);

    // Get all the Coverage records with state = FREE
    var freeCobertura = await _dbContext.Coberturas.Where(c => c.Estado == "FREE").ToListAsync();

    // Convert the Coverage records to CoverageMessage objects
    var coberturaMessages = freeCobertura.Select(c => new CoberturaMessage
    {
        NumAdmin = c.NumAdmin,
        Rua = c.Rua,
        Nume = (int)c.Numero,
        State = c.Estado
    }).ToList();

    // Return the result
    return new FreeCoberturaReply
    {
        Result = "Sucesso",
        Coberturas = { coberturaMessages }
    };

}
```

```
        //Função que vai buscar a base de dados e comparar a palavra passe e username do user para o login
        3 references
        public override async Task<GetUserReply> GetUser(GetUserRequest request, ServerCallContext context)
        {
            var user = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID && u.Password == request.Pass);
            if (user != null)
            {
                var reply = new GetUserReply
                {
                    UserID = user.Username,
                    Pass = user.Password,
                    IsAdmin = user.IsAdmin,
                    Operator = user.Operator
                };

                return reply;
            }

            // Caso o Utilizador não seja encontrado, retorne null ou um valor padrão
            return null;
        }

        //Função que vai guardar na base de dados  os dados do novo user ao registar
        3 references
        public override Task<RegisterUserReply> RegisterUser(RegisterUserRequest request, ServerCallContext context)
        {
            // Verifique se o Utilizador já está registrado
            var existingUser = _dbContext.Users.FirstOrDefault(u => u.Username == request.UserID);
            if (existingUser != null)
            {
                throw new RpcException(new Status(StatusCode.AlreadyExists, "O Utilizador já está registrado."));
            }

            // Crie um novo objeto User e preencha com os dados da solicitação
            var newUser = new User
            {
                Username = request.UserID,
                Password = request.Pass,
                IsAdmin = request.IsAdmin,
                Operator = request.Operator
            };

            // Salve o novo Utilizador no banco de dados ou em outro local adequado
            _dbContext.Users.Add(newUser);
            _dbContext.SaveChanges();

            // Crie e retorne uma resposta indicando que o Utilizador foi registrado com sucesso
            var reply = new RegisterUserReply
            {
                Message = "Utilizador registrado com sucesso."
            };

            return Task.FromResult(reply);
        }
    }
}
```

Models -> User.cs

```
9 references
public partial class User
{
    [Key]
    [Column("id")]
    1 reference
    public int Id { get; set; }

    [Column("username")]
    [StringLength(255)]
    [Unicode(false)]
    24 references
    public string? Username { get; set; }

    [Column("password")]
    [StringLength(255)]
    [Unicode(false)]
    16 references
    public string? Password { get; set; }

    [Column("operator")]
    [StringLength(255)]
    [Unicode(false)]
    11 references
    public string? Operator { get; set; }

    [Column("isAdmin")]
    7 references
    public bool IsAdmin { get; set; } = false;
}
```

Models -> Operações.cs

```
namespace GrpcServer.Models;


public partial class Operacoes
{
    [Key]
    [Column("id")]

    public int Id { get; set; }

    [Column("operacao")]
    [StringLength(255)]
    [Unicode(false)]

    public string? Operacao { get; set; }

    [Column("operador")]
    [StringLength(255)]
    [Unicode(false)]

    public string? Operador { get; set; }

    [Column("num_administrativo")]

    public string? NumAdministrativo { get; set; }

    [Column("dataatual", TypeName = "datetime")]

    public DateTime? Dataatual { get; set; }
}
```

Models -> Cobertura.cs

```
namespace GrpcServer.Models;

[Table("Cobertura")]
6 references
public partial class Cobertura
{
    [Key]
    [Column("num_administrativo")]
    14 references
    public string? NumAdmin { get; set; }

    [Column("rua")]
    [StringLength(255)]
    [Unicode(false)]
    2 references
    public string? Rua { get; set; }

    [Column("numero")]
    2 references
    public int? Numero { get; set; }

    [Column("operator")]
    [StringLength(255)]
    [Unicode(false)]
    6 references
    public string? Operator { get; set; }

    [Column("modalidade")]
    [StringLength(255)]
    [Unicode(false)]
    1 reference
    public string? Modalidade { get; set; }

    [Column("estado")]
    [StringLength(255)]
    [Unicode(false)]
    11 references
    public string? Estado { get; set; }
}
```

DataContext:

```
namespace GrpcService {

    public class dataContext : DbContext {

        public DbSet<Cobertura> Coberturas { get; set; }
        0 references
        public DbSet<Operacoes> Operacoes { get; set; }
        0 references
        public DbSet<User> Users { get; set; }


        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
            optionsBuilder.UseSqlServer(@"Server=(localdb)\MSSQLLocalDB;Database=DataDB;Trusted_Connection=True;");
        }

    }
}
```