

Guilherme Conti Teixeira - 7972262

Lucas dos Santos Lazarini - 8082810

Introdução

Nessa atividade vamos desenvolver um Webserver com as funcionalidades que contém RFC-1945. Durante todo o processo vamos buscar explicar cada decisão tomada, desenvolvendo todo o Webserver de forma explicativa a cada funcionalidade desenvolvida. Todo o nosso Webserver irá possuir 2 classes. Uma denominada WebServer e outra HttpRequest, ambas dentro do arquivo WebServer.java. Todas as compilações ocorrerão da seguinte forma: “javac WebServer.java” e a execução da seguinte forma: “java WebServer”. Todas as seções irão constituir o desenvolvimento de uma funcionalidade, acompanhando a explicação de cada passo e no fim de cada seção demonstraremos exemplos da funcionalidade criada.

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. Entretanto 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 já temos as informações da requisição realizada pelo usuário que precisamos, entretanto 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 vamos dividir 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 finalizar essa seção precisamos apenas tratar o arquivo requisitado apontando para o diretório atual

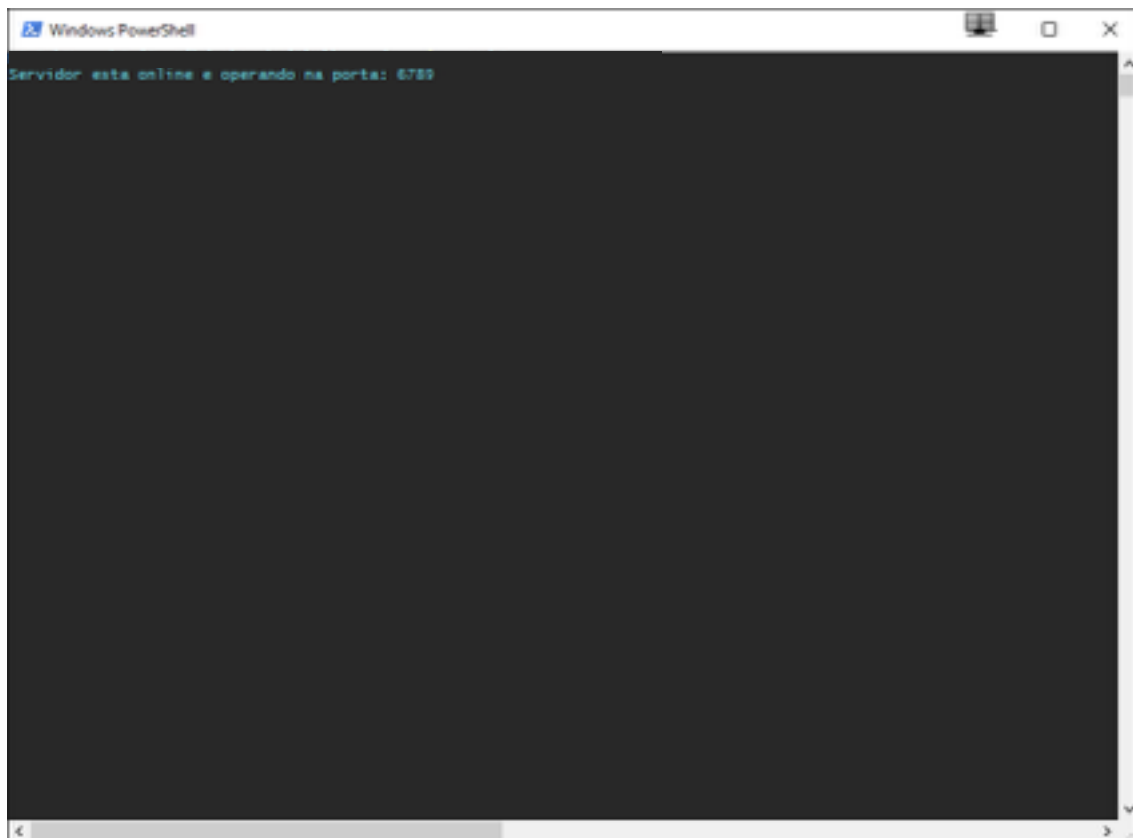
```
fileName = "." + fileName;
```

Para realizarmos alguns testes coloquei no nosso código a seguinte linha

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

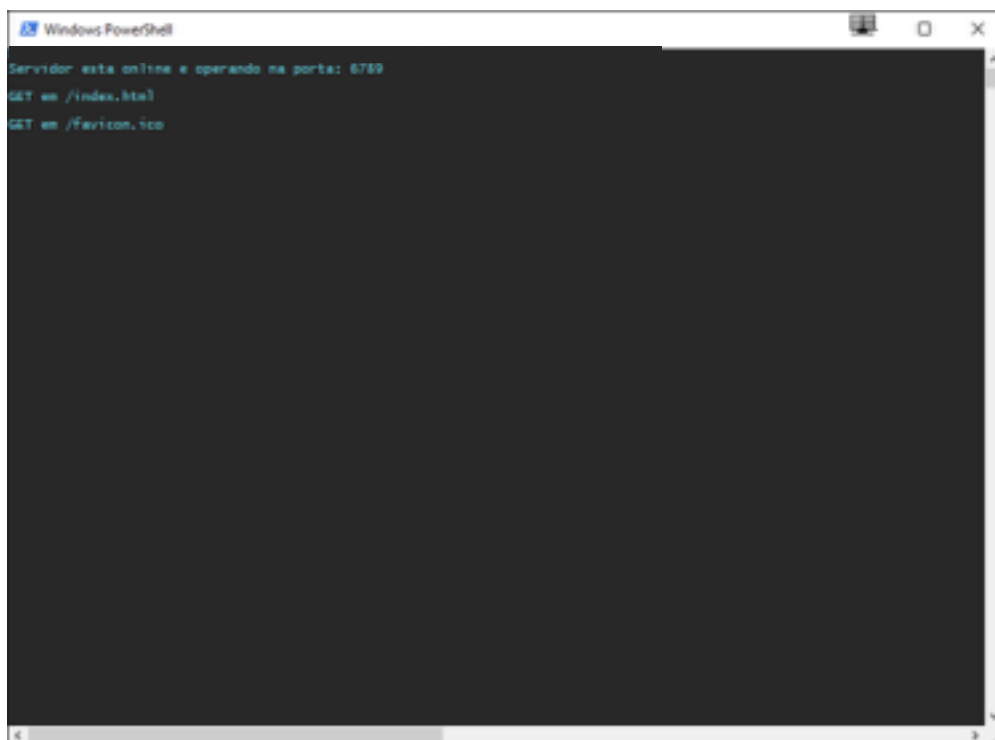
E seguem alguns testes

Inicializando servidor



```
Windows PowerShell
Servidor esta online e operando na porta: 6789
```

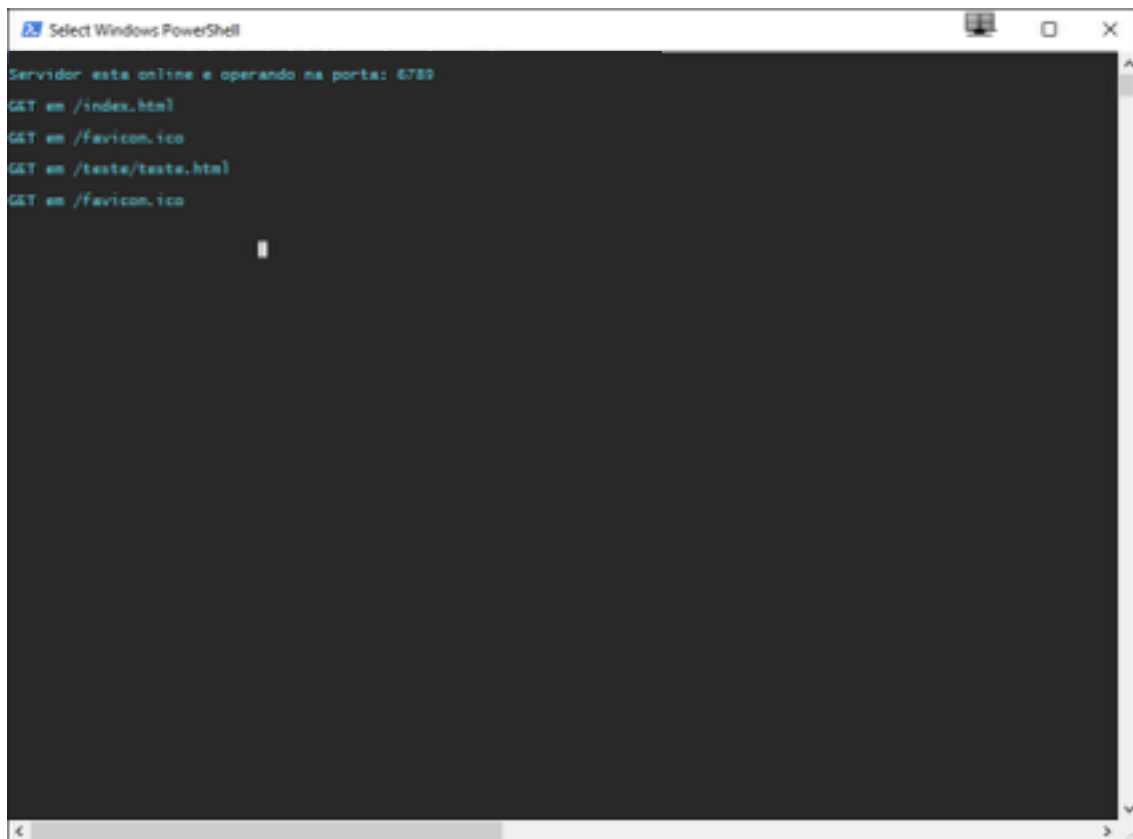
Acessando no browser um get em localhost:6789/index.html



```
Windows PowerShell
Servidor esta online e operando na porta: 6789
GET em /index.html
GET em /favicon.ico
```

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

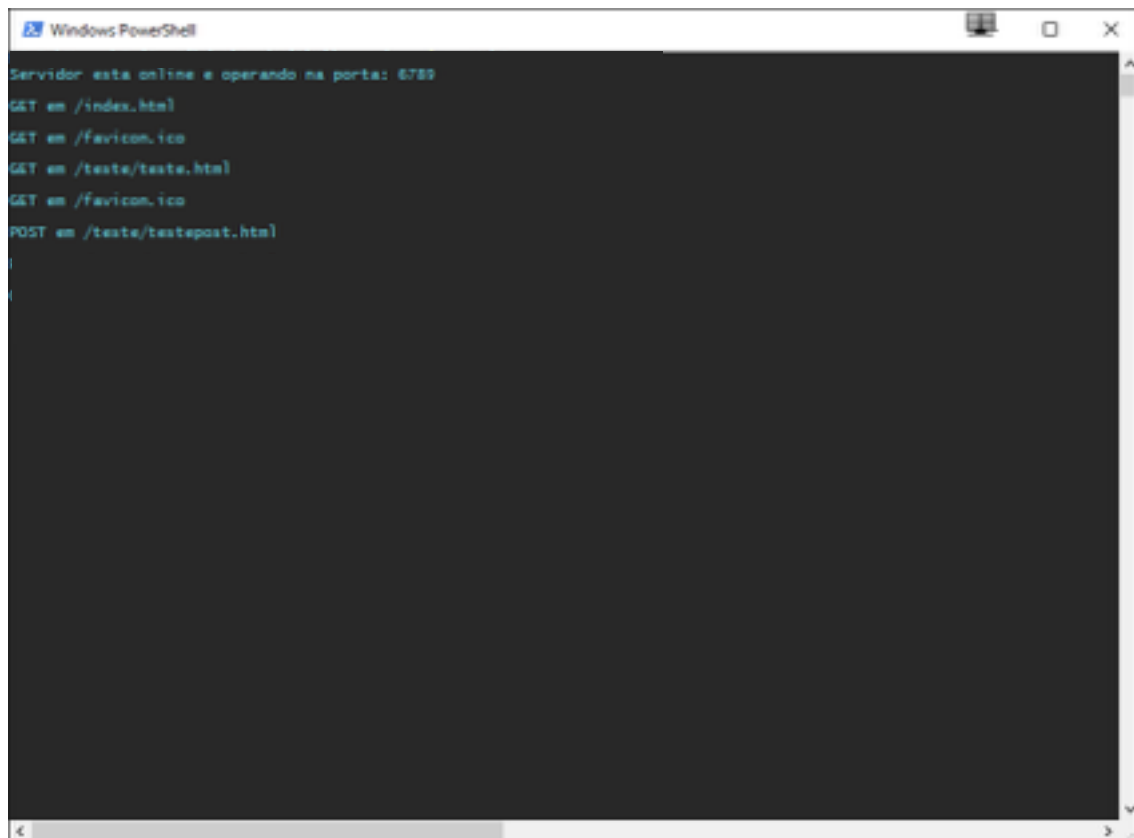
Realizando agora um GET em localhost:6789/teste/teste.html



```
Select Windows PowerShell

Servidor esta online e operando na porta: 6789
GET em /index.html
GET em /favicon.ico
GET em /teste/teste.html
GET em /favicon.ico
```

Por fim realizamos um POST em teste/testepost.html

A screenshot of a Windows PowerShell window. The title bar reads "Windows PowerShell". The window has a dark background with light-colored text. The text inside the window is as follows:

```
Servidor esta online e operando na porta: 6789  
GET em /index.html  
GET em /favicon.ico  
GET em /teste/teste.html  
GET em /favicon.ico  
POST em /teste/testepost.html
```

Pronto com isso temos as nossas requisições tratadas e podemos continuar para a nossa próxima 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 dá 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 seguinte 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.

Seção 4 - Gerando Logs

Nesta seção vamos gerar os Logs das requisições do usuário ao Webserver.

Para gerar os Logs nós precisamos primeiramente pegar as informações da requisição do usuário endereço origem, porta origem, horário da requisição, conteúdo requisitado, quantidade de bytes transmitidas em resposta a requisição. Os itens endereço de origem, porta origem e conteúdo requisitado nós já possuímos na requisição que o usuário só. Para pegar essas informações vamos utilizar as outras linhas do nosso Buffer de leitura. Para lembrar temos um Buffer que realiza a leitura da requisição e armazenamos a primeira linha desse Buffer nessa linha de código

```
String requestLine = br.readLine();
```

As outras informações da requisição estão nas outras linhas do buffer, logo realizar um loop nas linhas restantes do buffer e armazenada em uma variável String da seguinte forma.

```
String headerLine = null;
```

```
Date actualDate = new Date();
```

```
BufferedWriter out = new BufferedWriter(new FileWriter("log.txt", true));
```

```
while ((headerLine = br.readLine()).length() != 0) {
```

```
    try {
```

```
        out.write(headerLine + "\n");
```

```
    } catch (Exception e) {
```

```
        System.out.println("Write Exception");
```

```
    }
```

```
}
```

```
out.write("Data: " + actualDate + "\n\n");
```

```
out.close();
```

Primeiramente iniciamos a variável String que irá receber as próximas linhas da nossa requisição, uma outra variável que irá armazenar a data atual pedida no log.

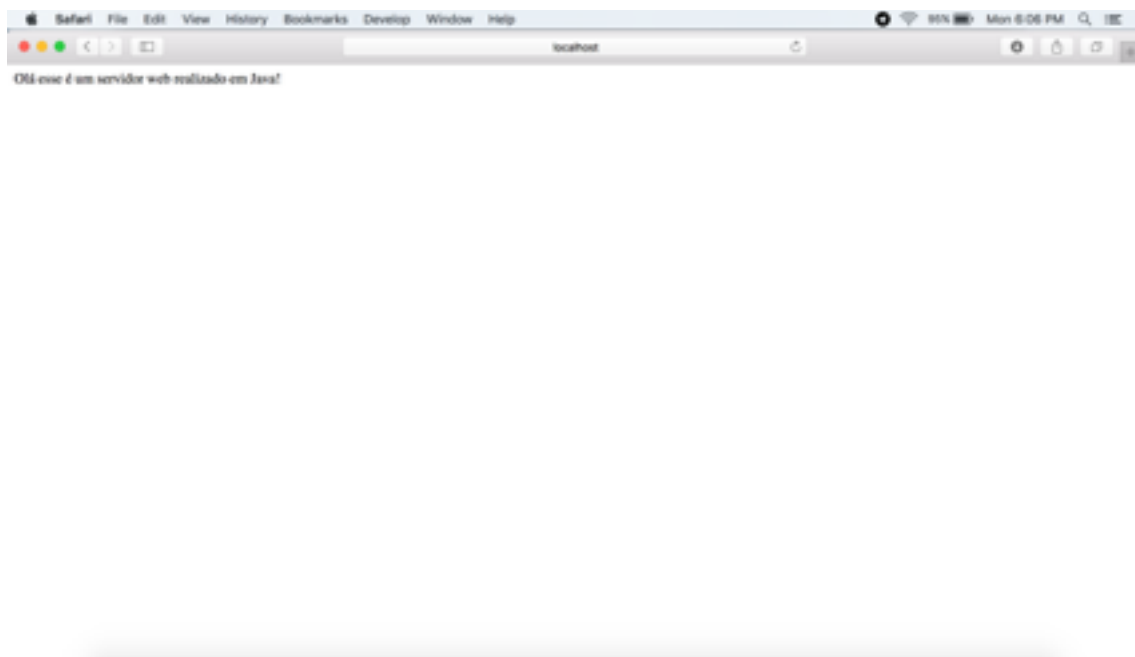
Então abrimos o nosso arquivo, que se encontrará no diretório root do servidor com o nome "log.txt" e escreveremos as informações nesse log. Inserimos o "\n" ao final de

cada inserção no arquivo para criarmos quebras de linha no log. Agora precisamos apenas do tamanho da resposta que enviamos para o usuário. Para isso vamos utilizar a nossa função `sendBytes` que criamos na seção anterior. Para reconhecer esse valor vamos usar a nossa variável `bytes` que em cada iteração do loop envia uma quantidade de bytes para o usuário. Adicionamos então uma variável do formato Integer denominada `bytesLogSum` e instanciamos ela com o valor 0. Em cada iteração do loop nós somamos a variável `bytes` a nossa variável auxiliar `bytesLogSum` e por fim retornamos a quantidade total de bytes (Obs: para isso mudamos o tipo da função de void para integer).

Esse é um bom momento para mudarmos a nossa resposta 404 para um html próprio. Para que possamos manter uma consistência na leitura dos bytes enviados. Para isso criamos um `notfound.html` no root do servidor e toda vez que não conseguimos ler o arquivo solicitado lemos então o `html notfound.html` e enviamos para o usuário utilizando a nossa função `sendBytes` novamente.

Por fim vamos realizar alguns testes. Para realizar o teste precisamos apenas acessar uma página qualquer e verificar no root do nosso servidor o arquivo `log.txt` e lá constará as informações do acesso atual e acessos anteriores.

Vamos começar acessar `localhost:6789/index.html`



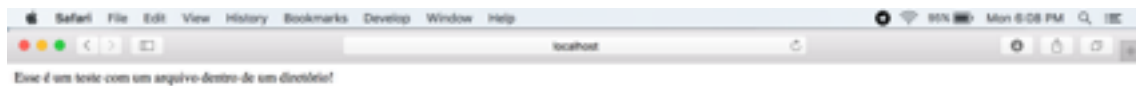
E no nosso arquivo `log.txt` temos

```
Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=18800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:06:13 BRT 2016
Arquivo requisitado: ./index.html
Total de bytes enviados: 188
```

Podemos validar esse resultado olhando o tamanho do arquivo index.html

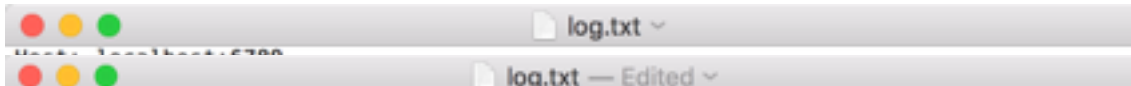


Para o nosso próximo arquivos vamos abrir teste/teste.html



E o resultado no nosso log.txt

Por fim abrimos teste/example.jpeg e temos



```

Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=10800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7
(KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:06:13 BRT 2016
Arquivo requisitado: ./index.html
Total de bytes enviados: 188

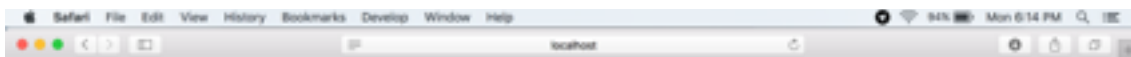
Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=10800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7
(KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:08:31 BRT 2016
Arquivo requisitado: ./teste/teste.html
Total de bytes enviados: 194

Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=10800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7
(KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:09:56 BRT 2016
Arquivo requisitado: ./teste/example.jpeg
Total de bytes enviados: 19411

```

E no log.txt

Caso queira acessar o log diretamente pela web podemos também acessar localhost: 6789/log.txt e teremos



```

Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=10800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:04:13 BRT 2016
Arquivo requisitado: ./index.html
Total de bytes enviados: 188

Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=10800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:08:31 BRT 2016
Arquivo requisitado: ./teste/teste.html
Total de bytes enviados: 194

Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: jenkins-timestamper-offset=10800000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Data: Mon Sep 26 18:09:56 BRT 2016
Arquivo requisitado: ./teste/example.jpeg
Total de bytes enviados: 19411

```

Pronto, com isso terminamos a parte do log do nosso servidor. Agora todas as requisições realizadas serão armazenadas dentro do nosso arquivo log.txt e estão disponíveis para consulta. Na próxima seção vamos realizar a funcionalidade de listagem de diretórios.

Seção 5 - Listagem de diretório

Primeiramente precisamos reconhecer se o acesso do usuário se refere a um diretório ou um arquivo em específico. Para isso vamos utilizar a nossa variável “fileName” que contém o caminho requisitado pelo usuário. Por exemplo caso o usuário acesse “localhost:6789/teste/teste.html” fileName possuirá /teste/teste.html, caso o usuário acesse “localhost:6789/teste/teste” fileName possuirá /teste/teste. Portanto podemos utilizar essa variável para verificar se o usuário está tentando acessar um diretório ou um arquivo em específico.

Para realizar essa verificação vamos utilizar uma nova função chamada contentType que recebe como parâmetro uma String que será o nosso fileName. A função é denotada a seguir.

```
private static String contentType(String fileName){
```

```
    if(!fileName.contains(".")) {
```

```
        return "directory";
```

```
    }
```

```

        if(fileName.endsWith(".htm") || fileName.endsWith(".html")) {
            return "text/html";
        }

        if(fileName.endsWith(".gif") || fileName.endsWith(".GIF")) {
            return "image/gif";
        }

        if(fileName.endsWith(".jpeg")) {
            return "image/jpeg";
        }

        if(fileName.endsWith(".txt")) {
            return "text/html";
        }

        return "application/octet-stream";

    }
}

```

Vamos utilizar essa função para determinar o tipo de requisição do usuário e tratar de acordo. Caso o usuário requisitar um diretório a função `contentType` irá retornar "directory" e então podemos evoluir com a listagem ou não do diretório. Para realizar essa verificação antes de lermos o arquivo requisitado pelo usuário vamos inserir o código a seguir.

```

    if (contentType(fileName) == "directory") {
    }

```

Com isso nós conseguiremos saber o tipo de requisição que o usuário está realizando.

Para prosseguirmos com a requisição caso o usuário tenha requisitado um diretório precisamos avaliar nas configurações qual tipo de exibição que iremos adotar. No caso teremos 3 configurações possíveis. 1 - Iremos listar os arquivos que estão dentro do diretório. 2 - Não iremos listar os arquivos que estão dentro do diretório, e caso haja a tentativa, vamos mostrar uma mensagem dizendo que a listagem não está

disponível. 3 - Na tentativa de acesso ao diretório vamos retornar uma página padrão (index.html) que irá possuir algum conteúdo.

Para tornarmos essa configuração de fácil manuseio e que permita que a mesma seja alterada sem o desligamento do servidor vamos manter a configuração em um arquivo txt dentro do root do servidor e vamos ler esse arquivo para verificar qual a configuração que o servidor se encontra no momento.

Para isso criamos um txt config.txt no root do servidor que irá possuir apenas 1 número. Sendo ele 1, 2 ou 3, que irá definir o tipo de listagem de diretório.

Com isso feito podemos inserir na nossa verificação de diretório também a verificação de configuração do servidor da seguinte forma

```
try {
```

```
    BufferedReader brConfig = new BufferedReader(new  
FileReader("config.txt"));
```

```
    String configLine = brConfig.readLine();
```

```
    StringTokenizer configTokens = new StringTokenizer(configLine);
```

```
    String config = configTokens.nextToken();
```

```
    brConfig.close();
```

```
} catch (Exception e) {
```

```
    System.out.println("Error reading config file.");
```

```
}
```

Nessa etapa lemos o arquivo config.txt e utilizamos tokens para separar a informação da configuração do resto que possivelmente pode conter no nosso txt. Após isso podemos verificar qual o tipo de configuração que temos e tratar de acordo com as nossas regras da seguinte forma

```
if (config.equals("1")){
```

```
    File folder = new File("." + fileName);
```

```
    File[] listOfFiles = folder.listFiles();
```

```
    String contentHtml = "<html>\n<body>";
```

```
    for (int i = 0; i < listOfFiles.length; i++) {
```

```
        if (listOfFiles[i].isFile()) {
```

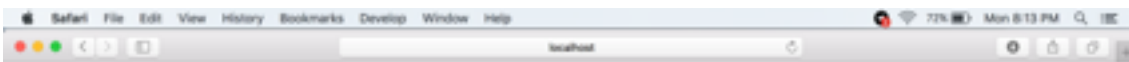
```
            contentHtml = contentHtml + "\n" + "<br/  
>" + "File " + listOfFiles[i].getName();
```

```
        } else if (listOfFiles[i].isDirectory()) {
```

```
            contentHtml = contentHtml + "\n" + "<br/  
>" + "Directory: " + "Directory " + listOfFiles[i].getName();
```

```
        }
```

```
    }
```



```
contentHtml = contentHtml + "\n" + "</body>" + "\n" +  
"</html>";
```

```
BufferedWriter outHtml = new BufferedWriter(new  
FileWriter("directory.html", false));
```

```
outHtml.write(contentHtml);
```

```
outHtml.close();
```

```
} else if (config.equals("2")) {
```

```
    fileName = "/forbidden.html";
```

```
} else {
```

```
    fileName = "/index.html";
```

```
}
```

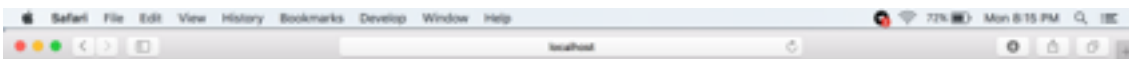
Nesse código identificamos a configuração do arquivo e caso a configuração seja 1 nós utilizamos uma funcionalidade do Java para ler os arquivos e diretórios dentro daquele diretório e criamos um html com essa informação para posteriormente indicarmos esse html para o fileName que será processador normalmente pela nossa função de resposta para o usuário. No caso 2 apenas indicamos o fileName como sendo o arquivo que contém a resposta de listagem não disponível para o diretório. E no caso 3 nós realizamos um processo semelhante ao caso 2, porém dessa vez indicamos o fileName para o arquivo index.html para ser enviado ao usuário.

Para testarmos seguem alguns exemplo.

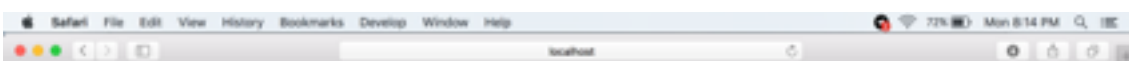
Com configuração em 1:

Acessando localhost:6789/teste

Acessando o localhost:6789/



```
File_D5_Store  
Directory: Directory.git  
File config.txt  
File directory.html  
File filename.txt  
File forbidden.html  
File HttpRequest.class  
File index.html  
File log.txt  
File notfound.html  
File README.MD  
File Relatorio.docx  
Directory: Directory teste  
File todo.txt  
File WebServer.class  
File WebServer.java
```

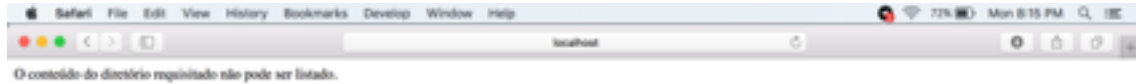


Esse é um teste com um arquivo dentro de um diretório!

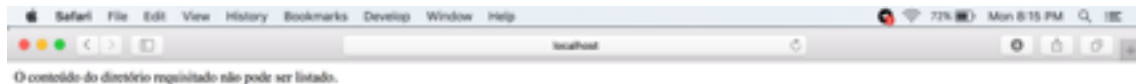
Acessando o localhost:6789/teste/teste.html

Agora com configuração 2

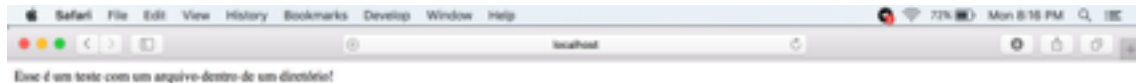
Acessando localhost:6789/teste



Acessando localhost:6789/

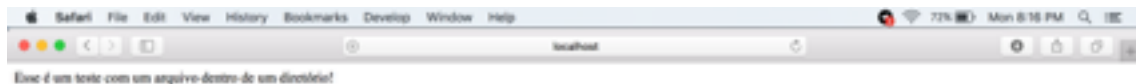


Acessando localhost:6789/teste/teste.html

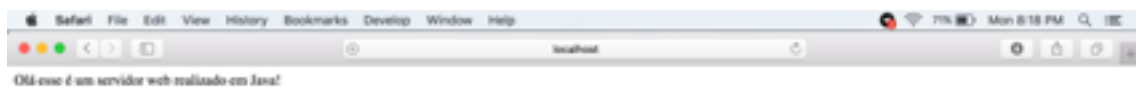


Agora com a configuração em 3

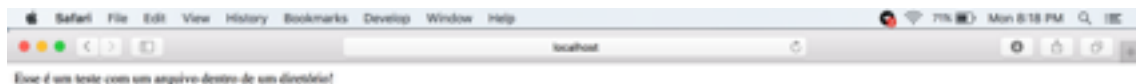
Acessando localhost:6789/teste (Obs: A página contém o index.html)



Acessando localhost:6789/ (Obs: A página contém o index.html)



Acessando localhost:6789/teste/teste.html



E com isso temos a listagem de diretório configurada e preparada para ser utilizada da maneira que desejarmos. Em seguida vamos finalizar o nosso webserver criando a funcionalidade de autenticação em diretórios privados.

Seção 6 - Autenticação