

# Pavilhão Multiusos



## Relatório do Projeto de Programação Orientada a Objetos

Realizado por:

Diogo Araújo - 21086

Manuel Fernandes - 23502

Daniel Pereira - 23507

## Conteúdo

Índice de imagens.....	2
1. Introdução .....	4
2.Código .....	4
2.1 Bibliotecas .....	4
2.2 Classes .....	5
2.2.1 Funcionário.....	5
2.2.2 Pessoa.....	8
2.2.3 Reserva .....	9
2.2.4 Utilizador .....	12
2.3 Enums.....	15
2.4 Exceptions .....	16
2.5 Forms.....	17
2.5.1 Criar Utilizador .....	17
2.5.2 Detalhes Espaço .....	18
2.5.3 Login Page .....	20
2.5.4 Menu .....	21
2.5.5 VerPerfil.....	22
2.6 Interfaces.....	26
3. Aplicação funcional .....	27
4. Conclusão .....	32
5. Bibliografia .....	32

## Índice de imagens

Figura 1 - Variáveis, Propriedades, Métodos .....	6
Figura 2 – Filtrar, Guardar e Carregar Lista .....	7
Figura 3 – Classe Pessoa, Variáveis e Propriedades .....	8
Figura 4 – Construtores e Variáveis .....	9
Figura 5 – Propriedades Reserva.....	10
Figura 6 – Dados da Lista Reserva .....	10
Figura 7 – Filtragem da lista em função do espaço.....	11
Figura 8 - Método que retorna a primeira reserva onde o numero da reserva é igual ao da string passada por parâmetro.....	11
Figura 9 – Método estado de reserva .....	11
Figura 10 – Guarda a lista de reservas em ficheiro .....	12
Figura 11 – Método que carrega o ficheiro lista na aplicação .....	12
Figura 12.....	12
Figura 13.....	13
Figura 14.....	14

Figura 15.....	14
Figura 16 - Enum espaços.....	15
Figura 17 - Exceção de sobreposição de reservas.....	16
Figura 18 - Exceção duplicação na criação de um utilizador.....	16
Figura 19 - Exceção da falta de uma reserva.....	16
Figura 20 - Exceção da ausência do utilizador de acordo com os dados introduzidos no login .	17
Figura 21.....	18
Figura 22 - Para evitar a sobreposição de labels de design, criamos este if.....	18
Figura 23 - Método para efetuar uma reserva, com as devidas exceptions.....	19
Figura 24 - Regras dos horários da reserva .....	19
Figura 25 - Página de login onde verifica se os dados introduzidos pertencem a um utilizador comum ou a um funcionário .....	20
Figura 26 - Fecha a página de login e abre a página para criar um utilizador.....	20
Figura 27.....	21
Figura 28 - Mostra todos os espaços possíveis para reservar.....	21
Figura 29 - Adiciona os espaços de reserva na ListView .....	22
Figura 30 - Método para o utilizador ter acesso ao seu perfil .....	22
Figura 31 - Construtor do cálculo do IMC e massa magra e visibilidades das labels .....	22
Figura 32 – ListView em Utilizador .....	23
Figura 33 – ListView dos funcionários.....	23
Figura 34 - Distinção de login .....	24
Figura 35 – Preenchimento de lista consoante o login do utilizador.....	24
Figura 36 - Preenchimento de dados físicos .....	25
Figura 37 - Remover Reservas.....	25
Figura 38 - Alteração de dados físicos.....	26
Figura 39 - Interface Utilizador.....	26
Figura 40 - Interface Funcionário .....	27
Figura 41 - No menu de login introduzimos o numero de processo do utilizador ou do funcionário e a devida password .....	27
Figura 42 - Aqui podemos criar a conta do utilizador preenchendo os campos requeridos .....	28
Figura 43 - No menu principal, o utilizador seleciona o espaço desejado para reservar .....	28
Figura 44 - Dado que o utilizador escolhe o espaço “Campo de Futebol de 7”, este pode ver as horas as que já estão ocupadas e efetua a sua reserva de acordo com as horas vagas.....	29
Figura 45 - Ao clicarmos no Ícone de utilizador junto ao seu nome, acedamos à página do utilizador. Aqui este pode consultar as suas reservas, as suas informações pessoais e o seu físico. Na aba “Ver Reservas”, este pode verificar as reservas agendadas e cancelar .....	29
Figura 46 - Na aba “Mudar Informações Pessoais”, o utilizador pode rever e alterar as informações que deseje .....	30
Figura 47 - No sector “Consultar Físico”, o utilizador pode consultar e definir o seu progresso físico. ....	30
Figura 48 - Na visão de um funcionário, este pode consultar a agenda de um ou mais espaços. ....	31
Figura 49 - Suponhamos que a funcionária Lurdes pretende cancelar a reserva efetuada pelo Manuel há momentos atras como demonstrado. A Lurdes, pode escolher um espaço e ver todas as reservas la efetuadas, e assim, cancelar as que pretenda.....	31

## 1. Introdução

No âmbito da unidade curricular Programação Orientada a Objetos, foi-nos atribuído um projeto à escolha dentro do tema “Smart Campus”. Sendo este desenvolvido em linguagem C#.

A criação de um complexo desportivo no IPCA, foi o tema escolhido para o desenvolvimento deste projeto. Este complexo desportivo vem a dispor de um ginásio, aluguer de campos de futebol de 7 ou 11 jogadores ou o aluguer do pavilhão por completo para fins académicos ou presidenciais. Neste projeto, desenvolvemos uma aplicação móvel do sistema completo do complexo. Esta aplicação inclui a criação do utilizador do complexo; o rastreio do estado físico do individuo como o peso, massa corporal e entre outros; a possibilidade de rever as suas reservas próximas e a eliminação das mesmas; e a alteração de informações pessoais como email e password. Dentro do menu principal, podemos efetuar as devidas reservas e ver também, que horários estão já ocupados ou reservados por outros utilizadores.

Para uma melhor gestão e facilidade deste sistema, implementamos na aplicação, os funcionários do ginásio encarregues da gerência do complexo. Estes, podem e devem retificar as reservas dos utilizadores, tendo a possibilidade de cancelar reservas de utilizadores por razões superiores.

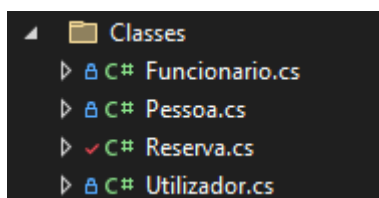
## 2. Código

### 2.1 Bibliotecas

Estas são as bibliotecas utilizadas ao longo do desenvolvimento do nosso projeto

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using Newtonsoft;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;
using Polidesportivo_Ipca.Enums;
using System.Xml.Linq;
using System.Windows.Forms;
using System.Security.Policy;
using System.Security.Permissions;
```

## 2.2 Classes



### 2.2.1 Funcionário

Na figura 1 é apresentado o código da classe Funcionário que herda de Pessoa onde se encontram todas as variáveis, propriedades, métodos e listas associados à mesma.

```
public class Funcionario : Pessoa, IFuncionario
{
    //Variaveis

    /// <summary>
    /// Criação da lista de funcionários
    /// </summary>
    public static List<Funcionario> listaFuncionario = new List<Funcionario>();

    //Propriedades

    /// <summary>
    /// Nr de Processo do funcionário
    /// </summary>
    public string NrProcessoEscola { get; set; }

    //Metodos

    /// <summary>
    /// Metodo para adicionar dados localmente na lista de funcionarios
    /// </summary>
    public static void DadosLista()
    {
        listaFuncionario.Add(new Funcionario()
        {
            Nome = "Lurdes",
            Email = "lurdesdascouves@gmail.com",
            Password = "Lurdes1",
            DataNascimento = new DateTime(2003, 05, 31),
            NrProcessoEscola = "1",
            NumeroTlm = "960424821",
        });
    }
}
```

Figura 1 - Variáveis, Propriedades, Métodos

No print seguinte é apresentado o método que filtra o primeiro funcionário que valida a condição, ou seja, que acerta no número de processo e na respetiva password se a mesma constatar no DadosLista (figura 1). Para além disso tem também os métodos de guardar e carregar a lista de funcionários em ficheiro .txt.

```
public Funcionario FuncionarioLogin(string nrprocesso, string pass)
{
    Funcionario funcionario = listaFuncionario.FirstOrDefault(r => r.NrProcessoEscola.Equals(nrprocesso) && r.Password.Equals(pass));
    return funcionario;
}

/// <summary>
/// Metodo para guardar a lista de funcionários em ficheiro .txt
/// </summary>
public static void GuardarListaFuncionario()
{
    string save = JsonConvert.SerializeObject(listaFuncionario);
    File.WriteAllText(Environment.CurrentDirectory + @"\listafuncionarios.txt", save);
}

/// <summary>
/// Metodo para carregar o ficheiro listafuncionarios.txt para a lista de funcionários
/// </summary>
public static void CarregarListaFuncionario()
{
    if (File.Exists(Environment.CurrentDirectory + @"\listafuncionarios.txt"))
        listaFuncionario = JsonConvert.DeserializeObject<List<Funcionario>>(File.ReadAllText(Environment.CurrentDirectory + @"\listafuncionarios.txt"));
    else
        File.WriteAllText(Environment.CurrentDirectory + @"\listafuncionarios.txt", "[]");
}
}
```

Figura 2 – Filtrar, Guardar e Carregar Lista

### 2.2.2 Pessoa

Na imagem seguinte está representada uma nova classe designada Pessoa onde são declaradas as variáveis para o funcionamento do login e as suas respetivas propriedades que contem as informações da classe.

```
public class Pessoa
{
    //Variaveis
    /// <summary>
    /// Declaração da variavel para verificar se o utilizador está logado
    /// </summary>
    public static Utilizador userLogado;
    /// <summary>
    /// Declaração da variavel para verificar se o funcionario está logado
    /// </summary>
    public static Funcionario funcionarioLogado;

    //Propriedades

    /// <summary>
    /// Nome da Pessoa
    /// </summary>
    public string Nome { get; set; }
    /// <summary>
    /// Password da Pessoa
    /// </summary>
    public string Password { get; set; }
    /// <summary>
    /// Email da Pessoa
    /// </summary>
    public string Email { get; set; }
    /// <summary>
    /// Data de Nascimento da Pessoa
    /// </summary>
    public DateTime DataNascimento { get; set; }
    /// <summary>
    /// Numero de telemovel da pessoa
    /// </summary>
    public string NumeroTlm { get; set; }
}
```

Figura 3 – Classe Pessoa, Variáveis e Propriedades



### 2.2.3 Reserva

Nova classe Reserva composta por um construtor, as suas variáveis, propriedades e métodos. O construtor foi criado para colocar as reservas ativas e verificar se estas foram ou não utilizadas, as variáveis foram criadas para verificar se o utilizador fez o login e para a criação da lista reserva. Nas propriedades tem as informações que caracterizam a class. Nos métodos foi criada uma lista local na qual são armazenados os dados da reserva. (figuras 4-11)

```
public class Reserva
{
    //Construtor
    /// <summary>
    /// Inicialização do construtor
    /// </summary>
    public Reserva()
    {
        //Coloca todas as reservas como ativas
        this.Estado = Estado.Ativa;
        //Verifica o estado das reservas em relação à data
        EstadoReserva(lista);
    }

    //Variáveis

    /// <summary>
    /// Variável para verificar o utilizador logado
    /// </summary>
    public static Reserva userLogado;
    /// <summary>
    /// Criação da lista reserva
    /// </summary>
    public static List<Reserva> lista = new List<Reserva> { };
}
```

Figura 4 – Construtores e Variáveis

```
//Propriedades

/// <summary>
/// Numero de Processo da reserva
/// </summary>
public string NrProcesso { get; set; }
/// <summary>
/// Data de Reserva
/// </summary>
public DateTime Data { get; set; }
/// <summary>
/// Hora do Inicio da Reserva
/// </summary>
public DateTime InicioReserva { get; set; }
/// <summary>
/// Hora do Fim da Reserva
/// </summary>
public DateTime FimReserva { get; set; }
/// <summary>
/// Espaço escolhido para a Reserva
/// </summary>
public Espacos Espaco { get; set; }
/// <summary>
/// Estado da Reserva
/// </summary>
public Estado Estado { get; set; }
```

Figura 5 – Propriedades Reserva

```
public static void DadosLista()
{
    lista.Add(new Reserva()
    {
        NrProcesso = "23507",
        Data = new DateTime(2022, 12, 19),
        InicioReserva = new DateTime(2022, 01, 01, 13, 30, 20),
        FimReserva = new DateTime(2022, 01, 01, 15, 30, 20),
        Espaco = Espacos.CampoFutebol7,
    });
    lista.Add(new Reserva()
    {
        NrProcesso = "23507",
        Data = new DateTime(2022, 12, 21),
        InicioReserva = new DateTime(2022, 01, 01, 14, 30, 20),
        FimReserva = new DateTime(2022, 01, 01, 16, 00, 20),
        Espaco = Espacos.CampoFutebol7,
    });
    lista.Add(new Reserva()
    {
        NrProcesso = "23507",
        Data = new DateTime(2022, 12, 19),
        InicioReserva = DateTime.Now,
        FimReserva = DateTime.Now,
        Espaco = Espacos.CampoFutebol11,
    });
    lista.Add(new Reserva()
    {
        NrProcesso = "23507",
        Data = new DateTime(2022, 12, 19),
        InicioReserva = DateTime.Now,
        FimReserva = DateTime.Now,
        Espaco = Espacos.Ginasio,
    });
}
```

Figura 6 – Dados da Lista Reserva

```
public List<Reserva> ListaReservas()
{
    List<Reserva> reservas = lista.Where(r => r.NrProcesso == userLogado.NrProcesso).ToList();
    return reservas;
}

/// <summary>
/// Metodo que filtra a lista de reservas em função do espaço dado por parametro
/// </summary>
/// <param name="espaco">Espaço escolhido em string</param>
/// <returns>Lista de reservas</returns>
public List<Reserva> Lista(string espaco)
{
    Espacos espaco2;
    Enum.TryParse(espaco, out espaco2);
    List<Reserva> reservas = lista.Where(r => r.Espaco.Equals(espaco2)).ToList();
    return reservas;
}
```

Figura 7 – Filtragem da lista em função do espaço

```
public Reserva ReservaLogin(string nrprocesso)
{
    Reserva reservado = lista.FirstOrDefault(r => r.NrProcesso.Equals(nrprocesso));
    return reservado;
}
```

Figura 8 - Método que retorna a primeira reserva onde o numero da reserva é igual ao da string passada por parâmetro

```
public void EstadoReserva(List<Reserva> lista)
{
    foreach (Reserva reserva in lista)
    {
        if (DateTime.Compare(reserva.Data.AddMinutes(29), DateTime.Now) < 0)
        {
            reserva.Estado = Estado.Utilizada;
        }
        else if (DateTime.Compare(reserva.Data, DateTime.Now) == 0)
        {
            reserva.Estado = Estado.Ativa;
        }
        else
        {
            reserva.Estado = Estado.Ativa;
        }
    }
}
```

Figura 9 – Método estado de reserva

```
public static void GuardarListaReserva()
{
    string save = JsonConvert.SerializeObject(lista);
    File.WriteAllText(Environment.CurrentDirectory + @"\listareservas.txt", save);
}
```

Figura 10 – Guarda a lista de reservas em ficheiro

```
public static void CarregarListaReserva()
{
    if (File.Exists(Environment.CurrentDirectory + @"\listareservas.txt"))
        lista = JsonConvert.DeserializeObject<List<Reserva>>(File.ReadAllText(Environment.CurrentDirectory + @"\listareservas.txt"));
    else
        File.WriteAllText(Environment.CurrentDirectory + @"\listareservas.txt", "[]");
}
```

Figura 11 – Método que carrega o ficheiro lista na aplicação

## 2.2.4 Utilizador

Nova classe utilizador que herda de pessoa e da interface utilizador (Utilizador) composta por variáveis, propriedades e métodos. Nas variáveis foi criado uma função que cria a lista de utilizadores e um delegado que calcula o IMC (Índice de massa corporal) e massa magra. Nas propriedades estão apresentadas as funções que recebem os dados dos utilizadores.

```
public class Utilizador : Pessoa, IUtilizador
{
    //Variavel
    /// <summary>
    /// Criação da lista de Utilizadores
    /// </summary>
    public static List<Utilizador> listaUtilizador = new List<Utilizador>();
    /// <summary>
    /// Delegado para calcular IMC e Massa Magra
    /// </summary>
    /// <param name="peso"></param>
    /// <param name="altura"></param>
    /// <returns></returns>
    public delegate string Calculo(string peso, string altura);

    //Propriedades
    /// <summary>
    /// Nr de Processo do Aluno
    /// </summary>
    public string NrProcessoEscola { get; set; }
    /// <summary>
    /// Peso do utilizador
    /// </summary>
    public string Peso { get; set; }
    /// <summary>
    /// Massa Gorda do utilizador
    /// </summary>
    public string MassaMagra { get; set; }
    /// <summary>
    /// Altura do utilizador
    /// </summary>
    public string Altura { get; set; }
    /// <summary>
    /// Perímetro dos ombros do utilizador
    /// </summary>
    public string PerimetroOmbros { get; set; }
```

Figura 12

```
public static void DadosLista()
{
    listaUtilizador.Add(new Utilizador()
    {
        Nome = "Daniel",
        Email = "danieldascouves@gmail.com",
        Password = "Daniel1",
        DataNascimento = new DateTime(2003, 05, 31),
        NrProcessoEscola = "23507",
        NumeroTlm = "960424821",
        Peso = "87",
        MassaMagra = "",
        Altura="178",
        Ombros = "60",
        Braco = "60",
        Antebraco = "30",
        Envergadura = "200",
        Gluteos = "50",
        Coxa="40",
        IMC = "",
    });
    listaUtilizador.Add(new Utilizador()
    {
        Nome = "Manuel",
        Email = "manueldascouves@gmail.com",
        Password = "Manuel1",
        DataNascimento = new DateTime(2003, 05, 31),
        NrProcessoEscola = "23502",
        NumeroTlm = "960424821",
        Peso = "87",
        MassaMagra = "",
        Altura = "178",
        Ombros = "60",
        Braco = "60",
        Antebraco = "30",
        Envergadura = "200",
        Gluteos = "50",
        Coxa = "40",
        TMC = ""
    });
}
```

Figura 13

```
public static string CalculoIMC(string altura,string peso)
{
    float n1 = float.Parse(altura);
    float n2 = float.Parse(peso);
    float imc = (n2 / (n1/100 * n1/100));
    return imc.ToString();
}
```

Figura 14

```
public static string CalculoMassaMagra(string altura,string peso)
{
    float n1 = float.Parse(altura);
    float n2 = float.Parse(peso);
    float mg =n1-n1 + n2 - (n2 * 2/10);
    return mg.ToString();
}
```

Figura 15

```
public Utilizador UtilizadorLogin(string nrprocesso, string pass)
{
    Utilizador utilizador = listaUtilizador.FirstOrDefault(r => r.NrProcessoEscola.Equals(nrprocesso) && r.Password.Equals(pass));
    return utilizador;
}
```

Figura

## 2.3 Enums

```

namespace Polidesportivo_Ipca.Enums
{
    /// <summary>
    /// Enumeração de espaços disponíveis
    /// </summary>
    13 references
    public enum Espacos
    {
        CampoFutebol7, CampoFutebol11, Ginasio, Pavilhao
    }

    /// <summary>
    /// Enumeração dos estados possíveis
    /// </summary>
    5 references
    public enum Estado
    {
        Ativa, Utilizada
    }
}

```

Figura 16 - Enum espaços

## 2.4 Exceptions

```
namespace Polidesportivo_Ipca
{
    2 references
    public class AlreadyExistingReserve : Exception
    {
        /// <summary>
        /// Mensagem da Exceção "AlreadyExistingReserve"
        /// </summary>
        0 references
        public override string Message => "Já existe uma reserva nesse horário";
    }
}
```

Figura 17 - Exceção de sobreposição de reservas

```
namespace Polidesportivo_Ipca
{
    1 reference
    public class AlreadyExistingUser : Exception
    {
        /// <summary>
        /// Mensagem da Exceção "AlreadyExistingUser"
        /// </summary>
        0 references
        public override string Message => "Esse Utilizador já existe";
    }
}
```

Figura 18 - Exceção duplicação na criação de um utilizador

```
namespace Polidesportivo_Ipca
{
    2 references
    public class NotExistingReserveException : Exception
    {
        /// <summary>
        /// Mensagem da Exceção "NotExistingReserve"
        /// </summary>
        public override string Message => "Esse Utilizador não tem Reservas";
    }
}
```

Figura 19 - Exceção da falta de uma reserva

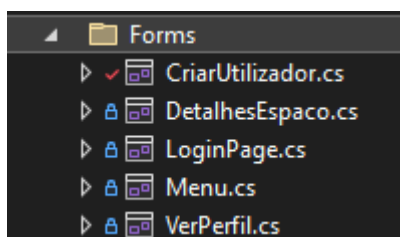


```
namespace Polidesportivo.Exceptions
{
    2 references
    public class NotExistingUserException : Exception
    {
        /// <summary>
        /// Mensagem da Exceção "NotExistingUserException"
        /// </summary>
        0 references
        public override string Message => "Não Existe Esse Utilizador";
    }
}
```

Figura 20 - Exceção da ausência do utilizador de acordo com os dados introduzidos no login

## 2.5 Forms

O design da nossa aplicação foi criado através do *WinForms*. Dentro da pasta Forms, temos 5 ficheiros que representam todas as janelas do programa que serão utilizadas. Ou seja, toda o código que é relacionado com o design e algumas funcionalidades, foi desenvolvido a partir destes ficheiros.



### 2.5.1 Criar Utilizador

Aqui estará o código da janela de criação de um novo utilizador, com as devidas exceções

```
1 reference
private void BtnCriarUtilizador_Click(object sender, EventArgs e)
{
    Utilizador utilizador = new Utilizador();
    string[] users = Utilizador.ListaUtilizador().ToArray();
    utilizador.Nome = TxtNome.Text;
    utilizador.NrProcessoEscola = TxtNrProcesso.Text;
    utilizador.Email = TxtEmail.Text;
    utilizador.Password = TxtPassword.Text;
    utilizador.NumeroTlm = TxtNrTelemovel.Text;
    bool existeuser = users.Contains(utilizador.NrProcessoEscola);
    if (existeuser)
    {
        throw new AlreadyExistingUser();
    }
    else
    {
        Utilizador.listaUtilizador.Add(utilizador);
        MessageBox.Show("Utilizador Criado com Sucesso");
    }
    Utilizador.GuardarListaUtilizador();
}
```

Figura 21

## 2.5.2 Detalhes Espaço

Todo o código relacionado com o sistema de reservas estará neste ficheiro.

```
RegrasHorario();
if (Pessoa.userLogado == null)//Verifica se é um utilizador logado
{
    //Se não for, o botão reservar fica invisível
    BtnReservar.Visible = false;
}
```

Figura 22 - Para evitar a sobreposição de labels de design, criamos este if

```

1 reference
private void BtnReservar_Click(object sender, EventArgs e)
{
    Reserva reserva = AdicionarReserva(); //Adiciona nova reserva
    bool existereserva = Reserva.lista.Any(r => r.InicioReserva == reserva.InicioReserva); //Verifica se já existe uma reserva com esses dados
    if (existereserva == false)
    {
        Reserva.lista.Add(reserva); //Adiciona na lista de reservas, a reserva criada
        Reserva.GuardarListaReserva(); //Guarda a lista de reservas em ficheiro
        LstHorasOcupadas.Refresh(); //Atualiza a listView
    }
    else throw new AlreadyExistingReserve(); //Atira uma exceção a avisar que já existe uma reserva
}
/// <summary>
/// Método que cria nova reserva com os dados do form
/// </summary>
/// <returns></returns>
1 reference
private Reserva AdicionarReserva()
{
    Reserva reserva = new Reserva();
    reserva.NrProcesso = Utilizador.userLogado.NrProcessoEscola;
    reserva.Data = DateTimeData.Value;
    reserva.InicioReserva = DateTimeInicio.Value;
    reserva.FimReserva = DateTimeFim.Value;
    reserva.Espaco = espacos;
    return reserva;
}

```

Figura 23 - Método para efetuar uma reserva, com as devidas exceptions

```

private void DateTimeInicio_ValueChanged(object sender, EventArgs e)
{
    DateTimeFim.MinDate = DateTimeInicio.Value.AddMinutes(30); //Hora minima de Fim de Reserva é a Inicial mais 30 minutos
}

/// <summary>
/// Metodo para garantir que não é possível marcar reservas em horas anteriores às atuais e que cada reserva tem no minimo 30 minutos
/// </summary>
1 reference
private void RegrasHorario()
{
    DateTimeFim.MinDate = DateTimeInicio.Value; //Hora minima de Fim de Reserva é igual à hora de Inicio da Reserva
    DateTimeInicio.MinDate = DateTime.Now; //Hora minima de inicio é igual às horas do utilizador
    DateTimeInicio.MinDate = DateTime.Now.AddMinutes(30); //Hora minima de inicio de reserva é igual às horas do utilizador mais 30 minutos
    DateTimeData.MinDate = DateTime.Now; //Data minima possível de reserva igual à data que o utilizador está a efetuar a reserva
}

```

Figura 24 - Regras dos horários da reserva

### 2.5.3 Login Page

Código do menu inicial de Login com uma exceção da ausência de um utilizador

```
private void BtnEntrar_Click(object sender, EventArgs e)
{
    Utilizador utilizador = new Utilizador();
    Reserva reserva = new Reserva();
    Funcionario funcionario = new Funcionario();
    Utilizador User = utilizador.UtilizadorLogin(TxtNrProcesso.Text, TxtPass.Text);
    Funcionario Func = funcionario.FuncionarioLogin(TxtNrProcesso.Text, TxtPass.Text);
    Reserva.userLogado = reserva.ReservaLogin(TxtNrProcesso.Text);
    if (User != null) //confirma se existe algum Utilizador
    {
        //Se existir
        Utilizador.userLogado = User; //Utilizador logado atualizado
        Pessoa.funcionarioLogado = null; //Funcionario logado é nulo
    }
    else
    {
        //Se não existir
        Funcionario.funcionarioLogado = Func; //Funcionário logado atualizado
        Pessoa.userLogado = null; //Utilizador logado é nulo
    }

    if (User != null || Func != null) //Se houve um Utilizador ou um Funcionário logado
    {
        //Abrir o form Menu
        new Menu().Show();
        //Esconder o form atual
        this.Hide();
    }
    //Se não
    else throw new NotExistingUserException(); //Atirar uma exceção
}
```

Figura 25 - Página de login onde verifica se os dados introduzidos pertencem a um utilizador comum ou a um funcionário

```
private void LinkLabelNovoUser_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    new CriarUtilizador().Show();
    this.Hide();
}
```

Figura 26 - Fecha a página de login e abre a página para criar um utilizador

## 2.5.4 Menu

Este é o Menu em que podemos selecionar em qual espaço o utilizador pretende efetuar a sua reserva. O utilizador pode também ter acesso ao seu perfil ao clicar no ícone ao lado do seu nome.

```
private void Menu_Load(object sender, EventArgs e)
{
    if (Pessoa.userLogado == null) // Verificar se a pessoa logada é um utilizador ou funcionário
    {
        // Se for um funcionário
        LblVerPerfil.Text = $"Bem-Vindo, {Pessoa.funcionarioLogado.Nome}";
        string label = "Ver agenda";
        Pessoa.userLogado = null;
        BtnCampo11.Text = label;
        BtnCampo7.Text = label;
        BtnGinasio.Text = label;
        BtnPavilhao.Text = label;
    }
    else
    {
        // Se for um utilizador
        LblVerPerfil.Text = $"Bem-Vindo, {Pessoa.userLogado.Nome}";
        Pessoa.funcionarioLogado = null;
    }
}
```

Figura 27

```
private void BtnCampo7_Click(object sender, EventArgs e)
{
    campo.Show(); // Mostra o forms Detalhes Campo
    campo.PctBoxImagem.Image = Resources.Campode7; // Altera a imagem para a que está no caminho indicado
    campo.LblInfo.Text = "Nome: Campo de Futebol de 7\nCapacidade: 14 pessoas\nPreço/hora: 50€ ";
    AdicionarEmListView("CampoFutebol7"); // Adiciona a Lista de reservas no campo à ListView passando o espaço por parâmetro
    campo.espacos = Enums.Espacos.CampoFutebol7;
    this.Hide(); // Esconder o form Menu
}

1 reference
private void BtnCampo11_Click(object sender, EventArgs e)
{
    campo.Show(); // Mostra o forms Detalhes Campo
    campo.PctBoxImagem.Image = Resources.pexels_mike_1171084; // Altera a imagem para a que está no caminho indicado
    campo.LblInfo.Text = "Nome: Campo de Futebol de 7\nCapacidade: 22 pessoas\nPreço/hora: 70€ ";
    AdicionarEmListView("CampoFutebol11"); // Adiciona a Lista de reservas no campo à ListView passando o espaço por parâmetro
    campo.espacos = Enums.Espacos.CampoFutebol11;
    this.Hide(); // Esconder o form Menu
}

1 reference
private void BtnPavilhao_Click(object sender, EventArgs e)
{
    campo.Show(); // Mostra o forms Detalhes Campo
    campo.PctBoxImagem.Image = Resources.Imagem4_1000x600; // Altera a imagem para a que está no caminho indicado
    campo.LblInfo.Text = "Nome: Pavilhão\nCapacidade: 10 pessoas\nPreço/hora: 40€ ";
    campo.espacos = Enums.Espacos.Pavilhao; // Adiciona a Lista de reservas no campo à ListView passando o espaço por parâmetro
    AdicionarEmListView("Pavilhao");
    this.Hide(); // Esconder o form Menu
}

1 reference
private void BtnGinasio_Click(object sender, EventArgs e)
{
    campo.Show(); // Mostra o forms Detalhes Campo
    campo.PctBoxImagem.Image = Resources.imagem; // Altera a imagem para a que está no caminho indicado
    campo.LblInfo.Text = "Nome: Ginásio\nPreço/hora: 5€ ";
    campo.espacos = Enums.Espacos.Ginasio; // Adiciona a Lista de reservas no campo à ListView passando o espaço por parâmetro
    AdicionarEmListView("Ginasio");
    this.Hide(); // Esconder o form Menu
}
```

Figura 28 - Mostra todos os espaços possíveis para reservar

```

/// <summary>
/// Este método insere todos os dados na ListView
/// </summary>
/// <param name="espaco"></param>
4 references
private void AdicionarEmListView(string espaco)
{
    Reserva reserv = new Reserva();
    List<Reserva> listareserva = reserv.Lista(espaco);
    foreach (Reserva reserva in listareserva)
    {
        campo.LstHorasOcupadas.Items.Add(reserva.Data.ToString("D"));
        campo.LstHorasOcupadas.Items[campo.LstHorasOcupadas.Items.Count - 1].SubItems.Add(reserva.InicioReserva.ToString("HH:mm"));
        campo.LstHorasOcupadas.Items[campo.LstHorasOcupadas.Items.Count - 1].SubItems.Add(reserva.FimReserva.ToString("HH:mm"));
    }
}

```

Figura 29 - Adiciona os espaços de reserva na ListView

```

/// <summary>
/// Acontece quando o evento click na Picture Box é ativado
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void PctBoxPerfil_Click(object sender, EventArgs e)
{
    new VerPerfil().Show(); //Mostra o Forms VerPerfil
    this.Close(); //Fecha o Forms Atual
}

```

Figura 30 - Método para o utilizador ter acesso ao seu perfil

### 2.5.5 VerPerfil

O Forms VerPerfil é composto pelo código de botões, labels, toolstrip, etc. onde encontramos as informações do utilizador na aplicação. Nesta página podemos encontrar as reservas efetuadas pelo usuário, as suas informações pessoais, os seus dados físicos e ainda a possibilidade de eliminar as reservas. Para além disso temos também os diferentes comportamentos entre funcionário e utilizador na aplicação.

```

public partial class VerPerfil : Form
{
    //Construtor
    public VerPerfil()
    {
        InitializeComponent();
        Utilizador.Calculo calculoimc = new Utilizador.Calculo(Utilizador.CalculoIMC); //Utilização do delegado calculo
        Utilizador.Calculo calculomm = new Utilizador.Calculo(Utilizador.CalculoMassaMagra); //Utilização do delegado calculo
        foreach (Utilizador user in Utilizador.listaUtilizador)
        {
            user.IMC = calculoimc(user.Altura, user.Peso);
            user.MassaMagra = calculomm(user.Altura, user.Peso);
        }
        Utilizador.GuardarListaUtilizador();

        //Definição de visibilidade das labels
        #region
        labelAltura.Visible = false;
        labelAntebraco.Visible = false;
        labelBraco.Visible = false;
        labelCoxa.Visible = false;
        labelEnvergadura.Visible = false;
        labelGluteos.Visible = false;
        labelIMC.Visible = false;
        labelMassaCorporal.Visible = false;
        labelOmbros.Visible = false;
        labelPeso.Visible = false;
        labelEmail.Visible = false;
        labelPassword.Visible = false;
        labelNome.Visible = false;
        labelTelemovel.Visible = false;
        #endregion
    }
}

```

Figura 31 - Construtor do cálculo do IMC e massa magra e visibilidades das labels

```
private void ToolStripVerReservas_Click(object sender, EventArgs e)
{
    //Definição de visibilidade das labels
    #region
    LstReservas.Items.Clear();//Limpar Listview
    LstReservas.View = View.Details;//Mostrar detalhes da Listview
    LstReservas.Visible = true;//Definir Listview como visível
    Reserva reserva1 = new Reserva();//Criar nova reserva
    if(Pessoa.funcionarioLogado == null)//Verificar se é um funcionário ou utilizador logado
    {
        //Se for um utilizador
        List<Reserva> listareserva = reserva1.ListaReservas();//Receber a lista de reservas do utilizador logado
        if (listareserva.Count > 0)//Verificar se existe alguma reserva
        {
            //Caso haja
            foreach (Reserva reserva in listareserva)
            {
                //Adicionar lista de reservas na listview
                LstReservas.Items.Add(Utilizador.userLogado.Nome);
                LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.Data.ToString("D"));
                LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.InicioReserva.ToString("HH:mm"));
                LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.FimReserva.ToString("HH:mm"));
                LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.Espaco.ToString());
                LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.Estado.ToString());
            }
        }
        //Se não
        else throw new NotExistingReserveException();
    }
}
```

Figura 32 – Listview em Utilizador

```
else
{
    //Se for um funcionário
    List<Reserva> listareserva = Reserva.lista;//Receber lista com todas as reservas existentes
    foreach (Reserva reserva in listareserva)
    {
        //Adicionar lista de reservas na listview
        LstReservas.Items.Add(reserva.NrProcesso);
        LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.Data.ToString("D"));
        LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.InicioReserva.ToString("HH:mm"));
        LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.FimReserva.ToString("HH:mm"));
        LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.Espaco.ToString());
        LstReservas.Items[LstReservas.Items.Count - 1].SubItems.Add(reserva.Estado.ToString());
    }
}
```

Figura 33 – ListView dos funcionários

```

if (Pessoa.userLogado == null)//Verificar se a pessoa logada é um funcionário ou um utilizador
{
    //Se for funcionário
    //Preecher as TextBox com informações
    TxtNome.Text = Funcionario.funcionarioLogado.Nome;
    TxtPassword.Text = Funcionario.funcionarioLogado.Password;
    TxtNrTelemovel.Text = Funcionario.funcionarioLogado.NumeroTlm;
    TxtEmail.Text = Funcionario.funcionarioLogado.Email;
}
else
{
    //Se for utilizador
    //Preecher as TextBox com informações
    TxtNome.Text = Utilizador.userLogado.Nome;
    TxtPassword.Text = Utilizador.userLogado.Password;
    TxtNrTelemovel.Text = Utilizador.userLogado.NumeroTlm;
    TxtEmail.Text = Utilizador.userLogado.Email;
}

```

Figura 34 - Distinção de login

```

private void BtnAlterar_Click(object sender, EventArgs e)
{
    if (Pessoa.userLogado == null)//Verificar se a pessoa logada é um funcionário ou um utilizador
    {
        //Se for funcionário
        //Preecher o funcionario com as informações das TextBox
        Funcionario.funcionarioLogado.Nome = TxtNome.Text;
        Funcionario.funcionarioLogado.Email = TxtEmail.Text;
        Funcionario.funcionarioLogado.Password = TxtPassword.Text;
        Funcionario.funcionarioLogado.NumeroTlm = TxtNrTelemovel.Text;
        Funcionario.GuardarListaFuncionario();//Guardar a lista de funcionários
    }
    else
    {
        //Se for utilizador
        //Preecher o funcionario com as informações das TextBox
        Pessoa.userLogado.Nome = TxtNome.Text;
        Pessoa.userLogado.Email = TxtEmail.Text;
        Pessoa.userLogado.Password = TxtPassword.Text;
        Pessoa.userLogado.NumeroTlm = TxtNrTelemovel.Text;
        Utilizador.GuardarListaUtilizador();//Guardar a lista de utilizadores
    }
}

```

Figura 35 – Preenchimento de lista consoante o login do utilizador



```

if (Pessoa.userLogado == null)//Verificar se a pessoa logada é um funcionário ou um utilizador
{
    //Se for funcionário
    //Definição de visibilidade
    #region
}
else
{
    //Se for utilizador
    //Preencher as TextBox com informações do utilizador
    TxtPeso.Text = Utilizador.userLogado.Peso;
    TxtMassaCorporal.Text = Utilizador.userLogado.MassaMagra;
    TxtAltura.Text = Utilizador.userLogado.Altura;
    TxtOmbros.Text = Utilizador.userLogado.Ombros;
    TxtBraco.Text = Utilizador.userLogado.Braco;
    TxtAntebraco.Text = Utilizador.userLogado.Antebraco;
    TxtEnvergadura.Text = Utilizador.userLogado.Envergadura;
    TxtGluteos.Text = Utilizador.userLogado.Gluteos;
    TxtCoxa.Text = Utilizador.userLogado.Coxa;
    TxtIMC.Text = Utilizador.userLogado.IMC;
}

```

Figura 36 - Preenchimento de dados físicos

```

private void BtnEliminarReserva_Click(object sender, EventArgs e)
{
    if (LstReservas.SelectedItems.Count > 0)//Verificar se está algum item selecionado
    {
        Reserva.lista.RemoveAt(LstReservas.SelectedIndex[0]);//Remover essa reserva da lista
        LstReservas.Items.Remove(LstReservas.SelectedItems[0]);//Remover essa reserva da ListView
    }
    Reserva.GuardarListaReserva();//Guardar a lista de reservas
}

```

Figura 37 - Remover Reservas

```
private void BtnAlterarDados_Click(object sender, EventArgs e)
{
    string texto = ComboBoxUser.Text;
    List<Utilizador> utilizador = Utilizador.listaUtilizador; //Receber a lista de utilizadores
    foreach (Utilizador user in utilizador)
    {
        if (texto == user.NrProcessoEscola) //Verificar se o numero de processo da combobox é igual ao do utilizador
        {
            //Se sim, atualizar os dados do utilizador
            user.Peso = TxtPeso.Text;
            user.MassaMagra = TxtMassaCorporal.Text;
            user.Altura = TxtAltura.Text;
            user.Ombros = TxtOmbros.Text;
            user.Brace = TxtBraco.Text;
            user.Antebraco = TxtAntebraco.Text;
            user.Envergadura = TxtEnvergadura.Text;
            user.Gluteos = TxtGluteos.Text;
            user.Coxa = TxtCoxa.Text;
            user.IMC = TxtIMC.Text;
        }
    }
    Utilizador.GuardarListaUtilizador(utilizador); //Guardar a lista de utilizadores
}
```

Figura 38 - Alteração de dados físicos

## 2.6 Interfaces

```
namespace Polidesportivo_Ipca
{
    1 reference
    interface IUtilizador
    {
        /// <summary>
        /// Metodo para filtrar o primeiro utilizador que valida a condição
        /// </summary>
        /// <param name="nrprocesso">Numero de Processo do Utilizador</param>
        /// <param name="password">Password do Utilizador</param>
        /// <returns>Retorna Utilizador que valida as condições</returns>
        2 references
        Utilizador UtilizadorLogin(string nrprocesso, string password);
    }
}
```

Figura 39 - Interface Utilizador

```
namespace Polidesportivo_Ipca
{
    1 reference
    interface IFuncionario
    {
        /// <summary>
        /// Metodo para filtrar o primeiro funcionario que valida a condição
        /// </summary>
        /// <param name="nrprocesso">Numero de Processo do funcionário</param>
        /// <param name="pass">Password do funcionario</param>
        /// <returns></returns>
        2 references
        Funcionario FuncionarioLogin(string nrprocesso, string pass);
    }
}
```

Figura 40 - Interface Funcionário

### 3. Aplicação funcional

Nas capturas de ecrã seguintes, é possível visualizar a aplicação no seu estado funcional. Aqui estão expostas todas as partes do programa.

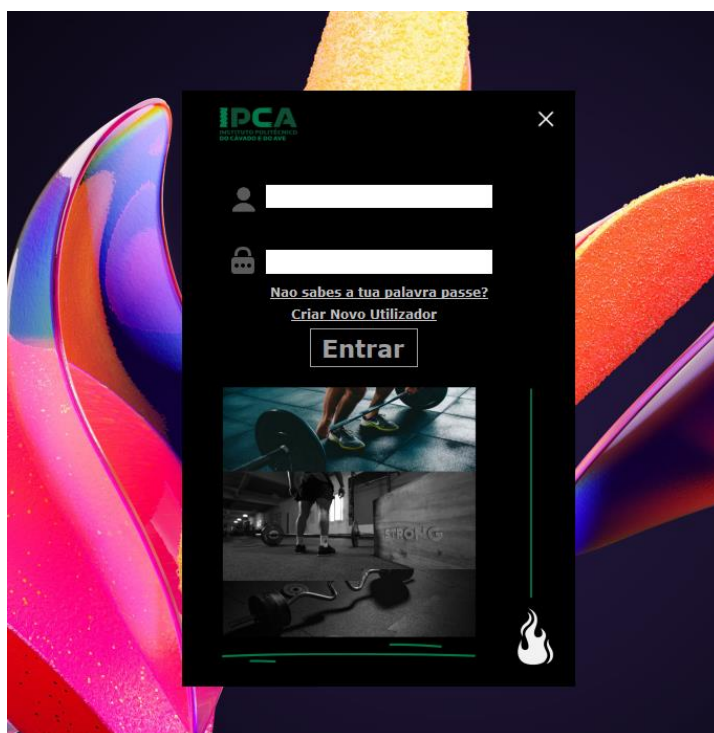
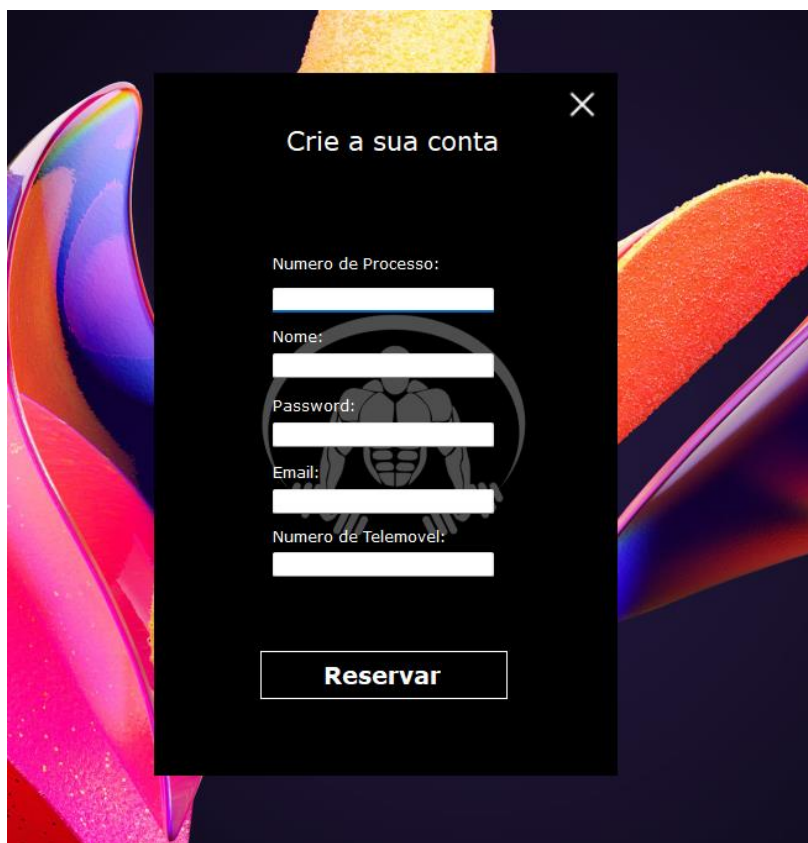


Figura 41 - No menu de login introduzimos o numero de processo do utilizador ou do funcionário e a devida password



**Crie a sua conta** ✕

Numero de Processo:

Nome:

Password:

Email:

Numero de Telemovel:

**Reservar**

Figura 42 - Aqui podemos criar a conta do utilizador preenchendo os campos requeridos



Figura 43 - No menu principal, o utilizador seleciona o espaço desejado para reservar

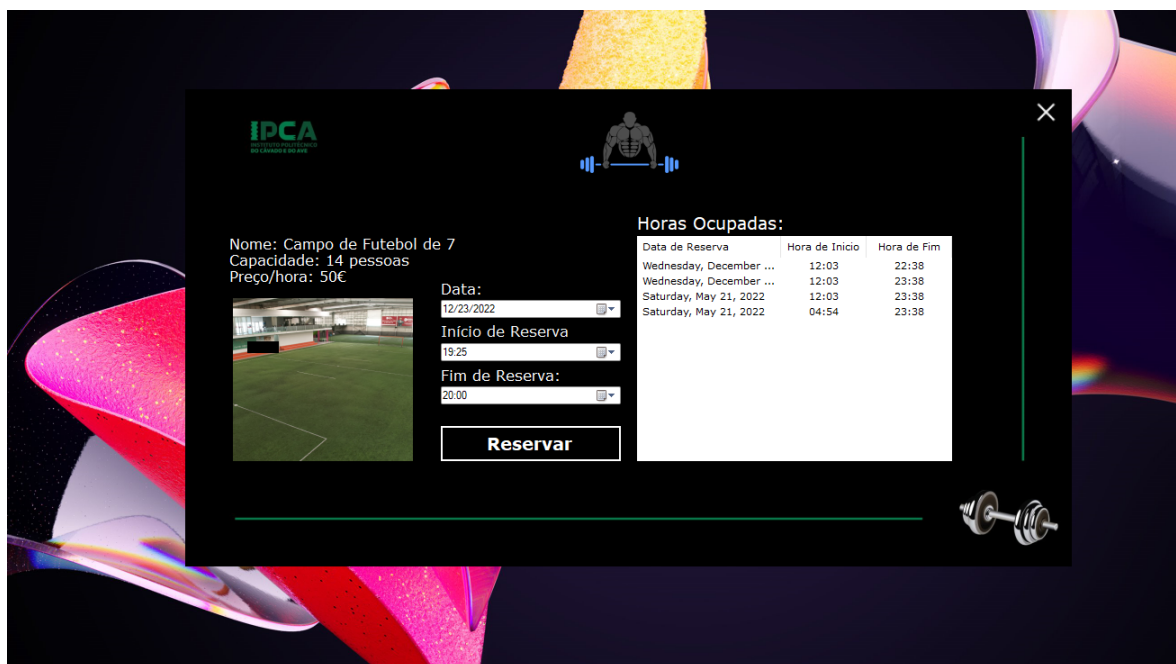


Figura 44 - Dado que o utilizador escolhe o espaço “Campo de Futebol de 7”, este pode ver as horas as que já estão ocupadas e efetua a sua reserva de acordo com as horas vagas

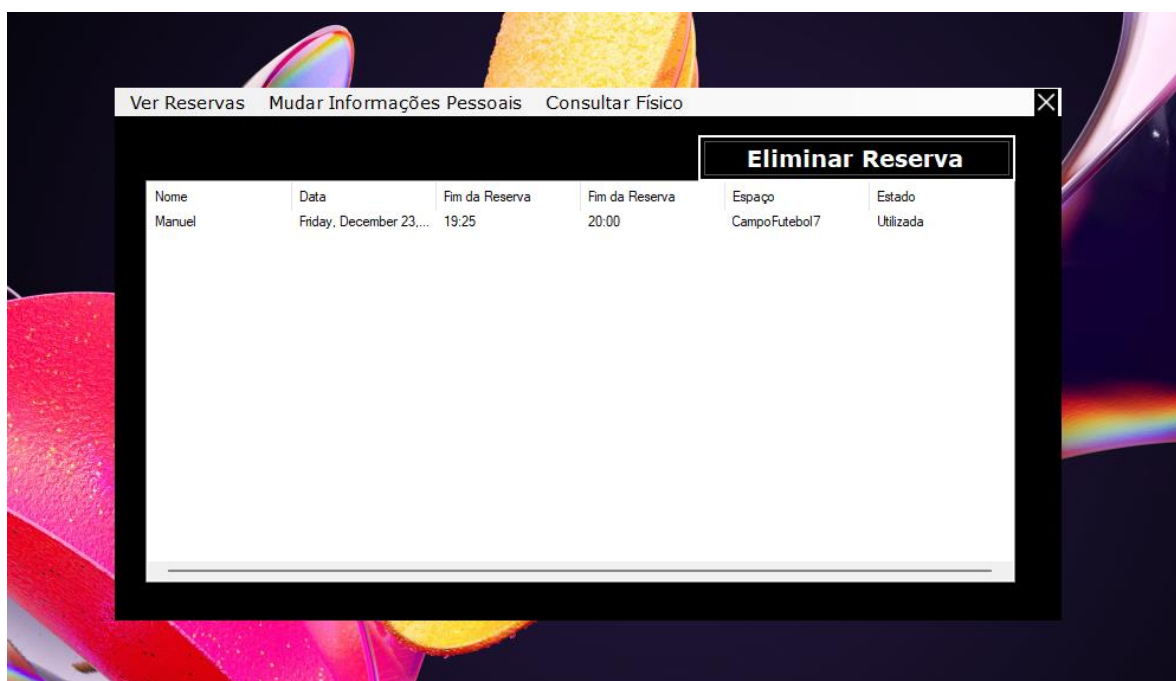


Figura 45 - Ao clicarmos no ícone de utilizador junto ao seu nome, acedamos à página do utilizador. Aqui este pode consultar as suas reservas, as suas informações pessoais e o seu físico. Na aba “Ver Reservas”, este pode verificar as reservas agendadas e cancelar

Ver Reservas   Mudar Informações Pessoais   Consultar Físico

Nome:    Password:

Email:    Telemovel:

**Alterar**

Figura 46 - Na aba “Mudar Informações Pessoais”, o utilizador pode rever e alterar as informações que deseje

Ver Reservas   Mudar Informações Pessoais   Consultar Físico

Nº Processo:

Peso:    Massa Corporal:

Altura:    Ombros:

Braço:    Antebraço:

Envergadura:    Gluteos:

Coxa:    IMC:

Figura 47 - No sector “Consultar Físico”, o utilizador pode consultar e definir o seu progresso físico.





Figura 48 - Na visão de um funcionário, este pode consultar a agenda de um ou mais espaços.

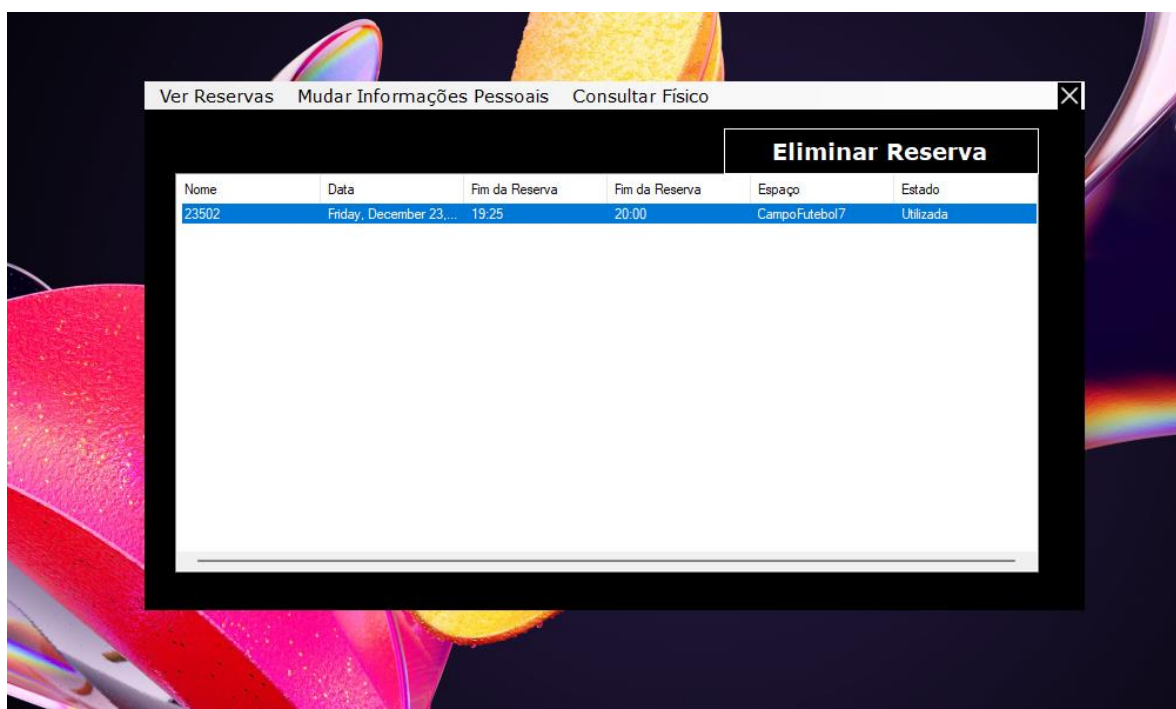


Figura 49 - Suponhamos que a funcionária Lurdes pretende cancelar a reserva efetuada pelo Manuel há momentos atrás como demonstrado. A Lurdes, pode escolher um espaço e ver todas as reservas lá efetuadas, e assim, cancelar as que pretenda.

## 4. Conclusão

Em suma, através deste trabalho, podemos afirmar que sentimos uma grande evolução no que toca a C#, programação orientada a objetos e organização. Foi realmente um projeto em que gostamos de trabalhar e nos esforçar, pois foi bastante desafiador. Até porque, é um tema que facilmente é aplicado no mundo real e no nosso campus.

Em suma, com a realização deste trabalho, aprofundamos os conceitos e fundamentos adquiridos em ambiente académico, seja em aula, seja em estudo nas sessões que organizamos em grupo para a elaboração deste trabalho. Adquirimos também, bem como desenvolvemos, as nossas capacidades de pesquisa e otimização de código-fonte, quer pelas vias digitais, quer por desenvolvimento próprio. Consideramos que a realização deste trabalho, pela complexidade do mesmo, contribuiu de forma positiva para o nosso desenvolvimento enquanto estudantes, no que diz respeito à colaboração entre colegas, bem como para uma melhor perceção do trabalho em si, uma vez que o mesmo se apresenta como um caso prático, preparando-nos assim para uma futura aplicação real, sendo até mesmo mais intuitiva a realização do mesmo, quando comparada com situações/exercícios hipotéticos.

## 5. Bibliografia

Documentação Microsoft - <https://learn.microsoft.com/en-us/dotnet/csharp/>