Relatório Final

Projeto Final Tecnologias Internet III

Bruno Silva 2022110 2LEI

Gestão Medicação

Prof João Rebelo

Ano Letivo 2023/2024

ISTEC – Instituto Tecnologias Avançadas Porto

Índice

Introdução

O projeto apresentado tenta solucionar e ajudar na organização do consumo de medicação para uma pessoa.

Em Portugal, e com uma população cada vez mais envelhecida, é normal os utentes esquecerem-se da toma de certos medicamentos, ou até nem terem referência sobre os mesmos para consulta futura, quer por perda da documentação, quer por esquecimento.

Principais objetivos da solução:

- Fazer a gestão de medicação de uma pessoa
- Fazer a gestão do calendário de consumo
- Criação, Leitura, Atualização e Eliminação de medicamentos na plataforma

Deixar a nota que a solução, apesar de ser um CRUD, tem apenas a parte de Criação e Leitura funcionais.

Metodologia

O projeto constitui a seguinte metodologia:

- Um utilizador
- Para cada utilizador, vários medicamentos (lista)
- Para cada utilizador, apenas um calendário
- Um utilizador pode ser tanto um utilizador padrão como um utilizador supervisor (para gestão da plataforma)

Diagrama de Classes

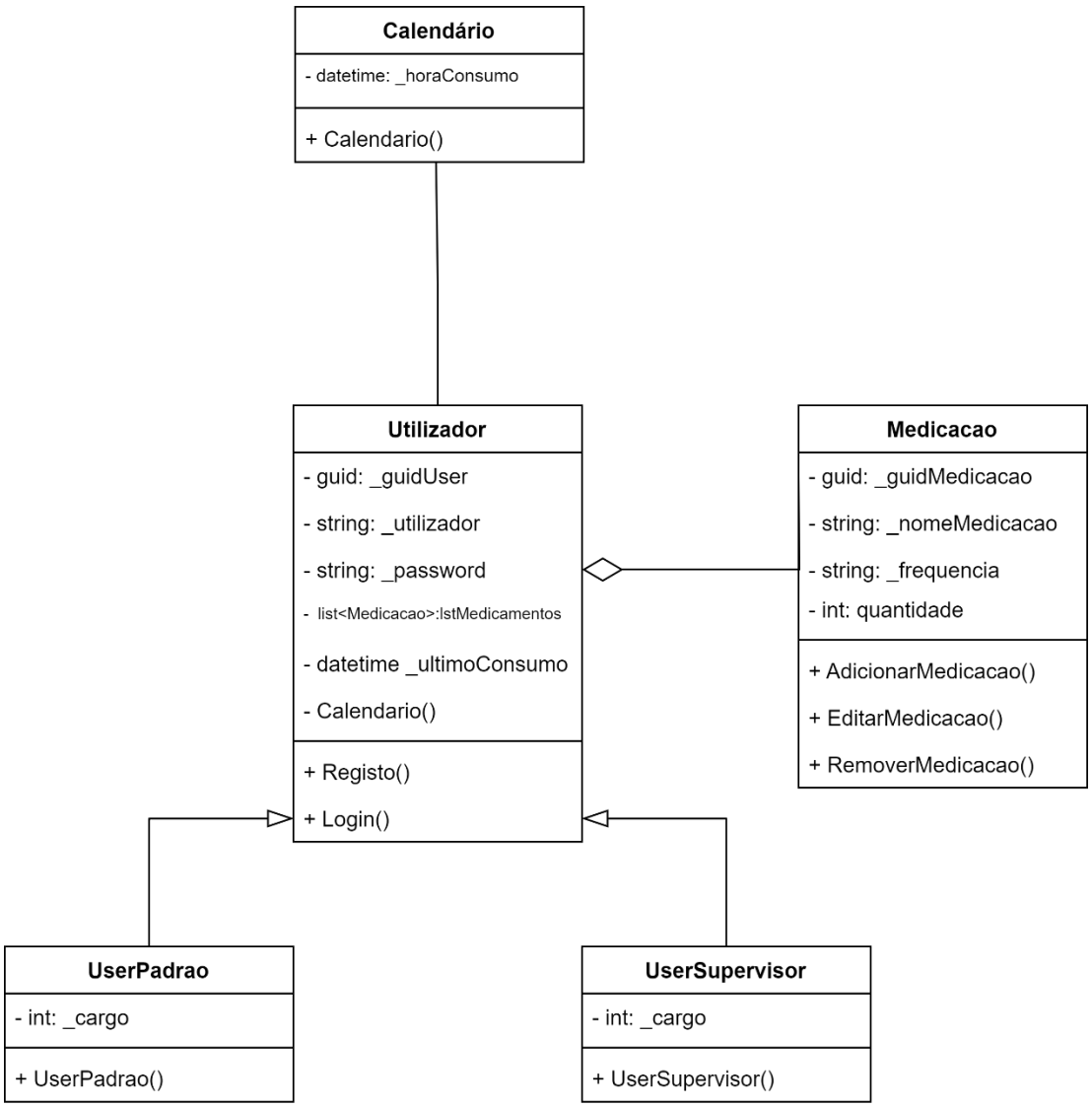


Diagrama de Use-Cases

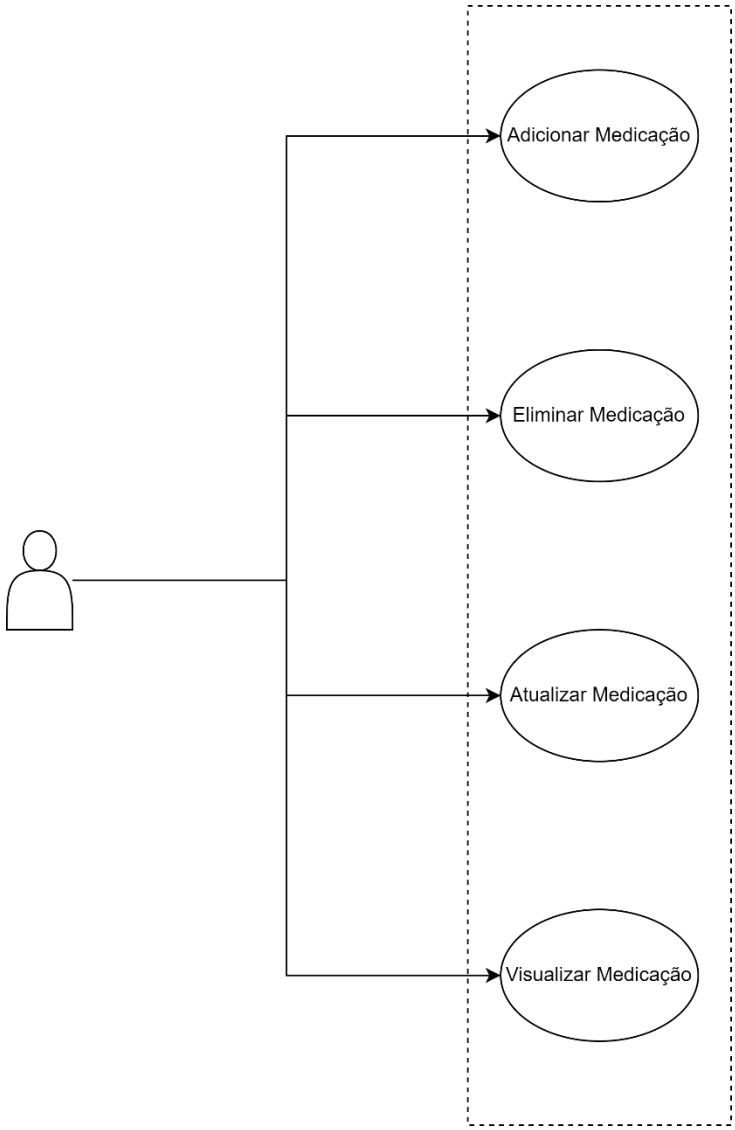
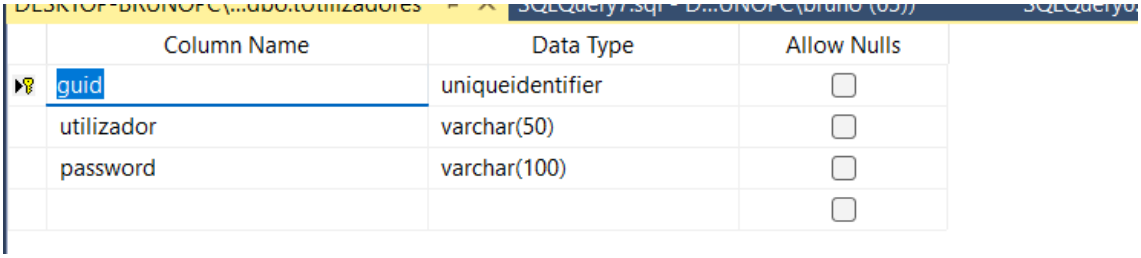


Tabela Base Dados

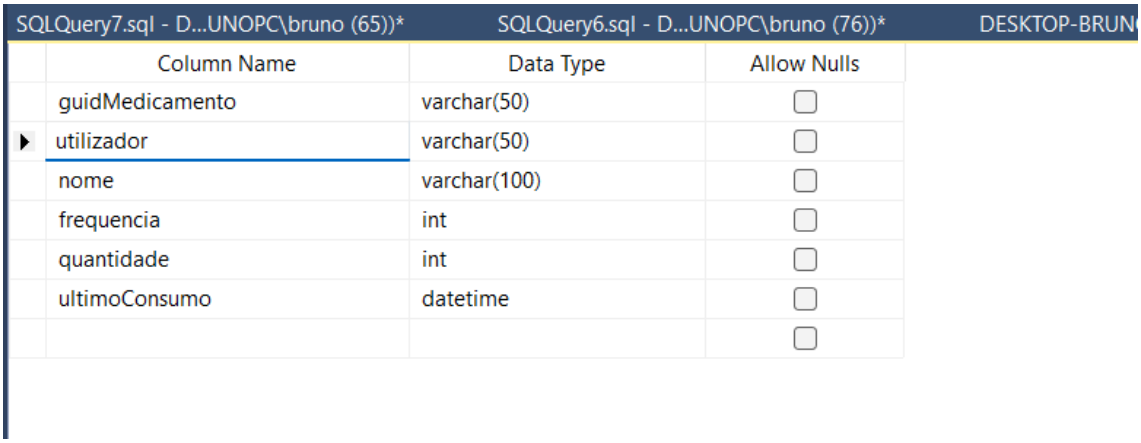
A base de dados é composta por tabelas tUtilizadores e tMedicação, ambas responsáveis por armazenar tanto os respetivos utilizadores e a medicação de cada um.

Tabela tUtilizadores:



Column Name	Data Type	Allow Nulls
guid	uniqueidentifier	☐
utilizador	varchar(50)	☐
password	varchar(100)	☐
		☐

Tabela tMedicação:



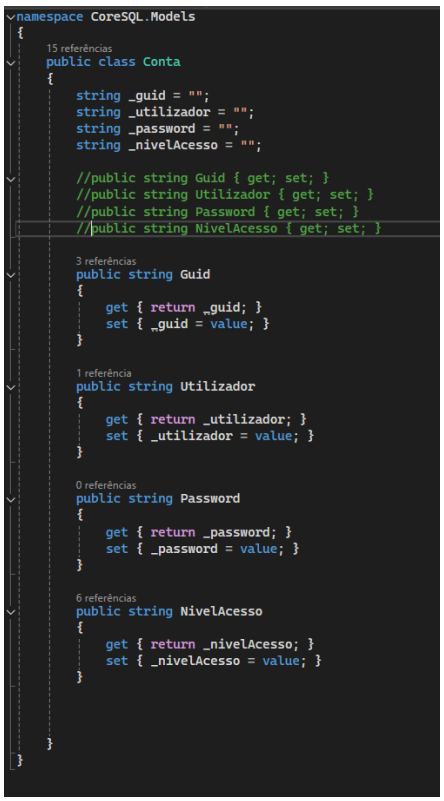
SQLQuery7.sql - D...UNOPC\bruno (65))* SQLQuery6.sql - D...UNOPC\bruno (76))* DESKTOP-BRUN

Column Name	Data Type	Allow Nulls
guidMedicamento	varchar(50)	☐
utilizador	varchar(50)	☐
nome	varchar(100)	☐
frequencia	int	☐
quantidade	int	☐
ultimoConsumo	datetime	☐
		☐

Models

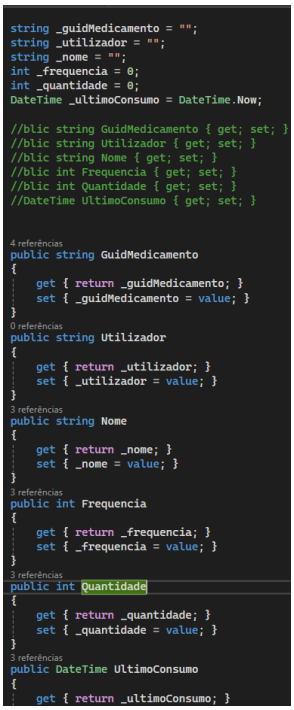
Nas classes Conta e Medicamento, defini as regras de negócio para cada classe.

A classe Conta define as regras de negócio para cada um dos seus atributos e métodos. No caso, todos os atributos são tanto de leitura como de escrita.



```
namespace CoreSQL.Models
{
    15 referências
    public class Conta
    {
        string _guid = "";
        string _utilizador = "";
        string _password = "";
        string _nivelAcesso = "";

        //public string Guid { get; set; }
        //public string Utilizador { get; set; }
        //public string Password { get; set; }
        //public string NivelAcesso { get; set; }

        3 referências
        public string Guid
        {
            get { return „guid; }
            set { „guid = value; }
        }

        1 referência
        public string Utilizador
        {
            get { return _utilizador; }
            set { _utilizador = value; }
        }

        0 referências
        public string Password
        {
            get { return _password; }
            set { _password = value; }
        }

        6 referências
        public string NivelAcesso
        {
            get { return _nivelAcesso; }
            set { _nivelAcesso = value; }
        }


    }
}
```

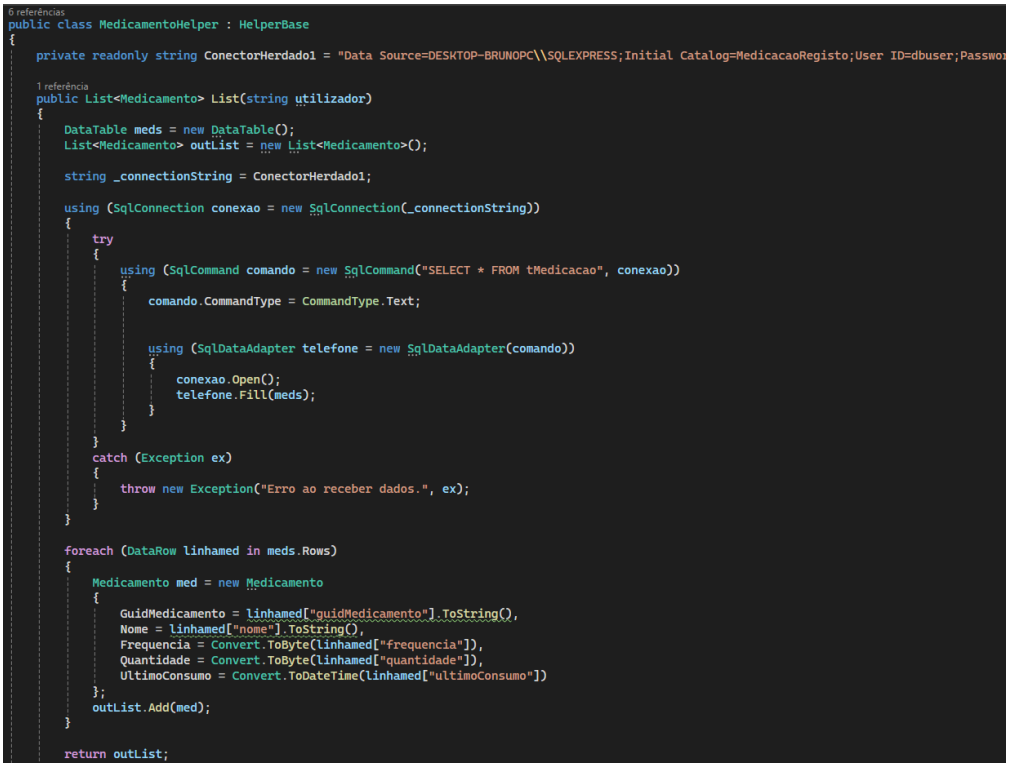
A classe Medicamento também segue a mesma ideia:



```
string _guidMedicamento = "";
string _utilizador = "";
string _nome = "";
int _frequencia = 0;
int _quantidade = 0;
DateTime _ultimoConsumo = DateTime.Now;

//blic string GuidMedicamento { get; set; }
//blic string Utilizador { get; set; }
//blic string Nome { get; set; }
//blic int Frequencia { get; set; }
//blic int Quantidade { get; set; }
//DateTime UltimoConsumo { get; set; }


4 referências
public string GuidMedicamento
{
    get { return _guidMedicamento; }
    set { _guidMedicamento = value; }
}
0 referências
public string Utilizador
{
    get { return _utilizador; }
    set { _utilizador = value; }
}
3 referências
public string Nome
{
    get { return _nome; }
    set { _nome = value; }
}
3 referências
public int Frequencia
{
    get { return _frequencia; }
    set { _frequencia = value; }
}
3 referências
public int Quantidade
{
    get { return _quantidade; }
    set { _quantidade = value; }
}
3 referências
public DateTime UltimoConsumo
{
    get { return _ultimoConsumo; }
```

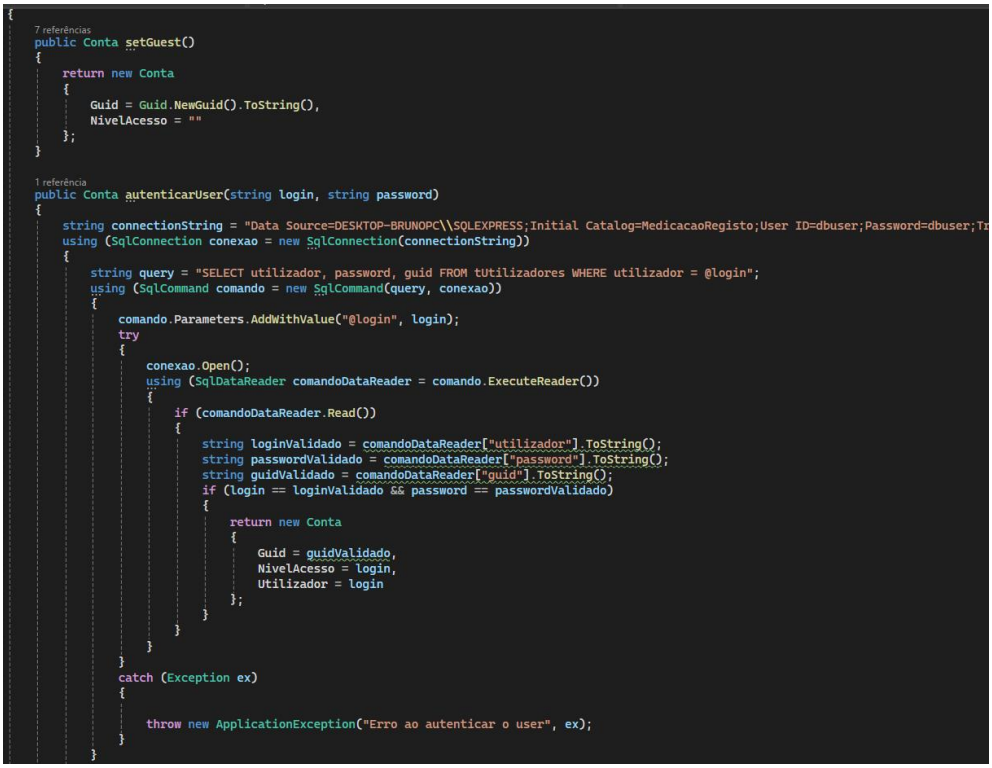
A classe MedicamentoHelper contém as funções essenciais do código, como a de Listar a medicação para um determinado utilizador, por exemplo.

A mesma herda a partir da classe HelperBase.



```
6 referências
public class MedicamentoHelper : HelperBase
{
    private readonly string ConectorHerdado1 = "Data Source=DESKTOP-BRUNOPC\\SQLEXPRESS;Initial Catalog=MedicacaoRegisto;User ID=dbuser;Passwo

    1 referência
    public List<Medicamento> List(string utilizador)
    {
        DataTable meds = new DataTable();
        List<Medicamento> outList = new List<Medicamento>();

        string _connectionString = ConectorHerdado1;

        using (SqlConnection conexao = new SqlConnection(_connectionString))
        {
            try
            {
                using (SqlCommand comando = new SqlCommand("SELECT * FROM tMedicacao", conexao))
                {
                    comando.CommandType = CommandType.Text;


                    using (SqlDataAdapter telefone = new SqlDataAdapter(comando))
                    {
                        conexao.Open();
                        telefone.Fill(meds);
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao receber dados.", ex);
            }
        }

        foreach (DataRow linhamed in meds.Rows)
        {
            Medicamento med = new Medicamento
            {
                GuidMedicamento = linhamed["guidMedicamento"].ToString(),
                Nome = linhamed["nome"].ToString(),
                Frequencia = Convert.ToByte(linhamed["frequencia"]),
                Quantidade = Convert.ToByte(linhamed["quantidade"]),
                UltimoConsumo = Convert.ToDateTime(linhamed["ultimoConsumo"])
            };
            outList.Add(med);
        }

        return outList;
```

A classe ContaHelper encarrega-se de todas as operações para as contas da solução, como a respetiva autenticação, registo e serialização para a sessão.



```
{
    7 referências
    public Conta setGuest()
    {
        return new Conta
        {
            Guid = Guid.NewGuid().ToString(),
            NivelAcesso = ""
        };
    }

    1 referência
    public Conta autenticarUser(string login, string password)
    {
        string connectionString = "Data Source=DESKTOP-BRUNOPC\\SQLEXPRESS;Initial Catalog=MedicacaoRegisto;User ID=dbuser;Password=dbuser;Tr
        using (SqlConnection conexao = new SqlConnection(connectionString))
        {
            string query = "SELECT utilizador, password, guid FROM tUtilizadores WHERE utilizador = @login";
            using (SqlCommand comando = new SqlCommand(query, conexao))
            {
                comando.Parameters.AddWithValue("@login", login);
                try
                {
                    conexao.Open();
                    using (SqlDataReader comandoDataReader = comando.ExecuteReader())
                    {
                        if (comandoDataReader.Read())
                        {
                            string loginValidado = comandoDataReader["utilizador"].ToString();
                            string passwordValidado = comandoDataReader["password"].ToString();
                            string guidValidado = comandoDataReader["guid"].ToString();
                            if (login == loginValidado && password == passwordValidado)
                            {
                                return new Conta
                                {
                                    Guid = guidValidado,
                                    NivelAcesso = login,
                                    Utilizador = login
                                };
                            }
                        }
                    }
                }
                catch (Exception ex)
                {

                    throw new ApplicationException("Erro ao autenticar o user", ex);
                }
            }
```

Views

As views possuem todas as interfaces gráficas que o utilizador irá diretamente interagir na solução.

Como base para a View, foi usado no Layout a navbar para toda a plataforma, e a respetiva função entre o Login/Logout.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
        }

        .navbar {
            background-color: #333;
            overflow: hidden;
        }

            .navbar a {
                float: left;
                display: block;
                color: #f2f2f2;
                text-align: center;
                padding: 14px 20px;
                text-decoration: none;
            }

                .navbar a:hover {
                    background-color: #ddd;
                    color: black;
                }
    </style>
</head>
<body>

    <div class="navbar">
        <a href="/">Inicio</a>
        <a href="#medicacao">Medicação</a>
        @if (!string.IsNullOrEmpty(ViewBag.ContaAtiva?.NivelAcesso))
        {
            <a href="/Conta/Logout">Logout (@ViewBag.ContaAtiva.Utilizador)</a>
        }
        else
        {
            <a href="/Conta/Registar">Registe-se</a>
            <a href="/Conta/Login">Login</a>
        }

    </div>

    @RenderBody()
```

Nas views de Login e Registar, foram usados inputs, para fazerem o request POST para o backend, para efetuar o Login ou o Registo, respetivamente.

```
@{
    ViewData["Title"] = "Login";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="container">
    <div class="row">
        <div class="col-lg-8 col-lg-offset-2 col-md-10 col-md-offset-1">
            <p>Acesso ao Portal:</p>
            <form name="frmLogin" method="post" action="/Conta/Login" validate>
                <div class="row control-group">
                    <div class="form-group col-xs-12 floating-label-form-group controls">
                        <label>Utilizador</label>
                        <input type="text" class="form-control" placeholder="Utilizador" name="Utilizador" id="Utilizador" requ
                        <p class="help-block text-danger"></p>
                    </div>
                </div>
                <div class="row control-group">
                    <div class="form-group col-xs-12 floating-label-form-group controls">
                        <label>Password</label>
                        <input type="password" class="form-control" placeholder="Password" name="Password" id="Password" requir
                        <p class="help-block text-danger"></p>
                    </div>
                </div>
                <br>
                <div id="success"></div>
                <div class="row">
                    <div class="form-group col-xs-12">
                        <button id="submit" type="submit" name="submit" class="btn btn-default">Entrar</button>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>

<hr>
```

```
@{
    ViewData["Title"] = "Registar";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="container">
    <div class="row">
        <div class="col-lg-8 col-lg-offset-2 col-md-10 col-md-offset-1">
            <p>Registo no Portal:</p>
            <form name="frmLogin" method="post" action="/Conta/Registar" validate>
                <div class="row control-group">
                    <div class="form-group col-xs-12 floating-label-form-group controls">
                        <label>Utilizador</label>
                        <input type="text" class="form-control" placeholder="Utilizador" name="Utilizador" id="Utilizador" required data-validation-required-message="Indique o seu utilizador">
                        <p class="help-block text-danger"></p>
                    </div>
                </div>
                <div class="row control-group">
                    <div class="form-group col-xs-12 floating-label-form-group controls">
                        <label>Password</label>
                        <input type="password" class="form-control" placeholder="Password" name="Password" id="Password" required data-validation-required-message="Indique a sua password">
                        <p class="help-block text-danger"></p>
                    </div>
                </div>
                <br>
                <div id="success"></div>
                <div class="row">
                    <div class="form-group col-xs-12">
                        <button id="submit" type="submit" name="submit" class="btn btn-default">Entrar</button>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>

<hr>
```

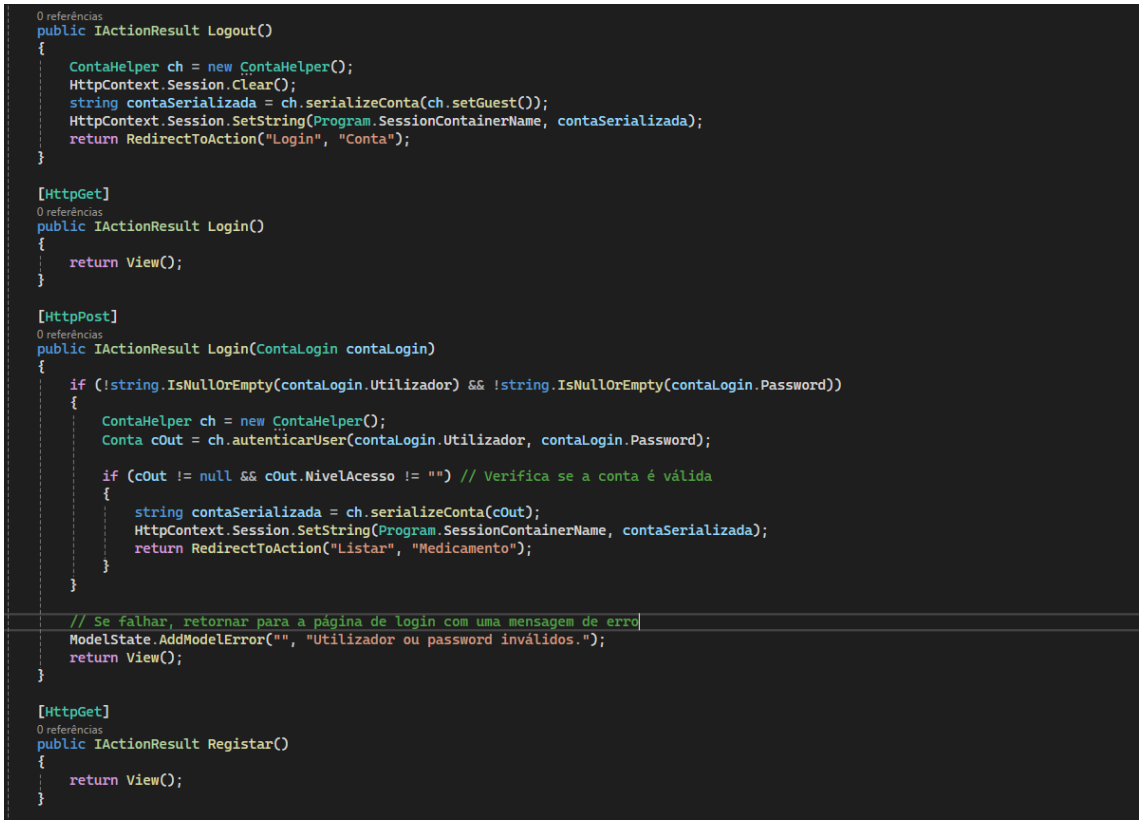
Para as views de Medicamento, em Editar e Listar, o acesso às mesmas é limitado na View, onde verifica se o utilizador tem uma sessão válida ativa; caso contrário, redireciona para a raíz do servidor.

```
@{
    ViewData["Title"] = "Lista de Medicamentos";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@ViewData["Editar"]</h2>

@if (!string.IsNullOrEmpty(ViewBag.ContaAtiva.Utilizador))
{
    <h1>AUTENTICADO!!</h1>

    <form asp-action="/Medicamento/Editar" method="post">
        <div class="form-group">
            <label asp-for="Nome" class="control-label">Nome do Medicamento:</label>
            <input asp-for="Nome" class="form-control" />
            <span asp-validation-for="Nome" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Frequencia" class="control-label">Frequência de Consumo:</label>
            <input asp-for="Frequencia" class="form-control" />
            <span asp-validation-for="Frequencia" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Quantidade" class="control-label">Quantidade (em unidades):</label>
            <input asp-for="Quantidade" class="form-control" />
            <span asp-validation-for="Quantidade" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="UltimoConsumo" class="control-label">Data do Último Consumo:</label>
            <input asp-for="UltimoConsumo" class="form-control" type="datetime-local" />
            <span asp-validation-for="UltimoConsumo" class="text-danger"></span>
        </div>
        <div class="form-group">
            <input type="submit" value="Salvar" class="btn btn-primary" />
        </div>
    </form>
}
else
{
    <h2>ANÓNIMO!!</h2>
    <meta http-equiv="refresh" content="0; URL=/" />
}
```

Controllers

Os controladores na solução estão a efetuar as funções principalmente do backend, para cada tipo de request presente vindo dos outros componentes da solução.

A classe ContaController faz interface com os requests e tipos de requests recebidos do frontend, neste caso para o Login, Logout, Sessão e Registo.



```
0 referências
public IActionResult Logout()
{
    ContaHelper ch = new ContaHelper();
    HttpContext.Session.Clear();
    string contaSerializada = ch.serializeConta(ch.setGuest());
    HttpContext.Session.SetString(Program.SessionContainerName, contaSerializada);
    return RedirectToAction("Login", "Conta");
}

[HttpGet]
0 referências
public IActionResult Login()
{
    return View();
}

[HttpPost]
0 referências
public IActionResult Login(ContaLogin contaLogin)
{
    if (!string.IsNullOrEmpty(contaLogin.Utilizador) && !string.IsNullOrEmpty(contaLogin.Password))
    {
        ContaHelper ch = new ContaHelper();
        Conta cOut = ch.autenticarUser(contaLogin.Utilizador, contaLogin.Password);

        if (cOut != null && cOut.NivelAcesso != "") // Verifica se a conta é válida
        {
            string contaSerializada = ch.serializeConta(cOut);
            HttpContext.Session.SetString(Program.SessionContainerName, contaSerializada);
            return RedirectToAction("Listar", "Medicamento");
        }
    }

    // Se falhar, retornar para a página de login com uma mensagem de erro
    ModelState.AddModelError("", "Utilizador ou password inválidos.");
    return View();
}

[HttpGet]
0 referências
public IActionResult Registar()
{
    return View();
}
```

```
[HttpGet]
0 referências
public IActionResult Registar()
{
    return View();
}

[HttpPost]
0 referências
public IActionResult Registar(ContaRegisto contaRegisto)
{
    if (contaRegisto.Utilizador != "" && contaRegisto.Password != "")
    {
        ContaHelper ch = new ContaHelper();
        //cOut é um admin nestas condições
        Conta cOut = ch.registarUser(contaRegisto.Utilizador, contaRegisto.Password);
        string contaSerializada = ch.serializeConta(cOut);
        HttpContext.Session.SetString(Program.SessionContainerName, contaSerializada);
    }

    return RedirectToAction("Login", "Conta");
}
```

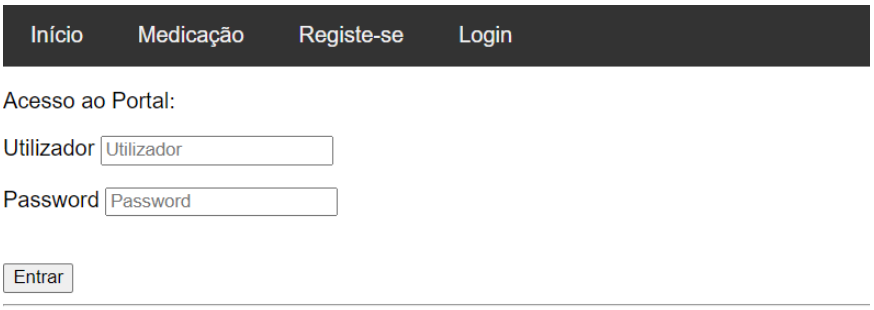
A mesma ideologia acontece para a classe MedicamentoController, com o Listar e Editar:

```
0 referências
public IActionResult Listar(string op)
{
    ViewBag.NivelAcesso = HttpContext.Session.GetString("nivelAcesso");
    string userSessao = HttpContext.Session.GetString("Utilizador");


    MedicamentoHelper mh = new MedicamentoHelper();
    List<Medicamento> lista = mh.List(userSessao);

    return View(lista);
}



[HttpGet]
0 referências
public IActionResult Criar() {
    //string ligacao = Program.conexaoGlobal;
    //DocumentoHelper dh = new DocumentoHelper(ligacao);
    //Documento doc = new Documento {
    //    DtPublicacao = DateTime.Now,
    //    Estado = 1,
    //    Resumo = "Criado automaticamente mas com guids distintos",
    //    Titulo = "Criado Automatico" };
    //dh.save(doc);
    //return RedirectToAction("Listar", "Documento");
    return View();
}



[HttpGet]
0 referências
public IActionResult Editar(string op)
{
    if (_conta.NivelAcesso != null || _conta.NivelAcesso != "")
    {
        MedicamentoHelper mh = new MedicamentoHelper();
        Medicamento? med = mh.Get(op);
        if (med == null) RedirectToAction("Listar", "Medicamento");
        return View(med);
    }
    return RedirectToAction("Login", "Conta");

}
```

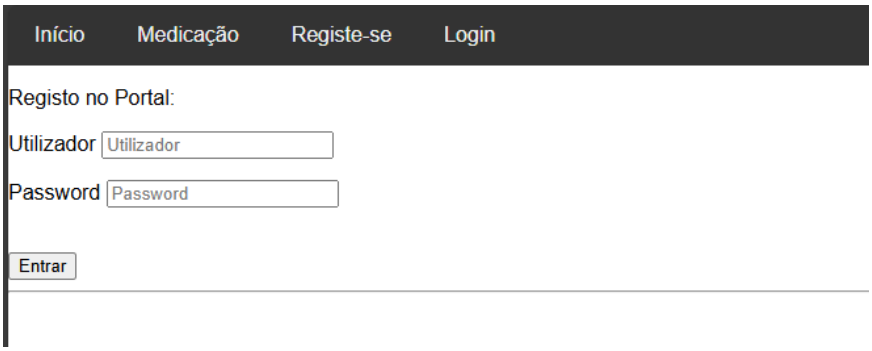
Visão FrontEnd do projeto

Em continuação com o desenvolvimento das Views, o FrontEnd dispõe-se do seguinte:

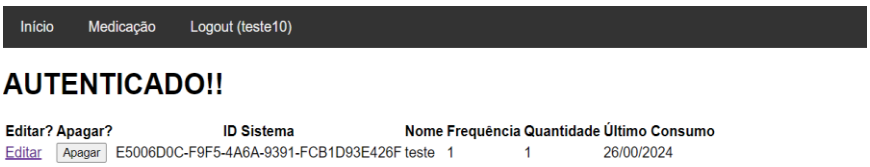
- Página de Login de Utilizadores:



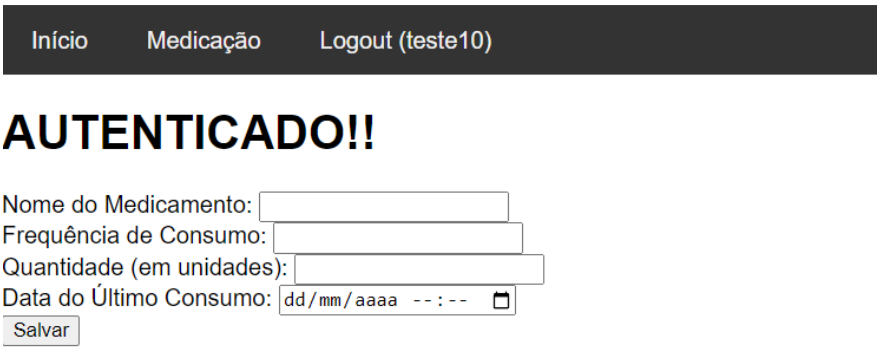
- Página de Registo de Utilizadores:



- Página de Listagem de Medicamentos no Utilizador:



- Página para edição de um medicamento:



Conclusão

O trabalho, mesmo não ficando com as funções do CRUD completas, foi desafiante, ainda mais em ASP .NET Core, tecnologia que foi iniciante para mim neste ano. Igualmente o modelo MVC mostrou ser um bocado desafiante para mim.

Porém, foi completo para entender melhor o modelo MVC e as suas propriedades, aprofundar algumas habilidades com o ASP .NET Core, e também no próprio Visual Studio.

De igual forma, também consegui explorar as ferramentas do Git/GitHub, e do GitHub Desktop, todas elas usadas em força neste projeto para logging de alterações e análise de commits.