

A stack of several old, yellowed books tied with blue leather straps, visible on the left side of the slide.

REST

*Integre sistemas de forma
simples e inteligente*



Congresso Científico da Região
Centro-Ocidental do Paraná

18 a 22 de maio de 2015

Douglas Nassif Roma Junior

douglas.nassif@grupointegrado.br

Conteúdo

- Um fato verídico
- REST
 - O que é?
 - Quando surgiu?
 - Quando usar?
- HTML versus REST
- Webservices
- Na prática
 - XML
 - Servlet
 - JSR 311 (JAX-RS)



REST

Integrando aplicações e disponibilizando serviços sem complicar a vida de ninguém.

Um fato verídico...

- Era uma vez um desenvolvedor...



Um fato verídico...

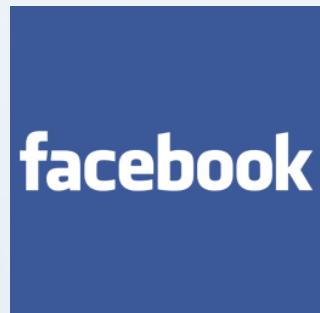
- Que adorava desenvolver...



Programming Language

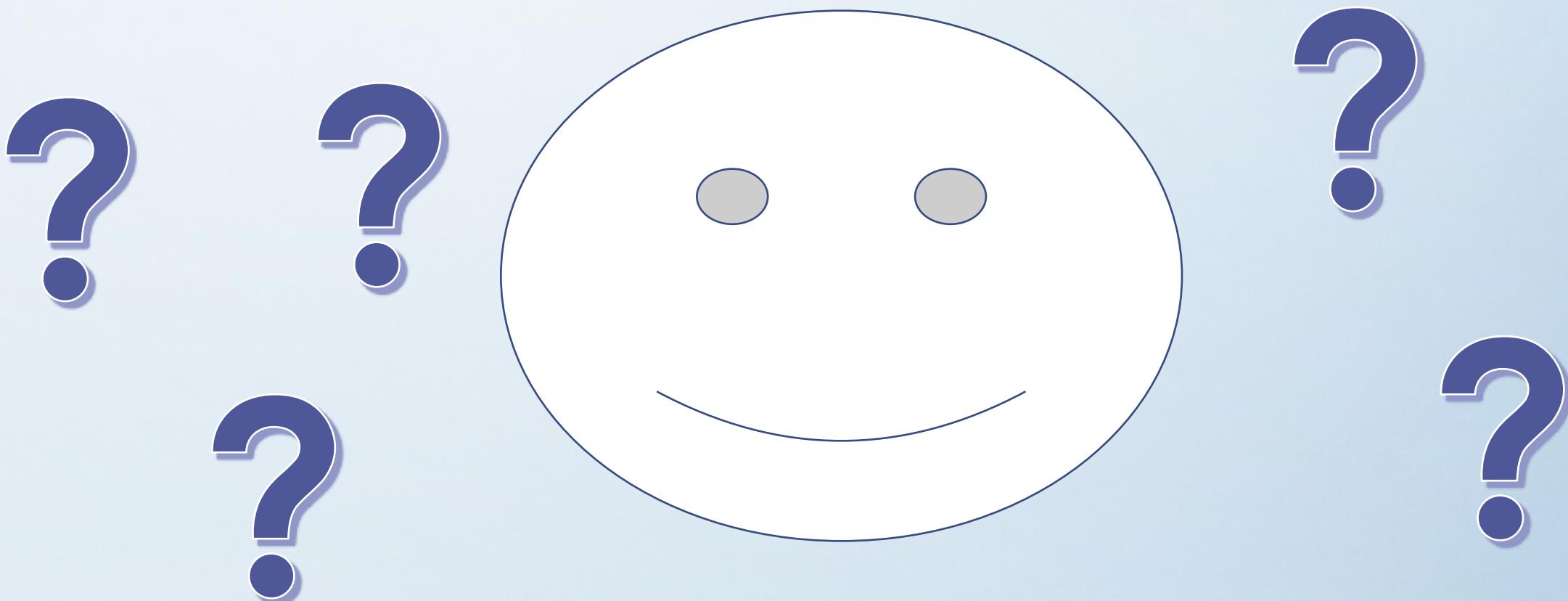
Um fato verídico...

- Além de adorar desenvolver, ele também adorava Web...



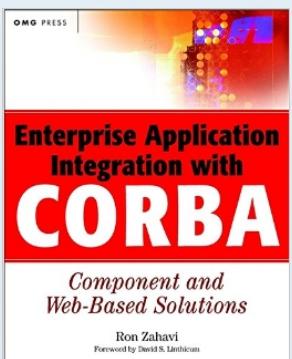
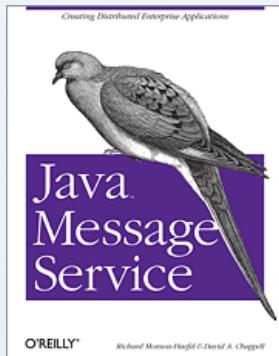
Um fato verídico...

- E como era descolado e desenvolvida em muitas plataformas, ele precisava integrá-las de alguma maneira...



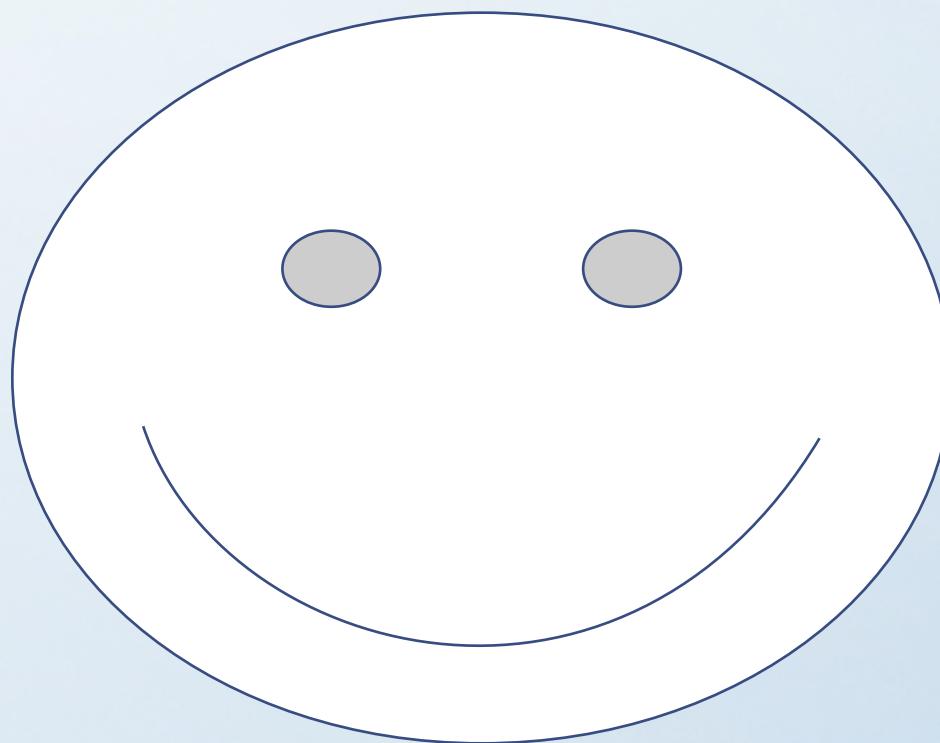
Um fato verídico...

- Então ele descobriu os padrões de integração de cada tecnologia...



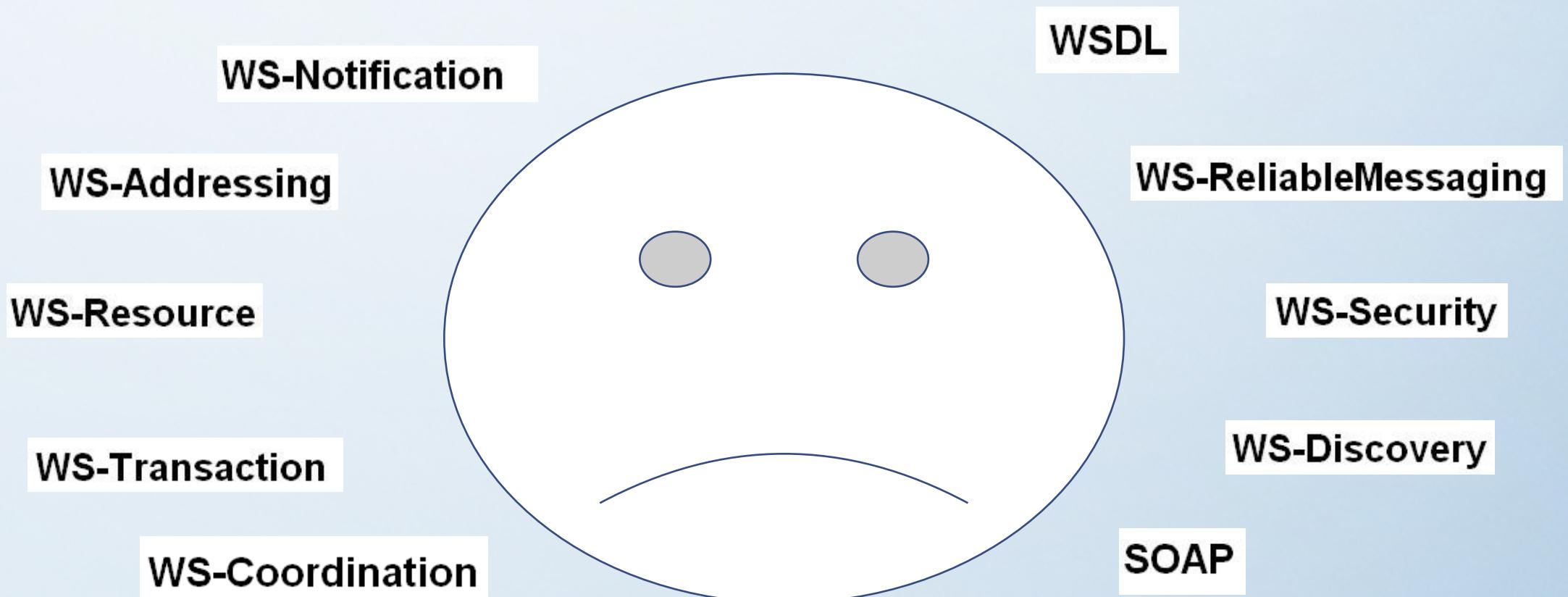
Um fato verídico...

- E também descobriu os WebServices...



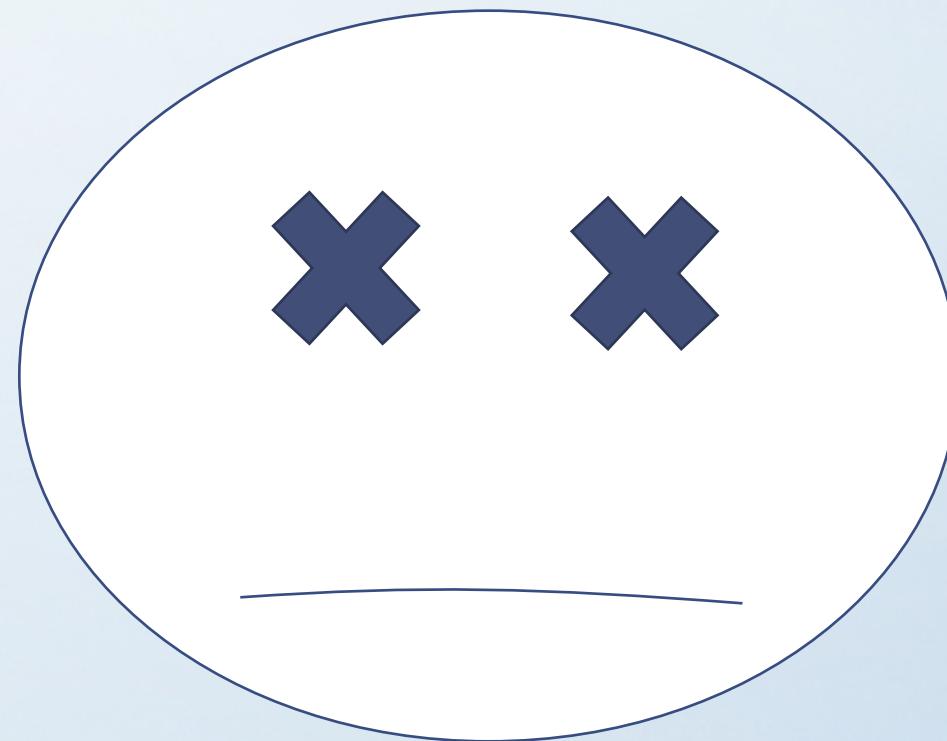
Um fato verídico...

- E todos os seus padrões

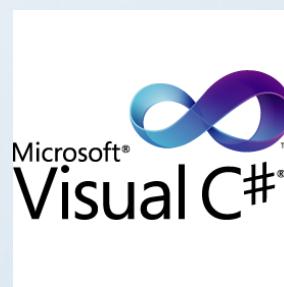
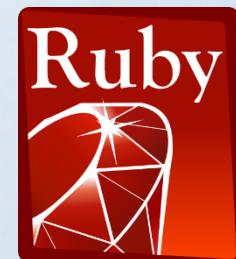


Um fato verídico...

- Aí ele morreu. Fim.



Plataformas distintas



Programming Language

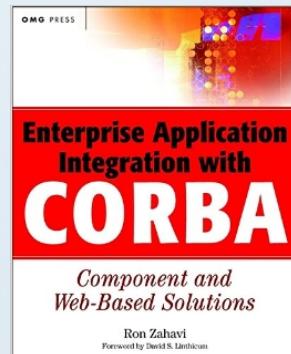
Sistemas terceiros



Padrões para integração



TXT



ORACLE
SOA Suite



WebServices

WS-Addressing

WSDL

WS-Notification

WS-Security

WS-Transaction

WS-Resource

WS-Discovery

WS-ReliableMessaging

WS-Coordination

SOAP

Um fato verídico...

- Mentira, ele não morreu!



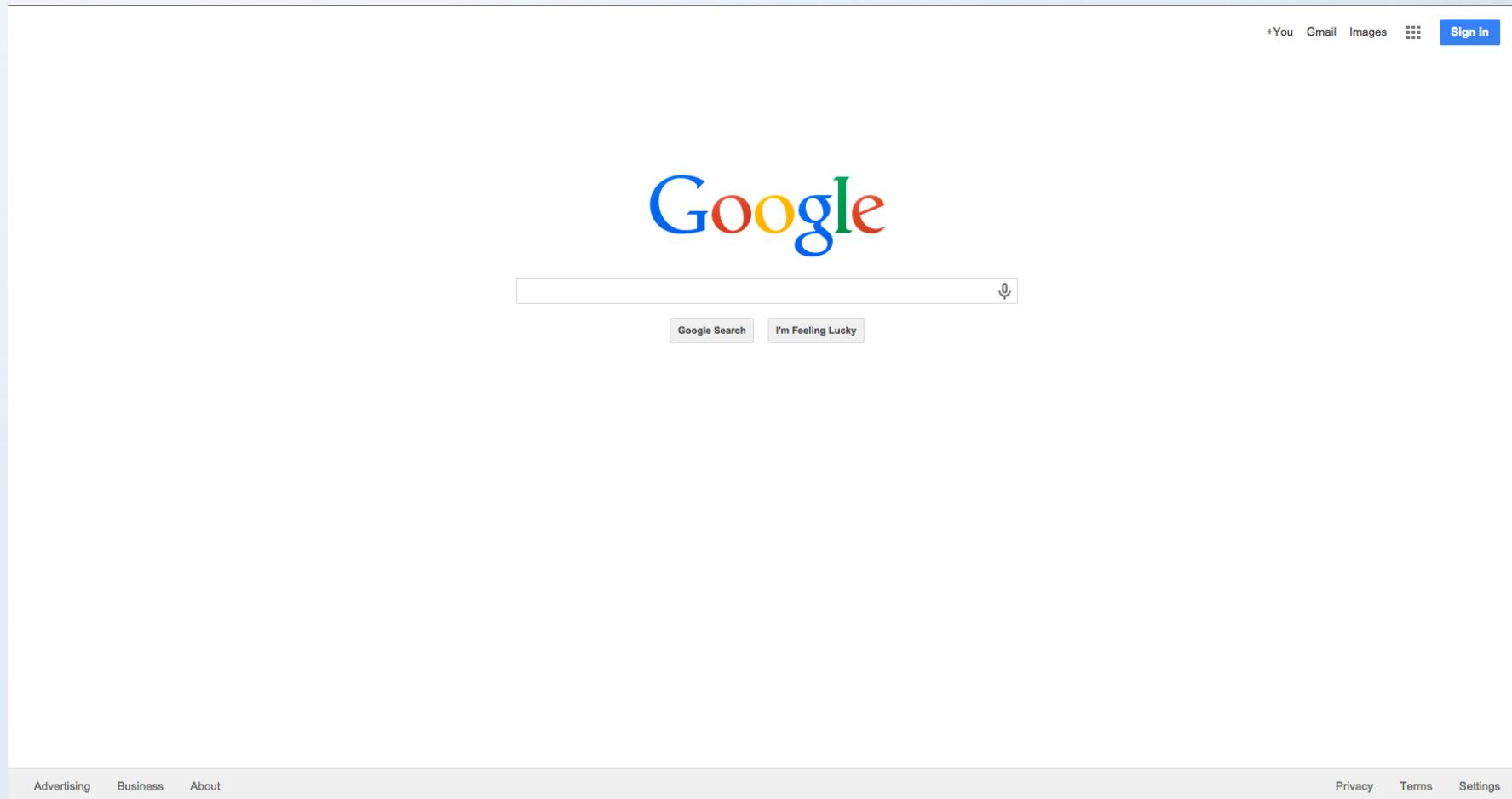
Um fato verídico...

- Mas perder um pouco da sanidade ao tentar entender todos aqueles padrões e aquela complexidade...



Por que?

- Porque o protocolo **HTTP** e o **Google** fazem sucesso?



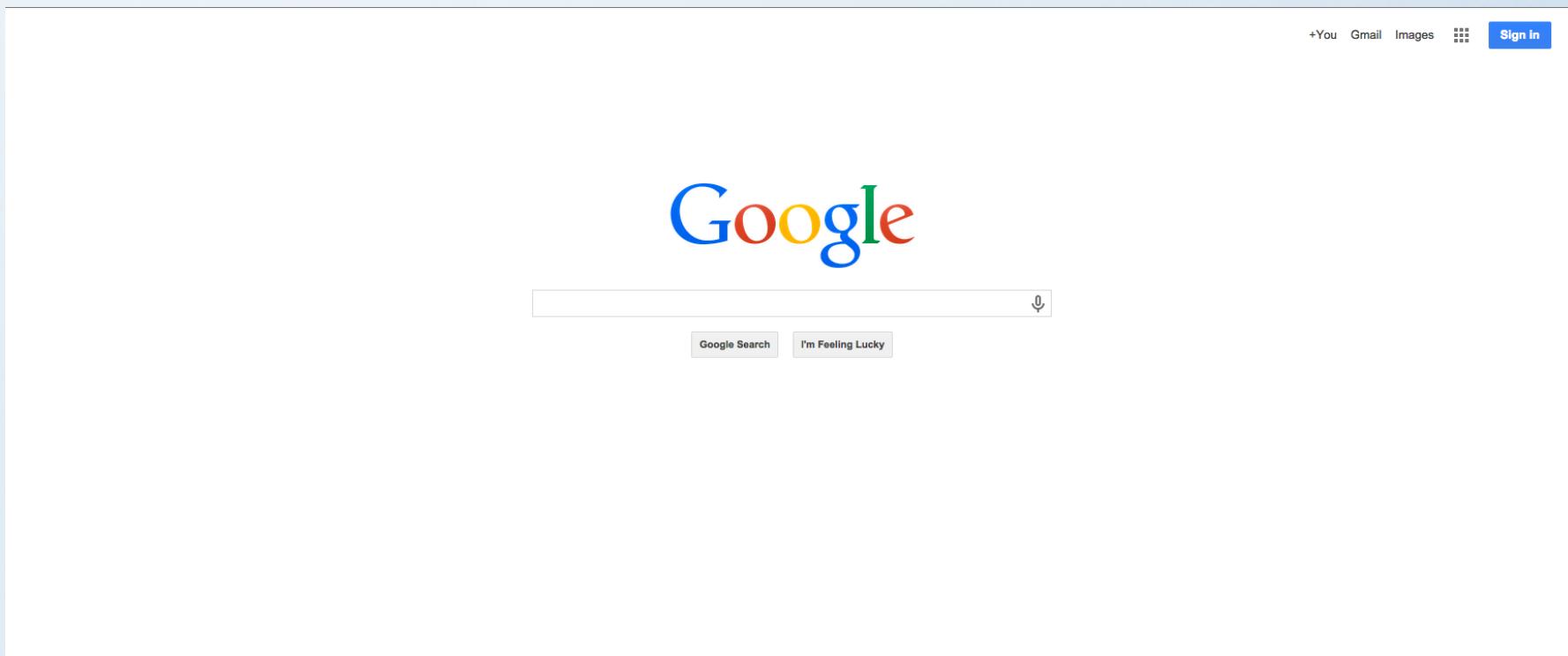
Simplicidade!!!

- Imagina se fosse assim...



Por que?

- HTTP – Transferência de Hiper Texto
- Google – Um campo de texto e dois botões



REST

REST

REpresentational State Transfer

(Transferência de Estado Representacional)

REST

- Surgiu da tese de doutorado de **Roy T. Fielding**
- Um dos principais autores da especificação do protocolo **HTTP**
- Co-fundador do Apache HTTP Server
- Diretor da Apache Software Foundation

REST

- Não é uma biblioteca, um framework e nem uma espécie de software.
- Mas sim, um **“estilo arquitetural”!**

Quando usar?

- Comunicação **stateless**. (Sem estado.)
- Performance (não há todo o **overhead** de SOAP).
- Forma simples de se criar um cliente.
- Possibilidade de cache.
- Sem necessidade de contrato formal.

Mundo real

Somos empreendedores, programadores e adoramos cerveja.

Que tal criar uma loja online de cervejas e com suporte a aplicação mobile?

O que utilizar?

- Começamos a avaliar as tecnologias e as funções que desejamos fornecer.
- Loja online
 - Website
- Aplicativo mobile
 - Android, iOS, Windows Phone, etc
- Integração com redes sociais
 - Facebook, twitter, etc

HTML tradicional versus REST

- Vamos considerar a tela de cadastro de clientes do site...

```
<html>
  <body>
    <form action="/cadastrar" method="post">
      <input type="text" name="nome" />
      <input type="text" name="dataNascimento" />
      <input type="submit" value="Cadastrar" />
    </form>
  </body>
</html>
```

HTML tradicional versus REST

- Este código parece atendê-lo bem.
- No entanto, após submeter o formulário, o servidor deve trazer uma pagina **HTML** como resposta.
- **HTML** não seria o ideal em um dispositivo Android ou iOS.

WebServices

- O próximo cenário em mente seria um webservice tradicionais, ou seja, **SOAP**.
- Transferência de dados via **XML**
- Podemos ter clientes em JavaScript (browsers) e também em Android ou iOS.

WebServices

- Porém, logo percebemos que SOAP não é tão simples assim.
- Vejamos um exemplo de consulta de cliente:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <listarClientes
      xmlns="http://geladaonline.com.br/administracao/1.0/service" />
  </soap:Body>
</soap:Envelope>
```

- Eis a resposta:

```
<soap:Envelope xmlns:domain=
    "http://geladaonline.com.br/administracao/1.0/domain"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <listarClientesResponse
            xmlns="http://geladaonline.com.br/administracao/1.0/service">
            <domain:clientes>
                <domain:cliente domain:id="1">
                    <domain:nome>Alexandre</domain:nome>
                    <domain:dataNascimento>2012-12-01</domain:dataNascimento>
                </domain:cliente>
                <domain:cliente domain:id="2">
                    <domain:nome>Paulo</domain:nome>
                    <domain:dataNascimento>2012-11-01</domain:dataNascimento>
                </domain:cliente>
            </domain:clientes>
        </listarClientesResponse>
    </soap:Body>
</soap:Envelope>
```

WebServices

- Estudando um pouco mais, nos deparamos com serviços **REST**.
- Logo notamos a sua simplicidade.
- Por exemplo, para realizar a consulta de clientes, basta abrir o navegador e digitar a URL:

<http://localhost:8080/cervejaria/clientes>

- Eis a resposta:

```
<clientes>
  <cliente id="1">
    <nome>Alexandre</nome>
    <dataNascimento>2012-12-01</dataNascimento>
  </cliente>
  <cliente id="2">
    <nome>Paulo</nome>
    <dataNascimento>2012-11-01</dataNascimento>
  </cliente>
</clientes>
```

Mas a final, o que aconteceu?

Princípios REST

- REST é um estilo de desenvolvimento de Webservices
- Foi criado pelo co-autor do **HTTP**
- É guiado pelo que seriam as boas práticas de uso do protocolo **HTTP**
 - Uso adequado dos métodos HTTP;
 - Uso adequado de URLs;
 - Uso de códigos de status padronizados para representação de sucessos ou falhas;
 - Uso adequado de cabeçalhos HTTP;
 - Interligações entre vários recursos diferentes.

Marco zero

- Orientado a Recursos.
 - Conjuntos de dados trafegados pelo protocolo.
- Recursos possuem identificados únicos representados por uma URL
- Exemplo:

<http://localhost:8080/cervejaria/clientes>

URL

- Vamos decompor: <http://localhost:8080/cervejaria/clientes>
- **http://** – indica o protocolo que está sendo utilizado
- **localhost:8080** – indica o servidor de rede que esta sendo utilizado e a porta
- **cervejaria** – indica o **contexto** da aplicação, ou seja, a raiz pela qual a aplicação esta sendo fornecida para o cliente
- **clientes** – é o endereço, de fato, do recurso – no caso, a listagem de clientes.

Elaborando a URL

- Se <http://localhost:8080/cervejaria/clientes> retorna todos os clientes cadastrados
- Então como recuperar um cliente específico?
- Neste caso, a URL mais correta seria
<http://localhost:8080/cervejaria/clientes/2>

- Eis a resposta:

```
<cliente id="2">
    <nome>Paulo</nome>
    <dataNascimento>2012-11-01</dataNascimento>
</cliente>
```

Prototipando (Rest-01)

- Vamos desenvolver um protótipo de um webservice REST
- Precisamos de:
 - Uma linguagem de programação web com suporte a HTTP
 - Uma fonte de dados, arquivo, banco de dados, etc
- Executar URL:

<http://localhost:8080/Rest-01/clientes>

Dúvidas



Alguns conceitos importantes

Parâmetros

- Em nosso exemplo prático não foram utilizados parâmetros, mas eles existem!
- Parâmetros podem ser utilizados para modificar ou descrever recursos.
- Há basicamente três tipos:
 - **Path param**
 - **Query param**
 - **Body param**

Path param

- Path param são passados no caminho da URL, e são recomendados quando o parâmetro é obrigatório.

<http://servidor/sistema/recurso/param>

- Vamos imaginar que desejamos recuperar a cerveja 2:

<http://localhost:8080/cervejaria/cervejas/2>

Query param

- Query param são passados através da URL e são comumente utilizados para parâmetros opcionais.

<http://servidor/sistema/recurso?nomeParam=valorParam>

- Vamos imaginar que desejamos recuperar todas as cervejas do tipo LAGER:

<http://localhost:8080/cervejaria/cervejas?tipo=LAGER>

Query param

- Query param são úteis, porém possuem suas limitações
- Suponha a necessidade de uma pesquisa de clientes com seus impostos devidos em um certo período.
- Para um imposto:

/pessoas?imposto.1=IR&data.inicio.1=2011-01-01&data.final.1=2012-01-01

Body param

- Para parâmetros complexos podemos utilizar o body param
- O parâmetro é enviado no corpo da requisição. Sem limitações.

```
<impostos>
  <imposto>
    <sigla>IR</sigla>
    <dataInicio>2011-01-01</dataInicio>
    <dataFinal>2012-01-01</dataFinal>
  </imposto>
  <imposto>
    <sigla>IPVA</sigla>
    <dataInicio>2013-01-01</dataInicio>
    <dataFinal>2014-01-01</dataFinal>
  </imposto>
</impostos>
```

Semântica de recursos

- REST é baseado em recursos, que possuem identificados próprios (URL)
- Podemos assumir que uma cerveja é um recurso.
- Desta forma, uma boa URL para recuperar cervejas pode ser:

/cervejas

Semântica de recursos

- De acordo como o modelo REST, esta URL deve ser capaz de interagir com todas as cervejas.
- Para cervejas específicas, podemos utilizar o código de barras.
- Suponha uma cerveja Erdinger Weissbier com código de barras 123456.

Semântica de recursos

- Desta forma, deve ser possível buscar esta cerveja através do código de barras, com a URL

/cervejas/123456

- Regras semelhantes aplicam-se a **qualquer** recurso.

Semântica de recursos

- Clientes tem tanto um identificador gerado pelo banco de dados quanto os seus CPFs:
 - Cliente numero 1: CPF 445.973.986-00
 - Cliente numero 2: CPF 428.193.616-50
 - Cliente numero 3: CPF 719.760.617-92
- Então para buscar o cliente de numero 1, deve ser possível utilizar tanto a URL **/cliente/1** quanto a URL **/cliente/44597398600**

Interação por métodos

- Para interagir com os recursos, utilizamos os métodos HTTP.
- URL são substantivos e os métodos são os verbos.
- Estas modificações são padronizadas, de maneira que:
 - **GET** – recupera os dados identificados pela URL;
 - **POST** – cria um novo recurso;
 - **PUT** – atualiza um recurso;
 - **DELETE** – apaga um recurso.

Interação por métodos

- Métodos podem ser ligados diretamente a operações do banco de dados.
- **GET** **/cliente/1** (para recuperar o cliente 1)
- **POST** **/cliente** (para criar um novo cliente)
- **PUT** **/cliente/1** (para atualizar este cliente 1)
- **DELETE** **/cliente/1** (para apagar o cliente 1)

Códigos de status e erros

- O protocolo HTTP possui códigos para os mais diversos status e erros que possam ocorrer dentro de um servidor Web.
- Faixa 100 a 199 para Informativa
- Faixa 200 a 299 para Sucesso
- Faixa 300 a 399 Redirecionamento
- Faixa 400 a 499 Erro de cliente
- Faixa 500 a 599 outros erros do Servidor

Prototipando (Rest-02)

- Consulta de cervejas estocadas.
- Consulta de uma cerveja específica.
- Inclusão de novas cervejas.
- Executar URL:

<http://localhost:8080/Rest-02/cervejas>

Dúvidas



Simplificando a simplicidade

- Especificação **JSR 311** (Conhecida como **JAX-RS** Java API for RESTful Web Services)
- **Jersey** (implementação de referência da JSR 311)
- Fornece todos os recursos necessários para trabalhar com REST
- Simplifica aquilo que já era simples e inclui várias funcionalidades úteis.

Prototipando (Rest-03)

- Configuração do **JAX-RS** com **Jersey**.
- Implementando um CRUD
 - Cadastro
 - Consulta
 - Alteração
 - Exclusão
- Executar URL:

<http://localhost:8080/Rest-03/cervejas>

Dúvidas



Obrigado !



Bibliografia

- Livro: REST – **Construa API's inteligentes de maneira simples.** – Por Alexandre Saudate (Casa do código)
- Minicurso: **RESTful WebServices** – Por Rafael Nunes (Global Code)
- Código disponível em:

<https://github.com/douglasjunior/MinicursoRest-VICONCCEPAR>