**Seção 1 – Construindo o WebServer**

Explicar a primeira parte do trabalho

**Seção 2 – Tratando as requisicoes**

No momento temos um Webserver recebendo chamadas de forma multithread. Entretando o nosso servidor ainda não consegue reconhecer as chamadas que está recebendo para dar ao usuário uma resposta de acordo.

Para começarmos a entender as requisições do usuário precisamos primeiramente entender qual o tipo de requisição HTTP o usuário está realizando (GET, POST, PUT, DELETE e etc) e após isso conseguimos verificar onde o usuário está realizando a requisição.

Dividindo esse processo em 2 etapas vamos primeiramente reconhecer em nosso servidor o tipo de requisição do usuário. Para isso vamos pegar o nosso leitor de Buffer que declaramos em

*BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));*

Vamos realizar a leitura desse buffer e atribuí-la em uma string

*String requestLine = br.readLine();*

Agora na variável requestLine ja temos as informações da requisição realizada pelo usuário que precisamos, entretando precisamos dividir essas informações entre o tipo de requisição do usuário e o caminho de requisição do usuário, como por exemplo no caso em que o usuário realizar uma requisição GET em “localhost:6789/about” precisamos reconhecer que essa foi uma requisição GET no diretório “about”. Para realizarmos essa operação vamosdividir a String em tokens da seguinte forma.

*StringTokenizer tokens = new StringTokenizer(requestLine);*

Dentro dos nossos tokens temos uma string no seguinte formato:

“Tipo de requisição” “diretório” “HTTP/1.1”

Logo no primeiro token da nossa variável tokens vamos ter o tipo de requisição e no segundo token o diretório que o usuário realizou a requisição. E podemos dividir essas duas informações dessa forma

*String requestType = tokens.nextToken();*

*String fileName = tokens.nextToken();*

Para finalizer essa seção precisamos apenas tratar o arquivo requisitado apontando para o diretório atual

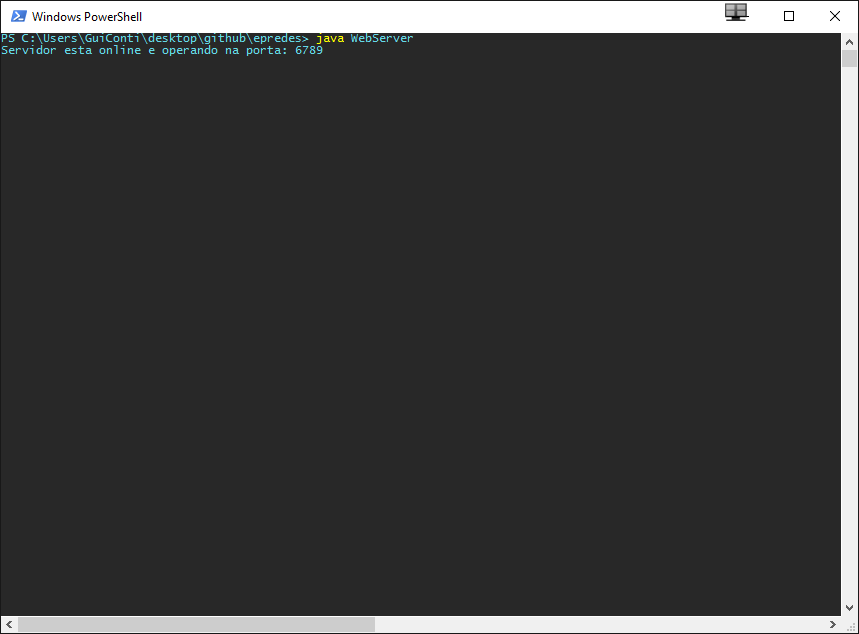
*fileName = "." + fileName;*

Para realizarmos alguns testes coloquei no nosso código a seguinte linha

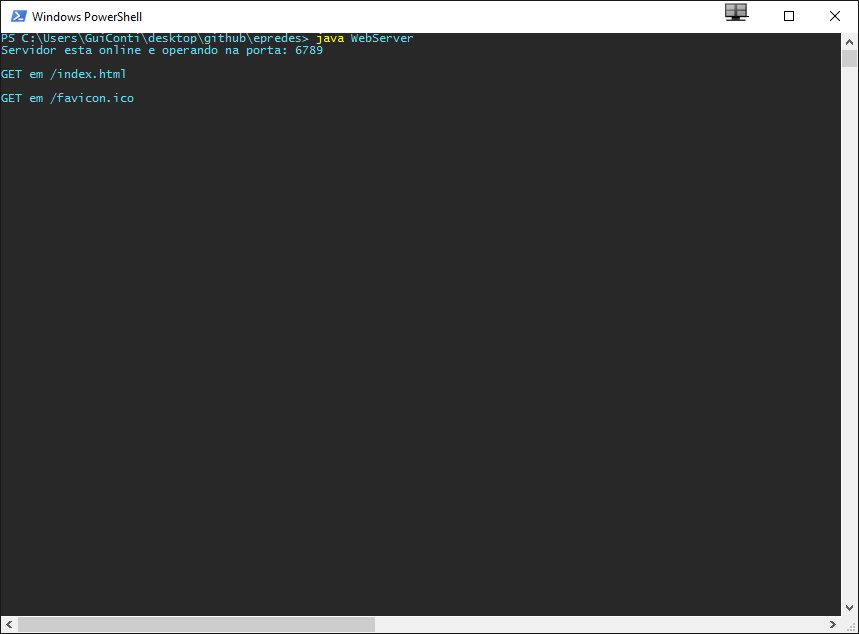
*System.out.println(requestType + " em " + fileName);*

E seguem alguns testes

Inicializando servidor

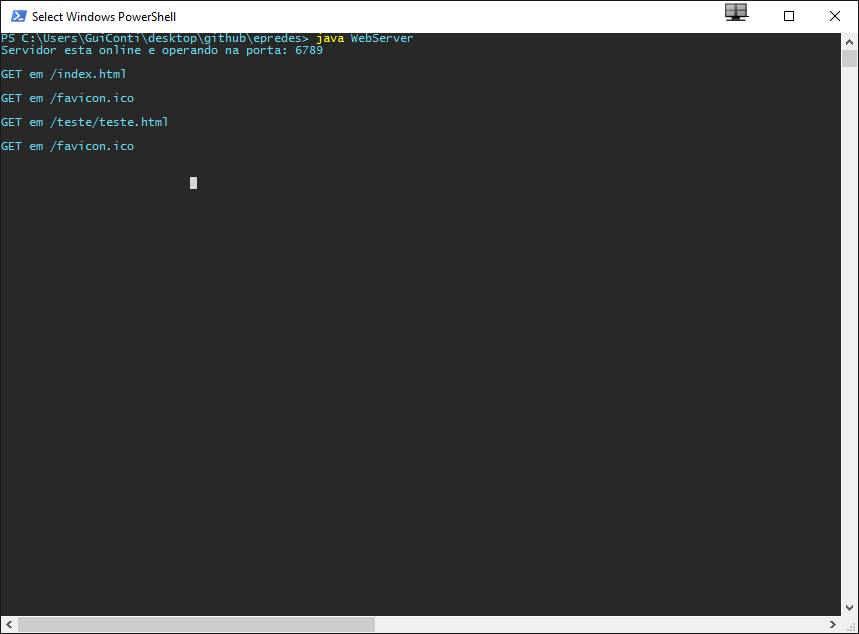


Acessando no browser um get em localhost:6789/index.html

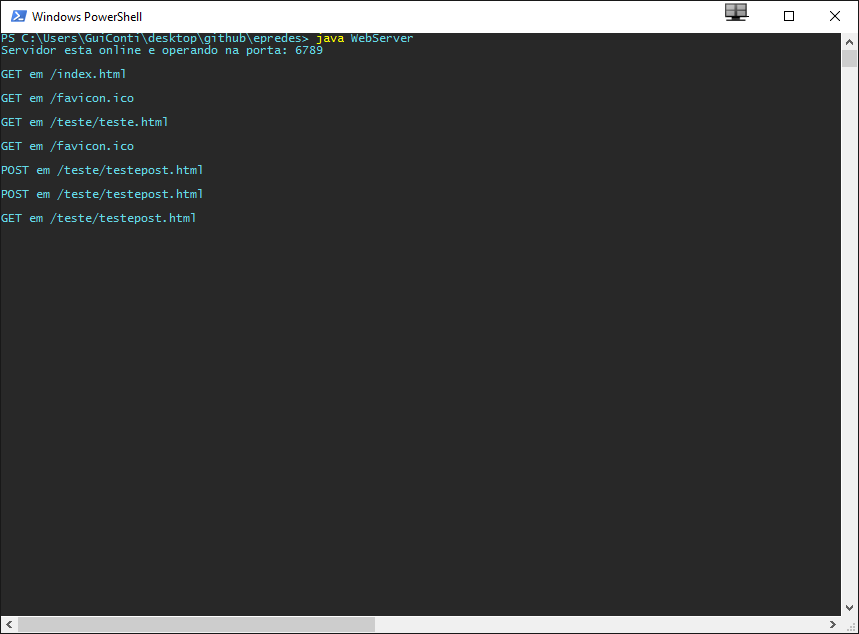


OBS: O favicon.ico é uma tentativa do browser de requisitar o ícone que fica na aba no navegador então por conveniência podemos ignorar.

Realizando agora um GET em localhost:6789/teste/teste.html



Por fim realizamos um POST em teste/testepost.html



Pronto com isso temos as nossas requisições tratadas e podemos continuar para a nossa proxima seção onde vamos entender as requisições do usuário e dar respostas de acordo com o que foi pedido.

**Seção 3 – Respondendo o usuário**

Nesse momento temos um Webserver que recebe e reconhece as requisições do usuário, entretanto o nosso servidor ainda não da nenhuma resposta independente da requisição que o usuário faça. Nesse seção nós vamos mudar isso.

Por conveniência nós não vamos dar diferentes respostas dependendo do tipo de requisição do usuário. Vamos apenas reconhecer o diretório que o usuário acessou e dar a resposta sobre o diretório.

Para começarmos essa estrutura primeiramente precisamos avaliar se o diretório que o usuário solicitou existe no nosso servidor. Vamos realizar essa operação da seguinte maneira

*FileInputStream fis1 = null;*

*boolean fileExists = true;*

*try {*

*fis1 = new FileInputStream(fileName);*

*} catch (FileNotFoundException e) {*

*fileExists = false;*

*}*

Primeiramente criamos um FileInputStream que, caso o arquivo exista, irá guardar as informações da página nele. Caso o servidor tenta pegar as informações do diretório requisitado e ocorra um erro de arquivo não encontrado vamos atribuir a uma variável auxiliar o valor de false para que dessa forma possamos devolver uma resposta de 404 Not found adequada.

Após ler o nosso arquivo precisamos devolver o arquivo para o usuário caso ele existe. Para isso primeiro vamos criar uma função que lê o nosso arquivo constrói os buffers e utiliza o nosso DataOutputStream para dar a resposta para o usuário.

*private static void sendBytes(FileInputStream fis, OutputStream os)*

*throws Exception {*

*byte[] buffer = new byte[1024];*

*int bytes = 0;*

*while((bytes = fis.read(buffer)) != -1 ) {*

*os.write(buffer, 0, bytes);*

*}*

*}*

A nossa função recebe o arquivo lido em fis. Cria um buffer de 1k para comportar os bytes no caminho para o socket e depois realiza um loop no nosso arquivo lendo os bytes e utilizando o nosso OutputStream para enviar para o usuário.

Com isso já conseguimos ler arquivos no nosso diretório e enviar para o usuário, para finalizar a construção desse seção precisamos tratar os casos em que o usuário tentou requisitar algo que não existe em nosso servidor. Para realizar essa tarefa vamos utilizar o seguinte código.

*String entityBody = null;*

*if (fileExists) {*

*// Envia o conteudo do arquivo requisitado*

*sendBytes(fis1, os);*

*// Enviar uma linha em branco para indicar o fim das linhas de cabeçalho.*

*os.writeBytes(CRLF);*

*} else {*

*entityBody = "<HTML>" +*

*"<HEAD><TITLE>404 Not Found</TITLE></HEAD>" +*

*"<BODY>404 Not Found</BODY></HTML>";*

*// Enviar a linha de conteudo.*

*os.writeBytes(entityBody);*

*// Enviar uma linha em branco para indicar o fim das linhas de cabeçalho.*

*os.writeBytes(CRLF);*

*}*

Dividimos a resposta em 2 cenários. O primeiro onde na nossa variável boolean auxiliar detectamos que o arquivo existe e dessa forma chamamos a função criada anteriormente para ler e devolve para o usuário o conteúdo. No segundo o arquivo solicitado não existe e então construímos uma resposta 404 no momento e devolvemos para o usuário. Não criamos um html para esse cenário por conveniência, entretanto o correto seria termos um html especial para tratar esse evento.

Para retratar alguns testes criei 1 HTML chamado index.html dentro do diretório base com o seuginte conteúdo:

*<html>*

*<head>*

*<meta charset="utf-8" />*

*<title>EP Redes</title>*

*</head>*

*<body>*

*<p>Olá esse é um servidor web realizado em Java!</p>*

*</body>*

*</html>*

Outro arquivo HTML chamado teste.html dentro do diretório teste, com o seguinte conteúdo

*<html>*

*<head>*

*<meta charset="utf-8" />*

*<title>Teste</title>*

*</head>*

*<body>*

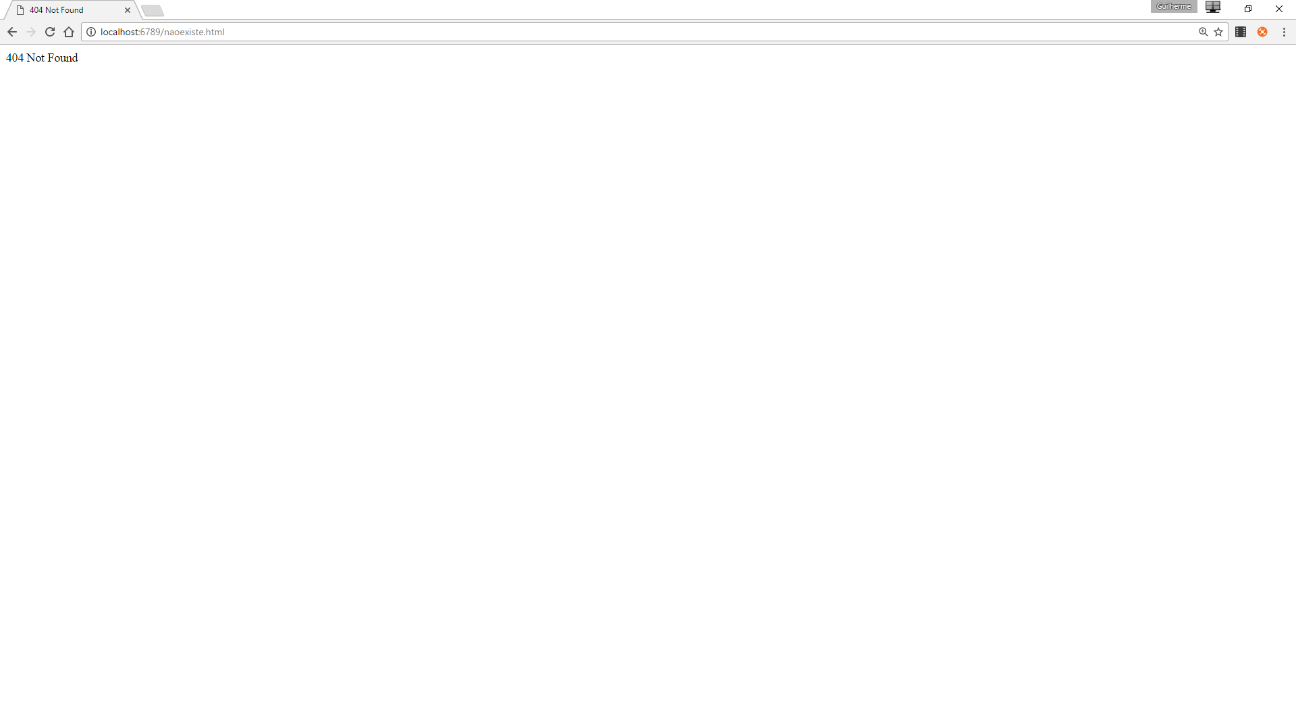
*<p>Esse é um teste com um arquivo dentro de um diretório!</p>*

*</body>*

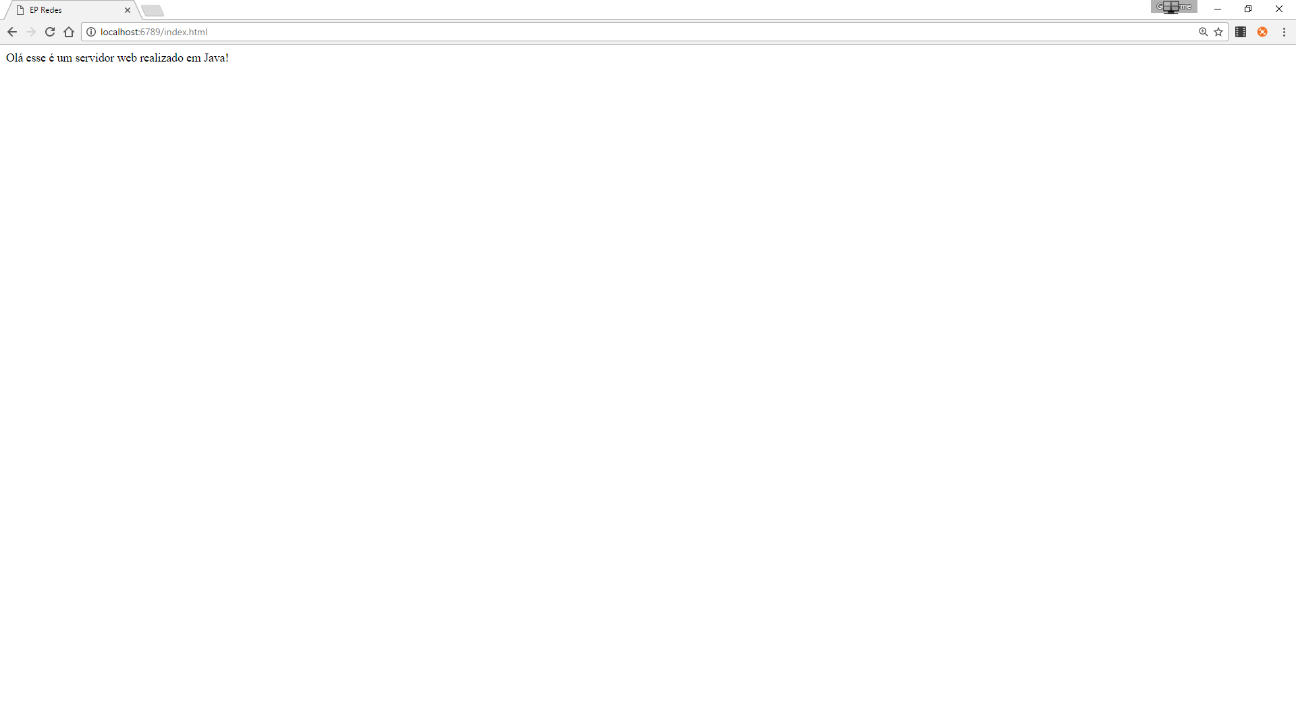
*</html>*

E por fim uma imagem em formato JPEG com o nome example.jpeg, também dentro do diretório teste. E os resultado dos testes seguem.

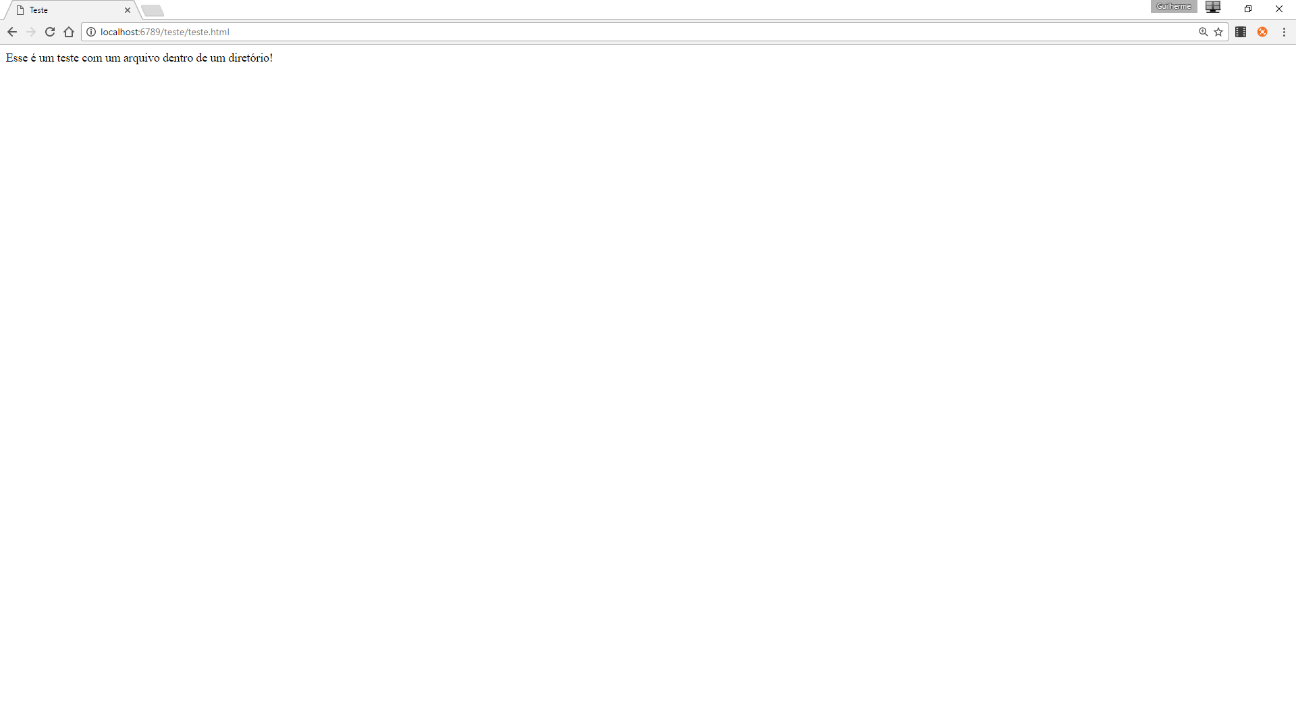
Primeiramente tentamos acessar algo que não existe



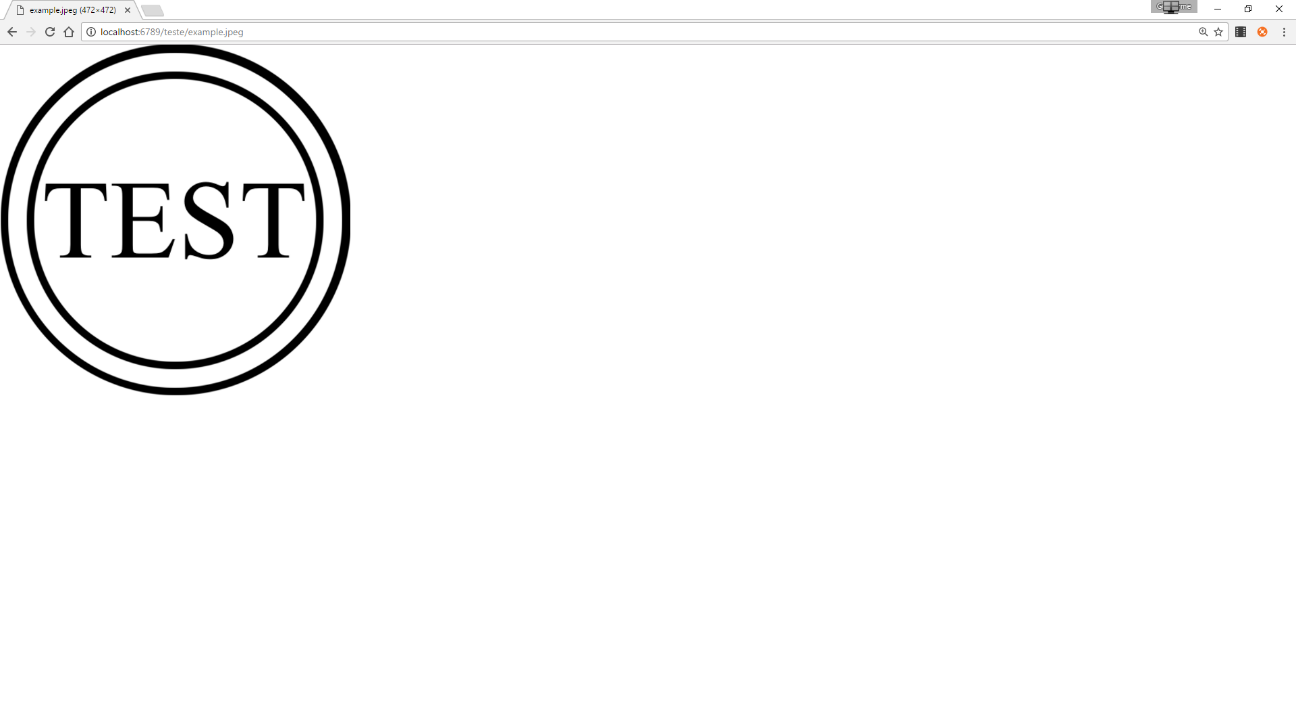
Depois tentamos acessar o nosso index.html



Posteriormente tentamos acessar o nosso teste.html dentro do sub-diretório teste



E por fim tentamos acessar a nossa imagem example.jpeg



Com isso feito já conseguimos devolver para o usuário qualquer as respostas para os diretórios que ele acessar com base nos arquivos que possuímos nos nossos diretórios, temos então um Webserver que já poderia ser aplicado em diversos cenários do mundo real. Vamos continuar na próxima seção a armazenar os Logs de requisição do usuário.