Relatório do Trabalho Prático II da disciplina de Integração de Sistemas Informáticos

Gestão de Contactos

Arthur Fellipe Cerqueira Gomes – 24200

Engenharia de Sistemas Informáticos – Pós-Laboral

Dezembro de 2024

|  |
| --- |
| Afirmo por minha honra que não recebi qualquer apoio não autorizado na realização deste trabalho prático. Afirmo igualmente que não copiei qualquer material de livro, artigo, documento web ou de qualquer outra fonte exceto onde a origem estiver expressamente citada. |

Arthur Fellipe Cerqueira Gomes – 24200

**Lista de Figuras**

[Figura 1 – Diagrama de Entidade-Relacionamento (ERD) 7](#_Toc186392529)

[Figura 2 - Endpoints 8](#_Toc186392530)

[Figura 3 - Acesso à Base de Dados - SOAP 9](#_Toc186392531)

[Figura 4 - Acesso à Base de Dados - REST 9](#_Toc186392532)

[Figura 5 - Modelo Contacto - SOAP 10](#_Toc186392533)

[Figura 6 - DatabaseHelper - Conexão BD - SOAP 10](#_Toc186392534)

[Figura 7 - DatabaseHelper - GetAllContacts() 11](#_Toc186392535)

[Figura 8 - DatabaseHelper - GetContactById(int id) 11](#_Toc186392536)

[Figura 9 - DatabaseHelper - AddContact(Contact contact) 12](#_Toc186392537)

[Figura 10 - DatabaseHelper - UpdateContact(Contact contact) 12](#_Toc186392538)

[Figura 11 - DatabaseHelper - DeleteContact(int id) 12](#_Toc186392539)

[Figura 12 - ContactService.svc.cs - SOAP 13](#_Toc186392540)

[Figura 13 - IContactService - SOAP 13](#_Toc186392541)

[Figura 14 - Connected Services - REST 14](#_Toc186392542)

[Figura 15 - ServiceReference1 - REST 14](#_Toc186392543)

[Figura 16 - Configuração Swagger - REST 15](#_Toc186392544)

[Figura 17 - Configuração JWT - REST 15](#_Toc186392545)

[Figura 18 - Valores JWT - REST 16](#_Toc186392546)

[Figura 19 - Habilitação Swagger e Autenticação - REST 16](#_Toc186392547)

[Figura 20 - Model User - REST 17](#_Toc186392548)

[Figura 21 - AuthController - Rota e Construtor - REST 17](#_Toc186392549)

[Figura 22 - AuthController - Register - REST 18](#_Toc186392550)

[Figura 23 - AuthController - Login - REST 18](#_Toc186392551)

[Figura 24 - AuthAuxiliar - GenerateToken - REST 19](#_Toc186392552)

[Figura 25 - ContactController - Rota e Construtor - REST 20](#_Toc186392553)

[Figura 26 - ContactController - GetContacts - REST 21](#_Toc186392554)

[Figura 27 - ContactController - CreateContact - REST 21](#_Toc186392555)

[Figura 28 - ContactController - UpdateContact - REST 22](#_Toc186392556)

[Figura 29 - ContactController - DeleteContact - REST 22](#_Toc186392557)

ÍNDICE

[1. Introdução ao Projeto 6](#_Toc186392558)

[1.1. Problema a resolver 6](#_Toc186392559)

[1.2. Objetivos do Projeto 6](#_Toc186392560)

[2. Modelo de Dados: Diagrama de Entidade-Relacionamento (ERD) 7](#_Toc186392561)

[3. Descrição das Operações Principais Implementadas 8](#_Toc186392562)

[4. Arquitetura do Sistema 9](#_Toc186392563)

[4.1. Implementação e Acesso à Base de Dados 9](#_Toc186392564)

[4.2. Operações no Serviço SOAP 10](#_Toc186392565)

[4.3. Operações na API RESTful 14](#_Toc186392566)

[4.4. Serviços Externos a Integrar 23](#_Toc186392567)

[5. Conclusão 24](#_Toc186392568)

[6. Bibliografia 25](#_Toc186392569)

# Introdução ao Projeto

O presente relatório descreve o desenvolvimento do projeto Gestão de Contactos, concebido para resolver a dificuldade frequente das organizações em gerir eficientemente informações de contacto, como nomes, emails e números de telefone, de forma centralizada e acessível. Muitas vezes, esses dados encontram-se dispersos em diferentes sistemas ou armazenados de forma pouco estruturada, dificultando o acesso rápido e seguro, bem como a sua manutenção e atualização.

## 1.1. Problema a resolver

O projeto visa abordar a falta de um sistema unificado que permita armazenar, consultar e atualizar contactos de forma eficiente, promovendo uma comunicação organizacional mais eficaz. Além disso, há a necessidade de assegurar a segurança das informações sensíveis e garantir que apenas utilizadores autenticados e autorizados tenham acesso aos dados.

## 1.2. Objetivos do Projeto

Os principais objetivos deste projeto incluem:

* Desenvolver uma aplicação robusta para gestão centralizada de contactos.
* Implementar uma arquitetura multicamadas para separar responsabilidades e facilitar a manutenção.
* Utilizar um serviço **SOAP XML Web Service** como camada de acesso a dados (**Data Layer**).
* Desenvolver uma **API RESTful Web API** para atuar como camada de apresentação e autenticação.
* Assegurar a segurança das comunicações através de **JWT (JSON Web Token)**.

Todos os ficheiros de implementação do projeto se encontram anexados ao presente relatório, bem como foram disponibilizados online através do repositório no *GitHub* <https://github.com/arthur-fellipe/ISI_TP2_24200>.

# Modelo de Dados: Diagrama de Entidade-Relacionamento (ERD)

O modelo de dados baseia-se em duas entidades principais: Contactos e Utilizadores. A entidade Contactos armazena informações essenciais como nome, email e telefone, sendo identificada por um campo único ID. A entidade Utilizadores é responsável pela autenticação e contém campos como Username e PasswordHash, também identificados por um ID exclusivo. As duas entidades se relacionam através do ID do utilizador que criou o contacto, que funciona como chave estrangeira na tabela Contactos.

A screenshot of a computer

Description automatically generated

Figura 1 – Diagrama de Entidade-Relacionamento (ERD)

# Descrição das Operações Principais Implementadas

As operações principais da aplicação são expostas através de endpoints RESTful que permitem a gestão de contactos e a autenticação de utilizadores. A API oferece métodos para criar, atualizar, listar e excluir contactos, além de permitir a criação de um utilizador e a autenticação segura, conforme lista de endpoints a seguir:

01- Autenticação:

* POST /api/auth/register: Cria um novo utilizador.
* POST /api/auth/login: Autentica o utilizador e gera um token JWT.

02 - Gestão de Contactos:

* GET /api/contact: Retorna todos os contactos.
* GET /api/contact/{id}: Retorna um contacto específico pelo ID.
* POST /api/contact: Adiciona um novo contacto.
* PUT /api/contact/{id}: Atualiza um contacto existente.
* DELETE /api/contact/{id}: Remove um contacto pelo ID.

A screenshot of a computer

Description automatically generated

Figura 2 - Endpoints

# Arquitetura do Sistema

A arquitetura da aplicação é dividida em duas camadas principais: a camada de dados implementada via **SOAP XML Web Service** e a camada de apresentação realizada pela **API RESTful Web API**.

## 4.1. Implementação e Acesso à Base de Dados

A base de dados, modelada consoante Diagrama apresentado no Tópico 2, foi implementada utilizando o SQL Server, sendo o acesso realizado diretamente através de consultas SQL. Tanto o serviço SOAP como a API RESTful possuem configurações específicas para estabelecer a conexão com a base de dados, garantindo a integridade e segurança dos dados.

No serviço SOAP, o acesso é gerido pelo componente DatabaseHelper, responsável por executar operações relacionadas com a gestão dos contactos. A configuração de acesso é realizada através de uma string de conexão definida no ficheiro de configuração *web.config* da aplicação SOAP (Figura 3).

A black screen with white text

Description automatically generated

Figura 3 - Acesso à Base de Dados - SOAP

Na API RESTful, o acesso à base de dados para autenticação de utilizadores é configurado diretamente na camada correspondente, utilizando também uma string de conexão armazenada no ficheiro de configuração *appsettings.json* da aplicação (Figura 4).



Figura 4 - Acesso à Base de Dados - REST

Em ambas as camadas, são utilizadas bibliotecas adequadas (System.Data.SqlClient ou Microsoft.Data.SqlClient) para estabelecer as conexões e executar consultas parametrizadas, mitigando riscos de SQL Injection e garantindo um acesso eficiente e seguro aos dados.

## 4.2. Operações no Serviço SOAP

Como já relatado, foram desenvolvidas operações SOAP que funcionam como camada de dados para gestão dos contactos. Para tal, o primeiro passo após a configuração do acesso à Base de Dados foi a criação de um modelo para o objeto Contacto (Figura 5), que representa a sua estrutura dos dados e é crucial para a troca de informações entre o cliente e o serviço.

A screen shot of a computer program

Description automatically generated

Figura 5 - Modelo Contacto - SOAP

No contexto das operações SOAP, foram utilizadas as tags *DataContract* e *DataMember*, que desempenham um papel essencial para garantir a integração com a camada REST:

* **DataContract**: Marca a classe que representa o modelo de dados, indicando que ela será usada em comunicação via SOAP. Isso facilita a serialização dos objetos em XML, o formato de transporte de dados nas mensagens SOAP.
* **DataMember**: Marca as propriedades da classe que devem ser expostas na comunicação SOAP. Apenas os membros que são anotados com DataMember serão incluídos na mensagem SOAP.

Estabelecido o modelo do objeto, optou-se por implementar o componente DatabaseHelper (Figura 6), responsável por centralizar as operações relacionadas com a base de dados referentes à gestão dos contactos, garantindo uma abordagem estruturada e segura para a manipulação dos dados.

A screen shot of a computer

Description automatically generated

Figura 6 - DatabaseHelper - Conexão BD - SOAP

As principais responsabilidades do DatabaseHelper incluem:

* **Estabelecimento de conexão segura com a base de dados** através da string de conexão configurada no ficheiro de configuração.
* **Execução de operações CRUD (Create, Read, Update, Delete)** na tabela Contactos.
* **Gestão de exceções e tratamento de erros** para garantir a estabilidade do sistema.

As operações no banco de dados são realizadas através de Stored Procedures e consultas SQL embutidas no DatabaseHelper, tendo sido implementados os métodos que são utilizados diretamente pela camada SOAP para interagir com a base de dados.

* GetAllContacts(): Retorna todos os contactos.

A computer screen shot of a program

Description automatically generated

Figura 7 - DatabaseHelper - GetAllContacts()

* GetContactById(int id): Retorna um contacto específico pelo ID.

A computer screen shot of a program

Description automatically generated

Figura 8 - DatabaseHelper - GetContactById(int id)

* AddContact(Contact contact): Adiciona um novo contacto.

A computer screen shot of a program code

Description automatically generated

Figura 9 - DatabaseHelper - AddContact(Contact contact)

* UpdateContact(Contact contact): Atualiza informações de um contacto existente.

A computer screen shot of a program code

Description automatically generated

Figura 10 - DatabaseHelper - UpdateContact(Contact contact)

* DeleteContact(int id): Remove um contacto pelo ID.

A computer screen shot of a code

Description automatically generated

Figura 11 - DatabaseHelper - DeleteContact(int id)

Após a criação das operações no DatabaseHelper, é necessário configurar o ficheiro *ContactService.svc.cs* (Figura 12), que implementa o serviço SOAP responsável pela gestão dos contactos. Esta classe funciona como intermediária entre os métodos expostos ao cliente via SOAP e a base de dados, utilizando o DatabaseHelper para realizar as operações de acesso ao banco de dados.

A screen shot of a computer program

Description automatically generated

Figura 12 - ContactService.svc.cs - SOAP

A classe ContactService contém um campo privado db, que é uma instância do DatabaseHelper, responsável pela interação direta com a base de dados. Cada operação de gestão de contactos, como adicionar, atualizar ou eliminar um contacto, é implementada como um método público na classe, sendo exposta como um serviço SOAP que pode ser chamada por um cliente.

Quando o cliente solicita uma operação, o ContactService processa a requisição, utiliza o DatabaseHelper para manipular os dados e retorna um código de sucesso ou falha, dependendo do resultado. A separação de responsabilidades entre as duas classes torna o código modular e facilita a manutenção, com o ContactService gerenciando a comunicação via SOAP e o DatabaseHelper lidando com a complexidade do acesso à base de dados.

Por fim, a interface IContactService (Figura 13) define os contratos das operações SOAP. Ela descreve os métodos que o serviço oferece, mas sem implementar a lógica de negócio, garantindo que o serviço respeite um contrato consistente e permitindo uma comunicação clara e estruturada entre o cliente e o servidor.

A screen shot of a computer program

Description automatically generated

Figura 13 - IContactService - SOAP

## 4.3. Operações na API RESTful

Criada a camada de dados, passou-se a desenvolver a API RESTful, responsável por gerir a autenticação dos utilizadores e encaminhar chamadas ao serviço SOAP para realizar operações CRUD. Inicialmente, foi preciso conectar as duas camadas de serviços, o que foi feito utilizando a ferramenta Connected Services no Visual Studio (Figura 14). A partir da URL do serviço SOAP, *http://localhost:55211/ContactService.svc*, foi possível estabelecer a conexão, permitindo que a aplicação RESTful consumisse os métodos expostos.

A black rectangular object with a black border

Description automatically generated

Figura 14 - Connected Services - REST

Esse processo gerou automaticamente uma referência, denominada *ServiceReference1* (Figura 15), que contém as definições e os métodos do serviço SOAP, facilitando a comunicação com a API. Com isso, foi possível invocar as operações SOAP diretamente na API, garantindo a interoperabilidade entre os dois sistemas.

A screenshot of a computer program

Description automatically generated

Figura 15 - ServiceReference1 - REST

Ato contínuo, procedeu-se com a configuração do arquivo *Program.cs* para definir os serviços e middlewares utilizados pela aplicação, bem como incluir o suporte ao Swagger, que documenta as APIs, e configurar a autenticação baseada em JWT (JSON Web Token), garantindo a segurança.

O Swagger foi configurado para gerar a documentação da API e disponibilizá-la de forma interativa através de uma interface gráfica (Figura 16). Isso facilita a visualização e testes das operações da API, tanto durante o desenvolvimento quanto após a implementação.

A computer screen shot of a program

Description automatically generated

Figura 16 - Configuração Swagger - REST

As principais partes dessa configuração incluem:

* **SwaggerDoc**: Define o nome e a versão da documentação. O título é "GestaoContactosAPI" e a versão é "v1".
* **SecurityDefinition**: Adiciona uma definição de segurança, neste caso, para autenticação com JWT. O JWT será enviado no cabeçalho da requisição com o prefixo "Bearer", e o Swagger permite que o usuário insira o token de forma interativa.
* **SecurityRequirement**: Define que a API requer autenticação via JWT. O Swagger mostra um campo para que o usuário possa inserir o token antes de fazer chamadas para os endpoints.

Em seguida, a autenticação por JWT é configurada para proteger os endpoints da API (Figura17). Este é um padrão utilizado para transmitir informações de forma segura entre duas partes, geralmente um servidor e um cliente, em formato JSON. O token é composto por três partes: o cabeçalho (header), que especifica o algoritmo de assinatura utilizado; o corpo (ou payload), que contém as informações ou reivindicações (claims) que podem incluir dados como o usuário autenticado, permissões ou outras informações relevantes; e a assinatura, que garante a integridade do token e a autenticidade da origem.

A computer screen with text on it

Description automatically generated

Figura 17 - Configuração JWT - REST

O *JwtBearerDefaults.AuthenticationScheme* indica que o tipo de autenticação é Bearer Token, que é o padrão para autenticação via JWT. Além disso, a configuração define os parâmetros de validação do token, como:

* **ValidateIssuer**: Verifica se o emissor do token é válido.
* **ValidateAudience**: Verifica se o público do token é válido.
* **ValidateLifetime**: Verifica se o token ainda está dentro do seu período de validade.
* **ValidateIssuerSigningKey**: Verifica se a chave de assinatura do token é válida.

A chave de assinatura é recuperada da configuração da aplicação, através de valores definidos no ficheiro *appsettings.json* (Figura 18), garantindo que os tokens enviados sejam validados corretamente.

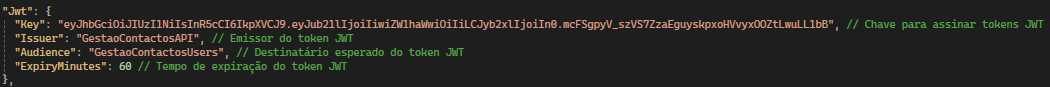


Figura 18 - Valores JWT - REST

Em seguida, a aplicação é construída com todas as configurações e serviços definidos anteriormente, sendo habilitado o Swagger somente quando o ambiente de desenvolvimento está ativo e com ativação dos middlewares que garantem que a aplicação valide as credenciais de autenticação (JWT) e verifique se o usuário tem permissão para acessar os recursos solicitados. Eles são executados antes do mapeamento dos controladores, para que as verificações de segurança sejam feitas antes de processar as requisições, conforem Figura 19.

A computer screen shot of a program code

Description automatically generated

Figura 19 - Habilitação Swagger e Autenticação - REST

A próxima etapa do desenvolvimento foi a definição da classe *User* (Figura 20), como estabelecida na base de dados.

A screen shot of a computer

Description automatically generated

Figura 20 - Model User - REST

A classe em questão é necessária para desenvolvimento dos métodos de registo e login, criados no ficheiro *AuthController.cs*, responsável pelo gerenciamento da autenticação e do registo de utilizadores na API. Tal ficheiro define, inicialmente, a rota base para o controlador, que ficará acessível via */api/Auth*, enquanto seu construtor injeta uma instância de IConfiguration para acessar configurações armazenadas no *appsettings.json*, como a string de conexão com a base de dados e as chaves JWT, conforme Figura 21.

A screen shot of a computer

Description automatically generated

Figura 21 - AuthController - Rota e Construtor - REST

O método Register (Figura 22) tem como principal objetivo permitir que novos utilizadores sejam registados no sistema de forma segura e eficiente. Ao receber uma requisição do tipo POST com os dados do utilizador no corpo da mensagem (propriedades Username e PasswordHash), o método inicia validando se essas informações foram devidamente preenchidas. Caso qualquer uma delas esteja ausente, o sistema responde imediatamente com um código 400 (Bad Request), informando que ambos os campos são obrigatórios.

Uma vez validada a entrada, o método estabelece uma conexão com a base de dados utilizando a string de conexão definida no ficheiro de configuração *appsettings.json*. Em seguida, é executado um comando SQL do tipo INSERT INTO, onde o nome de utilizador (Username) e a palavra-passe (PasswordHash) são armazenados na tabela Utilizadores. Antes de serem persistidos, os dados passam por um processo de segurança: a palavra-passe é criptografada utilizando o algoritmo BCrypt. Esse processo garante que, mesmo que a base de dados seja comprometida, as palavras-passe não poderão ser facilmente revertidas ao seu formato original.

A screen shot of a computer code

Description automatically generated

Figura 22 - AuthController - Register - REST

Se a operação for bem-sucedida, o servidor responde com um código 200 (OK), informando que o utilizador foi registado com sucesso. Caso ocorra algum erro durante a execução do comando SQL — como uma violação de chave única ou falha na conexão com a base de dados —, o método captura essa exceção e retorna um código 500 (Internal Server Error) com a mensagem detalhada do problema ocorrido. Assim, o método Register assegura que apenas entradas válidas sejam aceites, mantendo a integridade e segurança dos dados armazenados.

Já o método Login (Figura 23) é responsável por autenticar utilizadores existentes e fornecer-lhes um token JWT para acesso às funcionalidades protegidas da API. Assim como no método de registo, a primeira etapa consiste em validar se os campos Username e PasswordHash foram devidamente fornecidos na requisição POST.

A computer screen shot of text

Description automatically generated

Figura 23 - AuthController - Login - REST

Após a validação inicial, o método estabelece uma conexão com a base de dados e executa um comando SQL para procurar, na tabela Utilizadores, um registo correspondente ao nome de utilizador fornecido. Se nenhum utilizador for encontrado, ou se ocorrer algum problema durante a execução da query, o método retorna uma resposta com o código 401 (Unauthorized), informando que as credenciais fornecidas são inválidas.

Se um utilizador correspondente for encontrado, o próximo passo é verificar se a palavra-passe fornecida corresponde ao hash armazenado na base de dados. Essa verificação é feita utilizando o método BCrypt.Verify, que compara a palavra-passe introduzida com o PasswordHash armazenado. Caso haja uma correspondência bem-sucedida, o sistema gera um token JWT por meio de uma função auxiliar chamada GenerateToken (Figura 24), definida no ficheiro *AuthAuxiliar.cs*.

A computer screen shot of a program code

Description automatically generated

Figura 24 - AuthAuxiliar - GenerateToken - REST

Esse token contém informações essenciais sobre o utilizador, como seu ID e nome de utilizador, além de uma data de expiração definida nas configurações do sistema. O token é assinado digitalmente utilizando uma chave secreta configurada no ficheiro *appsettings.json*, garantindo sua integridade e autenticidade.

Por fim, o método Login retorna um código 200 (OK), juntamente com o token gerado, que deverá ser incluído nas requisições subsequentes pelo cliente no cabeçalho Authorization. Caso a validação da palavra-passe falhe, o método retorna uma resposta com o código 401 (Unauthorized), informando que as credenciais são inválidas.

Já desenvolvida toda a aplicação quanto à autenticação dos utilizadores, passou-se para o desenvolvimento do ContactController, responsável por expor endpoints RESTful para operações CRUD (Create, Read, Update, Delete) relacionadas aos contactos, integrando-se diretamente ao serviço SOAP previamente desenvolvido. Através dessa integração, o controlador atua como uma camada intermediária, facilitando a interação entre os clientes da API REST e os métodos do serviço SOAP.

Conforme Figura 25, o controlador é anotado com os atributos [Authorize], garantindo que apenas utilizadores autenticados possam acessar os seus endpoints, e [ApiController], que habilita funcionalidades específicas para controladores API no ASP.NET Core. Além disso, a rota api/[controller] define que todas as requisições direcionadas para /api/contact serão gerenciadas por esta classe.

No construtor do ContactController, é injetado um cliente HTTP através de IHttpClientFactory, que é então utilizado para criar uma instância do ContactServiceClient, cliente gerado automaticamente após adicionar a referência ao serviço SOAP via Connected Services. Esse cliente estabelece comunicação com os métodos disponibilizados pelo serviço SOAP, garantindo que todas as operações do controlador sejam devidamente encaminhadas.

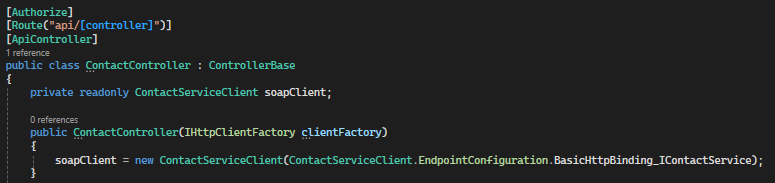


Figura 25 - ContactController - Rota e Construtor - REST

Em seguida, foram desenvolvidos os métodos das operações CRUD. Cada operação segue uma lógica semelhante:

* Métodos GetAllContacts e GetContactById (Figura 26) consultam os contactos existentes na base de dados através dos métodos GetAllContactsAsync e GetContactByIdAsync do serviço SOAP, retornando a lista completa ou um contacto específico, conforme a requisição. Em ambos os casos, verificações são feitas para garantir que os resultados sejam válidos, retornando 404 (Not Found) caso nenhum contacto seja encontrado.

A computer screen shot of a program

Description automatically generated

Figura 26 - ContactController - GetContacts - REST

* O método CreateContact (Figura 27) utiliza o método AddContactAsync do serviço SOAP para criar um novo contacto. Durante esta operação, o UserID do utilizador autenticado é extraído do token JWT, garantindo que o contacto criado seja associado corretamente ao responsável.

A computer screen shot of a program code

Description automatically generated

Figura 27 - ContactController - CreateContact - REST

* Para atualizar um contacto, o método UpdateContact (Figura 28) utiliza UpdateContactAsync, permitindo que as informações sejam ajustadas com base nos dados fornecidos na requisição.

A computer screen shot of code

Description automatically generated

Figura 28 - ContactController - UpdateContact - REST

* A remoção de contactos é realizada pelo método DeleteContact (Figura 29), que chama DeleteContactAsync, eliminando o registo correspondente com base no identificador fornecido.

A computer screen shot of a program code

Description automatically generated

Figura 29 - ContactController - DeleteContact - REST

Em todas essas operações, há tratamento de exceções para capturar possíveis erros durante a comunicação com o serviço SOAP ou na interação com a base de dados. Os retornos seguem convenções padrão de APIs RESTful, utilizando códigos de status apropriados (200 OK, 400 Bad Request, 404 Not Found ou 500 Internal Server Error) para informar o cliente sobre o sucesso ou falha da operação.

Essa estrutura modular e bem definida facilita a manutenção, além de proporcionar uma integração transparente entre dois padrões de comunicação distintos. O ContactController não apenas gerencia as operações relacionadas aos contactos, mas também garante que todas as requisições sejam devidamente validadas, autenticadas e tratadas de forma segura e eficiente.

## 4.4. Serviços Externos a Integrar

Para enriquecer a aplicação, será integrado o serviço externo NumVerify para verificar a validade do número de telefone sempre que um novo contacto for criado ou atualizado. Isso aumentará a confiabilidade do sistema. A conexão com a API externa será feita através da camada REST.

# Conclusão

Este projeto resultará numa aplicação segura e robusta, combinando um serviço SOAP para acesso a dados, uma API RESTful para apresentação e autenticação, e integração com serviços externos para fornecer uma experiência completa e fiável ao utilizador.

Além disto, o projeto contribuirá significativamente para o desenvolvimento de competências em integração de sistemas, consolidando conceitos aprendidos na Unidade Curricular e potenciando o uso de tecnologias modernas no contexto de fornecimento de serviços web. A partir do que foi construído, há espaço para expandir os serviços, de forma a explorar ainda mais o poder da integração de dados de forma segura.

# Bibliografia

*JWT.IO - JSON Web Tokens Introduction*. Disponível em: https://jwt.io/introduction. Acesso em: 15 dez. 2024.

*NumVerify - Free Phone Number Validation & Lookup API.* Disponível em: https://numverify.com/documentation. Acesso em: 15 dez. 2024.