

Desenvolvimento Web II

Aula 05 - Web Service e API

Prof. Fabricio Bizotto

Instituto Federal Catarinense

fabricio.bizotto@ifc.edu.br

Ciência da Computação

17 de janeiro de 2024

1 Web Service

- Definição
- SOAP
- REST
- Boas Práticas
- Monitoramento e Logs
- Testes
- Ataques

Definição

De acordo com a W3C Working Group 2004 diz que é um sistema de software responsável por proporcionar a **interação entre duas máquinas** através de uma rede.

Características

- **Interoperabilidade** - Comunicação entre diferentes plataformas.
- **Independência de Linguagem** - Permite a comunicação entre diferentes linguagens de programação.
- **Formato de Mensagem** - Utiliza XML ou JSON.
- **Padrões Abertos** - Utiliza padrões abertos como SOAP e REST.

Web Service

SOAP

Simple Object Access Protocol

Definição

- Protocolo de comunicação baseado em **XML**.
- As mensagens SOAP basicamente são **documentos XML** serializados seguindo o padrão W3C enviados em cima de um protocolo de rede como HTTP.
- Utiliza **WSDL**, um documento XML que descreve o serviço, especificando como acessá-lo, quais operações executar, quais parâmetros usar, e qual o formato das mensagens.

Estrutura

- *Envelope* - Define o início e o fim da mensagem. É o elemento raiz.
- *Header* - Define informações adicionais sobre a mensagem. Opcional
- *Body* - Define o conteúdo da mensagem. Obrigatório.
- *Fault* - Define informações sobre erros. Opcional

SOAP - Estrutura

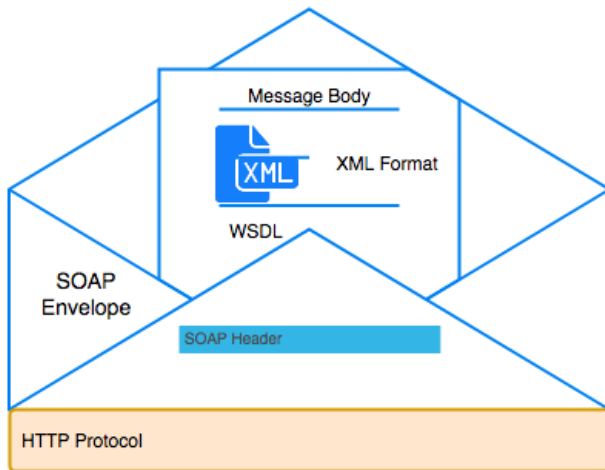


Figura: Estrutura SOAP

Web Service

SOAP - Exemplo

Requisição e Resposta

SOAP - Exemplo

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
2   <soapenv:Header/>
3   <soapenv:Body>
4     <sch:UserDetailsRequest>
5       <sch:name>John</sch:name>
6     </sch:UserDetailsRequest>
7   </soapenv:Body>
8 </soapenv:Envelope>
```

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
2   <soapenv:Header/>
3   <soapenv:Body>
4     <ns2:UserDetailsResponse xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/">
5       <ns2:User>
6         <ns2:name>John</ns2:name>
7         <ns2:age>5</ns2:age>
8         <ns2:address>Greenville</ns2:address>
9       </ns2:User>
10    </ns2:UserDetailsResponse>
11  </soapenv:Body>
12</soapenv:Envelope>
```

Figura: SOAP - Exemplo - Requisição e Resposta

Web Service

SOAP - Exemplo

Olá Mundo em SOAP com Python

SOAP - Servidor

```
exemplos > soap > server.py > HelloWorldService > say_hello
1  from spyne import Application, rpc, ServiceBase, Unicode, Integer
2  from spyne.protocol.soap import Soap11
3  from spyne.server.wsgi import WsgiApplication
4  from wsgiref.simple_server import make_server
5
6  class HelloWorldService(ServiceBase):
7
8      # O decorator @rpc define que o método say_hello é um método remoto
9      @rpc(Unicode, Integer, _returns=Unicode)
10     def say_hello(ctx, name, times):
11         ip_address = ctx.transport.req["REMOTE_ADDR"]
12
13         for i in range(times):
14             print(f"Hello {name} from {ip_address} #{i+1}")
15
16     soap_app = Application([HelloWorldService], 'spyne.examples.hello.soap',
17                             in_protocol=Soap11(validator='lxml'),
18                             out_protocol=Soap11())
19
20     # O objeto WsgiApplication é o que o Spyne usa para gerar o servidor WSGI
21     wsgi_app = WsgiApplication(soap_app)
22
23     if __name__ == '__main__':
24         server = make_server('0.0.0.0', 8000, wsgi_app)
25         server.serve_forever()
```

Figura: SOAP - Servidor

SOAP - Cliente

exemplos > soap >  client_soap.py > ...

```
1  from zeep import Client
2  from zeep.plugins import HistoryPlugin
3  from lxml import etree
4
5  # Criar um cliente Zeep com base no URL do WSDL
6  history = HistoryPlugin()
7  client = Client(f'http://localhost:8000/?wsdl', plugins=[history])
8
9  # Chamar o método do serviço
10 response = client.service.say_hello(name='Professor', times=3)
11
12 # Exibir a resposta
13 for hist in [history.last_sent, history.last_received]:
14     print(etree.tostring(hist["envelope"], encoding="unicode", pretty_print=True))
```

Figura: SOAP - Cliente

SOAP - Chamada e WSDL

```
(.venv) fabricio@DESKTOP-MG3SLC3:~/Projetos/Desenvolvimento-Web-II/exemplos/soap$ python server.py
127.0.0.1 - - [12/Jan/2024 12:39:01] "GET /?wsdl HTTP/1.1" 200 2613
Hello Professor from 127.0.0.1
Hello Professor from 127.0.0.1
Hello Professor from 127.0.0.1
127.0.0.1 - - [12/Jan/2024 12:39:01] "POST / HTTP/1.1" 200 235
[]
(.venv) fabricio@DESKTOP-MG3SLC3:~/Projetos/Desenvolvimento-Web-II/exemplos/soap$ python client.py
None
```

```
<wsdl:definitions
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:plink="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:wsdlsoap11="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdlsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap11enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap11env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soap12env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:soap12enc="http://www.w3.org/2003/05/soap-encoding"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:xop="http://www.w3.org/2004/08/xop/include"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:tns="spyne.examples.hello.soap" targetNamespace="spyne.examples.hello.soap" name="Application">
  <wsdl:types>
    <xs:schema targetNamespace="spyne.examples.hello.soap" elementFormDefault="qualified">
      <xs:complexType name="say_hello">
        <xs:sequence>
          <xs:element name="name" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="times" type="xs:integer" minOccurs="0" nillable="true"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="say_helloResponse">
        <xs:sequence>
          <xs:element name="say_helloResult" type="xs:string" minOccurs="0" nillable="true"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="say_hello" type="tns:say_hello"/>
      <xs:element name="say_helloResponse" type="tns:say_helloResponse"/>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="say_hello">
    <wsdl:part name="say_hello" element="tns:say_hello"/>
  </wsdl:message>
```



localhost:8000/?wsdl

Web Service

SOAP - Exemplo com Chamada Direta

Podemos enviar o arquivo XML diretamente para o servidor

SOAP - Código para Chamada Direta com XML

```
exemplos > soap > client_soap_xml.py > ...  
1  from zeep import Client  
2  from zeep.plugins import HistoryPlugin  
3  from lxml import etree  
4  import http.client  
5  
6  # ler o arquivo xml com a requisição  
7  with open("request.xml", "r") as f:  
8      |    xml_content = f.read()  
9  
10 # Criar um cliente Zeep com base no XML  
11 connection = http.client.HTTPConnection("localhost", 8000)  
12 connection.request("POST", "/", xml_content, headers={"Content-Type": "text/xml"})  
13  
14 # Exibir a resposta  
15 response = connection.getresponse()  
16 print(response.status, response.reason)  
17 print(response.read().decode())  
18  
19 # Fechar a conexão  
20 connection.close()
```

Figura: SOAP - Chamada Direta - Cliente

SOAP - Enviando XML para o Servidor via Postman

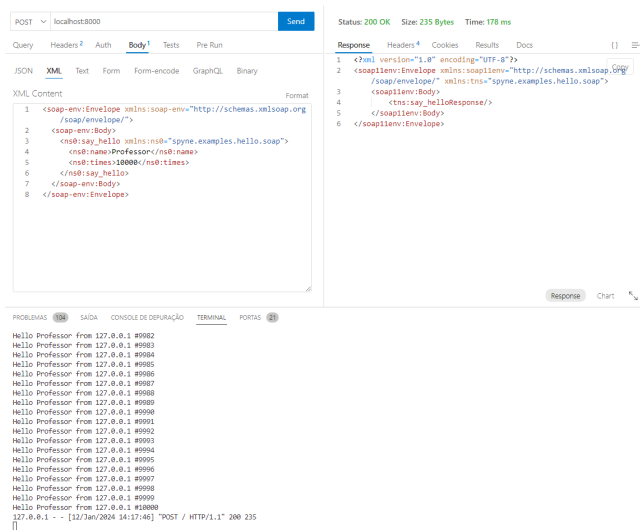


Figura: SOAP - Enviando XML

Web Service

REST

Representational State Transfer

REST

Como surgiu?

A arquitetura de sistema REST foi criada pelo cientista da computação **Roy Fielding em 2000**.

Anteriormente ele já havia trabalhado na criação do **protocolo HTTP e do URI**, um conjunto de elementos que identifica recursos nas aplicações web.

Buscando padronizar e organizar os protocolos de comunicação e desenvolvimento na internet, Fielding se uniu a um time de especialistas para desenvolver, **durante 6 anos**, as características da **REST**, que foi definida em sua tese de PhD.



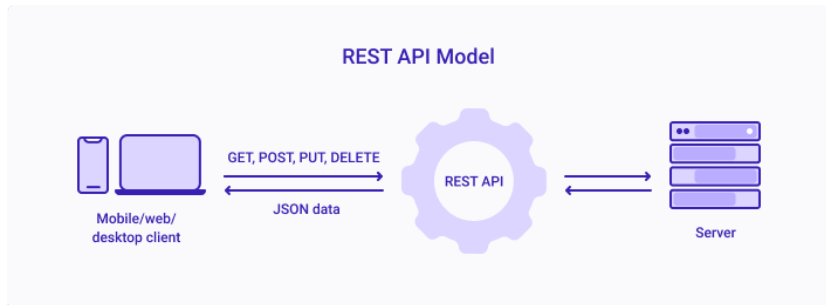


Figura: Estrutura REST

RESTful

Qual a diferença entre REST e RESTful?

REST

É uma **arquitetura** que define um conjunto de princípios para projetar aplicações web. Os critérios que devem ser cumpridos são:

- **Cliente-Servidor** - Separação entre o cliente e o servidor.
- **Stateless** - O servidor não armazena informações sobre o cliente. Cada requisição é independente.
- **Cache** - O servidor deve informar se a resposta pode ser armazenada em cache.
- **Interface Uniforme** - O cliente só precisa saber a URL do recurso e o servidor deve retornar os dados no formato apropriado.
- **Sistema em camadas** - O cliente não precisa saber se está se comunicando diretamente com o servidor ou com um intermediário.

RESTful

É uma API que **implementa os princípios REST**.

REST vs SOAP

SOAP	REST
SOAP é um protocolo	REST é uma arquitetura
Geralmente usa HTTP/HTTPS, mas pode usar outros	Usa apenas HTTP/HTTPS
XML	XML, JSON, HTML, etc
SOAP usa WSDL	Rest usa apenas a URL
Precisa fazer o parse da mensagem	Não precisa fazer o parse da mensagem
É mais pesado	É mais leve
Não usa cache	Pode usar cache
WS-Security ¹	HTTPS

Tabela: SOAP vs REST

¹Conhecendo o WS-Security

Web Service

REST

Boas práticas

1. Documentação Clara

Forneça uma documentação clara e abrangente para a API, descrevendo recursos, endpoints, parâmetros, cabeçalhos e exemplos de solicitações e respostas.

2. JSON

JSON é o formato de dados mais utilizado, embora você possa enviar dados em outros formatos como CSV, XML e HTML. A sintaxe JSON pode tornar os dados fáceis de ler para humanos. É fácil de usar e oferece avaliação e execução de dados rápida e fácil. Além disso, ele contém uma ampla gama de compatibilidade de navegadores suportados.

```
"produto": {  
  "id": 1,  
  "nome": "Produto 1",  
  "descricao": "Descricao do produto 1",  
  "preco": 100.00,  
  "categorias": [  
    {  
      "id": 1,  
      "nome": "Categoria 1"  
    }  
  ]  
}
```

3. Versionamento da API

Inclua versões na sua API para garantir a compatibilidade com versões anteriores e permitir evolução controlada. Pode ser feito por meio de versões na URI ou por meio de cabeçalhos.

Exemplo

- **URI** - `/api/v1/produtos` ou `/api/v2/produtos`
- **Cabeçalho** - `Accept: application/vnd.company.app-v1+json`

4. Nomes de Recursos Descritivos

- Use substantivos para nomear recursos.
- Use o plural para nomear coleções.
- Use o singular para nomear itens individuais.

Certo

- `/api/v1/produtos`
- `/api/v1/produtos/1`
- `/api/v1/produtos/1/categorias`

Errado

- `/api/v1/criarProduto`
- `/api/v1/obterProduto/1`
- `/api/v1/prodCat/1`

5. Verbos HTTP

Use métodos HTTP para operações CRUD. Por exemplo:
GET, POST, PUT e DELETE .

Exemplo

GET	/api/v1/produtos
GET	/api/v1/produtos/1
POST	/api/v1/produtos
PUT	/api/v1/produtos/1
DELETE	/api/v1/produtos/1
PATCH	/api/v1/produtos/1 <i>(atualiza apenas alguns campos)</i>

6. Códigos de Status HTTP

- **1xx** - Informação
- **2xx** - Sucesso
- **3xx** - Redirecionamento
- **4xx** - Erro do cliente
- **5xx** - Erro do servidor

Exemplo

- | | |
|------------------------------------|---|
| ■ 200 - OK | ■ 404 - Não encontrado |
| ■ 201 - Criado | ■ 500 - Erro interno do servidor |
| ■ 400 - Requisição inválida | ■ 501 - Não implementado |
| ■ 401 - Não autorizado | ■ 503 - Serviço indisponível |

7. Paginação

Para coleções muito grandes, use paginação para limitar o número de itens retornados.

Exemplo

- `/api/v1/produtos?page=1&limit=10`
- `/api/v1/produtos?page=2&limit=10`

8. Filtros

Para coleções muito grandes, use filtros para limitar os itens retornados.

Exemplo

GET `/api/v1/produtos?type=eletronicos`

GET `/api/v1/produtos?price_min=100&price_max=200`

GET `/api/v1/produtos?search=smartphone`

9. Ordenação

Para coleções muito grandes, use ordenação para limitar os itens retornados.

Exemplo

GET `/api/v1/produtos?sort=nome`

GET `/api/v1/produtos?sort=nome&asc=false`

GET `/api/v1/produtos?sort=preco,vendas&ordem=desc,desc`

10. HATEOAS

Hypermedia As The Engine Of Application State

Se possível, adote o HATEOAS para permitir que os clientes naveguem pela API dinamicamente usando links nos recursos.

Exemplo

GET `/api/v1/produtos?page=1&limit=10`

GET `/api/v1/produtos?page=2&limit=10`

GET `/api/v1/produtos?page=3&limit=100`

11. Segurança

- Utilize sempre HTTPS para garantir a criptografia dos dados durante a transmissão. Isso protege contra ataques de interceptação (man-in-the-middle) e assegura a confidencialidade das informações.

11. Segurança (cont.)

Use autenticação para proteger a API. Exemplos: **Basic Auth**¹, bearer token e OAuth.

Basic Auth

GET /api/resource HTTP/1.1

Host: example.com

Authorization: Basic base64(username:password)

¹Apesar de ser fácil de implementar, as credenciais são enviadas sem criptografia, o que torna esse método vulnerável a ataques de interceptação.

11. Segurança (cont.)

Use autenticação para proteger a API. Exemplos: **Basic Auth**, **bearer token²** e **OAuth**.

Bearer Token

GET /api/resource HTTP/1.1

Host: example.com

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

²Os tokens devem ser mantidos em segredo e geralmente têm um tempo de expiração. Este método é amplamente utilizado em autenticação de API REST.

11. Segurança (cont.)

Use autenticação para proteger a API. Exemplos: **Basic Auth**, **bearer token** e **OAuth³**.

OAuth - Fluxo de Autorização

Google, Facebook, Twitter, GitHub, etc.

- **Passo 1** - O cliente solicita autorização do usuário.
- **Passo 2** - O usuário autoriza o cliente.
- **Passo 3** - O cliente recebe um código de autorização.
- **Passo 4** - O cliente troca o código de autorização por um token de acesso.
- **Passo 5** - O cliente usa o token de acesso para acessar o recurso protegido.

³OAuth é um protocolo de autorização usado para permitir que aplicativos acessem recursos em nome do usuário. Ele fornece tokens de acesso que podem ser usados para autenticar solicitações.

12. CORS (Cross Origin Resource Sharing)

Permite que os clientes acessem a API de um domínio diferente .

Cabeçalho

Access-Control-Allow-Origin

http://localhost:3000, *, ...

Access-Control-Allow-Methods

Métodos HTTP permitidos (GET, POST, PUT, DELETE, ...)

Access-Control-Allow-Headers

Indica quais cabeçalhos podem ser expostos como parte da resposta (Content-Type, Authorization, ...)

Access-Control-Allow-Credentials

Indica se o navegador deve incluir credenciais (como cookies ou cabeçalhos de autenticação) na solicitação.

REST

Boas Práticas - CORS - Exemplo Prático

```
exemplos > cors > app.py > ...
1 # --- Servidor rodando na porta 5000
2 from flask import Flask, jsonify
3 from flask_cors import CORS
4
5 app = Flask(__name__)
6 # --- Cliente rodando em http://localhost:5500 terá acesso a API
7 CORS(app, origins=['http://localhost:5500'])
8
9 @app.route('/api/data', methods=['GET'])
10 def get_data():
11     data = {'message': 'Dados da API acessados com sucesso!'}
12     return jsonify(data)
13
14 if __name__ == '__main__':
15     app.run(debug=True)
```

servidor

```
8 <body>
9 <h1>CORS Test</h1>
10 <button onclick="getData()">Obter Dados da API</button>
11 <p id="result"></p>
12
13 <script>
14     function getData() {
15         fetch('http://localhost:5000/api/data')
16             .then(response => response.json())
17             .then(data => {
18                 document.getElementById('result').innerText =
19                     data.message;
20             })
21             .catch(error => {
22                 document.getElementById('result').innerText =
23                     'Erro ao obter dados da API'
24             });
25     }
26 </script>
27 </body>
```

cliente
(servidor web)

Figura: CORS - Exemplo - Servidor e Cliente

Figura: CORS - Exemplo - Simulação

13. Monitoramento e Logs

- Monitore a API para garantir que ela esteja sempre disponível.
- Registre todas as solicitações e respostas para fins de auditoria e depuração.

Exemplo

```
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

Middleware para monitoramento do tempo de resposta

```
@app.after_request
```

```
def after_request(response):
```

```
    duration = time.time() - request.start_time
```

```
    logger.info(f"{request.method} {request.path} - Tempo de Resposta: {duration} segundos")
```

```
    return response
```

14. Testes Automatizados

Crie testes automatizados para garantir a estabilidade da API e detectar rapidamente problemas de integração ou regressão.

Exemplo


```
# Integration Test
```

```
def test_get_all_products(self):  
    response = self.client.get("http://localhost:5000/api/v1/p  
    self.assertEqual(response.status_code, 200)
```


15. Proteção contra ataques

SQL Injection: Use **prepared statements** ou **ORM**.

Exemplo - SQL Injection

exemplos > seg >  sql.py > ...

```
1 @app.route('/login', methods=['POST'])
2 def login():
3     query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'" # Errado
4     query = f"SELECT * FROM users WHERE username = ? AND password = ?" # Correto
5     result = cursor.execute(query, (username, password)).fetchone()
6
7 # Simulation of SQL Injection
8 # SELECT * FROM users WHERE username = ' OR 'a'='a';-- AND password = '';
```

15. Proteja contra ataques (cont.)

Cross-Site Scripting (XSS): Use `escape` ou `sanitize` para evitar que os usuários insiram código HTML ou JavaScript nos dados.

Exemplo - XSS

```
@app.route("/")
def index():
    return """
    <form action="/search">
        <input name="query" type="text" />
        <input type="submit" value="Buscar" />
    </form>
    """

# <script>alert("XSS");</script>
# <style>body { background: red; }</style>
@app.route("/search")
def search():
    nome = request.args.get("query", "")
    nome_escapedo = escape(nome) # usar escape para evitar XSS
    return f"<h3>Buscando por: {nome_escapedo}</h3>"
```

15. Proteja contra ataques (cont.)

Cross-Site Request Forgery (CSRF): Use **tokens** para evitar que os usuários sejam enganados para executar ações indesejadas em nome deles. O token CSRF é um valor aleatório que é gerado pelo servidor web e enviado ao cliente. O cliente deve enviar o token CSRF de volta ao servidor web ao enviar um formulário.

Exemplo - CSRF

```
<form action="/login" method="post">
  <input type="text" name="email" value="admin@example.com">
  <input type="password" name="senha" value="admin">
  <input type="hidden" name="csrf_token" value="[CSRF_TOKEN]">
  <input type="submit" value="Enviar">
</form>
```