

# Agenda de Contatos

Realizado por: Irishédio Immbar-220000882

Disciplina: **Integração de Sistemas**

Professor: Professor Filipe Madeira

## **Índice**

- 1. Introdução**
  - 1.1 Objetivos
  - 1.2 Enquadramento
  - 1.3 Tecnologias Utilizadas
- 2. Metodologia**
  - 2.1 Abordagem de Desenvolvimento
- 3. Implementação**
  - 3.1 Serviço REST
    - 3.1.1 Endpoints disponíveis
    - 3.1.2 Armazenamento de dados
    - 3.1.3 Integração com o Cliente Web
    - 3.1.4 Testes e Validação
  - 3.2 Serviço SOAP
    - 3.2.1 Métodos Disponibilizados
    - 3.2.2 Conversão de Dados e Armazenamento
    - 3.2.3 Integração com o Cliente Web
    - 3.2.4 Testes com Postman
  - 3.3 Serviço GraphQL
    - 3.3.1 Estrutura do Schema
    - 3.3.2 Queries Personalizadas
    - 3.3.3 Integração com o Cliente Web
    - 3.3.4 Testes Realizados
  - 3.4 Serviço gRPC
    - 3.4.1 Arquitetura do Serviço
    - 3.4.2 Cliente Tkinter com Streaming
    - 3.4.3 Considerações
- 4. Exportação e Importação de Dados**
  - 4.1 Formatos Suportados
  - 4.2 Exportação de Dados
  - 4.3 Importação de Dados
  - 4.4 Interface Gráfica de Importação/Exportação
- 5. Conclusão**

# 1. Introdução

## 1.3 Tecnologias Utilizadas

O desenvolvimento do projeto recorreu a um conjunto diversificado de tecnologias modernas, organizadas em função do seu propósito no sistema:

### Linguagens de Programação

- **Python:** Utilizado para o desenvolvimento dos servidores REST, SOAP, GraphQL e gRPC, pela sua versatilidade, robustez e excelente suporte a bibliotecas para integração de sistemas.
- **JavaScript (Node.js):** Utilizado no desenvolvimento do cliente web, pela sua eficiência no tratamento de requisições HTTP e integração com APIs externas.

### Frameworks e Bibliotecas

- **Flask:** Microframework em Python usado para a implementação dos serviços REST e GraphQL.
- **Spyne:** Framework utilizada para o desenvolvimento do serviço SOAP.
- **Graphene:** Biblioteca Python utilizada para construção do schema GraphQL e respetivas resolvers.
- **gRPC + Protocol Buffers:** Framework de RPC de alto desempenho, utilizado para criar serviços eficientes baseados em streaming.
- **Express.js:** Framework minimalista para Node.js, utilizada para servir as páginas web e criar rotas de integração com os serviços.
- **Axios:** Biblioteca JavaScript para realizar chamadas HTTP no cliente web.
- **Tkinter:** Biblioteca gráfica nativa do Python utilizada para o desenvolvimento do cliente gRPC com interface desktop.

### Formato de Dados

- **JSON:** Principal formato de troca de dados entre cliente e servidor.
- **XML:** Suportado no serviço SOAP para exportação e importação de dados.
- **Base64:** Utilizado para codificação de ficheiros durante o processo de importação via SOAP.

# 2. Metodologia

O desenvolvimento do sistema seguiu uma abordagem iterativa e incremental, com foco na integração de múltiplas tecnologias de serviços web. Cada componente foi concebido, desenvolvido e testado de forma modular, permitindo validar funcionalidades isoladamente e depois em conjunto. A metodologia adotada garantiu a interoperabilidade entre diferentes padrões de comunicação e formatos de dados.

## 2.1 Abordagem de Desenvolvimento

O projeto foi estruturado em quatro etapas principais:

### 1. Planeamento

- Definição dos requisitos funcionais e técnicos.
- Escolha das tecnologias a utilizar, assegurando o cumprimento das exigências da unidade curricular.

### 2. Implementação modular

- Cada serviço (REST, SOAP, GraphQL e gRPC) foi implementado em ficheiros separados, respeitando o princípio da separação de responsabilidades.
- Foram utilizados ficheiros JSON como base comum de persistência de dados, sem recurso a bases de dados relacionais.

### 3. Desenvolvimento do cliente web e desktop

- O cliente web, construído com HTML, CSS e JavaScript (Node.js/Express), integra-se com os serviços REST, SOAP e GraphQL.
- O cliente gRPC foi implementado com Python e interface gráfica via Tkinter, utilizando comunicação por streaming.

### 4. Testes e Validação

- Todos os serviços foram testados individualmente com Postman ou interfaces gráficas (GraphQL/Tkinter).
- Foram realizados testes de integração com o cliente web para garantir a comunicação correta com os serviços.

## 3. Implementação

Nesta secção, descreve-se detalhadamente a implementação dos serviços desenvolvidos no sistema, com destaque para a estrutura dos endpoints, fluxos de funcionamento, exemplos práticos e integração com o cliente. A aplicação encontra-se organizada em dois grandes blocos: o **servidor**, responsável pelos serviços web, e o **cliente**, que consome esses serviços através de uma interface web ou gráfica.

### 3.1 Serviço REST

O serviço REST foi implementado com **Flask** em Python. Este serviço expõe três operações principais: listagem de contatos, criação de novos registros e eliminação de contatos existentes.

#### Endpoints disponíveis

Método	Endpoint	Descrição
GET	/contatos	Retorna a lista completa de contatos
POST	/contatos	Adiciona um novo contato

Método	Endpoint	Descrição
DELETE	/contatos/<id>	Elimina um contato específico pelo ID

### Exemplo de estrutura JSON de um contato

```
{
  "id": "1",
  "nome": "Joana Silva",
  "email": "joana.silva@email.com",
  "telefone": "912345678"
}
```

### Armazenamento de dados

Todos os contatos são armazenados num ficheiro JSON localizado em `data/contatos.json`. A manipulação do ficheiro é feita com funções auxiliares (`carregar_contatos` e `salvar_contatos`) que garantem leitura e escrita seguras no formato correto.

### Integração com cliente Web

O cliente web, desenvolvido com Node.js e Express, realiza chamadas REST através do módulo `axios`. As rotas implementadas em `rest.js` consomem os endpoints REST e apresentam os dados dinamicamente numa interface HTML moderna. A interface permite:

- Carregar todos os contatos (GET)
- Adicionar novos registros com formulário (POST)
- Remover contatos com botão interativo (DELETE)

### Testes e validação

O serviço foi amplamente testado com **Postman**, com chamadas simuladas de todos os métodos. A validação de funcionamento foi também reforçada através da interface do cliente web.

## 3.2 Serviço SOAP

O serviço SOAP foi desenvolvido com a biblioteca **Spyne**, utilizando Python como linguagem base. Este serviço expõe uma interface formal baseada em **WSDL** (Web Services Description Language), permitindo a interoperabilidade com diferentes clientes, independentemente da linguagem ou tecnologia utilizada.

## Métodos disponibilizados

Através do serviço SOAP foram disponibilizados os seguintes métodos:

Método	Descrição
<code>listar_contatos()</code>	Retorna uma lista de contatos em formato de string
<code>exportar_json()</code>	Devolve os dados do ficheiro JSON como string
<code>exportar_xml()</code>	Converte os dados JSON para XML e devolve o conteúdo como string
<code>importar_json_base64()</code>	Recebe um ficheiro JSON codificado em Base64 e atualiza o ficheiro local
<code>importar_xml_base64()</code>	Recebe um ficheiro XML em Base64, converte-o para JSON e atualiza os dados

### Formato de retorno do método `listar_contatos()`

Este método retorna uma lista de strings formatadas, onde cada linha representa um contato:

```
"1 - Ana Pereira - ana@email.com - 913456789"  
"2 - João Costa - joao@email.com - 912345678"
```

## Armazenamento de dados e conversão

O serviço utiliza o mesmo ficheiro `contatos.json` para leitura e escrita. As conversões entre JSON e XML são realizadas por funções utilitárias localizadas no módulo `utils/conversor_funcoes.py`. Estas funções garantem integridade na transformação dos dados, respeitando a estrutura esperada.

## Integração com cliente web

O cliente web comunica com o serviço SOAP utilizando o módulo `soap` da biblioteca `strong-soap`. As chamadas SOAP são feitas de forma assíncrona, e as respostas são tratadas dinamicamente para:

- Listar contatos
- Exportar os dados em ficheiros para download (`.json` e `.xml`)
- Importar ficheiros selecionados pelo utilizador, codificados automaticamente em Base64

A interface gráfica HTML permite ao utilizador:

- Carregar contatos via SOAP
- Efetuar download dos dados (JSON/XML)
- Importar novos dados através de upload de ficheiros

## Testes com Postman

Os métodos SOAP foram testados diretamente no **Postman** através do envio de requisições SOAP com payloads XML devidamente formatados. A interface gráfica do cliente também serviu como meio de verificação funcional de todas as operações.

## Exemplo de chamada SOAP no Postman

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="agenda.contatos">
  <soapenv:Body>
    <tns:listar_contatos/>
  </soapenv:Body>
</soapenv:Envelope>
```

## Considerações

O serviço SOAP demonstra a capacidade de manipular ficheiros complexos, realizar transformações estruturadas de dados e manter compatibilidade com ferramentas empresariais mais antigas, mantendo ainda integração moderna com interfaces web.

## 3.3 Serviço GraphQL

O serviço **GraphQL** foi implementado com recurso à biblioteca **Graphene** em Python, utilizando o framework **Flask** para a exposição do endpoint `/graphql`. Este serviço permite consultas dinâmicas e eficientes sobre os dados da agenda de contatos, com controlo total do que é requisitado pelo cliente.

### Estrutura do Schema

O schema GraphQL define o tipo `Contato` com os seguintes campos:

```
type Contato {
  id: String
  nome: String
  email: String
  telefone: String
}
```

Através da classe `Query`, foram definidos dois tipos principais de operações:

Query	Descrição
<code>contatos</code>	Devolve a lista completa de contatos existentes no sistema
<code>buscar_contato_por_nome</code>	Devolve um contato específico, dado o nome como argumento

## Exemplo de queries

### 1. Obter todos os contatos:

```
query {  
  contatos {  
    id  
    nome  
    email  
    telefone  
  }  
}
```

### 2. Buscar contato por nome:

```
query {  
  buscarContatoPorNome(nome: "Ana") {  
    id  
    nome  
    email  
    telefone  
  }  
}
```

## Integração com cliente web

Foi criado um cliente web moderno e funcional com HTML, CSS e JavaScript, que permite ao utilizador introduzir o nome de um contato e efetuar uma pesquisa diretamente contra o endpoint GraphQL.

A interface gráfica realiza uma chamada **POST** para o endpoint `http://localhost:5000/graphql`, enviando a query em formato JSON. O resultado é apresentado na interface de forma elegante.

## Testes realizados

O serviço foi testado de duas formas:

- Através do **GraphiQL** (interface gráfica integrada no próprio servidor GraphQL);
- Via **cliente web**, utilizando JavaScript `fetch()` para envio das queries.

## Considerações

O GraphQL destacou-se por oferecer:

- Consultas precisas e direcionadas
- Flexibilidade no lado do cliente
- Redução no volume de dados transferidos
- Integração simplificada com interfaces modernas



Este serviço é ideal para cenários com requisitos variáveis de consulta e para aplicações onde a eficiência e personalização das respostas são prioritárias.

### 3.4 Serviço gRPC

O serviço **gRPC** foi implementado em Python com suporte a **streaming de dados**, conforme exigido pelo enunciado. Esta tecnologia é altamente eficiente, utilizando o protocolo HTTP/2 e a serialização **Protocol Buffers**, o que garante comunicações rápidas e com baixa latência.

#### Arquitetura do serviço

O serviço foi dividido em duas partes principais:

- **Servidor gRPC:** expõe o método `StreamContatos` com streaming de dados;
- **Cliente gRPC:** desenvolvido com **Tkinter**, fornece uma interface gráfica que consome os dados em tempo real.

#### Método implementado

Método gRPC	Tipo	Descrição
<code>StreamContatos</code>	Server-side streaming	Envia todos os contatos em fluxo contínuo para o cliente

O servidor lê o conteúdo do ficheiro `contatos.json` e envia os registos um a um através do stream, simulando inclusive um pequeno atraso entre envios (`sleep`), de modo a evidenciar o comportamento assíncrono e contínuo do streaming.

#### Cliente gRPC com Tkinter

Foi desenvolvido um **cliente gráfico com Tkinter** que permite ao utilizador:

- Iniciar a comunicação com o servidor gRPC;
- Ver os contatos serem carregados em tempo real numa tabela;
- Usar de uma interface visual simples, moderna e intuitiva.

A interface apresenta os contatos à medida que são recebidos do servidor via streaming, utilizando o componente `Treewiew`.

#### Considerações

O gRPC demonstrou-se uma solução ideal para cenários onde a eficiência de comunicação e o envio contínuo de dados são críticos. Apesar de não ser compatível com clientes web diretamente, a integração com aplicações desktop revelou-se extremamente eficiente e simples de manter.

## 4. Exportação e Importação de Dados

A funcionalidade de **exportação e importação de dados** foi integrada em dois serviços: **SOAP** e **REST**, sendo que o **SOAP** apresenta suporte tanto para JSON como para XML com codificação em **Base64**.

### 4.1 Formatos Suportados

- **JSON**: formato principal de armazenamento dos dados dos contactos.
- **XML**: utilizado para exportação e importação adicional via serviço SOAP.

Todos os dados são guardados no ficheiro `contactos.json`, localizado na pasta `data`.

---

### 4.2 Exportação de Dados

#### SOAP

O serviço SOAP oferece duas rotas que permitem ao utilizador **descarregar os dados diretamente pelo cliente web**:

Método	Caminho	Formato	Descrição
<code>exportar_json</code>	<code>/soap/exportar/json</code>	JSON	Descarrega todos os contactos em JSON
<code>exportar_xml</code>	<code>/soap/exportar/xml</code>	XML	Converte JSON para XML e exporta

Ambos os ficheiros são gravados temporariamente no servidor e enviados como download através do cliente web (Node.js/Express), sendo apagados automaticamente após o envio.

---

### 4.3 Importação de Dados

#### SOAP com Base64


O serviço SOAP também permite a **importação de ficheiros JSON e XML**, que devem ser enviados codificados em **Base64**. Este processo foi escolhido por garantir compatibilidade com o protocolo SOAP e evitar problemas com envio binário.

Método	Tipo de ficheiro	Descrição
<code>importar_json_base64</code>	JSON	Recebe um ficheiro JSON codificado e importa
<code>importar_xml_base64</code>	XML	Converte XML recebido em JSON e importa

A importação é realizada no cliente web com JavaScript (browser), utilizando `FileReader` e `btoa()` para codificar o conteúdo.

## 4.4 Interface Gráfica de Importação/Exportação

No cliente SOAP (`soap.html`), foram integrados:

- **Botões de exportação** com ícones  ;
- **Seletor de ficheiros personalizado** para importar JSON/XML com visual moderno;
- **Mensagens de sucesso ou erro** após a operação.

Este processo permite ao utilizador realizar todas as operações de exportação e importação de forma simples, segura e sem sair da interface web.

## 5. Conclusão

Este projeto teve como objetivo desenvolver uma **Agenda de Contatos distribuída** e acessível por diferentes tecnologias de comunicação entre cliente e servidor. Ao longo do desenvolvimento, foram integradas as seguintes abordagens:

- **REST**, pela sua simplicidade e adoção massiva na indústria;
- **SOAP**, garantindo formalidade e validação de mensagens estruturadas;
- **GraphQL**, oferecendo flexibilidade e precisão na consulta dos dados;
- **gRPC**, permitindo comunicações eficientes e escaláveis, incluindo **streaming de dados**.

Foi também implementada uma camada de **exportação e importação** de dados, com suporte para os formatos **JSON e XML**, utilizando tanto chamadas diretas como codificação em Base64 (para SOAP), assegurando compatibilidade com múltiplos contextos de utilização.

Além dos serviços, foi desenvolvido um **cliente web completo com Node.js e Express**, com páginas HTML personalizadas para REST, SOAP e GraphQL, bem como um **cliente desktop em Tkinter** para consumo do serviço gRPC com streaming.

Durante o projeto, foi promovida uma **estrutura clara e modular**, mantendo a separação entre cliente, servidor e documentação. O código foi devidamente comentado

e testado, e foram utilizados princípios de boas práticas em termos de organização, reutilização e clareza.

Este sistema demonstra a capacidade de integrar diferentes tecnologias de serviços web num único projeto funcional, abrangente e com um forte foco na **interoperabilidade, acessibilidade e experiência do utilizador**. A entrega final inclui toda a documentação, interface gráfica e scripts necessários para execução, estando pronto para ser demonstrado e avaliado.