EE544 Project 2

Multi-Threaded Web Server

California State University Los Angeles.

Group Members:

Babar Hassan Baig	CIN 304xxxxxx
Money Gera	CIN 304xxxxxx

ACKNOWLEDGMENT

We would like to express our gratitude to all those who gave us the possibility to complete this report. We want to thank the Department of Electrical and Computer Engineering for giving us permission to commence this report in the first instance, to do the necessary research work and to use departmental data.

We are deeply indebted to our professor DR. Yi Zhao whose help, stimulating suggestions and encouragement helped us in all the time of research and writing of this report.

TABEL OF CONTENT

.

PROJECT OBJECTIVES:

In this project our team was given a task to create a multi-threaded web server which is capable of processing multiple simulation server requests in parallel, which means our web server is multi-threaded. Our primary task is to complete the following main task.

1. Test your website when there is no webpage existing.
2. When multiple users access the same webpage.
3. Run your web server on one of the campus computer, and run your client (IE) on another campus computer.
4. Run your web server on one of the campus computer, and run your client (IE) on a laptop (using wireless access);
5. (optional) Run your web server on one of the campus computer, and run your client (IE) on your home network;
6. (optional) Run your web server on your home network, and run your client (IE) on one of the campus computer.

We also need to get following test result:

Test results:

a. Your test homepage distributed by your Java-based Web Server.

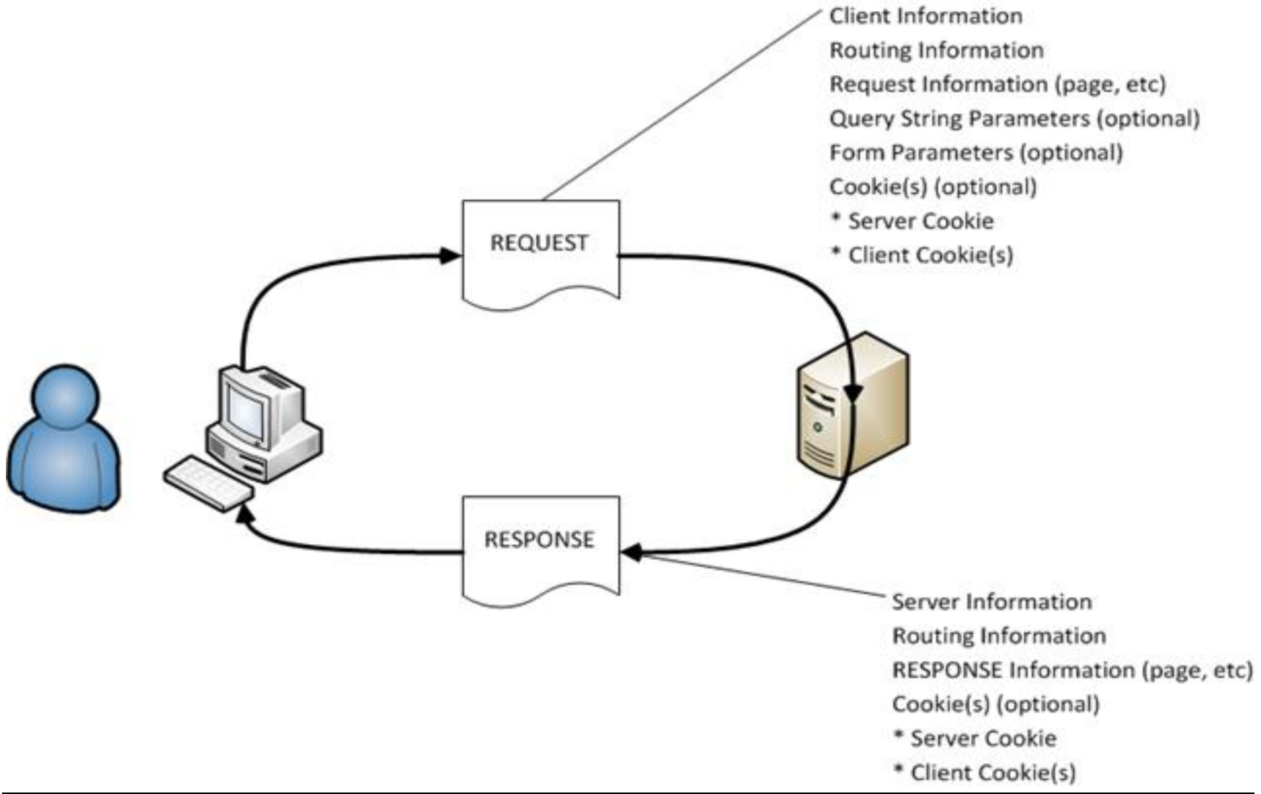
b. The message displayed when the requested homepage is not available;

c. The HTTP request messages displayed by the server during the transmission of your homepage.

WEB SERVER:

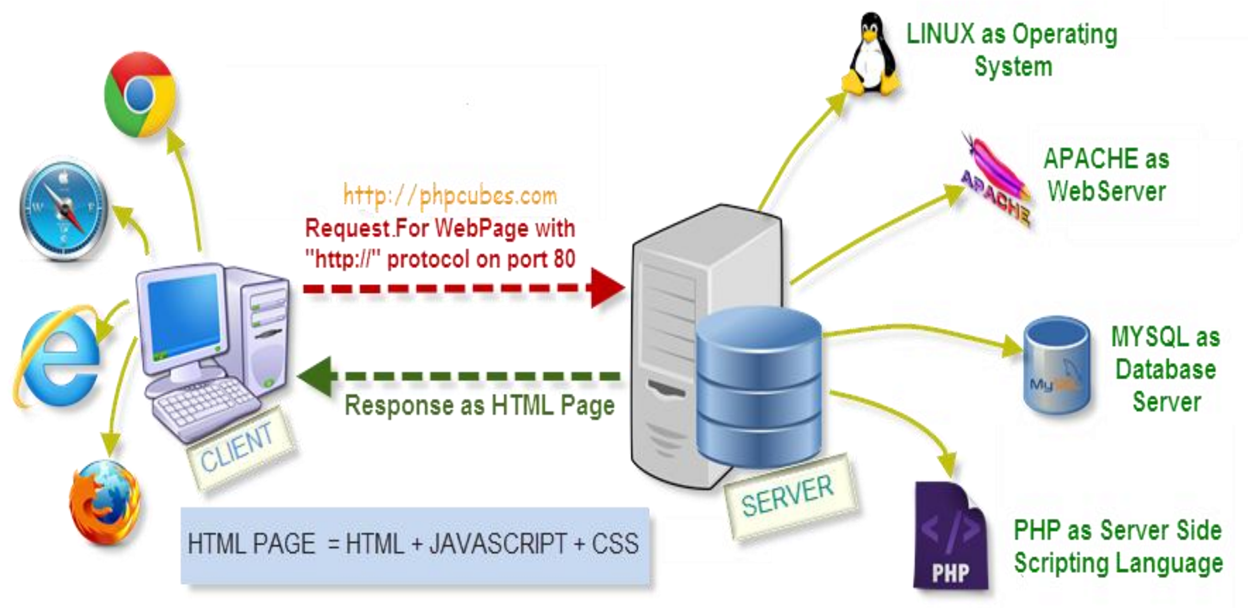
A web servers function is in its name it serves web pages. The primary function of the web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the HTTP. Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.

The most common use of web servers is to host websites, but there are other uses such as gaming, data storage or running enterprise applications.



INTRODUCTION To HTTP:

HTTP stands for Hypertext Transfer Protocol. It is foundation of data communication for the World Wide Web. HTTP is an application protocol for distributed, collaborative, hypermedia information system. Hypertext is the structured text that use hyperlinks between nodes containing text. HTTP is the protocol to exchange or transfer hypertext. The standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs).

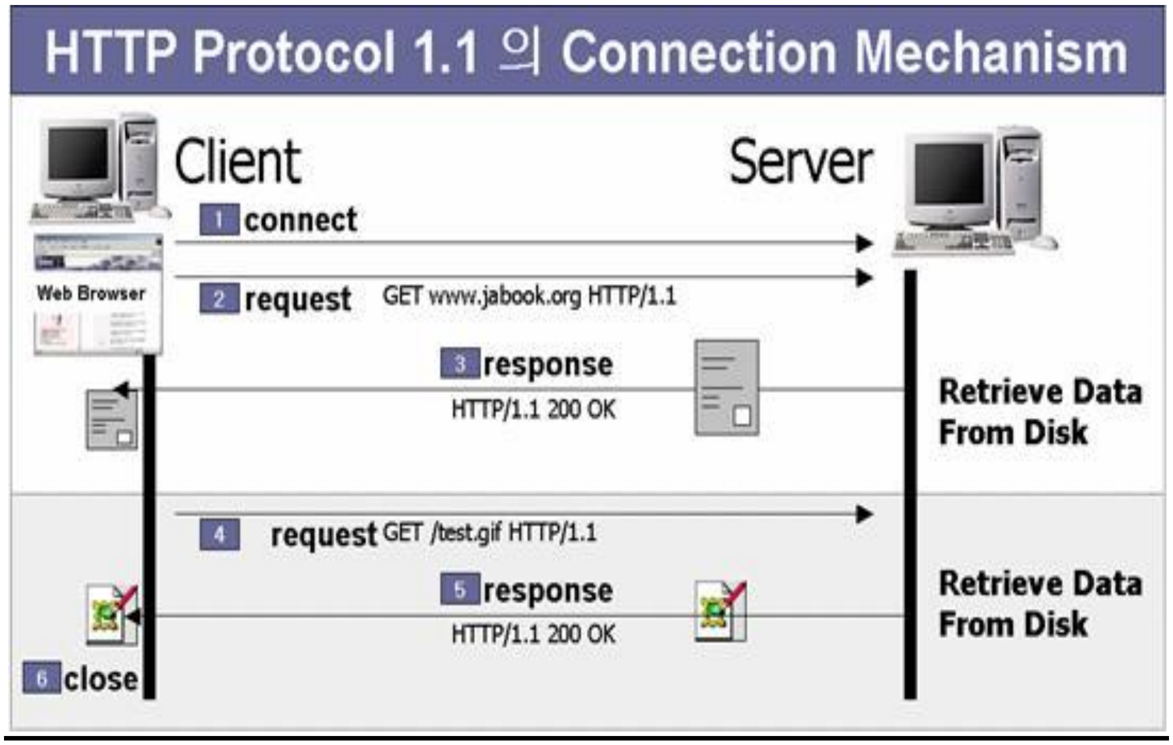

http://phpcubes.com
Request For WebPage with "http://" protocol on port 80
Response as HTML Page
CLIENT
SERVER
HTML PAGE = HTML + JAVASCRIPT + CSS
LINUX as Operating System
APACHE as WebServer
MYSQL as Database Server
PHP as Server Side Scripting Language

HTTP/1.0

The skeleton of functionality that HTTP/0.9 formed the basis for a rapid evolution of HTTP in the early 1990s. As the World Wide Web grew in size and acceptance, many new ideas and features were incorporated into HTTP. The result of a great deal of development effort was the formalization of the first HTTP standard: version 1.0. This much enhanced HTTP was published in May 1996 as RFC 1945, Hypertext Transfer Protocol — HTTP/1.0. It had been in use for several years prior to that formal publication date, however.

HTTP/1.0 transformed HTTP from a trivial request/response application to a true messaging protocol. It described a complete message format for HTTP, and explained how it should be used for client requests and server responses. One of the most important changes in HTTP/1.0 was the generalization of the protocol to handle many types of different media, as opposed to strictly hypertext documents. This was done by borrowing concepts and header constructs from the Multipurpose Internet Mail Extensions (MIME) standard defined for e-mail. At the same time that it defined much more capable Web servers and clients, HTTP/1.0 retained backwards compatibility with servers and clients still using HTTP/0.9.

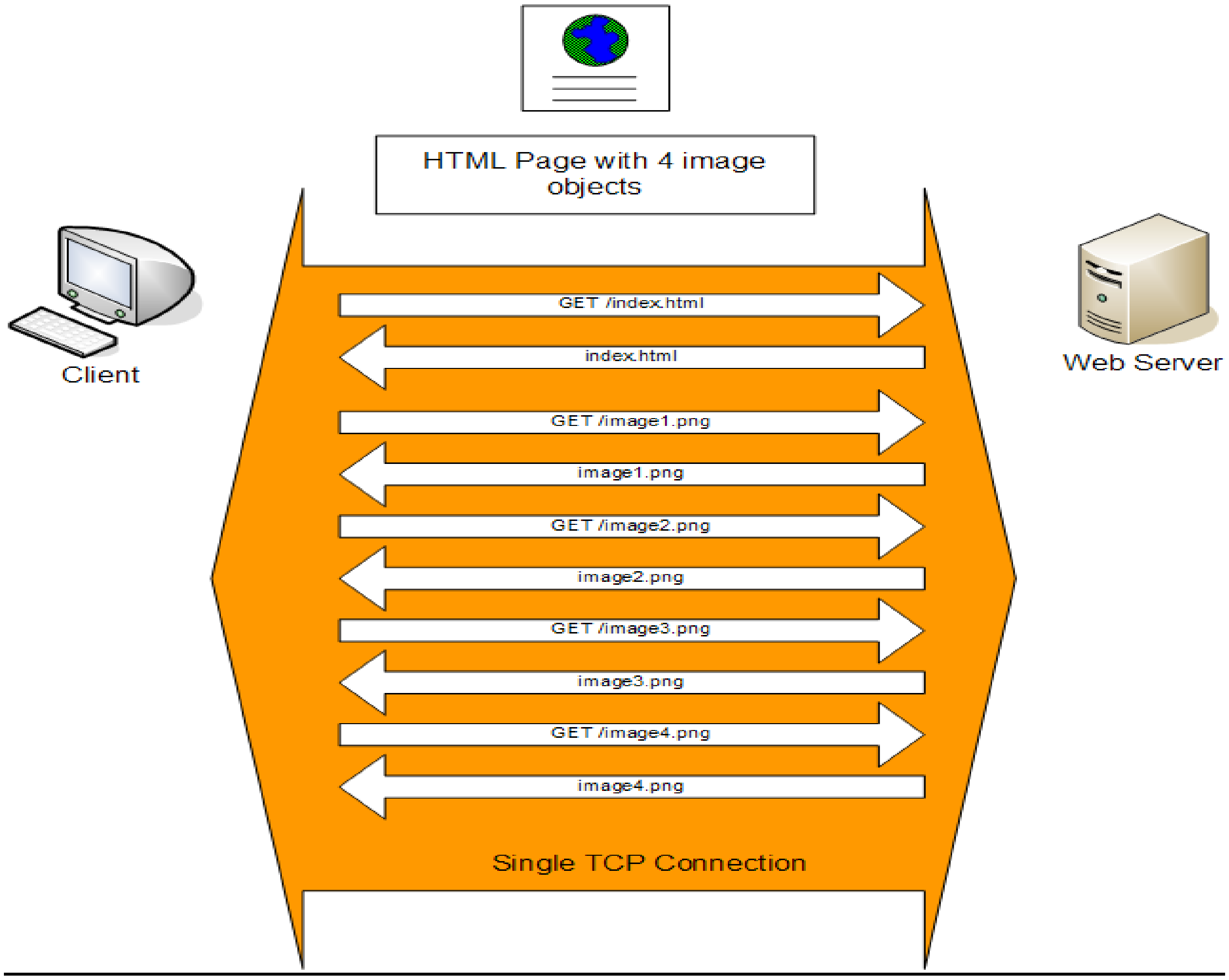
HTTP/1.0 was the version of HTTP that was widely implemented in the mid-1990s as the Web exploded in popularity. After only a couple of years, HTTP accounted for the majority of the traffic on the burgeoning Internet. The popularity of HTTP was in fact so great that it single-handedly prompted the installation of a lot of new hardware to handle the load of browser requests and Web server replies.


HTTP Protocol 1.1 의 Connection Mechanism
Client
Server
Web Browser
1 connect
2 request GET www.jabook.org HTTP/1.1
3 response HTTP/1.1 200 OK
Retrieve Data From Disk
4 request GET /test.gif HTTP/1.1
5 response HTTP/1.1 200 OK
Retrieve Data From Disk
6 close

SERVER RESPONSE TO REQUEST

There are 5 different classes of request for the web server. They are listed below:-

- Informational (1XX)
- Success (2XX)
- Redirection (3XX)
- Client Error (4XX)
- Server Error (5XX)



INFORMATIONAL: - This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. This request is 1xx. Since HTTP/1.0 did not define any 1xx status codes, servers must not send a 1xx response to an HTTP/1.0 client except under experimental conditions. The informational requests are:

- 100 CONTINUE
  This means that the server has received the request headers, and that the client should proceed to send the request body. If the request body is large, sending it to a server when a request has already been rejected based upon inappropriate headers is inefficient. To have a server check if the request could be accepted based on the request's headers alone, a client must send Expect: 100-continue as a header in its initial request and check if a 100 Continue status code is received in response before continuing (or receive 417 Expectation Failed and not continue).

- 101 SWITCHING PROTOCOLS
  This means the requester has asked the server to switch protocols and the server is acknowledging that it will do so.

- 102 PROCESSING
  As a WebDAV request may contain many sub-requests involving file operations, it may take a long time to complete the request. This code indicates that the server has received and is processing the request, but no response is available yet. This prevents the client from timing out and assuming the request was lost.

SUCCESS: -

This class of status codes indicates the action requested by the client was received, understood, accepted and processed successfully. This request is shown like 2xx. The request under this section are:

- 200 OK
  Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.

- 201 CREATED
  The request has been fulfilled and resulted in a new resource being created.

- 202 ACCEPTED
  The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.

- 203 NON-AUTHORITATIVE INFORMATION (SINCE HTTP/1.1)
  The server successfully processed the request, but is returning information that may be from another source.

- 204 NO CONTENT
  The server successfully processed the request, but is not returning any content. Usually used as a response to a successful delete request.

- 205 RESET CONTENT
  The server successfully processed the request, but is not returning any content. Unlike a 204 response, this response requires that the requester reset the document view.

- 206 PARTIAL CONTENT
  The server is delivering only part of the resource (byte serving) due to a range header sent by the client. The range header is used by tools like Wget to enable resuming of interrupted downloads, or split a download into multiple simultaneous streams.

- 207 MULTI-STATUS (WEBDAV; RFC 4918)
  The message body that follows is an XML message and can contain a number of separate response codes, depending on how many sub-requests were made.

- 208 ALREADY REPORTED (WEBDAV; RFC 5842)
  The members of a DAV binding have already been enumerated in a previous reply to this request, and are not being included again.

- 226 IM USED (RFC 3229)
  The server has fulfilled a request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

REDIRECTION: - This class of status code indicates the client must take additional action to complete the request. Many of these status codes are used in URL redirection. A user agent may carry out the additional action with no user interaction only if the method used in the second request is GET or HEAD. A user agent should not automatically redirect a request more than five times, since such redirections usually indicate an infinite loop.

- 300 MULTIPLE CHOICES
  Indicates multiple options for the resource that the client may follow. It, for instance, could be used to present different format options for video, list files with different extensions, or word sense disambiguation.

- 301 MOVED PERMANENTLY
  This and all future requests should be directed to the given URI.

- 302 FOUND
  This is an example of industry practice contradicting the standard. The HTTP/1.0 specification (RFC 1945) required the client to perform a temporary redirect (the original describing phrase was "Moved Temporarily"), but popular browsers implemented 302 with the functionality of a 303 See Other. Therefore, HTTP/1.1 added status codes 303 and 307 to distinguish between the two behaviors. However, some Web applications and frameworks use the 302 status code as if it were the 303.

- 303 SEE OTHER (since HTTP/1.1)
  The response to the request can be found under another URI using a GET method. When received in response to a POST (or PUT/DELETE), it should be assumed that the server has received the data and the redirect should be issued with a separate GET message.

- 304 NOT MODIFIED
  Indicates that the resource has not been modified since the version specified by the request headers If-Modified-Since or If-None-Match. This means that there is no need to retransmit the resource, since the client still has a previously-downloaded copy.

- 305 USE PROXY (SINCE HTTP/1.1)
  The requested resource is only available through a proxy, whose address is provided in the response. Many HTTP clients do not correctly handle responses with this status code, primarily for security reasons.

- 306 SWITCH PROXY
  No longer used. Originally meant "Subsequent requests should use the specified proxy."

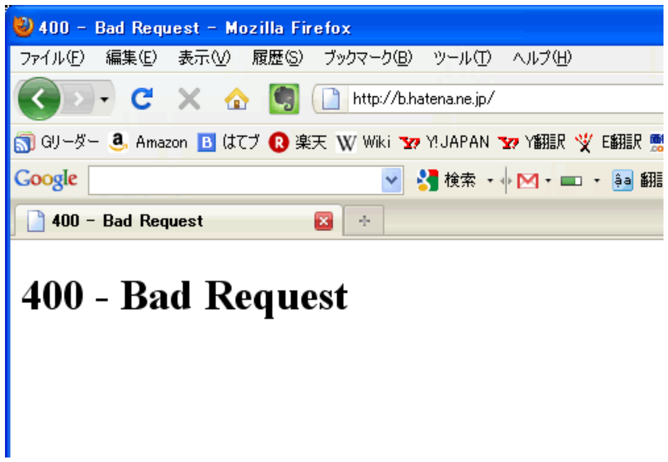
- 307 TEMPORARY REDIRECT (SINCE HTTP/1.1)
  In this case, the request should be repeated with another URI; however, future requests should still use the original URI. In contrast to how 302 was historically implemented, the request method is not allowed to be changed when reissuing the original request. For instance, a POST request should be repeated using another POST request.

- 308 PERMANENT REDIRECT (EXPERIMENTAL RFC; RFC 7238)
  The request, and all future requests should be repeated using another URI. 307 and 308 (as proposed) parallel the behaviors of 302 and 301, but *do not allow the HTTP method to change*. So, for example, submitting a form to a permanently redirected resource may continue smoothly.

CLIENT ERROR:-

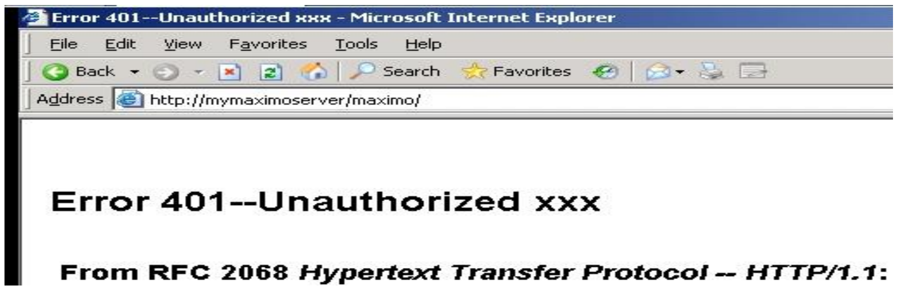
The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents should display any included entity to the user.

- 400 BAD REQUEST
  The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).


400 - Bad Request - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)
http://b.hatena.ne.jp/
400 - Bad Request

400 - Bad Request

- 401 UNAUTHORIZED
  Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.


Error 401--Unauthorized xxx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://mymaximoserver/maximo/

Error 401--Unauthorized xxx

From RFC 2068 *Hypertext Transfer Protocol -- HTTP/1.1:*

- 402 PAYMENT REQUIRED
  Reserved for future use. The original intention was that this code might be used as part of some form of digital cash or micropayment scheme, but that has not happened, and this code is not usually used.

- 403 FORBIDDEN
  The request was a valid request, but the server is refusing to respond to it. Unlike a 401 unauthorized response, authenticating will make no difference.

- 404 NOT FOUND
  The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.

- 405 METHOD NOT ALLOWED
  A request was made of a resource using a request method not supported by that resource; for example, using GET on a form which requires data to be presented via POST, or using PUT on a read-only resource.

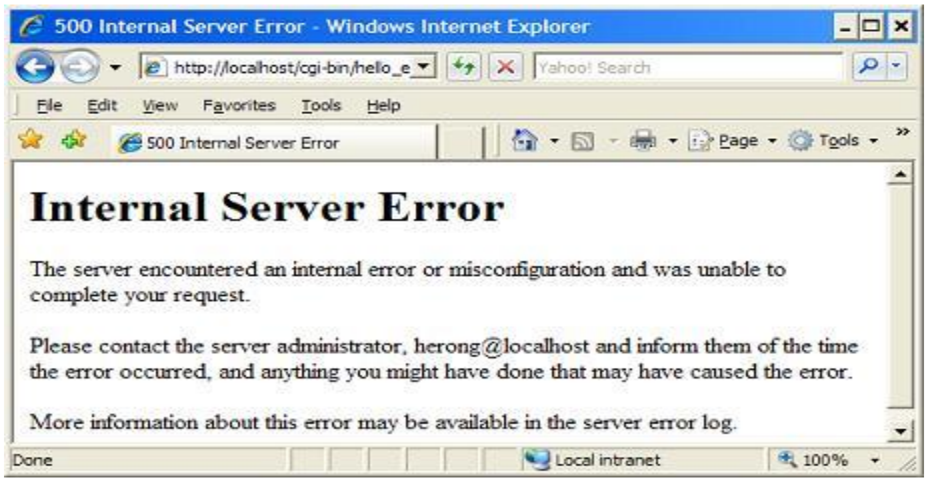
- 406 NOT ACCEPTABLE
  The requested resource is only capable of generating content not acceptable according to the Accept headers sent in the request.

- 407 PROXY AUTHENTICATION REQUIRED
  The client must first authenticate itself with the proxy.

- 408 REQUEST TIMEOUT
  The server timed out waiting for the request. According to HTTP specifications: "The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time."

- 409 CONFLICT
  Indicates that the request could not be processed because of conflict in the request, such as an edit conflict in the case of multiple updates.

SERVER ERROR: -The server failed to fulfill an apparently valid request. Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has encountered an error or is otherwise incapable of performing the request. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and indicate whether it is a temporary or permanent condition. Likewise, user agents should display any included entity to the user. These response codes are applicable to any request method. Few requests are:


500 Internal Server Error - Windows Internet Explorer
http://localhost/cgi-bin/hello_e
Yahoo! Search
File Edit View Favorites Tools Help
500 Internal Server Error
Page Tools
Done
Local intranet
100%

Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.

Please contact the server administrator, herong@localhost and inform them of the time the error occurred, and anything you might have done that may have caused the error.

More information about this error may be available in the server error log.

- 500 INTERNAL SERVER ERROR
  A generic error message, given when an unexpected condition was encountered and no more specific message is suitable.

- 501 NOT IMPLEMENTED
  The server either does not recognize the request method, or it lacks the ability to fulfil the request. Usually this implies future availability (e.g., a new feature of a web-service API).

- 502 BAD GATEWAY
  The server was acting as a gateway or proxy and received an invalid response from the upstream server.
- 503 SERVICE UNAVAILABLE
  The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state.

- 504 GATEWAY TIMEOUT
  The server was acting as a gateway or proxy and did not receive a timely response from the upstream server.

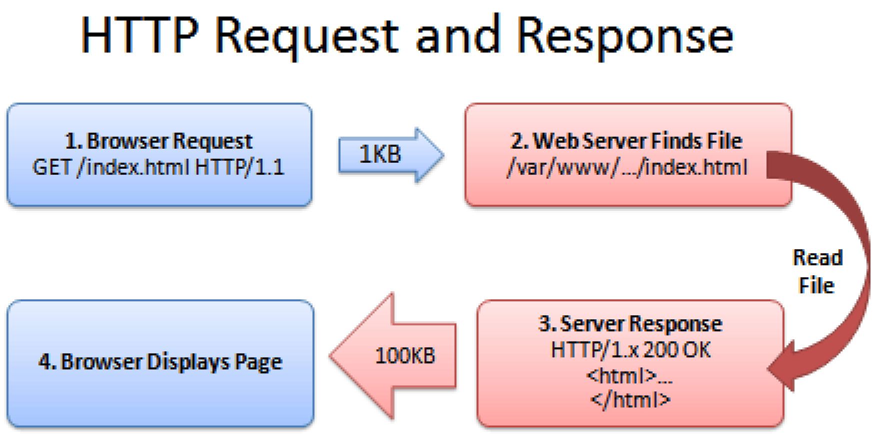
- 505 HTTP VERSION NOT SUPPORTED
  The server does not support the HTTP protocol version used in the request

OUR UNDERSTANDING OF THE PROJECT:

A server is a running instance of an application (software) capable of accepting requests from the client and giving responses accordingly. Servers can run on any computer including dedicated computers, which individually are also often referred to as "the server". In many cases, a computer can provide several services and have several servers running. The advantage of running servers on a dedicated computer is security. For this reason most of the servers are daemon processes and designed in that they can be run on specific computer(s).

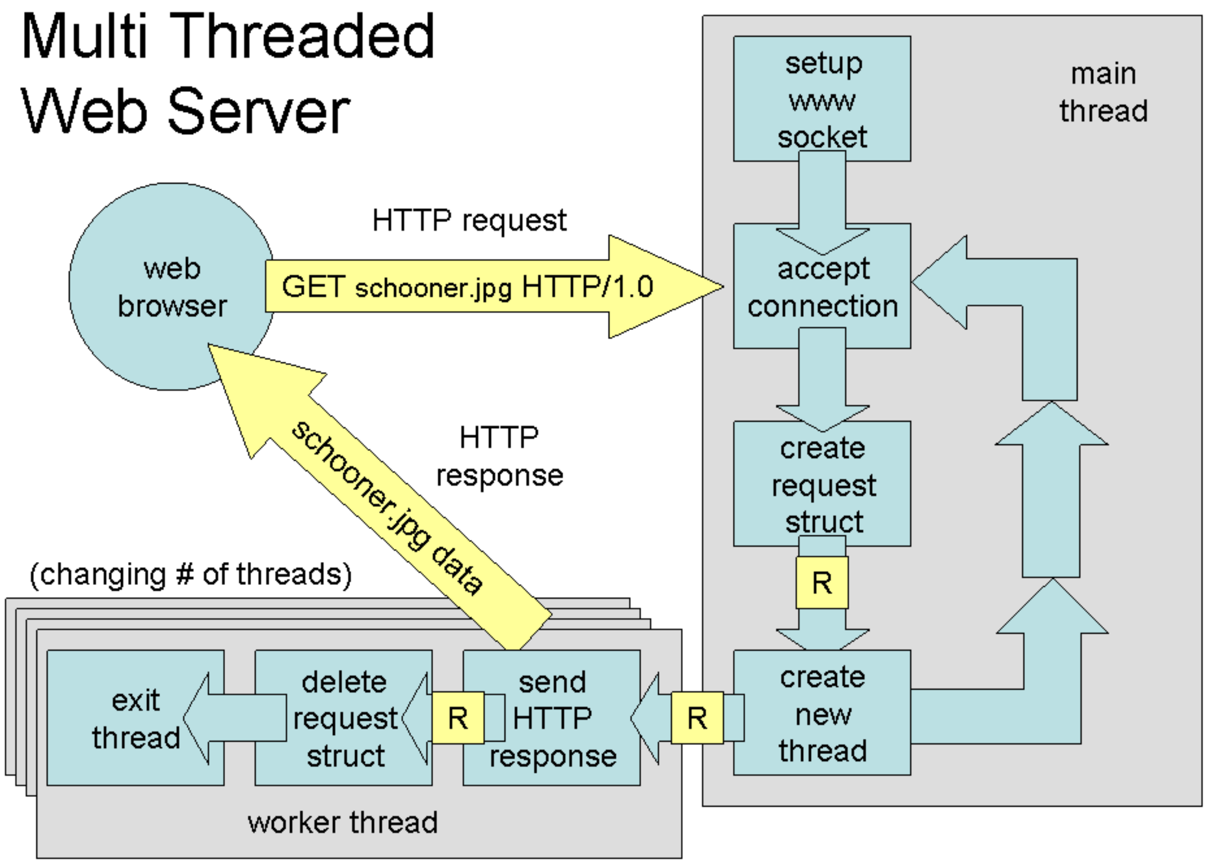
Servers operate within a client-server architecture. Servers are computer programs running to serve the requests of other programs, the clients. Thus, the server performs some tasks on behalf of clients. It facilitates the clients to share data, information or any hardware and software resources. The clients typically connect to the server through the network but may run on the same computer. In the context of Internet Protocol (IP) networking, a server is a program that operates as a socket listener.

Servers often provide essential services across a network, either to private users inside a large organization or to public users via the Internet. Typical computing servers are database server, file server, mail server, print server, web server, gaming server, and application server.



Multi client web server:

We provide you with a multi-threaded client that sends a single HTTP request to the server and prints out the results. This multi threaded client web server prints out the entity headers with the statistics that you added, so that you can verify the server is ordering requests as expected. The server enough with multiple simultaneous requests to ensure that the server is correctly scheduling or synchronizing threads.



STRUCTURE OF SOURCE CODE:

// import packages/ Imports Java libraries //

import java.io.* ;

import java.net.* ;

import java.util.* ;

/** Beginnning of class Server **/

public final class WebServer2

/** Main method **/

{

public static void main(String argv[]) throws Exception

{

// Set the port number//

int port = 9999;

// Establish the listen socket//

ServerSocket welcomeSocket = new ServerSocket(port);

// Process HTTP service requests in an infinite loop//

while (true) {

// Listen for a TCP connection request//

Socket connectionSocket = welcomeSocket.accept();

```
**// Construct an object to process the HTTP request message//**

HttpRequest request = new HttpRequest( connectionSocket );


**// Create a new thread to process the request//**

Thread thread = new Thread(request);


**// Start the thread//**

thread.start();

}}}

final class HttpRequest implements Runnable

{

final static String CRLF = "\r\n";//returning carriage return (CR) and a line feed (LF)

Socket socket;


**// Constructor//**

 public HttpRequest(Socket socket) throws Exception

 {

  this.socket = socket;

 }
```

<u>// Implement the run() method of the Runnable interface.</u>**

<u>Within run(), we explicitly catch and handle exceptions with a try/catch block.//</u>**

```
public void run()

{

try {processRequest();}catch (Exception e) {System.out.println(e);}

}


private void processRequest() throws Exception

{
```

<u>// Get a reference to the socket's input and output streams//**</u>**

```
InputStream instream = socket.getInputStream();

DataOutputStream os = new DataOutputStream(socket.getOutputStream());
```

<u>// Set up input stream filters.//**</u>**

```
BufferedReader br = new BufferedReader(new InputStreamReader(instream));//reads the input data
```

<u>// Get the request line of the HTTP request message//**</u>**

```
String requestLine = br.readLine();// get /path/file.html version of http
```

<u>// Display the request line//**</u>**

```
System.out.println();

System.out.println(requestLine);
```

__**// HERE WE NEED TO DEAL WITH THE REQUEST.Extract the filename from the request line.**__

__**this is a input method with deliminators//**__

```
StringTokenizer tokens = new StringTokenizer(requestLine);
tokens.nextToken();
String fileName = tokens.nextToken();
```

__**// Prepend a "." so that file request is within the current directory.//**__

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

__**//Open the requested file.//**__

```
FileInputStream fis = null;
boolean fileExists = true;
try {
 fis = new FileInputStream(fileName);
} catch (FileNotFoundException e) {
 fileExists = false;
}
```

__**//Construct the response message.//**__

```
String statusLine = null;
String contentTypeLine = null;
String entityBody = null;
if (fileExists) {
 statusLine = "HTTP/1.0 200 OK" + CRLF; //common success message
 contentTypeLine = "Content-type: " + contentType( fileName ) + CRLF;}
```

<u>//common error message, content info//**</u>**

```
else {
 statusLine = "HTTP/1.0 404 Not Found" + CRLF;
 contentTypeLine = "Content-type: " + "text/html" + CRLF;
 entityBody = "<HTML>" +
  "<HEAD><TITLE>Not Found</TITLE></HEAD>" +
  "<BODY>Not Found</BODY></HTML>";
}
```

<u>//Send the status line.//**</u>**

```
os.writeBytes(statusLine);
```

<u>//Send the content type line//**</u>**

```
os.writeBytes(contentTypeLine);
```

<u>//Send a blank line to indicate the end of the header lines.//**</u>**

```
os.writeBytes(CRLF);
```

<u>//Send the entity body//**</u>**

```
if (fileExists) {
 sendBytes(fis, os);
 os.writeBytes(statusLine);//Send the status line.
os.writeBytes(contentTypeLine);//Send the content type line.
```

```
 fis.close();

} else {

 os.writeBytes(statusLine);         //Send the status line.

 os.writeBytes(entityBody);         //Send the an html error message info body.

 os.writeBytes(contentTypeLine);            //Send the content type line.

}




System.out.println("*****");

System.out.println(fileName);    //print out file request to console

System.out.println("*****");
```

<u>****// Get and display the header lines//****</u>

```
String headerLine = null;

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

System.out.println(headerLine);

}


//code from part 1

// Right here feed the client something

//os.writeBytes("<html><body><h1>My First Heading</h1>");

//os.writeBytes(  "<p>My first paragraph.</p></body></html> ");

//os.flush();
```

****// Close streams and socket.//****

```
os.close();

br.close();

socket.close();
```

****//return the file types//****

```
private static String contentType(String fileName)

{

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

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

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

 return "application/octet-stream";

}
```

****//set up input output streams//****

```
private static void sendBytes(FileInputStream fis, OutputStream os) throws Exception

{
```

**** // Construct a 1K buffer to hold bytes on their way to the socket.//****

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

   int bytes = 0;
```

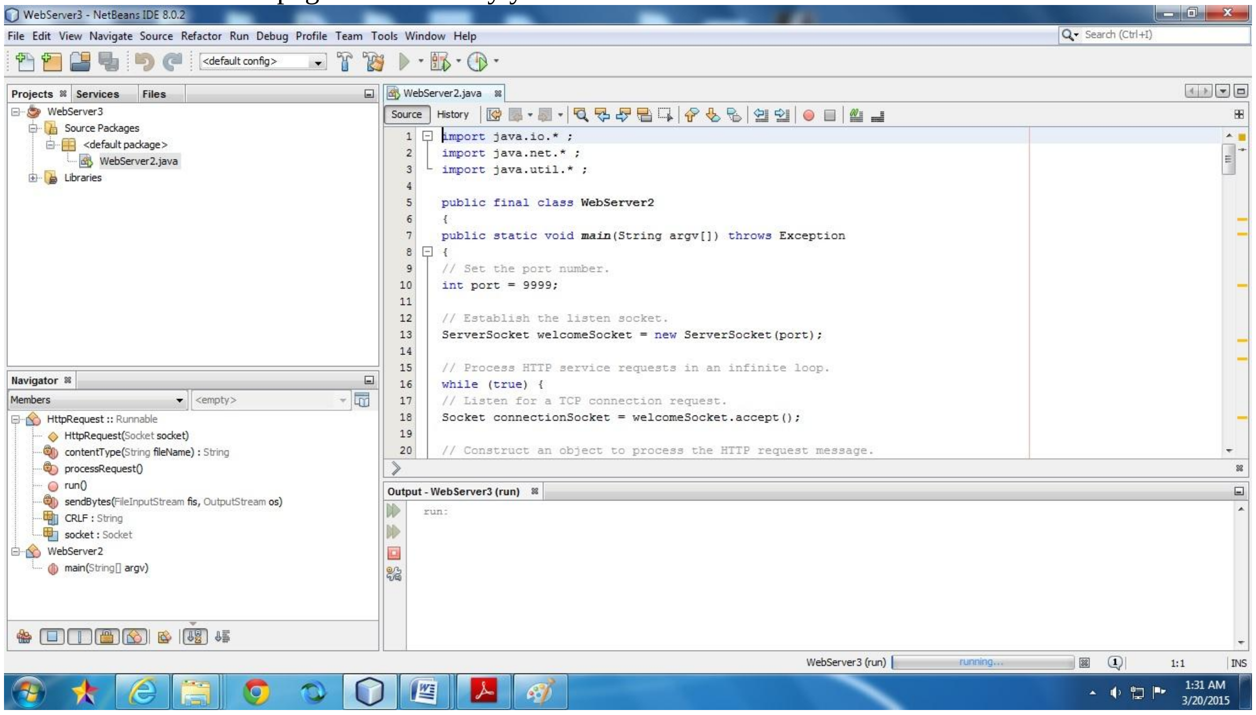
<u> // Copy requested file into the socket's output stream.//**</u>**

```
  while((bytes = fis.read(buffer)) != -1 )// read() returns minus one, indicating that the end of the file
  {
    os.write(buffer, 0, bytes);
  }}
}
```

Test Results:

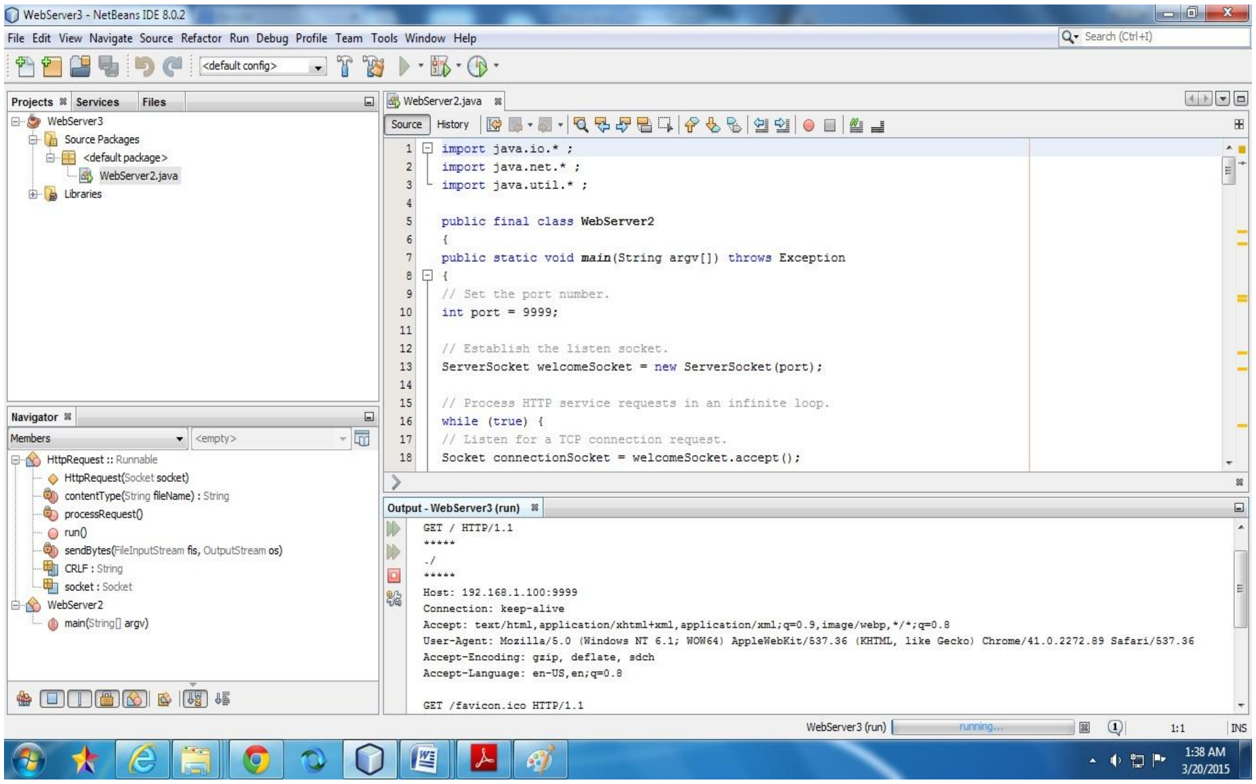
Following are the test results and screenshots of our given task.

1. Your test homepage distributed by your Java-based Web Server.



```
import java.io.* ;
import java.net.* ;
import java.util.* ;

public final class WebServer2
{
public static void main(String argv[]) throws Exception
{
// Set the port number.
int port = 9999;

// Establish the listen socket.
ServerSocket welcomeSocket = new ServerSocket(port);

// Process HTTP service requests in an infinite loop.
while (true) {
// Listen for a TCP connection request.
Socket connectionSocket = welcomeSocket.accept();

// Construct an object to process the HTTP request message.
```

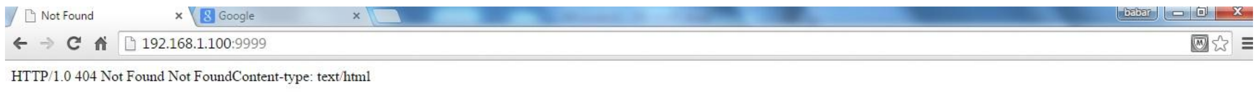
```
run:
```



```
import java.io.* ;
import java.net.* ;
import java.util.* ;

public final class WebServer2
{
public static void main(String argv[]) throws Exception
{
// Set the port number.
int port = 9999;

// Establish the listen socket.
ServerSocket welcomeSocket = new ServerSocket(port);

// Process HTTP service requests in an infinite loop.
while (true) {
// Listen for a TCP connection request.
Socket connectionSocket = welcomeSocket.accept();
```

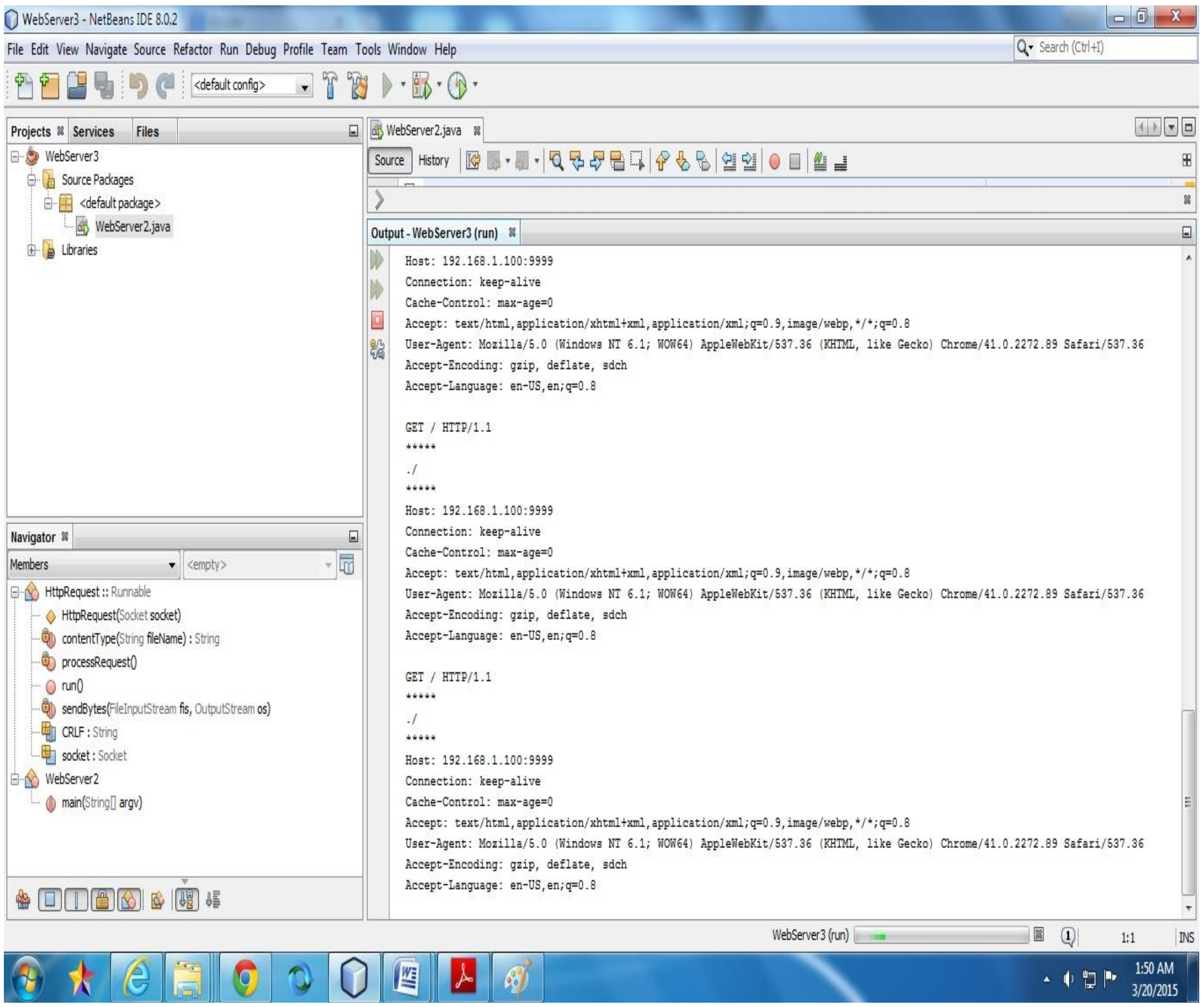
```
GET / HTTP/1.1
*****
./
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

GET /favicon.ico HTTP/1.1
```

2.The message displayed when the requested homepage is not available.
The HTTP request messages displayed by the server during the transmission of your homepage.



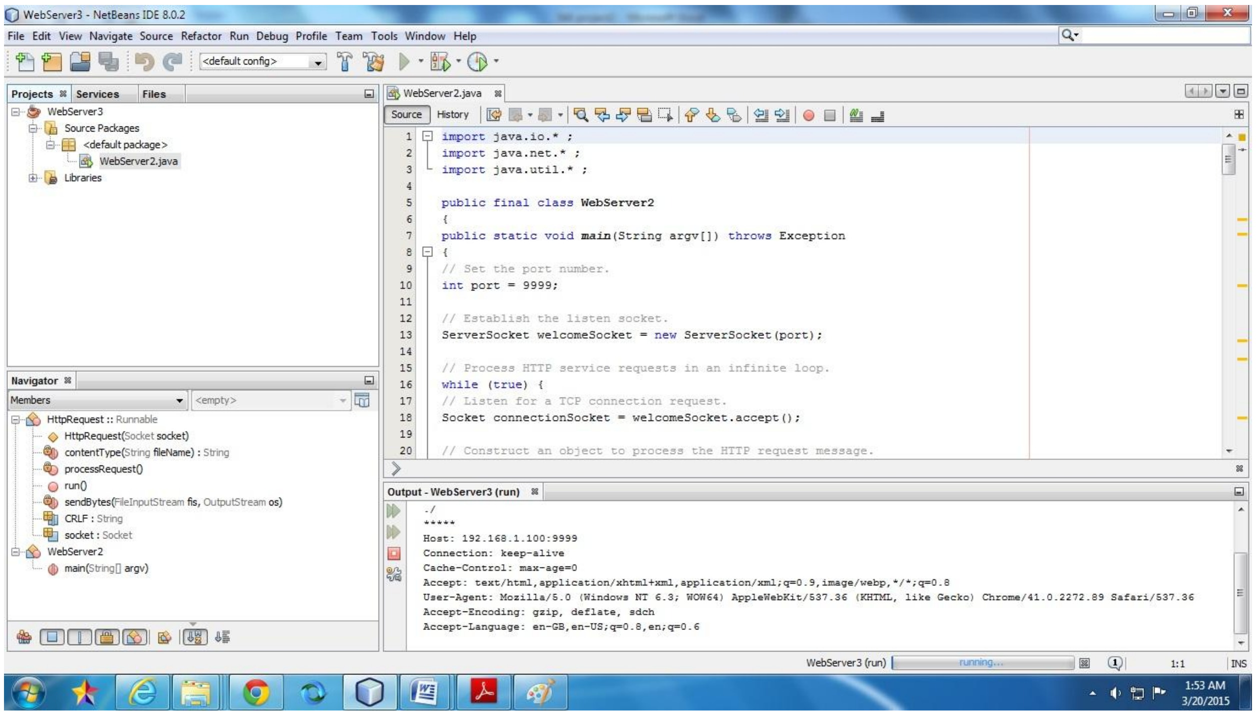
```
HTTP/1.0 404 Not Found Not FoundContent-type: text/html
```

2. When there is no webpage existing, when multiple users access the same webpage.



```
Host: 192.168.1.100:9999
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

GET / HTTP/1.1
*****
./
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

GET / HTTP/1.1
*****
./
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

3.Run your web server on one of the campus computer, and run your client (IE) on another campus computer .

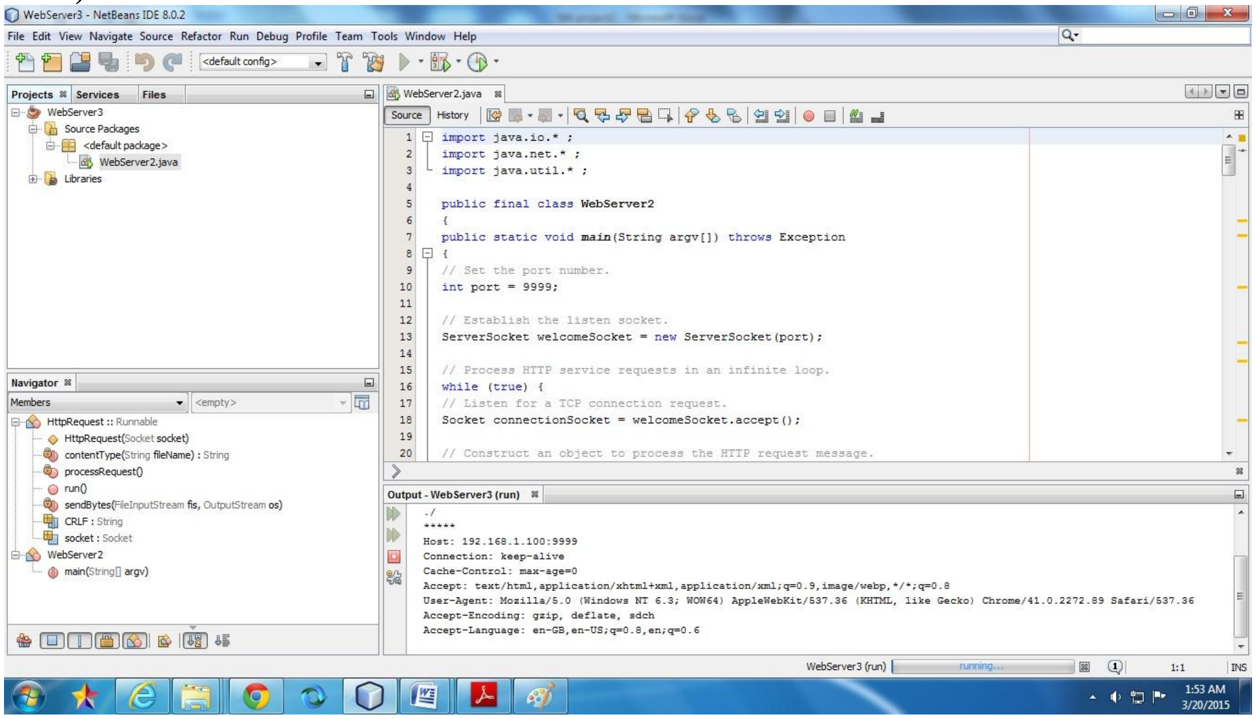


```java
import java.io.* ;
import java.net.* ;
import java.util.* ;

public final class WebServer2
{
public static void main(String argv[]) throws Exception
{
// Set the port number.
int port = 9999;

// Establish the listen socket.
ServerSocket welcomeSocket = new ServerSocket(port);

// Process HTTP service requests in an infinite loop.
while (true) {
// Listen for a TCP connection request.
Socket connectionSocket = welcomeSocket.accept();

// Construct an object to process the HTTP request message.
```

Output - WebServer3 (run)

```
./
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
```

4.Run your web server on one of the campus computer, and run your client (IE) on a laptop (using wireless access).

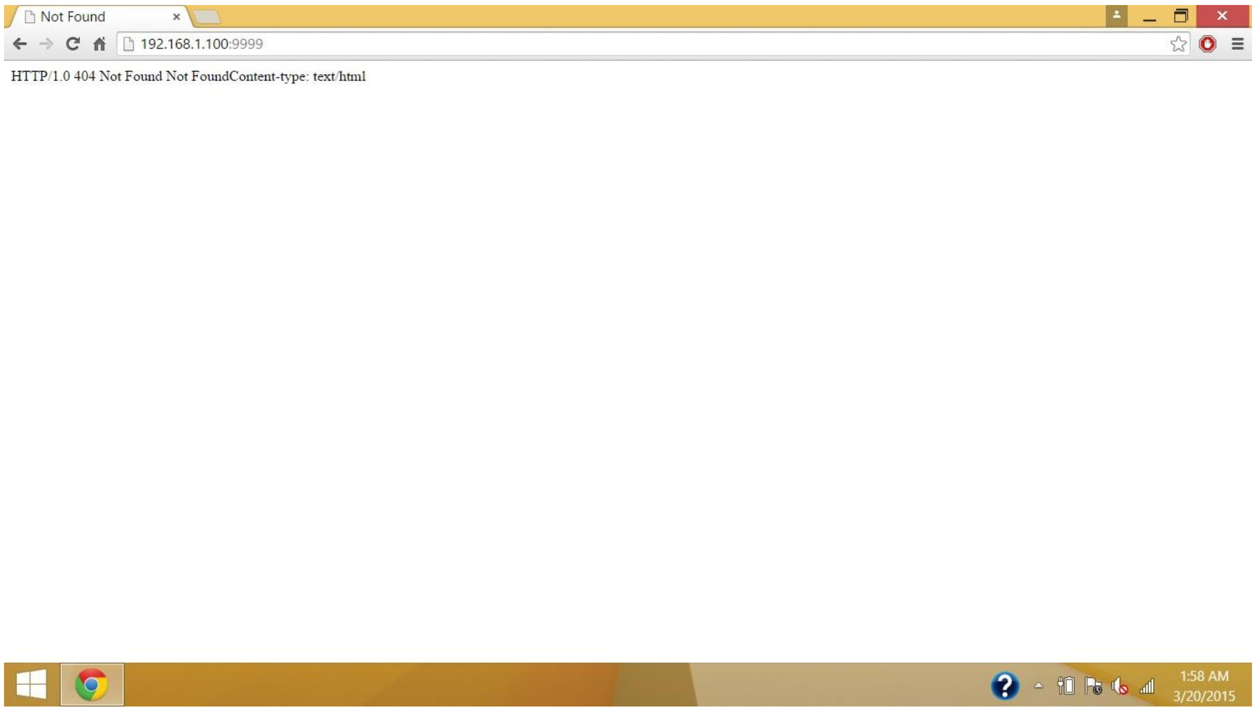


```java
import java.io.* ;
import java.net.* ;
import java.util.* ;

public final class WebServer2
{
public static void main(String argv[]) throws Exception
{
// Set the port number.
int port = 9999;

// Establish the listen socket.
ServerSocket welcomeSocket = new ServerSocket(port);

// Process HTTP service requests in an infinite loop.
while (true) {
// Listen for a TCP connection request.
Socket connectionSocket = welcomeSocket.accept();

// Construct an object to process the HTTP request message.
```

Output - WebServer3 (run)

```
./
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
```

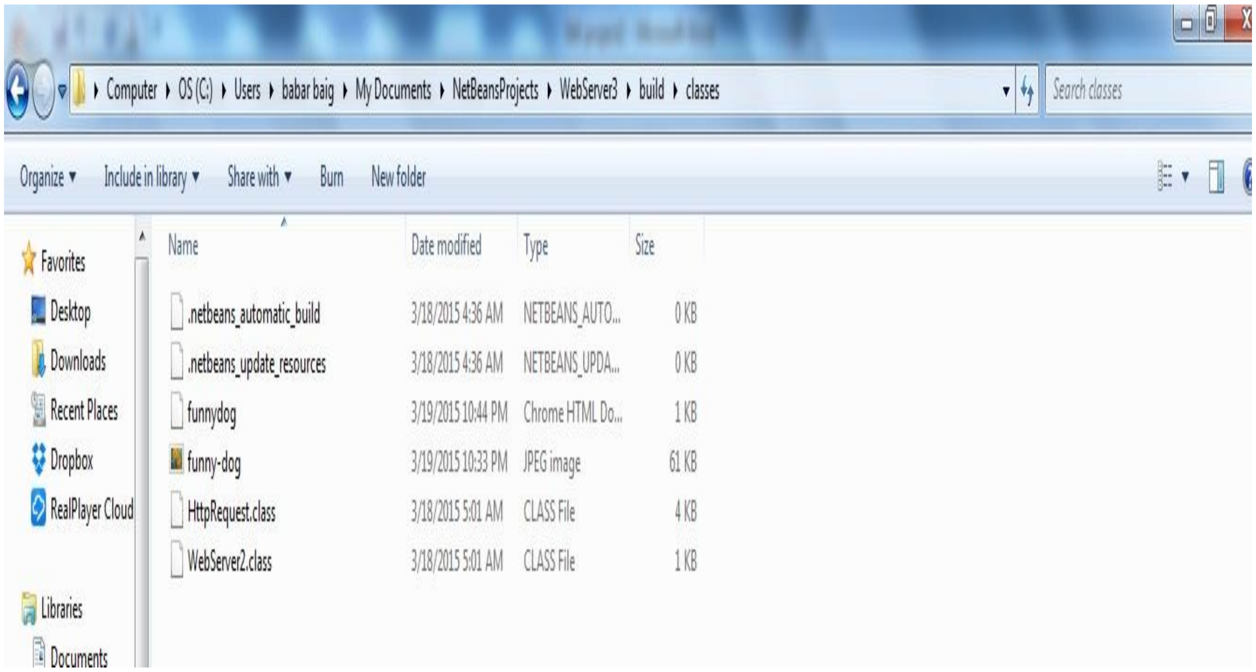
Client on wireless network.



```
HTTP/1.0 404 Not Found Not FoundContent-type: text/html
```

5. Now we gonna implement our final test. We create a small html file to demonstrate that our webpage could also show the jepg pictures. We called our page is “funnydog.html”. we put this html file in our webserver2 folder, we also put the funny-dog.jpeg into the same folder. The path is,

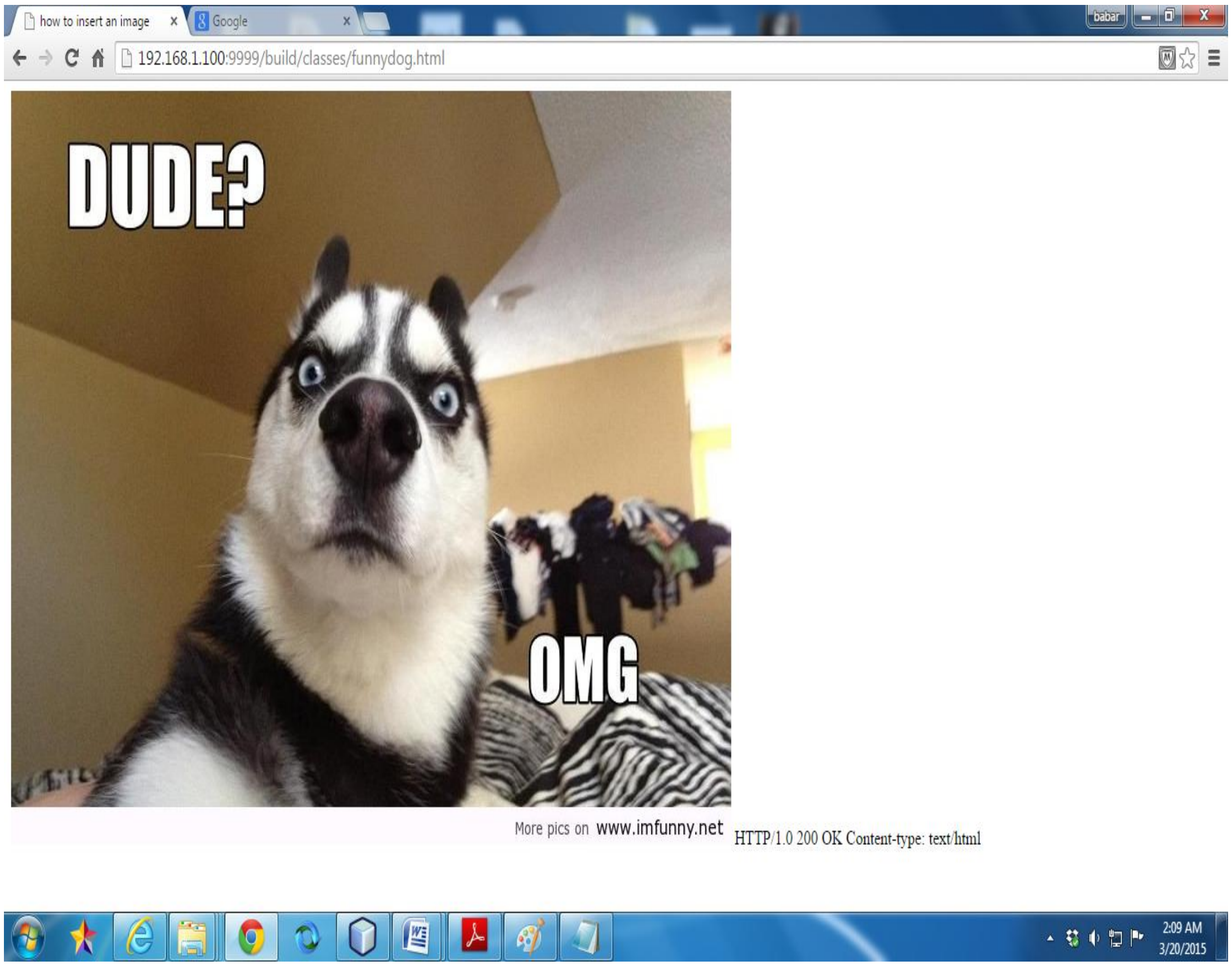
C:\Users\babar baig\Documents\NetBeansProjects\WebServer2\build\classes



Name	Date modified	Type	Size
.netbeans_automatic_build	3/18/2015 4:36 AM	NETBEANS_AUTO...	0 KB
.netbeans_update_resources	3/18/2015 4:36 AM	NETBEANS_UPDA...	0 KB
funnydog	3/19/2015 10:44 PM	Chrome HTML Do...	1 KB
funny-dog	3/19/2015 10:33 PM	JPEG image	61 KB
HttpRequest.class	3/18/2015 5:01 AM	CLASS File	4 KB
WebServer2.class	3/18/2015 5:01 AM	CLASS File	1 KB

Now we gonna start our server and search for our desire webpage. This time we use the link along with the path and html page. The link would be …

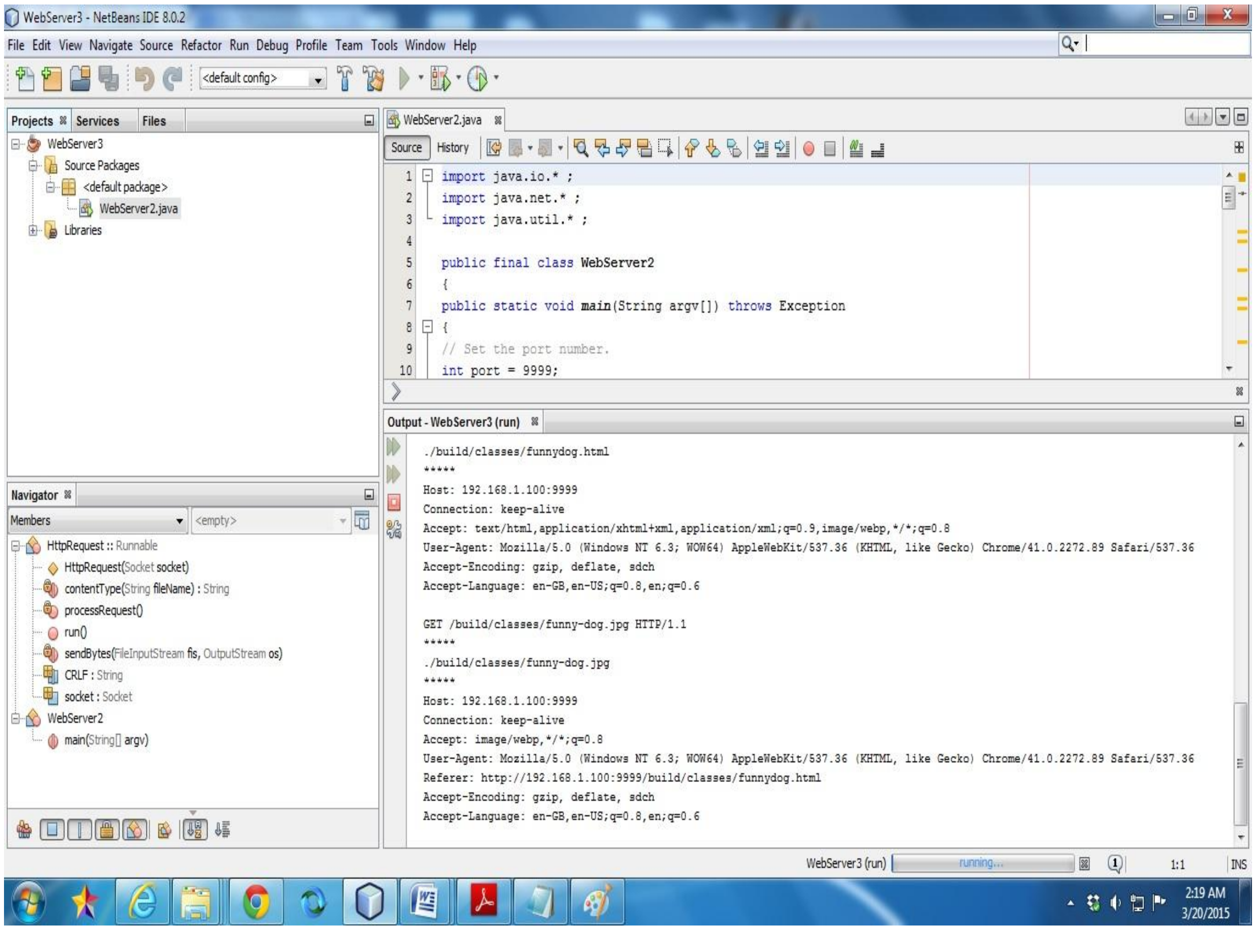
http://192.168.1.100:9999/build/classes/funnydog.html


how to insert an image
Google
babar
192.168.1.100:9999/build/classes/funnydog.html
DUDE?
OMG
More pics on www.imfunny.net
2:09 AM
3/20/2015

HTTP/1.0 200 OK Content-type: text/html

Here is our desire webpage "funnydog.html". We also get the one line statement at bottom right " HTTP/1.0.200 OK ". We also test our server for multi clients by connecting with different computers and laptops and access the same webpage.

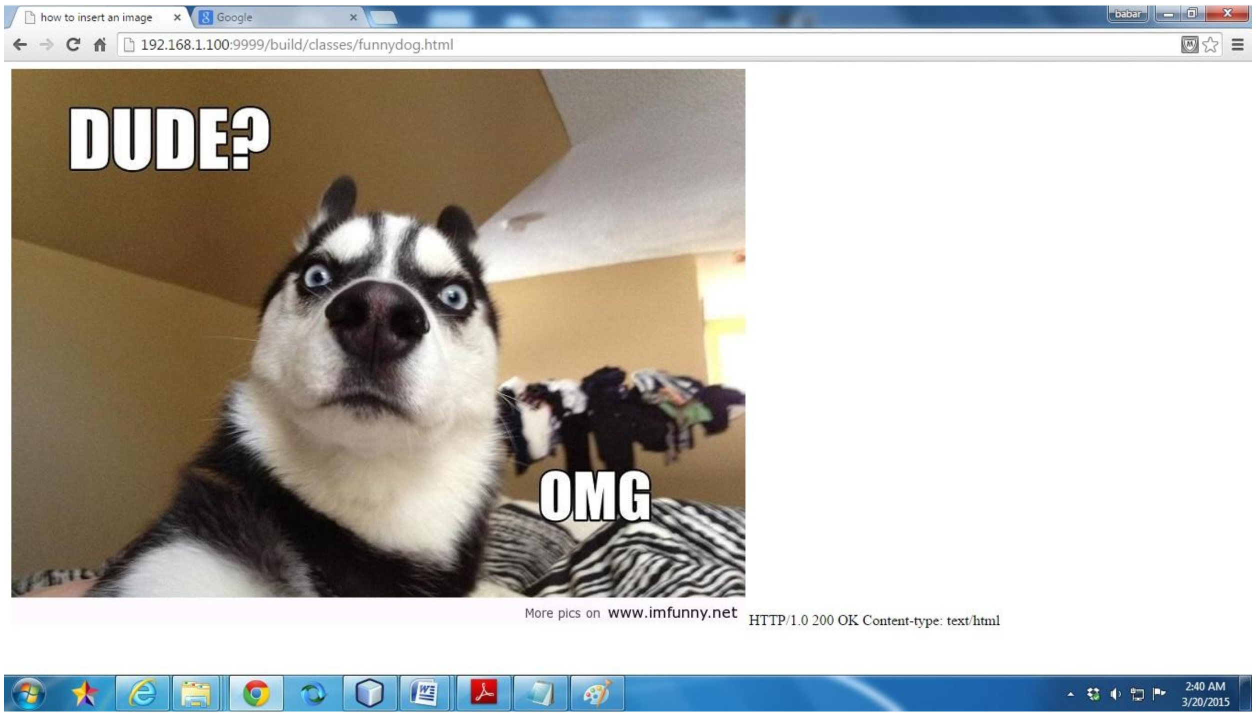
Accessing the same webpage from different laptop both wireless and wired connection. Also shows the incoming on the server.



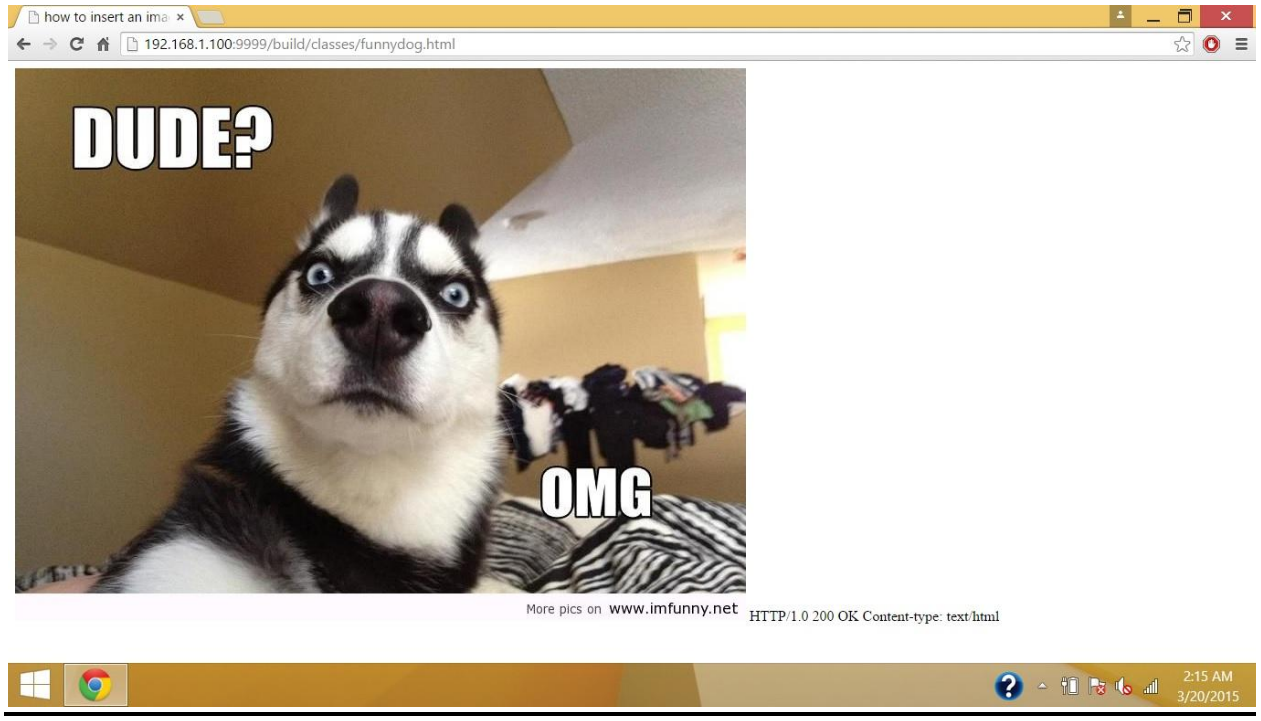
```java
import java.io.* ;
import java.net.* ;
import java.util.* ;

public final class WebServer2
{
public static void main(String argv[]) throws Exception
{
// Set the port number.
int port = 9999;
```

```
./build/classes/funnydog.html
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6

GET /build/classes/funny-dog.jpg HTTP/1.1
*****
./build/classes/funny-dog.jpg
*****
Host: 192.168.1.100:9999
Connection: keep-alive
Accept: image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.89 Safari/537.36
Referer: http://192.168.1.100:9999/build/classes/funnydog.html
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
```

Here we have our final result to access webpage with jpeg html file "funnydog.html" both from wireless, wired from different computers and different clients at the same time.


how to insert an image
Google
babar
192.168.1.100:9999/build/classes/funnydog.html
DUDE?
OMG
More pics on www.imfunny.net
2:40 AM
3/20/2015

HTTP/1.0 200 OK Content-type: text/html


how to insert an ima
192.168.1.100:9999/build/classes/funnydog.html
DUDE?
OMG
More pics on www.imfunny.net
2:15 AM
3/20/2015

HTTP/1.0 200 OK Content-type: text/html

ROLES OF THE TEAM MEMBERS IN THE PROJECT

Team Members	Designing	Coding	Testing	Report
Babar Baig	Lead in designing.	Coding for webserver and html page.	Test the webserver with single and multi client.	Data analysis, writing, designing, Screen shots.
Money Gera	Test the design and recommend upgrades.	Debugging and testing the codes.	Test the webserver.	Writing, charts, diagram.

APPENDIX (source code):

```
import java.io.* ;

import java.net.* ;

import java.util.* ;


public final class WebServer2

{

public static void main(String argv[]) throws Exception

{

// Set the port number.

int port = 9999;


// Establish the listen socket.

ServerSocket welcomeSocket = new ServerSocket(port);


// Process HTTP service requests in an infinite loop.

while (true) {

// Listen for a TCP connection request.

Socket connectionSocket = welcomeSocket.accept();


// Construct an object to process the HTTP request message.

HttpRequest request = new HttpRequest( connectionSocket );


// Create a new thread to process the request.
Thread thread = new Thread(request);


// Start the thread.

thread.start();

}}}




final class HttpRequest implements Runnable

{

final static String CRLF = "\r\n";//returning carriage return (CR) and a line feed (LF)

Socket socket;


// Constructor

public HttpRequest(Socket socket) throws Exception

{

this.socket = socket;

}


// Implement the run() method of the Runnable interface.

//Within run(), we explicitly catch and handle exceptions with a try/catch block.

public void run()

{

try {processRequest();}catch (Exception e) {System.out.println(e);}

}


private void processRequest() throws Exception
```

```
{

// Get a reference to the socket's input and output streams.

InputStream instream = socket.getInputStream();

DataOutputStream os = new DataOutputStream(socket.getOutputStream());


// Set up input stream filters.

// Page 169 10th line down or so...

BufferedReader br = new BufferedReader(new InputStreamReader(instream));//reads the input data


// Get the request line of the HTTP request message.

String requestLine = br.readLine();// get /path/file.html version of http


// Display the request line.

System.out.println();

System.out.println(requestLine);

// HERE WE NEED TO DEAL WITH THE REQUEST

// Extract the filename from the request line.

StringTokenizer tokens = new StringTokenizer(requestLine);// this is a input method with deliminators

tokens.nextToken(); // skip over the method, which should be "GET"

String fileName = tokens.nextToken();


// Prepend a "." so that file request is within the current directory.

fileName = "." + fileName;


//Open the requested file.
```

```
FileInputStream fis = null;

boolean fileExists = true;

try {

fis = new FileInputStream(fileName);

} catch (FileNotFoundException e) {

fileExists = false;

}


//Construct the response message.

String statusLine = null;

String contentTypeLine = null;

String entityBody = null;


if (fileExists) {

statusLine = "HTTP/1.0 200 OK" + CRLF; //common success message

contentTypeLine = "Content-type: " + contentType( fileName ) + CRLF;}//content info


else {

statusLine = "HTTP/1.0 404 Not Found" + CRLF;//common error message

contentTypeLine = "Content-type: " + "text/html" + CRLF;//content info

entityBody = "<HTML>" +

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

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

}
```

```
//Send the status line.

os.writeBytes(statusLine);



//Send the content type line.

os.writeBytes(contentTypeLine);



//Send a blank line to indicate the end of the header lines.

os.writeBytes(CRLF);









//Send the entity body.

if (fileExists) {

sendBytes(fis, os);

os.writeBytes(statusLine);//Send the status line.

os.writeBytes(contentTypeLine);//Send the content type line.

fis.close();

} else {

os.writeBytes(statusLine);//Send the status line.

os.writeBytes(entityBody);//Send the an html error message info body.

os.writeBytes(contentTypeLine);//Send the content type line.

}




System.out.println("*****");

System.out.println(fileName);//print out file request to console
System.out.println("*****");

// Get and display the header lines.

String headerLine = null;

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

System.out.println(headerLine);

}

// Close streams and socket.

os.close();

br.close();

socket.close();





}

//return the file types

private static String contentType(String fileName)

{

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

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

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

return "application/octet-stream";

}



//set up input output streams

private static void sendBytes(FileInputStream fis, OutputStream os) throws Exception
```

```
{
// Construct a 1K buffer to hold bytes on their way to the socket.
byte[] buffer = new byte[1024];
int bytes = 0;

// Copy requested file into the socket's output stream.
while((bytes = fis.read(buffer)) != -1 )// read() returns minus one, indicating that the end of the file
{
os.write(buffer, 0, bytes);
}}
}
```