



RELATÓRIO DE TRABALHO PRÁTICO 2

Relatório do Trabalho Pratico 2 ISI

LUÍS MARTINS

ALUNO Nº 16980

CARLOS RIBEIRO

ALUNO Nº 16986

Trabalho realizado sob a orientação de:
Luís Ferreira

Integração de Sistemas de Informação

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, Janeiro de 2021

Índice

Índice de Figuras	3
Introdução.....	4
Contextualização	4
Objetivos	4
Ferramentas Utilizadas.....	5
Estrutura do Projeto.....	5
Serviços REST.....	6
READ.....	8
CREATE	9
DELETE.....	11
UPDATE	12
API Externa Utilizada	14
Serviços SOAP.....	16
Client Criado	17
CONCLUSÃO	20

Índice de Figuras

Figura 1 – Modelo da Base de dados	5
Figura 2 - Design Básico de uma REST API	6
Figura 3 - Pasta Models que contem modelos criados para a API.....	7
Figura 4 - Pasta Controllers que contem Controllers criados para a API	7
Figura 5 - Método “GetPessoas”	8
Figura 6 - Definição de serviço GET "getPessoas"	9
Figura 7 - Método que insere uma pessoa nova na Base de Dados	10
Figura 8 - Método POST "inserePessoa"	10
Figura 9 - Método que permite eliminar uma pessoa da Base de dados	11
Figura 10 - Método DELETE que elimina uma pessoa da base de dados.....	12
Figura 11 - Método que atualiza dados de uma pessoa registada na Base de Dados	13
Figura 12 - Método PUT que atualiza dados de uma pessoa na base de dados.....	13
Figura 13 - Método que faz request de uma lista de marcas	14
Figura 14 - Método que faz request de uma lista de modelos de uma marca	15
Figura 15 - Metodos GET que devolvem dados sobre veículos	15
Figura 16 - Serviços SOAP criados	16
Figura 17 - Form de Registo de Conta Nova	17
Figura 18 - Form Gere Pessoas.....	18
Figura 19 - Form de Eliminação de Aluguer	19
Figura 20 - Instância de serviços SOAP	19
Figura 21 - Eliminação de Aluguer no Client	20

Introdução

Contextualização

Neste relatório visamos descrever o processo completo da realização do segundo trabalho prático da unidade curricular Integração de Sistemas de Informação.

Este projeto tem como objetivos o desenvolvimento de bibliotecas de serviços SOAP e RESTful complementada com a reutilização de serviços externos existentes.

O tema que funciona como base para este projeto revolve á volta da criação de uma API que gere o processo de aluguer de carros. Os dados destes, serão disponibilizados por uma API externa hospedada pela CIS Automotive.

Objetivos

Na fase de planeamento deste projeto definimos vários objetivos a cumprir, estes sendo:

- Assimilar conteúdos da Unidade Curricular
- Consolidar conceitos de Integração de Sistemas de Informação usando serviços web;
- Desenhar arquiteturas de integração de sistemas, recorrendo a APIs de interoperabilidade;
- Explorar ferramentas de suporte ao desenvolvimento de serviços web;
- Desenvolver um conjunto de serviços RESTful e SOAP, capazes de suportar todas as operações CRUD sobre o repositório existente;
- Documentar devidamente a API disponibilizada utilizando o standard Open API (swagger)
- Publicar todos os serviços desenvolvidos na cloud (ficará assim disponibilizada uma API de serviços). Os serviços desenvolvidos deverão ser alojados (hosted) numa PaaS –Platform as a Service : i) na Windows Azure; ii) na AppHarbor; iii) na Apprenda, ou noutra.

Ferramentas Utilizadas

Na realização deste projeto utilizamos as seguintes ferramentas:

- Visual Studio 2019
- OpenAPI 3.0 (swagger)
- Microsoft Azure

Estrutura do Projeto

Este projeto foi dividido em três partes quando nos referenciamos à componente C#:

- Restful Services;
- SOAP Services usando WCF para implementação;
- Aplicação criada para chamar e testar os serviços desenvolvidos;

Os serviços criados, serão capazes de realizar operações CRUD (Create, Read, Update e Delete) sobre uma base de dados SQL, alojada na cloud, usando o Microsoft Azure. Esta é constituída por duas tabelas: Pessoa e Aluguer, contendo as informações das pessoas registadas no sistema e os alugueres marcados respetivamente.

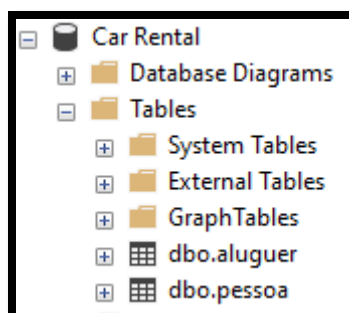


Figura 1 – Modelo da Base de dados

Os dados de acesso referentes a esta base de dados são os seguintes:
(**Servidor**: speedwayrental.database.windows.net, **Username**: admn, **Password**: Memelord101).

Os serviços Rest e SOAP desenvolvidos foram também publicados na cloud, mais uma vez utilizando o Microsoft Azure, onde os serviços Rest foram incluídos numa API criada no Azure (<https://speedwayrentalapi-apim.azure-api.net>).

Serviços REST

O objetivo desta componente do projeto, é o desenvolvimento de serviços REST, que devem ser implementados numa API e ter a capacidade de realizar operações CRUD sobre a base de dados criada, como anteriormente aludido.

Depois de ser devidamente implementado numa API, um individuo deve ter a possibilidade de aceder aos seus serviços, utilizando as indicações que o fornecedor da API tem disponibilizado.

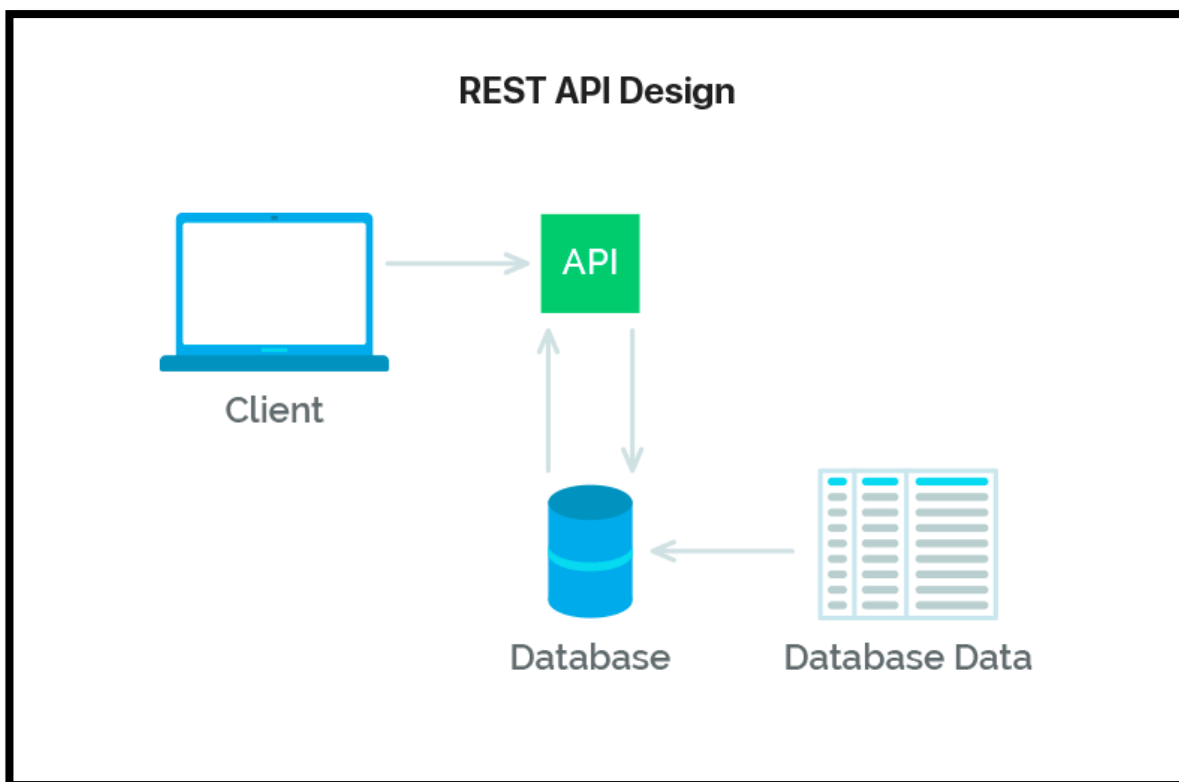


Figura 2 - Design Básico de uma REST API

Neste caso, o nosso grupo também fornece uma página com a API criada, devidamente documentada usando o OpenAPI(swagger), que pode ser acedida a partir deste link: <https://app.swaggerhub.com/apis-docs/Speedway-Rental/Speedway-Rental/v1.4>

A nossa solução “RestAPI” é constituída por duas componentes principais:

1. Os Modelos, classes que contem os métodos a ser chamados pelos controladores, como também as propriedades dos objetos que serão retornados por esses métodos. Os modelos criados para a nossa API encontram-se listados na figura seguinte:

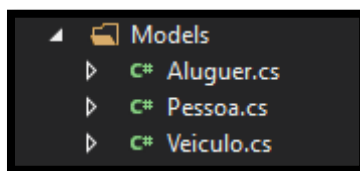


Figura 3 - Pasta Models que contem modelos criados para a API

2. Os Controladores, classes que, a partir dos recursos nas classes modelo, vão definir operações CRUD para serem utilizadas por eventuais utilizadores. Os controllers criados para a nossa API, são baseados da classe modelo como narrado anteriormente. Daí os nomes dos controllers, são inspirados das mesmas, e serão listados na seguinte figura:

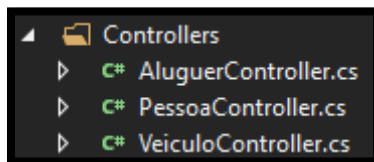


Figura 4 - Pasta Controllers que contem Controllers criados para a API

READ

Para este tipo de operação, criamos vários serviços, que permitem a leitura de dados registados na base de dados e a devolução dos mesmos em formato Json.

Como exemplo disto, podemos mencionar o serviço “getPessoas”. Este serviço, tem como função criar uma conexão à base de dados definida para este projeto e retirar uma lista de pessoas registadas.

```
public ListPessoas GetPessoas()
{
    ListPessoas listPessoas = new ListPessoas();
    DataSet ds = new DataSet();
    SqlConnection connection = new SqlConnection(connectionString);

    try
    {
        connection.Open();
    }
    catch
    {
        //Conexao falhou
        connection.Close();
        return null;
    }
    if (connection.State.ToString() == "Open")
    {
        //Construcao da query
        SqlCommand cmdins = new SqlCommand();
        string comando;
        cmdins.Connection = connection;

        comando = "SELECT * FROM pessoa";
        cmdins.CommandText = comando;

        SqlDataAdapter da = new SqlDataAdapter(cmdins.CommandText, connection);
        da.Fill(ds, "Pessoa");

        var json = JsonConvert.SerializeObject(ds);
        listPessoas = JsonConvert.DeserializeObject<ListPessoas>(json);
        connection.Close();
        return listPessoas;
    }
    else return null;
}
```

Figura 5 - Método “GetPessoas”

De seguida, no controller “PessoaController”, determinamos que tipo de operação vamos atribuir ao serviço. Para este caso, o método “getPessoas” vai ser definido como um método GET, ou seja, o serviço vai devolver um objeto tipo “ListaPessoas” que contem uma lista constituída por objetos do tipo “Pessoa”, ou, caso o método não for devidamente executado, retorna um “HttpStatusCode 404 – Not Found”.

```
[HttpGet("getPessoas")]
0 references
public ActionResult<ListPessoas> GetPessoas()
{
    ListPessoas listPessoas = new ListPessoas();
    listPessoas = pessoa.GetPessoas();
    if (listPessoas == null)
    {
        return NotFound(); // Retorna Codigo 404 visto que nao foi encontrada uma lista de pessoas
    }
    else return listPessoas;
}
```

Figura 6 - Definição de serviço GET "getPessoas"

CREATE

Para a criação deste tipo de operações, também foram concebidos serviços que permitem a adição de registos novos de dados na base de dados previamente mencionada.

Como exemplo, usaremos mais uma vez os métodos criados para uma Pessoa. Neste caso, vamos destacar o método que permite inserir uma nova pessoa na base de dados.

Este método mais uma vez, conecta-se à base de dados, mas antes de enviar uma query, é chamado um método criado chamado “ExistePessoa” que verifica a existência desse individuo na base de dados. Caso esse método retorne false, conseguimos proceder para a construção da query a ser utilizada para inserir os dados da pessoa na base de dados.

```

public bool InserirPessoaObjDB(Pessoa pessoaNova)
{
    SqlConnection connection = new SqlConnection(connectionString);

    try
    {
        connection.Open();
    }
    catch
    {
        //Conexao falhou
        connection.Close();
        return false;
    }
    if (connection.State.ToString() == "Open")
    {
        //Verificar se a pessoa ja existe
        if (ExistePessoa(pessoaNova.email_pessoa) == false)
        {
            //Construção da query...
            string comando;

            comando = "Insert into pessoa (primeiro_nome, ultimo_nome, password_pessoa, email_pessoa) values(@primeiro_nome, @ultimo_nome, @password, @email);";

            SqlCommand cmdins = new SqlCommand(comando, connection);

            cmdins.Parameters.AddWithValue("@primeiro_nome", pessoaNova.Primeiro_Nome);
            cmdins.Parameters.AddWithValue("@ultimo_nome", pessoaNova.Ultimo_Nome);
            cmdins.Parameters.AddWithValue("@password", pessoaNova.password_pessoa);
            cmdins.Parameters.AddWithValue("@email", pessoaNova.email_pessoa);

            //Executa a query
            int res = cmdins.ExecuteNonQuery();
            if (res > 0)
            {
                connection.Close();
                return true;
            }
            else return false;
        }
        else return false;
    }
    else return false;
}

```

Figura 7 - Método que insere uma pessoa nova na Base de Dados

Finalmente, no lado do Controller, definimos um método POST, que recebe um objeto Pessoa como parâmetro, e, caso o método que insere os dados dessa pessoa for bem sucedido, será retornado um "*HttpStatusCode 200 – Success*" e caso contrário, um "*HttpStatusCode 406 – Not Acceptable*".

```

[HttpPost("inserePessoa")]
public ActionResult InserirPessoaObjDB(Pessoa pessoaNova)
{
    if (pessoa.InserirPessoaObjDB(pessoaNova) == true)
    {
        return Ok(); //RetornaCodigo 200 visto que foi executado com sucesso
    }
    else return new StatusCodeResult(406); //Status code 406: Not Acceptable
}

```

Figura 8 - Método POST "inserePessoa"

DELETE

Para este tipo de operação, foram implementados serviços REST capazes de eliminar informações da base de dados utilizando o email de uma pessoa já registada na mesma como parâmetro. Foi criado um método que executa esse processo.

O método previamente mencionado, similarmente aos métodos anteriormente discutidos, efetua a conexão à base de dados em que, depois será construída a query com Statement Delete, depois usando o método `ExecuteNonQuery()`, que nos permite ver quantas linhas foram afetadas pela query. Verificamos que o número de linha afetadas são mais de 0, o que nos indica que essa pessoa foi de facto eliminada.

```
public bool DeletePessoa(string email)
{
    SqlConnection connection = new SqlConnection(connectionString);

    try
    {
        connection.Open();
    }
    catch
    {
        //Conexao falhou
        connection.Close();
        return false;
    }
    if (connection.State.ToString() == "Open")
    {
        //Construção da query
        string comando;

        comando = "DELETE FROM pessoa WHERE email_pessoa = @email;";
        SqlCommand cmdins = new SqlCommand(comando, connection);
        cmdins.Parameters.AddWithValue("@email", email);

        int res = cmdins.ExecuteNonQuery();
        if (res > 0)
        {
            connection.Close();
            return true;
        }
        else return false;
    }
    else return false;
}
```

Figura 9 - Método que permite eliminar uma pessoa da Base de dados

Agora, se nos deslocarmos para o lado do Controller, será definido um método DELETE, onde caso o método previamente discutido retornar true, no lado do cliente será recebido um “*HttpStatusCode 200 – Success*”, o que vai transmitir que o processo de eliminação foi executado com sucesso.

```
[HttpDelete("deletePessoa/{email}")]
0 references
public ActionResult DeletePessoa(string email)
{
    if (pessoa.DeletePessoa(email) == true)
    {
        return Ok(); //Retorna Codigo 200 visto que foi executado com sucesso
    }
    else return NotFound(); // Retorna Codigo 404 visto que nao foi encontrada uma pessoa
}
```

Figura 10 - Método DELETE que elimina uma pessoa da base de dados

UPDATE

Finalmente foram desenvolvidos serviços REST, que permitem atualizar os dados de certos componentes na base de dados.

Para este modo, utilizaremos na mesma, o processo de atualização dos dados de uma pessoa já registrada na base de dados. Para alcançar esse objetivo foi criado um método, que recebendo o ID da pessoa como parâmetro, mais uma vez, se conecta à base de dados e depois da construção da query com Statement Update, onde será utilizado o ID fornecido para identificar a linha a ser atualizada, esta será executada, utilizando o mesmo método previamente discutido na eliminação de uma pessoa, ExecuteNonQuery(), onde a mesma logica se aplica.

```
public bool UpdatePessoaObj(Pessoa pessoaUpdate)
{
    SqlConnection connection = new SqlConnection(connectionString);

    //Tenta verificar se a conexao é efetuada sem problema
    try
    {
        connection.Open();
    }
    catch
    {
        //Conexao falhou
        connection.Close();
        return false;
    }
    if (connection.State.ToString() == "Open")
    {
        //Construcao da query
        string comando;

        comando = "UPDATE pessoa SET primeiro_nome = @primeiro_nome, ultimo_nome = @ultimo_nome, password_pessoa = @password, email_pessoa = @email WHERE id_pessoa = @idpessoa";
        SqlCommand cmdins = new SqlCommand(comando, connection);

        cmdins.Parameters.AddWithValue("@primeiro_nome", pessoaUpdate.Primeiro_Nome);
        cmdins.Parameters.AddWithValue("@ultimo_nome", pessoaUpdate.Ultimo_Nome);
        cmdins.Parameters.AddWithValue("@password", pessoaUpdate.password_pessoa);
        cmdins.Parameters.AddWithValue("@email", pessoaUpdate.email_pessoa);
        cmdins.Parameters.AddWithValue("@idpessoa", pessoaUpdate.ID_Pessoa);

        int res = cmdins.ExecuteNonQuery();
        if (res > 0)
        {
            connection.Close();
            return true;
        }
        else return false;
    }
    else return false;
}
```

Figura 11 - Método que atualiza dados de uma pessoa registrada na Base de Dados

No lado do Controller, utilizamos a mesma metodologia empregada para a eliminação de uma pessoa. A única diferença sendo o tipo de parâmetros que cada uma recebe, onde a eliminação recebia uma email, o update recebe um objeto tipo Pessoa.

```
[HttpPut("updatePessoa")]
References
public ActionResult UpdatePessoaObj(Pessoa pessoaUpdate)
{
    if (pessoa.UpdatePessoaObj(pessoaUpdate) == true)
    {
        return Ok(); //Retorna Codigo 200 visto que foi executado com sucesso
    }
    else return NotFound(); // Retorna Codigo 404 visto que nao foi encontrada uma pessoa
}
```

Figura 12 - Método PUT que atualiza dados de uma pessoa na base de dados

API Externa Utilizada

De modo a ser capazes de fornecer uma lista variada de carros e os seus modelos, decidimos utilizar os dados disponibilizados por uma API externa hospedada pela CIS Automotive, como foi eludido anteriormente neste relatório.

Para ter acesso a esta API foi-nos pedido registar uma conta no site da RapidApi onde uma parte da API é hospedada. Depois, investigamos a forma de funcionamento de requests de dados, e que formato estes devem ser armazenados.

Com a utilização dos serviços fornecidos pelo site <https://json2csharp.com/> foi possível converter a string Json recebida da API para classes próprias de C#, daí utilizando esta informação, implementamos as mesmas no modelo Veiculo, para depois, utilizando o código modelo documentado no RapidApi, realizarmos um request que nos devolve as marcas disponiveis na API e de seguida, deserializamos a string json recebida para um objeto que as pode armazenar.

```
public Marcas SearchBrandAPI()
{
    var client = new RestClient("https://cis-automotive.p.rapidapi.com/getBrands");
    var request = new RestRequest(Method.GET);
    request.AddHeader("x-rapidapi-key", "ae6829c890mshafb5aa6e5e9eb4ap1c6a85jsn985f80de20a5");
    request.AddHeader("x-rapidapi-host", "cis-automotive.p.rapidapi.com");
    IRestResponse response = client.Execute(request);

    // Faz a desserialização dos dados em Json para um objeto
    Marcas listveiculos = JsonConvert.DeserializeObject<Marcas>(response.Content);

    return listveiculos;
}
```

Figura 13 - Método que faz request de uma lista de marcas

```
public Veiculos SearchModelsAPI(string marca)
{
    // String com url template para a pesquisa de modelos de uma marca na API
    string url = "https://cis-automotive.p.rapidapi.com/getModels?brandName=[MARCA]";

    // Construção da uri
    StringBuilder uri = new StringBuilder();
    uri.Append(url);
    uri.Replace("[MARCA]", HttpUtility.UrlEncode(marca));

    // Realização de um "request" à API externa
    var client = new RestClient(uri.ToString());
    var request = new RestRequest(Method.GET);
    request.AddHeader("x-rapidapi-key", "ae6829c890mshafb5aa6e5e9eb4ap1c6a85jsn985f80de20a5");
    request.AddHeader("x-rapidapi-host", "cis-automotive.p.rapidapi.com");
    IRestResponse response = client.Execute(request);

    // Faz a desserialização dos dados em Json para um objeto
    Veiculos listveiculos = JsonConvert.DeserializeObject<Veiculos>(response.Content);

    return listveiculos;
}
```

Figura 14 - Método que faz request de uma lista de modelos de uma marca

Por último implementamos um controller para o modelo Veículo onde é possível chamar estes métodos como métodos GET na nossa API.

```
// Serviço GET para buscar todas as marcas na API externa
[HttpGet("GetMarcas")]
References
public Marcas SearchMarcasAPI()
{
    return v.SearchBrandAPI();
}

// Serviço GET para a busca de dados de modelos de uma certa marca na API externa
[HttpGet("GetModelos/{marca}")]
References
public Veiculos SearchAPI(string marca)
{
    return v.SearchModelsAPI(marca);
}
```

Figura 15 - Metodos GET que devolvem dados sobre veículos

Serviços SOAP

No âmbito deste projeto, decidimos também, criar serviços SOAP que permitem gerir o lado dos alugueres. A componente dos alugueres, foi também implementada na REST API criada, mas, por motivos de experimentação, decidimos utilizar estes serviços SOAP na nossa aplicação Client criada, que vamos discutir mais à frente.

Estes serviços sendo similares aos serviços implementados na API REST, escolhemos não entrar em muito detalhe na explicação dos mesmos.

```
[OperationContract]
1 reference
bool AddAluguer(string email, string marca, string modelo, DateTime dataIn, DateTime dataOut);

/// <summary>
/// Metodo que devolve lista de alugueres
/// </summary>
/// <param name="email"></param>
/// <returns></returns>
[OperationContract]
1 reference
string ProcuraAlugueres(string email);

/// <summary>
/// Metodo que faz atualização de dados de um aluguer
/// </summary>
/// <param name="id_aluguer"></param>
/// <param name="nome_marca"></param>
/// <param name="nome_modelo"></param>
/// <param name="datain"></param>
/// <param name="dataout"></param>
/// <returns></returns>
[OperationContract]
1 reference
bool UpdateAluguer(int id_aluguer, string nome_marca, string nome_modelo, DateTime datain, DateTime dataout);

/// <summary>
/// Metodo que elimina uma aluguer
/// </summary>
/// <param name="id_aluguer"></param>
/// <returns></returns>
[OperationContract]
1 reference
bool EliminaAluguer(int id_aluguer);
```

Figura 16 - Serviços SOAP criados

Client Criado

Na criação do cliente para o teste dos métodos, decidimos criar uma aplicação Windows Forms desenvolvida em C#. Como foi mencionado anteriormente, esta aplicação utiliza os serviços disponibilizados pela API REST criada, assim como os serviços SOAP desenvolvidos em WCF.

Como exemplo da utilização de serviços da REST API, temos a forma como efetuamos o processo de registo de uma conta nova.

A screenshot of a Windows Forms application titled "Novo Registo". The form has a light gray background and a black border. It contains several text input fields and two buttons. The fields are labeled "Primeiro Nome", "Ultimo Nome", "Email", "Password", and "Confirmar Password". The "Password" and "Confirmar Password" fields are grouped together. At the bottom, there are two buttons: "Fazer Registo" and "Voltar para Menu".

Primeiro Nome	Ultimo Nome
<input type="text"/>	<input type="text"/>
Email <input type="text"/>	
Password <input type="text"/>	
Confirmar Password <input type="text"/>	
<input type="button" value="Fazer Registo"/> <input type="button" value="Voltar para Menu"/>	

Figura 17 - Form de Registo de Conta Nova

No form de registo apresentado na figura acima, quando efetuamos o registo, será realizado um request POST à nossa API utilizando o URL que nos foi fornecido após a publicação da API no Azure, mais o path que definimos antes de publicar os serviços.

De seguida, serializamos os dados de um objeto Pessoa que queremos introduzir na base de dados para Json e realizamos o pedido.

Foi também incluído um form adicional que implementa todos os métodos REST não utilizados no resto da aplicação, estes sendo a eliminação de uma pessoa da base de dados e a atualização de dados de uma pessoa na base de dados.

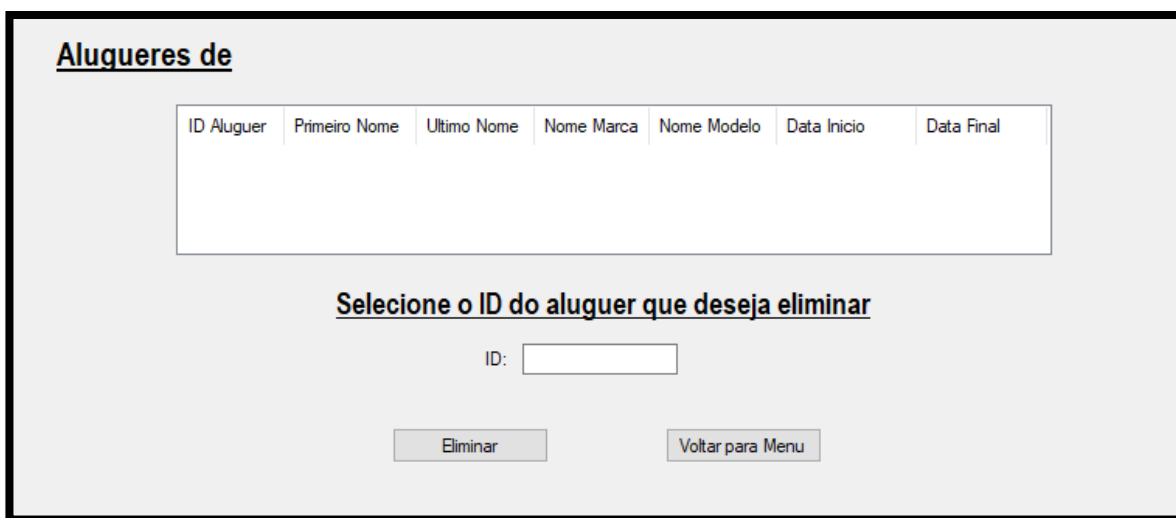
Note que todos estes métodos podem ser testados usando a documentação criada no OpenAPI fornecida anteriormente.

The screenshot displays a web interface titled "Gerir Pessoas". At the top, there is a table with five columns: "ID Pessoa", "Primeiro Nome", "Ultimo Nome", "Password", and "Email". The table is currently empty. Below the table, the interface is divided into two main sections: "Update Pessoa" on the left and "Delete Pessoa" on the right. The "Update Pessoa" section includes five input fields labeled "ID da Pessoa a Modificar:", "Primeiro Nome Novo:", "Ultimo Nome Novo:", "Password Nova:", and "Email Novo:", each followed by a text input box. Below these fields is a button labeled "Update Pessoa". The "Delete Pessoa" section includes a single input field labeled "Email da Pessoa a Eliminar:" followed by a text input box, and a button labeled "Delete Pessoa" below it. At the bottom center of the form is a button labeled "Voltar para Menu".

Figura 18 - Form Gere Pessoas

Mais à frente, é possível observar várias instâncias da utilização de serviços SOAP, maioritariamente na parte da realização e manipulação de alugueres.

A figura seguinte representa o form criado para eliminar um aluguer.



Alugueres de

ID Aluguer	Primeiro Nome	Ultimo Nome	Nome Marca	Nome Modelo	Data Inicio	Data Final
------------	---------------	-------------	------------	-------------	-------------	------------

Selecione o ID do aluguer que deseja eliminar

ID:

Figura 19 - Form de Eliminação de Aluguer

A forma como fazemos um pedido de serviços entre a REST API e os serviços SOAP variam em algumas componentes, neste caso como os serviços SOAP foram desenvolvidos em WCF, somos capazes de adicionar uma referencia de serviço no próprio projeto do Client. Desta forma, se colocarmos o URL dos serviços SOAP que nos foi dado quando os publicamos para o Azure (<http://wcfsoapservicesisi.azurewebsites.net/Service.svc?wsdl>), temos acesso aos mesmos quando iniciamos uma instância dos serviços no Client.

```
// Criar instancia de serviços
WCF_SOAP_Services_CloudVersion.AluguerClient aluguerClient = new WCF_SOAP_Services_CloudVersion.AluguerClient();
```

Figura 20 - Instância de serviços SOAP

Com esta instância criada conseguimos chamar os serviços SOAP criados. Na figura seguinte, mostramos um exemplo da eliminação de um aluguer no código do form visto anteriormente:

```
if (aluguerClient.EliminaAluguer(int.Parse(textBox1.Text)) == true)
{
    MessageBox.Show("Aluguer Eliminado com Sucesso");
}
else MessageBox.Show("O Processo não foi concluido");
```

Figura 21 - Eliminação de Aluguer no Client

CONCLUSÃO

Embora não tenhamos explorado tudo que nos foi ensinado ao longo desta cadeira, acreditamos que a realização deste projeto nos permitiu perceber o funcionamento básico do mundo dos serviços REST e SOAP, como também a importância das APIs desenvolvidas e documentadas por diferentes empresas à volta do mundo.

Ficamos também mais conscientes do propósito e das adversidades e dificuldades da publicação destes serviços online, e da forma como estes são transportados para a cloud, e a implicação deste processo.

Ao longo do desenvolvimento deste Trabalho Prático fomos assimilando a maior parte do conteúdo dado ao longo da cadeira de Integração de Sistemas de Informação, e com isto podemos dizer que esta experiência foi uma mais valia no enriquecimento do nosso conhecimento desta área.