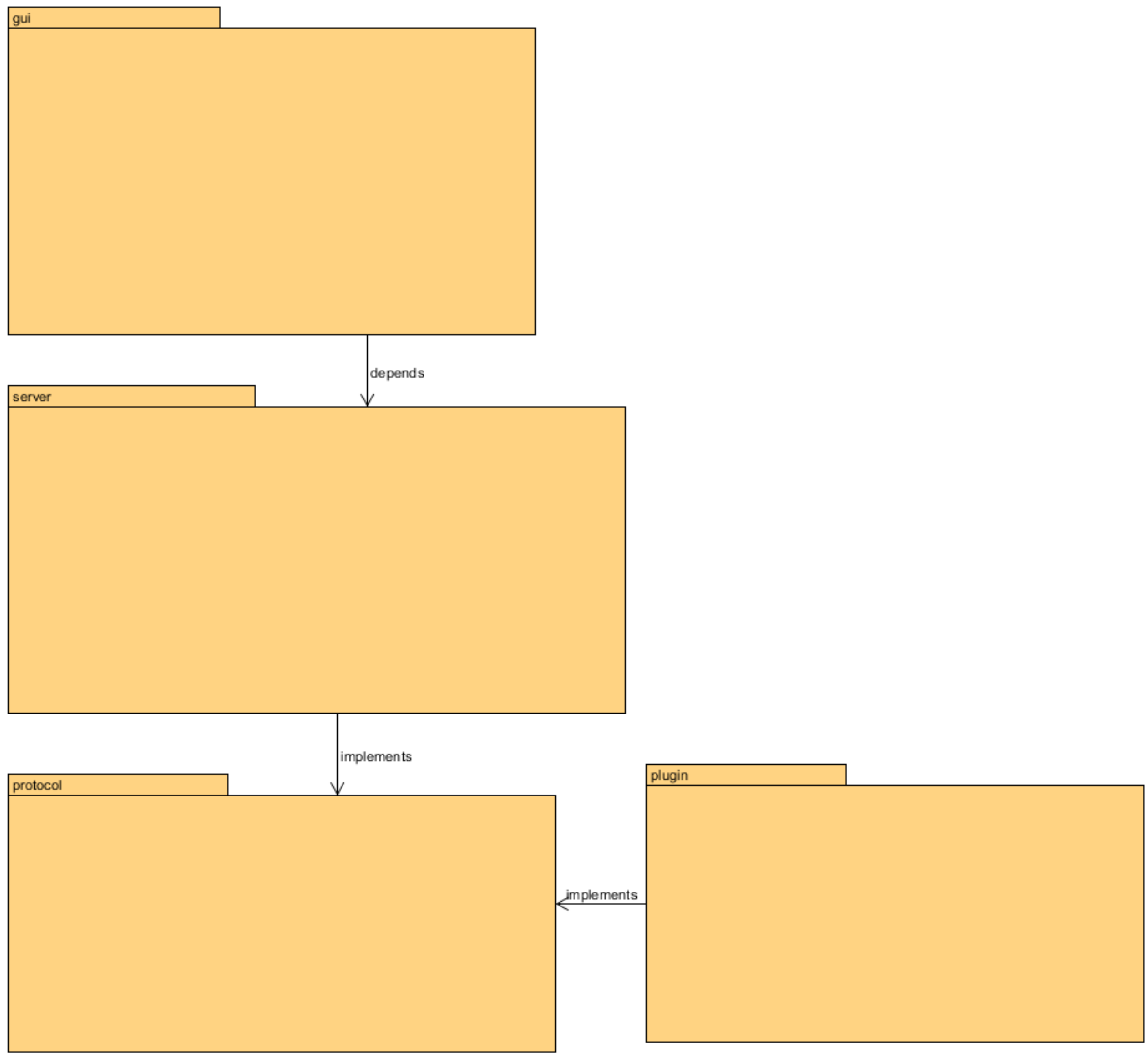
Change Log

Milestone	Changes
1	Started Document
2	Updated architecture and design diagrams. Updated Milestone Changes. Updated design patterns and added improvements.Updated test log.
3	Added Title Page. Added Evaluation of attributes. Added improvement tactics. Reordered the sections along with renaming some of them.
4	

Architecture and Design

Web Server Architecture

No changes were made to the original server architecture. All changes were made within the modules themselves. In addition plugins can be added to the server.



Web Server Design

Below can be found the architecture of the web server. It now implements interfaces for requests and responses instead of a generic HttpRequest and HttpResponse allowing for increased modifiability as long as the requirements of the interfaces are met.

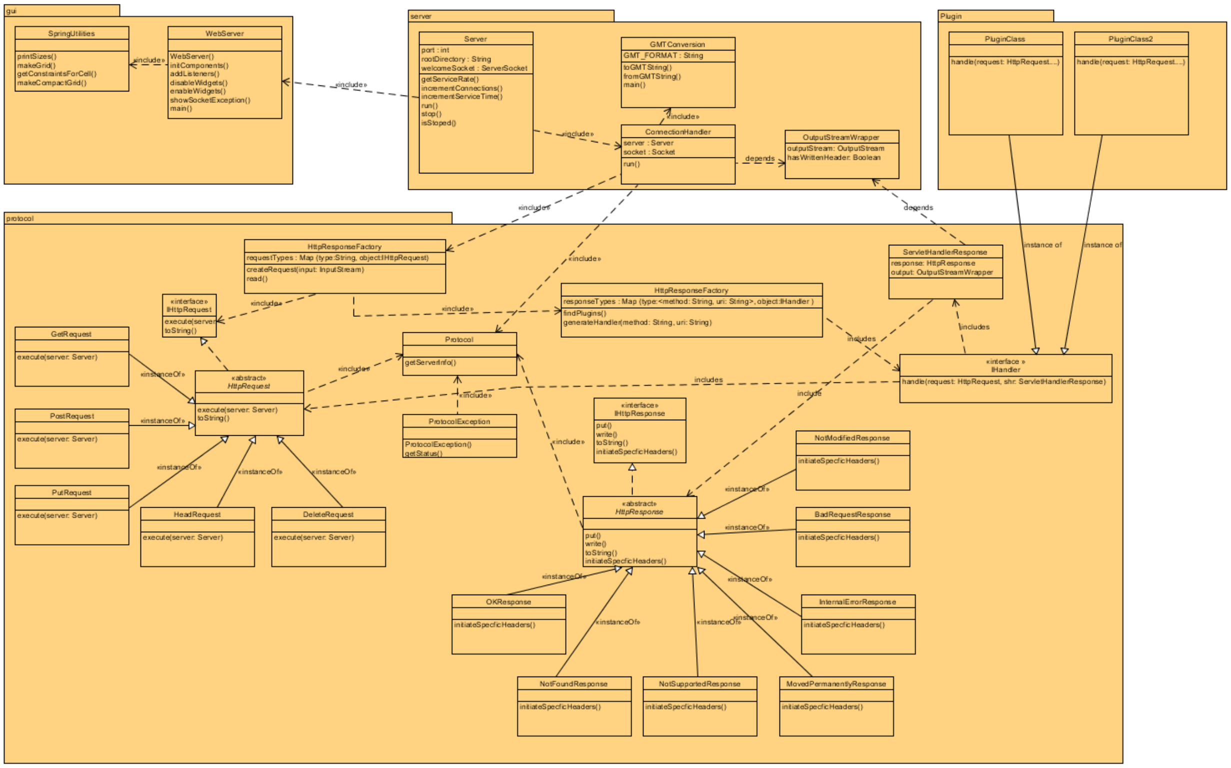
We added a HttpRequestFactory to facilitate additional request types. This factory uses a map of the request types and their associated objects. To allow additional types of requests to be easily added, we created the IHttpRequest interface and the HttpRequest abstract class. All of the existing requests make use of this. For the most recent milestone, we moved the functionality of the requests to the new IHandler class. This allows users to add their own handlers through plugins, and specify different execution behavior. The request header includes a uri which specifies which plugin will be used.

In addition to adding the already present methods in the original HttpResponse to the IHttpResponse interface, an additional method for generating response specific headers was added and the HttpResponse was made into an abstract class implementing IHttpResponse. This is currently only used in the 200 OkResponse. This is because the HttpResponseFactory originally added additional headers about the file returned in the response. In addition the individual create responses that were contained within the HttpResponseFactory were removed and a generic createResponse method was added that takes the response code in addition to the original parameters of the individual create methods. It uses a map which uses the string value of the response code to create the correct IHttpResponse object and returns it. Since the map does not have access to the file yet an additional method was added to the interface and abstract classes which allows the file to be set after initiation of the class. Each individual response type has a blank constructor. These constructors automatically call the super constructor with the correct parameters. In addition to these changes a MovedPermanentlyResponse and an InternalErrorResponse were added and their codes and text were added to the Protocol. The HttpResponseFactory will now generate an InternalErrorResponse whenever an attempt is made to send an unsupported response.

To allow each request to send its response as soon as possible, we also added the ServletHandlerResponse and OutputStreamWrapper classes. The ServletHandlerResponse includes the response and the OutputStreamWrapper and writes to the output when the response is generated. The response needs to wait for the wrapper confirm that it has sent the header before outputting the rest of the response.

The HttpResponseFactory was updated to read in plugins and create a mapping of them. It then generates the handler and returns it to the RequestFactory to handle the actual request. It still fills general headers for responses and generates a response but this call is now made in plugins instead when they implement the IHandler interface. The IHandler interface allows the plugins to access the request and the servlet for handling the response and the output stream. They then do whatever they need to and then create a response. For example right now all of the plugins are very simple and basic and only implement code that is the equivalent of the execute on the HttpRequest Objects.

In Milestone 3, we changed when we handle requests. Where before the server had been processing requests as they came in, now we queue all of the requests and process them every .1 seconds based on each user’s requests per minute. Users with fewer requests per minute have their requests handled first. Additionally, users with too many requests per minute, indicating a security attack, are completely denied for one minute.


gui
SpringUtilities
printSizes()
makeGrid()
getConstraintsForCell()
makeCompactGrid()
WebServer
WebServer()
initComponents()
addListeners()
disableWidgets()
enableWidgets()
showSocketException()
main()
«include»
server
Server
port : int
rootDirectory : String
welcomeSocket : ServerSocket
getServiceRate()
incrementConnections()
incrementServiceTime()
run()
stop()
isStoped()
GMTConversion
GMT_FORMAT : String
toGMTString()
fromGMTString()
main()
ConnectionHandler
server : Server
socket : Socket
run()
depends
OutputStreamWrapper
outputStream: OutputStream
hasWrittenHeader: Boolean
Plugin
PluginClass
handle(request: HttpRequest....)
PluginClass2
handle(request: HttpRequest....)
instance of
protocol
HttpResponseFactory
requestTypes : Map (type:String, object:IHttpRequest)
createRequest(input: InputStream)
read()
«interface»
IHttpRequest
execute(server
toString()
GetRequest
execute(server: Server)
PostRequest
execute(server: Server)
PutRequest
execute(server: Server)
HeadRequest
execute(server: Server)
DeleteRequest
execute(server: Server)
«abstract»
HttpRequest
execute(server: Server)
toString()
«instanceOf»
Protocol
getServerInfo()
ProtocolException
ProtocolException()
getStatus()
HttpResponseFactory
responseTypes : Map (type:<method: String, uri: String>, object:IHandler)
findPlugins()
generateHandler(method: String, uri: String)
includes
ServletHandlerResponse
response: HttpResponse
output: OutputStreamWrapper
«interface»
IHandler
handle(request: HttpRequest, shr: ServletHandlerResponse)
«interface»
IHttpResponse
put()
write()
toString()
initiateSpecficHeaders()
«abstract»
HttpResponse
put()
write()
toString()
initiateSpecficHeaders()
NotModifiedResponse
BadRequestResponse
InternalErrorResponse
OKResponse
NotFoundResponse
NotSupportedResponse
MovedPermanentlyResponse
initiateSpecficHeaders()

Design Patterns

We made use of several design patterns in our new design. We created abstractions of the HttpRequest and HttpResponse to allow these classes to be treated generically. This also allows for extensibility of the existing request and response types. We added a HttpRequestFactory and improved on the existing HttpResponseFactory. The Factory design pattern allows us to easily create specific instances of the abstract response and requests types.

We also added a Singleton HttpResponseFactory in HttpRequestFactory. This will allow us to use the response factory without having to rediscover plugins for each request.

We used the Decorator or Wrapper Pattern for the OutputSteamWrapper. This allowed us to add the additional functionality of checking that we sent the header while still including the same OutputSteam object.

Tactics

Predictive Modeling - Availability

By keeping track of our system performance metrics, we can predict when the server will be overloaded, which would normally result in request failures. We can then take appropriate measures, such as limiting access per user.

Detect Malicious Users - Security

For this tactic, we keep track of the number of requests originating from one user in a short time interval. If this number exceeds our expected number, than we can treat this user as a security threat, as they are probably attempting to bring down the server through a DDOS or similar attack. Our server can then take action against that user, such as limiting their access or simply denying their requests for a time.

Dynamic Priority Handling - Performance

Using this tactic allows our server to handle each request in turn. By evaluating the requests in a round-robin fashion, we can ensure that each request is handled in a reasonable amount of time.

Recover - Availability, Security

If the server goes down, either from a security attack or overloading, all data should be recovered and the server should restart within a few minutes. This will allow us to keep high availability in the long term as well as resolve any loss of data due to a security attack.

Maintain Multiple Copies of Data - Performance, Security

Keeping multiple copies of data on our server improves both performance and security. Under normal conditions, caching data can allow for easier access of frequently requested files. If the data is modified or deleted due to a security attack, we can load from one of the copies to restore the old data.

Revoke Access - Security
The server will check request uris to determine if any illegal paths are entered such as ones containing “..”. If this is found the server will deny access to that file and restrict the user’s access by limiting requests. Additionally, if the user has made too many requests per minute, indicating a security threat, the user is denied access.

Feature Listing

Milestone 2 Features:

1. Can dynamically add request handler plugins
2. Plugins are loaded upon each request after being dropped in the plugin directory
3. Added a basic GetPlugin
4. Added a basic PostPlugin
5. Added a basic PutPlugin
6. Added a basic DeletePlugin

Milestone 2 Work Assignment:

Lindsey:
- Added the IHandler class and plugin functionality. Users can now add their own Handlers to process new types of requests.
- Created a GetPlugin
- Created a PostPlugin
- Created a DeletePlugin
- Fixed PUTPlugin to work.

Logan:
- Removed the execution of requests to the new IHandler class
- Made a singleton HttpResponseFactory in HttpRequestFactory so that new plugins don’t have to be loaded for each request
- Added ServletResponseHandler and OutputSteamWrapper to allow each request to send its response
- Create a PUTPlugin

Milestone 3 Features:

1. Default HTML requests that work without any plugins.

Milestone 3 Work Assignment:

Lindsey:
- Modified code to all for default HTML requests without plugins.

- Security 1 testing and tactics
- Added 401 Response
- Performance 1, Availability 2

Logan:

- Performance 2, Availability 1, Security 2

Architectural Evaluation and Improvements

Availability 1

Scenario: Overloaded Server

Source: User
Stimulus: Normal Request
Environment: Overloaded Environment
Artifact: Server
Response: User’s request is queued until the server can handle it. Subsequent requests are limited by the time since the previous request.
Response Measure: The server will start serving the user’s request within 3 seconds of arrival

Testing:

For testing this, we can simulate the server being overloaded by having a different client sending multiple requests. We can then test whether the server handles an ordinary request within the response measure time.

Improvement Tactics:

This scenario uses predictive modeling to maintain availability. Predictive modeling allows us to keep track of the number of requests per minute a user makes and limit their access accordingly.

Improvement Results:

Although we were unable to fully test an overloaded environment, the system now handles requests independently from receiving them. Users with fewer requests per minute have priority in getting their requests processed.

Availability 2

Scenario: Recover deleted file

Source: User
Stimulus: Deletes system file
Environment: Normal Environment
Artifact: System Data
Response: The appropriate file is fetched from the copies of data and restored
Response Measure: Deleted file is detected within 30 seconds and then restored

Testing:

We can test this by manually deleting a file from the project. We can then measure how long it takes the server to restore that file.

Testing Results:

Before improving the system, a file such as index.html and upload.html which are important for testing our server and for posting new files to the server can be deleted. If they are deleted they are lost and never restores and must be retrieved through our repository. Editing to these files can also occur. While this is not deletion it also pertains to the same task here of making the files available. This will also be improved upon.

Improvement Tactics:

The scenario makes use of the multiple copies of data tactic. By keeping multiple copies, we can restore the original copy if it becomes corrupted or deleted.

Improvement Results:

To implement this a reoccuring scheduler was created to first backup the system's web directory every 3 hours. Three copies will be kept but it initial starts with 0 until 9 hours after running. The system automatically backups the web directory upon startup. Next another scheduler was made to run every 20 seconds to ensure that the deleted/changed files are detected within the 30 seconds of the deletion/change. This checks each file and restores it from the last backup. After testing, the system was able to successfully detect deletions and changes within 30 seconds and restore them.

Performance 1

Scenario: Accessing Same File

Source: Many Users or Requests
Stimulus: Accessing the same file
Environment: Normal Environment
Artifact: Server
Response: The data is cached for easier access
Response Measure: The average latency for the file request improves

Testing:

We can simulate this by making multiple requests for the same file. We can then check whether the server has cached that file.

Testing Results:

After testing the total time it took transfer two files of different sizes (1KB and 3536KB), average delivery times were calculated to be 39.5ms and 48642ms respectively.

Improvement Tactics:

This scenario uses multiple copies of data, this time for performance. By keeping frequently accessed files cached, they can be more easily accessed and should improve performance. The data is cached after 20 requests for the same file in 10 seconds and is cached for a short time, about 10 seconds, after which time the data is cleared from the cache if it has not been accessed.

Improvement Results:

To implement this a reoccuring scheduler was created to first backup the system's web directory every 3 hours. Three copies will be kept but it initial starts with 0 until 9 hours after running. The system automatically backups the web directory upon startup. Next another scheduler was made to run every 5 seconds to clean the cache. The cache was implemented as a map with the filename as the key and a list of attributes for that file. The first attribute was the body of the file and the second was the last accessed time in ms. The file was removed from the cache if the last accessed time was greater than 10 seconds. In order to use caching, a GET request must be made. This is a default request. In order to using caching with plugins, the plugin must be given access to the server and must implement the caching methods. After implementation and testing it was found that the latency for the same two files now was 25.1ms and 22527ms which is an improvement for both. However one strange thing occurs where a 404 error is presented within the index.html file when requested after caching.

Performance 2

Scenario: Multitude of Users under Overloaded conditions

Source: Many Users
Stimulus: Making requests
Environment: Overloaded Environment
Artifact: Server
Response: Users' requests are queued and handled so that users with fewer requests are handled first
Response Measure: Each user's request is processed within 3 seconds of submission.

Testing:

We can simulate this test by writing a mock client program submitting requests with randomized IPs. If the test is successful, then each request is handled in turn within the appropriate time.

Improvement Tactics:

This scenario uses dynamic priority handling to process each request in turn. Requests from users with fewer requests per minute are handled first.

Improvement Results:

Although we were unable to fully test an overloaded environment, the system now handles requests independently from receiving them. Users with fewer requests per minute have priority in getting their requests processed.

Security 1

Scenario: Illegal Access

Source: User
Stimulus: Accesses a system file (such as HTTPResponse.java)
Environment: Normal Operation
Artifact: System Data
Response: Send 401 Unauthorized Response
Response Measure: Was 401 Response sent

Testing:

In order to test this, a illegal uri will be sent including something of the format of “..\file.java” or “~\file.java”. The “..” and “~” would allow backwards navigation of the web folder giving access to the actual system files. If the request sends back a 401 response, it will be successful but if it sends back the data in the file it will be unsuccessful.

Testing Results:

After testing it was found that the browser automatically strips out the “..” from the url and the “~” doesn’t do anything. It was also determined that PostMan our other testing utility does the same thing. As such it currently appears that it would be impossible to navigate from the default folder to our system files; however, our code does not have protections itself so if someone was able to make requests with a tool that allowed the “..\” they would be able to navigate to system files so the improvement tactics are still going to be implemented. The returned result however is a 400 response so a 401 response will need to be implemented.

Improvement Tactics:

In order to prevent access to these files, the server will implement the revoke access tactic. This will be done by modifying the request or server code to check the uri for characters such as “..” before processing the request. The server will then implement a similar tactic to detect malicious users since it will then limit the access of the user but uses a different means to detect it. Also an additional response will be added for the 401 response. This addition can be seen in the Design UML in the Architecture and Design section.

Improvement Results:

After implementing the tactics, the testing results were the same as the original. This was due to us not being able to find a tool to allow us to send “..\” across.

Security 2

Scenario: DDOS Attack

Source: User
Stimulus: DDOS Attack
Environment: Normal Operation
Artifact: Server
Response: Limit User Requests per minute
Response Measure: Performance doesn't degrade

Testing:

Testing for a DDOS attack will be done by using a DDOS simulator provided by the professor. Before running the attack, the performance will be measured by the response time to a user request which should be within three seconds. The attack will then be ran on the server by tool and the performance will be measured during the attack through the response time of an additional client making normal requests. If the response time increases above three seconds then it will be unsuccessful but if the DDOS attacker cannot make as many requests after one minute and the performance for the additional client does not degrade past 3 seconds it will be successful.

Improvement Tactics:

In order to prevent DDOS attacks, the server will implement the detect malicious users tactic. It will do this by keeping a dictionary of the users that have made requests in the last minute which will use the IP as the key and the value will related to how many requests they have made in that minute. If the user exceeds 200 request in a minute the user's IP will be added to a restricted list that refreshes daily. If a user's IP is on this restricted list they will only be allowed to make 100 requests per minute.

Improvement Results:

Following our changes to the system, the user is denied access to the server for a minute once 200 requests per minute have been sent.

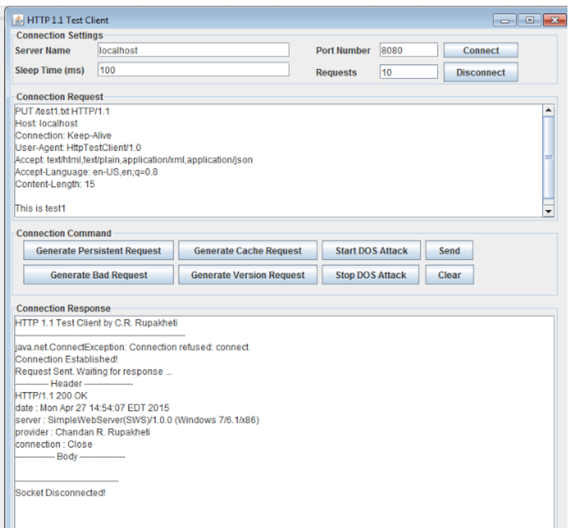
Future Improvements

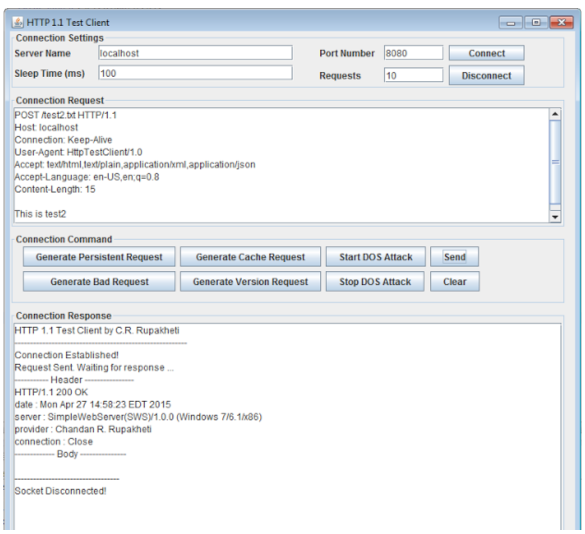
With our current implementation, all of the request types have similar execute methods. Some of the functionality and basic error checking could be abstracted to the HttpRequest class. The put and post request in particular are very similar; the only major difference being whether they overwrite the existing file. These requests could be combined into a file writer superclass, which both requests inherit. In addition, the performance could be improved by not creating a new HttpResponseFactory every time that a request is made. Instead we could initiate it once and check for changes in the plugin directory each time a request is made and only then reload the mappings.

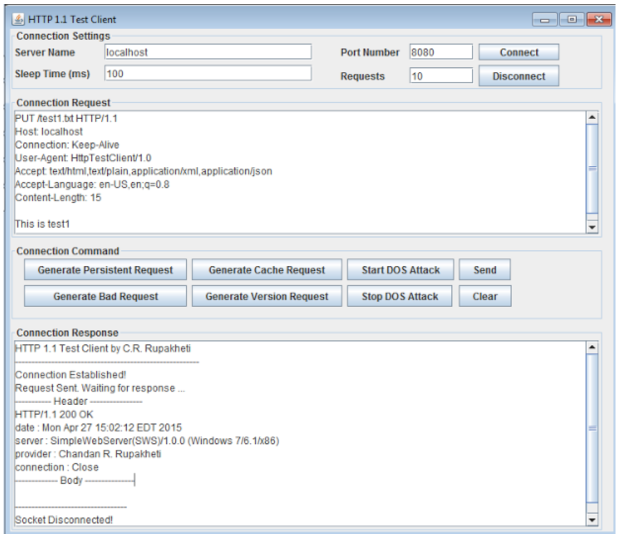
Test Report

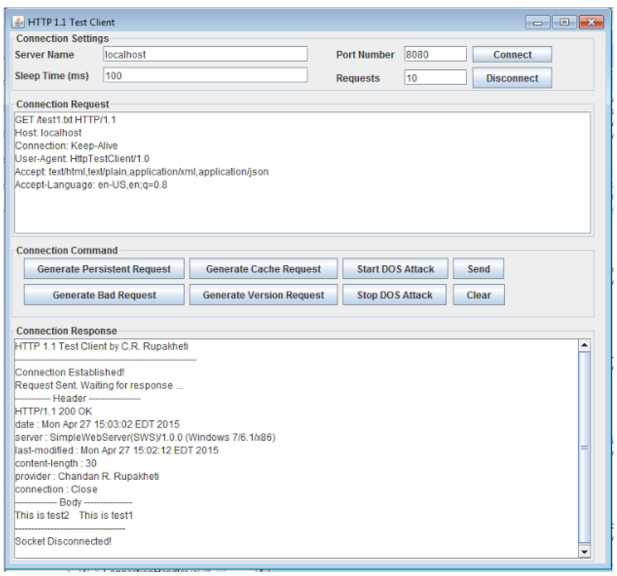
The following few pages show our testing results. First we create a file using PUT. Then we overwrite the contents of that file using POST. Next we DELETE the file. Finally we try a final GET method, which returns a 404, as the file has been deleted.

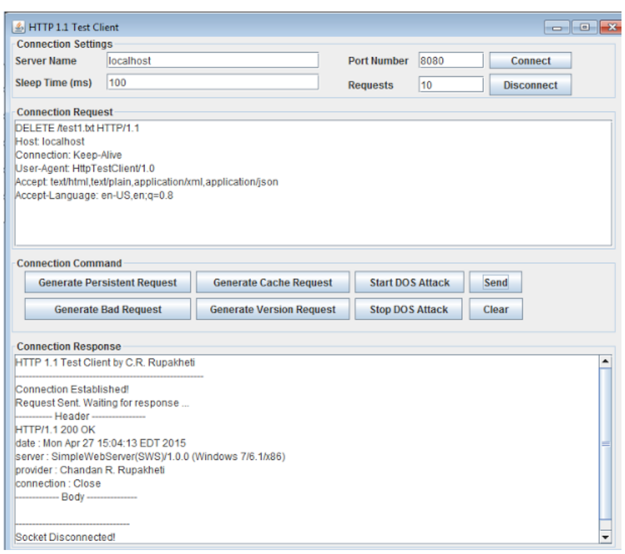
Milestone 1:

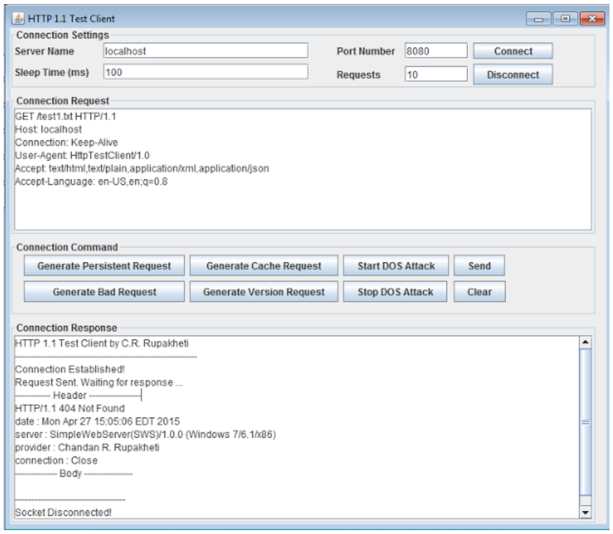



HTTP 1.1 Test Client
Connection Settings
Server Name
localhost
Port Number
8080
Connect
Sleep Time (ms)
100
Requests
10
Disconnect
Connection Request
POST /test2.txt HTTP/1.1
Host: localhost
Connection: Keep-Alive
User-Agent: HttpTestClient/1.0
Accept: text/html,text/plain,application/xml,application/json
Accept-Language: en-US,en;q=0.8
Content-Length: 15
This is test2
Connection Command
Generate Persistent Request
Generate Cache Request
Start DOS Attack
Send
Generate Bad Request
Generate Version Request
Stop DOS Attack
Clear
Connection Response
HTTP 1.1 Test Client by C.R. Rupakheti
Connection Established!
Request Sent. Waiting for response ...
Header
HTTP/1.1 200 OK
date : Mon Apr 27 14:58:23 EDT 2015
server : SimpleWebServer(SWS)/1.0.0 (Windows 7/6.1/x86)
provider : Chandan R. Rupakheti
connection : Close
Body
Socket Disconnected!


HTTP 1.1 Test Client
Connection Settings
Server Name
localhost
Port Number
8080
Connect
Sleep Time (ms)
100
Requests
10
Disconnect
Connection Request
PUT /test1.txt HTTP/1.1
Host: localhost
Connection: Keep-Alive
User-Agent: HttpTestClient/1.0
Accept: text/html,text/plain,application/xml,application/json
Accept-Language: en-US,en;q=0.8
Content-Length: 15
This is test1
Connection Command
Generate Persistent Request
Generate Cache Request
Start DOS Attack
Send
Generate Bad Request
Generate Version Request
Stop DOS Attack
Clear
Connection Response
HTTP 1.1 Test Client by C.R. Rupakheti
Connection Established!
Request Sent. Waiting for response ...
Header
HTTP/1.1 200 OK
date : Mon Apr 27 15:02:12 EDT 2015
server : SimpleWebServer(SWS)/1.0.0 (Windows 7/6.1/x86)
provider : Chandan R. Rupakheti
connection : Close
Body
Socket Disconnected!

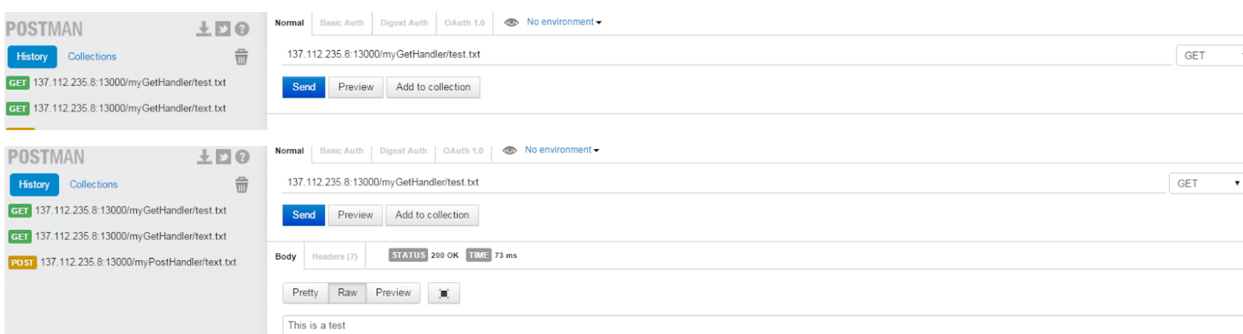

HTTP 1.1 Test Client
Connection Settings
Server Name
localhost
Port Number
8080
Connect
Sleep Time (ms)
100
Requests
10
Disconnect
Connection Request
GET /test1.txt HTTP/1.1
Host: localhost
Connection: Keep-Alive
User-Agent: HttpTestClient/1.0
Accept: text/html,text/plain,application/xml,application/json
Accept-Language: en-US,en;q=0.8
Connection Command
Generate Persistent Request
Generate Cache Request
Start DOS Attack
Send
Generate Bad Request
Generate Version Request
Stop DOS Attack
Clear
Connection Response
HTTP 1.1 Test Client by C.R. Rupakheti
Connection Established!
Request Sent. Waiting for response ...
Header
HTTP/1.1 200 OK
date : Mon Apr 27 15:03:02 EDT 2015
server : SimpleWebServer(SWS)/1.0.0 (Windows 7/6.1/x86)
last-modified : Mon Apr 27 15:02:12 EDT 2015
content-length : 30
provider : Chandan R. Rupakheti
connection : Close
Body
This is test2 This is test1
Socket Disconnected!


HTTP 1.1 Test Client
Connection Settings
Server Name
localhost
Port Number
8080
Connect
Sleep Time (ms)
100
Requests
10
Disconnect
Connection Request
DELETE /test1.txt HTTP/1.1
Host: localhost
Connection: Keep-Alive
User-Agent: HttpTestClient/1.0
Accept: text/html,text/plain,application/xml,application/json
Accept-Language: en-US,en;q=0.8
Connection Command
Generate Persistent Request
Generate Cache Request
Start DOS Attack
Send
Generate Bad Request
Generate Version Request
Stop DOS Attack
Clear
Connection Response
HTTP 1.1 Test Client by C.R. Rupakheti
Connection Established!
Request Sent. Waiting for response ...
Header
HTTP/1.1 200 OK
date : Mon Apr 27 15:04:13 EDT 2015
server : SimpleWebServer(SWS)/1.0.0 (Windows 7/6.1/x86)
provider : Chandan R. Rupakheti
connection : Close
Body
Socket Disconnected!



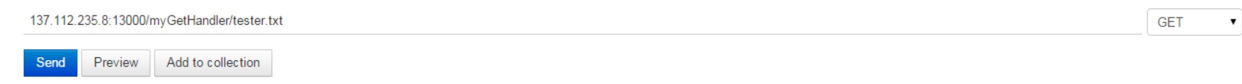
Milestone 2:

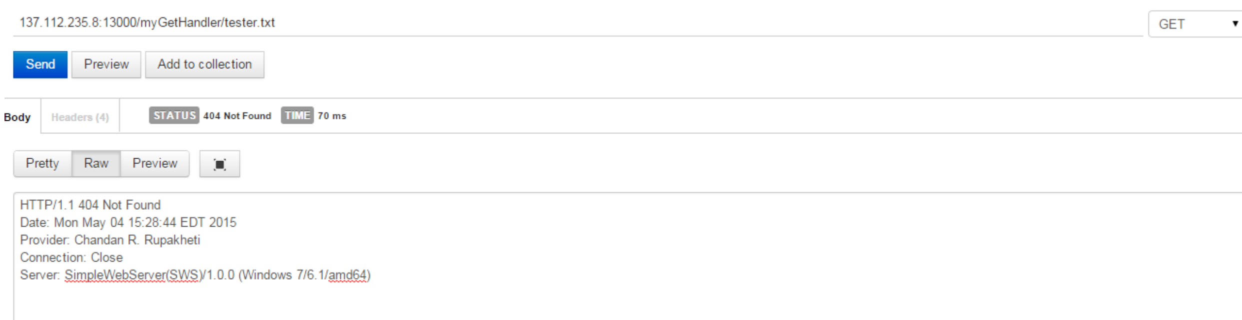
In this milestone we added functionality for user created plugins. Below is a test of a user created GET handler. Note that the request includes “/myGetHandler”, specifying which plugin should be used.



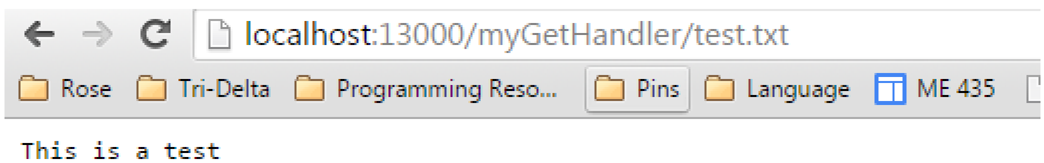
File Not there:

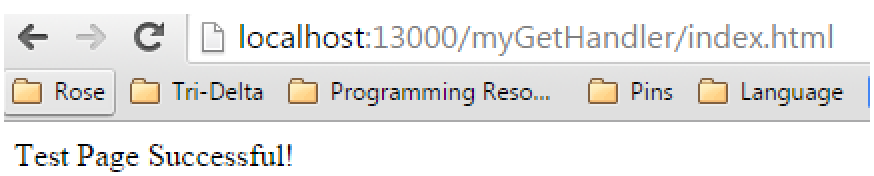
When trying to retrieve a file that does not exist, we return a 404



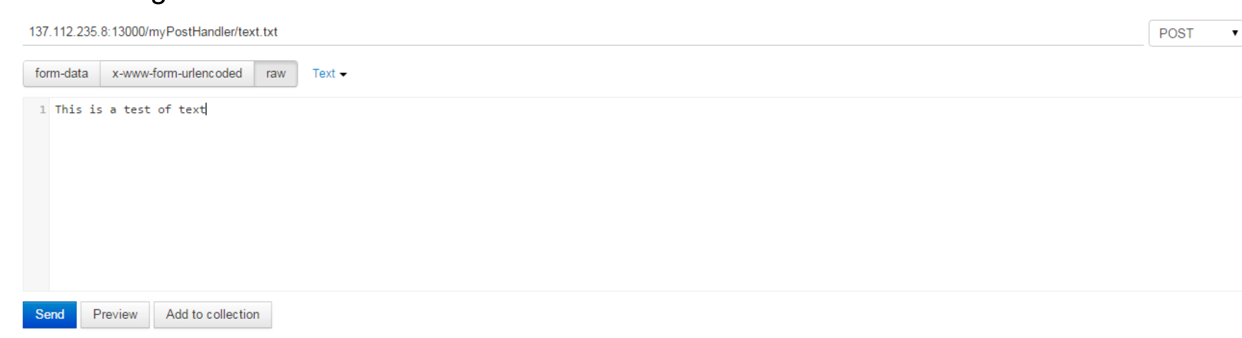


Get on browser:

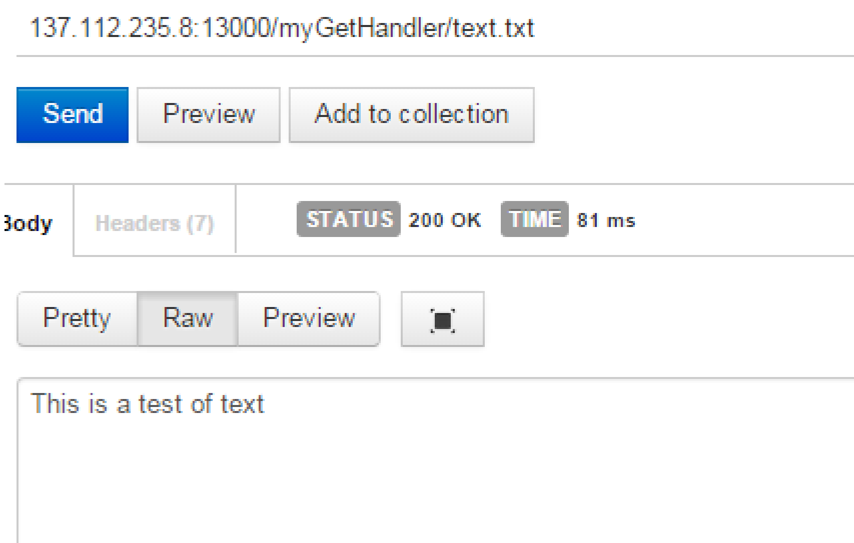




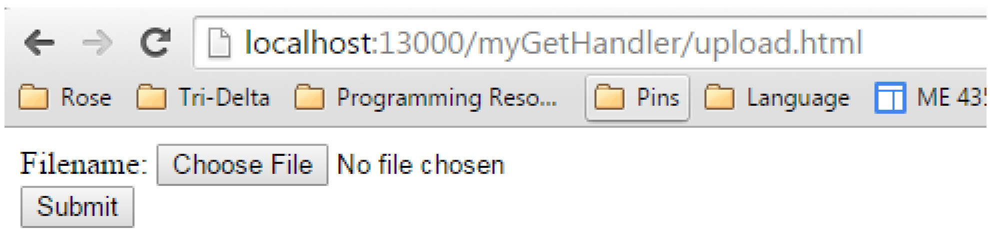
Post Testing:

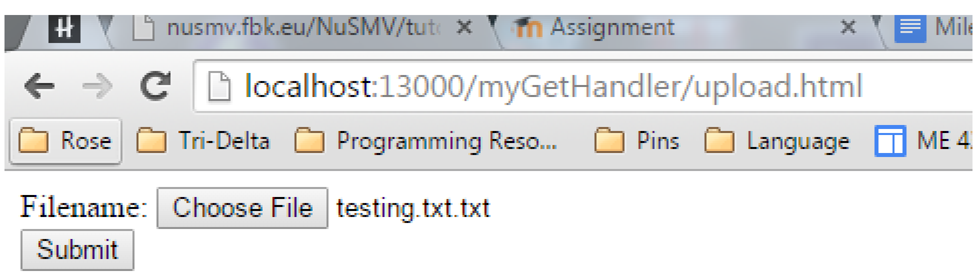


Results of Post as determined by GET.

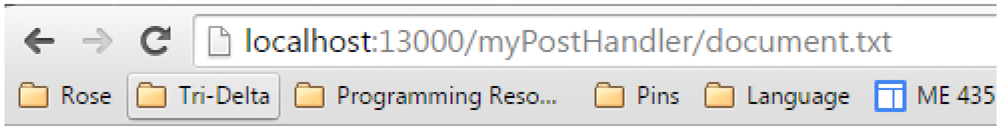


Post on Browser:





Response is an error since plugin does not have a GET request defined.




```
HTTP/1.1 400 Bad Request
Date: Mon May 04 15:32:15 EDT 2015
Provider: Chandan R. Rupakheti
Connection: Close
Server: SimpleWebServer(SWS)/1.0.0 (Windows 7/6.1/amd64)
```

Resulting document.txt in web folder. (Note document.txt is hard coded because we could not determine how to dynamically change the action of the form to include the filename of the uploaded document.)