



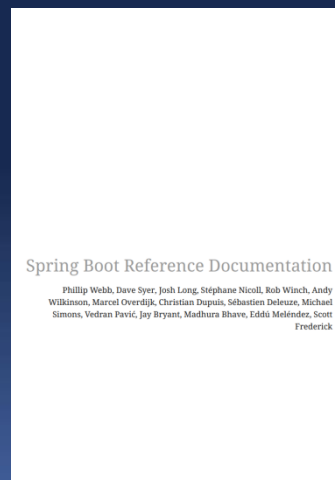
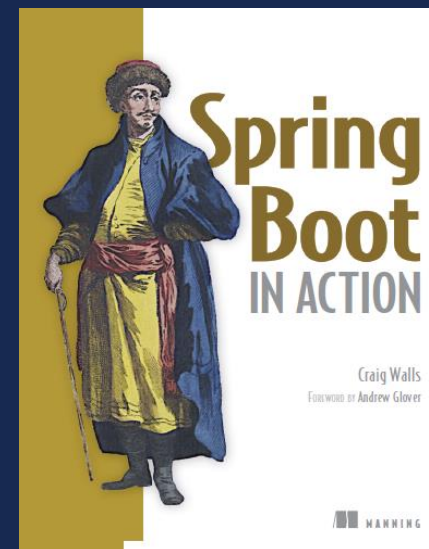
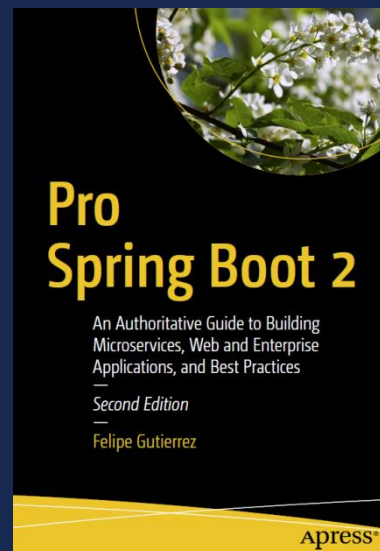
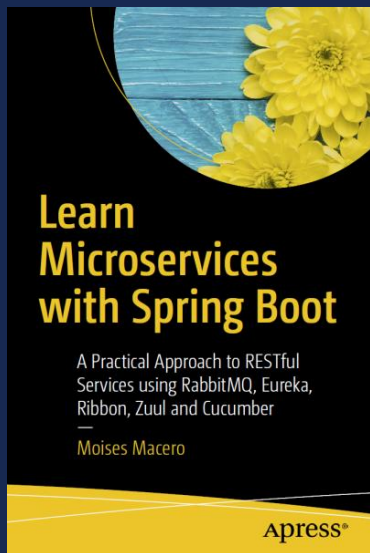
# Spring Boot

## Unidade 2 – API Rest sem Banco de Dados



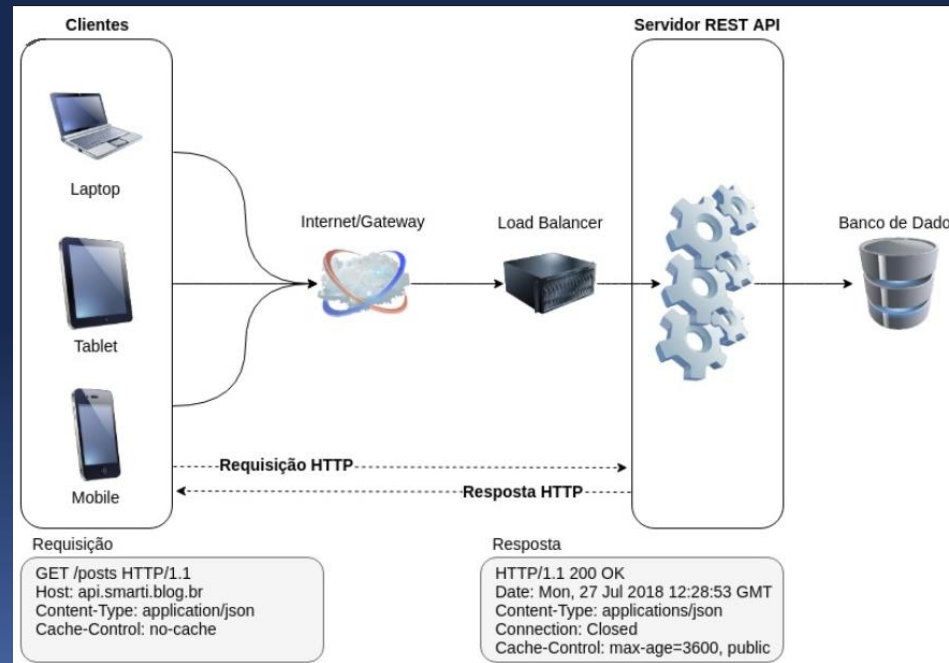
Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPUVSP  
[aparecidovfreitas@gmail.com](mailto:aparecidovfreitas@gmail.com)

# Bibliografia



# API's Rest

- Uma **API** (Application Programming Interface) permite a comunicação entre dois sistemas;
- Uma **API** fornece essencialmente a linguagem e o contrato para a interação entre dois sistemas;
- Cada **API** tem a especificação que determina a forma pela qual as informações podem ser transferidas;



# REST



- **REST** é uma abreviação de "**Representational State Transfer**" ou transferência de estado representacional;
- Trata-se de um modelo de arquitetura para sistemas distribuídos;
- O modelo foi criado por **Roy Fielding**, um dos criadores do protocolo **HTTP**;
- A ideia por trás desse modelo é viabilizar a transferência de dados via rede.



# REST



- **REST** tem sido a base para a evolução do protocolo **HTTP**;
- Tem sido usado como alternativa ao modelo **SOAP** o qual é baseado em **XML**;
- Atualmente, graças à sua flexibilidade, tem sido largamente utilizado na construção de **API's** para integração entre sistemas **Web**;



# Recursos



- Toda aplicação Web gerencia **recursos**;
- Nessa unidade, desenvolveremos uma **API Rest** que manipula cursos;
- À medida em que formos desenvolvendo novas funcionalidades, novos recursos também estarão presentes na aplicação, como por exemplo, disciplinas, professores, grades de curso, etc
- Assim, precisamos de alguma forma **diferenciar** o manuseio de um recurso de outro. Por exemplo, poderíamos ter uma API que retornasse os cursos, outra que retornasse os professores, e assim por diante.
- Para isso, definimos um identificador único para recurso a ser retornado. Esse identificador é chamado URI. Por exemplo, para retornar cursos a URI poderia ser **/cursos**, para professores poderia ser **/professores**, e assim por diante.







Ok ! Cada recurso está associado a uma URI !

Mas, como especificar o que deve ser feito com o recurso ?

# Operações com Recursos

- Imaginemos que a URI **/cursos** está associada ao recurso **"cursos"** ;
- Mas, como especificar o que será feito com o recurso **"cursos"** ?
- Pode haver **diferentes** formas de se tratar o recurso **"cursos"**;
- Por exemplo, pode-se **retornar** uma lista completa dos cursos, pode-se **atualizar** o recurso cursos, pode-se **excluir** o recurso, etc;
- Assim, para se definir a operação a ser utilizada com o recurso, utilizam-se os **verbos HTTP** ou os métodos **HTTP**;
- Por exemplo: **GET** para se recuperar recursos, **POST** para se cadastrar recursos, **PUT** para atualizar ou ainda **DELETE** para se excluir.

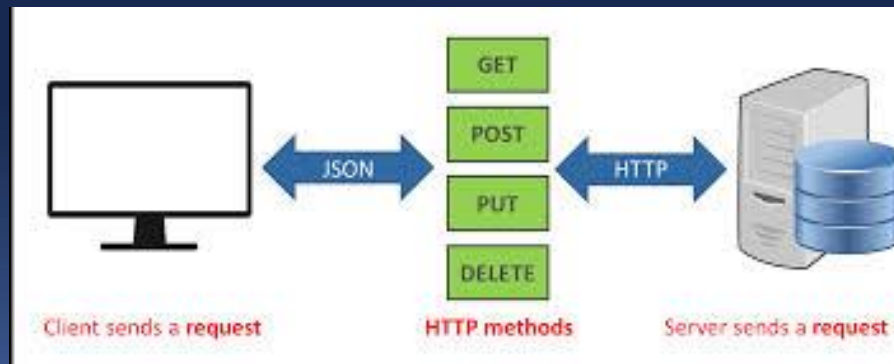


- GET /alunos
- POST /alunos
- PUT /alunos/{id}
- DELETE /alunos/{id}



# API's Rest

- Da mesma forma que uma página **Web** é renderizada pelo Browser, as API's podem usar requisições HTTP para obter informações de uma aplicação ou servidor web;
- API's Rest oferecem uma forma mais **leve**, usando URL's na maioria dos casos, para enviar e receber informações;
- Rest usa quatro verbos HTTP 1.1 diferentes (**GET**, **POST**, **PUT** e **DELETE**) para executar tarefas;
- Os recursos a serem manipulados são representados em um determinado **formato** (Json,XML, texto, etc....), daí o nome **REST**, onde **R** = Representational.



# Representação do recurso

- Embora **Json** tem sido muito usado, outras formas de se representar os dados é possível.

```
{
  "employee": "Max Mustermann",
  "items": [
    {
      "name": "TestData",
      "quantity": "2"
    },
    {
      "name": "Test2",
      "quantity": "3"
    },
    {
      "name": "Test3",
      "quantity": "2"
    }
  ],
  "table": "Tisch 5"
}
```

## JSON

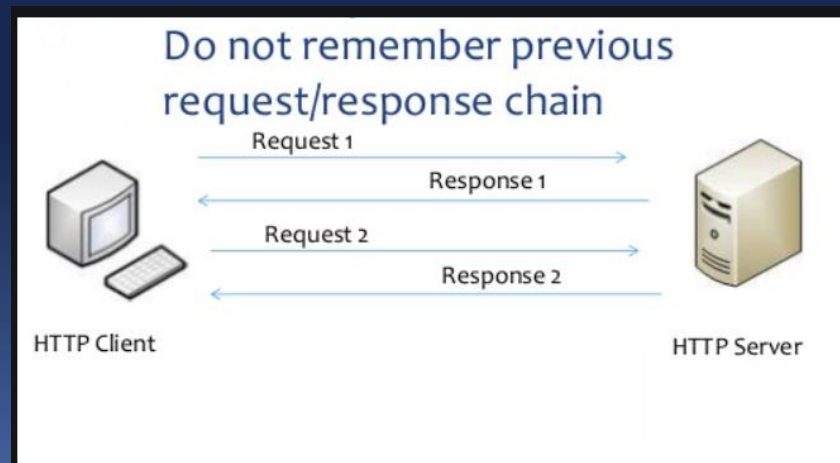
```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second
      president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

## XML

XML

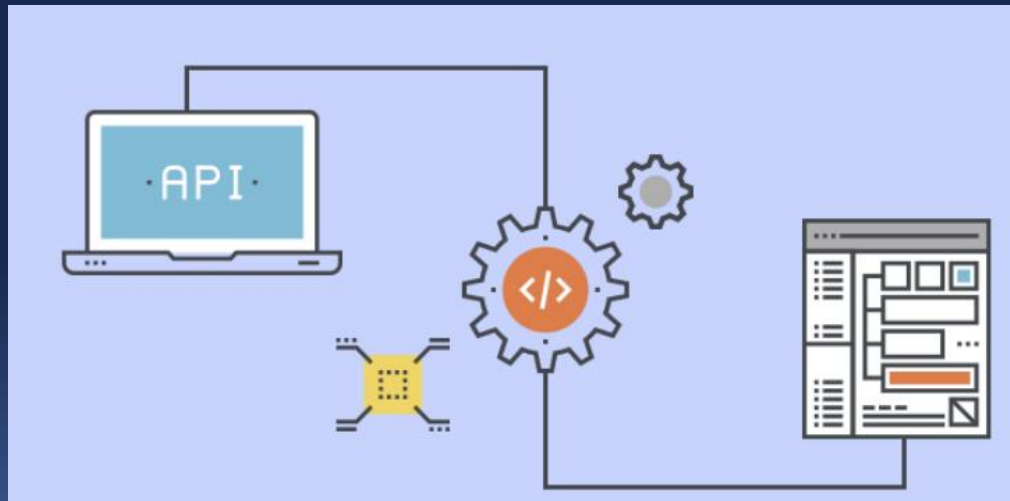
# O nome REST

- **R** como vimos está associado à forma de representação dos dados que são interoperacionalizados na integração das aplicações;
- A comunicação que se estabelece é **stateless**;
- Isso significa que como sendo a web baseada no protocolo HTTP, **NÃO** se guarda o estado da aplicação durante a comunicação;
- Ou seja, a comunicação é integralmente **stateless** (sem armazenamento do estado);
- Assim, quando se processa uma **API Rest**, a informação retornada está associada ao estado corrente, sem portanto, se usar seções para armazenamento de dados.



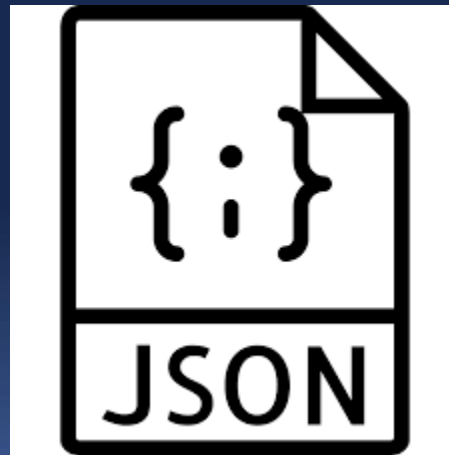
# Rest

- Ao contrário do SOAP, **Rest** não precisa usar **XML** para fornecer a resposta;
- Pode-se encontrar REST-based Web Services que entregam os dados em formato CSV (Command Separated Value) ou JSON (JavaScript Object Notation);
- Um ponto **importante** para ser destacado com REST é que se pode obter uma saída em um formato mais fácil de ser tratado na linguagem da aplicação que irá consumir as informações providas pela **API**;



# JSON

- É uma formatação **leve** para troca de dados;
- Fácil de **ler** e de se **escrever**;
- Baseia-se em um **subconjunto** da linguagem **JavaScript**;
- Formato texto e completamente **independente** de linguagem;
- Essas propriedades fazem com que JSON seja um formato ideal para troca de dados;



# Formato JSON

- Em **JSON** para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado;
- Essa sintaxe é derivada da forma utilizada pelo **JavaScript**, para representar informações;
- Por exemplo, para se representar o ano de 2020, usa-se a seguinte sintaxe:

```
"ano": 2020
```



# Formato JSON

- Um par **nome/valor** deve ser representado pelo nome entre aspas duplas, seguido de dois pontos, seguido do valor;
- Os valores podem possuir apenas 3 tipos básicos: **numérico** (inteiro ou real), **booleano** e **string**;

```
"site": "www.qualitsys.com"
```

```
"nota": 9.5
```

```
"aprovado": true
```

# Formato JSON

- A partir dos dados básicos, é possível construir-se tipos complexos: array e objeto;
- Arrays são delimitados por **colchetes**, com seus elementos separados por vírgulas;
- Por exemplo, abaixo segue um exemplo de um array representado números:

```
[34,6,888,34,23,124]
```

- Segue outro exemplo de um array representando valores booleanos e outro para strings:

```
[true,true,false,true,false,false]
```

```
["SP","RJ","MG","RS","PR","SC"]
```

# Formato JSON

- Segue um exemplo para armazenar uma matriz de inteiros:

```
[  
  [1,8],  
  [5,9],  
  [23,6]  
]
```

# Formato JSON

- Objetos são especificados entre **chaves** e podem ser compostos por múltiplos pares nome/valor, por arrays e também por outros objetos;

Desta forma, um objeto JSON pode representar, virtualmente, qualquer tipo de informação.

```
1 {  
2   "titulo": "JSON x XML",  
3   "resumo": "o duelo de dois modelos de representação de informações",  
4   "ano": 2012,  
5   "genero": ["aventura", "ação", "ficção"]  
6 }
```

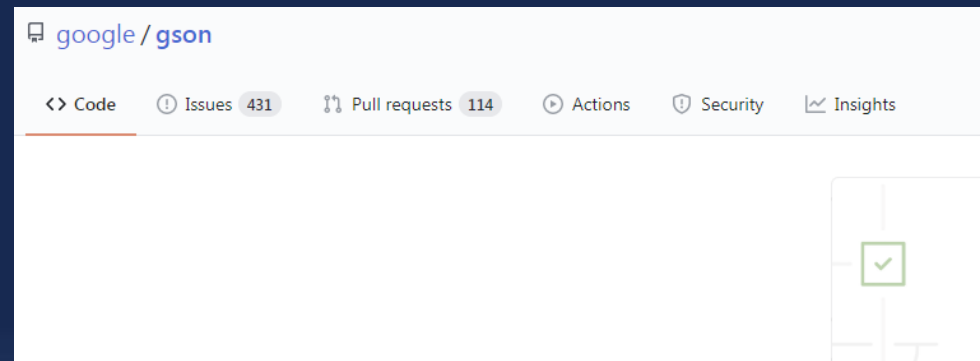
```
1 [  
2   {  
3     "titulo": "JSON x XML",  
4     "resumo": "o duelo de dois modelos de representação de informações",  
5     "ano": 2012,  
6     "genero": ["aventura", "ação", "ficção"]  
7   },  
8   {  
9     "titulo": "JSON James",  
10    "resumo": "a história de uma lenda do velho oeste",  
11    "ano": 2012,  
12    "genero": ["western"]  
13  }  
14 ]
```

# Parsers JSON

- Existem diversos **parsers** disponíveis para a Linguagem Java;
- Por exemplo, destaca-se a biblioteca **google-gson**, desenvolvida pela **Google**, que é bem documentada e relativamente simples de ser usada.

Java

- [JSON-java](#)
- JSONUtil
- jsonp
- Json-lib
- Stringtree
- SOJO
- json-taglib
- Flexjson
- Argo
- jsonij
- fastjson
- mjson
- jjson
- json-simple
- json-io
- google-gson**
- FOSS Nova JSON
- Corn CONVERTER
- Apache johnzon
- Genson
- cookjson
- progbase



# JSON com Google GSON

- Existem diversos **parsers** disponíveis para a Linguagem Java;
- Por exemplo, destaca-se a biblioteca `gson`, desenvolvida pela **Google**, que é bem documentada e relativamente simples de ser usada.

JSON com Google GSON. **JSON** é um acrônimo para “JavaScript Object Notation”, é um formato mais leve que xml e mais entendível para se trafegar dados entre sistemas computacionais, seu uso está sendo cada vez mais adotado por aplicações web e dispositivos móveis da atualidade.

## **Algumas das vantagens são:**

- ✓ Fácil de entender.
- ✓ Parsing facilitado.
- ✓ Suporta objetos.
- ✓ Extremamente leve.
- ✓ Usado pelos maiores serviços da web como Google, Facebook e etc.

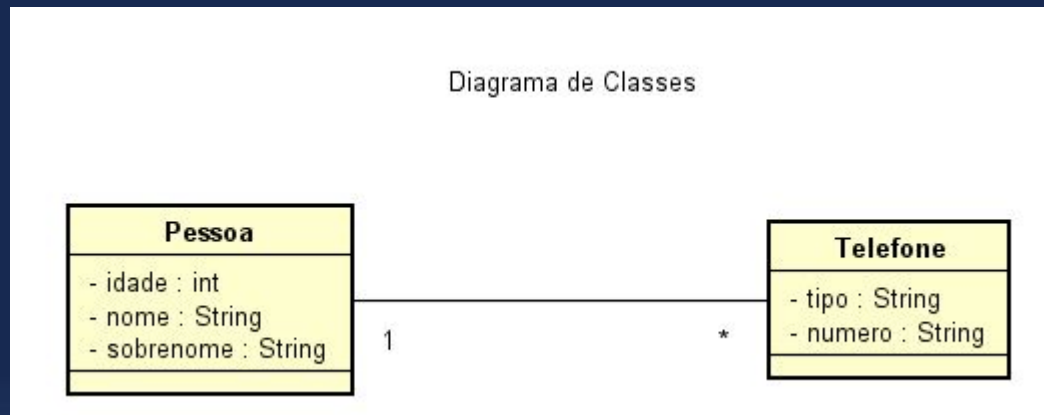
**JSON** é em formato texto e completamente independente de linguagem.

Aplicações web desenvolvidas com Spring MVC e AngularJS fazem o uso extremo de JSON tornando assim uma aplicação leve, robusta, eficiente e com baixo processamento do servidor.



# JSON com Google GSON

- Para exemplificar o uso do **google-gson**, vamos ver um caso simples de um relacionamento 1 x n.



# JSON com Google GSON

Seguem as implementações das classes:

```
01 public class Pessoa {
02
03     private String nome;
04     private String sobrenome;
05     private int idade;
06
07     private List telefones = new ArrayList();
08
09     public void setTelefones(List telefones) {
10         this.telefones = telefones;
11     }
12
13     public List getTelefones() {
14         return telefones;
15     }
16
17     public String getNome() {
18         return nome;
19     }
20
21     public void setNome(String nome) {
22         this.nome = nome;
23     }
24
25     public String getSobrenome() {
26         return sobrenome;
27     }
28
29     public void setSobrenome(String sobrenome) {
30         this.sobrenome = sobrenome;
31     }
32
33     public int getIdade() {
34         return idade;
35     }
36
37     public void setIdade(int idade) {
38         this.idade = idade;
39     }
}
```

```
public class Telefone {

    private String tipo;
    private String numero;

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public String getNumero() {
        return numero;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }
}
```

# API Gson Google

A referência correta do pacote desta biblioteca é **com.google.gson.Gson** e contém dentre outros com dois métodos que são os mais interessantes e usuais, são eles **toJson** e o **fromJson**. Vamos entender o que cada um faz:

- ✓ **toJson** transforma objetos em JSON com saída String.
- ✓ **fromJson** transforma String JSON em objetos novamente.

# Criando os objetos com dados

Nesta parte iremos desenvolver um exemplo simples, criando um objeto pessoa e adicionando telefones a ele.

```
01 Pessoa pessoa = new Pessoa();
02 pessoa.setIdade(29);
03 pessoa.setNome("Java");
04 pessoa.setSobrenome("Avançado");
05
06 Telefone telefone = new Telefone();
07 telefone.setTipo("celular");
08 telefone.setNumero("(44) 5555-8888");
09
10 Telefone telefone2 = new Telefone();
11 telefone2.setTipo("fixo");
12 telefone2.setNumero("(44) 8888-3333");
13
14 pessoa.getTelefones().add(telefone);
15 pessoa.getTelefones().add(telefone2);
```

# Convertendo para JSON

Simplesmente instanciamos um objeto Gson e chamamos o método toJson passando a pessoa e o tipo da classe.

```
1 | String json = new Gson().toJson(pessoa, Pessoa.class);
```

O resultado:

```
01 | {
02 |   "nome": "Java",
03 |   "sobrenome": "Avançado",
04 |   "idade": 29,
05 |   "telefones": [
06 |     {
07 |       "tipo": "celular",
08 |       "numero": "(44) 5555-8888"
09 |     },
10 |     {
11 |       "tipo": "fixo",
12 |       "numero": "(44) 8888-3333"
13 |     }
14 |   ]
15 | }
```

# Convertendo JSON para objeto

Simplesmente instanciamos um objeto Gson e chamamos o método fromJson passando o JSON e o tipo da classe.

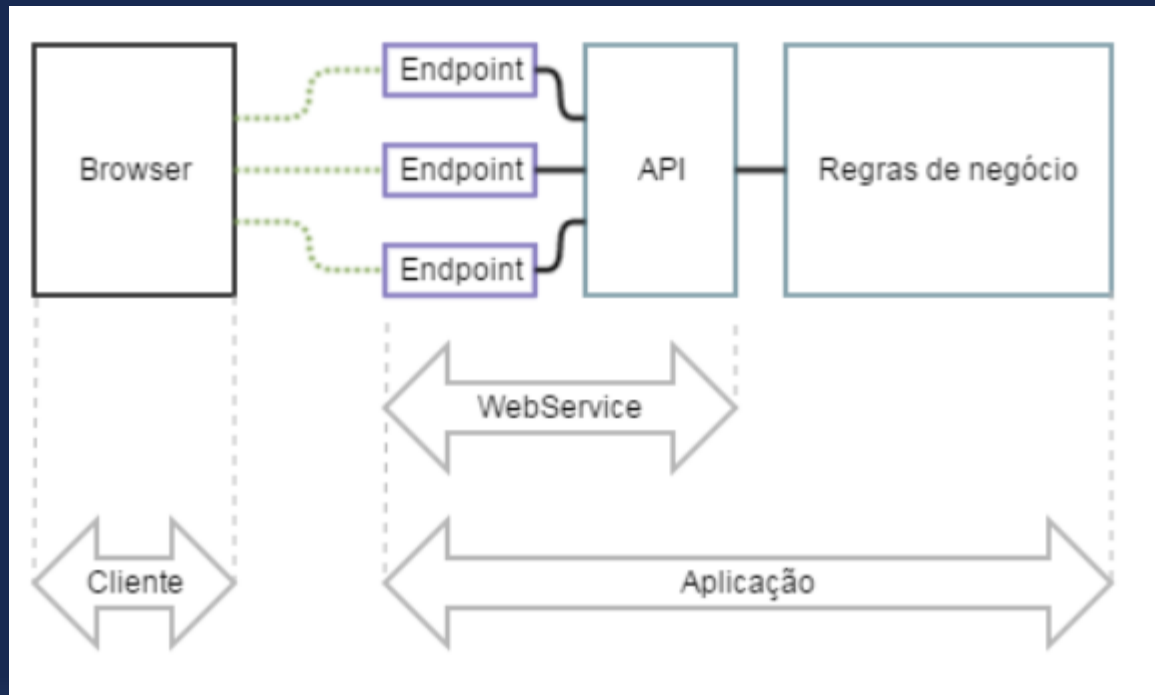
```
01 Pessoa pessoa = new Gson().fromJson("
02 {
03     "nome": "Java",
04     "sobrenome": "Avançado",
05     "idade": 29,
06     "telefones": [
07         {
08             "tipo": "celular",
09             "numero": "(44) 5555-8888"
10         },
11         {
12             "tipo": "fixo",
13             "numero": "(44) 8888-3333"
14         }
15     ], Pessoa.class);
```



# endpoint

- De forma simples, um **endpoint** corresponde a um ponto final de um canal de comunicação;
- Quando uma **API** interage com outro sistema, os pontos de contato dessa comunicação são considerados endpoints;
- Para API's um **endpoint** pode incluir uma URL de um servidor ou serviço;
- Cada **endpoint** corresponde ao local para o qual a **API** pode acessar os recursos necessários para processar sua funcionalidade;
- API's operam com requisições e respostas. Quando uma **API** requisita informações de uma aplicação web ou web server, ela receberá uma resposta;
- O local em que as **API's** enviam solicitações e onde o recurso está localizado é chamado **endpoint**.

# endpoint



# Classe de Domínio

- Nesta unidade, criaremos o nosso primeiro **endpoint**;
- Para isso, consideraremos uma classe que representa os **courses** de uma Universidade;
- Nossa **API** retornará uma **lista** de todos os cursos disponíveis;
- Criaremos um projeto chamada listcursos com spring initializr;
- Implementaremos uma classe chamada **Curso** num package chamado **br.com.qualitsys.model**.

# http://start.spring.io



The screenshot shows the Spring Initializr web application in a browser. The interface is clean and modern, with a white background and green accents. The top navigation bar includes the Spring logo and the text 'spring initializr'. The main content area is divided into several sections: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for various versions, with '2.3.1' selected; 'Project Metadata' with input fields for 'Group' (br.com.qualitsys), 'Artifact' (listcursos), 'Name' (listcursos), 'Description' (Projeto Rest com Spring Boot - listagem de cursos em Memória), and 'Package name' (br.com.qualitsys.listcursos); 'Packaging' with radio buttons for 'Jar' (selected) and 'War'; and 'Java' with radio buttons for '14', '11', and '8' (selected). On the right side, there is a 'Dependencies' section with a button 'ADD DEPENDENCIES... CTRL + B' and a 'Spring Web' section with a 'WEB' tag and a description: 'Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.' At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. The bottom of the browser window shows two tabs: 'listcursos.zip' and 'hellospringboot.zip'. The bottom right corner of the browser window has a 'Mostrar tudo' button.

Spring Initializr

start.spring.io

Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M1) ☐ 2.3.2 (SNAPSHOT) ☒ 2.3.1 ☐ 2.2.9 (SNAPSHOT) ☐ 2.2.8 ☐ 2.1.16 (SNAPSHOT) ☐ 2.1.15

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 14 ☐ 11 ☒ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web **WEB**

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE CTRL + G EXPLORE CTRL + SPACE SHARE...

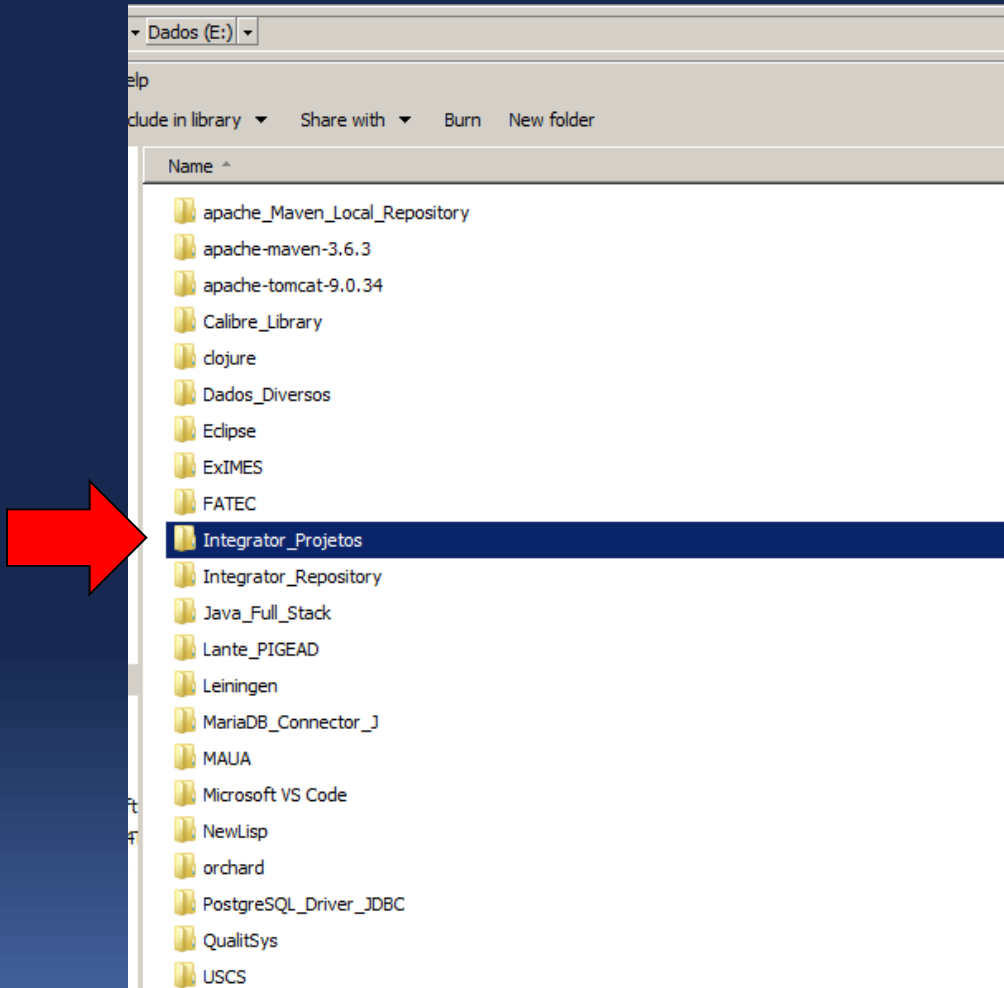
listcursos.zip hellopringboot.zip

Mostrar tudo



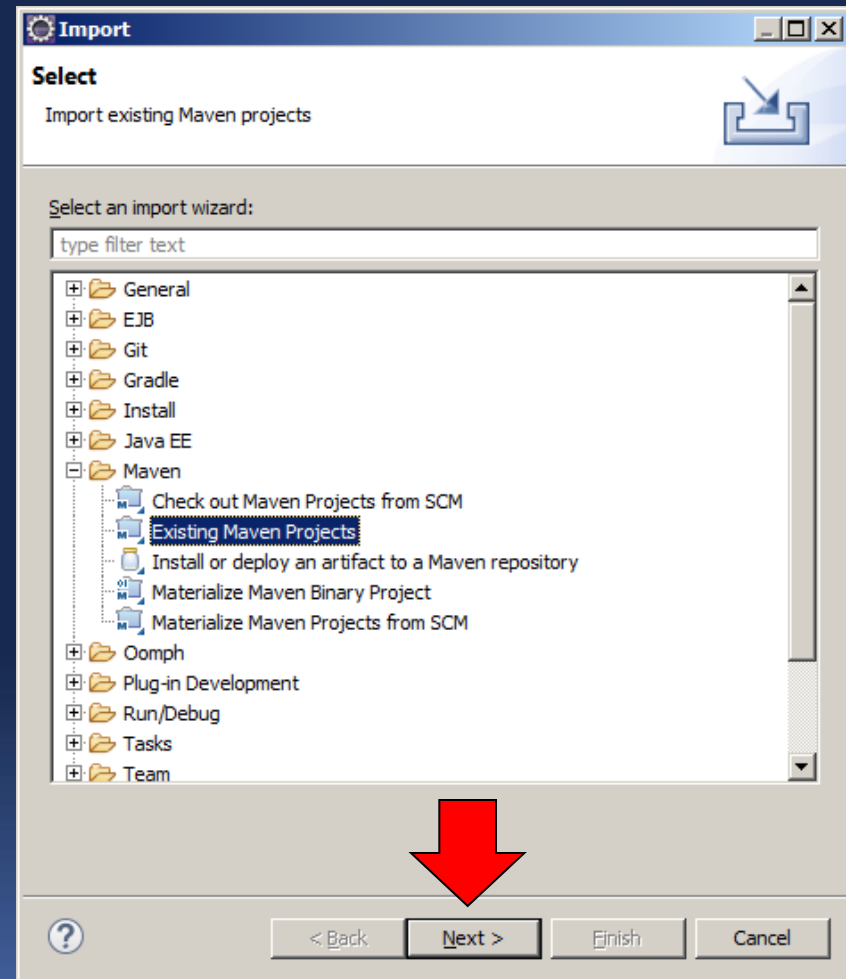
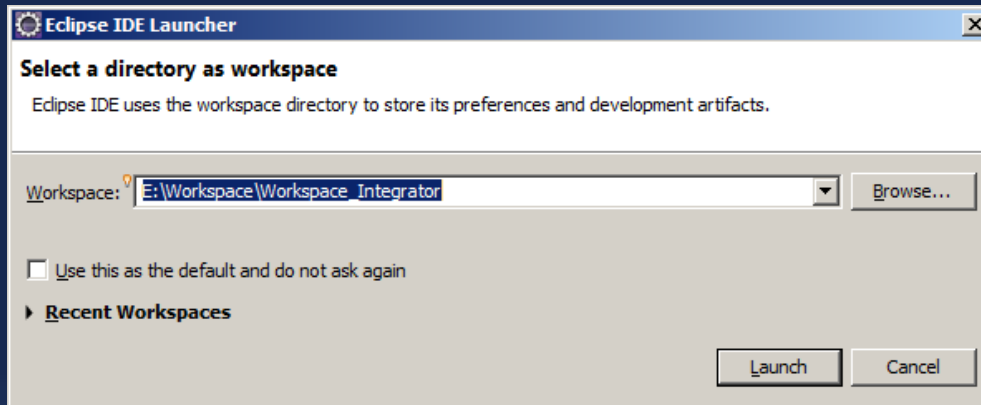
# Inicializar gera projeto em arquivo .zip

- Projeto **Maven** criado !
- Descompactaremos o projeto na pasta **E:\Integrator\_Projetos**



# Abrindo o projeto no Eclipse

- Abriremos o Eclipse no **Workspace/Integrator**
- **Import > Existing Maven Projects**





# Sprint criou uma classe....

```
package br.com.qualitsys.listcursos;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class ListcursosApplication {

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

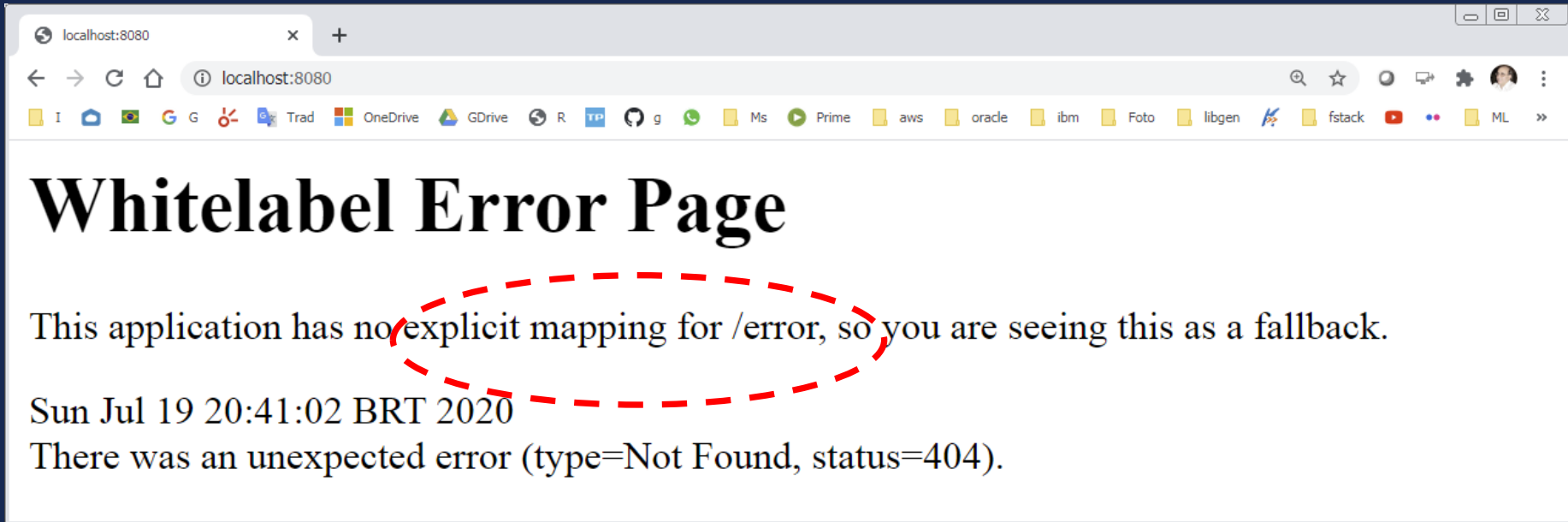
- A classe criada foi **ListcursosApplication** com o método **main**;
- O método **main** chama o método static **run** da classe **SpringApplication** e essa classe tem a anotação **@SpringBootApplication**;
- Essa classe é a classe de execução do projeto (**Tomcat** embarcado no **Spring Boot**).
- No **Spring Boot** essa é a classe principal. É a classe que contém o método **main**!

# Executando a aplicação

```
: Starting ListcursosApplication on Aparecido-PC with PID 9800 (E:\In
: No active profile set, falling back to default profiles: default
: Tomcat initialized with port(s): 8080 (http)
: Starting service [Tomcat]
: Starting Servlet engine: [Apache Tomcat/9.0.36]
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 1304 ms
: Initializing ExecutorService 'applicationTaskExecutor'
: Tomcat started on port(s): 8080 (http) with context path ''
```

# Executando

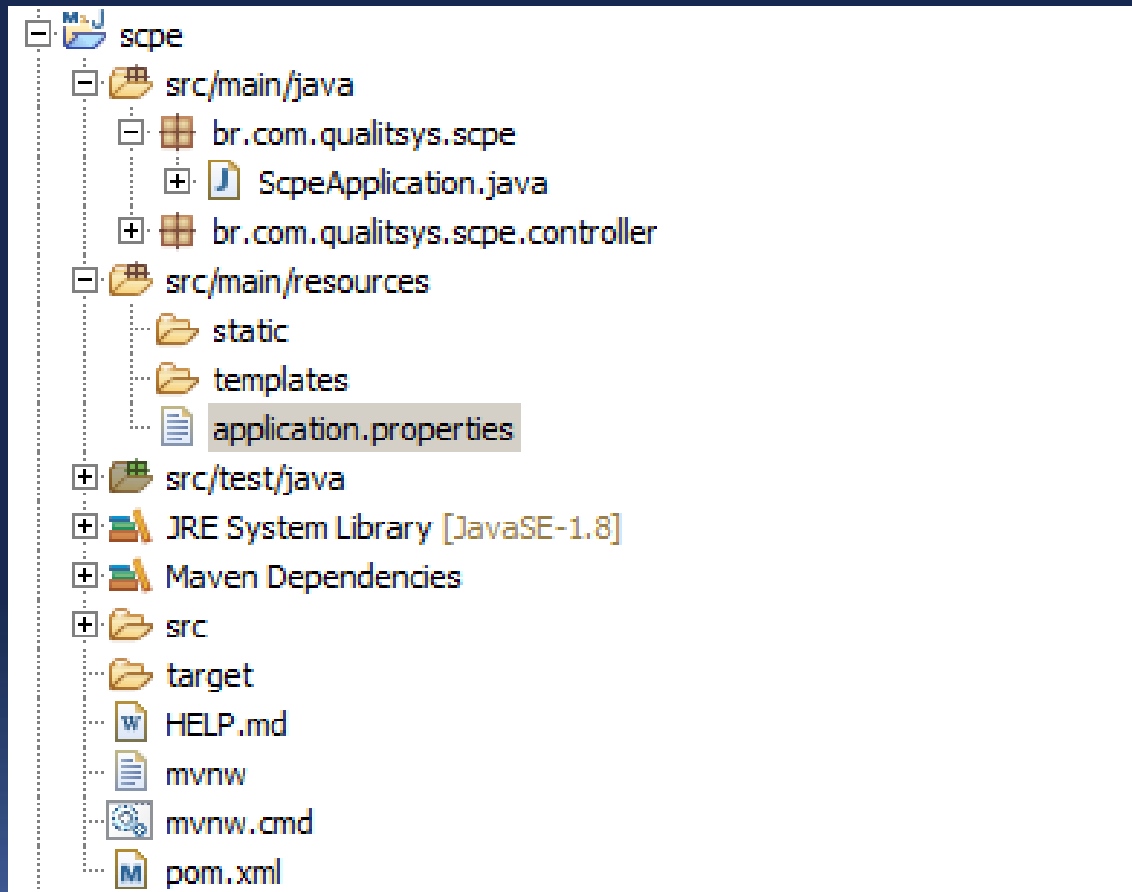
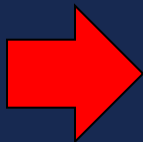
<http://localhost:8080>



- 🕒 **Tomcat já está rodando !!!!**
- 🕒 Foi retornada uma tela de erro do **Spring Boot** pois o contexto / da aplicação **não** está **mapeado**;
- 🕒 Mas, o **Spring Boot** foi executado com sucesso!

# Mudando a porta do Tomcat

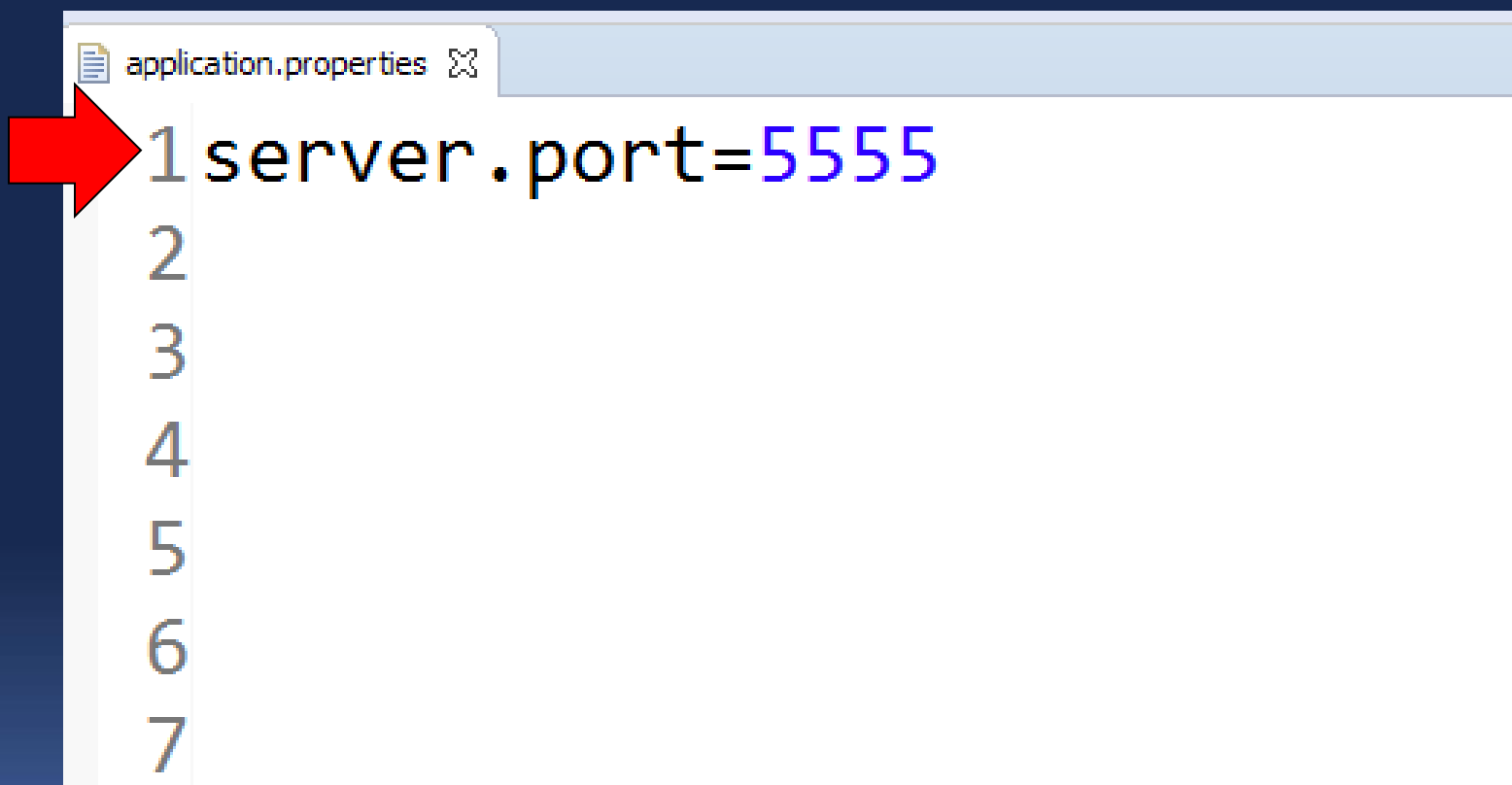
- A porta do **Tomcat** pode ser configurada no arquivo **application.properties**, na pasta **src/main/resources**.



# Mudando a porta do Tomcat



- A porta do Tomcat pode ser configurada no arquivo **application.properties**.

A screenshot of a code editor showing the 'application.properties' file. A red arrow points to the first line of the file, which contains the configuration 'server.port=5555'. The line numbers 1 through 7 are visible on the left side of the editor.

```
1 server.port=5555
2
3
4
5
6
7
```



# Start Tomcat na porta 5555

## Reiniciando a aplicação

```
: Starting ListcursosApplication on Aparecido-PC with PID 7152 (E:\In
: No active profile set, falling back to default profiles: default
: Devtools property defaults active! Set 'spring.devtools.add-propert
: For additional web related logging consider setting the 'logging.le
: Tomcat initialized with port(s): 5555 (http)
: Starting service [Tomcat]
: Starting Servlet engine: [Apache Tomcat/9.0.36]
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 1140 ms
: Initializing ExecutorService 'applicationTaskExecutor'
```

# Criando a Classe de Domínio

```
package br.com.qualitsys.model;

public class Curso {

    private Integer idCurso;
    private String nomeCurso;
    private String timestampCurso;
    private String datetimeCurso;

    public Curso(Integer idCurso, String nomeCurso) {

        this.idCurso = idCurso;
        this.nomeCurso = nomeCurso;
    }

    public Curso(        Integer idCurso,
                        String nomeCurso,
                        String timestampCurso,
                        String datetimeCurso) {

        this.idCurso = idCurso;
        this.nomeCurso = nomeCurso;
        this.timestampCurso = timestampCurso;
        this.datetimeCurso = datetimeCurso;
    }
}
```

# Classe de Domínio



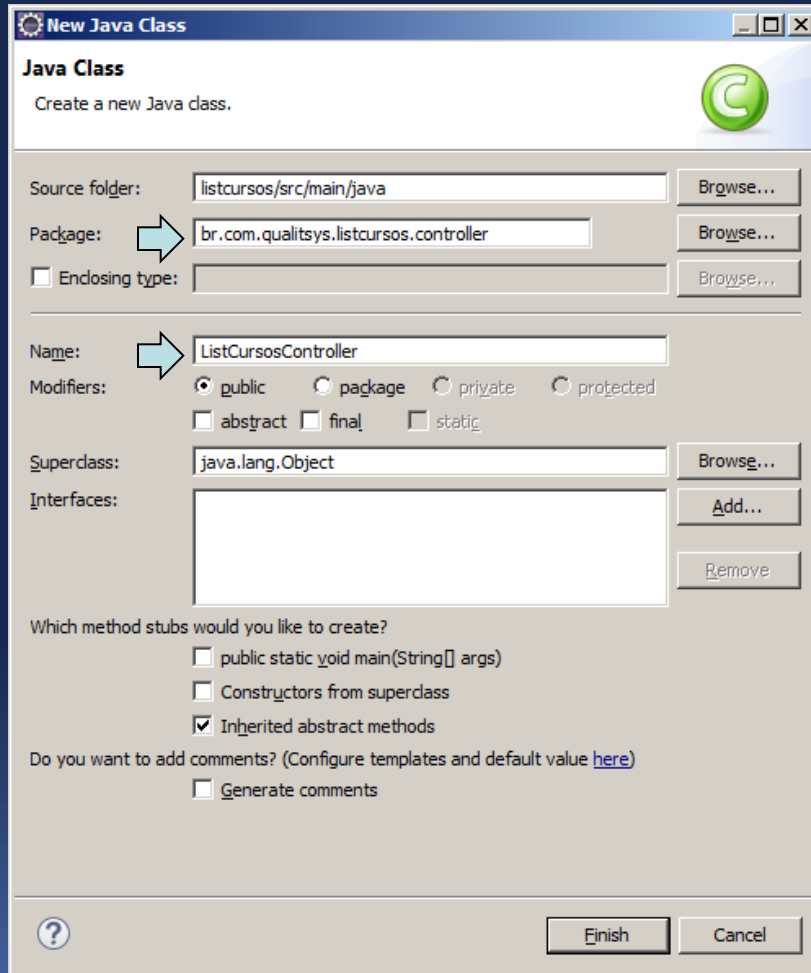
```
public Integer getIdCurso() {  
    return idCurso;  
}  
  
public void setIdCurso(Integer idCurso) {  
    this.idCurso = idCurso;  
}  
  
public String getNomeCurso() {  
    return nomeCurso;  
}  
  
public void setNomeCurso(String nomeCurso) {  
    this.nomeCurso = nomeCurso;  
}  
  
public String getTimestampCurso() {  
    return timestampCurso;  
}  
  
public void setTimestampCurso(String timestampCurso) {  
    this.timestampCurso = timestampCurso;  
}  
  
public String getDatetimeCurso() {  
    return datetimeCurso;  
}  
  
public void setDatetimeCurso(String datetimeCurso) {  
    this.datetimeCurso = datetimeCurso;  
}  
}
```





# Criando um Controller

- Criaremos um **controller** e o mapearemos para o endereço `/`;
- Verificaremos se o **Spring** faz a chamada desse controller;



**New Java Class**

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

# Criando um Controller

```
package br.com.qualitsys.listcursos.controller;  
  
public class ListCursosController {  
  
}  

```

# Criando um Controller



- O Spring Boot usa nos bastidores o **Spring MVC**;
- Para que essa classe seja considerada um controller pelo **Spring MVC** a anotaremos com a anotation **@Controller**.

```
package br.com.qualitsys.listcursos.controller;  
  
import org.springframework.stereotype.Controller;  
  
@Controller  
public class ListCursosController {  
  
}
```



# Criando um Controller

- O controller **ListCursosController** será mapeado no endpoint **/** ;
- O mapeamento será feito pela anotação **@RequestMapping**;
- Ao ser chamado deverá retornar a lista de cursos (no caso ainda simulada em **memória** - List).
- Criaremos nesse controller dois cursos em memória no método **listaCursos**.

# Criando um Controller

```
package br.com.qualitsys.listcursos.controller;
import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import br.com.qualitsys.modelCurso;

@Controller
public class ListCursosController {

    @RequestMapping("/")
    public List<Curso> listaCursos() {

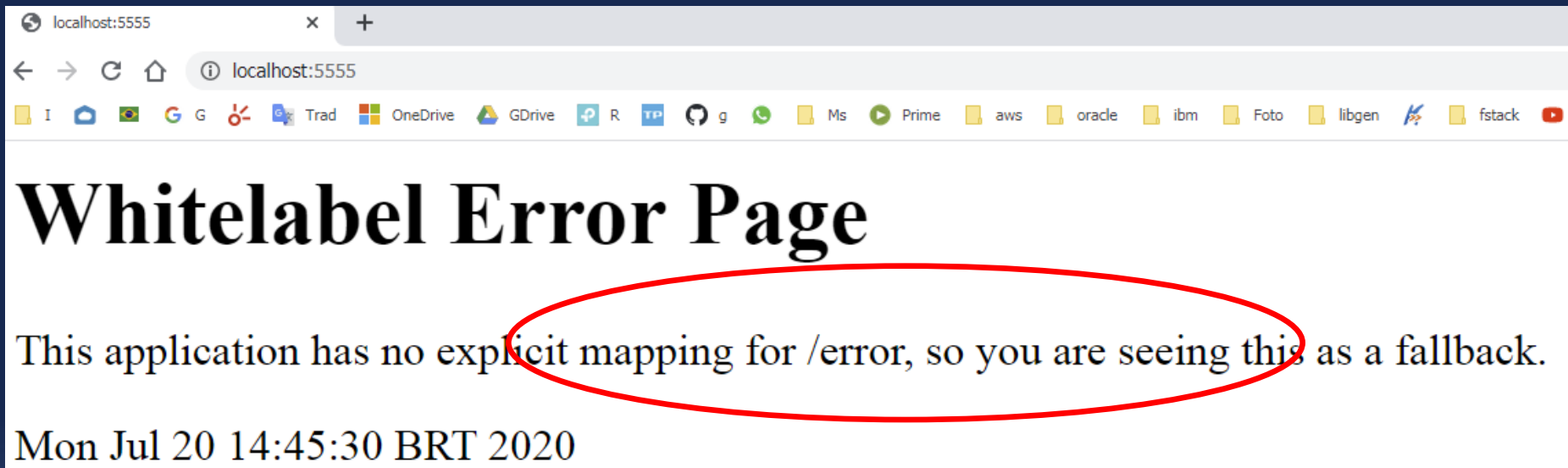
        Curso curso1 = new Curso ( 1,
                                    "Sistemas de Informação",
                                    "2020-07-17 09:20:39",
                                    "2020-07-17 09:20:39");

        Curso curso2 = new Curso ( 1,
                                    "Gestão de Tecnologia da Informação",
                                    "2020-07-17 09:21:27",
                                    "2020-07-17 09:21:27");

        return (Arrays.asList(curso1,curso2));
    }
}
```

# Executando a API no endpoint /cursos

- A aplicação retornou error type **Internal Server Error**, status = **500**
- O erro ocorreu, pois precisamos dizer ao Spring que o retorno será feito por string e **não** por uma página !



# Executando a API no endpoint /cursos

- Vamos então incluir a anotação **@ResponseBody** na classe **CursosController**

```
import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import br.com.qualitsys.modelCurso;
@Controller
public class ListCursosController {

    @RequestMapping("/")
    @ResponseBody
    public List<Curso> listaCursos() {

        Curso curso1 = new Curso ( 1,
                                    "Sistemas de Informação",
                                    "2020-07-17 09:20:39",
                                    "2020-07-17 09:20:39");

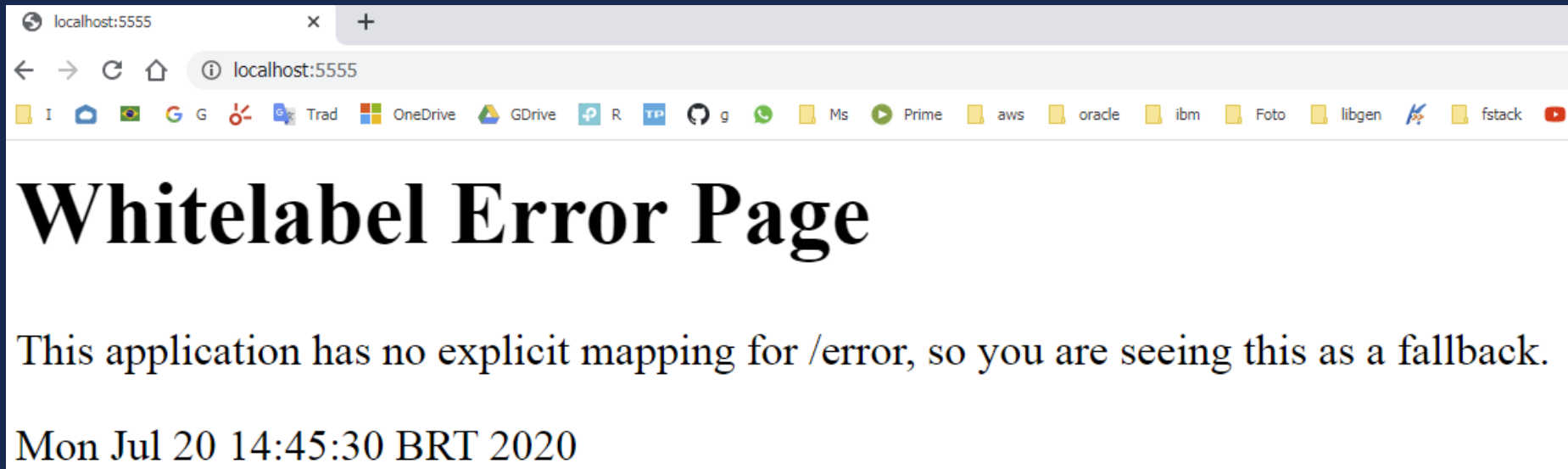
        Curso curso2 = new Curso ( 1,
                                    "Gestão de Tecnologia da Informação",
                                    "2020-07-17 09:21:27",
                                    "2020-07-17 09:21:27");

        return (Arrays.asList(curso1,curso2));
    }
}
```

# Reexecutando no endpoint /cursos



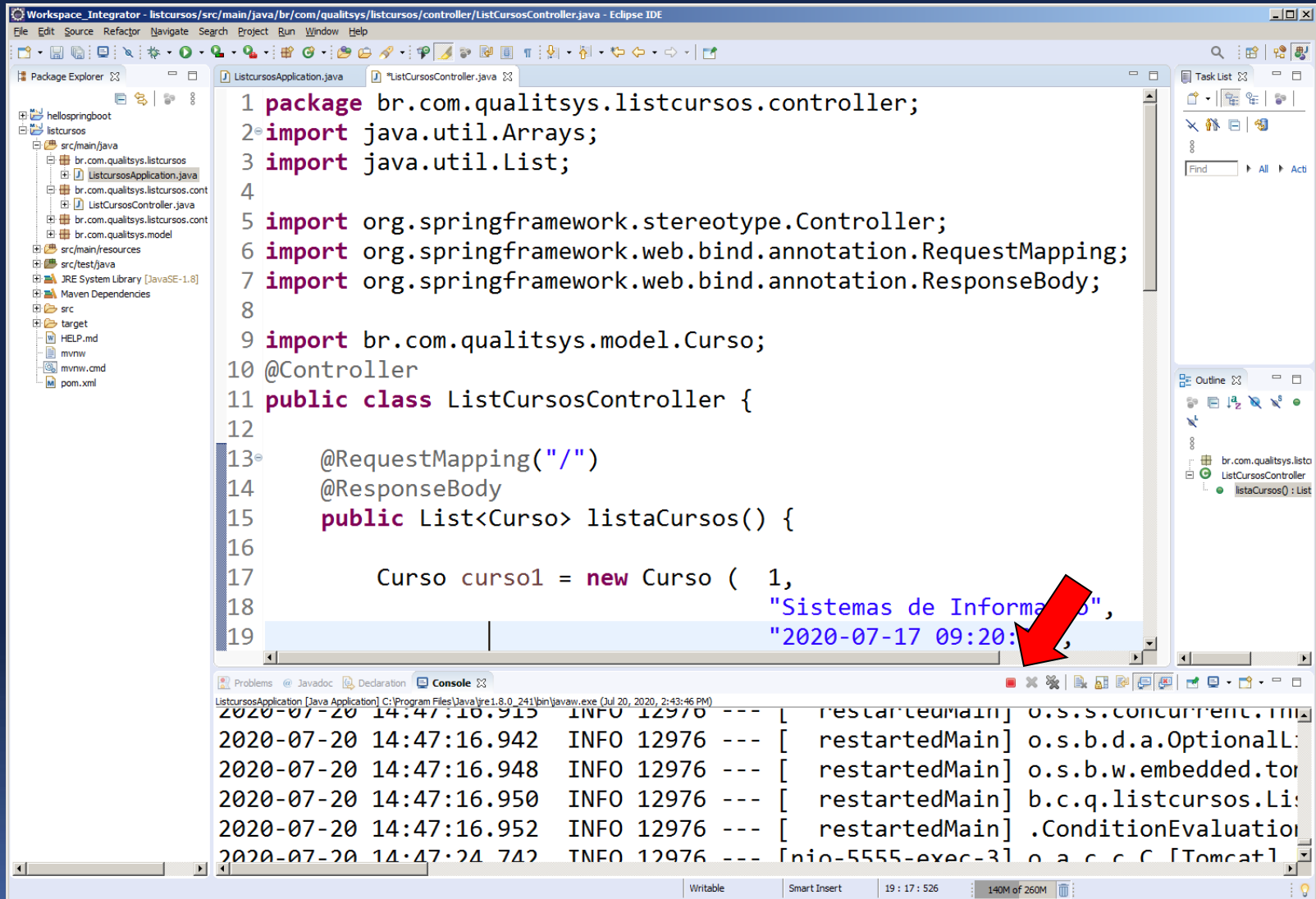
- ❌ O erro persiste, pois é necessário restart no servidor





# Stop no servidor

● Clicar no botão **vermelho** na console do Eclipse



The screenshot shows the Eclipse IDE workspace. The Package Explorer on the left shows the project structure. The main editor displays the `ListCursosController.java` file. The console at the bottom shows the application's output, including a red stop button icon.

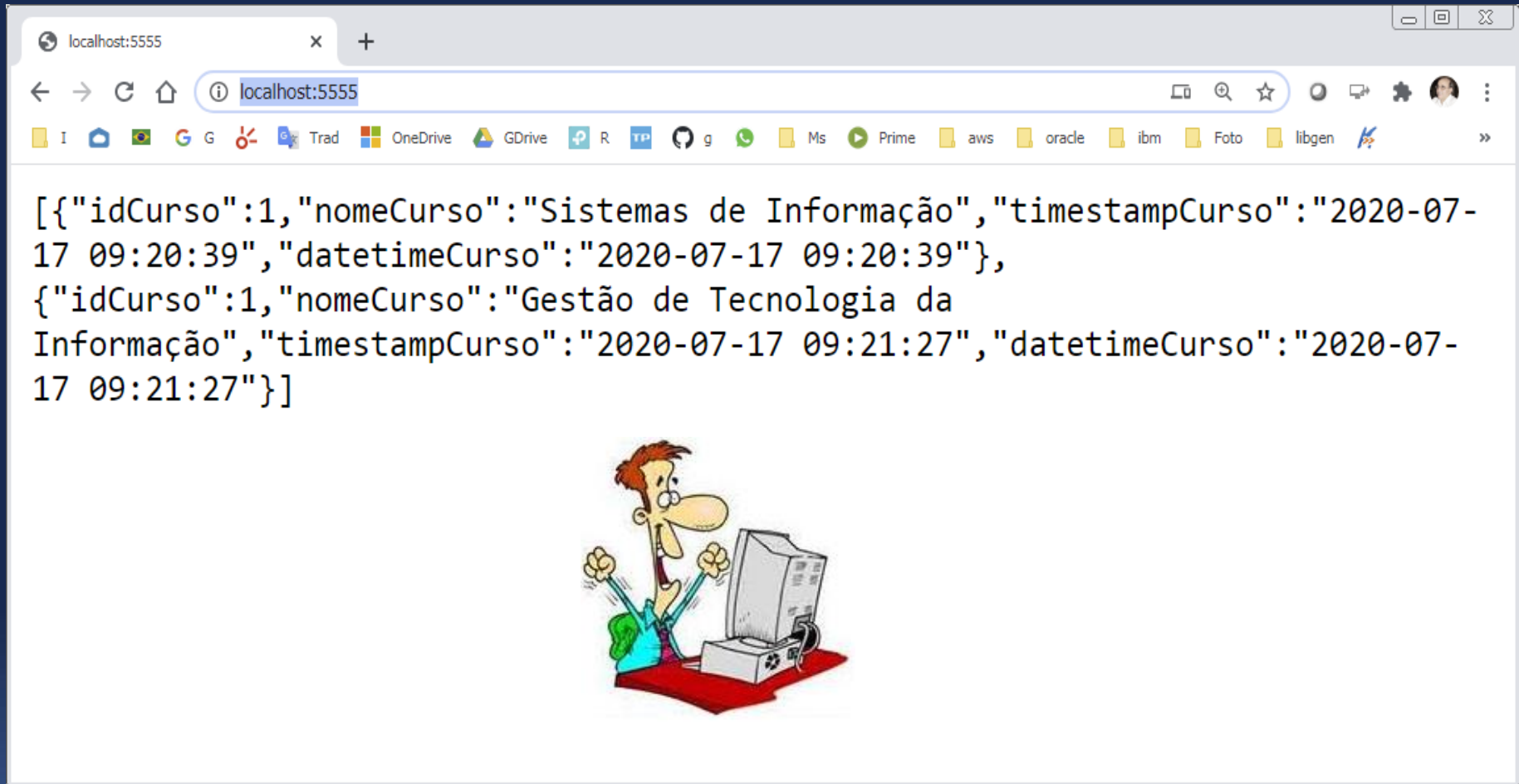
```

1 package br.com.qualitsys.listcursos.controller;
2 import java.util.Arrays;
3 import java.util.List;
4
5 import org.springframework.stereotype.Controller;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.ResponseBody;
8
9 import br.com.qualitsys.model.Curso;
10 @Controller
11 public class ListCursosController {
12
13     @RequestMapping("/")
14     @ResponseBody
15     public List<Curso> listaCursos() {
16
17         Curso curso1 = new Curso ( 1,
18                                     "Sistemas de Informa",
19                                     "2020-07-17 09:20:
  
```

The console output shows the application's startup logs, including the message "RestartedMain" and the application's version information.



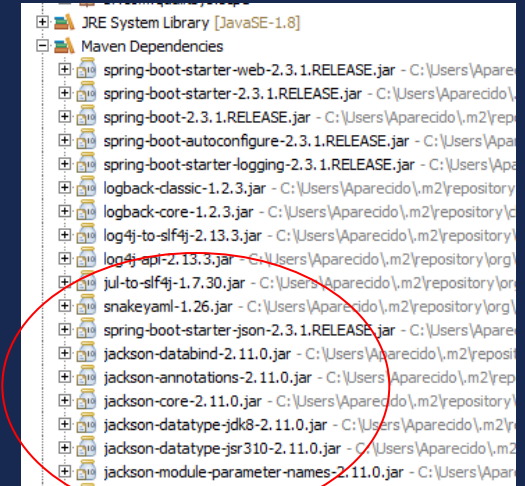
# Reexecutando a aplicação



# Observações



- O **Spring** automaticamente converteu a lista de cursos para o formato Json;
- Na verdade não foi o **Spring** que fez isso;
- O **Spring** usa uma biblioteca chamada Jackson;
- É o Jackson que faz a conversão de Java para Json;
- O **Spring** usa o **Jackson** de forma automática sem que seja necessária a escrita de código pelo programador;
- **Spring** então passou a lista para o **Jackson** que converteu para Json e em seguida foi devolvido um string;



# Algumas melhorias no Projeto

# Algumas melhorias no projeto

- Nos métodos codificados como controller tivemos que incluir a anotação **@ResponseBody** ;
- Sem essa anotação o **Spring**, por padrão, considera que iremos fazer uma navegação para alguma página;
- Mas no nosso caso, como estamos desenvolvendo uma **API Rest**, não temos navegação para páginas, ou seja, iremos devolver de fato o que estiver sendo retornado pelo método.

```
package br.com.qualitsys.listcursos.controller;
import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import br.com.qualitsys.modelCurso;
@Controller
public class ListCursosController {

    @RequestMapping("/")
    @ResponseBody
    public List<Curso> listaCursos() {

        Curso curso1 = new Curso ( 1,
                                    "Sistemas de Informação",
                                    "2020-07-17 09:20:39",
                                    "2020-07-17 09:20:39");

        Curso curso2 = new Curso ( 1,
                                    "Gestão de Tecnologia da Informação",
                                    "2020-07-17 09:21:27",
                                    "2020-07-17 09:21:27");

        return (Arrays.asList(curso1,curso2));
    }
}
```

# Anotação @RestController

- Para evitar a escrita dessa anotação em todos os métodos, no Spring Boot pode-se usar a anotação **@RestController** em cima da classe para definir a classe como um Rest Controller;
- Com isso, por padrão, o Spring Boot assume que todo método terá (embutido) um **@ResponseBody**;
- Ou seja, se usarmos **@RestController** não se precisa mais ficar colocando o **@ResponseBody** em cima dos métodos.

# Reescrevendo a classe ListCursosController com a anotação @RestController

```
package br.com.qualitsys.listcursos.controller;
import java.util.Arrays;

import java.util.List;

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

import br.com.qualitsys.modelCurso;
@RestController
public class ListCursosController {

    @RequestMapping("/listcursos")
    public List<Curso> listaCursos() {

        Curso curso1 = new Curso ( 1,
                                    "Sistemas de Informação",
                                    "2020-07-17 09:20:39",
                                    "2020-07-17 09:20:39");

        Curso curso2 = new Curso ( 1,
                                    "Gestão de Tecnologia da Informação",
                                    "2020-07-17 09:21:27",
                                    "2020-07-17 09:21:27");

        return (Arrays.asList(curso1,curso2));
    }
}
```



## Restart do servidor

- No desenvolvimento feito até o momento, toda vez que fizermos alguma alteração no nosso código, precisamos **restartar** o servidor para efetivar as alterações;
- Para se evitar esse constante restarte do servidor, pode-se usar um módulo do Spring Boot chamado **DevTools**;
- Se usarmos esse módulo, **não** será mais necessário restartar-se o servidor toda vez que procedermos alguma alteração no código.
- Para isso, precisamos incluir no **pom.xml** essa nova dependência do **DevTools**.

# Incluindo DevTools no pom.xml



```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-devtools</artifactId>  
  <scope>runtime</scope>  
</dependency>
```



# Arquivo pom.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>br.com.qualitsys</groupId>
  <artifactId>scpe</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>scpe</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>
</project>
```



# Arquivo pom.xml



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```



# Servidor restartado automaticamente

- Agora, com **Devtools** incorporado ao projeto, toda vez que se salvar alguma alteração no código, o servidor é automaticamente restartado.

```

22         "2020-07-17 09:21:27");
23
24     return (Arrays.asList(curso1,curso2));
25 }
26 }
27
28
29
30
31

```

Problems Javadoc Declaration Console

ScpeApplication [Java Application] C:\Program Files\Java\jre1.8.0\_241\bin\javaw.exe (Jul 17, 2020, 7:12:19 PM)

2020-07-17 19:13:14.746	INFO 17752	---	[ restartedMain]	br.com.qualitsys.scpe.ScpeApplication
2020-07-17 19:13:14.926	INFO 17752	---	[ restartedMain]	o.s.b.w.embedded.tomcat.TomcatWebServer
2020-07-17 19:13:14.927	INFO 17752	---	[ restartedMain]	o.apache.catalina.core.StandardService
2020-07-17 19:13:14.927	INFO 17752	---	[ restartedMain]	org.apache.catalina.core.StandardEngine
2020-07-17 19:13:14.935	INFO 17752	---	[ restartedMain]	o.a.c.c.C.[Tomcat].[localhost].[/]
2020-07-17 19:13:14.935	INFO 17752	---	[ restartedMain]	w.s.c.ServletWebServerApplicationContext
2020-07-17 19:13:14.964	INFO 17752	---	[ restartedMain]	o.s.s.concurrent.ThreadPoolTaskExecutor
2020-07-17 19:13:14.991	INFO 17752	---	[ restartedMain]	o.s.b.d.a.OptionalLiveReloadServer
2020-07-17 19:13:14.997	INFO 17752	---	[ restartedMain]	o.s.b.w.embedded.tomcat.TomcatWebServer
2020-07-17 19:13:15.000	INFO 17752	---	[ restartedMain]	br.com.qualitsys.scpe.ScpeApplication

## Mais uma observação

- No endpoint desenvolvido, o nosso controller retornou a lista completa do cursos;
- Porém, curso é uma **classe de domínio** da nossa aplicação;
- Na próxima unidade, iremos utilizar **JPA** e, assim, essa classe ficará sendo uma entidade do **JPA**, uma vez que haverá uma tabela associada a essa classe no banco de dados;
- Não é uma boa prática devolver entidades da **JPA** no nosso controller, uma vez que geralmente há nessa classe muitos atributos que podem ser outras entidades, outras classes que têm outros atributos.
- O **Jackson**, por padrão, serializa todos os atributos que estiverem dentro da classe e isso pode não ser conveniente.
- Outra razão, um dos atributos da classe de domínio, pode ser uma **senha** a qual usualmente **não** deve ser retornada.

## Mais uma observação

- Nossa classe de domínio `Curso` tem os seguintes atributos: `idCurso`, `nomeCurso`, `timestampCurso` e `datetimeCurso`.
- Ao invés de se devolver todos os atributos, conforme feito até o momento, vamos modificar o projeto para retornarmos nesse endpoint apenas **`idCurso`** e **`nomeCurso`**;
- Geralmente, utiliza-se o padrão **`DTO - Data Transfer Object`** para esse tipo de classe;
- Assim, vamos implementar uma nova classe chamada **`CursoDto`** com os atributos a serem retornados;
- Com essa providência teremos maior flexibilidade no desenvolvimento dos nossos endpoints.

# Classe CursoDto



- Criaremos essa classe num package que será criado dentro do package controller;
- Chamaremos esse package de [br.com.qualitsys.controller.dto](http://br.com.qualitsys.controller.dto)

```
package br.com.qualitsys.listcursos.controller.dto;

import br.com.qualitsys.modelCurso;

public class CursoDto {

    private Integer idCurso;
    private String nomeCurso;

    public CursoDto(Integer idCurso, String nomeCurso) {

        this.idCurso = idCurso;
        this.nomeCurso = nomeCurso;
    }
    public CursoDto(Curso curso) {

        this.idCurso = curso.getIdCurso();
        this.nomeCurso = curso.getNomeCurso();
    }
    public Integer getIdCurso() {
        return idCurso;
    }
    public void setIdCurso(Integer idCurso) {
        this.idCurso = idCurso;
    }
    public String getNomeCurso() {
        return nomeCurso;
    }
    public void setNomeCurso(String nomeCurso) {
        this.nomeCurso = nomeCurso;
    }
}
```





# Classe ListCursosController



```
package br.com.qualitsys.listcursos.controller;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

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

import br.com.qualitsys.listcursos.controller.dtoCursoDto;
import br.com.qualitsys.model.Curso;
@RestController
public class ListCursosController {

    @RequestMapping("/listcursos")
    public List<CursoDto> listaCursos() {

        Curso curso1 = new Curso ( 1,
                                    "Sistemas de Informação",
                                    "2020-07-17 09:20:39",
                                    "2020-07-17 09:20:39");

        Curso curso2 = new Curso ( 1,
                                    "Gestão de Tecnologia da Informação",
                                    "2020-07-17 09:21:27",
                                    "2020-07-17 09:21:27");

        return converter(Arrays.asList(curso1,curso2));
    }
}
```



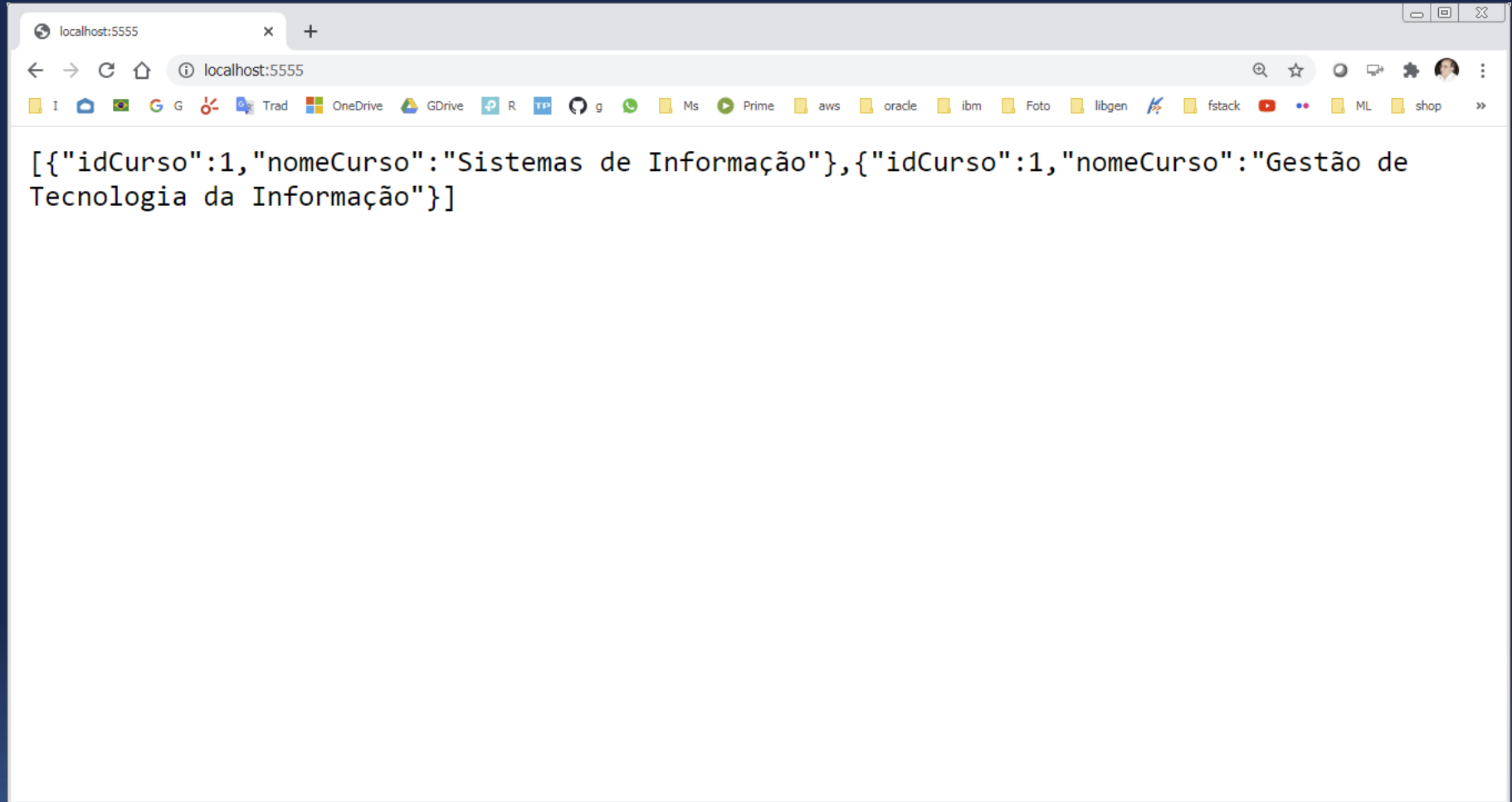
# Classe ListCursosController

```
//metodo que recebe lista de cursos e retorna lista de cursoDto
public static List<CursoDto> converter(List<Curso> listaCursos ) {

    List<CursoDto> listaCursosDto = new ArrayList<CursoDto>();
    int n = listaCursos.size();

    for (int i = 0; i < n; i++) {
        CursoDto c = new CursoDto(listaCursos.get(i).getIdCurso(), listaCursos.get(i).getNomeCurso());
        listaCursosDto.add(c);
    }
    return listaCursosDto;
}
```

# Executando a aplicação



Gerando nova versão do arquivo .war


# Criando .war da aplicação Spring Boot

- Modificar o arquivo pom.xml definindo empacotamento **war**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>br.com.qualitsys</groupId>
  <artifactId>listcursos</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>listcursos</name>
  <description>Projeto Rest com Spring Boot - listagem de cursos em Memória</
```



# Criando .war da aplicação Spring Boot

- 
- Vamos manter o **Tomcat** configurado para o ambiente de desenvolvimento, mas não o utilizaremos no empacotamento final (**war**);
  - Para isso, vamos incluir **nova** dependência no arquivo **pom.xml**.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-tomcat</artifactId>  
  <scope>provided</scope>  
</dependency>
```

# Criando .war da aplicação Spring Boot

- Arquivo pom.xml modificado

```

        <artifactId>junit-vintage-engine</artifactId>
    </exclusion>
</exclusions>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>

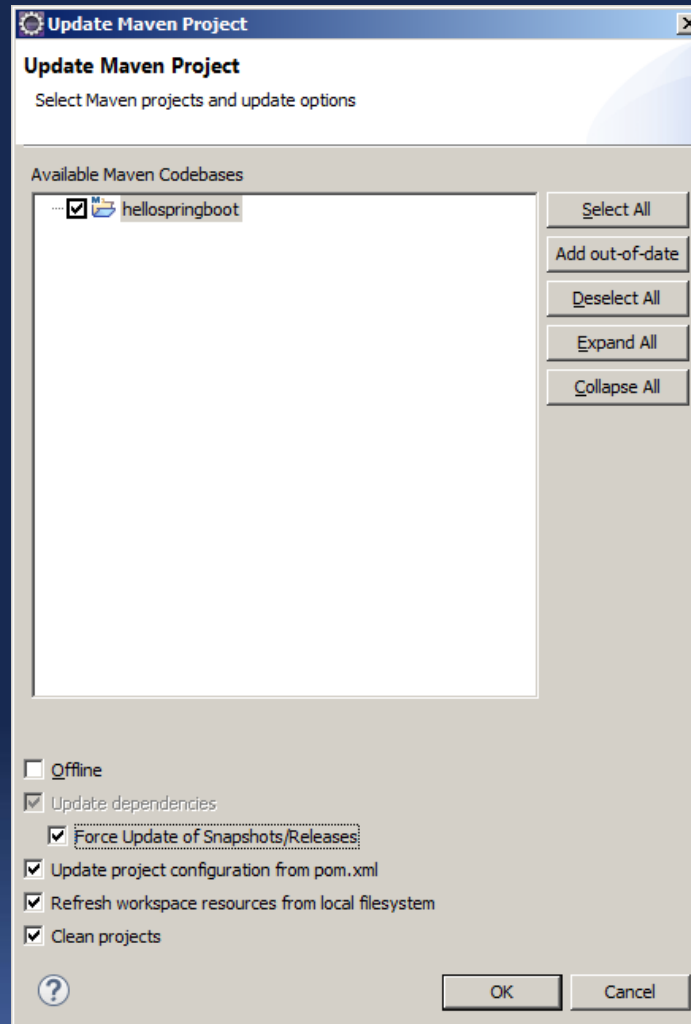
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>

```

# Atualizando o projeto

- Botão direito no projeto
- Maven > Update Project





# Alterando a classe principal



- Modificaremos a classe principal da aplicação (onde está o método **main**) para que a aplicação possa operar com um servidor de aplicações Tomcat externo;
- Isso é feito definindo-se a classe principal como filha da classe **SpringBootServletInitializer**;

```
package br.com.qualitsys.listcursos;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class ListcursosApplication extends SpringBootServletInitializer{

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

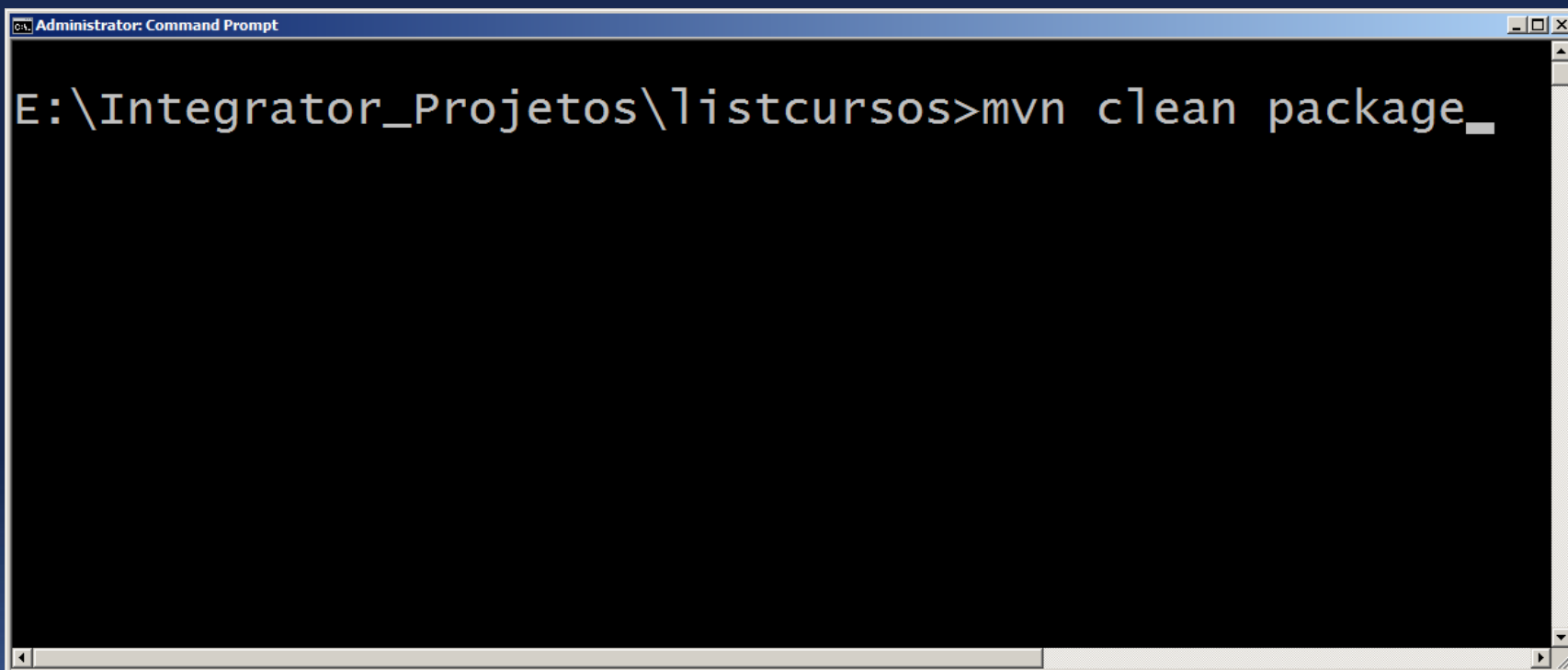
}
```



# Gerando o arquivo war

Na pasta da aplicação, executar o comando:

 `$ mvn clean package`



```
Administrator: Command Prompt
E:\Integrator_Projetos\listcursos>mvn clean package_
```

# Gerando o arquivo war

```
Administrator: Command Prompt
NAPSHOT.war
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.1.RELEASE:repackage
[INFO] ---
[INFO] Replacing main artifact with repackaged archive
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.066 s
[INFO] Finished at: 2020-07-20T12:31:39-03:00
[INFO] -----
E:\Integrator_Projetos\listcursos>
```

# arquivo war gerado na pasta target

```
Administrator: Command Prompt

Directory of E:\Integrator_Projetos\listcursos\target

20-Jul-20 12:31 PM <DIR> .
20-Jul-20 12:31 PM <DIR> ..
20-Jul-20 12:31 PM <DIR> classes
20-Jul-20 12:31 PM <DIR> generated-sources
20-Jul-20 12:31 PM <DIR> generated-test-sources
20-Jul-20 12:31 PM <DIR> listcursos-0.0.1-SNAPSHOT
20-Jul-20 12:31 PM      16,496,308 listcursos-0.0.1-SNAPSHOT.war
20-Jul-20 12:31 PM      14,670,319 listcursos-0.0.1-SNAPSHOT.war.original
20-Jul-20 12:31 PM <DIR> maven-archiver
20-Jul-20 12:31 PM <DIR> maven-status
20-Jul-20 12:31 PM <DIR> surefire-reports
20-Jul-20 12:31 PM <DIR> test-classes
                2 File(s)      31,166,627 bytes
               10 Dir(s)  350,866,759,680 bytes free


E:\Integrator_Projetos\listcursos\target>
```

# Renomeando arquivo war para listcursos.war

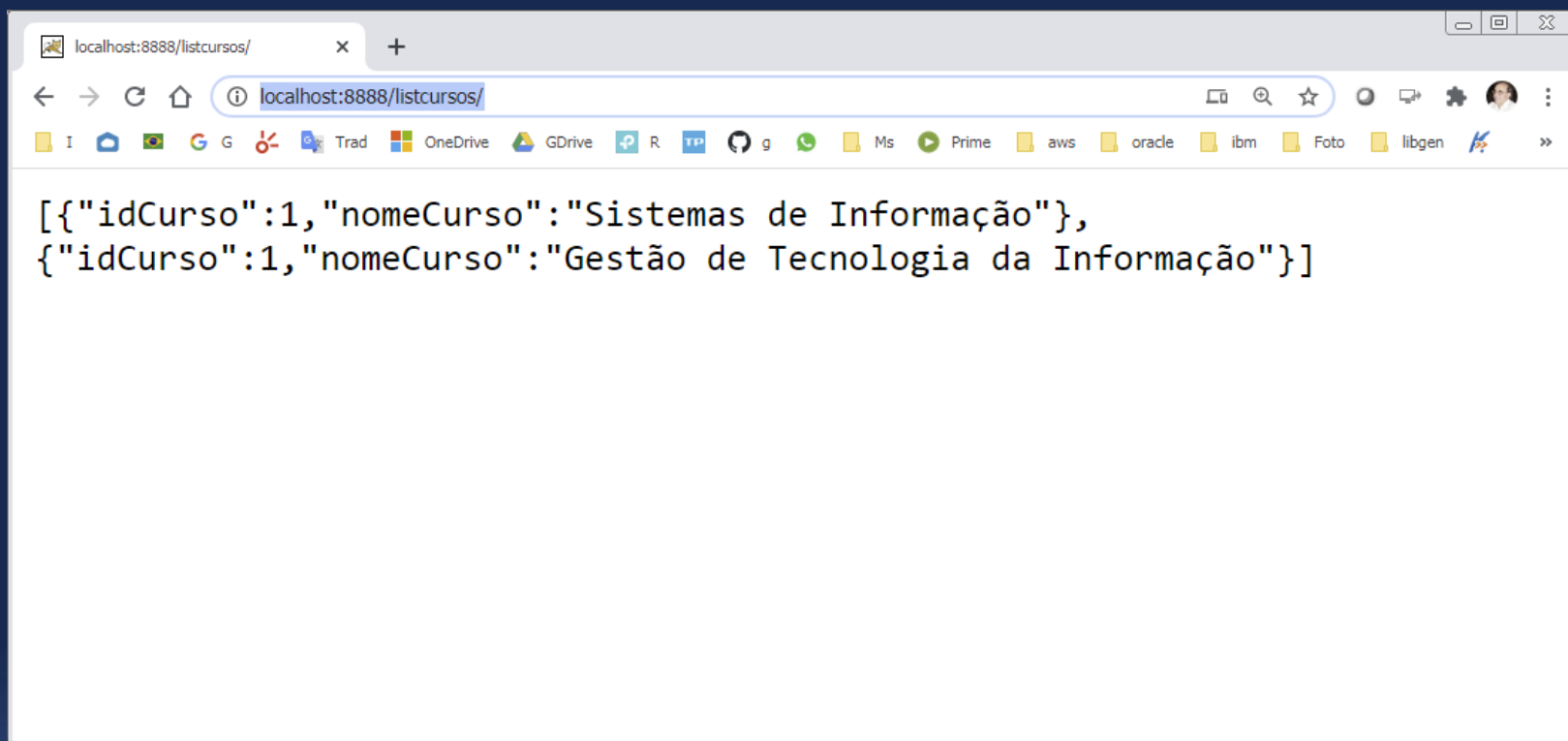
```
Administrator: Command Prompt

Directory of E:\Integrator_Projetos\listcursos\target
20-Jul-20  12:31 PM                16,496,308 listcursos.war
             1 File(s)            16,496,308 bytes
             0 Dir(s)  350,866,759,680 bytes free

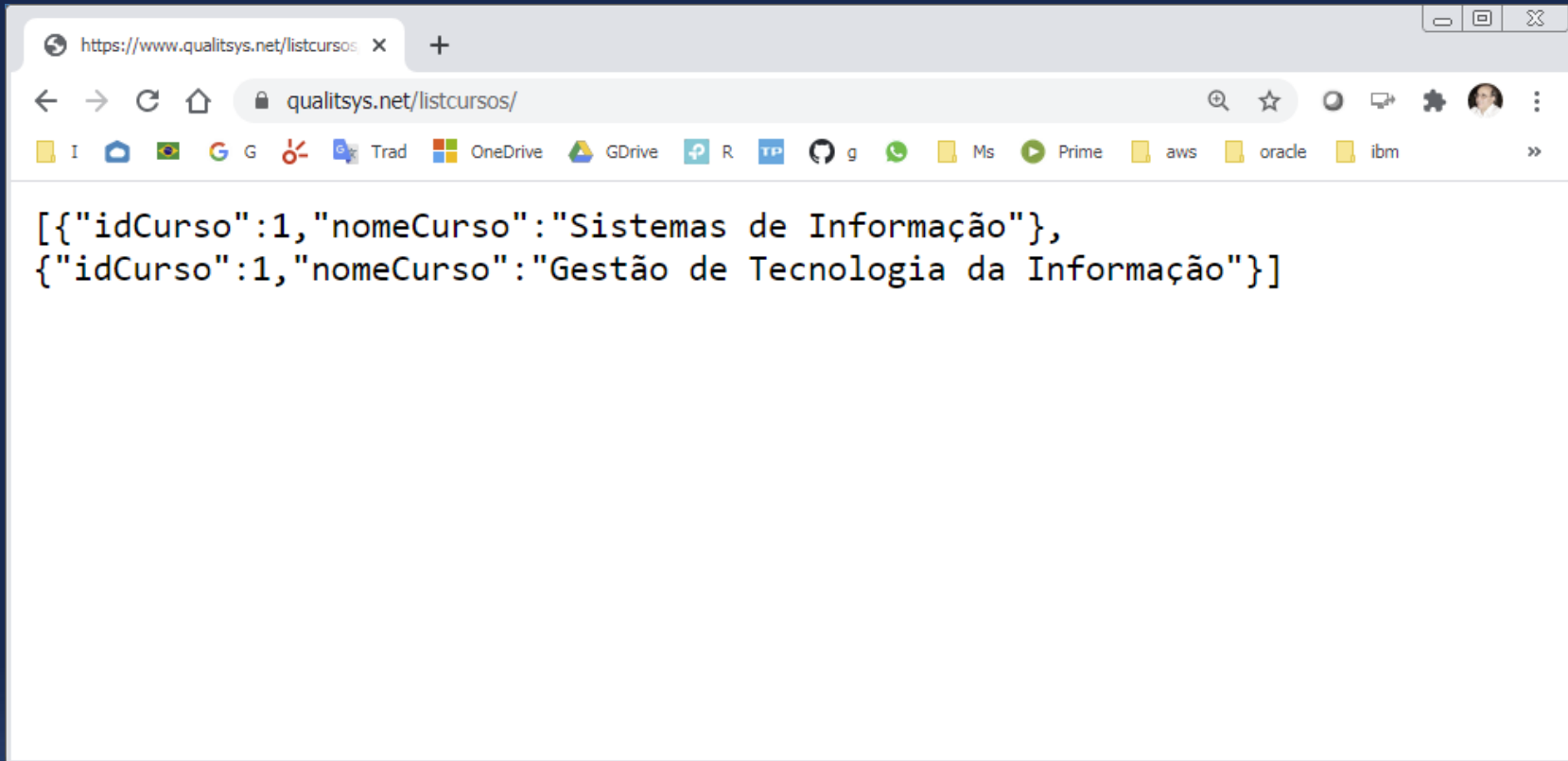
E:\Integrator_Projetos\listcursos\target>
```



# Portando num Tomcat externo



# Portando na nuvem – integrator.com.br



```
[{"idCurso":1,"nomeCurso":"Sistemas de Informação"},
{"idCurso":1,"nomeCurso":"Gestão de Tecnologia da Informação"}]
```

# Salvando projeto no Repositório git local

```
Command Prompt
E:\Integrator_Repository>dir
Volume in drive E is Dados
Volume Serial Number is 14D9-25F0

Directory of E:\Integrator_Repository

20-Jul-20   03:25 PM    <DIR>          .
20-Jul-20   03:25 PM    <DIR>          ..
19-Jul-20   06:44 PM    <DIR>          hellospringboot
20-Jul-20   03:25 PM    <DIR>          listcursos
               0 File(s)                0 bytes
               4 Dir(s)  350,771,617,792 bytes free

E:\Integrator_Repository>
```



# Enviando projeto para github



```
Command Prompt
E:\Integrator_Repository>git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 755 bytes | 58.00 KiB/s, done.
From https://github.com/avfreitas/Integrator_Repository
  6191a6a..f1d20bd  master    -> origin/master
Merge made by the 'recursive' strategy.
 README.md | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 README.md

E:\Integrator_Repository>
E:\Integrator_Repository>
E:\Integrator_Repository>git push
Enumerating objects: 39, done.
Counting objects: 100% (39/39), done.
Delta compression using up to 4 threads
Compressing objects: 100% (25/25), done.
Writing objects: 100% (37/37), 54.33 KiB | 6.04 MiB/s, done.
Total 37 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/avfreitas/Integrator_Repository.git
  f1d20bd..57e504e  master -> master

E:\Integrator_Repository>
```



# Consultando github



avfreitas/Integrator\_Repository

github.com/avfreitas/Integrator\_Repository

Search or jump to... Pull requests Issues Marketplace Explore

avfreitas / Integrator\_Repository

Watch 0 Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 0 tags

Go to file Add file Code

avfreitas Merge branch 'master' of https://github.com/... 57e504e 5 minutes ago 4 commits

hellospringboot	v1.0.0 - 19/07/2020	21 hours ago
<b>listcursos</b>	listcursos - v1.0.0 - api s/JPA	17 minutes ago
README.md	Create README.md	19 hours ago

README.md

## Integrator\_Repository

Hellospringboot - projeto de configuração (setup básico) do Spring Boot

About

No description, website, or topics provided.

Readme

Releases

No releases published  
[Create a new release](#)

Packages

No packages published  
[Publish your first package](#)

Languages



# Validando repositório git

```
Command Prompt

E:\Integrator_Repository>git pull
Already up to date.

E:\Integrator_Repository>git push
Everything up-to-date

E:\Integrator_Repository>_
```