PÓS-GRADUAÇÃO ALFA

UNIALFA
CENTRO UNIVERSITÁRIO
ALVES FARIA

Diego Américo Guedes

PROGRAMAÇÃO COM FRAMEWORKS E COMPONENTES

GRUPO JOSÉ ALVES

Motivação

- Modelo tradicional de aplicações web
  - Cliente: navegadores
  - Servidor: páginas web e outros recursos
  - Localização: URL
  - Protocolo: requisição resposta via HTTP
- Problema
  - Navegador: cliente de propósito geral
  - Restringe o escopo das aplicações

Motivação

- No modelo cliente-servidor original, tanto o cliente quanto o servidor executam funcionalidades específica
- Serviço Web
  - Permite retornar ao modelo cliente-servidor original na Internet
  - Clientes específicos interagem com servidores específicos pela Internet
  - **Provem uma infraestrutura para manter uma forma mais rica e estruturada de interoperabilidade entre cliente e servidor na Internet**
  - Permitem aplicações complexas serem desenvolvidas na Internet

Arquitetura

- Componentes
  - Servidor : Serviço Web
  - Cliente: Qualquer aplicação de propósito específico
- Protocolo de comunicação
  - HTTP
- Representação externa de dados
  - XML
- Localização
  - URI (URL ou URN)

Serviço Web

- Uma interface de serviço web especifica um conjunto de operações para manipular um ou mais recursos
- Essas operações podem ser implementadas por um programa, um objeto ou um banco de dados
- Um serviço web pode ser gerenciado por um servidor web ou um servidor completamente separado.
- Exemplos: Sites da amazon, yahoo, google, ebay
  - Na amazon: serviços web permitem programas obter informações sobre produto, efetuar compras e verificar status de transações

Propriedades da programação baseada em serviços web

- Separação clara entre interface e implementação de serviços
- Permite comunicação síncrona e assíncrona
- Mensagens são representadas em XML
  - Vantagem: a representação textual de mensagem é mais fácil de ser compreendida pelo humano
  - Desvantagem: o tamanho das mensagens são maiores e o parse das mensagens é mais lento que mensagens binárias

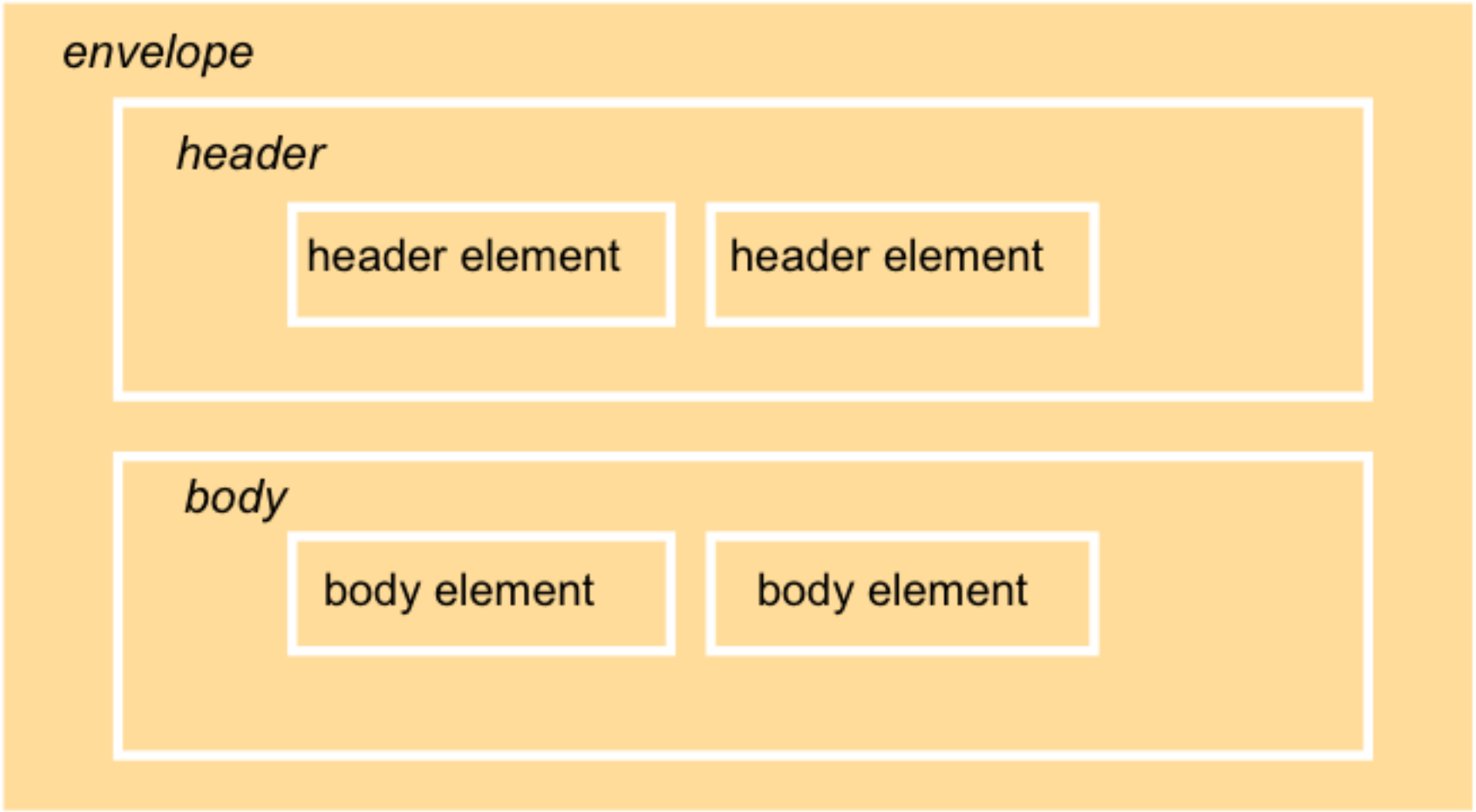
SOAP

- Simple Object Access Protocol (SOAP)
- Especifica as regras para usar XML para empacotar mensagens que serão enviadas a um serviço web
- Define um esquema para usar XML para representar o conteúdo de uma requisição ou de uma resposta de um serviço web
- SOAP é uma extensão de XML-RPC

SOAP

- O protocolo SOAP especifica
  - Como XML deve ser usado para representar o conteúdo de mensagens
  - Como um par de mensagens individuais podem ser combinadas para produzir um padrão requisição-reposta
  - As regras que definem como os receptores de mensagens devem processar os elementos da mensagem

Mensagem SOAP



Mensagem SOAP

URI para o esquema que descreve Envelope SOAP

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

URI para a descrição do serviço stock

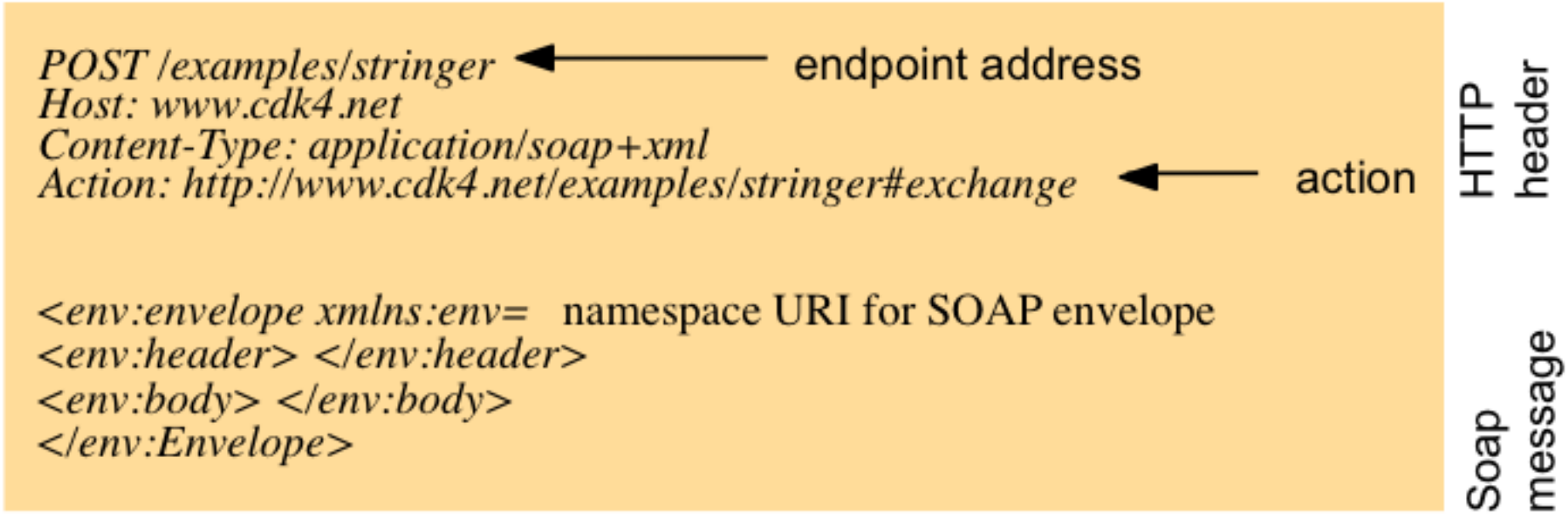
Mensagem SOAP

- Descrição do serviço
  - Cliente: usa a descrição do serviço para gerar o corpo da mensagem.
  - Servidor: usa a descrição do serviço para fazer o parse da mensagem e validar seu conteúdo.

Transporte de Mensagem SOAP

- Mensagens SOAP são independentes do tipo de transporte usado
- Elas não contem referências para o endereço do destino
- Portanto, é necessário um protocolo de transporte para enviar uma mensagem SOAP ao seu destino

Transporte de Mensagem SOAP

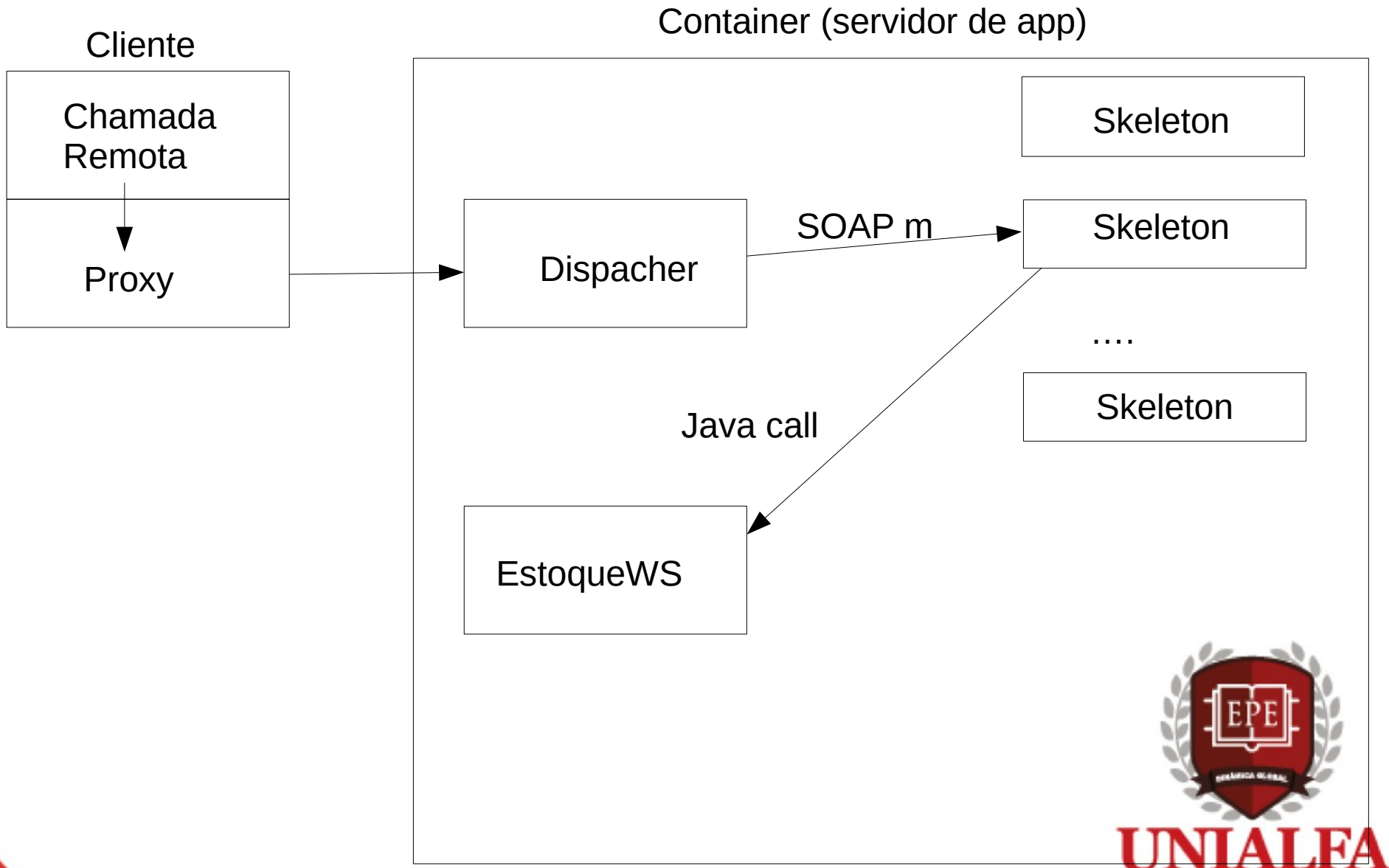


			header	corpo
POST	//www.cdk4.net/examples/stringer	HTTP/ 1.1		

Java SOAP

- A API utilizada para desenvolver web services e cliente sobre SOAP é chamada de JAX-WS
- JAX-RPC mapeia alguns dos tipos de Java para definições XML
- Interface de serviço
  - Deve extender a interface Remote
  - Os métodos devem lançar a exceção RemoteException
  - Parâmetros dos métodos e tipos de retorno devem ser permitidos pelos tipos JAX-WS

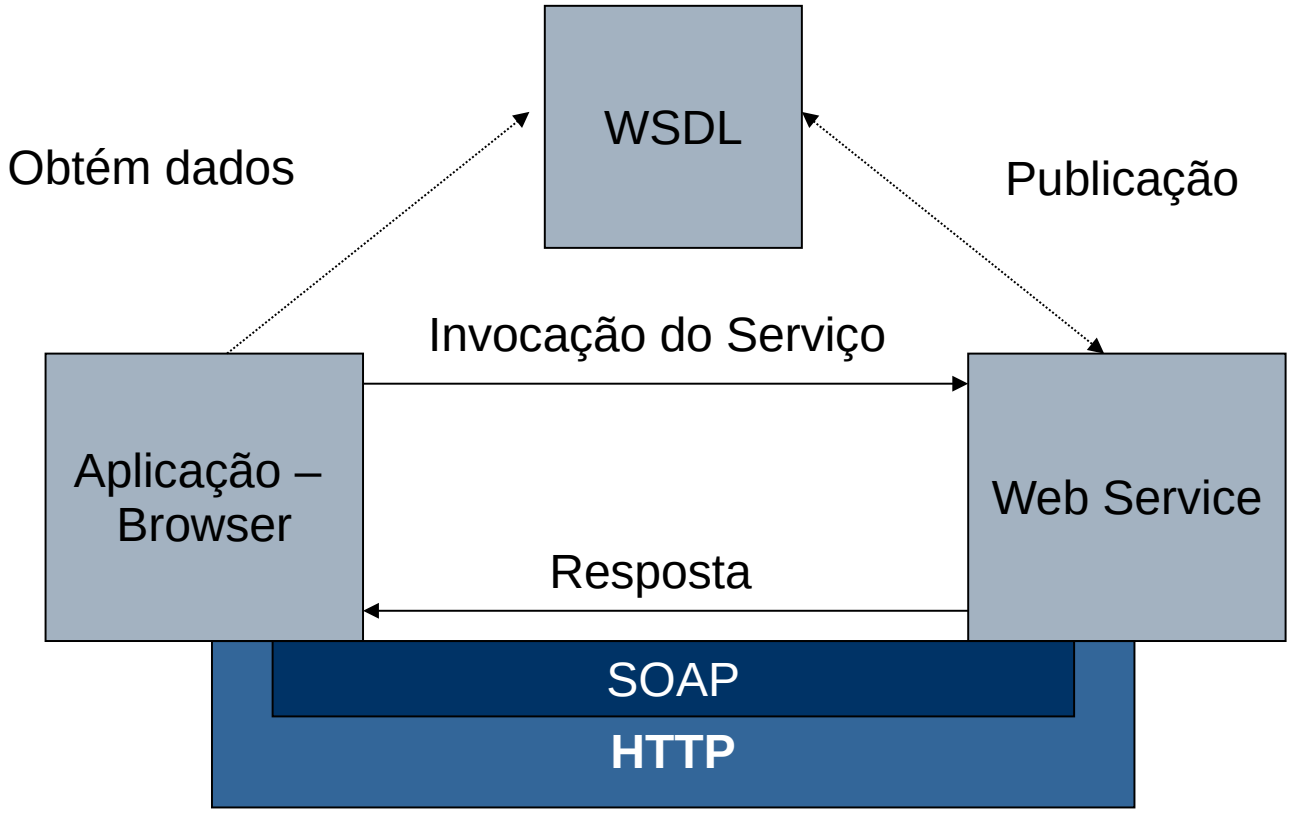
Java SOAP


Container (servidor de app)
Cliente
Chamada Remota
Proxy
Dispacher
SOAP m
Skeleton
Skeleton
....
Skeleton
Java call
EstoqueWS

Java SOAP

- Segue a mesma arquitetura de implementação do Java RMI
- Componentes da arquitetura de implementação do Java SOAP:
  - Módulo de comunicação
  - Proxy cliente
  - Dispatcher e Skeleton
- Não existe módulo de referência remota

WSDL


WSDL
Obtém dados
Publicação
Invocação do Serviço
Aplicação – Browser
Web Service
Resposta
SOAP
HTTP

WSDL

- *Web Service Description Language*
- Gramática XML para descrição de um serviço:
  - O que ele faz?
  - Onde esta localizado?
  - Qual o formato de sua interface?
  - Como acessar o serviço?
- Permite que o cliente aprenda como interagir com o provedor
- Usada também para estruturar as mensagens trocadas

WSDL

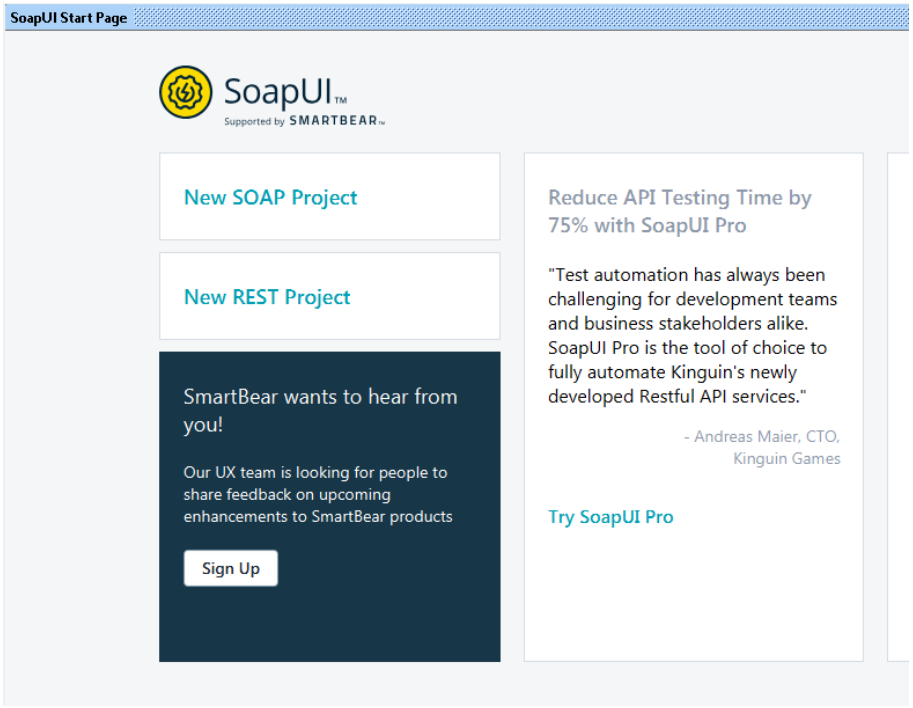
- O cliente não precisa ser associado com um proxy estático.
  - Dado que existe uma descrição do serviço (WSDL), proxies podem ser criados dinamicamente.

SOAP

- Baixar os arquivos no diretório soap do github
- Criar um projeto java e colar os arquivos
- Iniciar a classe “PublicaEstoqueWS”
- Acesse o serviço pelo browser com a URL http://localhost:8080/estoquews
- Agora com a URL http://localhost:8080/estoquews?wsdl

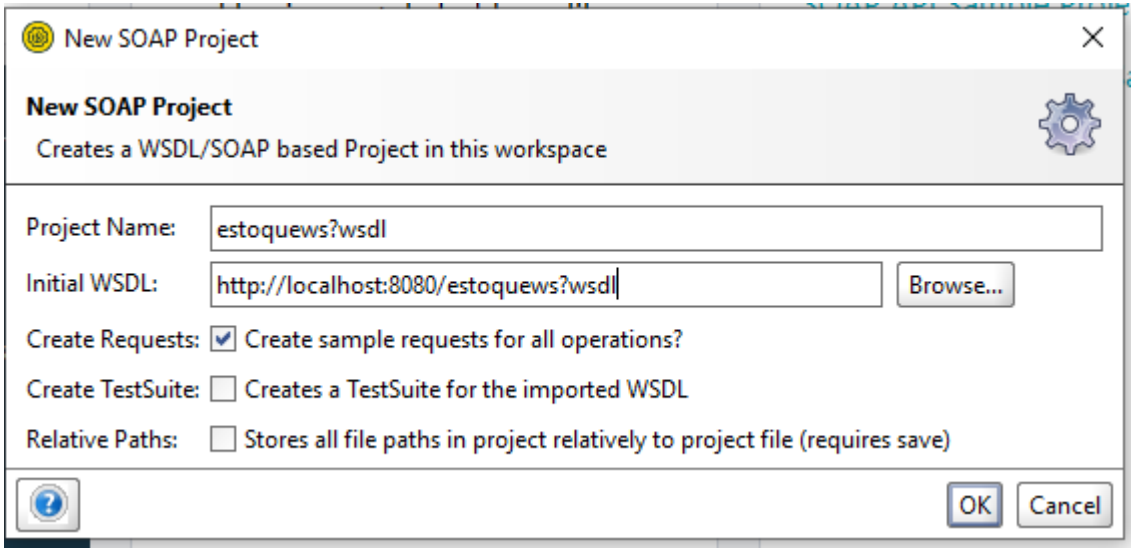
SOAP

- Baixar a ferramente SoapUI
- Clicar em “New SOAP Project”



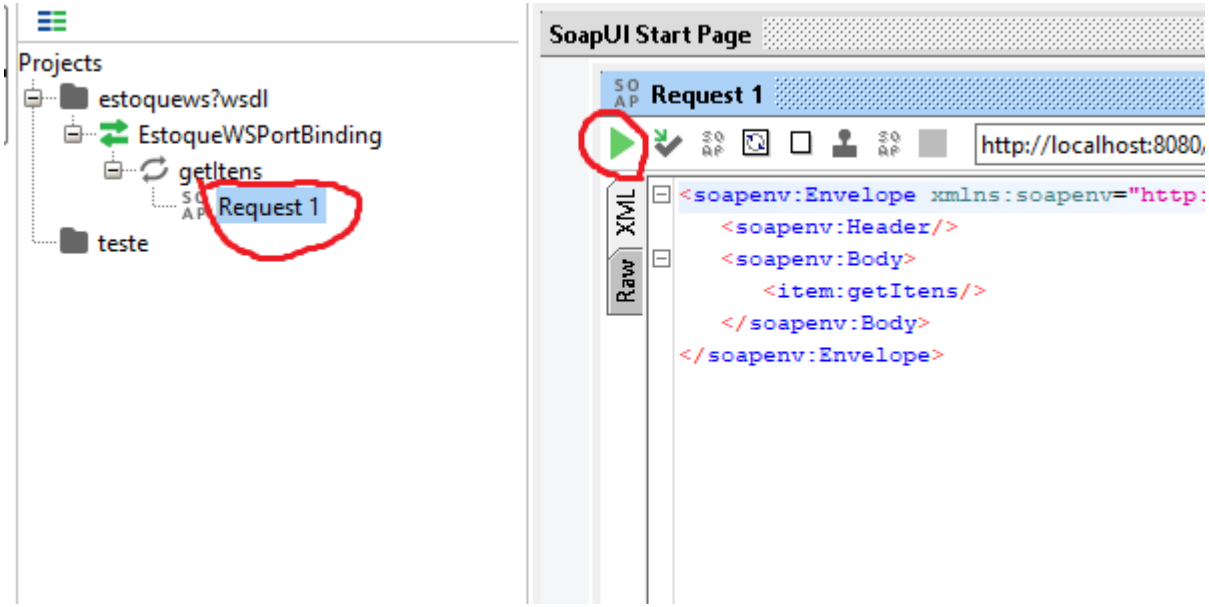
SOAP

- Copiar “http://localhost:8080/estoquews?wsdl” e colar em “Initial WSDL”



SOAP

- Clique no arquivo “Request 1”
- Clique no botão verde de play



REST

Prof. Me. Diego Guedes

SOAP

- SOAP é uma tecnologia desenvolvida pela Microsoft para acessar web services unicamente por meio de mensagens de requisições e respostas feitas em XML, substituindo as tecnologias que utilizavam mensagens binárias.
- Utiliza XML para criar o arquivo WSDL (Web Service Description Language), que é um contrato entre o provedor e o consumidor do serviço, como se fosse uma assinatura de método para o serviço web.
- SOAP é que é independente do protocolo de transporte, ou seja, pode ser enviado com a maioria dos protocolos, por exemplo HTTP, SMTP, TCP e JMS.

REST

- REST é uma arquitetura criada para ser mais simples de se usar que o SOAP.
- Este pode ser usado em vários formatos de texto, como CSV (Comma-separated Values), RSS (Really Simple Syndication), JSON e YAML.
- Porém, só pode ser utilizado com o protocolo HTTP/HTTPS, por exemplo utilizando os métodos GET, POST, PUT e DELETE.
- Por este motivo, o REST comporta-se como se fosse um navegador que sabe como usar um protocolo e seus métodos padronizados, sendo que o aplicativo deve se adequar a isto.
- Neste caso, os padrões do protocolo não são violados, por exemplo criando métodos extras, aproveitando assim os métodos nativos e criando as ações com eles em seu tipo de mídia.

REST

- Rest não impõe restrições ao formato da mensagem, apenas no comportamento dos componentes envolvidos.
- A maior vantagem do protocolo REST é sua flexibilidade
- Quase sempre Web Services que usam REST são mais "leves" e, portanto, mais rápidos.

SOAP ou REST?

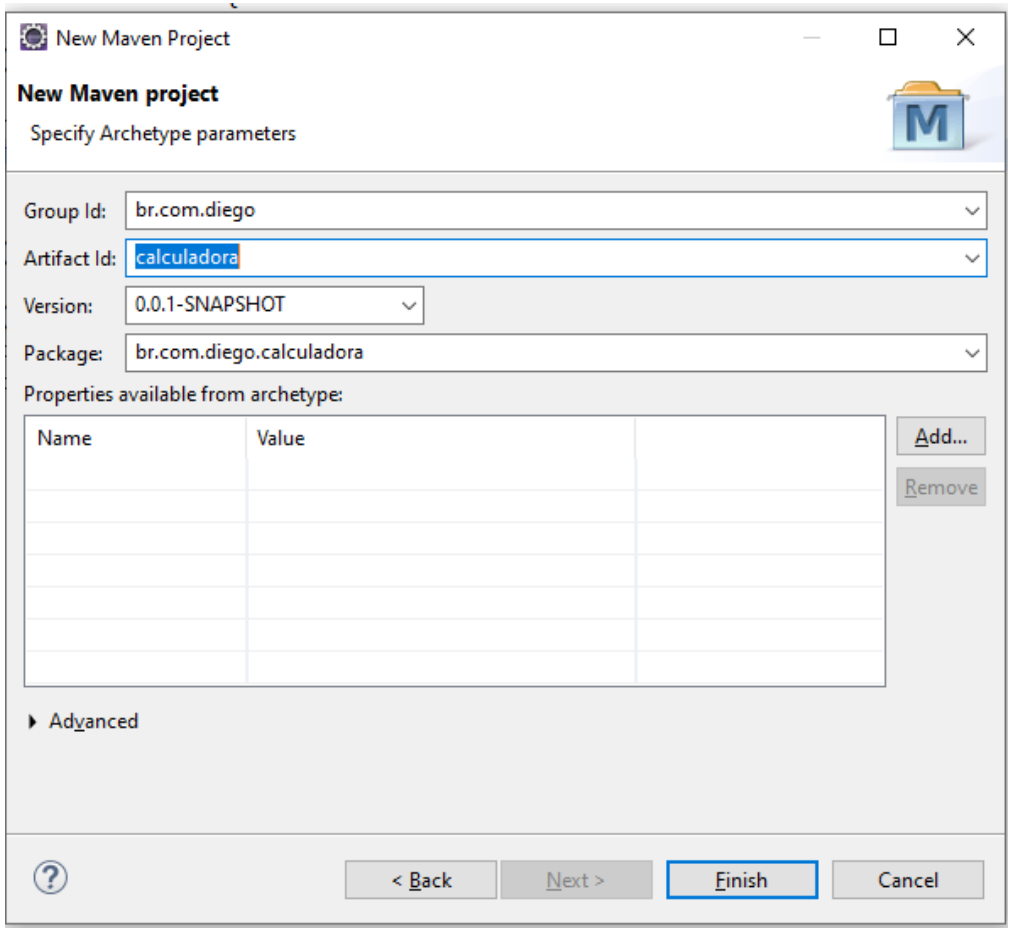
- Em geral, SOAP é uma boa opção para instituições com padrões rígidos e ambientes complexos (várias plataformas e sistemas).
- Muitas ferramentas corporativas (como ESB) tiram vantagem do padrão e possibilitam filtrarem, enfileiramento, classificação e redirecionamento das mensagens trocadas entre sistemas.
- No restante, para uso no dia-a-dia, tende-se a usar REST e JSON.

Rest em Java

- Para trabalhar com Rest em Java, pode-se utilizar o Spring MVC

Configurando o projeto

- New project → New Maven Project
- Criar o pacote br.com.diego.calculadora
-



- Adicione esta dependência ao pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>1.3.6.RELEASE</version>
</dependency>
```

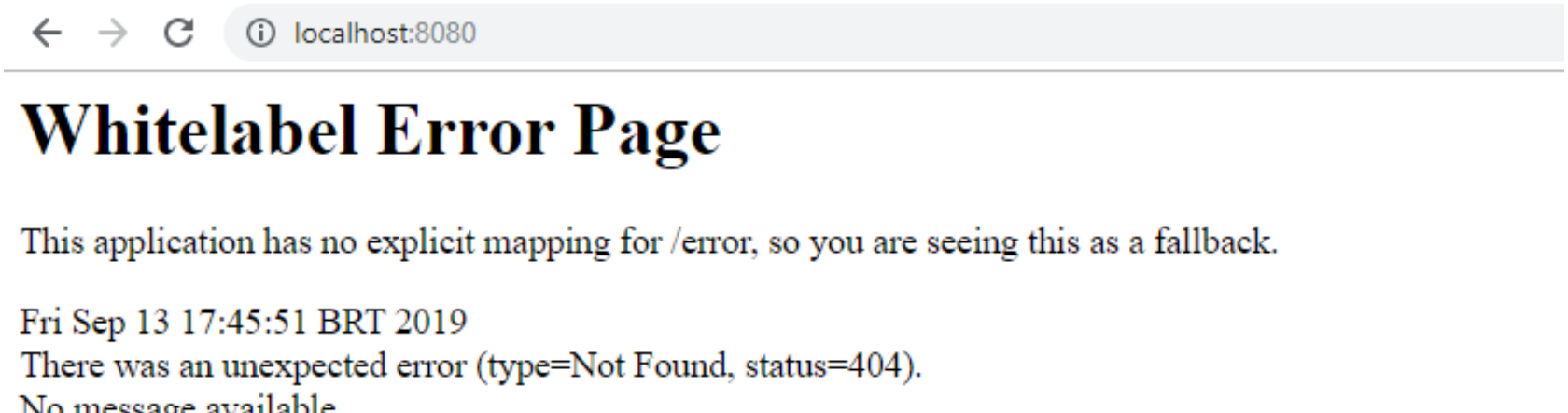
- A vantagem de usar Spring Boot é que não precisamos mais nos preocupar com a instalação e configuração do projeto em um container (tomcat, JBOSS, etc).
- Precisamos apenas configurar o Spring Boot para que inicie um container automaticamente e gerencie todos os nossos Beans.
- Faremos isso por meio das classes Java.

Configurando o projeto - Spring Boot

```
package br.com.diego.calculadora;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication
public class WebApplication {
public static void main(String[] args) {
SpringApplication.run(WebApplication.class, args);
}
}
```


localhost:8080
Whitelabel Error Page
This application has no explicit mapping for /error, so you are seeing this as a fallback.
Fri Sep 13 17:45:51 BRT 2019
There was an unexpected error (type=Not Found, status=404).
No message available

- Temos que ter uma classe e método que irá responder pelas solicitação na raiz do projeto (/)
- Assim, temos que criar uma classe controller e um método que responde pelo serviço /

Configurando o projeto

```
package br.com.diego.calculadora;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class WebApplication {

@RequestMapping("/")
public String ola() {
return "Ola, Bem vindo ao sistema de calculadora";
}

public static void main(String[] args) {
SpringApplication.run(WebApplication.class, args);
}
}
```

Configurando o projeto

- Vamos dividir a classe de inicialização (Spring Boot) da classe de controller (Spring MVC)

Configurando o projeto

```
package br.com.diego.calculadora;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class WebApplication {

public static void main(String[] args) {
SpringApplication.run(WebApplication.class, args);
}
}
```

Configurando o projeto

```
package br.com.diego.calculadora;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CalculadoraController {
@RequestMapping("/")
public String ola() {
return "Ola, Bem vindo ao sistema de calculadora";
}

}
```

Configurando o projeto

- Vamos criar uma método da adição no controller
- Para chamá-lo:
  - http://localhost:8080/adicao?n1=2&n2=3

Configurando o projeto

```
package br.com.diego.calculadora;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CalculadoraController {
@RequestMapping("/")
public String ola() {
return "Ola, Bem vindo ao sistema de calculadora";
}

@RequestMapping("/adicao")
public int adicao(int n1, int n2) {
return n1 + n2;
}
}
```