UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Relatório Projeto 2: Jogo Pedra-Papel-Tesoura com implementação de gRPC

Licenciatura em Engenharia Informática

Sistemas Distribuídos

Hugo Paredes

Arsénio Reis

Dennis Paulino

Autores:

Daniel Guedes: al66551

Luís Moreira: al62679

Vila Real, 2020

1. Introdução

No âmbito da unidade curricular Sistemas Distribuídos, foi-nos proposto o desenvolvimento de um projeto que consistia no mesmo jogo do trabalho prático 1, "Pedra, Papel e Tesoura", no entanto com implementação na **Framework gRPC**.

De modo a cumprir os requisitos estabelecidos para o projeto, o jogo foi desenvolvido e implementando num servidor, onde este potencialmente comunica com vários clientes, através de mensagens definidas por um protocolo gRPC. O gRPC é uma *framework* baseada na tecnologia **RPC** que consiste num método de comunicação remota entre serviços, através da chamada de procedimentos. Permite desta forma, que um determinado processo execute um procedimento que não se encontra no seu espaço de endereçamento.

Para guardar todos os dados de jogadores necessários foi criada uma base de dados, através do **Entity Framework** do ASP.NET, que guarda os dados de autenticação de jogadores, tal como as estatísticas destes jogadores.

Para tornar a interface do cliente mais intuitiva, a aplicação do cliente foi criada com base no **Windows Forms** da *framework* .Net Core, sendo que foram criadas *views* para todas as operações que o cliente pudesse fazer com o servidor.

2. Protocolo de comunicação

Sempre que o cliente abre a aplicação do jogo, é apresentada uma página onde o cliente pode inserir o endereço IP do servidor. Após inserir, é adicionado o *port* referente ao respetivo endereço do servidor, e é criado, no cliente, um canal de comunicações de teste e um cliente de comunicação do tipo **GreeterClient**. O cliente envia um **HelloRequest** ao servidor, e, no caso de obter uma resposta, cria um canal de comunicação que será usado para todas as operações do cliente-servidor. Esta verificação de comunicação de sucesso é feita través de um bloco *try*, *catch* para a exceção **RpcException**. Esta verificação também é feita para qualquer pedido que o cliente envie para o servidor. Todas estas operações de conexão são tratadas pelo **ConnectController**.

Após a conexão ser estabelecida, é iniciada a fase de autenticação, que é tratada pelo **AuthController** no cliente. É iniciado um novo cliente de comunicação, denominado **UserClient**, que irá comunicar com o **UserService** no servidor. Nesta fase, o cliente pode autenticar-se ou criar uma conta.

Para se registar, o cliente terá de inserir um *username*, um *email* e uma *password*. O *username* e o *email* tem de ser únicos, sendo que o cliente não se poderá registar se já existirem. Relativamente à *password*, antes de ser adicionada à base de dados, é encriptada através da classe **SHA512** do C#. O registo é tratado pelo método **Register** do **UserService**.

Para fazer login, o cliente terá de inserir o *username* e a *password*, sendo estes processados pelo método Login do **UserService**. Se o login for feito com sucesso, é gerado um ID de sessão, que é guardado no cliente. Este id será usado para identificar o jogador em cada chamada que o cliente faça ao servidor, uma vez que através deste protocolo, não é possível armazenar o estado de uma sessão. Este ID de sessão é uma *string* alfanumérica de 20 carateres, que é única para cada jogador, e gerada sempre que um jogador faça login. Foi usada um ID de sessão em vez do ID de jogador para identificar o cliente, para reduzir drasticamente a probabilidade de o cliente usar, sem autorização, a conta de outro utilizador através da alteração

do identificador que se encontra na aplicação do cliente. Tal como no **ConnectController**, sempre que é enviada uma mensagem ao servidor, é sempre verificada a ocorrência de qualquer erro.

Após fazer a autenticação, o cliente é reencaminhado para uma *view* que o permite jogar e ver as suas estatísticas. Sempre que esta *view* é aberta, é desencadeada uma função no **PlayController**, que cria um cliente do tipo **GameClient**, responsável por comunicar com o **GameService**. Para verificar se a conexão ainda se mantém, é enviado um pedido do tipo **StatsLookupModel**, com o ID de sessão do cliente, onde o servidor, cria uma linha na tabela de estatísticas de base de dados, no caso do jogador ainda não ter estatísticas associadas, e responde com as estatísticas do cliente, através de uma resposta do tipo **StatsModel**. Caso não ocorra nenhum erro, as estatísticas são carregadas para a *view*, e o jogador pode assim fazer jogadas.

Para realizar uma jogada, o jogador tem 3 botões na *view*, cada um para cada opção, Pedra, Papel e Tesoura. Sempre que o utilizador clica num destes botões, é desencadeado um evento que é "apanhado" pelo **PlayController** do cliente, através da função Play(), que recebe como parâmetro de entrada o ID da jogada, que está associado ao botão que foi pressionado. Dependendo da jogada, envia uma mensagem do tipo **PlayLookupModel**, com o ID da jogada e o ID de sessão. Quando o servidor recebe esta mensagem, faz também uma jogada, e dependendo de ser vitória, empate ou derrota, atualiza as estatísticas do jogador, onde depois envia o resultado através de uma mensagem do tipo **StatsModel**.

Sempre que é feita uma jogada, após receber o resultado, o cliente apresenta o resultado e envia posteriormente uma mensagem do tipo **StatsLookupModel** para receber as estatísticas atualizadas após a jogada.

Tal como referido, sempre que é enviada uma mensagem do cliente para o servidor, é usado sempre um bloco *try and catch*, que é usado para apanhar exceções do tipo **RpcException**, que são lançadas sempre que o cliente não consegue comunicar com o servidor. Se isto acontecer, independentemente da fase em que o cliente estiver (conexão, autenticação ou jogo) é passada a responsabilidade ao **ConnectController**, que vai encerrar todos os clientes usados para cada fase e o canal de comunicação, tal como o ID de sessão é apagado, ou seja, o **ConnectController** tem como responsabilidade repor a aplicação para um estado inicial no caso de alguma exceção ser lançada.

Para demonstrar os métodos dos serviços gRPC criados, temos a seguinte lista:

- **UserService**
  - **Register**, onde o cliente envia dados para se registar, e o servidor retorna se o registo foi feito com sucesso, e no caso de falhar, qual o campo que causou a falha;
  - **Login,** onde o cliente envia dados para se autenticar e o servidor retorna se a autenticação foi feita com sucesso ou não.
- **GameService**
  - **Play,** onde o cliente envia o seu ID de sessão e jogada, e o servidor responde com a sua jogada e o resultado do jogo;
  - **Stats**, onde o cliente envia o seu ID de sessão, e o servidor responde com as estatísticas, no caso de existir um utilizador na base de dados com aquele ID de sessão.

Vídeo do trabalho: https://www.youtube.com/watch?v=o5uON2KTnyM

3. Código

3.1. Protos

```
1  // user.proto //
2
3  syntax = "proto3";
4
5  option csharp_namespace = "GrpcServerRPS";
6
7  service User {
8
9      rpc Login (UserLoginLookupModel) returns (UserLoginModel);
10
11     rpc Regist (UserRegistLookupModel) returns (UserRegistModel);
12
13 }
14
15 message UserLoginLookupModel {
16
17     string username = 1;
18     string password = 2;
19
20 }
21
22 message UserRegistLookupModel {
23
24     string username = 1;
25     string email = 2;
26     string password = 3;
27
28 }
29
30 message UserLoginModel {
31
32     bool valid = 1;  // True: Login feito com sucesso, False: Utilizador/
       Password errados
33     string sessionID = 2;
34 }
35
36 message UserRegistModel {
37
38     int32 valid = 1;    // 1: Válido, -1: Username já existente, -2: Email já
       existente, 3: Username e email já existentes
39 }
```

```
1  // game.proto //
2
3  syntax = "proto3";
4
5  option csharp_namespace = "GrpcServerRPS";
6
7  service Game {
8
9      rpc Play (PlayLookupModel) returns (PlayModel);
10
11     rpc Stats (StatsLookupModel) returns (StatsModel);
12
13 }
14
15 message PlayLookupModel {
16
17     string sessionId = 1;
18     int32 play = 2;
19
20 }
21
22 message StatsLookupModel {
23
24     string sessionId = 1;
25
26 }
27
28 message PlayModel {
29
30     int32 result = 1; // 0: Empate; 1: Utilizador ganhou; 2: Servidor ganhou;
         -1: Utilizador não existe
31     string serverPlay = 2;
32
33 }
34
35 message StatsModel {
36
37     int32 gamesPlayed = 1;  // Se o número de jogos for igual a -1, significa
         que o utilizador não existe
38     int32 wins = 2;
39     int32 draws = 3;
40     int32 losts = 4;
41 }
```

3.2. Cliente

```
1  // Program.cs //
2
3  using Grpc.Net.Client;
4  using GrpcClientWindowsForms.Controllers;
5  using GrpcClientWindowsForms.Views;
6  using System;
7  using System.Windows.Forms;
8
9  namespace GrpcClientWindowsForms
10 {
11     static class Program
12     {
13         // Guarda o ID e Username do utilizador que está a usar o cliente
14         public static AuthenticatedUser AuthUser { get; private set; }
15
16         // Endereço do servidor a qual é feita a conexão
17         public static GrpcChannel ConnectionChannel { get; private set; }
18
19         // Views
20         public static AuthView AuthView { get; private set; }
21         public static ConnectView ConnectView { get; private set; }
22         public static LoginView LoginView { get; private set; }
23         public static PlayView PlayView { get; private set; }
24         public static RegisterView RegisterView { get; private set; }
25
26         // Controllers
27         public static AuthController AuthController { get; private set; }
28         public static ConnectController ConnectController { get; private set; }
29         public static PlayController PlayController { get; private set; }
30
31         [STAThread]
32         static void Main()
33         {
34             Application.SetHighDpiMode(HighDpiMode.SystemAware);
35             Application.EnableVisualStyles();
36             Application.SetCompatibleTextRenderingDefault(false);
37
38             AuthView = new AuthView();
39             ConnectView = new ConnectView();
40             LoginView = new LoginView();
41             PlayView = new PlayView();
42             RegisterView = new RegisterView();
43
44             AuthController = new AuthController();
45             ConnectController = new ConnectController();
46             PlayController = new PlayController();
47
48             Application.Run(ConnectView);
49         }
50
51         // Método para atribuir um valor ao field ServerAddress, que é usado
             para iniciar os channels das conexões GRPC
52         public static void SetConnectionChannel(string address)
53         {
54             ConnectionChannel = GrpcChannel.ForAddress(address);
55         }
```

```
56
57          // Método usado para repor o channel usado para comunicar com o
              servidor
58          public static void ResetConnection()
59          {
60              ConnectionChannel = null;
61          }
62
63          // Método usado para guardar os dados do utilizador autenticado (ID de
              sessão e username) no cliente
64          public static void SetAuthenticatedUser(string sessionID, string
              username)
65          {
66              AuthUser = new AuthenticatedUser(sessionID, username);
67          }
68
69          // Remove os dados do utilizador autenticado no cliente
70          public static void ResetAuthenticatedUser()
71          {
72              AuthUser = null;
73          }
74      }
75  }
76
```

```
1  // AuthenticatedUser.cs //
2
3  namespace GrpcClientWindowsForms
4  {
5      // Guarda os dados do utilizador que está autenticado na view
6      class AuthenticatedUser
7      {
8          public string SessionID { get; private set; }
9          public string Username { get; private set; }
10
11         public AuthenticatedUser(string sessionID, string username)
12         {
13             SessionID = sessionID;
14             Username = username;
15         }
16     }
17 }
18
```

```
1  // ConnectController.cs //
2
3  using Grpc.Net.Client;
4  using GrpcServicerRPC;
5  using System;
6
7  namespace GrpcClientWindowsForms.Controllers
8  {
9      class ConnectController
10     {
11         // Channel e client usados para verificar se é possível realizar uma
               conexão com o endereço enviado
12         public static GrpcChannel TestChannel { get; private set; }
13         public static Greeter.GreeterClient ConnectClient { get; private
               set; }
14
15         public ConnectController()
16         {
17             Program.ConnectView.ConnectRequest += ConnectToServer;
18             Program.ConnectView.RestartConnectionRequest += EndConnection;
19         }
20
21         // Método usado para criar uma conexão com o servidor e testar a
               conexão
22         private async void ConnectToServer(string address)
23         {
24             // Se a conexão já tiver sido estabelecida, é retornado uma
                   mensagem de sucesso para a view
25             if (TestChannel != null || ConnectClient != null)
26             {
27                 Program.ConnectView.SuccessfulConnection();
28                 return;
29             }
30
31             // Estabelece uma conexão com um channel de test com o servidor
                   com o address especificado com o utilizador
32             // No caso de não ter sido especificado endereço IP, retorna uma
                   mensagem de erro para a view
33             try
34             {
35                 TestChannel = GrpcChannel.ForAddress("https://" + address +
                       ":5001");
36                 ConnectClient = new Greeter.GreeterClient(TestChannel);
37             }
38             // No caso de o endereço especificado ter um formato errado
39             catch (UriFormatException)
40             {
41                 EstablishingConnectionFailed();
42                 Program.ConnectView.ShowError("Invalid address!");
43                 return;
44             }
45             // No caso de não ter sido especificado um endereço
46             catch (ArgumentNullException)
47             {
48                 EstablishingConnectionFailed();
49                 Program.ConnectView.ShowError("Invalid address!");
```

```
50                  return;
51              }
52
53              // Após ter sido criada a conexão, é enviado um pedido Hello para verificar se conexão foi feita com sucesso
54              try
55              {
56                  var helloRequest = new HelloRequest
57                  {
58                      Name = "Client"
59                  };
60
61                  var reply = await ConnectClient.SayHelloAsync(helloRequest);
62              }
63              // No caso de a conexão falhar é apanhada a exceção respetiva, e é apresentada uma mensagem de erro na view
64              catch (Grpc.Core.RpcException)
65              {
66                  EstablishingConnectionFailed();
67                  Program.ConnectView.ShowError("Connection failed!");
68                  return;
69              }
70
71
72              // No caso de a conexão de teste suceder, é criado um channel na class Program.cs, para ser usado por todos os
73              // os clients GRPC (AuthClient, ConnectClient e PlayClient) usados para comunicar com o servidor
74              Program.SetConnectionChannel("https://" + address + ":5001");
75              ConnectClient = new Greeter.GreeterClient(Program.ConnectionChannel);
76              TestChannel = null;
77
78              // É avisado o utilizador, através da view, que a conexão foi feita com sucesso
79              Program.ConnectView.SuccessfulConnection();
80          }
81
82          // Usado para repor o channel e clients GRPC usados por todos os controllers
83          private void EndConnection()
84          {
85              Program.PlayController.EndConnection();
86              Program.AuthController.EndConnection();
87              ConnectClient = null;
88              Program.ResetConnection();
89              Program.ConnectView.RestartView();
90          }
91
92          // Método que é chamado sempre que ocorre alguma falha de conexão por qualquer um dos controllers do client
93          public void ConnectionError()
94          {
95              EndConnection();
96              Program.ConnectView.Show();
97              Program.ConnectView.ShowError("Connection failed!");
```

```
 98          }
 99
100          // Método que é chamado sempre que o utilizador autenticado com o
               cliente não existir no servidor
101          public void UserNotFound()
102          {
103              EndConnection();
104              Program.ConnectView.ShowError("User not found!");
105          }
106
107          // No caso da conexão inicial falhar, é usado este método para repor o
               canal e cliente usados para o teste de conexão
108          public void EstablishingConnectionFailed()
109          {
110              TestChannel = null;
111              ConnectClient = null;
112          }
113      }
114  }
115
```

```
1  // AuthController.cs //
2
3  using GrpcServerRPS;
4
5  namespace GrpcClientWindowsForms.Controllers
6  {
7      class AuthController
8      {
9          // Channel e client usados para autenticação
10         public static User.UserClient AuthClient { get; private set; }
11
12         public AuthController()
13         {
14             Program.AuthView.GRPCStartRequest += StartGRPCConnection;
15             Program.LoginView.LoginRequest += Login;
16             Program.RegisterView.RegisterRequest += Register;
17         }
18
19         // Método para inicializar o client GRPC usado para a autenticação
20         private void StartGRPCConnection()
21         {
22             // No caso de a conexão já ter sido inicializada, não necessita de
                   fazer mais nenhuma operação
23             if (AuthClient != null)
24             {
25                 return;
26             }
27
28             // É criado um client de autenticação
29             AuthClient = new User.UserClient(Program.ConnectionChannel);
30         }
31
32         // Método usado para enviar um pedido para o servidor GRPC com o
               intuito de registar um utilizador
33         private async void Register(string username, string email, string
               password, string passwordConfirmation)
34         {
35             // Se o client estiver a null, significa que ocorreu algum erro, e
                   é finalizada a conexão
36             if (AuthClient == null)
37             {
38                 Program.ConnectController.ConnectionError();
39                 return;
40             }
41
42             // Verifica se as passwords inseridas são idênticas
43             if (password != passwordConfirmation)
44             {
45                 Program.RegisterView.ShowError("The passwords don't match!");
46             }
47
48             // Tenta registar o utilizador
49             int validRegistration;
50             try
51             {
52                 UserRegistLookupModel registerRequest = new
```

```
                    UserRegistLookupModel
                {
                    Username = username,
                    Email = email,
                    Password = password
                };

                var outcome = await AuthClient.RegistAsync(registerRequest);
                validRegistration = outcome.Valid;
            }
            // No caso de a conexão com o servidor falhar, é chamado o método de ConnectController para finalizar todas as conexões
            catch (Grpc.Core.RpcException)
            {
                Program.ConnectController.ConnectionError();
                return;
            }

            switch (validRegistration)
            {
                // Registo feito com sucesso
                case 1:
                    Program.RegisterView.SuccessfulRegistration();
                    return;
                // Já existe uma conta registada com o mesmo username
                case -1:
                    Program.RegisterView.ShowError("The username is already used!");
                    return;
                // Já existe uma conta registada com o mesmo email
                case -2:
                    Program.RegisterView.ShowError("The email is already used!");
                    return;
                // Já existe uma conta registada com o mesmo username e mail
                case -3:
                    Program.RegisterView.ShowError("The username and email are already taken!");
                    return;
                // Erro desconhecido
                default:
                    Program.RegisterView.ShowError("Unknown error!");
                    return;
            }
        }

        // Método usado para enviar um pedido para o servidor GRPC com o intuito de autenticar um utilizador
        private async void Login(string username, string password)
        {
            // Se o client estiver a null, significa que ocorreu algum erro, e é finalizada a conexão
            if (AuthClient == null)
            {
                Program.ConnectController.ConnectionError();
                return;
```

```
102             }
103
104             // Tenta autenticar o utilizador
105             try
106             {
107                 UserLoginLookupModel loginRequest = new UserLoginLookupModel
108                 {
109                     Username = username,
110                     Password = password
111                 };
112
113                 var outcome = await AuthClient.LoginAsync(loginRequest);
114
115                 // No caso de autenticação falhar, é enviada a mensagem de
                      erro ao utilizador
116                 if (outcome.Valid == false)
117                 {
118                     Program.LoginView.ShowError("User or/and password do not
                        match any user!");
119                     return;
120                 }
121
122                 // Se a autenticação for feita com sucesso, é guardado o ID de
                       sessão do Username do utilizador no client
123                 Program.SetAuthenticatedUser(outcome.SessionID, username);
124             }
125             // No caso de a conexão com o servidor falhar, é chamado o método
                  de ConnectController para finalizar todas as conexões
126             catch (Grpc.Core.RpcException)
127             {
128                 Program.ConnectController.ConnectionError();
129                 return;
130             }
131
132             // O utilizador é avisado que a autenticação foi feita com sucesso
133             Program.LoginView.SuccessfulLogin();
134         }
135
136         // Se for necessário finalizar a conexão, devido a algum erro ou por
              opção do utilizador, é apagado as informações do utilizador que
137         // está autenticado e é reposto o client usado para a autenticação
138         public void EndConnection()
139         {
140             Program.ResetAuthenticatedUser();
141             Program.AuthView.ResetView();
142             AuthClient = null;
143         }
144     }
145 }
146
```

```
1  // PlayController.cs //
2
3  using GrpcServerRPS;
4
5  namespace GrpcClientWindowsForms.Controllers
6  {
7      class PlayController
8      {
9          // Cliente gRPC responsável por tratar das operações de jogar e
               estatísticas
10         public static Game.GameClient PlayClient { get; private set; }
11
12
13         public PlayController()
14         {
15             Program.PlayView.GRPCStartRequest += StartGRPCConnection;
16             Program.PlayView.PlayRequest += Play;
17         }
18
19         // Método usado para criar uma conexão GRPC com o servidor
20         private async void StartGRPCConnection()
21         {
22             // Estabelece uma conexão com o servidor com o address
                   especificado com o utilizador
23             // No caso de não ter sido especificado endereço IP, retorna uma
                   mensagem de erro para a view
24             StatsModel stats;
25             try
26             {
27                 PlayClient = new Game.GameClient(Program.ConnectionChannel);
28
29                 // Após ter sido criada a conexão, é enviada um pedido para
                       obter as estatísticas do jogador, que é usada tanto para
30                 // testar a conexão e obter as estatísticas do jogador para
                       serem apresentadas na view
31                 var statsRequest = new StatsLookupModel
32                 {
33                     SessionId = Program.AuthUser.SessionID
34                 };
35
36                 stats = await PlayClient.StatsAsync(statsRequest);
37             }
38             // No caso de a conexão falhar é apanhada a exceção respetiva, e é
                   apresentada uma mensagem de erro na view
39             catch (Grpc.Core.RpcException)
40             {
41                 Program.ConnectController.ConnectionError();
42                 return;
43             }
44
45
46             // Se o número de jogados for -1 significa que o utilizador com o
                   ID de sessão não existe no servidor
47             if (stats.GamesPlayed == -1)
48             {
49                 Program.ConnectController.UserNotFound();
```

```
50                  return;
51              }
52
53
54              // Se a conexão for feita com sucesso, são carregadas as estatísticas para a view, e são ativados os butões para jogar
55              Program.PlayView.ShowStats(stats.GamesPlayed, stats.Wins, stats.Draws, stats.Losts);
56              Program.PlayView.EnablePlayButtons();
57              return;
58          }
59
60          // Método usado para realizar uma jogada
61          private async void Play(int play)
62          {
63              // Se o client estiver a null, significa que ocorreu algum erro, e é finalizada a conexão
64              if (PlayClient == null)
65              {
66                  Program.ConnectController.ConnectionError();
67                  return;
68              }
69
70              try
71              {
72                  PlayLookupModel playRequest = new PlayLookupModel
73                  {
74                      SessionId = Program.AuthUser.SessionID,
75                      Play = play
76                  };
77
78                  var outcome = await PlayClient.PlayAsync(playRequest);
79
80                  Program.PlayView.ShowGameOutcome(outcome.Result, outcome.ServerPlay);
81              }
82              // No caso de a conexão falhar é apanhada a exceção respetiva, e é apresentada uma mensagem de erro na view
83              catch (Grpc.Core.RpcException)
84              {
85                  Program.PlayView.ResetAndHide();
86                  Program.ConnectController.ConnectionError();
87                  return;
88              }
89
90
91              // Após ter sido feito o pedido para jogar, é feito o pedido para obter os stats do jogador, para serem mostrados
92              // a medida que o utilizador vai jogando
93              try
94              {
95                  StatsLookupModel statsRequest = new StatsLookupModel
96                  {
97                      SessionId = Program.AuthUser.SessionID
98                  };
99
```

```
100                 var outcome = await PlayClient.StatsAsync(statsRequest);
101
102                 Program.PlayView.ShowStats(outcome.GamesPlayed, outcome.Wins,
                      outcome.Draws, outcome.Losts);
103             }
104             // No caso de a conexão falhar
105             catch (Grpc.Core.RpcException)
106             {
107                 Program.PlayView.ResetAndHide();
108                 Program.ConnectController.ConnectionError();
109                 return;
110             }
111         }
112
113         public void EndConnection()
114         {
115             PlayClient = null;
116         }
117     }
118 }
119
```

3.3. Servidor

```
1  // User.cs //
2
3  using System;
4  using System.ComponentModel.DataAnnotations;
5  using System.ComponentModel.DataAnnotations.Schema;
6
7  namespace GrpcServerRPS.Models
8  {
9      public class User
10     {
11         [Key]
12         public int Id { get; set; }
13         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
14         [Required]
15         [MaxLength(30)]
16         public string Username { get; set; }
17         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
18         [Required]
19         [MaxLength(50)]
20         public string Email { get; set; }
21         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
22         [Required]
23         public string Password { get; set; }
24
25         // É usado um ID único de sessão aleatório em vez de o ID do utilizador, para diminuir a probabilidade de obtenção de um ID de autenticação
26         // de contas de utilizadores através de pesquisas de força bruta
27         [MaxLength(20)]
28         public string SessionID { get; set; }
29
30         [InverseProperty("User")]
31         public virtual History History { get; set; }
32
33
34
35         // Método usado para gerar um ID de sessão para o utilizador
36         public void GenerateSessionID()
37         {
38             // Carateres que o ID de sessão pode conter
39             var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
40
41             var stringChars = new char[20];
42
43             var random = new Random();
44
45             for (int i = 0; i < stringChars.Length; i++)
46             {
47                 stringChars[i] = chars[random.Next(chars.Length)];
48             }
49
50             SessionID = new string(stringChars);
51         }
52     }
53 }
```

```
1  // History.cs //
2
3  using System.ComponentModel.DataAnnotations;
4  using System.ComponentModel.DataAnnotations.Schema;
5
6  namespace GrpcServerRPS.Models
7  {
8      public class History
9      {
10         [Key]
11         public int Id { get; set; }
12         public int userId { get; set; }
13         public int Games { get; set; } = 0;
14         public int lost { get; set; } = 0;
15         public int draw { get; set; } = 0;
16         public int win { get; set; } = 0;
17
18
19         [ForeignKey(nameof(userId))]
20         [InverseProperty("History")]
21         public virtual User User { get; set; }
22     }
23 }
24
```

```
1  // UserService.cs //
2
3  using GrpcServerRPS.Data;
4  using Grpc.Core;
5  using Microsoft.Data.SqlClient;
6  using Microsoft.EntityFrameworkCore;
7  using Microsoft.Extensions.Logging;
8  using System;
9  using System.Linq;
10 using System.Security.Cryptography;
11 using System.Text;
12 using System.Threading.Tasks;
13
14
15 namespace GrpcServerRPS.Services
16 {
17     public class UserService : User.UserBase
18     {
19         private readonly ILogger<UserService> logger;
20         private readonly RPSGameDbContext _context;
21
22         public UserService(ILogger<UserService> logger, RPSGameDbContext
             context)
23         {
24             this.logger = logger;
25             _context = context;
26         }
27
28
29         /**
30          * Login de utilizadores já existentes. Devem ser recebidos o username
              e password.
31          * Sempre que o login é feito com sucesso, é enviado um ID de sessão
              único para o cliente
32          * que é usado por este para se identificar sempre que envia algum
              pedido.
33          */
34         public override Task<UserLoginModel> Login(UserLoginLookupModel
             request, ServerCallContext context)
35         {
36             UserLoginModel output = new UserLoginModel();
37
38             // Transformação de todos os carateres do nome para upper case
39             string username = request.Username.ToUpper();
40
41             // Encriptação da password
42             SHA512 sha512 = SHA512Managed.Create();
43             byte[] bytes = sha512.ComputeHash(Encoding.UTF8.GetBytes
                 (request.Password));
44             string password = Convert.ToBase64String(bytes);
45
46             // Verifica-se se o utilizador existe na base de dados e se a
                 password está correta
47             Models.User u = _context.User.FirstOrDefault(u => u.Username ==
                 request.Username && u.Password == password);
48
```

```
49              if (u == null)
50              {
51                  // Se não existir nenhuma entrada na base de dados com o
                      username e password igual, então o output tem de ser
                      autenticação inválida.
52                  output.Valid = false;
53                  output.SessionID = "";
54              }
55              else
56              {
57                  // Sempre que um utilizador se autentica, é gerado sempre um
                      novo ID de sessão.
58                  // É usado um ciclo para, no caso do ID de sessão gerado já
                      existir, poder gerar um novo
59                  bool success;
60                  do
61                  {
62                      success = true;
63                      try
64                      {
65                          u.GenerateSessionID();
66                          _context.SaveChanges();
67                      }
68                      // Exceção que é lançada sempre que é quebrado o
                          constraint UNIQUE do ID de sessão, no caso do ID de sessão
                           gerado já existir
69                      catch (DbUpdateException e) when (e.InnerException is
                          SqlException sqlEx && (sqlEx.Number == 2627 ||
                          sqlEx.Number == 2601))
70                      {
71                          success = false;
72                      }
73                  }
74                  while (success == false);
75
76                  // Se o login for feito com sucesso, é enviada uma confirmação
                       ao cliente, com o seu ID de sessão
77                  output.Valid = true;
78                  output.SessionID = u.SessionID;
79              }
80
81              return Task.FromResult(output);
82
83          }
84
85
86          /**
87           * Registo de novos utilizadores. Devem ser recebidos o username,
                email e password para criar o novo utilizador.
88           * Quando é guardado um novo utilizador na base de dados, o email e
                username são transformados em upper case, e
89           * a password é encriptada.
90           */
91          public override Task<UserRegistModel> Regist(UserRegistLookupModel
              request, ServerCallContext context)
92          {
```

```
93              UserRegistModel output = new UserRegistModel();
94
95              // Transformação dos carateres do email e username para upper case
96              string email = request.Email.ToUpper();
97              string username = request.Username.ToUpper();
98
99              // Verificações se um outro utilizador tem o mesmo email e/ou o
                  mesmo username
100             Models.User u1 = new Models.User(), u2 = new Models.User();
101             u1 = _context.User.FirstOrDefault(u => u.Username == username);
102             u2 = _context.User.FirstOrDefault(u => u.Email == email);
103
104             // Um utilizador tem o mesmo username
105             if (u1 != null)
106             {
107                 if (u2 == null)
108                     output.Valid = -3; // Um utilizador tem apenas o mesmo
                        username
109                 else
110                     output.Valid = -1; // Um utilizador tem o mesmo username e
                         email
111             }
112             // Um utilizador tem o mesmo email
113             else if (u2 != null)
114             {
115                 output.Valid = -2; // Um utilizador tem apenas o mesmo email
116             }
117             // Nenhum utilizador tem o mesmo username e/ou email
118             else
119             {
120                 // Encriptação da password
121                 SHA512 sha512 = SHA512Managed.Create();
122                 byte[] bytes = sha512.ComputeHash(Encoding.UTF8.GetBytes
                      (request.Password));
123                 string password = Convert.ToBase64String(bytes);
124
125                 Models.User u = new Models.User
126                 {
127                     Username = username,
128                     Email = email,
129                     Password = password
130                 };
131
132                 // Verifica e garante que a BD existe
133                 _context.Database.EnsureCreated();
134
135                 // Guarda-se o novo utilizador na base de dados
136                 _context.User.Add(u);
137                 _context.SaveChanges();
138
139                 // Envia-se uma mensagem uma mensagem de confirmação de
                      registo ao cliente
140                 output.Valid = 1;
141             }
142
143             return Task.FromResult(output);
```

```
144             }
145         }
146     }
```

```
1  // GameService.cs //
2
3  using Grpc.Core;
4  using GrpcServerRPS.Data;
5  using GrpcServerRPS.Models;
6  using Microsoft.EntityFrameworkCore;
7  using Microsoft.Extensions.Logging;
8  using System;
9  using System.Linq;
10 using System.Threading.Tasks;
11
12 namespace GrpcServerRPS.Services
13 {
14     public class GameService : Game.GameBase
15     {
16         private readonly ILogger<GameService> logger;
17         private readonly RPSGameDbContext _context;
18
19         public GameService(ILogger<GameService> logger, RPSGameDbContext
             context)
20         {
21             this.logger = logger;
22             _context = context;
23         }
24
25         public override Task<PlayModel> Play(PlayLookupModel request,
             ServerCallContext context)
26         {
27             PlayModel output = new PlayModel();
28
29             // Obtemos o utilizador da base de dados com o ID de sessão, para
                depois obtermos o seu ID, se não existir é retornado o código
30             // de erro para o cliente
31             Models.User user = _context.User.FirstOrDefault(u => u.SessionID
                == request.SessionId);
32             if (user == null)
33             {
34                 output.Result = -1;
35                 return Task.FromResult(output);
36             }
37
38             History h = _context.History.Include(i => i.User).FirstOrDefault(u
                => u.userId == user.Id);
39             if (h == null) // No caso do utilizador nunca ter jogado, criamos
                uma linha na tabela de estatísticas para este
40             {
41                 _context.Database.EnsureCreated();
42
43
44                 h = new History
45                 {
46                     userId = user.Id,
47                     User = user
48                 };
49
50                 _context.History.Add(h);
```

```
51                  _context.SaveChanges();
52
53              }
54
55              // É gerada a jogada do servidor
56              Random rnd = new Random();
57              int serverPlay = rnd.Next(1, 4);
58
59
60              switch (request.Play)
61              {
62                  case 1: // Pedra
63
64                      switch (serverPlay)
65                      {
66                          case 1: // Pedra
67                              output.ServerPlay = "Rock";
68                              output.Result = 0; // Empate
69                              h.draw++;
70                              break;
71                          case 2: // Papel
72                              output.ServerPlay = "Paper";
73                              output.Result = 2; // Servidor venceu
74                              h.lost++;
75                              break;
76                          case 3: // Tesoura
77                              output.ServerPlay = "Scissors";
78                              output.Result = 1; // Utilizador venceu
79                              h.win++;
80                              break;
81                          default:
82                              break;
83                      }
84
85
86                      break;
87                  case 2: // Papel
88                      switch (serverPlay)
89                      {
90                          case 1: // Pedra
91                              output.ServerPlay = "Rock";
92                              output.Result = 1; // Utilizador venceu
93                              h.win++;
94                              break;
95                          case 2: // Papel
96                              output.ServerPlay = "Paper";
97                              output.Result = 0; // Empate
98                              h.draw++;
99                              break;
100                         case 3: // Tesoura
101                             output.ServerPlay = "Scissors";
102                             output.Result = 2; // Servidor venceu
103                             h.lost++;
104                             break;
105                         default:
106                             break;
```

```
107                     }
108                     break;
109                 case 3: // Tesoura
110                     switch (serverPlay)
111                     {
112                         case 1: // Pedra
113                             output.ServerPlay = "Rock";
114                             output.Result = 2; // Servidor venceu
115                             h.lost++;
116                             break;
117                         case 2: // Papel
118                             output.ServerPlay = "Paper";
119                             output.Result = 1; // Utilizador venceu
120                             h.win++;
121                             break;
122                         case 3: // Tesoura
123                             output.ServerPlay = "Scissors";
124                             output.Result = 0; // Empate
125                             h.draw++;
126                             break;
127                         default:
128                             break;
129                     }
130                     break;
131 
132                 default:
133                     break;
134             }
135 
136             h.Games++;
137 
138             _context.SaveChanges();
139 
140 
141             return Task.FromResult(output);
142         }
143 
144         public override Task<StatsModel> Stats(StatsLookupModel request,
              ServerCallContext context)
145         {
146             StatsModel output = new StatsModel();
147 
148             // Obtemos o utilizador da base de dados com o ID de sessão, para
                  depois obtermos o seu ID, se não existir é retornado o código
149             // de erro para o cliente
150             Models.User user = _context.User.FirstOrDefault(u => u.SessionID
                  == request.SessionId);
151             if (user == null)
152             {
153                 output.GamesPlayed = -1;
154                 return Task.FromResult(output);
155             }
156 
157             History h = _context.History.FirstOrDefault(u => u.userId ==
                  user.Id);
158             if (h == null) // No caso do utilizador nunca ter jogado, cria-se
```

```
                uma entrada na base de dados para este jogador.
159             {
160                 _context.Database.EnsureCreated();
161
162                 h = new History
163                 {
164                     Games = 0,
165                     win = 0,
166                     lost = 0,
167                     draw = 0,
168                     userId = user.Id
169                 };
170
171                 _context.History.Add(h);
172                 _context.SaveChanges();
173
174             }
175
176             output.Draws = h.draw;
177             output.GamesPlayed = h.Games;
178             output.Losts = h.lost;
179             output.Wins = h.win;
180
181             return Task.FromResult(output);
182         }
183     }
184 }
185
```