



## MANUAL DE PROGRAMADOR

**Programa:** MyHealth - Management

**Criador:** Vicente Loução Duarte

## Índice

MANUAL DE PROGRAMADOR .....	1
Introdução.....	2
Requisitos do Sistema .....	2
Base de Dados.....	3
Principais classes e Funções .....	4
Entity Framework (Acesso a Dados) .....	4
ConfiguracoesApp .....	7
ConfigManager .....	8
Contas.....	9
Theme .....	10
Funcionalidades Técnicas.....	10
Como Instalar? .....	11

## Introdução

Este manual tem como objetivo fornecer toda a informação técnica necessária para compreender, instalar, modificar e dar manutenção ao sistema "**MyHealth - Management**". O projeto foi desenvolvido em C# com Windows Forms, utilizando o MaterialSkin para o design da interface e o Entity Framework para a camada de acesso a dados.

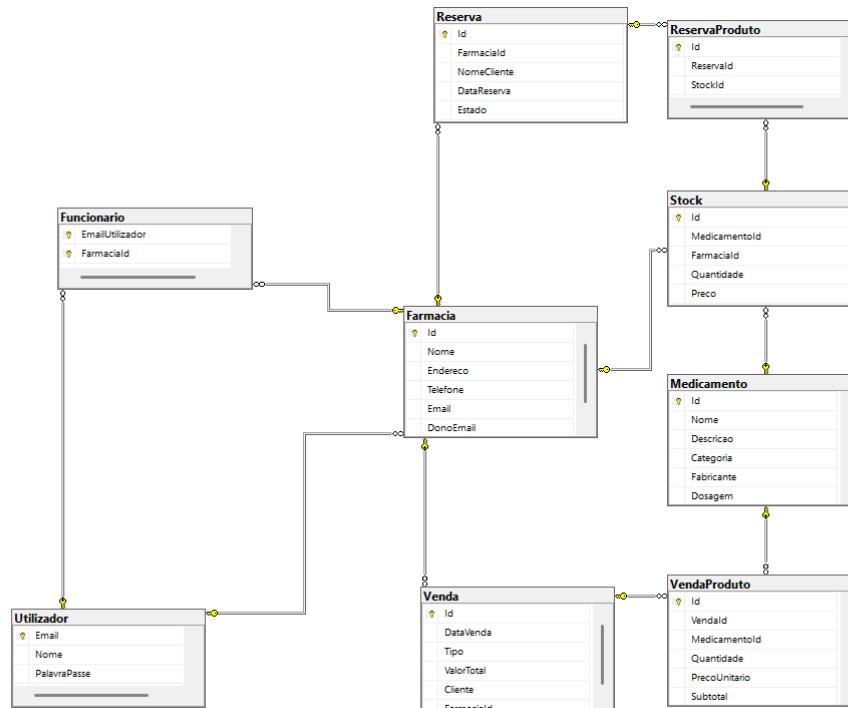
## Requisitos do Sistema

Requisitos de Software:

- Microsoft Windows 10 ou superior
- .NET Framework 4.7.2 ou superior
- Visual Studio 2019 ou superior
- SQL Server (local ou remoto)
- Entity Framework 6
- Pacotes NuGet:
  - MaterialSkin.2
  - LiveCharts
  - Newtonsoft.Json

## Base de Dados

- Criada com SQL Server.
- Ligação feita com Entity Framework 6.
- Tabelas principais (funcionalidade central):
  - **Utilizador** – guarda dados dos utilizadores da plataforma (ex: nome, email, palavra-passe, imagem).
  - **Funcionario** – representa os funcionários associados a uma farmácia, com a respetiva categoria.
  - **Reserva** – regista reservas de medicamentos feitas pelos clientes.
  - **Medicamento** – contém informações sobre os medicamentos disponíveis.
  - **Farmacia** – representa farmácias registadas na plataforma.
  - **Stock** – associa medicamentos às farmácias e indica a quantidade e preço de cada um.
  - **ReservaProduto** – relaciona reservas com os produtos/medicamentos reservados.
  - **Venda** – regista vendas (normais ou por encomenda) feitas pelas farmácias.
  - **VendaProduto** – detalha os medicamentos vendidos em cada venda (quantidade, preço, subtotal).



# Principais classes e Funções

## Entity Framework (Acesso a Dados)

Todas as classes relacionadas com dados provêm do modelo Entities, gerado automaticamente pelo Entity Framework a partir da base de dados MyHealth.

### **Tabela: Utilizador**

Representa os utilizadores da aplicação.

- Campos:
  - - Email (chave primária)
  - - Nome
  - - PalavraPasse
  - - Imagem

### **Tabela: Farmacia**

Representa as farmácias registadas.

- Campos:
  - - Id (chave primária)
  - - Nome
  - - Endereco
  - - Telefone
  - - Email
  - - DonoEmail (chave estrangeira para Utilizador.Email)

### **Tabela: Funcionario**

Representa os funcionários de uma farmácia.

- Campos:
  - - Chave primária composta: EmailUtilizador, Farmaciald
  - - Categoria

### **Tabela: Medicamento**

Contém os dados dos medicamentos disponíveis.

- Campos:

- - Id (chave primária)
- - Nome
- - Descricao
- - Categoria
- - Fabricante
- - Dosagem

### **Tabela: Stock**

Representa o stock de medicamentos em cada farmácia.

- Campos:

- - Id (chave primária)
- - Medicamentoid (chave estrangeira para Medicamento.Id)
- - Farmacicaid (chave estrangeira para Farmacia.Id)
- - Quantidade
- - Preco

### **Tabela: Reserva**

Representa as reservas feitas por clientes em farmácias.

- Campos:

- - Id (chave primária)
- - Farmacicaid (chave estrangeira para Farmacia.Id)
- - NomeCliente
- - DataReserva
- - Estado

### **Tabela: ReservaProduto**

Lista os medicamentos incluídos em cada reserva.

- Campos:

- - Id (chave primária)
- - Reservaid (chave estrangeira para Reserva.Id)
- - StockId (chave estrangeira para Stock.Id)
- - Quantidade

## Tabela: Venda

Regista as vendas efetuadas, com ou sem reserva.

- Campos:

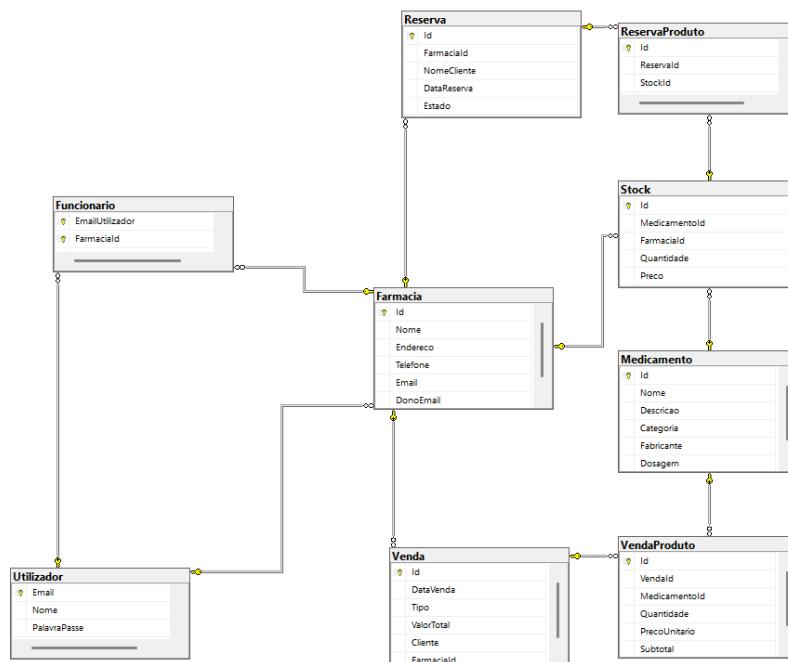
- - Id (chave primária)
- - DataVenda
- - Tipo ('encomenda' ou 'normal')
- - ValorTotal
- - Cliente
- - FarmaciaId (chave estrangeira para Farmacia.Id)

## Tabela: VendaProduto

Lista os medicamentos vendidos em cada venda.

- Campos:

- - Id (chave primária)
- - Vendaid (chave estrangeira para Venda.Id)
- - Medicamentoid (chave estrangeira para Medicamento.Id)
- - Quantidade
- - PrecoUnitario
- - Subtotal



## Classe: Entities

- Representa o contexto da base de dados gerado pelo Entity Framework.
- Permite aceder às tabelas da base de dados através de propriedades do tipo DbSet<T>, como:
  - - DbSet<Utilizador> Utilizador
  - - DbSet<Farmacia> Farmacia
  - - DbSet<Funcionario> Funcionario
  - - DbSet<Medicamento> Medicamento
  - - DbSet<Stock> Stock
  - - DbSet<Reserva> Reserva
  - - DbSet<ReservaProduto> ReservaProduto
  - - DbSet<Venda> Venda
  - - DbSet<VendaProduto> VendaProduto

## ConfiguracoesApp

Representa as preferências/configurações do utilizador (tema, idioma, tamanho da fonte, notificações, etc.).

### Propriedades:

- ModoEscuro (*bool*)
- Idioma (*string*)
- TamanhoFonte (*string*)
- NotificacoesAtivas (*bool*)
- AtualizacoesAutomaticas (*bool*)
- ManterSessaoIniciada (*bool*)
- UtilizadorAtual (*string*)

```
namespace ProjetoFinal
{
    26 referências
    internal class Theme
    {
        9 referências
        public static bool TemaAtual => ConfigManager.Configuracoes?.ModoEscuro ?? false;

        17 referências
        public static void AplicarTema(MaterialForm form, bool modoEscuro)
        {
            var materialSkinManager = MaterialSkinManager.Instance;
            materialSkinManager.AddFormToManage(form);
            materialSkinManager.Theme = modoEscuro ? MaterialSkinManager.Themes.DARK : MaterialSkinManager.Themes.LIGHT;

            materialSkinManager.ColorScheme = new ColorScheme(
                Primary.Blue800, Primary.Blue900, Primary.Blue800,
                Accent.Blue200, TextShade.WHITE);
        }
    }
}
```

## ConfigManager

Classe estática responsável por carregar e guardar as configurações da aplicação no ficheiro config.json.

### Funções principais:

- **Carregar()**  
Carrega as configurações do ficheiro JSON ou cria um ficheiro com valores por defeito.
- **Guardar()**  
Guarda as configurações atuais no ficheiro config.json.
- **Load()**  
Lê diretamente as configurações, se existirem.

```
 1 referência
public class ConfiguracoesApp
{
    //referências
    public bool ModoEscuro { get; set; } = false;
    //referências
    public string Idioma { get; set; } = "Português";
    //referências
    public string TamanhoFonte { get; set; } = "Média";
    //referências
    public bool NotificacoesAtivas { get; set; } = true;
    //referências
    public bool AtualizacoesAutomaticas { get; set; } = true;
    //referências
    public bool ManterSessaoIniciada { get; set; } = false;
    //referências
    public string UtilizadorAtual { get; set; } = string.Empty;
}

34 referências
public static class ConfigManager
{
    private static readonly string CaminhoFicheiro = "config.json";

    30 referências
    public static ConfiguracoesApp Configuracoes { get; private set; }

    // Carrega definições ou cria padrão
    1 referência
    public static void Carregar()
    {
        if (File.Exists(CaminhoFicheiro))
        {
            string json = File.ReadAllText(CaminhoFicheiro);
            Configuracoes = JsonConvert.DeserializeObject<ConfiguracoesApp>(json);
        }
        else
        {
            Configuracoes = new ConfiguracoesApp();
            Guardar(); // cria o ficheiro com defaults
        }
    }

    6 referências
    public static void Guardar()
    {
        string json = JsonConvert.SerializeObject(Configuracoes, Formatting.Indented);
        File.WriteAllText(CaminhoFicheiro, json);
    }

    1 referência
    public static ConfiguracoesApp Load()
    {
        if (!File.Exists(CaminhoFicheiro))
        {
            return new ConfiguracoesApp();
        }

        string json = File.ReadAllText(CaminhoFicheiro);
        return JsonConvert.DeserializeObject<ConfiguracoesApp>(json);
    }
}
```

## Contas

Classe responsável pela gestão de conta e autenticação dos utilizadores.

### Campos Estáticos:

- Email (*string*) – Email do utilizador autenticado.
- Farmacia (*int*) – ID da farmácia associada ao funcionário (se aplicável).

### Funções principais:

- Login(*string email, string password*)  
Autentica um utilizador ao verificar as credenciais na base de dados.
- CriarConta(*string email, string password*)  
Cria um novo utilizador, após validar que o email ainda não existe.
- ValidarEmail(*string email*)  
Valida o formato do email (trim, lowercase e expressão regular).

```
internal class Contas
{
    public static string Email = "";
    public static int Farmacia = -1;
    private static bool valido;

    public static bool Login(string email, string password)
    {
        using (var context = new Entities())
        {
            var utilizador = context.Utilizador.FirstOrDefault(u => u.Email == email && u.Password == password);
            if (utilizador != null)
            {
                Email = email;
                MessageBox.Show("Sucesso ao Iniciar Sessão!", "Sucesso", MessageBoxButtons.OK, MessageBoxIcon.Information);
                return true;
            }
            else
            {
                MessageBox.Show("Credenciais inválidas!", "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
        }
    }

    internal static bool CriarConta(string email, string password)
    {
        using (var context = new Entities())
        {
            bool existe = context.Utilizador.Any(u => u.Email == email);

            if (existe)
            {
                MessageBox.Show("Já existe uma conta com esse e-mail.", "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }

            var random = new Random();
            int numeroAleatorio = random.Next(1000, 9999);

            var novoUtilizador = new Utilizador
            {
                Email = email,
                Password = password,
                Nome = "user" + numeroAleatorio
            };

            context.Utilizador.Add(novoUtilizador);
            context.SaveChanges();

            MessageBox.Show("Conta registada com sucesso!", "Sucesso", MessageBoxButtons.OK, MessageBoxIcon.Information);
            return true;
        }
    }

    internal static bool ValidarEmail(string email)
    {
        if (email != email.Trim() || email != email.ToLower())
        {
            MessageBox.Show("O email deve ter o formato correto!", "Aviso", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
        string pattern = @"^[\w\.-]+@[^\w\.-]+\.[^\w\.-]+$";
        if (!Regex.IsMatch(email, pattern))
        {
            MessageBox.Show("O email deve ter o formato correto (ex: exemplo@dominio.com)!", "Aviso", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
        return true;
    }
}
```

## Theme

Classe responsável pela aplicação dinâmica do tema claro ou escuro através da biblioteca MaterialSkin.

### Funções principais:

- TemaAtual  
Retorna o tema atual definido nas configurações.
- AplicarTema(MaterialForm form, bool modoEscuro)  
Aplica dinamicamente o tema selecionado ao formulário.

```
26 referências
internal class Theme
{
    9 referências
    public static bool TemaAtual => ConfigManager.Configuracoes?.ModoEscuro ?? false;

    17 referências
    public static void AplicarTema(MaterialForm form, bool modoEscuro)
    {
        var materialSkinManager = MaterialSkinManager.Instance;
        materialSkinManager.AddFormToManage(form);
        materialSkinManager.Theme = modoEscuro ? MaterialSkinManager.Themes.DARK : MaterialSkinManager.Themes.LIGHT;

        materialSkinManager.ColorScheme = new ColorScheme(
            Primary.Blue800, Primary.Blue900, Primary.Blue800,
            Accent.Blue200, TextShade.WHITE);
    }
}
```

## Funcionalidades Técnicas

### a) Gestão de Funcionários

- Listagem, adição e pesquisa.
- Integração com a tabela de Utilizador.

### b) Gráficos com LiveCharts

- Dashboard com gráfico de barras e linha.
- Ligado a dados reais da base de dados.

### c) Sistema de Preferências

- Guardar tema, idioma e fonte em AppSettings.json.
- Carregado automaticamente no arranque.

# Como Instalar?

## 1. Criar a Base de Dados

- Abra o SQL Server Management Studio.
- Crie uma base de dados chamada MyHealth.
- Corra o script SQL fornecido para criar as tabelas.

## 2. Definir a Connection String

- No ficheiro App.config, dentro de <connectionStrings>, coloque:

```
<add name="Entities" connectionString="Data
source=(localdb)\MSSQLLocalDB;Initial Catalog=MyHealth;Integrated
Security=True;" providerName="System.Data.SqlClient" />
```

- Alternativamente, use os dados do seu servidor SQL.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <entityFramework>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
  <connectionStrings>
    <add name="Entities"
      connectionString="metadata=res://*/MyHealthModel.csdl|res://*/MyHealthModel.ssdl|res://*/MyHealthModel.msl;provider=System.Data.SqlClient;
      providerName="System.Data.EntityClient" /></connectionStrings>
  </configuration>
```

## 3. Atualizar o Modelo

- Botão direito no ficheiro .edmx → **"Update Model from Database"**.

## 4. Executar a Aplicação

- Carregue em **F5** ou no botão **Start** no Visual Studio.