Relatório do trabalho da disciplina de Integração de Sistema de Informação

Projeto ISI: Gestão Stock

Nuno Filipe Fernandes de Castro | Nº 4944

André Filipe de Freitas Rodrigues | Nº 25975

Regime Pós-laboral

Ano letivo 2024/2025

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e Ave

RESUMO

Este projeto tem como objetivo o desenvolvimento de um sistema de gestão de stock para um armazém, com foco na interoperabilidade e integração de serviços. O sistema permitirá que utilizadores façam a gestão de produtos e registem movimentação de stock. A solução será construída utilizando uma arquitetura híbrida, combinando **serviços SOAP(WCF)** como camada de acesso a dados e uma **API RESTful** que consumirá os serviços SOAP para disponibilizar operações em formato JSON.

A Base de dados será implementada no **SQL Server**, com consultas SQL puras para **manipulação dos dados** (não foram utilizadas ORM). O modelo de dados incluirá tabelas para servir as várias operações disponibilizadas.

ABSTRACT

This project aims to develop a stock management system for a warehouse, focusing on interoperability and service integration. The system will enable users to manage products and register stock movements. The solution will be built using a hybrid architecture, combining **SOAP services (WCF)** as the data access layer and a **RESTful API** that will consume the SOAP services to provide operations in JSON format.

The database will be implemented in **SQL Server**, using pure SQL queries for data manipulation (no ORM was used). The data model will include tables to support the various operations provided by the system.

Índice Figuras

Índice Tabelas

Glossário

API – Interface de programação de aplicação, que define as regras que precisamos de seguir para comunicar com outros sistemas. (aws, s.d.)

Back-end – Camada principal, que dá a inteligência para o software, processando dados e executando tarefas. (Softplan, 2021)

Business Layer – Agrupamento lógico de componentes que provêm funcionalidades à aplicação. (Meier, 2016)

Claims – Papel ou função atribuída a um utilizador emitida por uma fonte confiável, como "Email". (Baltieri, s.d.)

Connection String – Sequência de caracteres que contém informações necessárias para estabelecer uma conexão com a base de dados. (O que é Connection String, s.d.)

Data Layer – Agrupamento lógico de componentes que provêm acesso aos dados da aplicação. (Meier, 2016)

Endpoints – Forma através da qual a aplicação solicita um serviço a outra aplicação. (cloudflare, s.d.)

Framework – Arquitetura de programação que reúne códigos genéricos para simplificar o desenvolvimento. (Bastos, 2023)

Front-end – O que os utilizadores veem, inclui elementos visuais como botões e mensagens de texto. (Softplan, 2021)

JWTBearer – Representa de forma segura solicitações entre duas partes. (IBM, s.d.)

Postman – Aplicação para construir e utilizar APIS. (POSTMAN, s.d.)

Queries – Solicitação de informação feita à base de dados. (Clemente, s.d.)

RESTful API – Interface que dois sistemas de computador usam para trocar informações de forma segura. (aws, s.d.)

SQL Server – Sistema de gerenciamento de banco de dados relacional. (rwestMSFT & MikeRayMSFT, 2024)

SOAP.UI – Ferramenta utilizada para consumir e testar WebServices. (Cristiano, 2016)

Swagger – Ferramenta utilizada para documentar API. (Rodrigues, s.d.)

Tokens – Dispositivo eletrónico que gera senhas. (TOTVS, 2023)

Stored Procedures – Conjunto de comandos em SQL que podem ser executados de uma só vez. (Thiago, 2008)

Siglas e Acrónimos

BD – Base de Dados

CRUD – Create, Read, Update, delete

ER – Entidade relação

JWT – JSON Web Token

JSON- Java Script Object Notation

ORM- Object-Relatiotion Mapping

SOAP- Simple Object Access Protocol

WCF – Windows Communication Foundation

URL - Uniform Resource Locator

1. Introdução

O presente trabalho centra-se no desenvolvimento de um sistema de gestão de stocks para um armazém, com foco na interoperabilidade e **integração de serviços web** para melhorar a eficiência na gestão de produtos e movimentações de stock. A gestão de armazéns, especialmente em contextos onde diferentes sistemas necessitam de comunicar, exige soluções que garantam a consistência e acessibilidade dos dados.

O projeto utiliza uma abordagem híbrida, combinando serviços **SOAP(WCF)**, para acesso aos dados e uma **API RESTful** que consome esses serviços para disponibilizar as operações em **formato (JSON).** O acesso à base de dados será realizado com consultas SQL puras, eliminando o uso de **frameworks ORM**, garantindo maior controlo sobre as operações.

Este relatório detalha o processo de design e implementação do sistema, cobrindo desde o modelo de dados até à **definição dos endpoints**. O trabalho também destaca a importância de soluções híbridas que maximizem a interoperabilidade e a eficiência de gestão de armazém.

1.1. Objetivos

- Desenvolver um sistema utilizando **serviços SOAP e API RESTful** para gerir o stock;
- Implementar uma **base de dados SQL Server** para armazenar informações de produtos, utilizadores, movimentações de stock e permissões.
- Garantir segurança no acesso aos dados, utilizando **tokens de autenticação.**
- Promover a interoperabilidade entre diferentes sistemas, como suporte a vários formatos de dados.

1.2. Estrutura do Documento

Este relatório está divido em várias secções que cobrem cada fase do projeto, desde a análise inicial até à implementação e avaliação. A estrutura é a seguinte:

- Especificação e Arquitetura do sistema: nesta fase do projeto, foram realizados as principais definições e o planeamento da solução. Iniciou-se com a descrição do cenário, contextualizando as principais funcionalidades. Foi elaborado o modelo de dados representado por um **Diagrama Entidade Relacionamento (ER)**, que define a estrutura da base de dados. Além disso foram descritas as operações principais que o sistema irá implementar, dividindo as funcionalidades entre os serviços SOAP e os endpoints da API RESTful. Foi também definida a forma de acesso à base de dados, utilizando o SQL Server e consultas SQL puras.
- Codificação e implementação: na fase de codificação, o sistema foi **estruturado em diferentes camadas para garantir modularidade e clareza**. Os serviços SOAP, desenvolvidos em WCF, foram responsáveis por lidar diretamente com as operações de base de dados, centralizando a logica do negócio oferecendo funcionalidades como gestão de utilizadores, produtos, fornecedores e encomendas. Por outro lado, a API RESTFUL foi **projetada para consumir os serviços SOAP** e disponibilizar os dados em formato JSON, permitindo interoperabilidade com sistemas externos. A segurança do sistema foi implementada com tokens JWT, garantindo acesso seguro e controlado às funcionalidades, com permissões baseadas em níveis de acesso, como admin, gestor e operador. Além disso foi **desenvolvido um cliente em C# com Windows Forms** para oferecer uma interface gráfica, facilitando a interação com o sistema e possibilitando operações como registo de fornecedores e consulta de encomendas.
- Teste e validação: Para assegurar a confiabilidade do sistema, foi realizada uma extensa bateria de testes em cada componente. Os serviços SOAP foram validados com o uso do SOAP.UI, garantindo que todas as operações

respondessem corretamente às solicitações e executassem consultas precisas na base de dados. A API RESTFUL foi testada com ferramentas como Postman e Swagger, confirmando o funcionamento correto dos endpoints e a comunicação fluida com os serviços SOAP. **A aplicação cliente foi submetida a testes práticos, simulando operações reais realizadas pelos utilizadores**, como registo de novas encomendas e consulta de fornecedores. Por fim, foi efetuada uma validação integrada de todo o sistema, garantindo a interoperabilidade entre as camadas e a consistência dos dados durante o fluxo de operações.

2. Especificação e Arquitetura

Neste capítulo, são detalhas as **especificações do sistema e a arquitetura** definida para a implementação. São apresentados o modelo de dados, as operações a serem implementadas e a divisão entre os serviços e a estratégia de acesso à base de dados.

2.1. Arquitetura

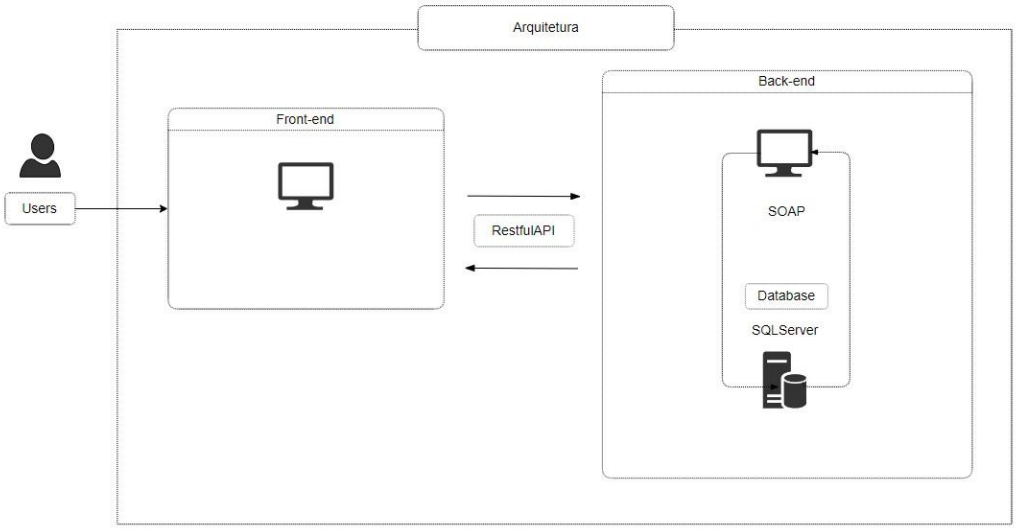

Figura 1 - Arquitetura do sistema

Na figura 1, podemos verificar a arquitetura desenvolvida para o projeto, destacando **as principais camadas e fluxos de comunicação** entre os componentes.

- **Front-end**, comunica com o **back-end**, via RESTful API, garantindo flexibilidade e compatibilidade com diferentes interfaces.
- **Back-End**, utiliza o SOAP como camada de dados, acedendo diretamente o SQL Server para garantir controlo, segurança e consistência na manipulação dos dados.

2.2. Operações SOAP (Data Layer)

Os serviços SOAP implementam a camada de acesso direto aos dados (**Data Layer)** no sistema de gestão de stock. Esta camada é responsável por executar as operações fundamentais de manipulação de dados na base de dados. **Todas as interações com a base de dados** são realizadas utilizando **comandos SQL** simples ou **stored procedures**, garantindo total controlo sobre as operações.

Os serviços SOAP são projetados para:

- **Inserir novos registos na base de dados,** como produtos ou movimentações de stock.
- **Atualizar informações existentes**, como preços de produtos ou quantidade em stock.
- **Eliminar registos** obsoletos ou inválidos.
- **Implementar validações** a nível da base de dados.
- **Proteger as operações** contra erros manuais ou manipulações incorretas.

A camada SOAP será implementada utilizando WCF, fornecendo uma interface estruturada e padronizada para acesso à base de dados.

2.2.1. Endpoints SOAP

Endpoint	Descrição
ValidateUser	Responsável pela verificação dos dados no processo de login
GetUserByid	Responsável por retornar a informação referente a determinado utilizador

Tabela 1 - SOAP Utilizador

Na tabela 1 temos representadas as operações associadas às tabelas da base de dados relacionadas aos utilizadores. A operação booleana *"ValidateUser"* desempenha um papel essencial no funcionamento da aplicação, recebendo como

entrada o email e a senha do utilizador, permitindo que as demais camadas da arquitetura utilizem esses dados de forma segura em operações futuras.

Endpoint	Descrição
GetAllFornecedores	Responsável por retornar uma lista com todos os fornecedores
GetFornecedorById	Responsável por retornar os dados específicos de um fornecedor
AddFornecedor	Responsável por adicionar um fornecedor na base de dados
UpdateFornecedor	Responsável por atualizar os dados de um fornecedor na base de dados
DeleteFornecedor	Responsável por eliminar os dados de um fornecedor da base de dados

Tabela 2 - SOAP Fornecedores

Os *endpoints* apresentados na tabela 2 facilitam a execução das operações *CRUD* relacionadas à gestão de fornecedores, proporcionando um controlo eficiente sobre o ciclo de vida das suas informações.

Service	Descrição
AddEncomenda	Responsável por adicionar uma nova encomenda no sistema
UpdateEncomendaEstado	Responsável por atualizar o estado de uma encomenda entre (P – Pendente | E- Entregue)
GetEncomendaById	Responsável por retornar os dados de uma encomenda através do Id
GetAllEncomendas	Responsável por retornar todas as encomendas presentes na base de dados

Tabela 3 - SOAP Encomendas

Na tabela 3 estão descritos os métodos responsáveis pela gestão das encomendas no sistema. As consultas realizadas para manipular os dados das encomendas foram projetadas para integrar informações provenientes de duas tabelas distintas na base de dados, tornando-as mais complexas em comparação

com as demais operações. Essa abordagem permite consolidar os dados gerais das em encomendas e os detalhes dos itens associados num único conjunto de informações, garantindo uma visão completa e integrada.

Endpoint	Descrição
GetAllProdutos	Responsável por retornar todos os produtos disponíveis na base de dados
GetProdutoByID	Retorna os detalhes de um produto específico com base no seu identificador único (ID).
AddProduto	Responsável por adicionar um novo produto à base de dados
UpdateProduto	Atualiza as informações de um produto existente

Tabela 4 - SOAP Produtos

Na Tabela 4 são descritos os métodos do serviço SOAP relacionados à gestão de produtos. Estes endpoints foram concebidos para permitir a gestão completa do catálogo de produtos no sistema. Combinando funcionalidades de leitura (consulta) e escrita (adição ou atualização), as operações de produtos desempenham um papel crucial na manutenção da integridade dos dados e na disponibilidade do catálogo.

Endpoint	Descrição
GetAllMovimentos	Retorna todos os movimentos de stock registados no sistema
AddMovimento	Regista um novo movimento de stock
GetMovimentoByProduto	Obtém todos os movimentos de stock relacionados a um produto específico
VerificarStockDisponivel	Verifica se existe stock suficiente de um produto específico
GetHistoricoMovimentosByUtilizador	Retorna o histórico de movimentos de stock efetuados por um utilizador específico

Tabela 5 - SOAP Movimentos

Na Tabela 5 são descritos os métodos do serviço SOAP para a gestão de movimentos de stock. Estes endpoints foram projetados para garantir o controlo e

a rastreabilidade das alterações de stock, fundamentais para a gestão de inventário.

2.3 Operações RESTful (Business Layer)

Os serviços RESTful representam a camada de negócio (**Business Layer**) do sistema, atuando como intermediários entre os clientes (aplicações) e os serviços SOAP. Esta camada é responsável por receber as requisições de clientes, validar os dados, aplicar a lógica de negócio necessária e delegar as operações de manipulação de dados à camada SOAP.

Principais características:

- Traduzir as solicitações dos clientes em chamadas para os serviços.
- Validar os dados recebidos, garantindo consistência e conformidade.
- Implementação de autenticação e autorização, garantindo que apenas utilizadores com permissões adequadas possam aceder a determinadas funcionalidades.

2.3.1 Endpoints RESTful

Auth (api/)	Descrição
login	Responsável pela chamada para verificar os dados de autenticação
register	Responsável pela chamada criar novo utilizador

Tabela 6 - RESTFUL Login

Os *endpoints* apresentados na tabela 4 tratam da gestão dos utilizadores da aplicação. Ficam responsáveis pelas chamadas para verificação dos dados na autenticação e criação de novos utilizadores.

Fornecedores (api/)	Descrição
GetFornecedores	Responsável pela chamada para retorno de uma lista de fornecedores
CriarFornecedor	Responsável pela chamada para criar um fornecedor
AtualizarFornecedor	Responsável pela chamada para atualizar um fornecedor
RemoverFornecedor	Responsável pela chamada para remover um fornecedor

Tabela 7 - RESTFUL Forncecedores

Na tabela 5, temos a descrição dos endpoints responsáveis pelas chamadas ao serviço *SOAP*, para lidar com todas as requisições referentes a interações com a tabela de fornecedores da base de dados.

Encomendas (api/)	Descrição
AdicionarEncomenda	Responsável pela chamada para criar uma entrada nas tabelas referentes às encomendas
AtualizarEstado	Responsável pelas alterações na coluna de estado
EncomendasPorId	Responsável pela chamada para retorno de uma encomenda
TodasEncomendas	Responsável pela chamada para retornar todas as encomendas

Tabela 8 - RESTFUL Encomendas

Na tabela 6, temos representados os *endpoints* responsáveis pelas chamadas ao serviço SOAP referentes às tabelas das encomendas. Estes *endpoints* lidam com as operações que são efetuadas em duas tabelas em simultâneo devido à relação direta entre a tabela encomendas e detalhe de encomendas.

Produtos (api/)	Descrição
GetProdutos	Responsável por retornar uma lista de todos os produtos disponíveis na base de dados.
CriarProdutos	Responsável por criar um registo de produto na base de dados.
AtualizarProdutos	Responsável por atualizar as informações de um produto existente.

Tabela 9 - RESTful Produtos

Na tabela 9, são descritos os endpoints que gerem os produtos do sistema. Estes endpoints permitem consultar, adicionar e atualizar informações relacionadas aos produtos, facilitando a gestão eficiente dos mesmos no contexto da base de dados.

Movimentos (api/)	Descrição
GetMovimentos	Responsável por retornar todos os movimentos de stock registados na base de dados.
CriarMovimentos	Responsável por adicionar um novo registo de movimento de stock, indicando entrada ou saída.
GetMovimentoByProduto	Responsável por consultar os movimentos associados a um produto específico.
VerificarStockDisponivel	Responsável por verificar se existe stock suficiente de um produto antes de realizar saídas.
GetMovimentosByUtilizador	Responsável por retornar o histórico de movimentos de stock efetuados por um utilizador específico.

Tabela 10 - RESTful Movimentos

A tabela 10 apresenta os endpoints destinados à gestão de movimentos de stock. Estes métodos asseguram o controlo sobre entradas e saídas de produtos, verificações de stock disponíveis e permitem análises detalhadas dos históricos de movimentação.

2.4 Base de dados

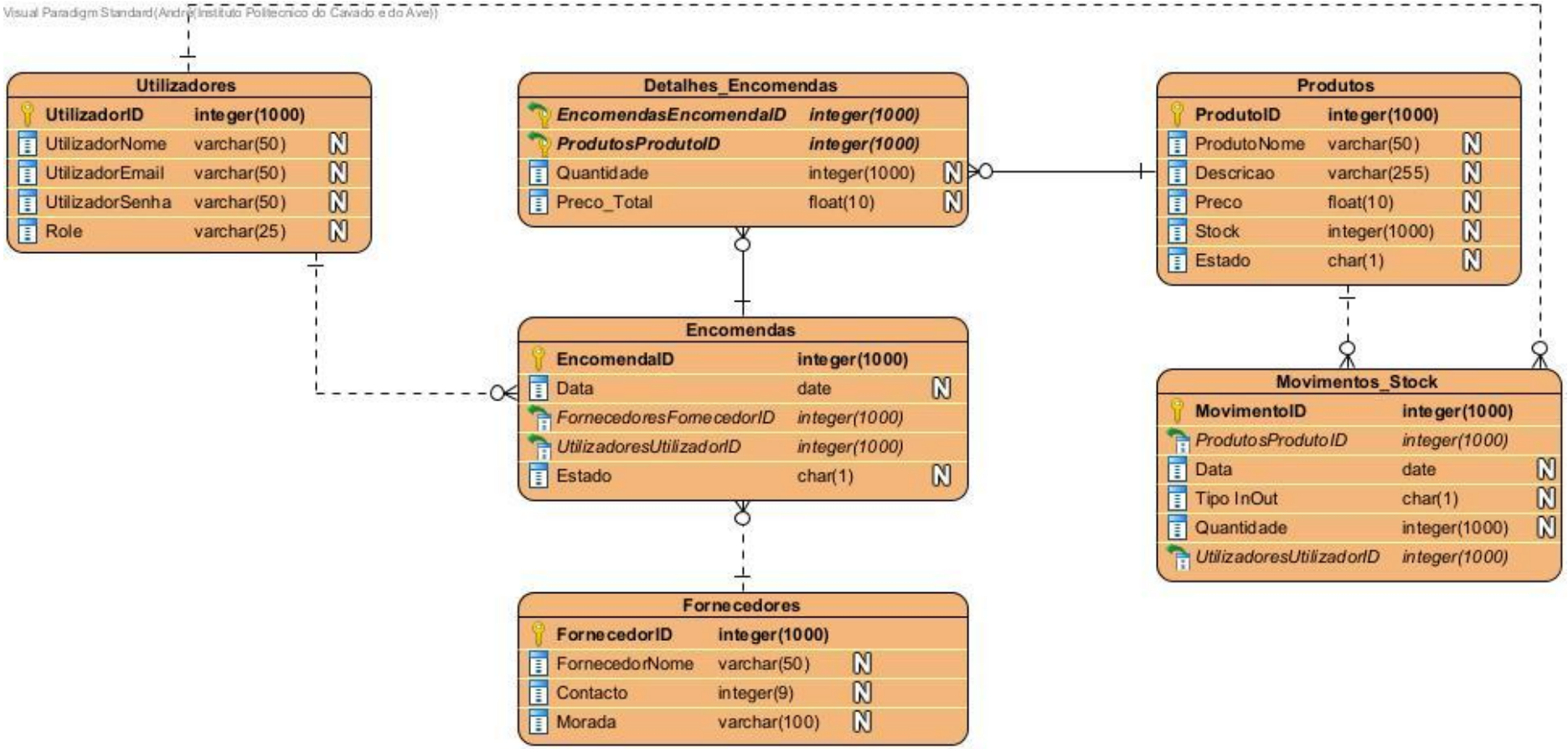

Figura 2 - Diagrama ER

A base de dados é **o repositório central de todas as informações relacionadas com o sistema**. A arquitetura, da base de dados, foi projetada para atender aos principais requisitos funcionais e de negócio, garantindo **integridade e consistência da informação.**

Como podemos observar na figura 2, as tabelas foram estruturadas de forma a atender às operações do sistema.

- Utilizadores: Tabela que armazena a informação sobre os utilizadores que podem aceder ao sistema. Está relacionada com outras tabelas para autenticação, gestão de permissões e registo de ações.
- Produtos: Itens disponíveis no stock, relacionada com os produtos que serão movimentados ou incluídos em encomendas.
- Fornecedores: Armazena a informação de quem envia produtos, associada diretamente às encomendas.
- Encomendas: Representa pedidos realizados para os fornecedores, organiza o processo de aquisição de produtos.
- Detalhes_Encomendas: Tabela intermediária que liga encomendas e produtos, detalhando os itens em cada encomenda.
- Movimentos_Stock: Representa entradas e saídas de produtos do stock, registando as movimentações realizadas pelos utilizadores.

2.4.1. Conexão SOAP-BD

Antes de implementar as operações *CRUD* foi configurada uma ***connection string*** para estabelecer a ligação entre a base de dados e a aplicação. A *connection string* **contém informações sensíveis**, como credenciais de acesso à base de dados, sendo importante existir um **cuidado acrescido no armazenamento.** No caso deste desenvolvimento optamos por armazenar as informações no ficheiro *Webconfig*.

```
<connectionStrings>
<add name="MyDBConnection"
```

```
 connectionString="Data Source=localhost;Initial Catalog=STOCKV1;User
Id=bd;Password=bd;"
providerName="System.Data.SqlClient" />
</connectionStrings>
//Variável criada no serviço para lidar utilizar a conexão
public Service()
{
   _connectionString =
ConfigurationManager.ConnectionStrings["MyDBConnection"].ConnectionString;
}
```

Foi desenvolvido um método para validar o correto funcionamento da conexão com a base de dados. A implementação utiliza a classe *sqlConnection* e o comando *conn.Open()* para estabelecer a conexão. O método retorna uma *string* que informa sobre o estado da conexão, indicando se foi bem-sucedida ou se ocorre uma falha. Esta abordagem simplificou o diagnóstico de problemas relacionados com o acesso à base de dados.

```
   using (SqlConnection conn = new SqlConnection(_connectionString))
   {
      conn.Open();
      return "Conexao com a base de dados bem-sucedida!";
   }
}
catch (Exception ex)
{
   return string.Format("Erro ao conectar: {0}", ex.Message);
}
```

O mesmo modelo foi seguido para testar a conexão entre a API RESTful e a base de dados utilizando o método de teste do SOAP como intermediário para a conexão.

```
 try
 {
    // Chama o método SOAP para testar a conexão
```

```
    string resultado = await _soapClient.TestarConexaoAsync();


    // Retorna a resposta em formato JSON
    return Ok(new { mensagem = resultado });
}
catch (Exception ex)
{
    // Retorna erro com mensagem detalhada em caso de exceção
    return StatusCode(500, new { mensagem = "Erro ao testar conexão", erro = ex.Message });
}
```

3. Codificação

A implementação do sistema aderiu estritamente ao modelo arquitetural previamente delineado na Figura 1, que estabelece o *Data Layer*, contruído sobre uma arquitetura *SOAP*, como **único canal de comunicação** entre a base de dados e as demais aplicações. Esta decisão arquitetural centralizou o acesso a dados, **promovendo a consistência e a manutenção**. Consequentemente, a interação entre o cliente e a aplicação *RESTFUL* ocorre através **da troca de dados em formato *JSON***, que se caracteriza pela sua facilidade de processamento e interpretação por diferentes linguagens. Internamente, a aplicação *RESTFUL* atua como uma camada de adaptação, **traduzindo as requisições *JSON* recebidas em mensagens *XML***, o formato padrão utilizado pelo serviço *SOAP*, para comunicar com o *Data Layer*. Este processo de tradução garante a interoperabilidade entre os dois estilos arquiteturais.

3.1. Codificação SOAP

A camada SOAP garante a conexão direta com a base de dados utilizando comandos SQL simples, **assegurando o acesso eficiente aos dados** e a execução de operações especificas. Cada função desenvolvida nesta camada estabelece uma conexão segura e temporária com a base de dados, utilizando o padrão *using*. Este padrão garante o correto descarte da conexão e a libertação de recursos associados, mesmo em caso de exceções, evitando fugas de memória e outros problemas de desempenho. O excerto de código abaixo ilustra este comportamento.

```
//Abertura da conexão
connection.Open();
//Bloco responsável pela interação com a bd após sair do bloco a conexão é fechada
using (var connection = new SqlConnection(_connectionString))
```

Todos os valores fornecidos como entrada são parametrizados (@), prevenindo riscos como a injeção de SQL. Cada método encapsula uma operação especifica, promovendo modularidade e reutilização do código.

O uso de parâmetros nas consultas SQL garante que os valores fornecidos pelo utilizador não sejam interpretados como parte do comando SQL. Eles tratam automaticamente e de forma segura caracteres especiais evitando que entradas maliciosas manipulem a lógica da query. No seguinte excerto de código demonstramos o que foi explicado, onde cada parâmetro de entrada se encontra parametrizado, tornando as comunicações mais seguras.

```
string query = @"
   UPDATE Fornecedores
   SET FornecedorNome = @FornecedorNome,
      Contacto = @Contacto,
      Morada = @Morada
   WHERE FornecedorID = @FornecedorID";

using (var command = new SqlCommand(query, connection))
{
   command.Parameters.AddWithValue("@FornecedorNome", fornecedor.FornecedorNome);
   command.Parameters.AddWithValue("@Contacto", fornecedor.Contacto);
   command.Parameters.AddWithValue("@Morada", fornecedor.Morada);
   command.Parameters.AddWithValue("@FornecedorID", fornecedor.FornecedorID);

   connection.Open();
   return command.ExecuteNonQuery() > 0;
}
```

3.2. Codificação RESTFUL

Além da função central de intermediar as solicitações ao serviço *SOAP*, esta camada desempenhou um papel significativo na implementação de mecanismos **de autenticação e na integração com APIs externas**, enriquecendo a aplicação com dados relevantes para o contexto do projeto.

Uma parte significativa dos controladores desenvolvidos, dedicou-se à **tradução bidirecional entre os formatos** *RESTFUL(JSON)* e *SOAP(XML)*, esta conversão assegurou a comunicação eficaz entre os diferentes paradigmas arquiteturais, permitindo que clientes *REST* consumissem serviços baseados em *SOAP*. Para garantir a robustez da aplicação, implementou-se um tratamento robusto de exceções através de bloco *try-catch* em todos os métodos relevantes. Adicionalmente, o uso de modelos desacoplou a representação de dados na API da estrutura interna dos objetos *SOAP*, promovendo a manutenibilidade e a evolução independente das duas camadas. A segurança da aplicação foi reforçada com o uso de *[authorize]*, restringindo o acesso a determinados métodos apenas a utilizadores autenticados e autorizados, de acordo com as políticas de segurança definidas. Finalmente, a documentação da API foi integralmente gerada a partir de comentários XML, utilizando o *Swagger/OpenAPI,* o que facilitou a compreensão e o consumo da API por parte de outras aplicações.

3.2.1. Controladores

Os controladores desenvolvidos, têm como **principal função servir como ponte entre as aplicações existentes**, permitindo que a correta comunicação entre as aplicações de forma a **corretamente gerir informação**.

O desenvolvimento dos controladores, utilizou uma construção similar entre os vários componentes em que uma **parte crucial da lógica** envolve a conversão entre modelos. Os dados são recebidos nas requisições *RESTFUL* (No formato *JSON*, representados por uma classe criado no modelo), **são convertidos para o formato esperado** pelo *SOAP* (representado pela classe *SOAPServiceReference.Fornecedor*) antes de serem enviados para o serviço *SOAP*. O **processo inverso ocorre quando o controlador recebe dados** do serviço *SOAP* e os converte para o formato *FornecedorModel* antes de serem enviados na resposta *RESTFUL*.

```
try {
 // ... (validação) // Converte o modelo RESTful para o modelo SOAP
var fornecedorSoap = new SOAPServiceReference.Fornecedor
{
```

```
FornecedorNome = fornecedor.Nome,
Contacto = fornecedor.Contacto,
Morada = fornecedor.Morada }; // Chama o método SOAP
var resultado = await _soapClient.AddFornecedorAsync(fornecedorSoap);
 // ... (tratamento do resultado) }
catch (Exception ex)
{ // ...
}
}
```

O exemplo acima, e utilizando o controlador fornecedor para demonstrar o conceito, demonstra uma chamada para retornar os dados dos fornecedores, a função *GET*, **primeiro faz a chamada ao serviço *SOAP* e só depois ocorre a conversão dos dados**, enquanto na função abaixo verificamos o inverso. No caso do *Add*, **primeiro é feita a conversão dos dados e só depois e chamado o serviço**.

```
{ try {
// ... (validação) // Converte o modelo RESTful para o modelo SOAP
 var fornecedorSoap = new SOAPServiceReference.Fornecedor
 { FornecedorNome = fornecedor.Nome,
Contacto = fornecedor.Contacto,
 Morada = fornecedor.Morada };
// Chama o método SOAP
var resultado = await _soapClient.AddFornecedorAsync(fornecedorSoap);
 // ... (tratamento do resultado) }
catch (Exception ex)
 { // ... }
}
```

3.2.2. Tokens de validação (JWT)

Para o desenvolvimento das funcionalidades de autenticação da aplicação, optamos pela utilização de *Tokens JWT*, para a troca segura de informações entre os componentes da aplicação, em formato *JSON*. O padrão *JWT* é amplamente utilizado para **autenticação** e **autorização** em aplicações modernas.

O *Token* é composto por três partes, separadas por pontos (.):

- **Cabeçalho**, informação sobre o tipo de token e o algoritmo de assinatura;
- **Corpo**, contem informações do utilizador e *claims*;
- **Assinatura**, garante que o *token* não foi alterado;

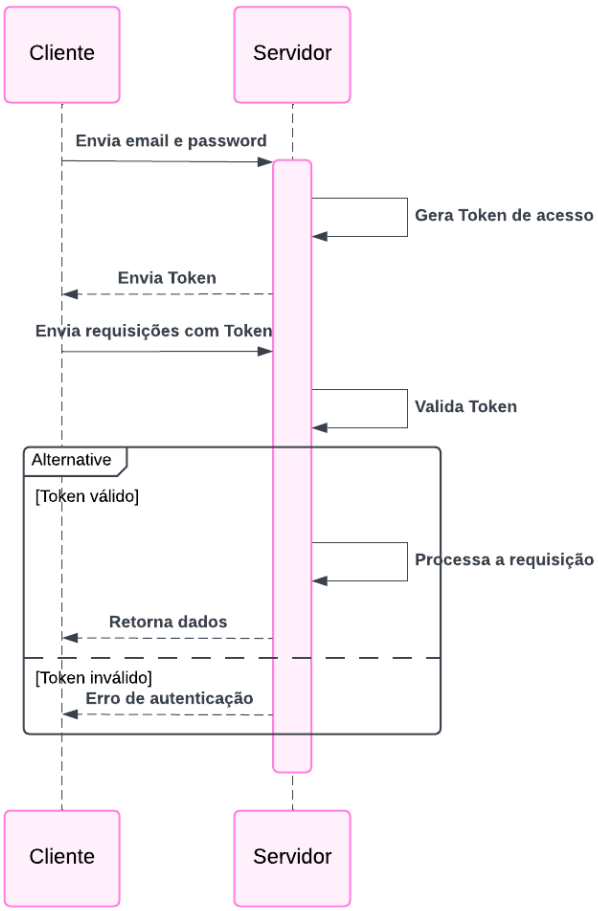

Figura 3 - Processo Autenticação (Tokens JWT)

Como podemos observar na figura 3, o processo de autenticação foi criado recorrendo à utilização de *tokens* para verificação dos dados de acesso e recuperação do papel do utilizador na aplicação, de forma a permitir/negar acesso a chamadas *REST*.

3.2.2.2. Configuração do servidor (*Program.cs)*

Antes da criação do serviço de autenticação, foi necessário proceder à configuração do servidor para lidar corretamente com *tokens JWT* e assim garantir a segurança e a proteção das operações exposta pela API. A biblioteca utilizada para a configuração foi a *JWTBearer*.

1. Configuração do serviço *AuthService* como responsável por criar e validar os *tokens*. Definição do tempo de expiração e chave de acesso.
2. Registo do serviço acima mencionado na aplicação
3. Configuração da autenticação para validar *tokens* recebidos. Esta configuração garante que apenas *tokens* no formato correto sejam aceites. (Demonstração das configurações abaixo)

```
.AddJwtBearer(x =>
{ x.RequireHttpsMetadata = false;
x.SaveToken = true;
 x.TokenValidationParameters = new TokenValidationParameters
 {ValidateIssuerSigningKey = true,
IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSecret)),
ValidateIssuer = false, ValidateAudience = false
 }; });
```

4. Configuração das políticas de autorização, para restringir o acesso a diferentes endpoints. (Exemplo abaixo)

```
options.AddPolicy("Admin", policy => policy.RequireRole("admin"));
```

3.2.2.3. Configuração do serviço de Autenticação (*AuthService.cs)*

1. Foram criadas duas variáveis globais para armazenar a chave secreta para assinar e validar os tokens e definir o tempo de expiração dos tokens. Optamos por armazenar a informação no ficheiro *appsetings.json*.

```
private readonly string _privateKey;
private readonly int _tokenExpirationMinutes;
```

2. De seguida, iniciamos o construtor com as variáveis, do excerto de código acima para garantir que os valores essenciais são configurados na criação.
3. Passamos então para a **função que gera o *token***:
    - Codificação da chave *private key* em bytes, necessário para a assinatura do *token*;
    - Gera as credencias de assinatura do *token*, usando o algoritmo *HMAC SHA265*;
    - Define informações do *token*, email e o papel do utilizador;
    - Configuração do *token*, com as informações do utilizador, assinatura com a chave e tempo de expiração. (função demonstrada no excerto de código abaixo);

```
var tokenDescriptor = new SecurityTokenDescriptor
{Subject = new ClaimsIdentity(claims),
 Expires = DateTime.UtcNow.AddMinutes(_tokenExpirationMinutes),
 SigningCredentials = credentials };
```

    - Por fim é criado o token utilizando *JwtSecurityTokenHandler* e retornado ao cliente como uma string.

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

4. Foi criado um método de validação para completar o método de gerar *tokens* anteriormente descrito. Ele verifica a assinatura do token e o tempo até expirar. Caso o *token* seja válido retorna um objeto *ClaimsPrincipal* contendo as informações do utilizador, se o *token* for inválido, o método retorna *null*.
    4.1. Estrutura dos parâmetros de validação:

```
var tokenValidationParameters = new TokenValidationParameters {
ValidateIssuerSigningKey = true, // Valida a assinatura do token IssuerSigningKey
```

```
= new SymmetricSecurityKey(key), // Define a chave usada para validar a assinatura ValidateIssuer = false, // Ignora a validação do emissor
ValidateAudience = false, // Ignora a validação do público
ValidateLifetime = true, // Valida se o token está expirado
ClockSkew = TimeSpan.Zero // Não permite tolerância na expiração };
```

3.2.2.4. Controlador de autenticação *(AuthController)*

O controlador de autenticação implementa o *endpoint* para autenticação dos utilizadores. Interage com o *authservice*, descrito anteriormente, para gerar os tokens e o com o *SOAP* para a validação das credenciais.

Utilizando o *SOAP* faz a verificação do email e password retornando um booleano como resposta à tentativa de autenticação. Em caso de autenticação correta obtém os detalhes do utilizador e utiliza o *role* para gerar o *token*. Nesta fase é armazenado igualmente o Id do utilizador numa variável global para posterior utilização.

3.3. Chamada de API externa

Dado que o contexto da nossa aplicação é de natureza académica, e considerando que a maioria das opções disponíveis para **operações mais avançadas em API externas requer subscrições ou acessos pagos**, optámos por implementar uma solução que demonstre o conceito de integração com uma API externa de forma simplificada. Assim conseguimos manter foco no objetivo principal de **demonstrar o funcionamento de chamadas a serviços externos**.

A solução foi projetada para realizar uma chamada básica a uma API pública que permita operações gratuitas dentro das limitações do plano base. **Por isso optamos pela API, *ExchangeRate.Host*, que oferece o endpoint */Latest*** para obter a taxa de câmbio de uma especifica em relação ao euro.

1. Na primeira fase foi preciso definir serviço para fazer as chamadas *HTTP*, estabelecer o *url* base da *API* e recolher a chave da *API* fornecida. Estas duas informações são fornecidas pelo provedor da *API* e optamos por armazenas no ficheiro *Appsetting.json*. (excerto de código abaixo)

```
//AppSetting.json
"CurrencyApiSettings": {
  "BaseUrl": "https://api.exchangerate.host",
  "AccessKey": "b927e45e7663b7380a3e0cac04e761a1"
},
//CurrencyService
  private readonly HttpClient _httpClient; //definido no program.cs
  private readonly string _baseUrl;
  private readonly string _accessKey;
```

2. De seguida procedemos à construção do *URL* para a chamada da API utilizando o *URL* base, endpoint */latest*, a chave de acesso e passando o parâmetro correspondente à moeda para a qual precisamos de informações. (Função utilizada abaixo)

```
var url = $"{_baseUrl}/latest?access_key={_accessKey}&symbols={currency}";
```

3. Por fim, foi feita a requisição *HTTP GET* para a *URL* anteriormente criada, caso a resposta não seja bem-sucedida é lançada uma exceção, do mesmo modelo anteriormente explicada na parte do relatório destinada aos *Endpoints*.
4. No final do processo é processada a resposta, sendo lido o *JSON* de resposta e extraída e armazenada a taxa de cambio como um decimal. Sendo retornado um objeto *CurrencyRate* contendo a moeda, taxa e a hora de consulta.

3.4. Codificação cliente

A aplicação cliente foi desenvolvida apenas como prova de conceito contendo apenas simples visualizações da chamada dos métodos sem interações nem funcionalidades muito complexas.

Devido a esse facto neste capítulo vamos apenas destacar o código e funcionalidades que envolvam a interações com a API *RESTFUL*.

Optamos pela criação de um ficheiro *GlobalConfig.cs* centralizando a configuração global do cliente neste ponto.

1. Começamos por definir uma instância única e global de *HTTPClient* contendo como endereço base o url da nossa API.

```
public static HttpClient HttpClient { get; } = new HttpClient
 { BaseAddress = new Uri("https://localhost:7292/api/") // Config da URL base
};
```

2. Criamos uma propriedade estática para armazenar e gerir os tokens, sempre que for definido o valor é configurado automaticamente o cabeçalho no formato esperado pela API.

```
HttpClient.DefaultRequestHeaders.Authorization=
new AuthenticationHeaderValue("Bearer", _jwtToken);
```

No formulário de login foi necessário implementar a logica necessária para corretamente validar as credenciais de acesso e criar o token necessários para interações futuras.

1. Criação de um objeto contendo o email e a senha do utilizador, conversão do objeto para *JSON* e empacotamento do mesmo numa instância.
2. Envio da requisição para o para o endpoint de login.

```
var response = await GlobalConfig.HttpClient.PostAsync("auth/login", content);
```

3. Processamento e de serialização da resposta, para posteriormente ser armazenada nas variáveis globais apresentadas acima.

```
if (response.IsSuccessStatusCode)
{ var responseContent = await response.Content.ReadAsStringAsync();
 Var tokenResponse=
JsonSerializer.Deserialize<TokenResponse>(responseContent, new
JsonSerializerOptions
```

```
{ PropertyNameCaseInsensitive = true });
(...)
GlobalConfig.JwtToken = tokenResponse.Token;
GlobalConfig.UserID = tokenResponse.UserId;
```

Nos formulários para lidar com as várias tabelas da base de dados é utilizada a mesma conexão explicada anteriormente, sendo posteriormente chamados os endpoints necessários para lidar com as operações.

Em cada função é feita uma chamada à API e posteriormente conforme o tipo de operação pretendida são serializados ou desserializados os dados, por fim é feita a verificação dos dados e resolvida a operação. Abaixo temos excertos do código com as operações efetuadas.

1. Chamada a um endpoint

```
var response = await
GlobalConfig.HttpClient.PostAsync("Fornecedores/RemoverFornecedor",
content);
```

2. Serialização de dados

```
// Serialização para JSON var jsonContent = JsonSerializer.Serialize(newSupplier);
// Configura o conteúdo da requisição
var content = new StringContent(jsonContent, Encoding.UTF8, "application/json");
```

3. Desserialização de dados

```
// Lê o conteúdo da resposta JSON
var content = await response.Content.ReadAsStringAsync();
// Desserializa o JSON para uma lista de objetos Supplier
var suppliers = JsonSerializer.Deserialize<List<Supplier>>(content, new
JsonSerializerOptions
{ PropertyNameCaseInsensitive = true // Ignora diferenças entre maiúsculas e
minúsculas });
```

4. Testes

O processo de testes realizados visou validar o correto funcionamento dos diferentes componentes da aplicação, garantir a robustez, funcionalidade e integração entre os sistemas, bem como verificar a conformidade dos requisitos definidos.

Para validar os métodos e *endpoints* de cada camada, foram utilizadas ferramentas como o SOAP.UI, Postman e Swagger, além de testes diretos no C#. Cada componente foi testada individualmente e, posteriormente, submetido a testes de integração para assegurar a integração entre os módulos.

4.1. Testes SOAP

Todos os testes efetuados sobre a camada SOAP utilizaram a aplicação SOAP.UI como forma de verificar a funcionalidade. A escolha desta aplicação prendeu-se com o facto de visualmente conseguirmos ver como são mensagens XML de *request* e *response*.

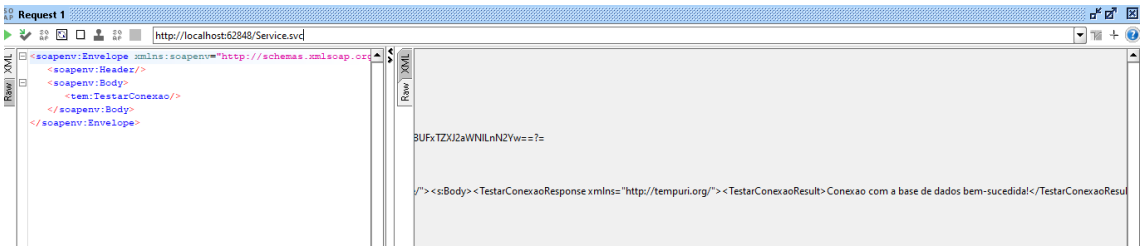

Figura 4 - Teste Conexão (SOAP.UI)

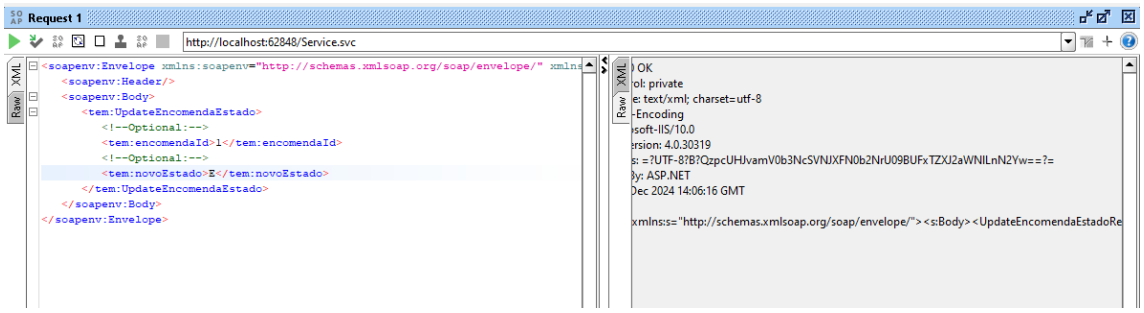

Figura 5 - Teste Atualizar fornecedor (SOAP.UI)

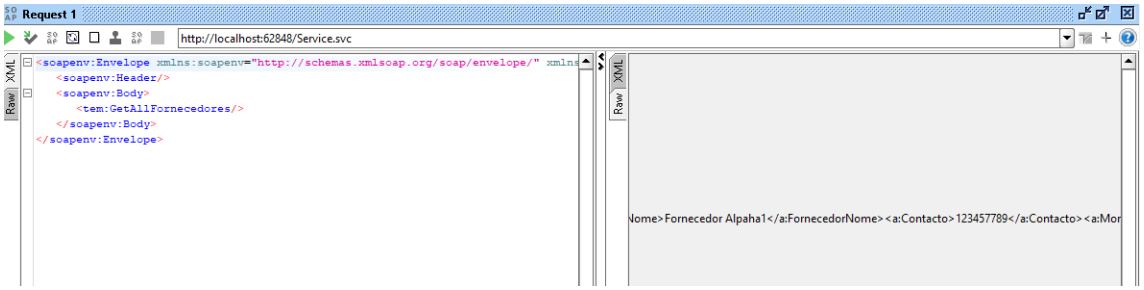

Figura 6 - Teste GetFornecedores (SOAP.UI)

Como podemos verificar nas figuras acima (4,5 e 6), o retorno dos métodos foi o esperado, sendo necessário durante a bateria de testes efetuada, fazer ajustes na base de dados principalmente no tipo de dados criados que geraram problemas de conversão de dados.

4.2. Testes RESTFUL

De forma a explorarmos diferentes opções para efetuar testes aplicacionais e assim alargar os nossos horizontes neste capítulo optamos por utilizar duas formas diferentes de efetuar os testes da aplicação *RESTFUL*. Por isso algumas funções foram testadas usando a aplicação Postman enquanto as restantes recorrendo ao Swagger.

Nas figuras a seguir (7 e 8), podemos verificar dois exemplos de chamadas utilizadas para testar a API recorrendo à aplicação Postman.

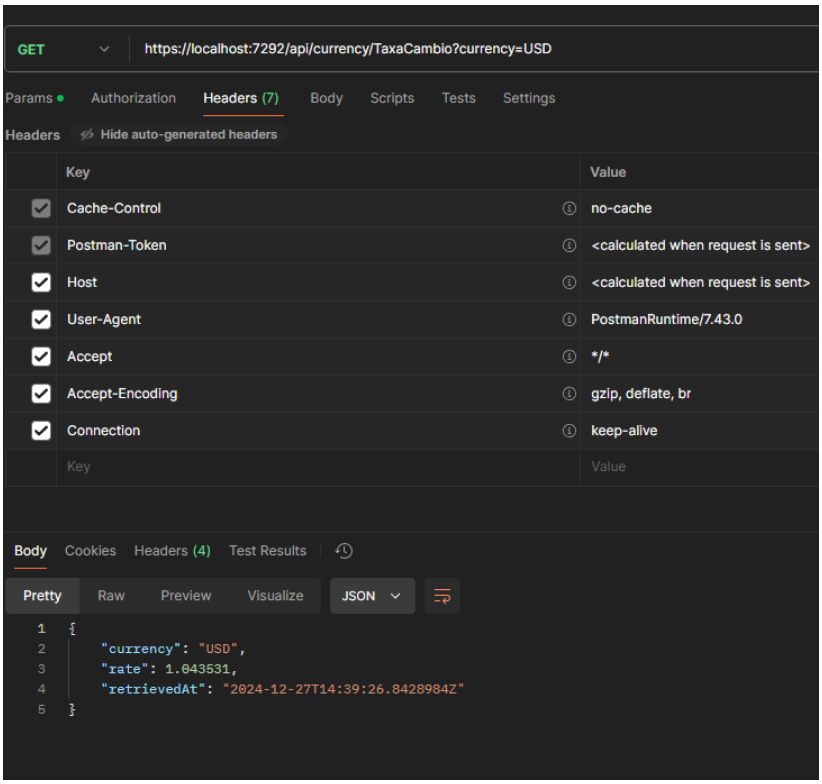

Figura 7 - Teste Cambio (Postman)

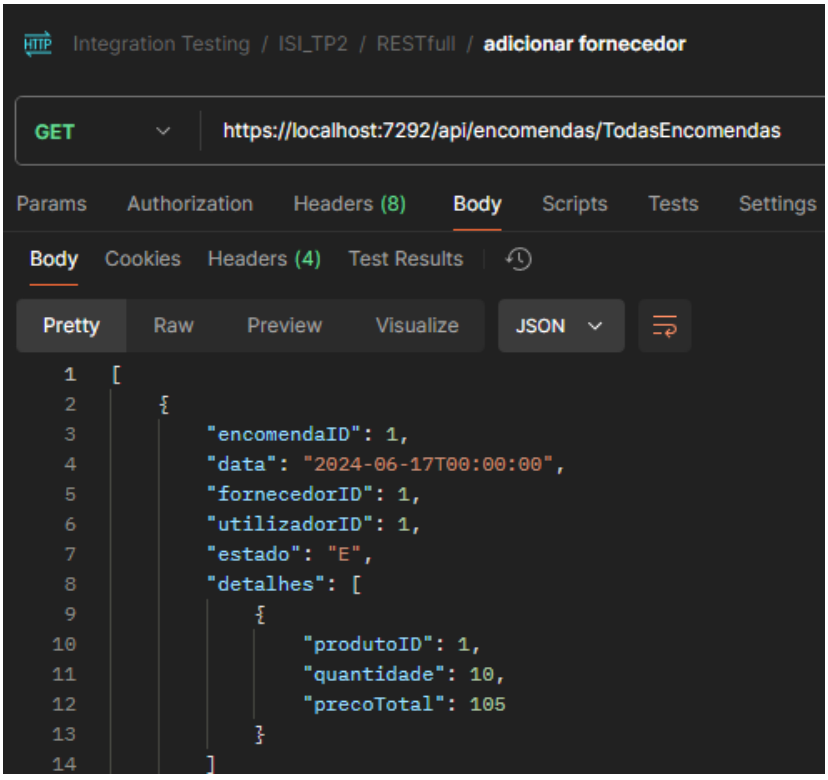

Figura 8 - Teste GET (Postman)

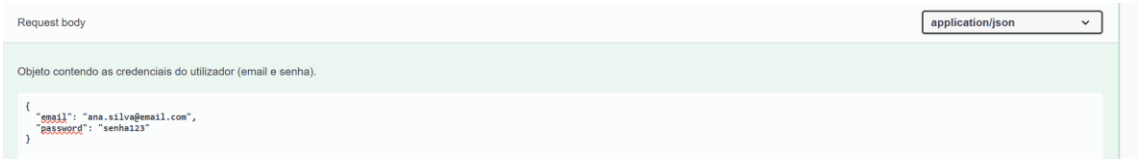

Figura 9 - Teste Entrada de dados Login (Swagger)

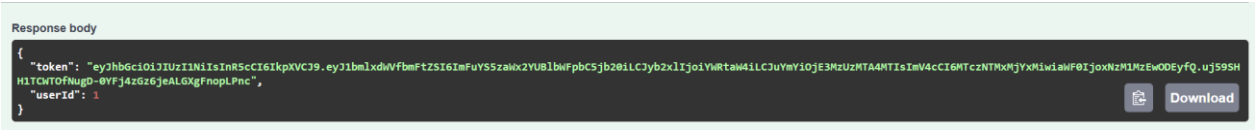

Figura 10 - Resposta com o Token (Swagger)

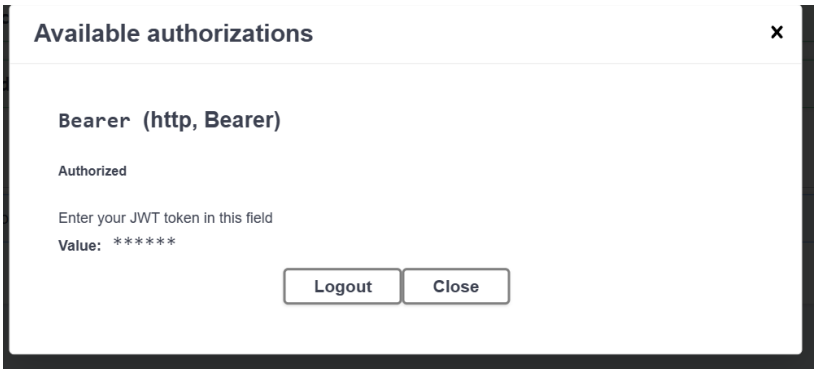

Figura 11 - Inserção do Token (Swagger)

Maioritariamente dos testes efetuados recorreram ao Swagger, devido à facilidade de introdução do token para autenticação (figura 11), após recorrer ao método de login e a aplicação devolver o token de acesso (figuras 9 e 10), introduzimos-lho nos métodos que precisam de autenticação e assim podemos fazer vários testes aplicacionais como demonstrado abaixo na figura 12.

Figura 12 - Teste conexão com Autenticação (Swagger)

4.3. Testes Cliente

Para garantir o correto funcionamento dos métodos desenvolvidos na API RESTFUL e validar a interação entre o cliente e o servidor, foi realizado um conjunto abrangente de testes. Esses testes foram concebidos para verificar a funcionalidade, robustez e confiabilidade das operações expostas pelo REST, cobrindo cenários de sucesso e erro. Demonstração dos testes efetuados nas imagens abaixo. (Figura 13 e 14)

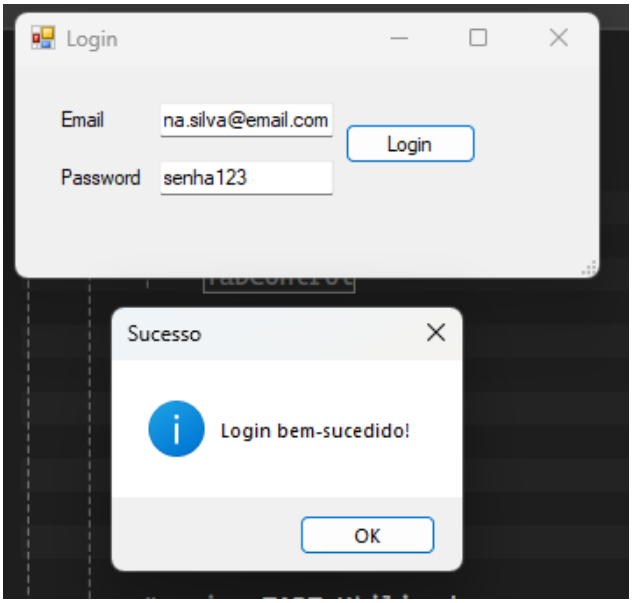

Figura 13 - Tentativa de Login (Cliente)

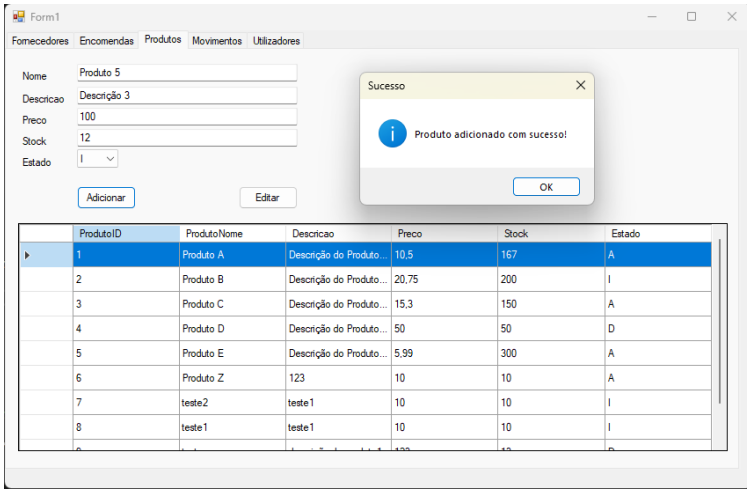

Figura 14 - Adição de produto (Cliente)

5. Conclusão

O projeto desenvolvido representa uma solução funcional para gestão de stock, explorando o uso combinado de tecnologias SOAP e RESTFUL API. Esta abordagem híbrida permitiu-nos maximizar a interoperabilidade e flexibilidade do sistema, enquanto mantivemos a segurança e o controlo sobre os dados através do uso de tokens JWT para autenticação.

Através da utilização de ferramentas como SOAP.UI, Postman e Swagger, garantimos que todas as funcionalidades fossem desenvolvidas e testadas, tanto individualmente quanto em conjunto. Este processo assegurou a consistência e o correto funcionamento das operações em todas as camadas do sistema.

A camada SOAP foi essencial para fornecer acesso direto e seguro aos dados da base de dados SQL Server. Por outro lado, a API RESTFUL desempenhou um papel central na interface com os clientes, traduzindo as operações SOAP para um formato JSON mais acessível e moderno.

Adicionalmente, a integração com uma API externa demonstrou a capacidade do sistema de consumir serviços externos, um requisito fundamental para soluções modernas e integradas. Mesmo com as limitações das APIs gratuitas, foi possível demonstrar o conceito de uma chamada bem-sucedida a um serviço externo.

Este trabalho permitiu-nos compreender e aplicar os conceitos ensinados ao longo do semestre sobre os conceitos essenciais de integração de sistemas, autenticação segura e boas práticas de desenvolvimento em aplicações multicamada. Como conclusão, o projeto atingiu os objetivos propostos, destacando-se como uma solução eficiente e escalável para gestão de stocks.

6. Bibliografia

aws. (s.d.). Obtido de aws, amazon: https://aws.amazon.com/pt/what-is/restful-api/#:~:text=A%20API%20RESTful%20é%20uma,terceiros%20para%20executar%20várias%20tarefas.

Baltieri, A. (s.d.). *https://balta.io/blog/aspnet-core-autenticacao-autorizacao*. Obtido de ASP.NET Core - Autenticação e Autorização: https://balta.io/blog/aspnet-core-autenticacao-autorizacao?utm_source=chatgpt.com

Bastos, R. Q. (2023). *GoDaddy*. Obtido de https://www.godaddy.com/resources/br/artigos/o-que-e-framework-em-programacao

Clemente, M. (s.d.). *O que é e como usar uma Query*. Obtido de rockcontent.com: https://rockcontent.com/br/blog/query/

cloudflare. (s.d.). Obtido de https://www.cloudflare.com/pt-br/learning/security/api/what-is-api-endpoint/

Cristiano. (2016). *SoapUI: testes de Web Services rápido e descomplicado*. Obtido de DEVMEDIA: https://www.devmedia.com.br/soapui-testes-de-web-services-rapido-e-descomplicado/37461#:~:text=SoapUI%20%C3%A9%20uma%20ferramenta%20open,consumir%20e%20testar%20Web%20Services.&text=r%C3%A1pido%20e%20descomplicado-,SoapUI%20%C3%A9%20uma%20ferramenta%20ope

IBM. (s.d.). *JSON Web Token (JWT)*. Obtido de www.ibm.com: https://www.ibm.com/docs/pt-br/cics-ts/6.x?topic=cics-json-web-token-jwt

Meier, J. D. (2016). Obtido de https://pt.stackoverflow.com/questions/22469/qual-a-diferença-entre-aplicações-multi-layer-e-multi-tier#:~:text=Business%20Layer%20(Regras%20de%20Negócio)%3A%20O

%20agrupamento%20lógico%20de,sejam%20fornecidos%20por%20um%20webService.

O que é Connection String. (s.d.). Obtido de Programar sem codigo: https://programarsemcodigo.com.br/glossario/o-que-e-connection-strings/

POSTMAN. (s.d.). *What is Postman?* Obtido de www.postman.com: https://www.postman.com/product/what-is-postman/

Rodrigues, L. (s.d.). *Entenda o que é Swagger, e sua aplicação prática!* Obtido de ACADEMIA TECH: https://academiatech.blog.br/o-que-e-swagger/

rwestMSFT, & MikeRayMSFT. (2024). *O que é o SQL Server?* Obtido de learn.Microsoft: https://learn.microsoft.com/pt-br/sql/sql-server/what-is-sql-server?view=sql-server-ver16

Softplan, A. S.-e. (2021). Obtido de https://sgpe.sea.sc.gov.br/capdoc/front-end-e-back-end-o-que-sao-e-quais-as/

Thiago. (2008). *Introdução aos Stored Procedures no SQL Server*. Obtido de DEVMEDIA: https://www.devmedia.com.br/introducao-aos-stored-procedures-no-sql-server/7904

TOTVS, E. (2023). *O que é token: como funciona, tipos e para que serve?* Obtido de TOTVS: https://www.totvs.com/blog/gestao-para-assinatura-de-documentos/o-que-e-token/