

# Gestão de uma Unidade Hospitalar

Gonçalo Costa, 26024  
Daniela Pereira, 25988

Docente: Óscar Ribeiro

December 28, 2024

## Contents

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Introdução ao Problema</b>	<b>3</b>
<b>3</b>	<b>Metodologia</b>	<b>3</b>
<b>4</b>	<b>Base de Dados</b>	<b>4</b>
4.1	Diagrama ER . . . . .	4
4.2	SQL . . . . .	5
4.3	Azure . . . . .	6
4.3.1	Criação do Azure . . . . .	6
<b>5</b>	<b>SOAP</b>	<b>6</b>
5.1	ASMX . . . . .	7
5.1.1	Criação dos Models . . . . .	7
5.1.2	Serviço HospitalService . . . . .	9
5.1.3	Execução Serviço SOAP - ASMX - Testes . . . . .	12
5.2	WCF . . . . .	12
5.2.1	Instalação <i>SoapUI</i> . . . . .	12
5.2.2	Criação dos Models . . . . .	13
5.2.3	Interface IHospitalService . . . . .	14
5.2.4	Serviço HospitalService . . . . .	15
5.2.5	Execução Serviço SOAP - WCF - Testes . . . . .	18
<b>6</b>	<b>RESTFUL Services</b>	<b>19</b>
6.1	Criação de Conexões . . . . .	19
6.2	Criação de Controllers . . . . .	19
6.3	Métodos em Controllers . . . . .	20
6.4	Logins e Token JWT . . . . .	21
6.5	Swagger . . . . .	23
6.6	Forms: Interação Cliente . . . . .	25
<b>7</b>	<b>Conclusão</b>	<b>28</b>

## List of Figures

1	Diagrama Entidade Relação . . . . .	4
2	Query SQL . . . . .	5
3	Base de Dados Criada . . . . .	5
4	Base de Dados Remota Criada . . . . .	6
5	Models para o ASMX . . . . .	7
6	Model Utente no ASMX . . . . .	8
7	Hospital Service no ASMX - Connection String . . . . .	9
8	Hospital Service no ASMX - Create . . . . .	9
9	Hospital Service no ASMX - UPDATE . . . . .	10
10	Hospital Service no ASMX - Delete . . . . .	10
11	Hospital Service no ASMX - GetAll . . . . .	11
12	Testes serviço ASMX . . . . .	12
13	Model Médico no WCF . . . . .	13
14	IhospitalService no WCF . . . . .	14
15	IhospitalService no WCF . . . . .	15
16	Hospital Service no WCF - Create . . . . .	15
17	Hospital Service no WCF - UPDATE . . . . .	16
18	Hospital Service no WCF - Delete . . . . .	16
19	Hospital Service no WCF - GetBY . . . . .	17
20	Testes serviço WCF . . . . .	18
21	Referencias aos Servicos SOAP . . . . .	19
22	Lista de Controllers . . . . .	19
23	Lista de Controllers . . . . .	20
24	Função de autenticação . . . . .	21
25	Obter Token . . . . .	22
26	Swagger métodos . . . . .	23
27	Método obter funcionário . . . . .	24
28	Página de Login . . . . .	25
29	Página Inicial . . . . .	26
30	Página adicionar funcionário . . . . .	27
31	GetAllFuncionários . . . . .	27

## 1 Introdução

Neste documento, será apresentada a arquitetura para o segundo trabalho prático de Integração de Sistemas de Informação. Para a unidade curricular de Integração de Sistemas de Informação, foca-se a exploração e desenvolvimento de processos de interoperabilidade entre sistemas, assentes em serviços web. Pretende-se que seja desenvolvida uma biblioteca de novos serviços (*SOAP*, *RESTful*), complementada com a reutilização de serviços externos existentes.

## 2 Introdução ao Problema

A gestão hospitalar enfrenta desafios complexos que envolvem a gestão de funcionários, a qualidade do atendimento ao paciente e a sustentabilidade das instituições de saúde. Com o aumento da demanda por serviços de saúde, aliado à escassez de recursos e à necessidade de inovação tecnológica, os gestores hospitalares precisam desenvolver estratégias eficazes para melhorar a operação dos hospitais. A gestão eficiente dos processos internos, desde o agendamento de consultas até a administração de funcionários, é fundamental para garantir a prestação de cuidados de saúde de qualidade. Este trabalho busca analisar as práticas atuais de gestão hospitalar, identificar os principais problemas enfrentados pelas instituições de saúde e sugerir soluções que possam contribuir para a melhoria dos serviços prestados aos pacientes e à comunidade em geral.

## 3 Metodologia

Para este trabalho, foi necessário utilizar diferentes ferramentas, como o *Azure*, para migrações e acesso à base de dados em *Cloud*, o *Visual Paradigm* para a construção do diagrama entidade relação, *SQL Server* para a Base de Dados utilizada neste projeto. Todo o código foi desenvolvido no *Visual Studio*.

## 4 Base de Dados

### 4.1 Diagrama ER

O Diagrama Entidade Relação (**ER**) é uma representação gráfica usada para representar a estrutura de uma base de dados, que apresenta as entidades (objetos ou elementos de interesse), os atributos das entidades, e os relacionamentos entre elas.

Na figura 1 pode-se observar a estrutura da base dados da aplicação que vai ser desenvolvida para este projeto. É representada pelas entidades de forma a armazenar os dados dos Funcionários, Utentes, Consultas e Médicos.

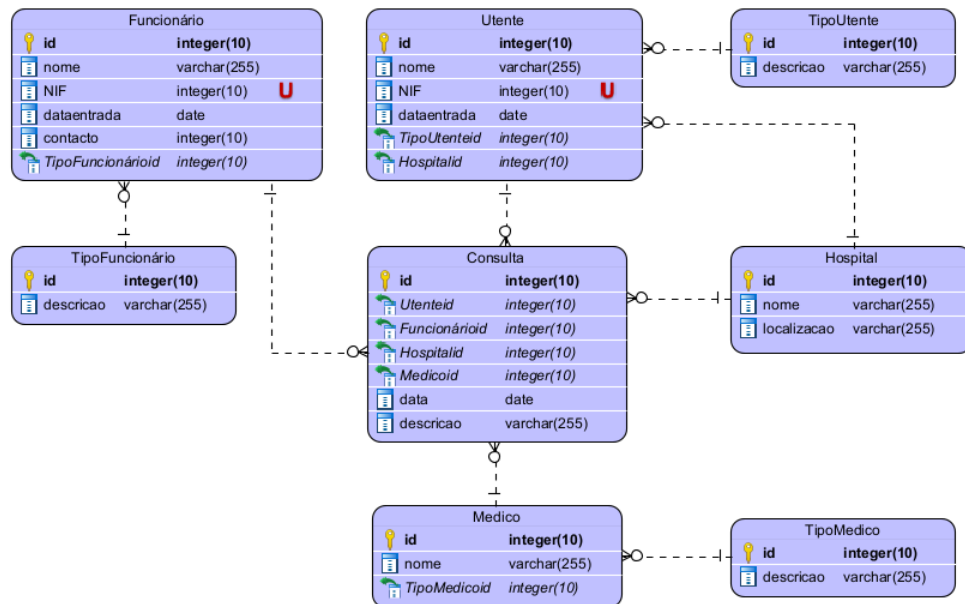


Figure 1: Diagrama Entidade Relação

## 4.2 SQL

Após criado o Diagrama Entidade Relação, foi gerado o código SQL para criar a base de dados no *SQL Server*.

Na figura 2, podemos ver a *Query SQL* utilizada para a criação do base de dados no *SQL Server*. Após isto, a Base de Dados foi criada.

```
CREATE TABLE Utente (id int IDENTITY NOT NULL, nome varchar(255) NOT NULL, NIF int NOT NULL UNIQUE, dataentrada datetime NOT NULL,
TipoUtenteid int NOT NULL, Hospitalid int NOT NULL, PRIMARY KEY (id));
CREATE TABLE TipoUtente (id int IDENTITY NOT NULL, descricao varchar(255) NOT NULL, PRIMARY KEY (id));
CREATE TABLE Hospital (id int IDENTITY NOT NULL, nome varchar(255) NOT NULL, localizacao varchar(255) NOT NULL, PRIMARY KEY (id));
CREATE TABLE Funcionário (id int IDENTITY NOT NULL, nome varchar(255) NOT NULL, NIF int NOT NULL UNIQUE, dataentrada datetime NOT NULL,
contacto int NOT NULL, password varchar(255) NOT NULL UNIQUE, TipoFuncionárioid int NOT NULL, JWT varchar(255) NOT NULL, PRIMARY KEY (id));
CREATE TABLE TipoFuncionário (id int IDENTITY NOT NULL, descricao varchar(255) NOT NULL, PRIMARY KEY (id));
CREATE TABLE Consulta (id int IDENTITY NOT NULL, Utenteid int NOT NULL, Funcionárioid int NOT NULL, Hospitalid int NOT NULL,
Medicoid int NOT NULL, data datetime NOT NULL, hora datetime NOT NULL, descricao varchar(255) NOT NULL, PRIMARY KEY (id));
CREATE TABLE Medico (id int IDENTITY NOT NULL, nome varchar(255) NOT NULL, TipoMedicoid int NOT NULL, PRIMARY KEY (id));
CREATE TABLE TipoMedico (id int IDENTITY NOT NULL, descricao varchar(255) NOT NULL, PRIMARY KEY (id));
ALTER TABLE Utente ADD CONSTRAINT FKUtente933320 FOREIGN KEY (Hospitalid) REFERENCES Hospital (id);
ALTER TABLE Funcionário ADD CONSTRAINT FKFuncionári804187 FOREIGN KEY (TipoFuncionárioid) REFERENCES TipoFuncionário (id);
ALTER TABLE Consulta ADD CONSTRAINT FKConsulta444710 FOREIGN KEY (Utenteid) REFERENCES Utente (id);
ALTER TABLE Consulta ADD CONSTRAINT FKConsulta219789 FOREIGN KEY (Funcionárioid) REFERENCES Funcionário (id);
ALTER TABLE Consulta ADD CONSTRAINT FKConsulta917040 FOREIGN KEY (Hospitalid) REFERENCES Hospital (id);
ALTER TABLE Consulta ADD CONSTRAINT FKConsulta626270 FOREIGN KEY (Medicoid) REFERENCES Medico (id);
ALTER TABLE Medico ADD CONSTRAINT FKMedico888667 FOREIGN KEY (TipoMedicoid) REFERENCES TipoMedico (id);
```

Figure 2: Query SQL

Na figura 3, podemos observar que a Base de Dados foi criada com sucesso.

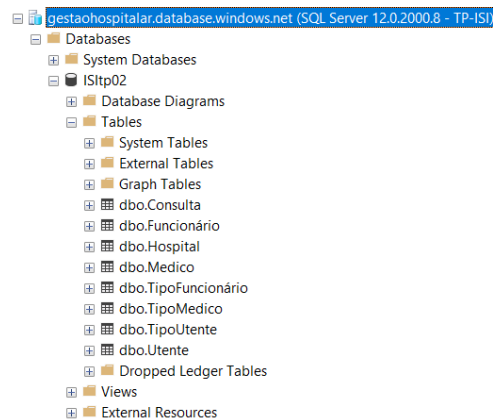


Figure 3: Base de Dados Criada

## 4.3 Azure

### 4.3.1 Criação do Azure

Utilizar uma base de dados no Azure oferece benefícios significativos, como alta disponibilidade, escalabilidade automática, gestão simplificada com automação de backups e atualizações, além de robusta segurança com criptografia e conformidade com diversas regulamentações. A plataforma permite integração fácil com outros serviços, neste caso com SQL e proporciona uma gestão de custos flexível. Com desempenho otimizado, ferramentas de monitoramento e recuperação de desastres, o Azure proporciona uma solução confiável, segura e económica para empresas de todos os tamanhos.



Nome	Tipo	Última Visualização
 gestaohospitalar	SQL Server	há 2 horas
 ISItP02	Base de dados SQL	há uma semana
 ISITP02	Grupo de recursos	há uma semana
 gestaohospitalar	Serviço de Aplicações	há 2 semanas

Figure 4: Base de Dados Remota Criada

## 5 SOAP

Foram explorados ambos os métodos SOAP para a realização deste projeto, neste caso, o modelo ASMX e o modelo WCF.

Tendo em conta o tema escolhido para este trabalho, neste caso, a gestão da Unidade de Saúde, foram divididas as classes entre ambos os métodos. As entidades internas da Unidade, como funcionários, utentes e suas consultas no ASMX e no WCF médicos e hospitais.

## 5.1 ASMX

### 5.1.1 Criação dos Models

Para os Models no ASMX, foi criada a sua respetiva pasta com todos os models deste sistema.

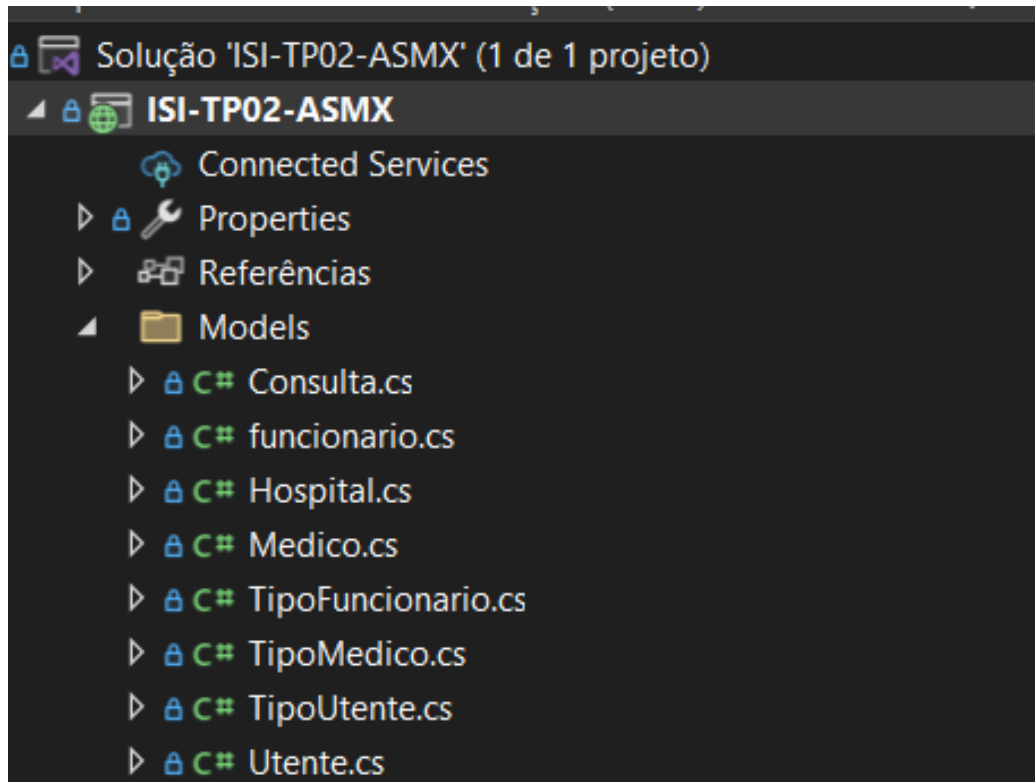


Figure 5: Models para o ASMX



Abaixo na figura 6, temos um exemplo de um Model, neste caso, o Model de Utente.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ISITP02.Models
{
    7 referências
    public class Utente
    {
        2 referências
        public int Id { get; set; }
        2 referências
        public string Nome { get; set; }
        2 referências
        public int NIF { get; set; }
        2 referências
        public DateTime DataEntrada { get; set; }
        2 referências
        public int TipoUtenteId { get; set; }
        2 referências
        public int HospitalId { get; set; }
    }
}
```

Figure 6: Model Utente no ASMX

### 5.1.2 Serviço HospitalService

Abaixo na figura 7, podemos observar a inicialização do Hospital Service. Aqui temos a conexão à base de dados remota Azure.

```
namespace ISI_TP02_ASMX
{
    /// <summary>
    /// Descrição resumida de HospitalService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // Para permitir que esse serviço da web seja chamado a partir do script, usando ASP.NET AJAX, remova os comentários da linha a seguir.
    // [System.Web.Script.Services.ScriptService]
    1 referência
    public class HospitalService : System.Web.Services.WebService
    {
        private readonly string connectionString = "Server=tcp:gestaohospitalar.database.windows.net,1433;Initial Catalog=ISItp02;Persist Security Info=False;User ID=TP02;Password=TP02;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;";
        private SqlConnection db = new SqlConnection();
        0 referências
        public HospitalService()
        {
            db.ConnectionString = connectionString;
        }
        10 referências
        private SqlConnection GetConnection()
        {
            return new SqlConnection(connectionString);
        }
    }
}
```

Figure 7: Hospital Service no ASMX - Connection String

No ASMX, foram desenvolvidos métodos CRUD (Create, Read, Update e Delete) para as entidades deste Service ASMX.

Abaixo na figura 8, está apresentado o método 'Create' de um Funcionário no Hospital Service no ASMX.

```
[WebMethod]
0 referências
public bool CreateFuncionario(Funcionario funcionario)
{
    string query = "INSERT INTO Funcionario (Nome, NIF, DataEntrada, Contacto, Password, TipoFuncionárioId) " +
        "VALUES (@Nome, @NIF, @DataEntrada, @Contacto, @Password, @TipoFuncionárioId)";

    try
    {
        using (var connection = GetConnection())
        {
            connection.Open();

            using (var command = new SqlCommand(query, connection))
            {
                command.Parameters.Add("@Nome", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Nome;
                command.Parameters.Add("@NIF", System.Data.SqlDbType.Int).Value = funcionario.NIF;
                command.Parameters.Add("@DataEntrada", System.Data.SqlDbType.DateTime).Value = funcionario.DataEntrada;
                command.Parameters.Add("@Contacto", System.Data.SqlDbType.Int).Value = funcionario.Contacto;
                command.Parameters.Add("@Password", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Password;
                command.Parameters.Add("@TipoFuncionárioId", System.Data.SqlDbType.Int).Value = funcionario.TipoFuncionárioId;
                int rowsAffected = command.ExecuteNonQuery();
                return rowsAffected > 0;
            }
        }
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error creating Funcionário.", ex);
    }
}
```

Figure 8: Hospital Service no ASMX - Create

Abaixo na figura 9, está apresentado o método 'UPDATE' de um Funcionário no Hospital Service no ASMX.

```
[WebMethod]
0 referências
public bool UpdateFuncionario(Funcionario funcionario)
{
    string query = "UPDATE Funcionário SET Nome = @Nome, NIF = @NIF, DataEntrada = @DataEntrada, " +
        "Contato = @Contato, Password = @Password, TipoFuncionárioId = @TipoFuncionárioId WHERE Id = @Id";

    try
    {
        using (var connection = GetConnection())
        {
            connection.Open();

            using (var command = new SqlCommand(query, connection))
            {
                command.Parameters.Add("@Id", System.Data.SqlDbType.Int).Value = funcionario.Id;
                command.Parameters.Add("@Nome", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Nome;
                command.Parameters.Add("@NIF", System.Data.SqlDbType.Int).Value = funcionario.NIF;
                command.Parameters.Add("@DataEntrada", System.Data.SqlDbType.DateTime).Value = funcionario.DataEntrada;
                command.Parameters.Add("@Contato", System.Data.SqlDbType.Int).Value = funcionario.Contato;
                command.Parameters.Add("@Password", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Password;
                command.Parameters.Add("@TipoFuncionárioId", System.Data.SqlDbType.Int).Value = funcionario.TipoFuncionárioId;
                command.ExecuteNonQuery();
            }
        }
        return true;
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error updating Funcionário.", ex);
    }
}
```

Figure 9: Hospital Service no ASMX - UPDATE

Abaixo, na figura 10, está apresentado o método 'DELETE' de um Funcionário no Hospital Service no ASMX.

```
[WebMethod]
0 referências
public bool DeleteFuncionario(int id)
{
    string query = "DELETE FROM Funcionário WHERE Id = @Id";

    try
    {
        using (var connection = GetConnection())
        {
            connection.Open();

            using (var command = new SqlCommand(query, connection))
            {
                command.Parameters.Add("@Id", System.Data.SqlDbType.Int).Value = id;
                int rowsAffected = command.ExecuteNonQuery();
                return rowsAffected > 0;
            }
        }
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error deleting Funcionário.", ex);
    }
}
```

Figure 10: Hospital Service no ASMX - Delete

Abaixo na figura 11, está apresentado o método 'GETALL' de um Funcionário no Hospital Service no ASMX.

```
[WebMethod]
0 referências
public List<Utente> GetAllUtentes()
{
    var utentes = new List<Utente>();
    string query = "SELECT Id, Nome, NIF, DataEntrada, TipoUtenteId, HospitalId FROM Utente";

    try
    {
        using (var connection = GetConnection())
        {
            connection.Open();

            using (var command = new SqlCommand(query, connection))
            using (var reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    utentes.Add(new Utente
                    {
                        Id = reader.GetInt32(0),
                        Nome = reader.GetString(1),
                        NIF = reader.GetInt32(2),
                        DataEntrada = reader.GetDateTime(3),
                        TipoUtenteId = reader.GetInt32(4),
                        HospitalId = reader.GetInt32(5)
                    });
                }
            }
        }
        return utentes;
    }
    catch (SqlException sqlEx)
    {
        throw new ApplicationException($"Database error: {sqlEx.Message}", sqlEx);
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error retrieving Utente data.", ex);
    }
}
```

Figure 11: Hospital Service no ASMX - GetALL

### 5.1.3 Execução Serviço SOAP - ASMX - Testes

Para a realização de testes, foi utilizado o SOAPUI. Todos os métodos desenvolvidos acima foram testados, isto é muito importante para verificar o seu correto funcionamento.

Abaixo na figura 12, podemos verificar esses testes e comprovar que foram realizados com sucesso.

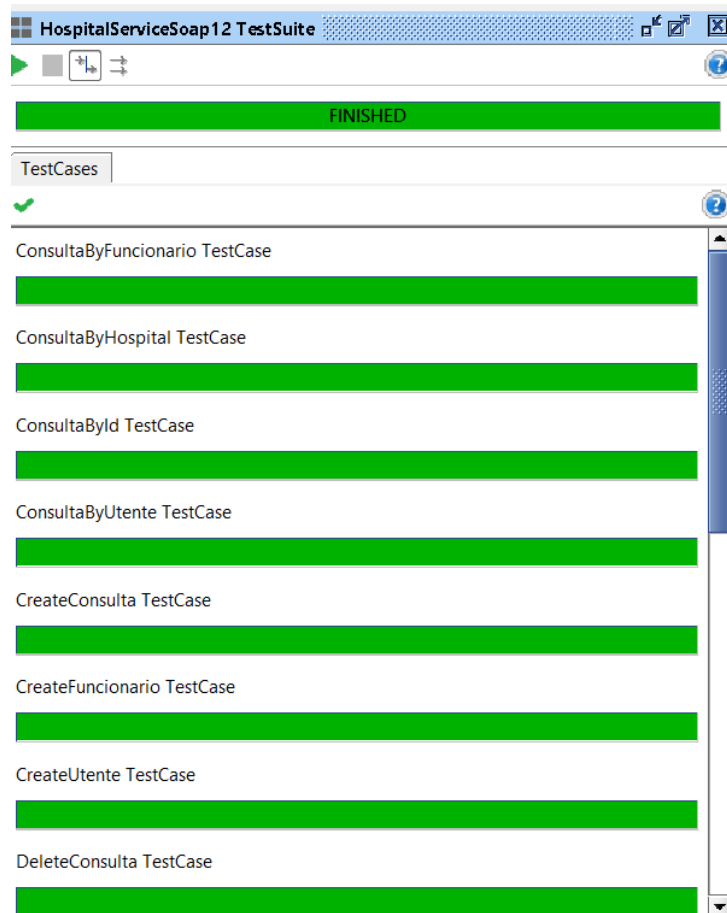


Figure 12: Testes serviço ASMX

## 5.2 WCF

### 5.2.1 Instalação *SoapUI*

Inicialmente foi instalado o SoapUI para verificar o correto funcionamento deste serviço.

### 5.2.2 Criação dos Models

Para este serviço, assim como no ASMX, foram criadas inicialmente os models para as entidades.

Abaixo na figura 13, podemos observar o model para a entidade Médico no serviço WCF.

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ISITP02.Models
{
    9 referências
    public class Medico
    {
        2 referências
        public int Id { get; set; }
        2 referências
        public string Nome { get; set; }
        2 referências
        public TipoMedico TipoMedico { get; set; }

        0 referências
        public List<Consulta> Consultas { get; set; }
    }
}
```

Figure 13: Model Médico no WCF

### 5.2.3 Interface IHospitalService

Na interface IHospitalService, foram colocados todos os métodos que vão ser desenvolvidos neste sistema, todas com [OperationContract].

```
using ISI7P02.Models;
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.Threading.Tasks;

[ServiceContract]
1 referência
public interface IHospitalService
{
    [OperationContract]
    1 referência
    bool CreateMedico(string nome, int tipoMedicoId);

    [OperationContract]
    1 referência
    List<Medico> GetAllMedicos();

    [OperationContract]
    1 referência
    Medico GetMedicoById(int id);

    [OperationContract]
    1 referência
    bool UpdateMedico(int id, string nome, int tipoMedicoId);

    [OperationContract]
    1 referência
    bool DeleteMedico(int id);

    [OperationContract]
    1 referência
    bool CreateTipoMedico(string descricao);

    [OperationContract]
    1 referência
    bool DeleteTipoMedico(string descricao);

    [OperationContract]
    1 referência
    bool CreateHospital(string nome, string localizacao);

    [OperationContract]
    1 referência
    List<Hospital> GETALLHOSPITAIS();

    [OperationContract]
    1 referência
    Hospital GetHospitalByLoc(string localizacao);

    [OperationContract]
    1 referência
    Task<Funcionario> AutenticarUtilizador(Funcionario f);
}
```

Figure 14: IhospitalService no WCF

### 5.2.4 Serviço HospitalService

Primeiro, inicializou-se o HospitalService, referenciando-se à sua Interface e com a comunicação à base de dados remota.

```
using ISI/TP02.Models;
using System;
using System.Activities.Statements;
using System.Collections.Generic;
using System.Data.Common;
using System.Data.SqlClient;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.ServiceModel;
using System.Threading.Tasks;
using System.Web.Services;

[WebService]
public class HospitalService : IHospitalService
{
    private string connectionString = "Server=tp:gestaohospitalar.database.windows.net,1433;Initial Catalog=ISItp02;Persist Security Info=False;User Id=TP-ISI;Password=Goncalo18;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;";

    [WebMethod]
    public HospitalService()
    {
    }
}
```

Figure 15: IHospitalService no WCF

Foram desenvolvidos vários métodos para as entidades deste serviço, mais especificamente os métodos CRUD (Create, Read, Update, Delete). Abaixo, podemos ver exemplos desses métodos.

Abaixo na figura 16, está apresentado o método de Criar um Médico no Hospital Service no WCF.

```
[WebMethod]
public bool CreateFuncionario(Funcionario funcionario)
{
    string query = "INSERT INTO Funcionario (Nome, NIF, DataEntrada, Contacto, Password, TipoFuncionarioId) " +
        "VALUES (@Nome, @NIF, @DataEntrada, @Contacto, @Password, @TipoFuncionarioId)";

    try
    {
        using (var connection = GetConnection())
        {
            connection.Open();

            using (var command = new SqlCommand(query, connection))
            {
                command.Parameters.Add("@Nome", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Nome;
                command.Parameters.Add("@NIF", System.Data.SqlDbType.Int).Value = funcionario.NIF;
                command.Parameters.Add("@DataEntrada", System.Data.SqlDbType.DateTime).Value = funcionario.DataEntrada;
                command.Parameters.Add("@Contacto", System.Data.SqlDbType.Int).Value = funcionario.Contacto;
                command.Parameters.Add("@Password", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Password;
                command.Parameters.Add("@TipoFuncionarioId", System.Data.SqlDbType.Int).Value = funcionario.TipoFuncionarioId;
                int rowsAffected = command.ExecuteNonQuery();
                return rowsAffected > 0;
            }
        }
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error creating Funcionario.", ex);
    }
}
```

Figure 16: Hospital Service no WCF - Create



Abaixo na figura 17, está apresentado o método 'UPDATE' de um Médico no Hospital Service no WCF.

```
[WebMethod]
0 referências
public bool UpdateFuncionario(Funcionario funcionario)
{
    string query = "UPDATE Funcionário SET Nome = @Nome, NIF = @NIF, DataEntrada = @DataEntrada, " +
        "Contato = @Contato, Password = @Password, TipoFuncionárioId = @TipoFuncionárioId WHERE Id = @Id";

    try
    {
        using (var connection = GetConnection())
        {
            connection.Open();

            using (var command = new SqlCommand(query, connection))
            {
                command.Parameters.Add("@Id", System.Data.SqlDbType.Int).Value = funcionario.Id;
                command.Parameters.Add("@Nome", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Nome;
                command.Parameters.Add("@NIF", System.Data.SqlDbType.Int).Value = funcionario.NIF;
                command.Parameters.Add("@DataEntrada", System.Data.SqlDbType.DateTime).Value = funcionario.DataEntrada;
                command.Parameters.Add("@Contato", System.Data.SqlDbType.Int).Value = funcionario.Contato;
                command.Parameters.Add("@Password", System.Data.SqlDbType.NVarChar, 100).Value = funcionario.Password;
                command.Parameters.Add("@TipoFuncionárioId", System.Data.SqlDbType.Int).Value = funcionario.TipoFuncionárioId;
                command.ExecuteNonQuery();
            }
        }
        return true;
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error updating Funcionário.", ex);
    }
}
```

Figure 17: Hospital Service no WCF - UPDATE

Abaixo na figura 18, está apresentado o método 'Delete' de um Médico no Hospital Service no WCF.

```
[WebMethod]
1 referência
public bool DeleteMedico(int id)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            string query = "DELETE FROM Medico WHERE Id = @Id";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Id", id);

                int rowsAffected = command.ExecuteNonQuery();
                return rowsAffected > 0;
            }
        }
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error deleting medico", ex);
    }
}
```

Figure 18: Hospital Service no WCF - Delete

Abaixo na figura 19, está apresentado o método 'Get' de um Médico pelo seu id no Hospital Service no WCF.

```
[WebMethod]
[OperationContract]
public Medico GetMedicoById(int id)
{
    List<TipoMedico> tiposMedico = new List<TipoMedico>();

    try
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            string query = "SELECT Id, Nome, TipoMedicoId FROM Medico WHERE Id = @Id";
            string obterQuery = "SELECT id, descricao FROM TipoMedico";
            using (SqlCommand obterTipoMedico = new SqlCommand(obterQuery, connection))
            {
                using (SqlDataReader reader = obterTipoMedico.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        tiposMedico.Add(new TipoMedico
                        {
                            Id = reader.GetInt32(0),
                            Descricao = reader.GetString(1)
                        });
                    }
                }
            }

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Id", id);

                using (SqlDataReader reader = command.ExecuteReader())
                {
                    if (reader.Read())
                    {
                        int tipoMedicoId = reader.GetInt32(2);
                        TipoMedico tipoMedico = tiposMedico.FirstOrDefault(t => t.Id == tipoMedicoId);

                        return new Medico
                        {
                            Id = reader.GetInt32(0),
                            Nome = reader.GetString(1),
                            TipoMedico = tipoMedico
                        };
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error fetching medico by ID", ex);
    }
}
```

Figure 19: Hospital Service no WCF - GetBY

### 5.2.5 Execução Serviço SOAP - WCF - Testes

Para a realização de testes, foi utilizado o SOAPUI. Todos os métodos desenvolvidos acima foram testados, isto é muito importante para verificar o seu correto funcionamento.

Abaixo na figura 20, podemos verificar esses testes e comprovar que foram realizados com sucesso.

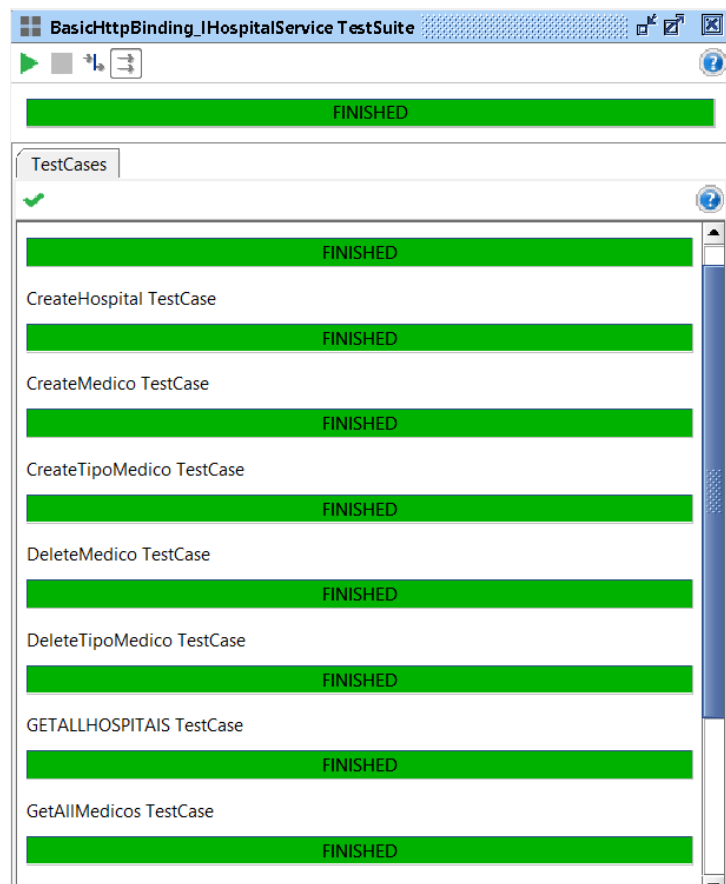


Figure 20: Testes serviço WCF

## 6 RESTFUL Services

Após o desenvolvimento e conclusão dos métodos SOAP, o próximo passo é desenvolver a API RESTFUL.

### 6.1 Criação de Conexões

Foram utilizadas como referência para o desenvolvimento desta API, ambos os métodos SOAP desenvolvidos anteriormente, como podemos comprovar pela imagem abaixo.

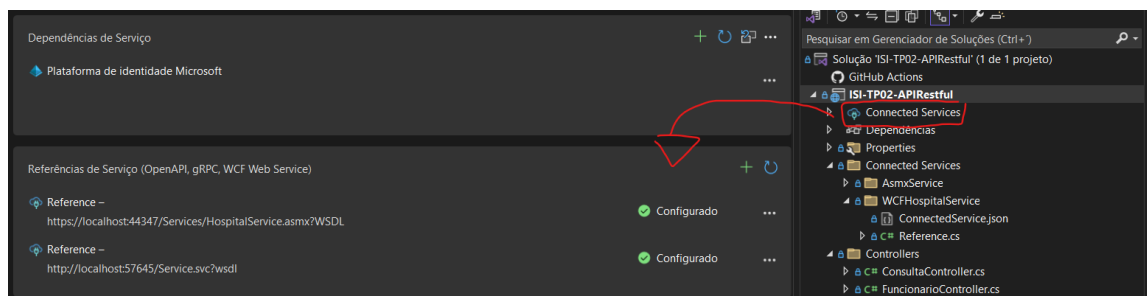


Figure 21: Referencias aos Servicos SOAP

### 6.2 Criação de Controllers

Foram criados controllers para cada entidade deste projeto. Abaixo podemos observar a lista de Controllers.

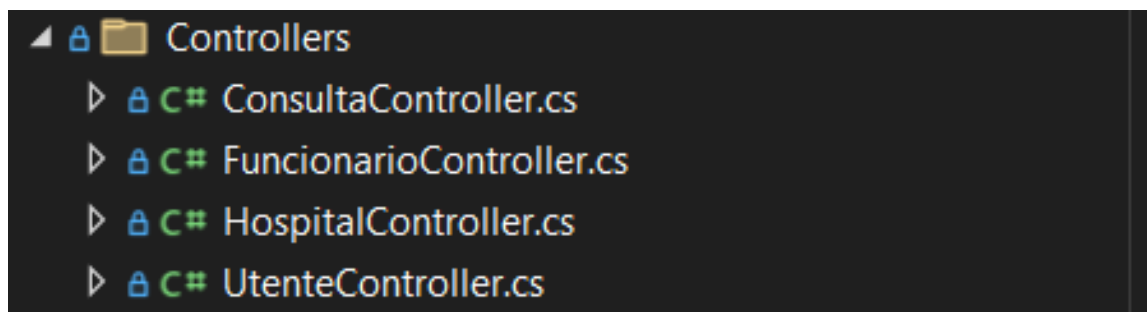


Figure 22: Lista de Controllers

### 6.3 Métodos em Controllers

Em cada controller existem métodos específicos em cada entidade, além dos CRUD, cada entidade poderá ter os seus próprios e únicos métodos.

Abaixo está apresentado o controller para Funcionário com os métodos de criar, eliminar, obter e atualizar.

```
namespace ISI_TP02_APIRestful.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class FuncionarioController : ControllerBase
    {
        public HospitalServiceSoapClient asmxClient;

        public HospitalServiceClient WCFcliente = new HospitalServiceClient();

        public FuncionarioController()
        {
            var endpointAddress = new EndpointAddress("https://localhost:44347/Services/HospitalService.asmx");
            asmxClient = new HospitalServiceSoapClient(HospitalServiceSoapClient.EndpointConfiguration.HospitalServiceSoap, endpointAddress);
        }

        [HttpGet("obter")]
        public async Task<ActionResult> Obter()
        {
            var request = await asmxClient.GetAllFuncionariosAsync();
            if (request != null)
                return Ok(request.Body.GetAllFuncionariosResult.ToList());
            return NotFound();
        }

        [HttpDelete("delete")]
        public async Task<ActionResult> Delete(int id)
        {
            var request = await asmxClient.DeleteFuncionarioAsync(id);
            if (request)
                return Ok(request);
            return BadRequest();
        }

        [HttpPost("create")]
        public async Task<ActionResult> Create(AsmxService.Funcionario funcionario)
        {
            var request = await asmxClient.CreateFuncionarioAsync(funcionario);
            if (request.Body.CreateFuncionarioResult)
                return Ok(request.Body.CreateFuncionarioResult);
            return BadRequest();
        }

        [HttpPut("update")]
        public async Task<ActionResult> Update(AsmxService.Funcionario funcionario)
        {
            var request = await asmxClient.UpdateFuncionarioAsync(funcionario);
            if (request.Body.UpdateFuncionarioResult)
                return Ok(request.Body.UpdateFuncionarioResult);
            return BadRequest();
        }
    }
}
```

Figure 23: Lista de Controllers

## 6.4 Logins e Token JWT

Para a autenticação, foi inicialmente desenvolvido um método no controller funcionário para verificar se os dados estão corretos e assim atribuir o token de acesso.

Abaixo, na figura 24, podemos observar essa função.

```
[HttpPost("RealizarLogin")]
0 referências
public async Task<IActionResult> LoginFuncionario([FromBody] Login loginRequest)
{
    obterJWT funcoesAuxiliares = new obterJWT();
    try
    {
        WCFHospitalService.Funcionario f = new WCFHospitalService.Funcionario();
        f.NIF = loginRequest.NIF;
        f.Password = loginRequest.Password;
        var userValido = await WCFcliente.AutenticarUtilizadorAsync(f);
        if (userValido == null)
        {
            return Unauthorized("Email ou password inválido.");
        }
        var token = funcoesAuxiliares.GenerateJwtToken(userValido);

        return Ok(token);
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

Figure 24: Função de autenticação

Após isso, foi criada a classe onde se obtém o token JWT, função auxiliar da apresentada acima.

```
public class obterJWT
{
    1 referência
    public string GenerateJwtToken(MCFHospitalService.Funcionario func)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.ASCII.GetBytes("i14Ff4US29DUE3pLXHLFqIHwcy/o+MkyJ0BXQ1uiOLzP+beIDIFnREKfYn34NCed");

        var claims = new[]
        {
            new Claim("Nome", func.Nome),
            new Claim("NIF", func.NIF.ToString()),
            new Claim("id", func.Id.ToString()),
            new Claim("TipoFuncionarioId", func.TipoFuncionario.Id.ToString())
        };

        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Issuer = "ISI-TP02-26024-25988",
            Audience = "api",
            Subject = new ClaimsIdentity(claims),
            Expires = DateTime.UtcNow.AddHours(1),
            SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }
}
```

Figure 25: Obter Token

Com estas funções, apresentadas acima, é possível realizar login com sucesso.

## 6.5 Swagger

Para facilitar o desenvolvimento e a validação dos métodos da API RESTful implementada, foi utilizada a ferramenta Swagger. O Swagger oferece uma interface interativa para documentar e testar os endpoints da API, proporcionando maior eficiência e agilidade durante o processo de desenvolvimento.

Na imagem abaixo podemos ver a página do swagger com alguns dos métodos implementados.

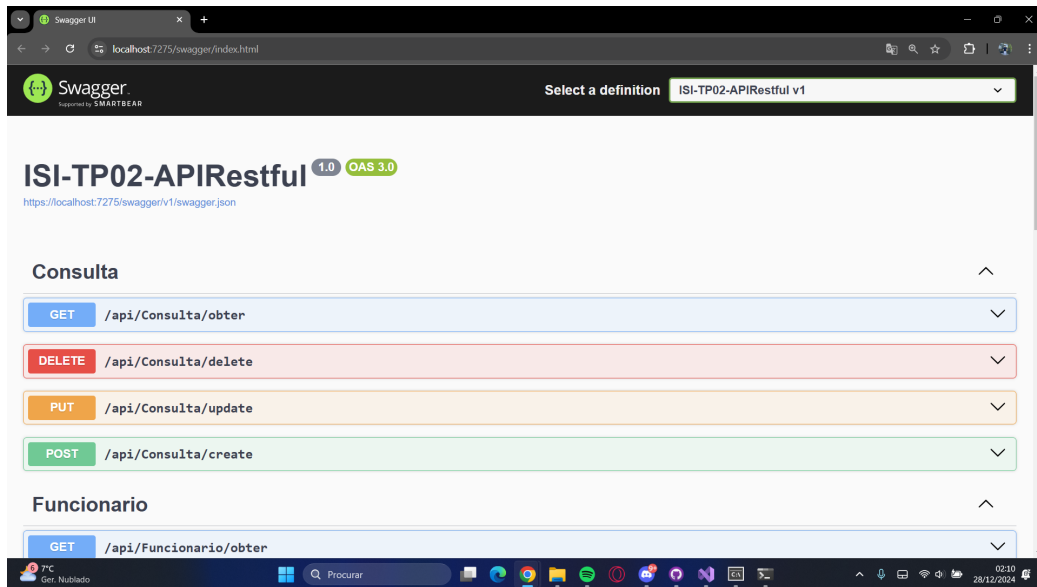


Figure 26: Swagger métodos



Já nesta abaixo, podemos ver um exemplo de um dos métodos da imagem anterior. O método GET /api/Funcionario/obter foi testado através do Swagger, permitindo visualizar os dados dos funcionários registrados no sistema.

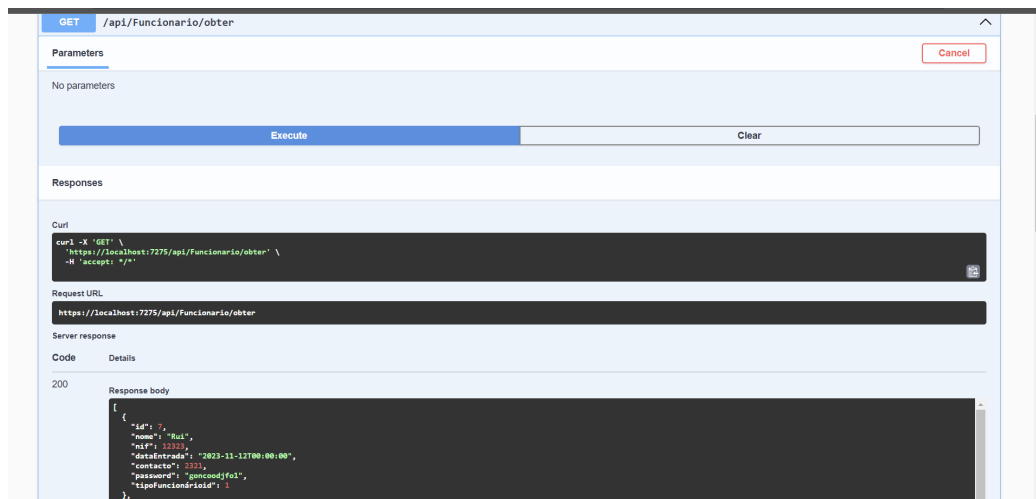


Figure 27: Método obter funcionário

O pedido retorna uma resposta HTTP 200 (sucesso) e um objeto JSON contendo informações como ID, nome, NIF, data de entrada, contato, senha e o tipo de funcionário. Este teste validou o funcionamento correto do endpoint, garantindo o acesso às informações de forma eficiente e estruturada.

## 6.6 Forms: Interação Cliente

O Forms, a última etapa deste trabalho, representa a interação com o cliente, demonstrando as funções que um funcionário após se ter autenticado pode fazer.

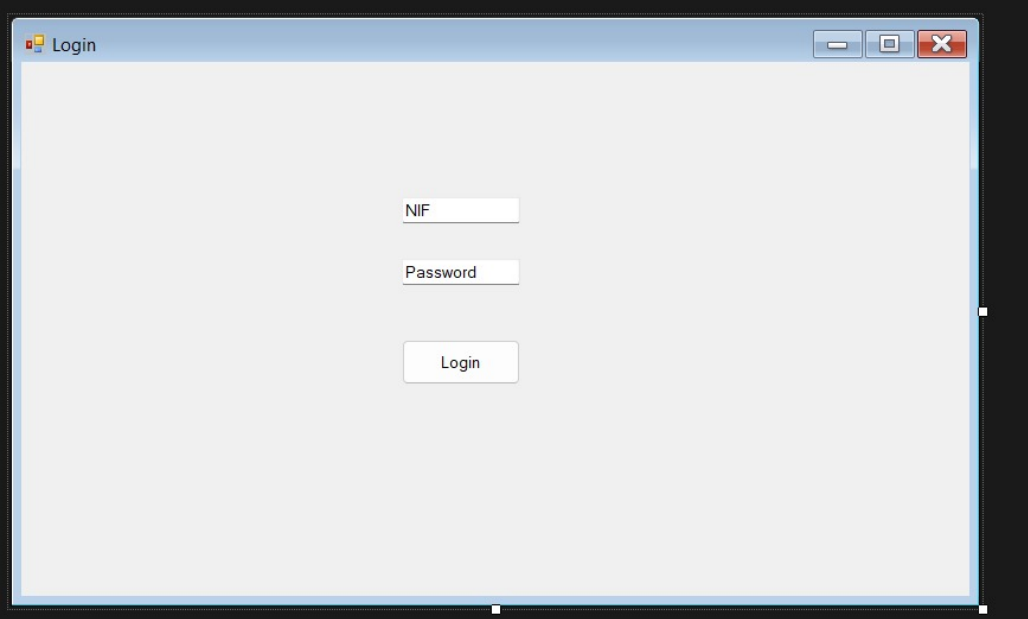


Figure 28: Página de Login

Após um Login bem-sucedido, será reencaminhado para a página central, onde poderá selecionar entre as opções disponíveis. Como é possível ver pela imagem abaixo, poderá escolher entre ver todos os funcionários, todas as consultas ou todos os utentes. Poderá ainda adicionar novos funcionários, novas consultas ou até novos utentes. Para a edição de algum dado ou até mesmo eliminação, pode-se fazer diretamente na tabela que nos é mostrada no GetAll de cada um.

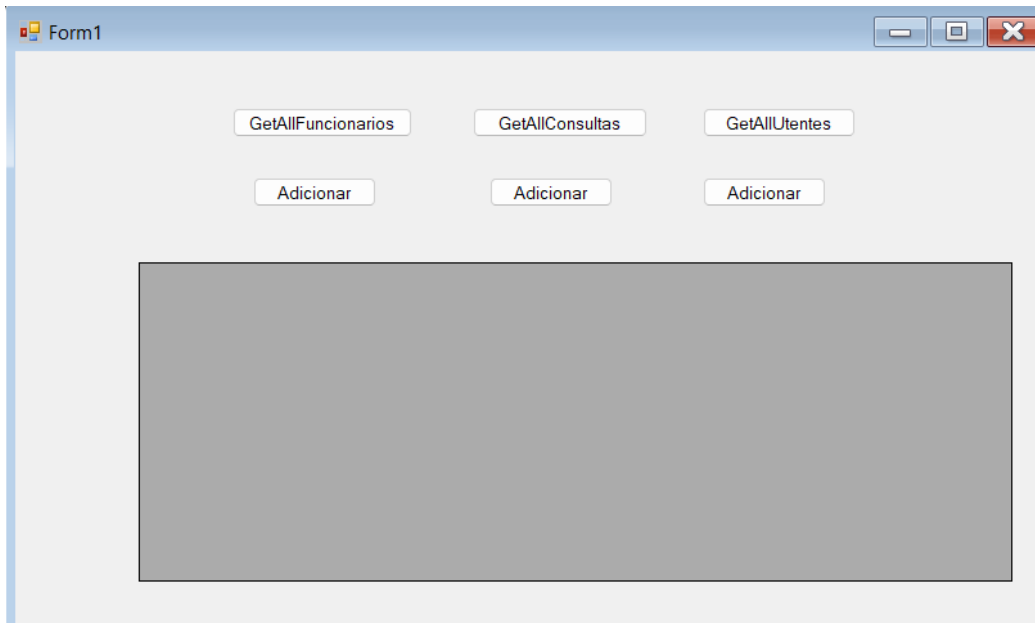
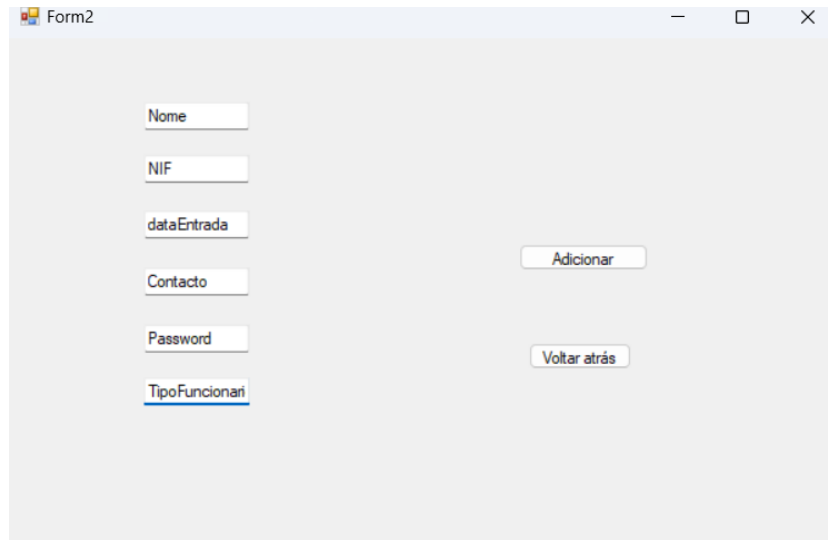


Figure 29: Página Inicial

Neste formulário abaixo podemos ver o página para adicionar um funcionário.

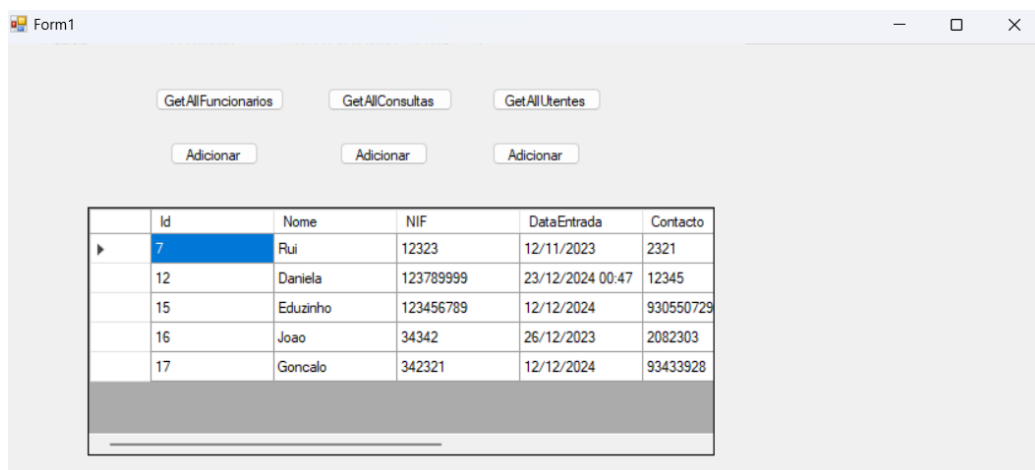


A screenshot of a web form titled 'Form2'. It contains several input fields arranged vertically on the left: 'Nome', 'NIF', 'dataEntrada', 'Contacto', 'Password', and 'TipoFuncionari'. To the right of these fields are two buttons: 'Adicionar' and 'Voltar atrás'.

Figure 30: Página adicionar funcionário

Após o preencher os dados e selecionado o adicionar, ele adiciona automaticamente na tabela getAll.

As páginas de adicionar consultas ou utentes seguem o mesmo modelo da de adicionar funcionários.



A screenshot of a web form titled 'Form1'. At the top, there are three buttons: 'GetAllFuncionarios', 'GetAllConsultas', and 'GetAllUtentes'. Below these are three 'Adicionar' buttons. The main part of the form is a table with the following data:

	Id	Nome	NIF	DataEntrada	Contacto
▶	7	Rui	12323	12/11/2023	2321
	12	Daniela	123789999	23/12/2024 00:47	12345
	15	Eduzinho	123456789	12/12/2024	930550729
	16	Joao	34342	26/12/2023	2082303
	17	Goncalo	342321	12/12/2024	93433928

Figure 31: GetAllFuncionários

## 7 Conclusão

Este projeto permitiu o planeamento e desenvolvimento de uma solução prática e robusta para a gestão de unidades hospitalares, integrando diferentes serviços e tecnologias juntas. Foi possível compreender e aplicar os conceitos abordados em aula, desenvolvendo-se assim um projeto de grande qualidade e cumprindo todos os requisitos pretendidos.

O processo abrangeu desde a conceção da base de dados no SQL Server e a sua migração para o Azure, até à criação de serviços distribuídos que facilitam a gestão de entidades fundamentais neste meio da saúde. A utilização de ferramentas como o *Visual Paradigm*, o *Visual Studio*, o *SoapUI* e a implementação de autenticação JWT foram essenciais para alcançar os objetivos propostos.

A integração entre métodos SOAP e RESTful destaca-se como um dos pontos mais significativos do projeto, mostrando a importância de compreender e adaptar diferentes metodologias para que esta comunicação tão importante aconteça.

Concluimos assim que este projeto contribuiu para melhorar o desempenho, a eficiência e a fiabilidade na gestão de unidades de saúde, propondo soluções que podem ser adaptadas para diferentes cenários hospitalares. Por fim, destaca-se a importância da integração de tecnologias modernas, respeitando as melhores práticas do mercado, para enfrentar os desafios do setor de saúde e fornecer uma experiência melhorada para os utilizadores e pacientes.

## References

- [1] SOAP UI - Disponível em: <https://www.soapui.org/>. Consultado em: [14,15,16,26,27 de dezembro].
- [2] How to work with Azure - Disponível em : <https://www.youtube.com/watch?v=EKqXAMLSnKQ&t=89s> Consultado em: [11,12 de dezembro].
- [3] Overleaf - Disponível em: <https://pt.overleaf.com/learn/latex/>.