UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Gestão de Clientes e Serviços de Mobilidade

Engenharia Informática

Sistemas Distribuídos

Hugo Paredes

Tiago Pinto

Autores

Gonçalo Costa - 76131

Moisés Santos – 73876

João Marques - 77209

Vila Real, 2024

Índice

1. Introdução

Neste relatório, abordamos a implementação de um sistema distribuído para a gestão de clientes e serviços de mobilidade, com o foco específico no desenvolvimento de um protocolo de comunicação cliente/servidor/administrador.

O Servidor deve:

- Identificar e autenticar cada cliente (mota e administrador) que executa os pedidos.
- Possibilitar que o Administrador possa gerir e listar toda a informação relativa ao serviço ao qual está associado.
- Garantir que cada cliente (mota e administrador) esteja sempre associado a um único serviço.
- O Servidor deve ser programado em .NET Core, RPC e usar RabbitMQ para o tratamento assíncrono de mensagens.

O Administrador deve:

- Receber como parâmetro ou pedir ao utilizador: O nome para identificação do Administrador e a palavra-passe para autenticação do Administrador.
- Ligar ao servidor e permitir listar a informação relativa ao serviço ao qual está associado;
- Ligar ao servidor e permitir introduzir ou atualizar informação relativa ao serviço ao qual está associado, incluindo a introdução de novas tarefas e motas e pessoas.
- Este cliente pode ser programado em .NET Core com uma interface de texto simples (aplicação de linha de comando) ou, em alternativa, como uma aplicação com interface gráfica do Windows (Windows forms).

O Cliente (mota) deve:

- Receber como parâmetro ou pedir ao utilizador: O nome para identificação do utilizador e a palavra-passe para autenticação do utilizador.
- Ligar ao servidor e permitir realizar as ações definidas (alocação e conclusão de uma tarefa; e associação ou desassociação de um serviço).
- Ligar ao servidor e permitir subscrever o tópico definido para as tarefas relativas ao serviço ao qual está associado.
- O cliente pode ser programado para outra plataforma que não o .NET Core, por exemplo para Android, PHP ou Python.

2. Desenvolvimento

2.1 Desenho do protocolo de Comunicação cliente/servidor:

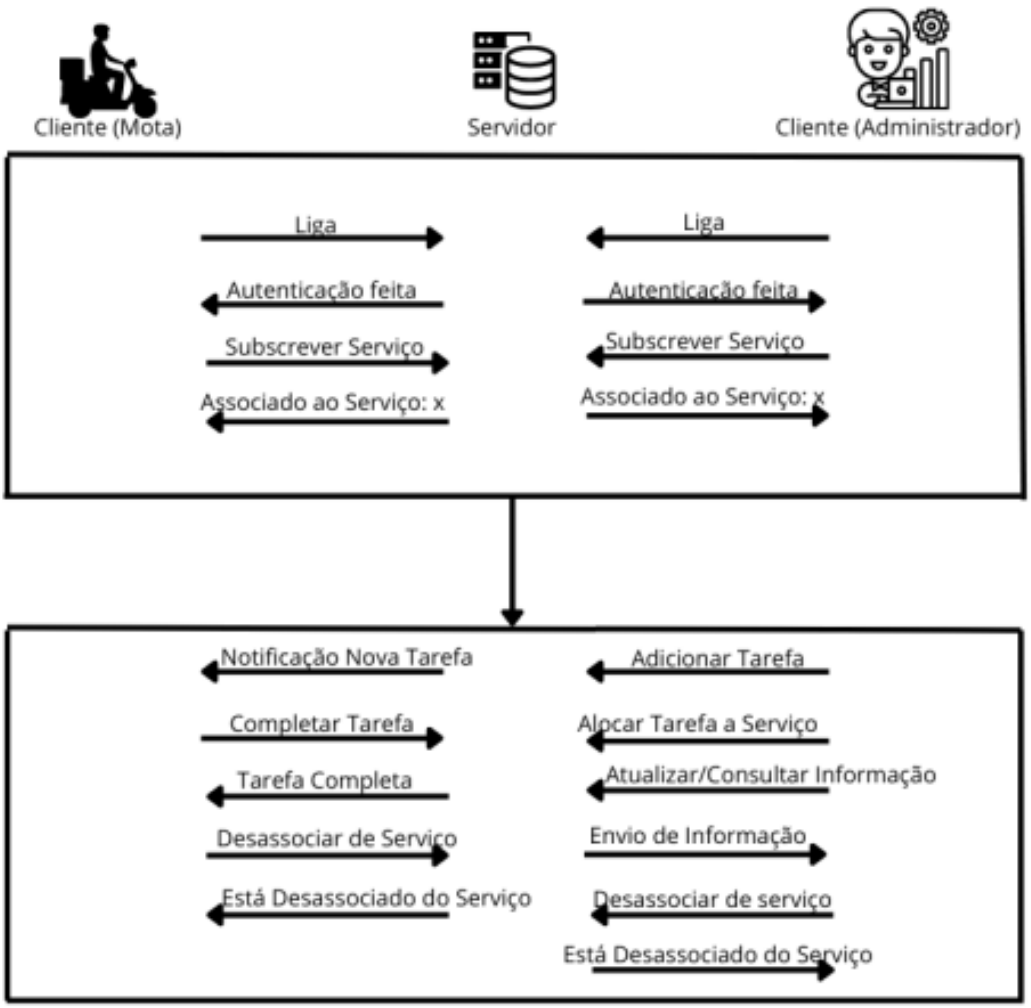

Figura 1 - Desenho do Protocolo

2.2 Implementação

Cliente:

A aplicação **Cliente** é destinada aos utilizadores finais. Permite a criação de novos utilizadores e autenticação de utilizadores existentes. Os utilizadores podem completar tarefas associadas ao serviço ao qual estão subscritos. A aplicação também permite que os utilizadores se subscrevam a serviços para receber novas tarefas e se desassociem de serviços quando necessário. Além disso, inclui uma funcionalidade para receber notificações em tempo real sobre novas tarefas através de subscrições em RabbitMQ.

- Os clientes interagem com o sistema através de um menu de opções.

```
while (true)
{
    Console.WriteLine("1. Completar Tarefa");
    // Outras opções...
    var option = Console.ReadLine();
    switch (option)
    {
        case "1":
            CompleteTask(rpcClient);
            break;
        // Outras opções...
    }
}
```

- **Comunicação com cada cliente:** As mensagens são enviadas para a fila RPC e aguardam respostas.

```
var response = rpcClient.Call(JsonConvert.SerializeObject(request));
```

- **Procedimentos remotos expostos:** Métodos como “CompleteTask”, “SubscriveToTask”, fazem chamadas RPC.

```
var request = new { action = "getTasksForUserService", userId = int.Parse(userId) };
var response = rpcClient.Call(JsonConvert.SerializeObject(request));
var tasks = JsonConvert.DeserializeObject<List<UserTask>>(response);
```

- Mensagens definidas na comunicação assíncrona:

```
var request = new { action = "completeTask", taskId, serviceId = subscribedServiceId, username };
```

Servidor:

A aplicação **Servidor** é responsável por processar diferentes tipos de ações recebidas via RabbitMQ. Estas ações incluem adicionar tarefas, completar tarefas, autenticar utilizadores, criar novos utilizadores, atualizar descrições de tarefas, alocar tarefas a serviços e associar ou desassociar utilizadores de serviços. O servidor interage com uma base de dados SQL Server para armazenar e recuperar informações sobre utilizadores, tarefas e associações de serviços.

Admin:

A aplicação **Admin** é destinada a administradores e permite a criação de novos utilizadores administradores, autenticação de utilizadores existentes e verificação dos privilégios de administrador. Além disso, possibilita a adição de novas tarefas ao serviço associado, atualização da descrição de tarefas existentes, consulta de informações sobre tarefas, e alocação de tarefas a serviços específicos. O administrador também pode associar-se ou desassociar-se de um serviço.

Admin e cliente:

- **Atendimento dos clientes:** Estes componentes são responsáveis por enviar mensagens e aguardar respostas dos servidores.

```
public string Call(string message)
{
    var messageBytes = Encoding.UTF8.GetBytes(message);
    waitHandle.Reset();

    channel.BasicPublish(exchange: "", routingKey: QUEUE_NAME, basicProperties: props, body: messageBytes);

    waitHandle.Wait();

    return response;
}
```

- **Procedimentos remotos expostos**: Não expõem procedimentos diretamente, mas facilitam a comunicação entre os componentes Admin/Cliente e o Servidor.

```
consumer.Received += (model, ea) =>
{
    if (ea.BasicProperties.CorrelationId == correlationId)
    {
        response = Encoding.UTF8.GetString(ea.Body.ToArray());
        waitHandle.Set();
    }
};
```

2.3 Anexo – Código Fonte

Admistrador:

```
using System;
using System.Text;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using Newtonsoft.Json;
using System.Threading;
using System.Collections.Generic;

public class AuthResult
{
    public bool Success { get; set; }
    public int? UserId { get; set; }
    public bool IsAdmin { get; set; }
    public string Message { get; set; }
}

public class UserTask
{
    public int Id { get; set; }
    public string TaskID { get; set; }
    public string Description { get; set; }
    public string Status { get; set; }
    public int ServiceID { get; set; }
    public string ConcluidoPor { get; set; }
}

public class RpcAdminClient : IDisposable
{
    private const string QUEUE_NAME = "rpc_queue";
```

```
    private readonly IConnection connection;
    private readonly IModel channel;
    private readonly string replyQueueName;
    private readonly EventingBasicConsumer consumer;
    private readonly IBasicProperties props;
    private readonly string correlationId;
    private string response;
    private readonly ManualResetEventSlim waitHandle = new
ManualResetEventSlim(false);

    public RpcAdminClient()
    {
        var factory = new ConnectionFactory { HostName = "localhost" };
        connection = factory.CreateConnection();
        channel = connection.CreateModel();
        replyQueueName = channel.QueueDeclare().QueueName;

        consumer = new EventingBasicConsumer(channel);
        correlationId = Guid.NewGuid().ToString();

        props = channel.CreateBasicProperties();
        props.CorrelationId = correlationId;
        props.ReplyTo = replyQueueName;

        consumer.Received += (model, ea) =>
        {
            if (ea.BasicProperties.CorrelationId == correlationId)
            {
                response = Encoding.UTF8.GetString(ea.Body.ToArray());
                waitHandle.Set();
            }
        };
```

```
        channel.BasicConsume(consumer: consumer, queue: replyQueueName, autoAck:
true);
    }

    public string Call(string message)
    {
        var messageBytes = Encoding.UTF8.GetBytes(message);
        waitHandle.Reset();

        channel.BasicPublish(exchange: "", routingKey: QUEUE_NAME, basicProperties:
props, body: messageBytes);

        waitHandle.Wait();

        return response;
    }

    public void Dispose()
    {
        channel.Close();
        connection.Close();
    }
}

public class Admin
{
    private static string adminId;
    private static int? associatedServiceId;

    public static void Main(string[] args)
    {
```

```
try
{
    Console.WriteLine("Cliente Administrador");
    using var rpcClient = new RpcAdminClient();

    bool isAuthenticated = false;
    while (!isAuthenticated)
    {
        Console.WriteLine("1. Criar novo utilizador");
        Console.WriteLine("2. Utilizador já registado");
        Console.Write("Escolha uma opção: ");
        var option = Console.ReadLine();

        switch (option)
        {
            case "1":
                isAuthenticated = CreateUser(rpcClient);
                break;
            case "2":
                isAuthenticated = AuthenticateUser(rpcClient);
                break;
            default:
                Console.WriteLine("Opção inválida.");
                break;
        }
    }

    LoadAssociatedService(rpcClient);

    while (true)
    {
        Console.WriteLine("1. Adicionar Tarefa");
```

```
Console.WriteLine("2. Atualizar Informação");
Console.WriteLine("3. Consultar Informação");
Console.WriteLine("4. Alocar Tarefa a Serviço");
Console.WriteLine("5. Subscrever um Serviço");
Console.WriteLine("6. Desassociar-se de Serviço");
Console.WriteLine("7. Sair");

Console.Write("Escolha uma opção: ");
var option = Console.ReadLine();

switch (option)
{
    case "1":
        AddTask(rpcClient);
        break;
    case "2":
        UpdateInformation(rpcClient);
        break;
    case "3":
        QueryInformation(rpcClient);
        break;
    case "4":
        AllocateTaskToService(rpcClient);
        break;
    case "5":
        AssociateSelfToService(rpcClient);
        break;
    case "6":
        DisassociateSelfFromService(rpcClient);
        break;
    case "7":
        return;
```

```
                default:
                    Console.WriteLine("Opção inválida.");
                    break;
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro: {ex.Message}");
        Console.ReadLine();
    }
}

// Solicita ao utilizador a criação de um novo utilizador administrador
private static bool CreateUser(RpcAdminClient rpcClient)
{
    Console.Write("Nome de utilizador: ");
    var username = Console.ReadLine();
    Console.Write("Senha: ");
    var password = Console.ReadLine();
    var createRequest = new { action = "createUser", username, password, isAdmin =
true };
    var createResponse = rpcClient.Call(JsonConvert.SerializeObject(createRequest));
    Console.WriteLine("Response: {0}", createResponse);

    if (createResponse.Contains("User created"))
    {
        Console.WriteLine("Utilizador criado com sucesso.");
        return AuthenticateUser(rpcClient);
    }
    else
    {
```

```
        Console.WriteLine("Falha ao criar utilizador.");
        return false;
    }
}

// Autentica um utilizador existente
private static bool AuthenticateUser(RpcAdminClient rpcClient)
{
    Console.Write("Nome de utilizador: ");
    var username = Console.ReadLine();
    Console.Write("Senha: ");
    var password = Console.ReadLine();
    var authRequest = new { action = "authenticateUser", username, password };
    var authResponse = rpcClient.Call(JsonConvert.SerializeObject(authRequest));
    var authResult = JsonConvert.DeserializeObject<AuthResult>(authResponse);

    if (authResult == null || !authResult.Success)
    {
        Console.WriteLine("Autenticação falhou.");
        return false;
    }

    if (authResult.UserId.HasValue)
    {
        adminId = authResult.UserId.ToString();
        if (authResult.IsAdmin)
        {
            return true;
        }
        else
        {
            Console.WriteLine("Acesso negado. Utilizador não é administrador.");
```

```
            return false;
        }
    }
    else
    {
        Console.WriteLine("Erro: userId não encontrado.");
        return false;
    }
}

// Carrega o serviço associado ao administrador autenticado
private static void LoadAssociatedService(RpcAdminClient rpcClient)
{
    var request = new { action = "getAssociatedService", userId = int.Parse(adminId) };
    var response = rpcClient.Call(JsonConvert.SerializeObject(request));

    if (int.TryParse(response, out var serviceId))
    {
        associatedServiceId = serviceId;
        Console.WriteLine($"Serviço associado: {associatedServiceId}");
    }
    else
    {
        associatedServiceId = null;
        Console.WriteLine("Nenhum serviço associado encontrado.");
    }
}

// Adiciona uma nova tarefa ao serviço associado
private static void AddTask(RpcAdminClient rpcClient)
{
    if (!associatedServiceId.HasValue)
```

```
        {
            Console.WriteLine("Você não está associado a nenhum serviço.");
            return;
        }

        Console.Write("Descrição da Tarefa: ");
        var taskDescription = Console.ReadLine();
        var request = new { action = "addTask", taskDescription, serviceId =
associatedServiceId.Value };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        Console.WriteLine("Response: {0}", response);
    }

    // Atualiza a informação de uma tarefa existente
    private static void UpdateInformation(RpcAdminClient rpcClient)
    {
        if (!associatedServiceId.HasValue)
        {
            Console.WriteLine("Você não está associado a nenhum serviço.");
            return;
        }

        var request = new { action = "getTasksForAssociatedService", userId =
int.Parse(adminId) };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        var tasks = JsonConvert.DeserializeObject<List<UserTask>>(response);

        if (tasks.Count == 0)
        {
            Console.WriteLine("Nenhuma tarefa disponível para atualização.");
            return;
        }
```

```
        Console.WriteLine("Tarefas disponíveis para atualização:");
        foreach (var task in tasks)
        {
            Console.WriteLine($"ID: {task.Id}, Descrição: {task.Description}, Estado:
{task.Status}");
        }

        Console.Write("Insira o ID da tarefa que deseja atualizar: ");
        var taskId = Console.ReadLine();
        var selectedTask = tasks.Find(t => t.Id.ToString() == taskId);

        if (selectedTask == null)
        {
            Console.WriteLine("Tarefa não encontrada.");
            return;
        }

        Console.WriteLine($"Tarefa selecionada: ID: {selectedTask.Id}, Descrição:
{selectedTask.Description}, Estado: {selectedTask.Status}");
        Console.Write("Nova Descrição da Tarefa: ");
        var newDescription = Console.ReadLine();

        var updateRequest = new
        {
            action = "updateTask",
            taskId = selectedTask.Id,
            newDescription
        };

        var updateResponse =
rpcClient.Call(JsonConvert.SerializeObject(updateRequest));
```

```
        Console.WriteLine("Response: {0}", updateResponse);
    }

    // Consulta e exibe informações sobre as tarefas associadas ao serviço do
administrador
    private static void QueryInformation(RpcAdminClient rpcClient)
    {
        if (!associatedServiceId.HasValue)
        {
            Console.WriteLine("Você não está associado a nenhum serviço.");
            return;
        }

        var request = new { action = "getTasksForAssociatedService", userId =
int.Parse(adminId) };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        var tasks = JsonConvert.DeserializeObject<List<UserTask>>(response);
        Console.WriteLine("Tarefas do Serviço Associado:");
        foreach (var task in tasks)
        {
            Console.WriteLine($"ID: {task.TaskID}, Descrição: {task.Description}, Estado:
{task.Status}, Conluida por: {task.ConcluidoPor}");
        }
    }

    // Aloca uma tarefa específica a um serviço específico
    private static void AllocateTaskToService(RpcAdminClient rpcClient)
    {
        Console.Write("ID da Tarefa: ");
        var taskId = Console.ReadLine();
        Console.Write("ID do Serviço: ");
        var serviceId = Console.ReadLine();
```

```
        var request = new { action = "allocateTaskToService", taskId = int.Parse(taskId),
serviceId = int.Parse(serviceId) };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        Console.WriteLine("Response: {0}", response);
    }

    // Associa o administrador autenticado a um novo serviço
    private static void AssociateSelfToService(RpcAdminClient rpcClient)
    {
        if (associatedServiceId.HasValue)
        {
            Console.WriteLine("Você já está associado a um serviço. Desassocie-se
primeiro para associar-se a outro.");
            return;
        }

        Console.Write("ID do Serviço: ");
        var serviceId = Console.ReadLine();
        var request = new { action = "associateService", userId = int.Parse(adminId),
serviceId = int.Parse(serviceId) };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        Console.WriteLine("Response: {0}", response);

        if (response.Contains("associated"))
        {
            associatedServiceId = int.Parse(serviceId);
        }
    }

    // Desassocia o administrador autenticado do serviço atualmente associado
    private static void DisassociateSelfFromService(RpcAdminClient rpcClient)
    {
```

```
        if (!associatedServiceId.HasValue)
        {
            Console.WriteLine("Você não está associado a nenhum serviço.");
            return;
        }

        var serviceId = associatedServiceId.Value;
        var request = new { action = "disassociateService", userId = int.Parse(adminId),
serviceId };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        Console.WriteLine("Response: {0}", response);

        if (response.Contains("disassociated"))
        {
            associatedServiceId = null;
        }
    }
}
```

Cliente:

```
using System;
using System.Text;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using Newtonsoft.Json;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;

public class AuthResult
{
    public bool Success { get; set; }
    public int? UserId { get; set; }
```

```
    public bool IsAdmin { get; set; }
    public string Message { get; set; }
}

public class UserTask
{
    public int Id { get; set; }
    public string TaskID { get; set; }
    public string Description { get; set; }
    public string Status { get; set; }
    public int ServiceID { get; set; }
    public string ConcluidoPor { get; set; }
}

public class RpcClient : IDisposable
{
    private const string QUEUE_NAME = "rpc_queue";

    private readonly IConnection connection;
    private readonly IModel channel;
    private readonly string replyQueueName;
    private readonly EventingBasicConsumer consumer;
    private readonly IBasicProperties props;
    private readonly string correlationId;
    private string response;
    private readonly ManualResetEventSlim waitHandle = new
ManualResetEventSlim(false);

    public RpcClient()
    {
        var factory = new ConnectionFactory { HostName = "localhost" };
        connection = factory.CreateConnection();
```

```
        channel = connection.CreateModel();
        replyQueueName = channel.QueueDeclare().QueueName;

        consumer = new EventingBasicConsumer(channel);
        correlationId = Guid.NewGuid().ToString();

        props = channel.CreateBasicProperties();
        props.CorrelationId = correlationId;
        props.ReplyTo = replyQueueName;

        consumer.Received += (model, ea) =>
        {
            if (ea.BasicProperties.CorrelationId == correlationId)
            {
                response = Encoding.UTF8.GetString(ea.Body.ToArray());
                waitHandle.Set();
            }
        };

        channel.BasicConsume(consumer: consumer, queue: replyQueueName, autoAck:
true);
    }

    public string Call(string message)
    {
        var messageBytes = Encoding.UTF8.GetBytes(message);
        waitHandle.Reset();

        channel.BasicPublish(exchange: "", routingKey: QUEUE_NAME, basicProperties:
props, body: messageBytes);

        waitHandle.Wait();
```

```
        return response;
    }

    public void Dispose()
    {
        channel.Close();
        connection.Close();
    }
}

public class Client
{
    private static string userId;
    private static string username;
    private static string subscribedServiceId;

    public static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("Cliente");
            using var rpcClient = new RpcClient();

            bool isAuthenticated = false;
            while (!isAuthenticated)
            {
                Console.WriteLine("1. Criar novo utilizador");
                Console.WriteLine("2. Utilizador já registado");
                Console.Write("Escolha uma opção: ");
                var option = Console.ReadLine();
```

```
    switch (option)
    {
        case "1":
            isAuthenticated = CreateUser(rpcClient);
            break;
        case "2":
            isAuthenticated = AuthenticateUser(rpcClient);
            break;
        default:
            Console.WriteLine("Opção inválida.");
            break;
    }
}

LoadAssociatedService(rpcClient);

if (!string.IsNullOrEmpty(subscribedServiceId))
{
    Task.Run(() => SubscribeToNewTasks(subscribedServiceId));
}

while (true)
{
    Console.WriteLine("1. Completar Tarefa");
    Console.WriteLine("2. Subscrever Serviço");
    Console.WriteLine("3. Desassociar Serviço");
    Console.WriteLine("4. Sair");

    Console.Write("Escolha uma opção: ");
    var option = Console.ReadLine();

    switch (option)
```

```
            {
                case "1":
                    CompleteTask(rpcClient);
                    break;
                case "2":
                    SubscribeToTasks(rpcClient);
                    break;
                case "3":
                    DisassociateService(rpcClient);
                    break;
                case "4":
                    return;
                default:
                    Console.WriteLine("Opção inválida.");
                    break;
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro: {ex.Message}");
        Console.ReadLine();
    }
}

// Cria um novo utilizador e tenta autenticar
private static bool CreateUser(RpcClient rpcClient)
{
    Console.Write("Nome de utilizador: ");
    username = Console.ReadLine();
    Console.Write("Senha: ");
    var password = Console.ReadLine();
```

```
        var createRequest = new { action = "createUser", username, password, isAdmin =
false };
        var createResponse = rpcClient.Call(JsonConvert.SerializeObject(createRequest));
        Console.WriteLine("Response: {0}", createResponse);

        if (createResponse.Contains("User created"))
        {
            Console.WriteLine("Utilizador criado com sucesso.");
            return AuthenticateUser(rpcClient);
        }
        else
        {
            Console.WriteLine("Falha ao criar utilizador.");
            return false;
        }
    }

    // Autentica um utilizador existente
    private static bool AuthenticateUser(RpcClient rpcClient)
    {
        Console.Write("Nome de utilizador: ");
        username = Console.ReadLine();
        Console.Write("Senha: ");
        var password = Console.ReadLine();
        var authRequest = new { action = "authenticateUser", username, password };
        var authResponse = rpcClient.Call(JsonConvert.SerializeObject(authRequest));
        var authResult = JsonConvert.DeserializeObject<AuthResult>(authResponse);

        if (authResult == null || !authResult.Success)
        {
            Console.WriteLine("Autenticação falhou.");
            return false;
```

```
    }

    if (authResult.UserId.HasValue)
    {
        userId = authResult.UserId.ToString();
        return true;
    }
    else
    {
        Console.WriteLine("Erro: userId não encontrado.");
        return false;
    }
}

// Carrega o serviço associado ao utilizador autenticado
private static void LoadAssociatedService(RpcClient rpcClient)
{
    var request = new { action = "getAssociatedService", userId = int.Parse(userId) };
    var response = rpcClient.Call(JsonConvert.SerializeObject(request));

    if (int.TryParse(response, out var serviceId))
    {
        subscribedServiceId = serviceId.ToString();
        Console.WriteLine($"Serviço associado: {subscribedServiceId}");
    }
    else
    {
        subscribedServiceId = null;
        Console.WriteLine("Nenhum serviço associado encontrado.");
    }
}
```

```
    // Completa uma tarefa
    private static void CompleteTask(RpcClient rpcClient)
    {
        if (string.IsNullOrEmpty(subscribedServiceId))
        {
            Console.WriteLine("Você não está subscrito em nenhum serviço.");
            return;
        }

        var request = new { action = "getTasksForUserService", userId = int.Parse(userId)
};
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        var tasks = JsonConvert.DeserializeObject<List<UserTask>>(response);

        if (tasks.Count == 0)
        {
            Console.WriteLine("Nenhuma tarefa disponível para completar.");
            return;
        }

        Console.WriteLine("Tarefas disponíveis para completar:");
        foreach (var task in tasks)
        {
            Console.WriteLine($"ID: {task.TaskID}, Descrição: {task.Description}");
        }

        Console.Write("ID da Tarefa: ");
        var taskId = Console.ReadLine();
        var selectedTask = tasks.Find(t => t.TaskID == taskId);

        if (selectedTask == null)
        {
```

```
            Console.WriteLine("Tarefa não encontrada.");
            return;
        }

        var completeRequest = new { action = "completeTask", taskId =
selectedTask.TaskID, serviceId = subscribedServiceId, username };
        var completeResponse =
rpcClient.Call(JsonConvert.SerializeObject(completeRequest));
        Console.WriteLine("Response: {0}", completeResponse);
    }

    // Subscreve a tarefas de um serviço
    private static void SubscribeToTasks(RpcClient rpcClient)
    {
        if (!string.IsNullOrEmpty(subscribedServiceId))
        {
            Console.WriteLine("Você já está subscrito a um serviço. Desassocie-se primeiro
para associar-se a outro.");
            return;
        }

        Console.Write("ID do Serviço para subscrição: ");
        var serviceId = Console.ReadLine();
        var request = new { action = "associateService", userId = int.Parse(userId),
serviceId = int.Parse(serviceId) };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        if (response.Contains("associated"))
        {
            subscribedServiceId = serviceId;
            Task.Run(() => SubscribeToNewTasks(serviceId));
            Console.WriteLine($"Subscrito no serviço {serviceId}.");
        }
```

```
        else
        {
            Console.WriteLine("Falha ao subscrever ao serviço.");
        }
    }

    // Subscreve-se para receber novas tarefas em tempo real
    //Publish/Subscriber
    private static void SubscribeToNewTasks(string serviceId)
    {
        var factory = new ConnectionFactory { HostName = "localhost" };
        var connection = factory.CreateConnection();
        var channel = connection.CreateModel();

        var queueName = $"user_{userId}";

        channel.QueueDeclare(queue: queueName, durable: false, exclusive: false,
autoDelete: false, arguments: null);

        channel.QueueBind(queue: queueName, exchange: "tasks", routingKey:
queueName);

        var consumer = new EventingBasicConsumer(channel);
        consumer.Received += (model, ea) =>
        {
            var body = ea.Body.ToArray();
            var message = Encoding.UTF8.GetString(body);
            DisplayNewTaskNotification(message);
        };

        channel.BasicConsume(queue: queueName, autoAck: true, consumer: consumer);
```

```
        new ManualResetEventSlim(false).Wait();
    }

    // Exibe uma notificação de nova tarefa
    private static void DisplayNewTaskNotification(string message)
    {
        var taskInfo = JsonConvert.DeserializeObject<Dictionary<string,
string>>(message);
        var taskID = taskInfo["TaskID"];
        var description = taskInfo["Description"];

        Console.WriteLine($"Nova Tarefa Recebida - ID: {taskID}, Descrição:
{description}");
    }

    // Desassocia-se de um serviço
    private static void DisassociateService(RpcClient rpcClient)
    {
        if (string.IsNullOrEmpty(subscribedServiceId))
        {
            Console.WriteLine("Você não está subscrito em nenhum serviço.");
            return;
        }

        var serviceId = subscribedServiceId;
        var request = new { action = "disassociateService", userId = int.Parse(userId),
serviceId = int.Parse(serviceId) };
        var response = rpcClient.Call(JsonConvert.SerializeObject(request));
        Console.WriteLine("Response: {0}", response);
        if (response.Contains("disassociated"))
        {
            subscribedServiceId = null;
```

```
            Console.WriteLine("Desassociado do serviço com sucesso.");
        }
    }
}
```

Servidor:

```
using System;
using System.Collections.Generic;
using System.Text;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using Newtonsoft.Json;
using System.Data.SqlClient;
using System.Threading.Tasks;

public class UserTask
{
    public int Id { get; set; }
    public string TaskID { get; set; }
    public string Description { get; set; }
    public string Status { get; set; }
    public int ServiceID { get; set; }
    public string ConcluidoPor { get; set; }
}

public class Program
{
    private static string connectionString =
@"Server=(localdb)\MSSQLLocalDB;Database=servimoto;Integrated
Security=True;Encrypt=False";

    static async Task Main(string[] args)
    {
```

```
var factory = new ConnectionFactory { HostName = "localhost" };
using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

channel.QueueDeclare(queue: "rpc_queue",
                     durable: false,
                     exclusive: false,
                     autoDelete: false,
                     arguments: null);

channel.ExchangeDeclare(exchange: "tasks", type: ExchangeType.Direct);

channel.BasicQos(prefetchSize: 0, prefetchCount: 1, global: false);

var consumer = new EventingBasicConsumer(channel);
channel.BasicConsume(queue: "rpc_queue",
                     autoAck: false,
                     consumer: consumer);

Console.WriteLine(" [x] Awaiting RPC requests");

consumer.Received += async (model, ea) =>
{
    string response = string.Empty;

    var body = ea.Body.ToArray();
    var props = ea.BasicProperties;
    var replyProps = channel.CreateBasicProperties();
    replyProps.CorrelationId = props.CorrelationId;

    try
    {
```

```
            var message = Encoding.UTF8.GetString(body);
            var request = JsonConvert.DeserializeObject<Dictionary<string,
string>>(message);

            Console.WriteLine($" [.] Received request: {request["action"]}");

            switch (request["action"])
            {
                case "addTask":
                    response = await AddTaskAsync(request["taskDescription"],
int.Parse(request["serviceId"]));
                    break;
                case "completeTask":
                    response = await CompleteTaskAsync(request["taskId"],
request["serviceId"], request["username"]);
                    break;
                case "listTasks":
                    response = await ListTasksAsync();
                    break;
                case "allocateTask":
                    response = await AllocateTaskAsync(int.Parse(request["taskId"]),
int.Parse(request["userId"]));
                    break;
                case "authenticateUser":
                    response = await AuthenticateUserAsync(request["username"],
request["password"]);
                    break;
                case "createUser":
                    bool isAdmin = request.ContainsKey("isAdmin") &&
bool.Parse(request["isAdmin"]);
                    response = await CreateUserAsync(request["username"],
request["password"], isAdmin);
```

```
                break;
            case "updateInfo":
                response = await UpdateInfoAsync(request["newInfo"]);
                break;
            case "queryInfo":
                response = await QueryInfoAsync();
                break;
            case "associateService":
                response = await AssociateServiceAsync(int.Parse(request["userId"]),
int.Parse(request["serviceId"]));
                break;
            case "disassociateService":
                response = await DisassociateServiceAsync(int.Parse(request["userId"]),
int.Parse(request["serviceId"]));
                break;
            case "getAssociatedService":
                response = await
GetAssociatedServiceAsync(int.Parse(request["userId"]));
                break;
            case "getTasksForAssociatedService":
                response = await
GetTasksForAssociatedServiceAsync(int.Parse(request["userId"]));
                break;
            case "getTasksForUserService":
                response = await
GetTasksForUserServiceAsync(int.Parse(request["userId"]));
                break;
            case "updateTask":
                response = await UpdateTaskAsync(int.Parse(request["taskId"]),
request["newDescription"]);
                break;
            case "allocateTaskToService":
```

```
                    response = await
AllocateTaskToServiceAsync(int.Parse(request["taskId"]),
int.Parse(request["serviceId"]));
                    break;
                default:
                    response = "Unknown action";
                    break;
            }
        }
        catch (Exception e)
        {
            Console.WriteLine($" [.] Error: {e.Message}");
            response = "Error processing request";
        }
        finally
        {
            var responseBytes = Encoding.UTF8.GetBytes(response);
            channel.BasicPublish(exchange: string.Empty,
                                 routingKey: props.ReplyTo,
                                 basicProperties: replyProps,
                                 body: responseBytes);
            channel.BasicAck(deliveryTag: ea.DeliveryTag, multiple: false);
        }
    };

    Console.WriteLine(" Press [enter] to exit.");
    Console.ReadLine();
}

// Adiciona uma nova tarefa à base de dados
private static async Task<string> AddTaskAsync(string taskDescription, int serviceId)
{
```

```
        Console.WriteLine($" [x] Adding task: {taskDescription}");
        try
        {
            await using var connection = new SqlConnection(connectionString);

            var nextTaskId = await GetNextTaskIDAsync(connection);
            if (nextTaskId == null)
            {
                return "Error generating TaskID";
            }

            var taskID = $"T{nextTaskId:D3}";
            var command = new SqlCommand("INSERT INTO Tarefas (TaskID, Descricao,
Estado, ServicoID) VALUES (@TaskID, @Descricao, @Estado, @ServicoID)",
connection);
            command.Parameters.AddWithValue("@TaskID", taskID);
            command.Parameters.AddWithValue("@Descricao", taskDescription);
            command.Parameters.AddWithValue("@Estado", "Nao alocado");
            command.Parameters.AddWithValue("@ServicoID", serviceId);
            await connection.OpenAsync();
            await command.ExecuteNonQueryAsync();
            var newTask = new UserTask { TaskID = taskID, Description = taskDescription,
Status = "Nao alocado", ConcluidoPor = null, ServiceID = serviceId };
            await NotifyNewTaskAsync(newTask, serviceId);
            return "Task added";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error adding task: {ex.Message}");
            return "Error adding task";
        }
    }
```

```
    // Gera o próximo ID de tarefa
    private static async Task<int?> GetNextTaskIDAsync(SqlConnection connection)
    {
        try
        {
            await using var command = new SqlCommand("SELECT MAX(Id) + 1 FROM
Tarefas", connection);
            await connection.OpenAsync();
            var result = await command.ExecuteScalarAsync();
            return result != DBNull.Value ? (int?)result : 1;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error generating next TaskID: {ex.Message}");
            return null;
        }
        finally
        {
            await connection.CloseAsync();
        }
    }

    // Marca uma tarefa como concluída
    private static async Task<string> CompleteTaskAsync(string taskId, string serviceId,
string username)
    {
        Console.WriteLine($" [x] Completing task: {taskId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
```

```
        var commandCheckStatus = new SqlCommand("SELECT Estado, ServicoID
FROM Tarefas WHERE TaskID = @TaskID", connection);
        commandCheckStatus.Parameters.AddWithValue("@TaskID", taskId);

        await connection.OpenAsync();
        using var reader = await commandCheckStatus.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            var status = reader.GetString(reader.GetOrdinal("Estado"));
            var taskServiceId =
reader.GetInt32(reader.GetOrdinal("ServicoID")).ToString();

            if (taskServiceId != serviceId)
            {
                return "Você não está subscrito nesse serviço.";
            }
            if (status == "Concluido")
            {
                return "A tarefa já está completa";
            }

            reader.Close();
            var commandCompleteTask = new SqlCommand("UPDATE Tarefas SET
Estado = 'Concluido', Concluido_por = @Username WHERE TaskID = @TaskID",
connection);
            commandCompleteTask.Parameters.AddWithValue("@TaskID", taskId);
            commandCompleteTask.Parameters.AddWithValue("@Username",
username);

            int rowsAffected = await commandCompleteTask.ExecuteNonQueryAsync();
            return rowsAffected > 0 ? "Task completed" : "Error completing task";
        }
```

```
        else
        {
            return "Task not found";
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error completing task: {ex.Message}");
        return "Error completing task";
    }
}

// Lista todas as tarefas
private static async Task<string> ListTasksAsync()
{
    Console.WriteLine(" [x] Listing tasks");
    try
    {
        await using var connection = new SqlConnection(connectionString);
        var command = new SqlCommand("SELECT * FROM Tarefas", connection);
        await connection.OpenAsync();
        var reader = await command.ExecuteReaderAsync();
        var tasks = new List<UserTask>();
        while (await reader.ReadAsync())
        {
            tasks.Add(new UserTask
            {
                Id = reader.GetInt32(reader.GetOrdinal("Id")),
                TaskID = reader.GetString(reader.GetOrdinal("TaskID")),
                Description = reader.GetString(reader.GetOrdinal("Descricao")),
                Status = reader.GetString(reader.GetOrdinal("Estado")),
                ServiceID = reader.GetInt32(reader.GetOrdinal("ServicoID")),
```

```
                ConcluidoPor = reader.IsDBNull(reader.GetOrdinal("Concluido_por")) ? null
: reader.GetString(reader.GetOrdinal("Concluido_por"))
            });
        }
        return JsonConvert.SerializeObject(tasks);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error listing tasks: {ex.Message}");
        return "Error listing tasks";
    }
}

// Aloca uma tarefa a um utilizador
private static async Task<string> AllocateTaskAsync(int taskId, int userId)
{
    Console.WriteLine($" [x] Allocating task: {taskId} to user: {userId}");
    try
    {
        await using var connection = new SqlConnection(connectionString);
        var command = new SqlCommand("UPDATE Tarefas SET AllocationID =
@UserID WHERE Id = @TaskID", connection);
        command.Parameters.AddWithValue("@UserID", userId);
        command.Parameters.AddWithValue("@TaskID", taskId);
        await connection.OpenAsync();
        int rowsAffected = await command.ExecuteNonQueryAsync();
        if (rowsAffected > 0)
        {
            return "Task allocated";
        }
        return "Task not found";
    }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"Error allocating task: {ex.Message}");
            return "Error allocating task";
        }
    }

    // Autentica um utilizador
    private static async Task<string> AuthenticateUserAsync(string username, string
password)
    {
        Console.WriteLine($" [x] Authenticating user: {username}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("SELECT * FROM Utilizadores WHERE
Username = @Username AND PasswordHash = @PasswordHash", connection);
            command.Parameters.AddWithValue("@Username", username);
            command.Parameters.AddWithValue("@PasswordHash", password);
            await connection.OpenAsync();
            var reader = await command.ExecuteReaderAsync();
            if (await reader.ReadAsync())
            {
                var userId = reader.GetInt32(reader.GetOrdinal("Id"));
                var isAdmin = reader.GetBoolean(reader.GetOrdinal("IsAdmin"));
                return JsonConvert.SerializeObject(new { success = true, userId, isAdmin });
            }
            return JsonConvert.SerializeObject(new { success = false });
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error authenticating user: {ex.Message}");
```

```
            return JsonConvert.SerializeObject(new { success = false, message =
ex.Message });
        }
    }

    // Cria um novo utilizador
    private static async Task<string> CreateUserAsync(string username, string password,
bool isAdmin)
    {
        Console.WriteLine($" [x] Creating user: {username}");
        try
        {
            await using var connection = new SqlConnection(connectionString);

            var checkUserCommand = new SqlCommand("SELECT COUNT(*) FROM
Utilizadores WHERE Username = @Username", connection);
            checkUserCommand.Parameters.AddWithValue("@Username", username);
            await connection.OpenAsync();
            var userExists = (int)await checkUserCommand.ExecuteScalarAsync() > 0;

            if (userExists)
            {
                return $"User '{username}' already exists";
            }

            var command = new SqlCommand("INSERT INTO Utilizadores (Username,
PasswordHash, IsAdmin) VALUES (@Username, @PasswordHash, @IsAdmin)",
connection);
            command.Parameters.AddWithValue("@Username", username);
            command.Parameters.AddWithValue("@PasswordHash", password);
            command.Parameters.AddWithValue("@IsAdmin", isAdmin);
            int rowsAffected = await command.ExecuteNonQueryAsync();
```

```
            return rowsAffected > 0 ? "User created" : "Failed to create user";
        }
        catch (SqlException sqlEx)
        {
            Console.WriteLine($"SQL Error creating user: {sqlEx.Message}");
            return $"SQL Error creating user: {sqlEx.Message}";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating user: {ex.Message}");
            return $"Error creating user: {ex.Message}";
        }
    }

    // Atualiza informação na base de dados
    private static async Task<string> UpdateInfoAsync(string newInfo)
    {
        Console.WriteLine($" [x] Updating information: {newInfo}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("UPDATE Informacoes SET Info =
@NewInfo WHERE Id = 1", connection);
            command.Parameters.AddWithValue("@NewInfo", newInfo);
            await connection.OpenAsync();
            int rowsAffected = await command.ExecuteNonQueryAsync();
            return rowsAffected > 0 ? "Information updated" : "Update failed";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error updating information: {ex.Message}");
```

```
            return "Error updating information";
        }
    }

    // Consulta informação na base de dados
    private static async Task<string> QueryInfoAsync()
    {
        Console.WriteLine(" [x] Querying information");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("SELECT Info FROM Informacoes WHERE
Id = 1", connection);
            await connection.OpenAsync();
            var result = await command.ExecuteScalarAsync();
            return result != null ? result.ToString() : "No information found";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error querying information: {ex.Message}");
            return "Error querying information";
        }
    }

    // Associa um utilizador a um serviço
    private static async Task<string> AssociateServiceAsync(int userId, int serviceId)
    {
        Console.WriteLine($" [x] Associating user: {userId} to service: {serviceId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
```

```
            var command = new SqlCommand("UPDATE Utilizadores SET ServiceID =
@ServiceID WHERE Id = @UserID", connection);
            command.Parameters.AddWithValue("@ServiceID", serviceId);
            command.Parameters.AddWithValue("@UserID", userId);
            await connection.OpenAsync();
            int rowsAffected = await command.ExecuteNonQueryAsync();
            return rowsAffected > 0 ? "Service associated" : "Association failed";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error associating service: {ex.Message}");
            return "Error associating service";
        }
    }

    // Desassocia um utilizador de um serviço
    private static async Task<string> DisassociateServiceAsync(int userId, int serviceId)
    {
        Console.WriteLine($" [x] Disassociating user: {userId} from service: {serviceId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("UPDATE Utilizadores SET ServiceID =
NULL WHERE Id = @UserID AND ServiceID = @ServiceID", connection);
            command.Parameters.AddWithValue("@UserID", userId);
            command.Parameters.AddWithValue("@ServiceID", serviceId);
            await connection.OpenAsync();
            int rowsAffected = await command.ExecuteNonQueryAsync();
            Console.WriteLine($"Rows affected: {rowsAffected}");
            return rowsAffected > 0 ? "Service disassociated" : "Disassociation failed";
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Error disassociating service: {ex.Message}");
            return $"Error disassociating service: {ex.Message}";
        }
    }

    // Obtém o serviço associado a um utilizador
    private static async Task<string> GetAssociatedServiceAsync(int userId)
    {
        Console.WriteLine($" [x] Querying associated service for user: {userId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("SELECT ServiceID FROM Utilizadores
WHERE Id = @UserID", connection);
            command.Parameters.AddWithValue("@UserID", userId);
            await connection.OpenAsync();
            var result = await command.ExecuteScalarAsync();
            return result != null ? result.ToString() : "No service associated";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error querying associated service: {ex.Message}");
            return "Error querying associated service";
        }
    }

    // Obtém as tarefas para o serviço associado de um utilizador
    private static async Task<string> GetTasksForAssociatedServiceAsync(int userId)
    {
        Console.WriteLine($" [x] Querying tasks for the service associated with user:
{userId}");
```

```
        try
        {
            await using var connection = new SqlConnection(connectionString);

            var commandGetServiceId = new SqlCommand("SELECT ServiceID FROM
Utilizadores WHERE Id = @UserID", connection);
            commandGetServiceId.Parameters.AddWithValue("@UserID", userId);
            await connection.OpenAsync();
            var serviceId = await commandGetServiceId.ExecuteScalarAsync();

            if (serviceId == null)
            {
                return "No service associated with user";
            }

            var commandGetTasks = new SqlCommand("SELECT * FROM Tarefas WHERE
ServicoID = @ServiceID", connection);
            commandGetTasks.Parameters.AddWithValue("@ServiceID", (int)serviceId);
            var reader = await commandGetTasks.ExecuteReaderAsync();
            var tasks = new List<UserTask>();
            while (await reader.ReadAsync())
            {
                tasks.Add(new UserTask
                {
                    Id = reader.GetInt32(reader.GetOrdinal("Id")),
                    TaskID = reader.GetString(reader.GetOrdinal("TaskID")),
                    Description = reader.GetString(reader.GetOrdinal("Descricao")),
                    Status = reader.GetString(reader.GetOrdinal("Estado")),
                    ServiceID = reader.GetInt32(reader.GetOrdinal("ServicoID")),
                    ConcluidoPor = reader.IsDBNull(reader.GetOrdinal("Concluido_por")) ? null
: reader.GetString(reader.GetOrdinal("Concluido_por"))
                });
```

```
            }
            return JsonConvert.SerializeObject(tasks);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error querying tasks: {ex.Message}");
            return "Error querying tasks";
        }
    }

    // Obtém as tarefas para o serviço associado a um utilizador específico
    private static async Task<string> GetTasksForUserServiceAsync(int userId)
    {
        Console.WriteLine($" [x] Querying tasks for the service associated with user:
{userId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);

            var commandGetServiceId = new SqlCommand("SELECT ServiceID FROM
Utilizadores WHERE Id = @UserID", connection);
            commandGetServiceId.Parameters.AddWithValue("@UserID", userId);
            await connection.OpenAsync();
            var serviceId = await commandGetServiceId.ExecuteScalarAsync();

            if (serviceId == null)
            {
                return "No service associated with user";
            }
```

```
            var commandGetTasks = new SqlCommand("SELECT TaskID, Descricao
FROM Tarefas WHERE ServicoID = @ServiceID AND Estado != 'Concluido'",
connection);
            commandGetTasks.Parameters.AddWithValue("@ServiceID", (int)serviceId);
            var reader = await commandGetTasks.ExecuteReaderAsync();
            var tasks = new List<UserTask>();
            while (await reader.ReadAsync())
            {
                tasks.Add(new UserTask
                {
                    TaskID = reader.GetString(reader.GetOrdinal("TaskID")),
                    Description = reader.GetString(reader.GetOrdinal("Descricao")),
                });
            }
            return JsonConvert.SerializeObject(tasks);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error querying tasks: {ex.Message}");
            return "Error querying tasks";
        }
    }

    // Obtém os utilizadores subscritos a um serviço
    private static async Task<List<int>> GetSubscribedUsersAsync(int serviceId)
    {
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("SELECT Id FROM Utilizadores WHERE
ServiceID = @ServiceID", connection);
            command.Parameters.AddWithValue("@ServiceID", serviceId);
```

```
            await connection.OpenAsync();
            var reader = await command.ExecuteReaderAsync();
            var userIds = new List<int>();
            while (await reader.ReadAsync())
            {
                userIds.Add(reader.GetInt32(reader.GetOrdinal("Id")));
            }
            return userIds;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error retrieving subscribed users: {ex.Message}");
            return new List<int>();
        }
    }

    // Notifica novos utilizadores sobre uma nova tarefa
    //Publish/Subscriber
    private static async Task NotifyNewTaskAsync(UserTask task, int serviceId)
    {
        Console.WriteLine($" [x] Notifying new task: {task.TaskID} - {task.Description}");
        try
        {
            var subscribedUsers = await GetSubscribedUsersAsync(serviceId);
            var factory = new ConnectionFactory { HostName = "localhost" };
            using var connection = factory.CreateConnection();
            using var channel = connection.CreateModel();

            channel.ExchangeDeclare(exchange: "tasks", type: ExchangeType.Direct);

            var message = JsonConvert.SerializeObject(new { task.TaskID, task.Description
});
```

```
            var body = Encoding.UTF8.GetBytes(message);

            foreach (var userId in subscribedUsers)
            {
                var routingKey = $"user_{userId}";
                channel.BasicPublish(exchange: "tasks",
                                     routingKey: routingKey,
                                     basicProperties: null,
                                     body: body);
                Console.WriteLine($" [x] Sent {message} to user {userId}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error notifying new task: {ex.Message}");
        }
    }

    // Atualiza a descrição de uma tarefa
    private static async Task<string> UpdateTaskAsync(int taskId, string newDescription)
    {
        Console.WriteLine($" [x] Updating task: {taskId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("UPDATE Tarefas SET Descricao =
@Descricao WHERE Id = @TaskID", connection);
            command.Parameters.AddWithValue("@Descricao", newDescription);
            command.Parameters.AddWithValue("@TaskID", taskId);
            await connection.OpenAsync();
            int rowsAffected = await command.ExecuteNonQueryAsync();
            return rowsAffected > 0 ? "Task updated" : "Update failed";
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error updating task: {ex.Message}");
            return "Error updating task";
        }
    }

    // Aloca uma tarefa a um serviço específico
    private static async Task<string> AllocateTaskToServiceAsync(int taskId, int
serviceId)
    {
        Console.WriteLine($" [x] Allocating task: {taskId} to service: {serviceId}");
        try
        {
            await using var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("UPDATE Tarefas SET ServicoID =
@ServiceID WHERE Id = @TaskID", connection);
            command.Parameters.AddWithValue("@ServiceID", serviceId);
            command.Parameters.AddWithValue("@TaskID", taskId);
            await connection.OpenAsync();
            int rowsAffected = await command.ExecuteNonQueryAsync();
            return rowsAffected > 0 ? "Task allocated to service" : "Allocation failed";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error allocating task to service: {ex.Message}");
            return "Error allocating task to service";
        }
    }
}
```