UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Trabalho Prático nº2 – Sistema de Reserva de Bilhetes de Teatro

Engenharia Informática

Sistemas Distribuídos

Ivan Miguel Serrano Pires

Hugo Paredes

Autores

Luís Pimenta al70827

Pedro Guerra al70596

Tiago Morais al71395

Tomás Silva al70680

Introdução

No âmbito da disciplina de Sistemas Distribuídos, foi-nos proposto a construção de um sistema de reserva de bilhetes de teatro, que visa a retratar uma aplicação de bilheteira virtual de teatros. Com o tema já definido, tivemos que usar a framework gRPC de forma a criar uma comunicação cliente/servidor. O gRPC (Google Remote Procedure Call) é um modelo de comunicação, que permite que o cliente e o servidor comuniquem de modo transparente. Por último, a aplicação foi construída em C#.

Servidor e Serviços RPC

É no servidor que está presente a conexão à base de dados. A função principal do servidor é guardar toda a informação, por exemplo guardar uma compra, um user e ou uma sessão. Também tem a implementação dos serviços. Os serviços são classes, com um conjunto de métodos.

Nesta aplicação, os serviços são:

- O Login, é responsável por verificar se o user tem permissão para aceder à página que está a tentar aceder;
- O Register, que contém as funções responsáveis por registar os diferentes tipos de clientes;
- O UserService, que retorna a informação do user conectado bem como a lista dos utilizadores, e tem a capacidade de adicionar dinheiro à conta virtual;
- O LocalizationService, o serviço contem funções capazes de retorna as localizações consoante a localização do user, bem como adicionar localizações novas;
- O TheaterService, com funções responsáveis por retornar teatros, mudar o estado de um teatro, retornar a informação de um teatro em especifico a partir do ID, atualizar os dados de um teatro e adicionar um teatro;
- O ShowService, com funções responsáveis por retornar espetáculos, mudar o estado de um espetáculo, retornar a informação de um espetáculo em especifico a partir do ID, atualizar os dados de um espetáculo e adicionar um espetáculo;
- O SessionService, com funções responsáveis por retornar sessões, mudar o estado de uma sessão, retornar a informação de uma sessão

em especifico a partir do ID, atualizar os dados de uma sessão e adicionar uma sessão;

- O CartService, com a capacidade de ver que há lugares disponíveis, bem como reservar um lugar e cancelar a reserva do lugar;
- O CompraService, com funções responsáveis por comprar as sessões previamente adicionadas ao carrinho de compras, retornar o histórico de compras de um user, retornar a informação de uma compra, cancelar a compra caso seja possível e retornar todas as compras já feitas.

Exemplos de código:Anexos 1-4 & 7

Cliente para Utilizador

O utilizador antes de realizar qualquer movimento, tem que fazer login na aplicação, caso já esteja registado. O utilizador não está constantemente ligado ao servidor, só quando efetua algum movimento é que é criado um canal que é responsável pela conexão cliente/servidor. Para haver uma comunicação, o utilizador a partir de uma interface gráfica é capaz:

- Visualizar a lista de espetáculos já assistidos;
- Visualizar bilhetes comprados;
- Adicionar fundos à conta virtual;
- Comprar bilhetes para sessões;
- Pesquisar teatros, espetáculo e sessões;
- Anular compra do bilhete;
- Guardar espetáculos já assistidos;

Cliente para Administração

O administrador antes de realizar qualquer movimento, tem que fazer login na aplicação, caso já esteja registado. O administrador não está constantemente ligado ao servidor, só quando efetua algum movimento é que é criado um canal que é responsável pela conexão cliente/servidor. Para haver uma comunicação, o administrador a partir de uma interface gráfica é capaz:

- Visualizar a lista de teatros;
- Visualizar a lista de espetáculos;

- Visualizar a lista de sessões;
- Visualizar a lista de bilhetes comprados;
- Visualizar o histórico de pedidos de logging;
- Adicionar managers e users.

Exemplos de código: Anexos 5-6

Cliente para Gestão

O administrador antes de realizar qualquer movimento, tem que fazer login na aplicação, caso já esteja registado. O gestor não está constantemente ligado ao servidor, só quando efetua algum movimento é que é criado um canal que é responsável pela conexão cliente/servidor. Para haver uma comunicação, o gestor a partir de uma interface gráfica é capaz:

- Adicionar teatros;
- Editar teatros;
- Adicionar espetáculos;
- Editar espetáculos;
- Remover espetáculos.

Anexos

```
string pass = Password.Password;

if (!string.IsNullOrEmpty(email) && !string.IsNullOrEmpty(pass))
{
    var channel = new Channel(App.IPAdd, ChannelCredentials.Insecure);
    var client = new LoginClient(channel, new gRPCProto.Login.LoginClient(channel));
    UserLogin userLogin = new()
```

Fig 1 – Conexão

```
// Send and receive some notes.
UserConnected userConnected = client.CheckLogin(userLogin).Result;

if (userConnected.Exists())
{
    if (userConnected.Id == -2)
    {
        MessageBox.Show("Erro ao Iniciar sessão. Utilizador ou Password Incorretos.", "TeatrosLand", MessageBoxButton.OK, MessageBoxImage.Warning);
    }
    else
    {
        MainWindow mainWindow = new MainWindow(userConnected);
        mainWindow.Show();
        Close();
    }
}
else
{
    MessageBox.Show("Erro ao Iniciar sessão. Por favor contactar a entidade", "TeatrosLand", MessageBoxButton.OK, MessageBoxImage.Error);
}

channel.ShutdownAsync().Wait();
```

Fig 2 - UserConnected

```
service LocalizationService {

    /*
    Returns all localizations
    */
    rpc GetLocalizations(UserConnected) returns (stream LocalizationInfo) {}

    //Add new localization
    rpc AddLocalization(LocalizationInfo) returns (Confirmation) {}
}

service TheaterService {

    /*
    Returns all theaters
    */
    rpc GetTheaters(UserConnected) returns (stream TheaterInfo) {}

    // Change a state of a theter
    rpc ChangeState(TheaterInfoState) returns (Confirmation) {}

    //Returns info about the thetaer ID send
    rpc GetTheater(TheaterInfo) returns (TheaterInfo) {}

    // Update the data of a theather
    rpc UpdateTheater(TheaterInfo) returns (Confirmation) {}

    //Add new Theater
    rpc AddTheater(TheaterInfo) returns (Confirmation) {}
}
```

Fig 3 - Serviços

```
172
173 message UserLogin {
174     string email = 1;
175     string password = 2;
176     string type = 3;
177 }
178
179 message UserConnected {
180     int32 id = 1;
181     string type = 2;
182 }
183
184 message UserRegister {
185     string name = 1;
186     string email = 2;
187     string password = 3;
188     int32 idLocalization = 4;
189 }
```

Fig 4 – Mensagens

```
                await responseStream.WriteAsync(p);
            }
        }
        catch (ArgumentNullException ex)
        {
            LogServiceImpl logServiceImpl = new(DBcontext);

            await logServiceImpl.LogError(new LogInfo()
            {
                Msg = $"'ArgumentNullException': [{DateTime.Now}] - Error - Erro ao receber as sessões. Argumento nulo.\nCode Msg: {ex.Message}",
                LevelLog = 3
            }, context);
        }
        catch (Exception ex)
        {
            LogServiceImpl logServiceImpl = new(DBcontext);

            await logServiceImpl.LogError(new LogInfo()
            {
                Msg = $"'Exception': [{DateTime.Now}] - Error - Erro ao receber as sessões.\nCode Msg: {ex.Message}",
                LevelLog = 3
            }, context);
        }
    }
```

Fig 5 – Logs Error

```
    refCompra.Reference = "";
    LogServiceImpl logServiceImpl = new(DBcontext);

    await logServiceImpl.LogWarning(new LogInfo()
    {
        Msg = $"'NotFoundDB': [{DateTime.Now}] - Error - Não foi encontrado o utlizador na BD com o id {sessions.ElementAt(0).UserId}.",
        LevelLog = 2
    }, context);
}
```

Fig 6 – Logs Warning

```
static void Main(string[] args)
{
    try
    {
        const int Port = 45300;

        var context = new ServerContext();

        Grpc.Core.Server server = new Grpc.Core.Server
        {
            Services = {
            Login.BindService(new LoginImpl(context)),
            UserService.BindService(new UserServiceImpl(context)),
            Register.BindService(new RegisterImpl(context)),
            LocalizationService.BindService(new LocalizationServiceImpl(context)),
            TheaterService.BindService(new TheaterServiceImpl(context)),
            ShowService.BindService(new ShowServiceImpl(context)),
            SessionService.BindService(new SessionServiceImpl(context)),
            CartService.BindService(new CartServiceImpl(context)),
            CompraService.BindService(new CompraServiceImpl(context)),
        },
            Ports = { new ServerPort("10.144.10.2", Port, ServerCredentials.Insecure) }
        };
        server.Start();

        Console.WriteLine("RouteGuide server listening on port " + Port);
        Console.WriteLine("Press any key to stop the server...");
        Console.ReadKey();

        server.ShutdownAsync().Wait();
```

Fig 7 – Configurar o servidor