Tópicos sobre Web Services

Introdução

- Web services
  - Solução arquitetural para o desenvolvimento de Aplicações distribuídas do tipo cliente-servidor
  - Recurso ao protocolo HTTP (*HyperText Transfer Protocol*)
  - O protocolo HTTP é um mecanismo normalizado e consolidado de interação entre componentes de software heterogéneos através da Internet (pilha protocolar TCP/IP)
  - Uma sessão HTTP é uma sequência de transações do tipo pedido-resposta baseadas em mensagens de texto (ascii)
  - Aplicações do tipo servidor: fornecem serviços
  - Aplicações do tipo cliente: consomem serviços

Introdução

- Exemplo de uma transacção entre um cliente e um servidor HTTP

PEDIDO

```
GET /index.html HTTP/1.0
```

RESPOSTA

```
HTTP/1.0 200 OK
Last-Modified: Monday, 27-Dec-99 22:14  GMT
Content-Type: text/html
Content-Length: 1639
```

Cabeçalho

```
<HTML>
   <HEAD>
      <TITLE> Our homepage </TITLE>
   </HEAD>
   <BODY>
   ..... // BODY GOES HERE
   </BODY>
</HTML>
```

Corpo (opcional)

Introdução

- URI (*Uniform Resource Identifier*)
  - Permite identificar vários tipos de recursos
  - O alvo de um pedido HTTP é um dos possíveis tipos de recurso identificáveis
  - Os mensagens de pedido HTTP incluem uma URI
- Um *web service* pode ser considerado como sendo um componente de *software* localizado num sistema distribuído e identificável através de uma URI

Introdução

- Soluções baseadas em *web services* tiram partido de uma infraestrutura distribuída abrangente e consolidada, que está na base da *World Wide Web*, para suportar a interação de componentes de *software* em ambientes distribuídos e heterogéneos
- Várias abordagens
  - SOAP (*Simple Object Access Protocol*)
  - REST (*Representational State Transfer*)

Web Services SOAP

- SOAP
  - Protocolo de comunicação entre clientes e serviços
  - Formato das mensagens baseadas na linguagem XML (*Extensible Markup Language*)
  - Recorre aos serviços de transporte do protocolo HTTP
- As operações suportadas por um *web service* SOAP são descritas através de WSDL (*Web Services Description Language*)
- Um documento WSDL é escrito em XML e permite que as aplicações cliente identifiquem os serviços oferecidos e saibam de que forma os devem aceder (interfaces oferecidas)

Web Services SOAP

Estrutura de um documento WSDL

```
<definitions>

<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

https://www.w3schools.com/xml/xml_wsdl.asp

Web Services SOAP

Exemplo: definição de uma operação

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

https://www.w3schools.com/xml/xml_wsdl.asp

Web Services SOAP

Definição de um web service SOAP

```
<!- ... Mensagens e portType:
    ver acetato anterior... -->
```

O corpo/conteúdo da mensagem SOAP é um documento XML:
- "Qualquer" (style="**document**")
- Que representa especificamente a chamada a um método (style="**rpc**")

```
<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

https://www.w3schools.com/xml/xml_wsdl.asp

"**literal**": <x>5.1</x>
"**encoded**": <x xsi:type="xsd:float">5.1</x>

Web Services SOAP

Estrutura de uma mensagem SOAP

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
</soap:Body>

</soap:Envelope>
```

https://www.w3schools.com/xml/xml_wsdl.asp

Web Services SOAP

Exemplo: pedido SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

https://www.w3schools.com/xml/xml_soap.asp

Web Services SOAP

Exemplo: resposta SOAP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

https://www.w3schools.com/xml/xml_soap.asp

Web Services REST

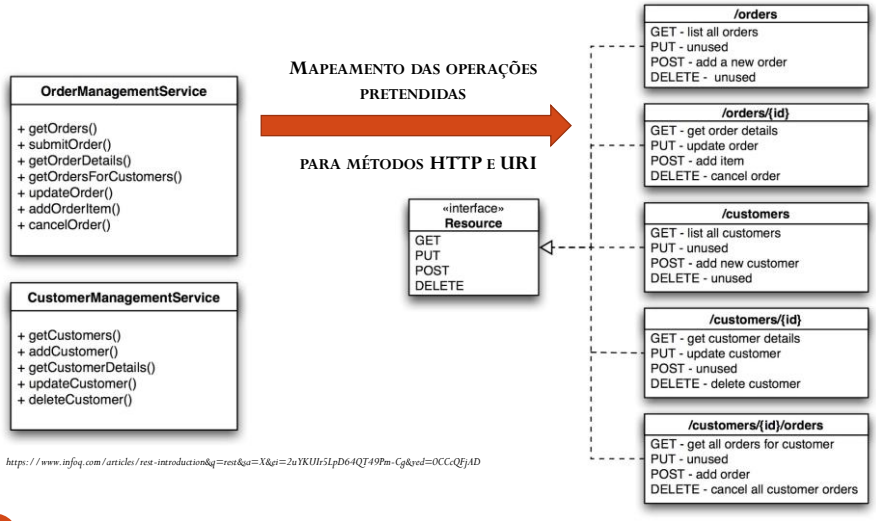
- Estilo arquitetural para sistemas distribuídos heterogéneos
- Define um conjunto de princípios que estipulam de que forma normas próprias à web (e.g., HTTP e URI) devem/deveriam ser usadas
- Um recurso (i.e., qualquer informação nomeável) deve possuir um identificador único e global: URI
  - Um recurso pode ser do tipo *singleton* ou coleção

```
http://sotintas.com/clientes
http://sotintas.com/clientes/56
http://sotintas.com/clientes/56/encomendas
http://sotintas.com/clientes/56/encomendas/2018
http://sotintas.com/clientes/56/encomendas/2018?valor-minimo=1000
```


Web Services REST

- Deve ser usado um conjunto simples e bem definido de métodos para agir sobre os recursos → interface uniforme
  - Uma mensagem de pedido deve incluir, além da identificação do recurso alvo, o método que corresponde à operação pretendida
  - Recorrendo ao HTTP, os verbos (i.e., métodos) habitualmente usados nas interfaces REST são GET, POST, PUT e DELETE
  - A semântica dos verbos HTTP deve ser respeitada
  - Qualquer aplicação que "entenda" HTTP pode aceder ao serviço
  - Com exceção do verbo POST, que é usado para enviar dados, todos os outros são *idempotent* (repetições de um determinado pedido produzem sempre o mesmo resultado)

Web Services REST



https://www.infoq.com/articles/rest-introduction&q=rest&a=X&ei=2uYKUIr5LpD64QT49Pm-Cg&ved=0CCcQfjAD

Web Services REST

```
POST /customers HTTP/1.1
HOST: localhost
User-Agent: curl/7.55.1
Accept: */*
Content-Type: application/json
Content-Length: 20

Marguerite Yourcenar
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Sun, 09 Dec 2018 23:49:05 GMT

true
```

Os parâmetros e resultado, quando existem, são devolvidos nos corpos das mensagens

```
GET /customers/1 HTTP/1.1
HOST: localhost
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Sun, 09 Dec 2018 23:50:02 GMT

{"id":1,"name":"Marguerite Yourcenar"}
```

Web Services REST

Para efeitos de filtragem, podem ser usadas *queries* nas URI

```
GET /customers?home-country=france
HTTP/1.1
HOST: localhost
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Sun, 09 Dec 2018 23:50:02 GMT

[{"id":1,"name":"Marguerite Yourcenar"}]
```


Web Services REST

- Usando, como exemplo, a *framework* Spring / Spring Boot para desenvolver o serviço…

```
/* ... */

@RestController
public class CustomersController {

    /* ... */

    @GetMapping("/customers/{id}")
    public Customer getCustomers(@PathVariable("id") int id)
    {
       Customer result;
       /* ... */
       return result;
    }
```

Web Services REST

```
    @GetMapping("/customers")
    public List<Customer> getCustomers(@RequestParam(value="home-
              country", required=false) String homeCountry)
    {
       ArrayList<Customer> result = new ArrayList<>();
       /* ... */
       return result;
    }

    @PostMapping("/customers")
     public boolean addNewCustomer(@RequestBody String payload)
     {
        boolean result;
        /* ... */
        return result;
     }

}
```


Web Services REST

- … e um cliente para o serviço…

```
import org.springframework.web.client.RestTemplate;

/* ... */

       RestTemplate restTemplate = new RestTemplate();

       boolean result = restTemplate.postForObject(
              "http://localhost:8080/customers/",
              "Marguerite Yourcenar", boolean.class);
       /* ... */
       String r1 = restTemplate.getForObject(
          "http://localhost:8080/customers/1", String.class));
       Customer r2 = restTemplate.getForObject(
          "http://localhost:8080/customers/1", Customer.class));
       List<Customer> r3 = restTemplate.getForObject(
          "http://localhost:8080/customers", List.class));

/* ... */
```

Web Services REST

- A representação de um recurso (dados, *metadados* e hiperligações que representam o seu estado num determinado instante) pode seguir múltiplos formatos
  - Os formatos aceites são indicados em pedidos HTTP
  - Um servidor pode suportar vários formatos para os resultados devolvidos (HTML, XML, texto, PDF, JSON, etc.)

```
GET /api/3 HTTP/1.1
Host: gturnquist-quoters.cfapps.io
User-Agent: curl/7.55.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Sun, 02 Dec 2018 23:40:03 GMT
Content-Length: 177

{"type":"success","value":{"id":3,"quote":"Spr
ing has come quite a ways in … using it."}}
```


Web Services REST

- Abordagem cliente-servidor
  - Aplicações cliente e servidor sem interdependências
  - Os clientes apenas precisam de conhecer as URI dos recursos
- A comunicação deve processar-se de um modo *stateless*
  - Qualquer pedido enviado a um *web service* REST deve incluir toda a informação necessária à sua execução
  - Os servidores não guardam qualquer estado de comunicação com os requentes dos serviços que oferece
  - Qualquer informação de contexto necessária deve ser mantida do lado dos clientes
- Suporte de arquiteturas *multi-tier* (com múltiplas camadas)

Web Services REST

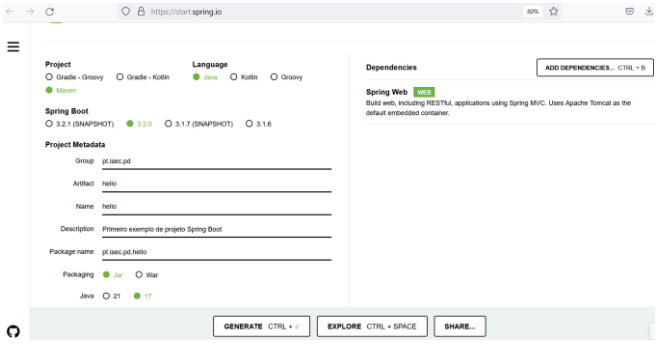
- HATEAS (*Hypermedia As The Engine of Application State*)
  - A resposta a um pedido (representação do recurso solicitado) pode incluir ligações hipermédia para outros recursos disponibilizados pela mesmo servidor ou por outro servidor qualquer
  - Fornecer ligações aos clientes de um *web service* permite tornar as aplicações dinâmicas

```
{
    "name": "Alice",
    "links": [ {
        "rel": "self",
        "href": "http://localhost:8080/customer/1"
    } ]
}
```

https://spring.io/understanding/HATEOAS

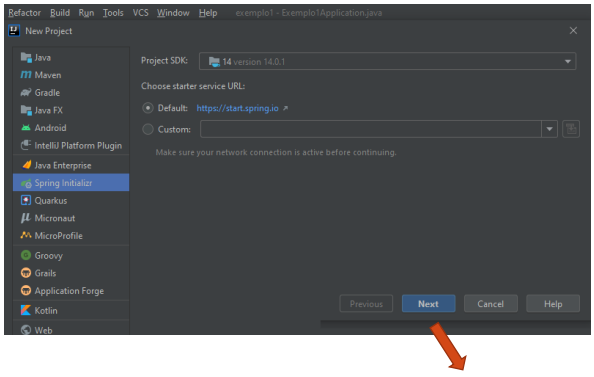
Projetos Spring Boot

- Cria um projeto a partir do *site* Spring Initializer
  - https://start.spring.io/
  - Inportar o projecto no IDE a partir do zip gerado/obtido

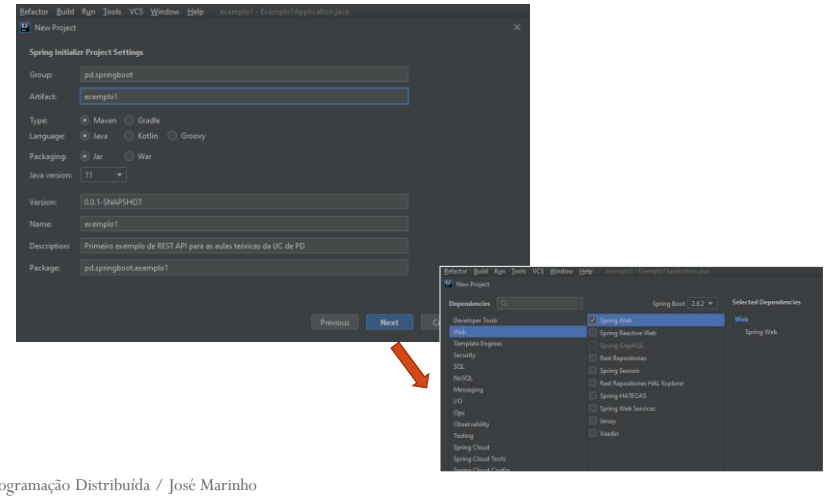


Projetos Spring Boot

- No IDE IntelliJ, pode ser criado um projeto do tipo "Spring Initialzr" diretamente

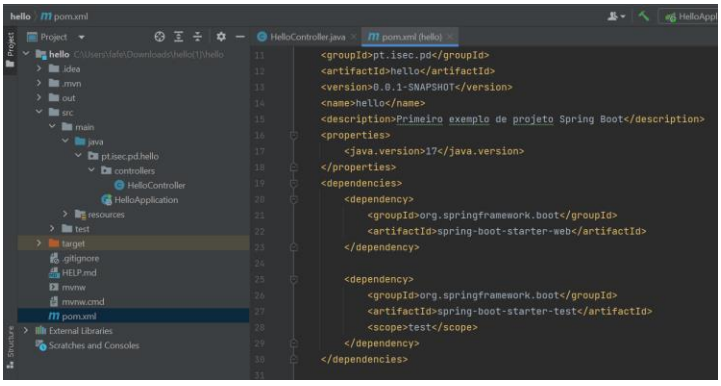


Projetos Spring Boot



Projetos Spring Boot

- Ficheiro "pom.xml" (*Project Object Model*) usado pelo Maven para construir um projecto



Projetos Spring Boot

- Definição dos controladores (mapeamento entre pedidos e métodos)