

*Instituto Federal de Educação, Ciência e Tecnologia do Paraná - IFPR*

*disciplina de*

## **DESENVOLVIMENTO WEB III**

# **SERVLETS**

*Professor Jefferson de Oliveira Chaves  
jefferson.chaves@ifpr.edu.br*

# SERVLETS

**UMA SERVLET É UMA CLASSE JAVA,  
CUJO OBJETIVO É RECEBER  
REQUISIÇÕES HTTP, PROCESSÁ-LAS E  
RESPONDER AO CLIENTE;**

# SERVLETS



**POST**

**GET**



**MP3**

**HTML**

**IMG**

## ANATOMIA DE UMA SERVLET

- ∴ Todo Servlet é sub-classe de **HttpServlet**;
- ∴ Diferente do que ocorre com uma aplicação Java desktop, aplicações Java Web **não possui o método main( );**
- ∴ Deve implementar um dos seguintes métodos:  
**service( ), doGet( ) e doPost( ) e etc...**

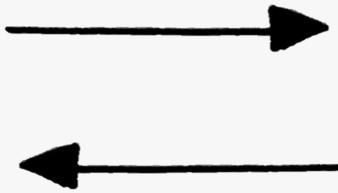
# OUTROS MÉTODOS DA SERVLET

- ∴ **service(...);**
- ∴ **doGet(...)** → oferece suporte aos métodos GET do protocolo HTTP;
- ∴ **doPost(...)** → oferece suporte aos métodos POST do HTTP;
- ∴ **doPut(...)** → oferece suporte aos métodos PUT do HTTP;
- ∴ **doDelete(...)** → oferece suporte aos métodos DELETE do HTTP;
- ∴ **init(...)** e **destroy(...)** → gerenciar o ciclo de vida de um servlet;

# SERVLETS: MÉTODO SERVICE

- :: Repare que o método recebe dois objetos que representam, respectivamente, a requisição feita pelo usuário e a resposta que será exibida ao final;
- :: Podemos usar esses objetos para obter informações sobre a requisição e para construir a resposta final para o usuário.



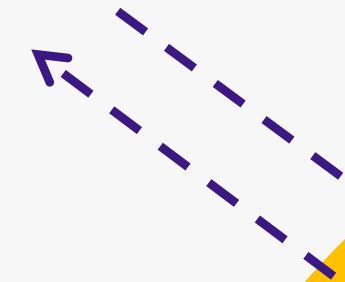


# SERVLETS

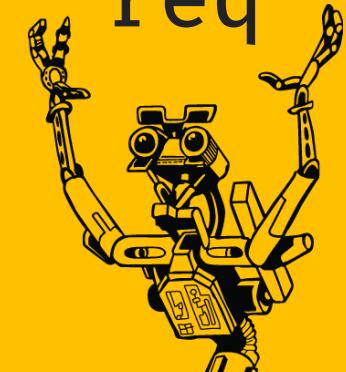
GET  
POST



HttpServletRequest



req



# ANATOMIA DE UMA SOLICITAÇÃO HTTP

O Método HTTP.

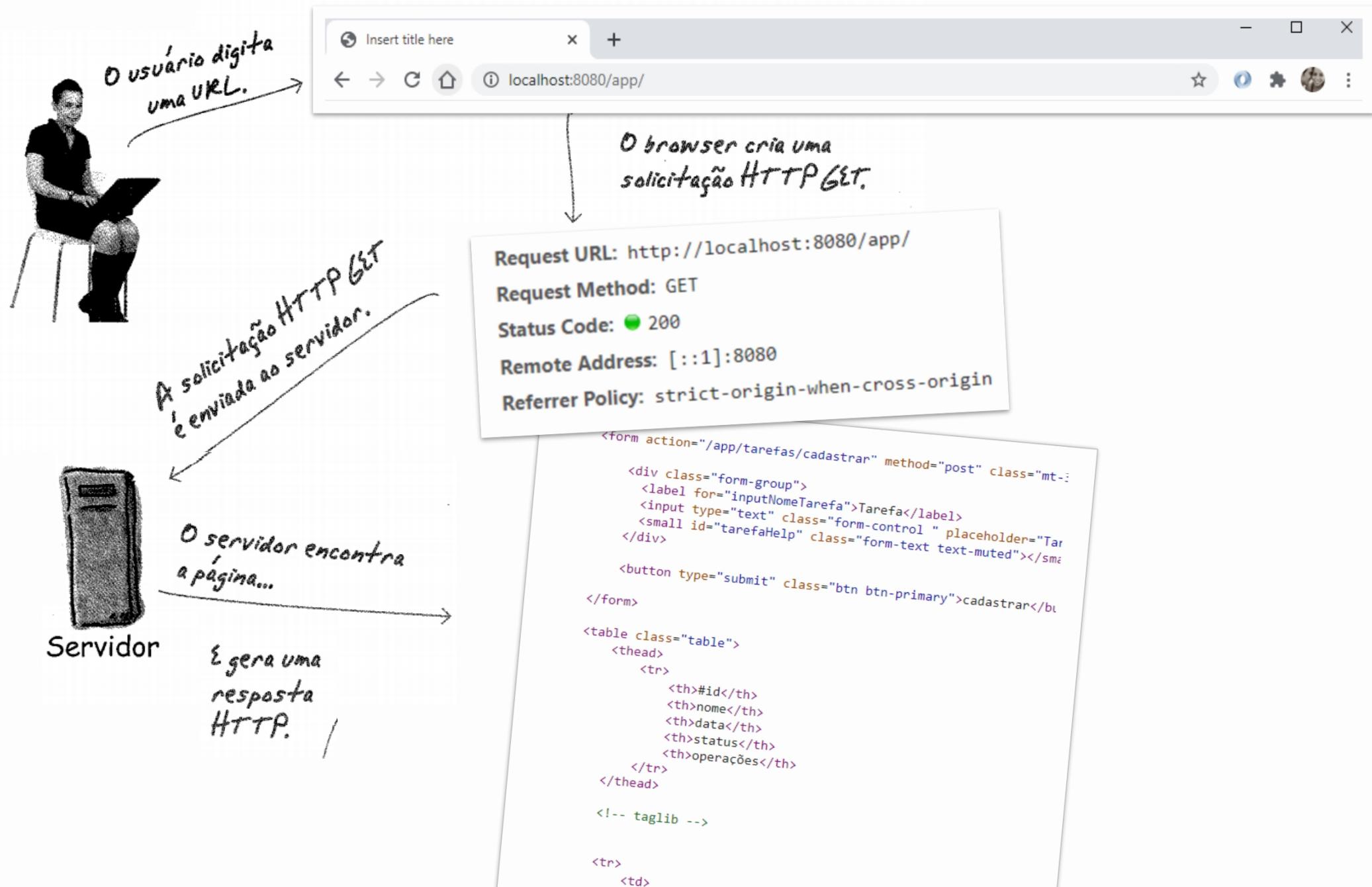
A linha de Solicitação.

O caminho para o recurso no servidor.

A versão do protocolo que o browser está solicitando.

Os headers da Solicitação.

```
GET /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;
rv:1.4) Gecko/20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/
gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```



# MAPEANDO UMA SERVLET

∴ Acabamos de definir uma Servlet:

∴ Como vamos acessá-la pelo navegador?

∴ Qual o endereço para acesso?

∴ Para que seja possível acessar um servlet é necessário mapeá-lo;

∴ Para isso, é possível fazer um mapeamento de uma URL específica para uma servlet através do arquivo `web.xml`, que fica dentro do WEB-INF ou por meio de uma `@annotation` na servlet;

## MAPEANDO UMA SERVLET (USANDO O DEPLOYMENT DESCRIPTOR)

1

```
<servlet>
    < servlet-name>primeiraServlet</servlet-name>
    < servlet-class>servlets.MinhaPrimeiraServlet</servlet-class>
</servlet>
```

2

```
<servlet-mapping>
    < servlet-name>primeiraServlet</servlet-name>
    < url-pattern>/inicio</url-pattern>
</servlet-mapping>
```

# SERVLETS

```
package br.edu.ifpr.servlets;

import javax.servlet.http.HttpServlet;

@WebServlet("/endereco_logico_para_servlet")
public class MinhaServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) {
        //....
    }
}
```

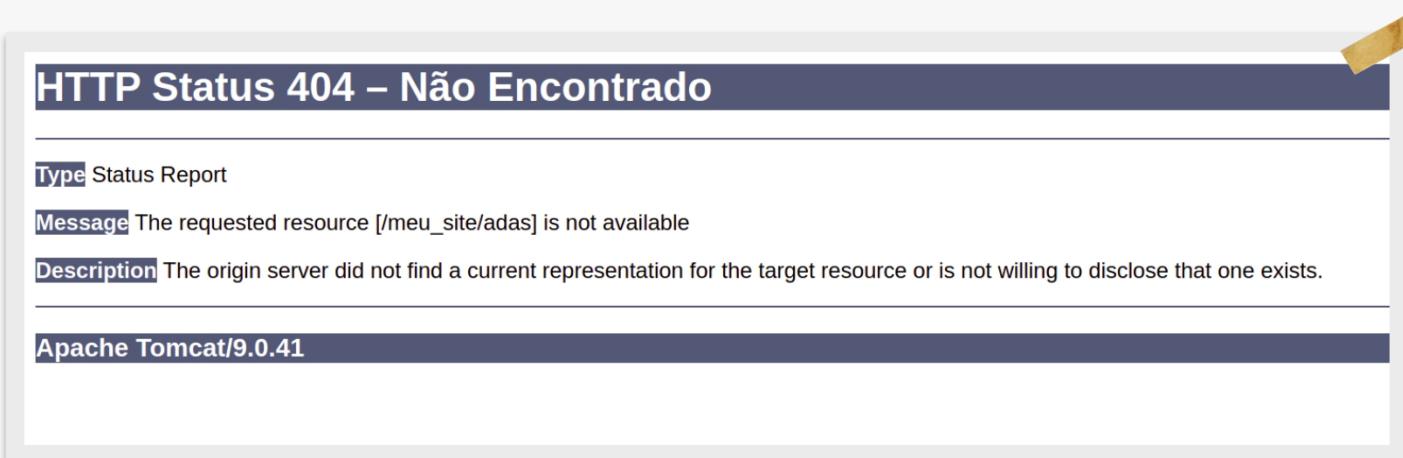
## MAPEANDO UMA SERVLET: ERROS COMUNS

- Esquecer da barra inicial no URL pattern:

```
<url-pattern>oi</url-pattern>
```

- Digitar errado o nome do pacote da sua servlet:

```
<servlet-class>servlets.Classe</servlet-class>
```



# MAPEANDO UMA SERVLET

∴ @WebServlet(name = "userServlet")

```
<servlet-mapping>
  <servlet-name>userServlet</servlet-name>
  <url-pattern>/user</url-pattern>
</servlet-mapping>
```

∴ @WebServlet(name = "userServlet", urlPatterns = {"", "/users/\*"})

## MAIS SOBRE O DEPLOYMENT DESCRIPTOR

```
<web-app>

    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>test.HelloWorld</servlet-class>
        <init-param>
            <param-name>title</param-name>
            <param-value>Hello, World</param-value>
        </init-param>
    </servlet>

    <servlet servlet-name='cron' servlet-class='test.DailyChores'>
        <init-param title='Daily Chores' />
        <load-on-startup/>
        <run-at>3:00</run-at>
    </servlet>

    <!-- mapping a url to use the servlet -->
    <servlet-mapping url-pattern='/hello.html' servlet-name='hello' />

</web-app>
```

## MAIS SOBRE O DEPLOYMENT DESCRIPTOR

```
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/erro.html</location>
</error-page>

<error-page>
    <error-code>404</error-code>
    <location>/404.html</location>
</error-page>
```

# CICLO DE VIDA

*Ciclo de vida de uma Servlet*

# CICLO DE VIDA

- ∴ Os servlets possuem um **ciclo de vida bem definido** que é gerenciado pelo container web;
- ∴ Este ciclo é dividido em 05 etapas:
  1. Carregar em memória o servlet especificado;
  2. Criar sua instância;
  3. Invocar o método **init( )** do servlet;
  4. Invocar o método **service( )** (ou outro método que responda ao HTTP);
  5. Invocar o método **destroy( )** do servlet.

# 1

## método init()

- *O container web chama esse método depois que a instância do servlet for criada;*
- *Esse método é usado para carregar códigos que são executados apenas uma vez;*
- *Um exemplo são cases que são necessários para carregar driver de banco de dados, ou efetuar alguma inicialização de valores.*

```
@Override  
public void init() throws ServletException {  
    super.init();  
}
```

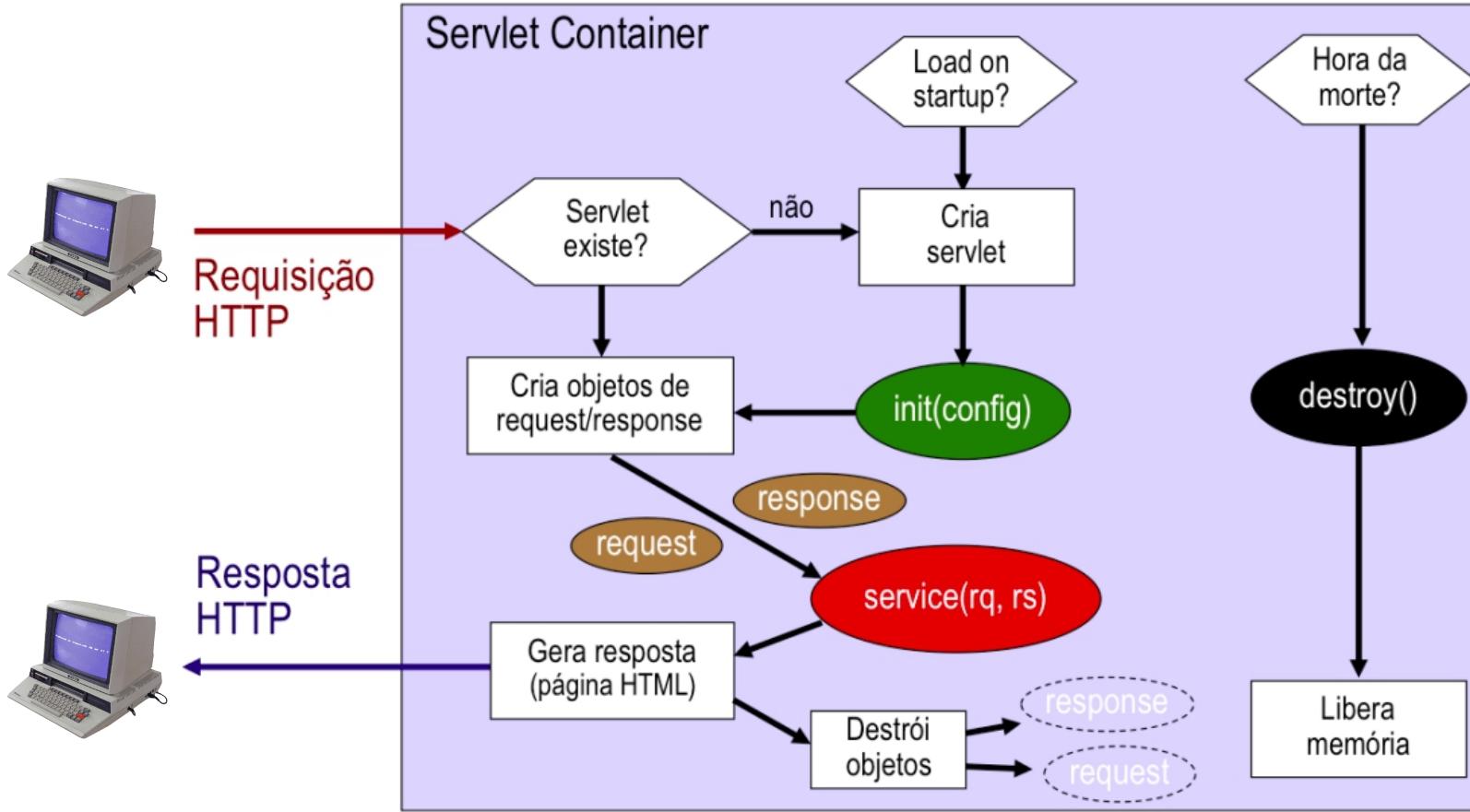
# 2 método `service()`

- *Esse método é invocado quando chega uma solicitação do cliente;*
- *Os parâmetros da assinatura desse método são definidos por um objeto `ServletRequest` e um objeto `ServletResponse`;*
- *O objeto `ServletRequest` contém a solicitação do cliente e o objeto `ServletResponse` contém a resposta do servlet.*
- *Na prática, esse método determina qual método HTTP (GET, POST ou outro) chamar no servlet.*

# 3 método `destroy()`

- *Esse método tem como objetivo remover o servlet;*
- *Geralmente isso ocorre quando o Container é fechado ou a memória está livre, ou seja, esse método limpa qualquer recurso que está ainda ativo.*

# VISÃO GERAL DO CICLO DE VIDA



# CICLO DE VIDA DO SERVLET

- .. O container controla o ciclo de vida de um WebServlet;
- .. O servlet pode ser carregado e inicializado (ou inicializado com a primeira requisição);
- .. **Enquanto estiver ativo**, pode receber requisições;
- .. A cada requisição, objetos request e response serão criados, configurados e passados para o método **service( )**, que o repassa a um método **doGet()**, **doPost()**, **ou outro** de acordo com o **protocolo HTTP** recebido;
- .. Ao final da execução do método, os dois objetos são destruídos;
- .. O servlet permanece ativo até que seja destruído.

# CICLO DE VIDA DO SERVLET

- ∴ A requisição é multithread:
  - ∴ *O mesmo objeto;*
  - ∴ *Thread separada;*
- ∴ Atributos serão compartilhados;
  - ∴ *Ao se alterar um atributo, todas as threads terão seus atributos alterados também;*
  - ∴ *Race condition;*
  - ∴ *Variáveis dentro de métodos doXXX() são thread-safe;*
  - ∴ *Objetos request e response são thread-safe;*

# REDIRECIONAMENTOS

# Redirecionar ou Despachar?

```
resp.sendRedirect("http://google.com");
```

```
RequestDispatcher rd = req.getRequestDispatcher("/outra-pagina.jsp");
rd.forward(req, resp);
```

# OBRIGADO

perguntas?

[Jefferson.chaves@ifpr.edu.br](mailto:Jefferson.chaves@ifpr.edu.br)

45 998508359

[github.com/jeffersonchaves](https://github.com/jeffersonchaves)