Milestone Three Document

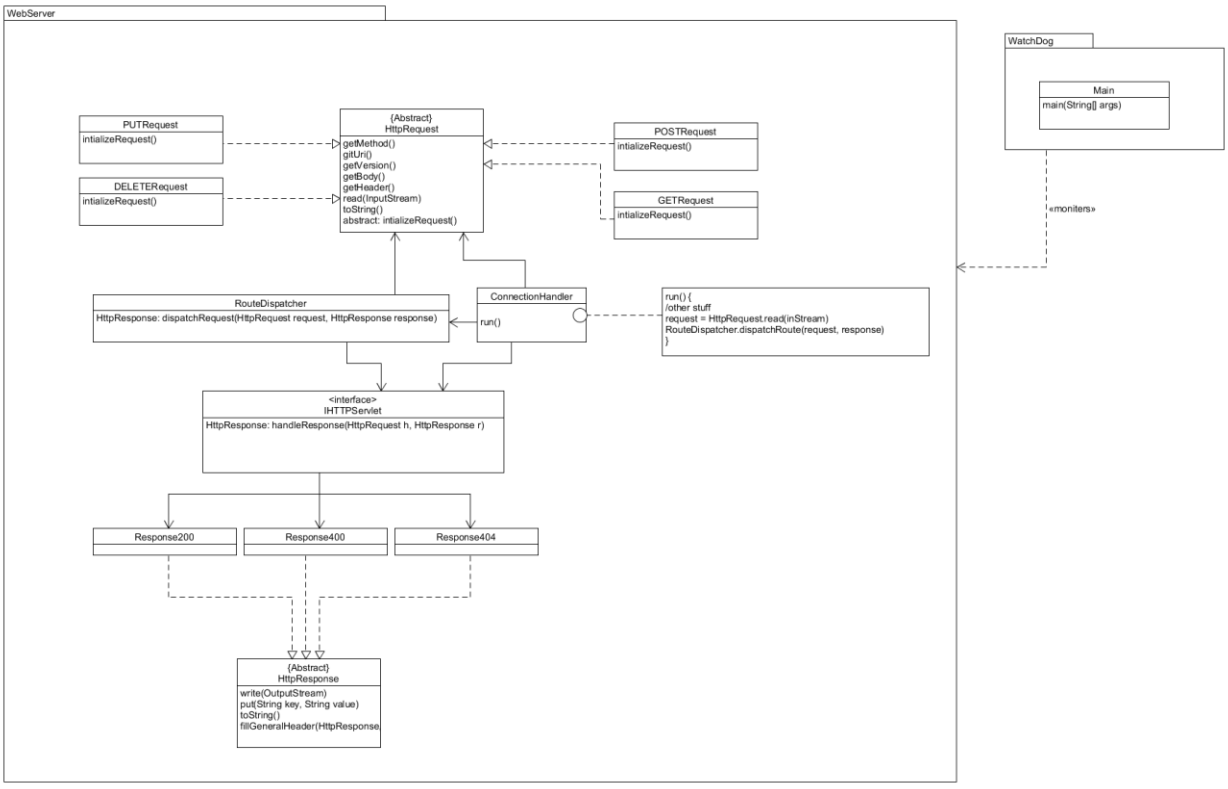
Gabriel Glenn and Jack Petry

Design Patterns

Command Pattern – The client sends request and responds. These are created as objects that are easily extensible.

Prototype Pattern – Our response and request classes are all abstracted to the same objects. This allows the user or later programmers to easily add both responses and request without changing the source code.

Updated UML



Change Log

IHttpServlet – and extensible interface that allows user to create their own Servlets. These servlets uniquely handle requests based on the user's implementation.

RouteDispatcher – loads new Servlet plugins and maps their routes. Also handles incoming requests dispatching them to the correct servlet.

Watchdog – monitors our server to make sure that it is servicing requests. If it cannot be requested to or doesn't respond in a reasonable amount of time, it restarts the server. Certainly a large number of requests are lost by doing this but it is better than it entirely crashing and stopping functioning.

Resource Manager – The resource management in the system prevents any single user from consuming too many resources from the server. This allows the system to work more fairly and to keep running under higher stress loads

Possible Improvements

Currently RouteDispatcher both loads and dispatches requests. Those two jobs should be separated.

Currently adding request classes requires naming convention that follows [request name in uppercase] + Request. A proxy protecting the server could help to protect against DDOS. Adding a larger set of operations and responses would make the server better as well. Also, probably could make the request reflection area more robust.

Feature list

GET, POST, PUT, and DELETE – (Jack) the user can choose how and where each of these requests are handled

Servlet Creation – (Jack) allows the user to create their own unique Servlet

Dispatch Requests-(Gabe) Allows the user to dynamically add servlets and access them via their browser

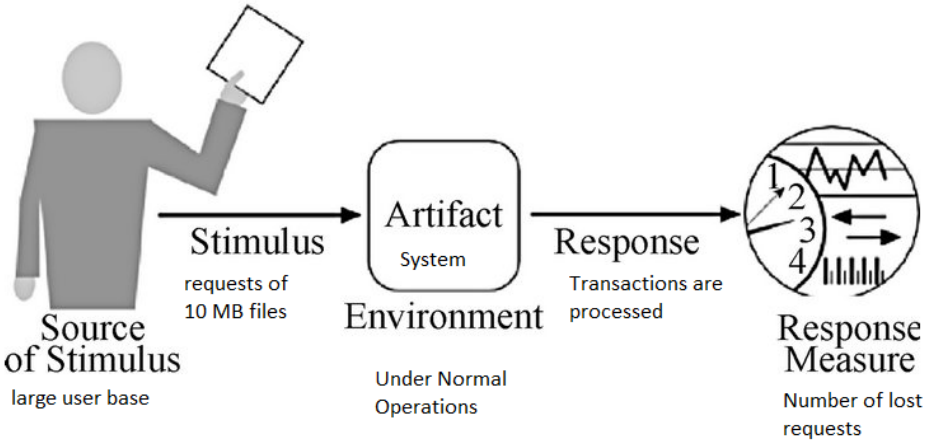
Watchdog – (Gabe) monitors the web server and turns it on if it's disconnected

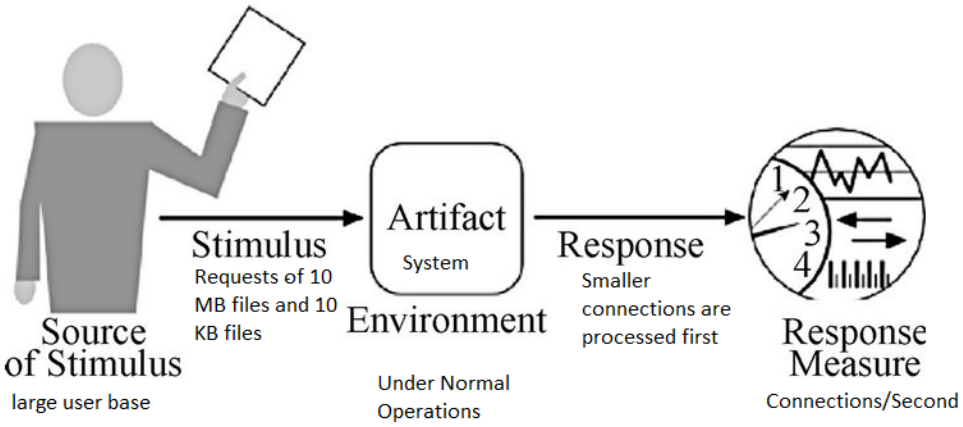
Resource Manager – Allocates a set amount of resources per user in order to more fairly accommodate all of the users fairly

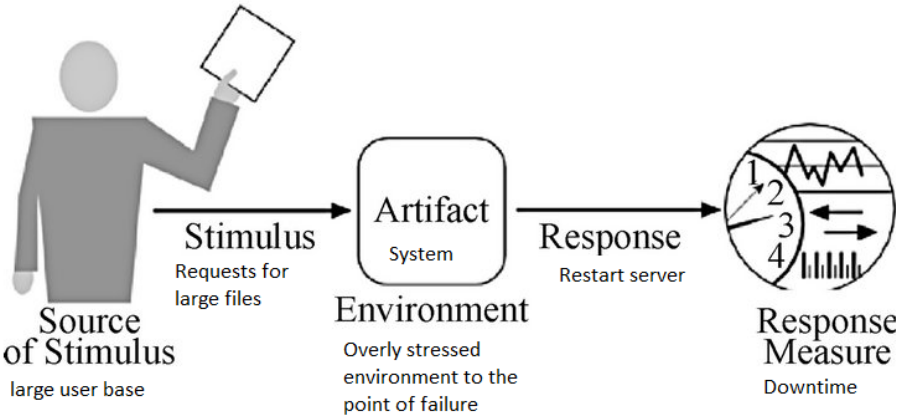
Architectural Evaluation

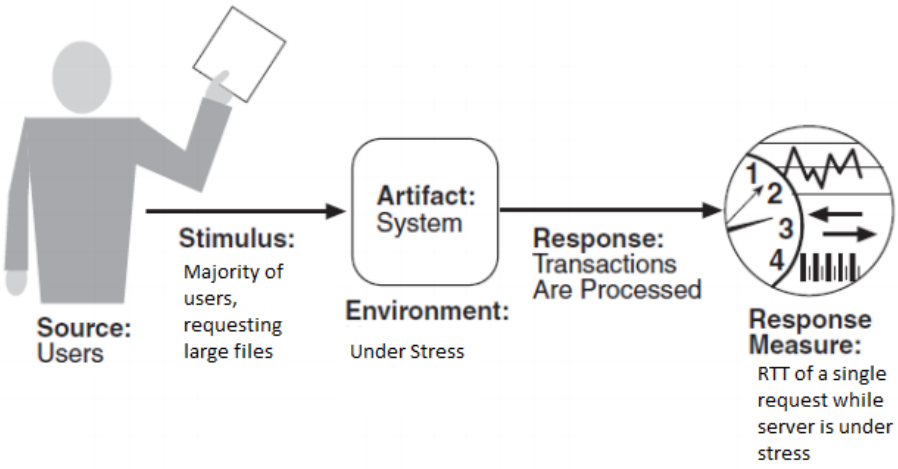
Test	Metric	Stress	Result Before Improvements	Result After Improvements
Denial of Service (Performance)	Miss Rate	50 requests per second of a 10 Megabyte file for 20 seconds	448 requests were lost	44 requests were lost
Large File Request (Performance)	Average Request Time	40 requests per second for a 10 Megabyte file and a 10 Kilobyte file	34.87 connections per second	37.21 Connections per second
Denial of Service (Availability)	RTT of a request while server is under stress	50 requests per second for a 10 Megabyte file	3.28 seconds RTT for request of a 10 Kilobyte file	21 ms RTT for request of a 10 Kilobyte file
Server Throttled to Death (Availability)	Server Downtime	60 requests per second for a 10 Megabyte file	Infinite downtime	Restarts with at most 5 seconds of downtime
Denial of Service (Security)	Latency of a request while the server is under DOS	60 requests per second for a 10 Megabyte and a 10 Kilobyte	It crashed	23 ms RTT for request of a 10 Kilobyte file
Data Source Integrity (Security)	Amount of Data Logged	Send a normal level of requests say 30 per second from multiple users	Zero data is logged	Every request is now logged

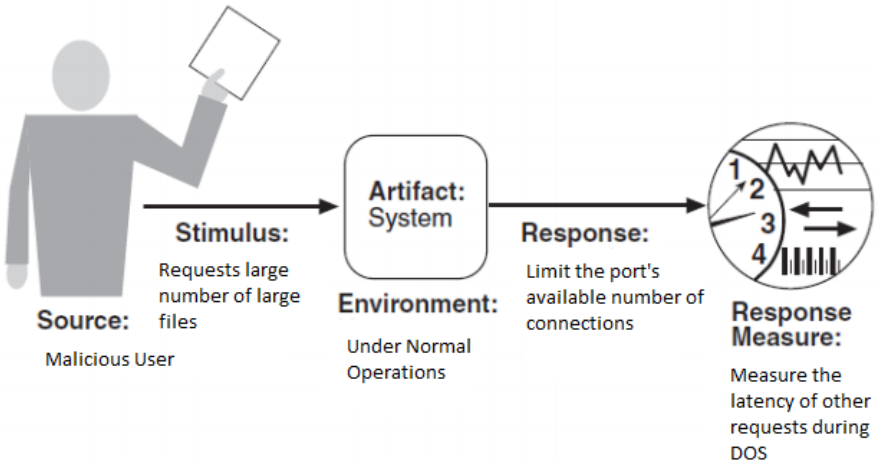
Concrete Scenario Diagrams

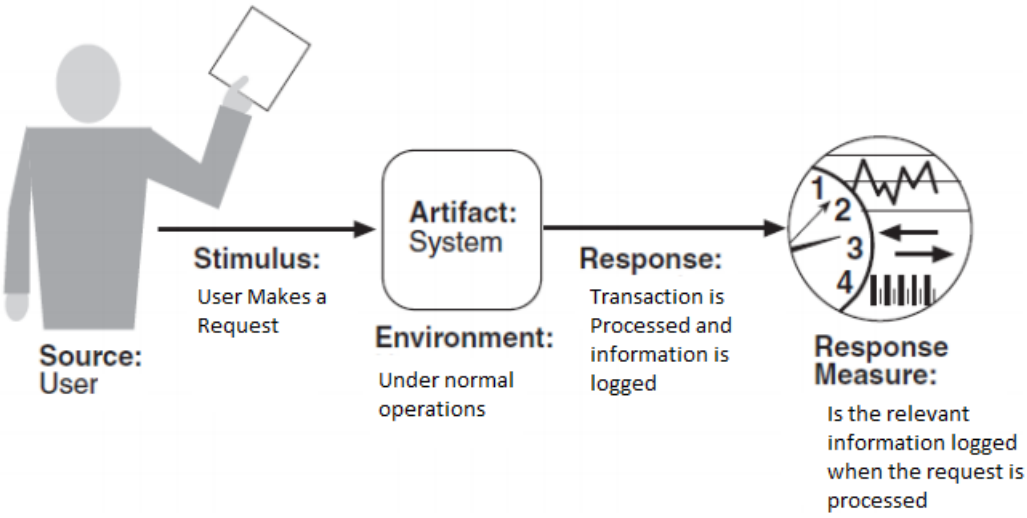







Source:
Users
Stimulus:
Majority of users, requesting large files
Artifact:
System
Environment:
Under Stress
Response:
Transactions Are Processed
1
2
3
4
Response Measure:
RTT of a single request while server is under stress


Source:
Malicious User
Stimulus:
Requests large number of large files
Artifact:
System
Environment:
Under Normal Operations
Response:
Limit the port's available number of connections
1
2
3
4
Response Measure:
Measure the latency of other requests during DOS


Source:
User
Stimulus:
User Makes a Request
Artifact:
System
Environment:
Under normal operations
Response:
Transaction is Processed and information is logged
1
2
3
4
Response Measure:
Is the relevant information logged when the request is processed

Improvement Tactics

All – Limiting sockets per host. This prevents any user from either uninternationally and maliciously consuming too many system resources. This helps prevent DOS attacks and improves performance for typical users.

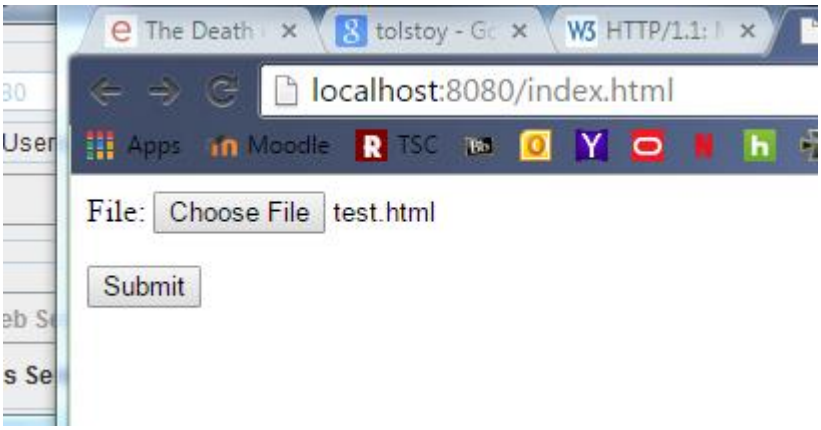
Performance – Prioritizing smaller files. We set up our server to prioritize smaller files in ordr to reduce average wait time. In a typlical environment, most users will be using the smaller files and people won't mind waiting a little longer in order to get really big files.

Availability – Watchdog. We developed a watchdog service in order to restart our system if it comes down.

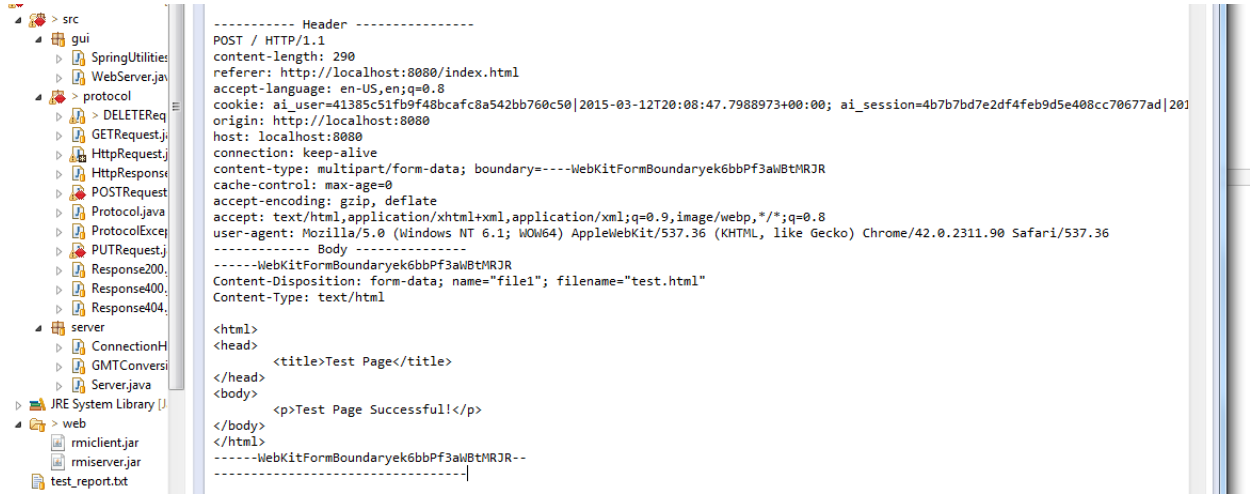
Security – Logging. We implemented basic logging functionality in order to make security threats identifiable and retracable so the authority can remedy the problem.

Test Report – MS1

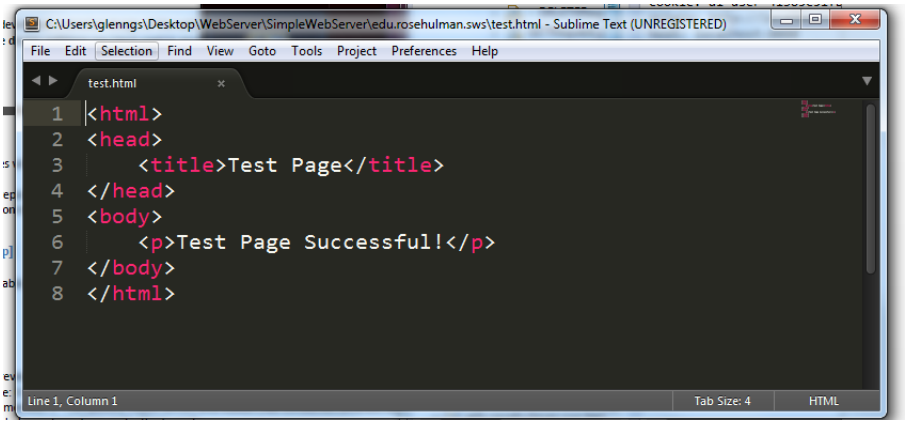
POST test



Submitting test.html



This is the request the server received after the submit button is pressed

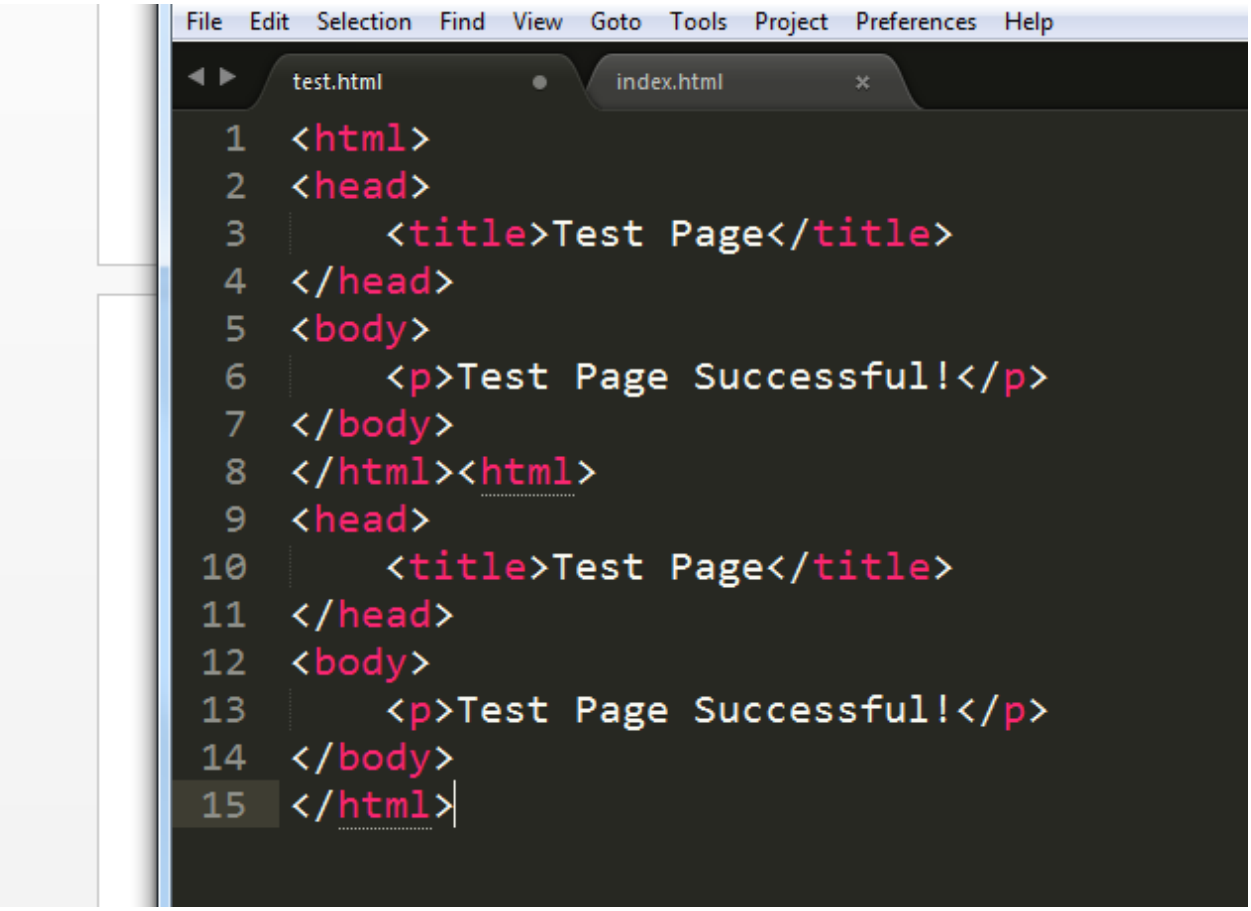


This is the file in the root dir that the request posted to

PUT Test

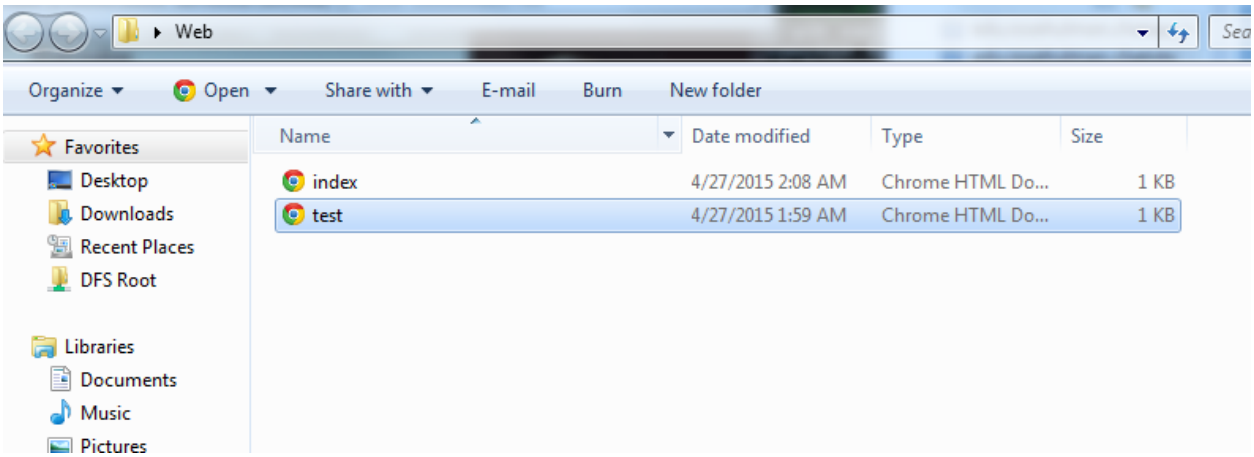
```
WebServer [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (Apr 27, 2015, 1:58:44 AM)
PUT / HTTP/1.1
content-length: 290
referer: http://localhost:8080/index.html
accept-language: en-US,en;q=0.8
cookie: ai_user=41385c51fb9f48bcafc8a542bb760c50|2015-03-12T20:08:47.7988973+00:00; ai_session=4b7b7bd7e2df4feb9d5e408cc70677ad|20
origin: http://localhost:8080
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
content-disposition: form-data; name="file1"; filename="test.html"
host: localhost:8080
connection: keep-alive
content-type: text/html
cache-control: max-age=0
accept-encoding: gzip, deflate
user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.90 Safari/537.36
------------- Body ---------------
<html>
<head>
        <title>Test Page</title>
</head>
<body>
        <p>Test Page Successful!</p>
</body>
</html>
------WebKitFormBoundaryek6bbPf3aWBtMRJR--
```

Working with the same file we again have the request reaching the server

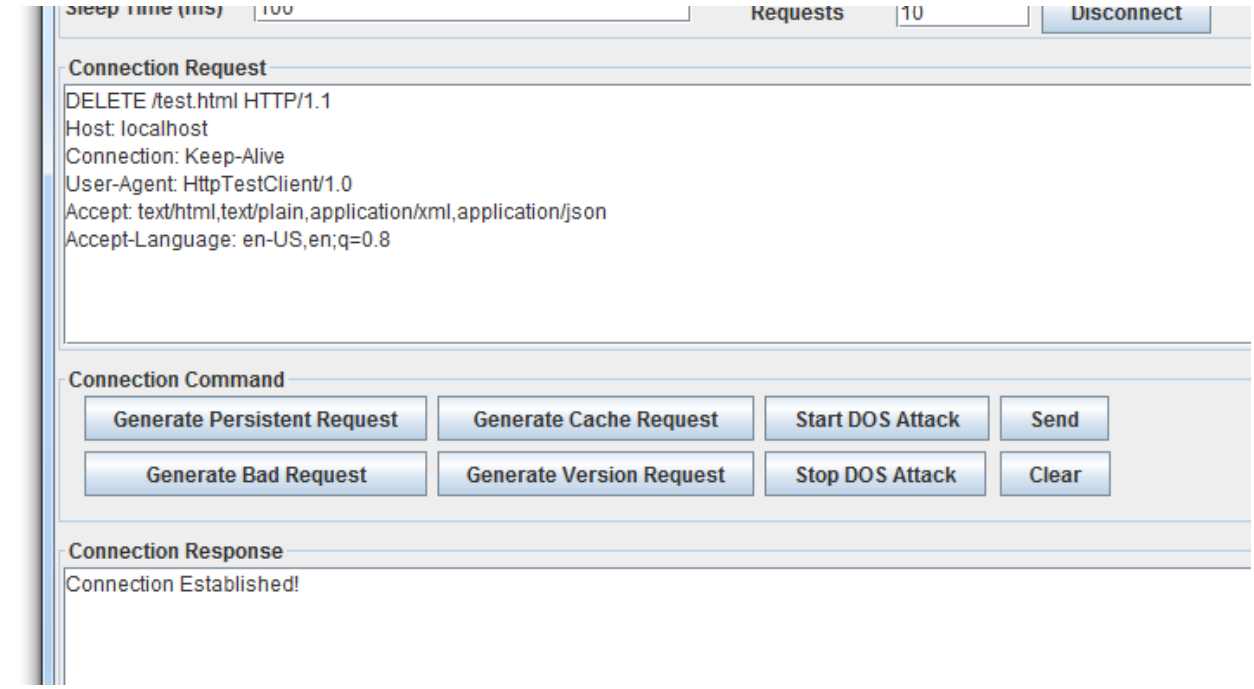


And here we have the result

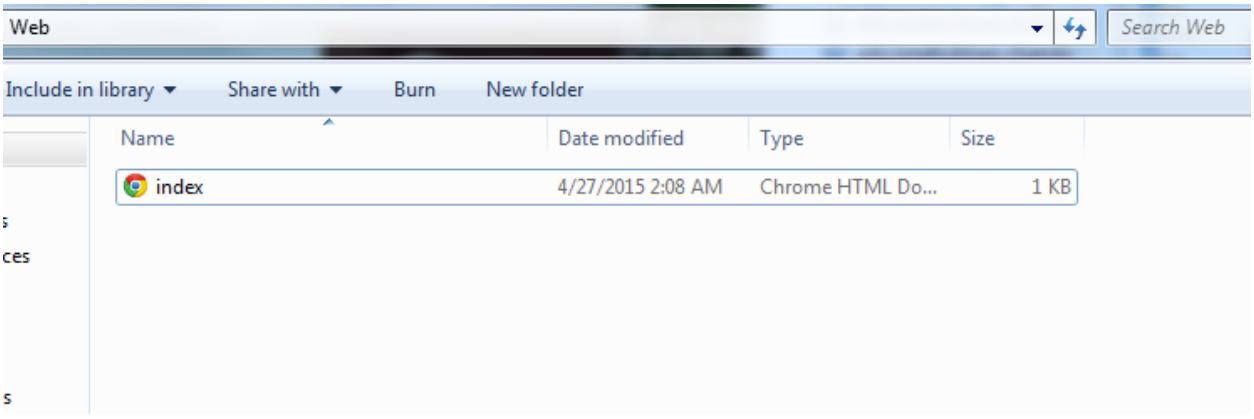
DELETE Test



There is the file we are going to delete



Here is the request



And now it has been deleted

Test Report - MS 2

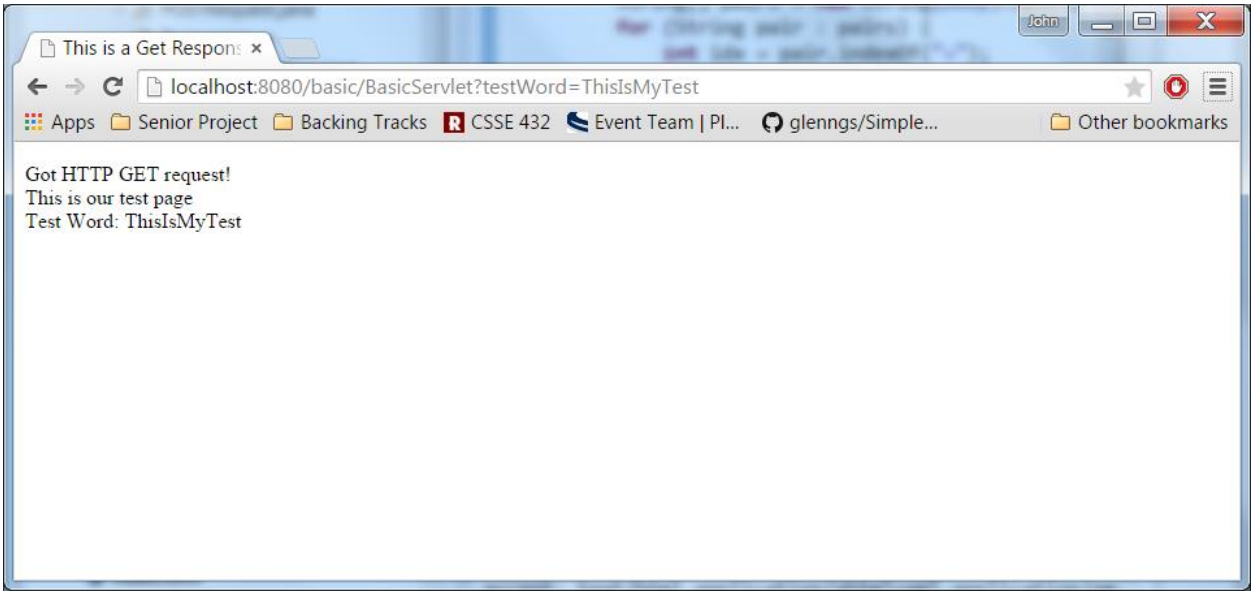
Start the server. In the directory where the server is running, there either initially needs to be basic.jar which is the creation of the TestServlet project or you need to add it after it starts running (and it will be automatically added).

GET.

There are a couple of different pages for GET.

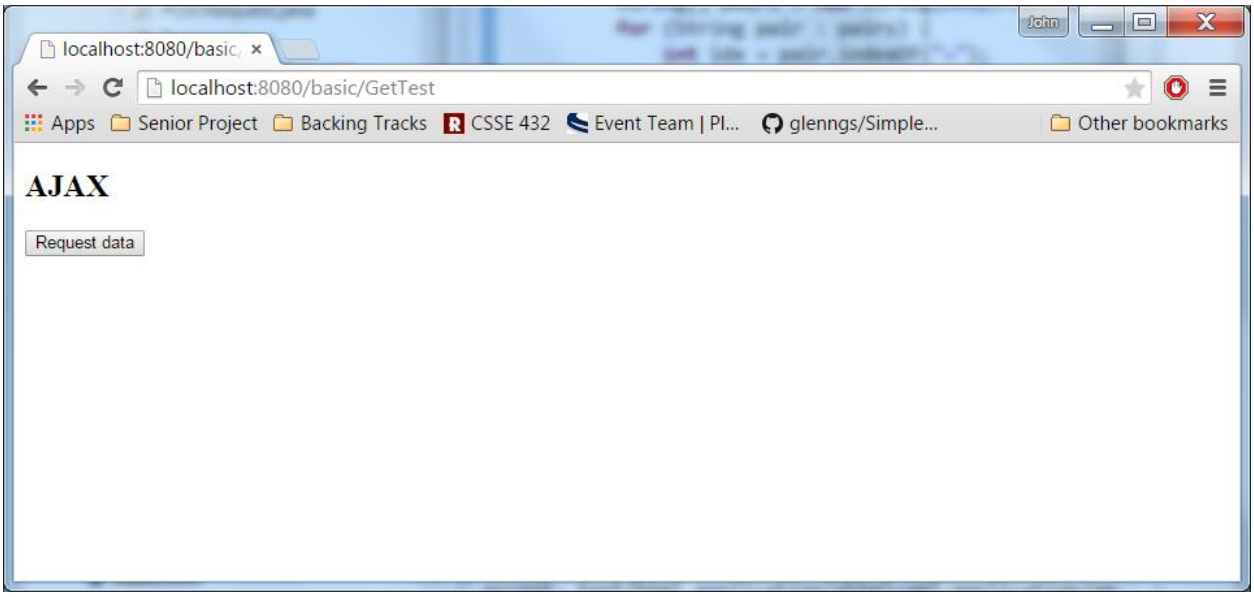
Go to http://localhost:8080/basic/BasicServlet?testWord=ThisIsMyTest

Gives a response like the following:



Another page that we have serves static GET content and useful for testing the POST command.

http://localhost:8080/basic/GetTest


John
localhost:8080/basic/
localhost:8080/basic/GetTest
Apps
Senior Project
Backing Tracks
CSSE 432
Event Team | Pl...
glenngs/Simple...
Other bookmarks
AJAX
Request data

POST.

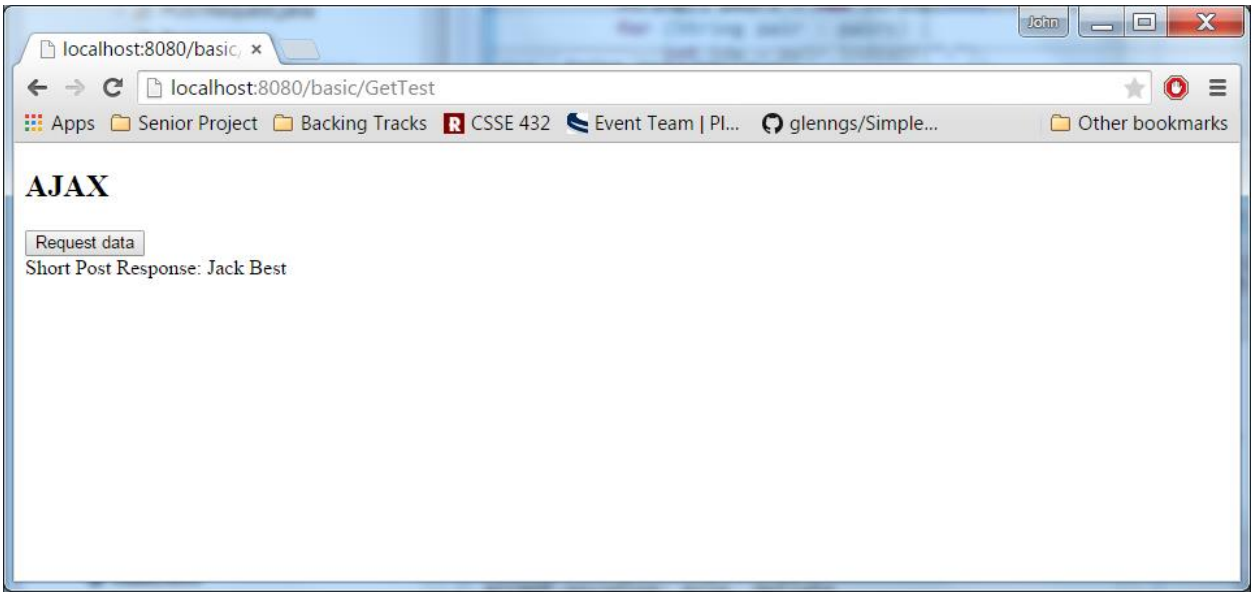
The POST command can be tested using the button of the last page.

It uses an ajax call like the following.

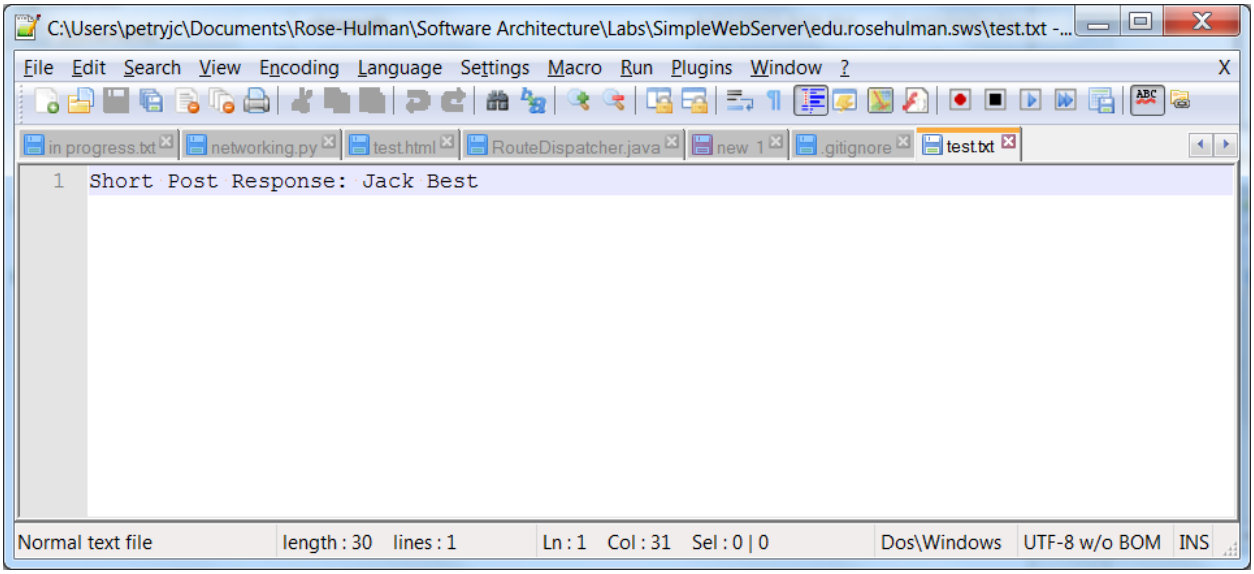
xmlhttp.open("POST","http://localhost:8080/basic/PostTest",true);

xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");

xmlhttp.send("word1=Jack&word2=Best&filename=test.txt");

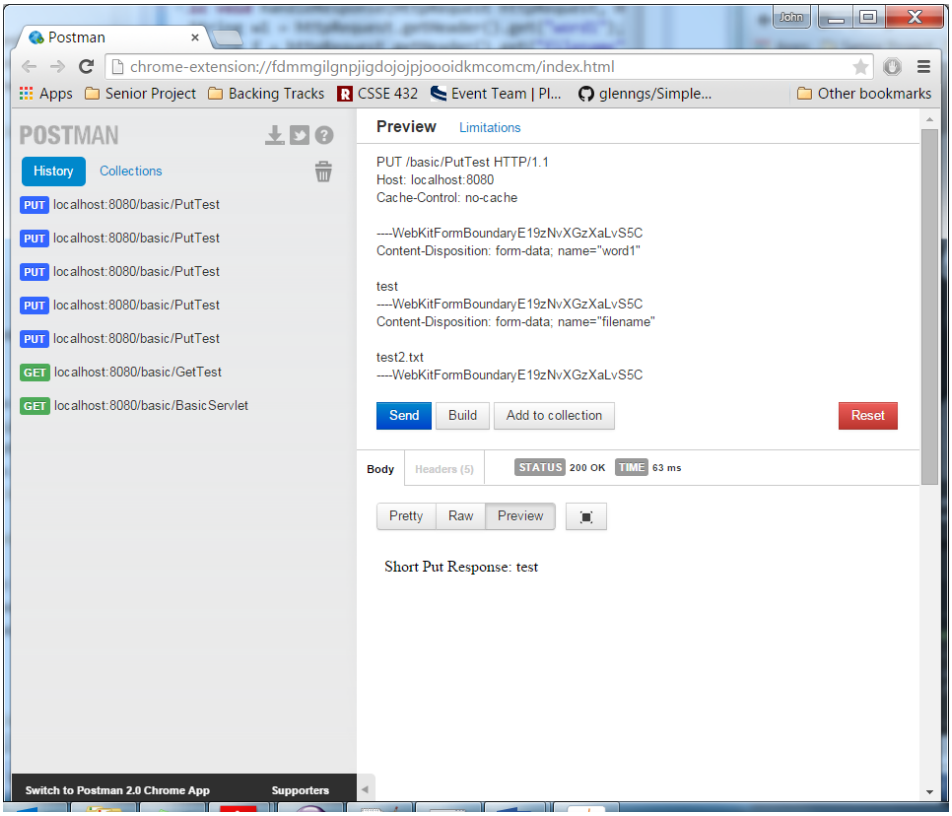


It also creates a file (test.txt) that has the same response contents.

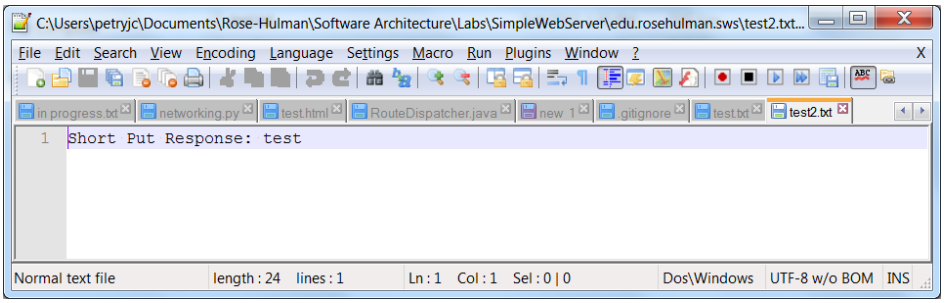


PUT.

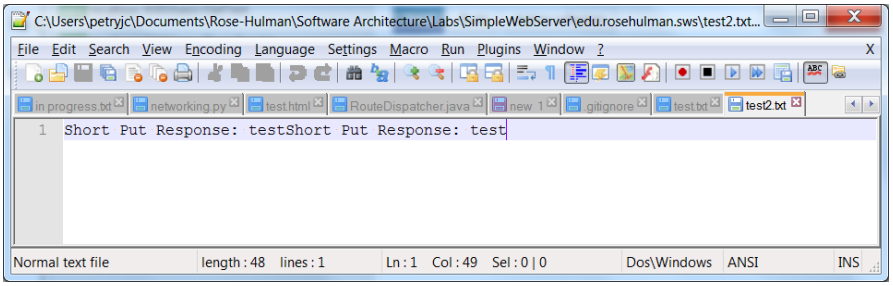
The PUT command can easily be tested using POSTMAN.



It also creates a file (test2.txt)

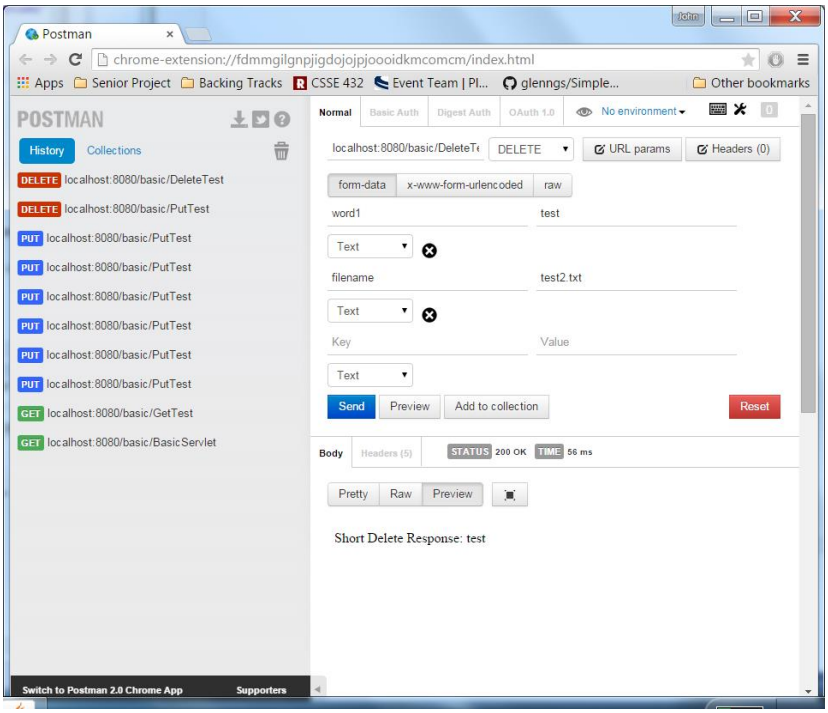


If you do the put again. It appends because I thought that would be nice of the servlet to do.



DELETE

DELETE can also be easily tested using POSTMAN.



It also deleted the given file because I thought that sounded cool. (Notice test2.txt is gone)