

Change History

Revision Milestone 2:

- Modified architecture to follow Servlets and Plugin Architecture
- Updated Architecture and UML Diagrams
- Added testing section for Servlet Features
- Added explanation of extension mechanism
- Feature Listing
- Expanded improvement section

Brief Explanation of Milestone Requirements

W1 – GET Requests

GET Requests are handled by either the default implementation in the Servlet abstract class, or if overridden by the handle method, then they can be handled by the Plugin implementer. The default implementation of the GET request behavior is always available by calling the Servlet superclass handle method. GET requests are specified as accepted by a servlet by the accepted methods function. By default they are accepted. Overriding this behavior changes the accepted methods.

W2 – POST Requests

POST Requests are handled in the same fashion as GET requests. The single point of entry through the handle method.

W3 – PUT Requests

PUT Requests are handled in the same fashion as GET requests. The single point of entry through the handle method.

W4 – DELETE Requests

DELETE Requests are handled in the same fashion as GET requests. The single point of entry through the handle method.

The architecture allows for the plugin developers to either take advantage of the base system, with standard behavior for PUT, POST, GET, DELETE as implemented in Milestone 1, or to extend and override the handling of any or all of these methods.

P1 – Dynamic Loading

The Plugins Directory located at the execution directory for the web server, not necessarily within the hosted directory, is monitored for the addition of any new plugin files. Plugin files are bundled jars which have at least one class implementing the Plugin interface provided with our server. These plugins are loaded at startup from this directory and any new added during runtime are also loaded.

E1 – Root Context and Configurable Route

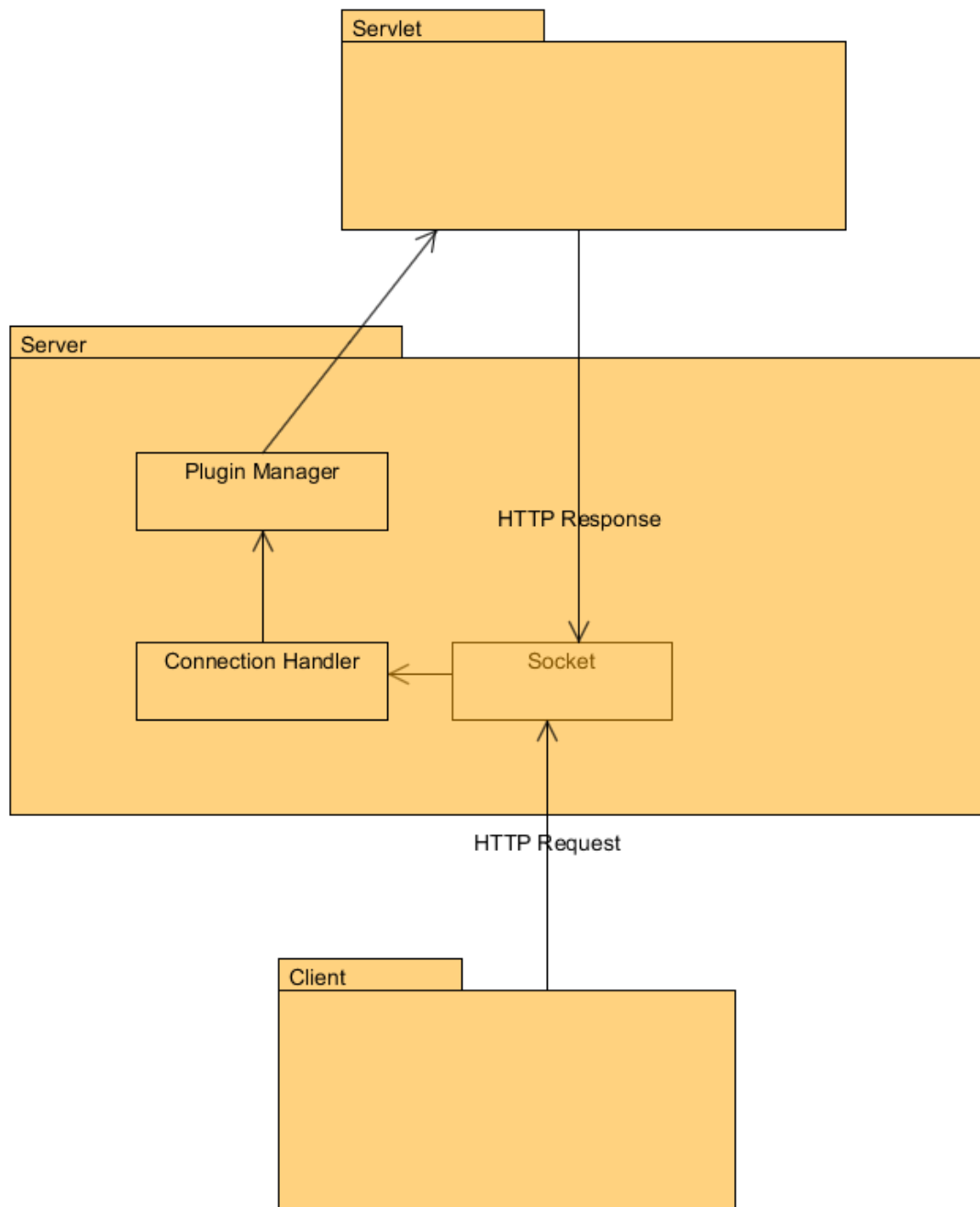
Each plugin specifies what servlets that it has at load time. Once the plugin is loaded, the servlets are requested from it. These servlets are registered by our server by using the configurable name of the plugin followed by the configurable name of the servlet. I.E. if we have a plugin named Log with developer specified name LogHandler and a servlet within that plugin of PostHandler registered as

post by the developer then the servlet will be available at <server path>/LogHandler/post for clients. The name of the plugin and the servlets are managed by the developer that must implement the respective methods to return these names.

This does mean that there can be a conflict if there is more than one plugin with the same name and a servlet with the same name in each plugin. In this conflict case, the server will route all requests to the servlet registered last.

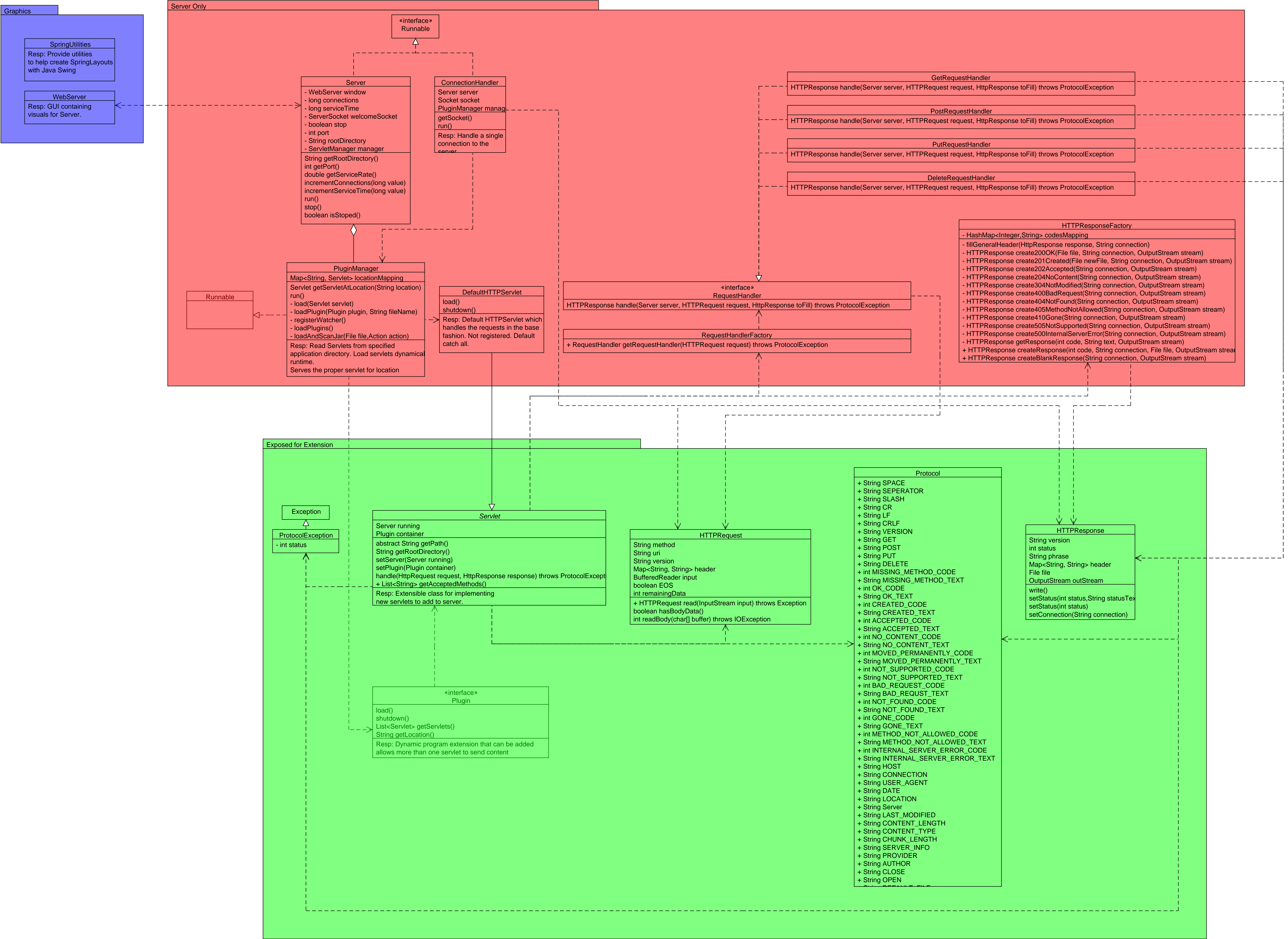
An example of this in action will be shown during the testing section with the plugins that we have implemented.

Architecture Diagram



Changes from previous: There is now a servlet layer below the Server layer that handles requests.

The UML for our server follows.



Feature Listing

GET Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

POST Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

PUT Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

DELETE Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

Dynamic Loading: Design is was done by Caleb and Matt. Implementation was completed by Matt.

Root Context and Configurable Route: Design is was done by Caleb and Matt. Implementation was completed by Matt.

Testing Milestone 2

Tests for Milestone 2 were made using Postman REST client.

Test 1: GET Request with Refactored Default Servlet

Request: No File specified

Expected Behavior: Index Page returned.

Actual Behavior: Index Page returned.

Images

Preview Limitations

GET HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS

200 OK

TIME

21 ms

Pretty

Raw

Preview



JSON

XML

```
1 <html>
2   <head>
3     <title>Test Page</title>
4   </head>
5   <body>
6     <p>Test Page Successful!</p>
7   </body>
8 </html>
```

Test 2: POST Request with Refactored Default Servlet

Test: Post to create new file

Expected Behavior: New file is created with body contents specified by POST

Actual Behavior: New File is created with body contents specified by POST

Preview [Limitations](#)

POST /new.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

hello this is my text

Send

Build

Add to collection

Body

Headers (4)

STATUS

201 Created

TIME

22 ms

Pretty

Raw

Preview



JSON

XML

1

Test 3: PUT Request with Refactored Default Servlet

Put File: New File

Expected Behavior: New File is created with text from PUT request

Actual Behavior: New File is created with text from PUT Request

Preview [Limitations](#)

PUT /new1.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

this is my text to append

Send

Build

Add to collection

Body

Headers (4)

STATUS

201 Created

TIME

11 ms

Pretty

Raw

Preview



JSON

XML

1

Test 4: DELETE Request with Refactored Default Servlet

Delete: Created file from PUT request

Expected Behavior: File is deleted and 204 No Content is returned

Actual Behavior: File is deleted and 204 No Content is returned

Preview [Limitations](#)

DELETE /new1.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS

204 No Content

TIME

16 ms

Pretty

Raw

Preview



JSON

XML

1

Test 5: Plugin Architecture

Explanation: The Parrot Plugin is a plugin developed and designed to echo back the contents of any POST request to the client. It rejects anything other than a POST request using the method as described earlier, by overriding the default accepted methods in its servlet post.

Parrot with POST request, returning back the content sent to it:

Preview [Limitations](#)

POST /Parrot/post HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache

this is my content to get sent back

Send

Build

Add to collection

Body

Headers (5)

STATUS

200 OK

TIME

33 ms

Pretty

Raw

Preview



JSON

XML

1 this is my content to get sent back

Parrot with GET request, returning back method not allowed because of the overridden servlet behavior

Preview [Limitations](#)

GET /Parrot/post HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS

405 Method Not Allowed

TIME

16 ms

Pretty

Raw

Preview



JSON

XML

1

Test 6: Data Plugin

This plugin implements two different servlets which handle simple data manipulation. The first servlet is the Fill servlet. This servlet fills an HTML response with the body of a PUT request if it receives it by overriding the default implementation for PUT requests. It however does pass through non-PUT requests to the default Servlet behavior and so handles GET, DELETE, and POST like the default servlet. The second servlet is the Log servlet. This servlet records the time that any GET request is made of it, and then writes this to a dedicated log file which is located in the root directory of the server. It rejects any other form of request.

Fill Servlet with Put request: overridden behavior

Preview [Limitations](#)

PUT /data/fill HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

this is my text

Send Build Add to collection

body Headers (5) **STATUS** 200 OK **TIME** 11 ms

Pretty Raw Preview   JSON XML

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Hi there</title>
5   </head>
6   <body>The user PUT the data: this is my text</body>
7 </html>
```

Fill Servlet with Post request: default behavior

Preview [Limitations](#)

POST /data/fill/hello.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

this is my text

Send Build Add to collection

Body Headers (4) **STATUS** 201 Created **TIME** 17 ms

Pretty Raw Preview   JSON XML

1

Log Servlet with Get request:

Preview [Limitations](#)

GET /data/log HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS 204 No Content

TIME 42 ms

Pretty

Raw

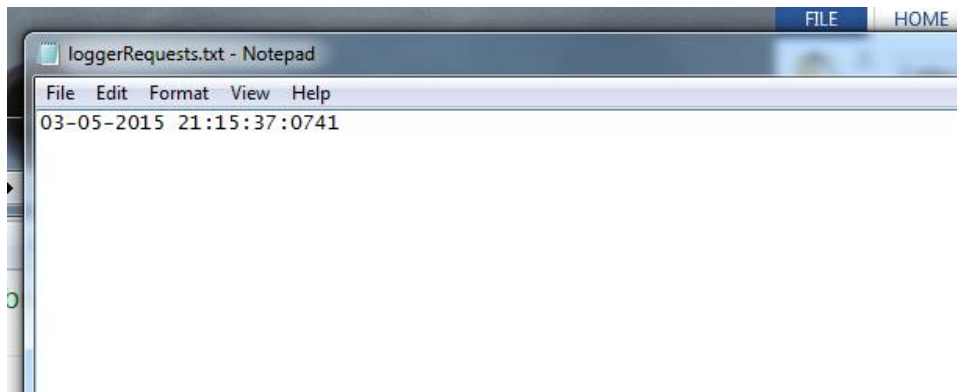
Preview



JSON

XML

1



Log Servlet non GET:

Preview [Limitations](#)

POST /data/log HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

this is my text

Send

Build

Add to collection

Body

Headers (4)

STATUS 405 Method Not Allowed

TIME 17 ms

Pretty

Raw

Preview



JSON

XML

1

Design Improvements

Milestone 2

Throttling mechanisms could be implemented to allow for the server to monitor connections that are consuming too many resources and limit them. This could be done proactively by blocking requests that are too consuming, or it could be done actively by killing threads that are consuming too many resources and blocking other clients.

Recovery mechanisms could be put into place such as request queueing and blocking if the server becomes stressed under too many requests being made at once.

If resources become a problem we may need to implement servlet cloning. Currently there is a single servlet instance hosted which handles all requests forwarded to that servlet. This works in the small scale scenario that we have been testing. We may need to multithread these servlets so that the same servlet can handle more than one request at a time, or we may need to clone the servlet in order to make each thread have a copy of it.

Preventative measures could be implemented such as monitoring connections and requests, blocking any users who appear to be malicious and consuming more connections. This would help with preventing DDOS attacks against the server.

Milestone 1

As suggested by Chandan, we should do some work with a command pattern perhaps with the generation of the responses. The way that the code currently is, all the work is done within the factory and the system is not very extensible with the responses.

Also, we could encapsulate the handling the HTTP messages into a more plugin like architecture. This would make it so that we could also handle different types of requests beyond HTTP requests without extensive modification of the system, only through adding a new plugin which respects the design we have put in place.

Design Patterns

Factory Method – We make use of factory methods in our code getting the required handler for requests. This simplifies the code and decouples the handlers from the connection and the requests.

Thread Per Connection – We make use of this concurrency pattern to make the system be able to have more efficient multi-user support. It gives each user a dedicated thread for their requests.

Builder – We use a modification of the builder pattern when we generate the responses from the requests. There is a smaller number of parameters than is usual for the builder pattern, but it allows us to hide the complexity of creating and initializing a response from the connection handler and instead encapsulate it within the response factory.

Testing Milestone 1

All testing is being done using both the Google Chrome browser (Version 42.0.2311.90 m) and the Postman REST client (v1) extension for Google Chrome, or by using the HTTP test utility supplied by Chandan Rupakheti for this project.

Get Request Successful

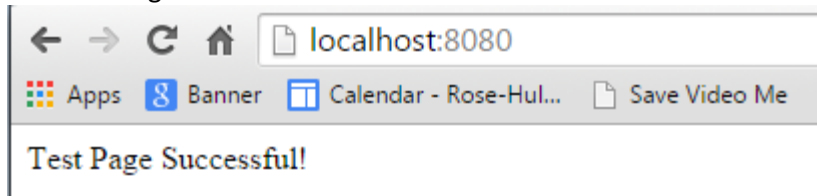
Test 1: Tested Get request on existing resource using browser

Resource: /index.html

Expected Behavior: Screen renders the test page.

Actual Behavior: Screen renders the test page.

Image:



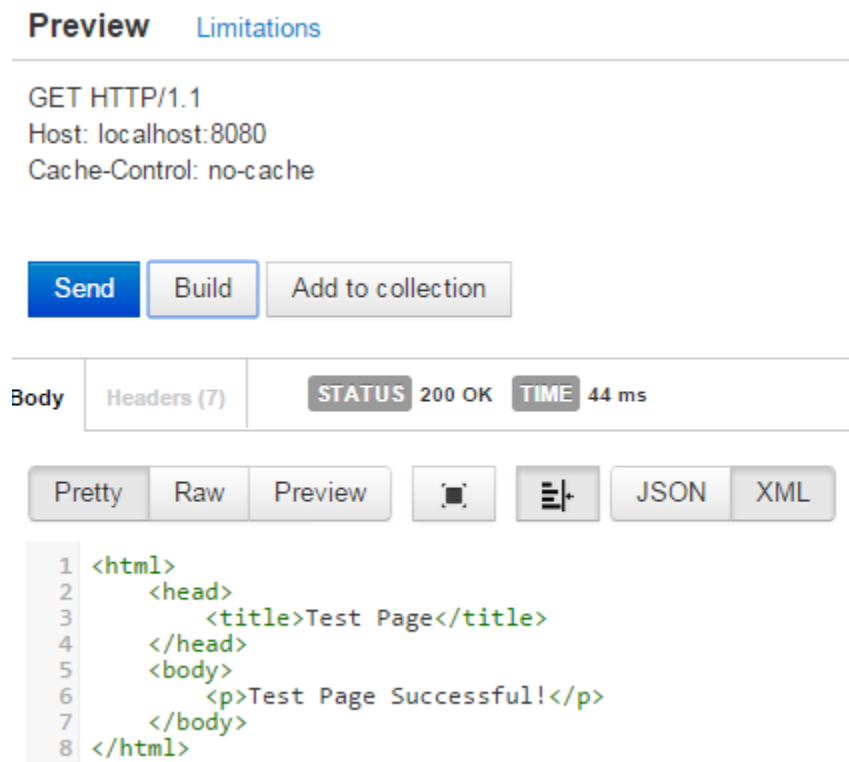
Test 2: Tested Get request on existing resource using Postman REST client

Resource: /index.html

Expected Behavior: HTML page is returned as xml and rendered in REST client with 200 OK

Actual Behavior: HTML page is returned as xml and rendered in REST client with 200 OK

Image:



Get Request Missing Resource

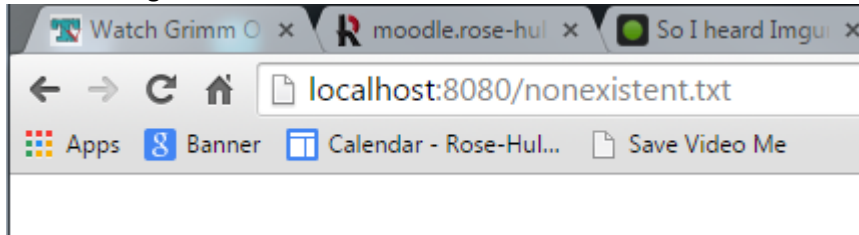
Test One: Missing resource using browser

Resource: /nonexistent.txt

Expected Result: Nothing is rendered.

Actual Result: Nothing is rendered.

Image:



Test Two: Missing resource using Postman

Resource: /nonexistent.txt

Expected Result: 404 Not Found is returned

Actual Result: 404 Not Found is returned

Image:

Preview [Limitations](#)

GET /nonexistent.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS

404 Not Found

TIME

29 ms

Pretty

Raw

Preview

■

≡

JSON

XML

1

Malformed HTTP Request

Test One: Malformed HTTP request made with HTTP Test Utility

Resource: N/A

Expected Result: 400 Bad Request Returned

Actual Result: 400 Bad Request Returned

The screenshot shows the 'HTTP 1.1 Test Client' application window. It has a title bar with standard Windows window controls. The main interface is divided into several sections:

- Connection Settings:** Contains input fields for 'Server Name' (localhost), 'Port Number' (8080), 'Sleep Time (ms)' (100), and 'Requests' (10). There are 'Connect' and 'Disconnect' buttons.
- Connection Request:** A text area showing the request details: 'GET /index.html', 'Host: localhost', 'Connection: Keep-Alive', 'User-Agent: HttpTestClient/1.0', 'Accept: text/html,text/plain,application/xml,application/json', and 'Accept-Language: en-US,en;q=0.8'.
- Connection Command:** A row of buttons: 'Generate Persistent Request', 'Generate Cache Request', 'Start DOS Attack', 'Send', 'Generate Bad Request', 'Generate Version Request', 'Stop DOS Attack', and 'Clear'.
- Connection Response:** A text area showing the response details: 'Connection Established!', 'Request Sent. Waiting for response ...', '----- Header -----', 'HTTP/1.1 400 Bad Request', 'date : Sun Apr 26 14:49:21 EDT 2015', 'server : SimpleWebServer(SWS)/1.0.0 (Windows 7/6.1/amd64)', 'provider : Chandan R. Rupakheti', 'connection : Close', '----- Body -----', and 'Socket Disconnected!'.

The malformed piece of this packet is that we did not specify protocol.

Unsupported Method

Test One: Test unsupported method using Postman

Method: OPTIONS

Expected Behavior: 405 Method Not Allowed Returned

Actual Behavior: 405 Method Not Allowed Returned

Image:

Preview [Limitations](#)

OPTIONS /test.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS 405 Method Not Allowed

TIME 39 ms

Pretty

Raw

Preview



JSON

XML

1

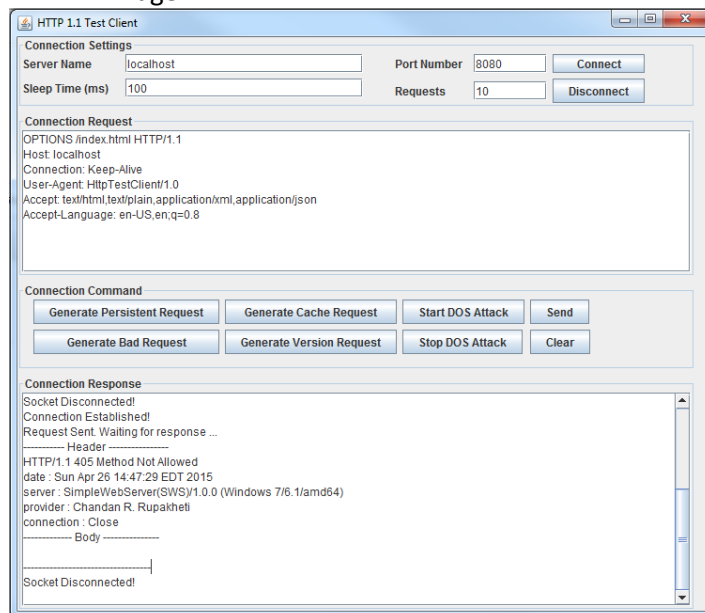
Test Two: Test Unsupported Method using HTTP Test Utility

Method: OPTIONS

Expected Behavior: 405 Method not Allowed Returned

Actual Behavior: 405 Method not Allowed Returned

Image:



HTTP Version Not Supported

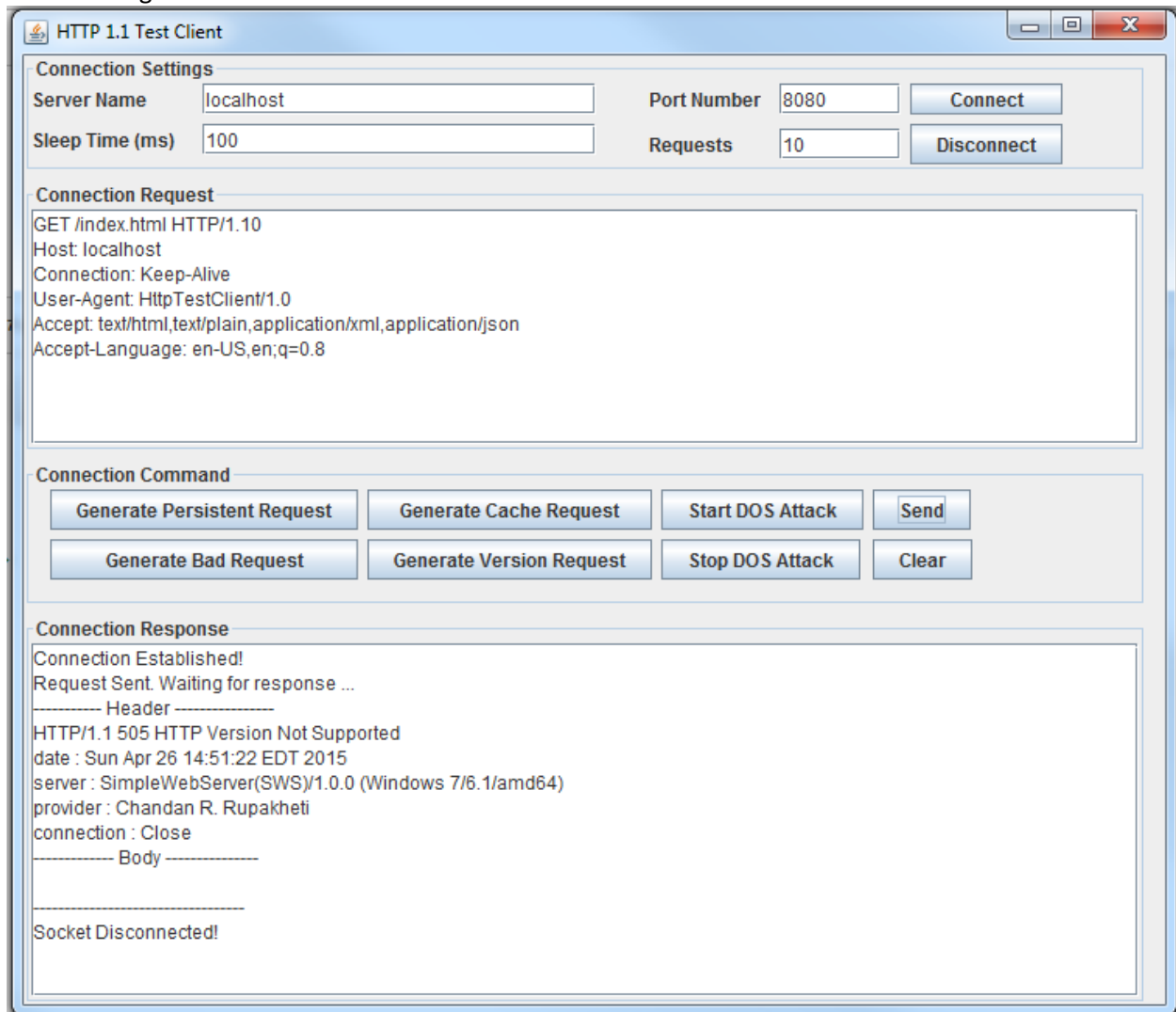
Test 1: Test version not supported using HTTP Test Client

HTTP Version: 1.0

Expected Result: 505 Version not supported returned

Actual Result: 505 Version not supported returned

Image:



The server that we have built explicitly only supports HTTP 1.1. It rejects outdated clients that are using 1.0.

Post Request

Our post request implementation writes the body of the http message to disk. This means that when form encoded data is sent, it looks strange on disk.

Test 1: Post request made from browser:

HTML Image:

```
1 <html>
2 <body>
3 <form action="http://localhost:8080/uploaded.txt" method="post" enctype="multipart/form-data">
4 <p>File: <input type="file" name="file1">
5 <p><button type="submit">Submit</button>
6 </form>
7 </body>
8 </html>
9
```

This means that it will upload said file to a file called uploaded.txt on the server.

Expected behavior: File is uploaded. When requested via GET it will be shown.

Actual Behavior: File is uploaded. Sequence below.

Image 1: Post

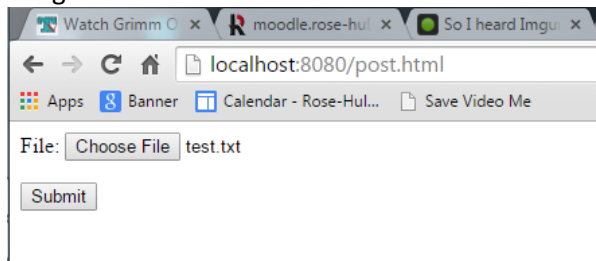
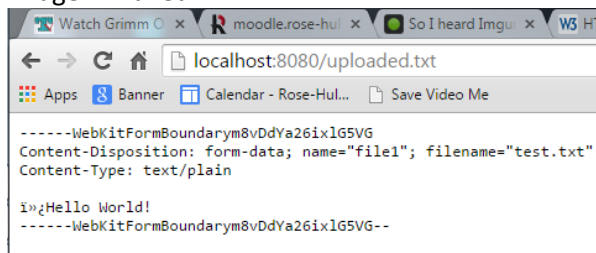


Image 2: Pulled



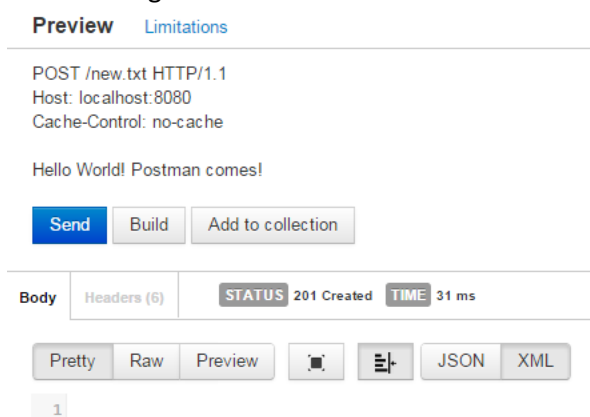
Test 2: POST made with Postman

File: new.txt

Expected Behavior: 201 Created returned and file exists.

Actual Behavior: 201 Created and file exists.

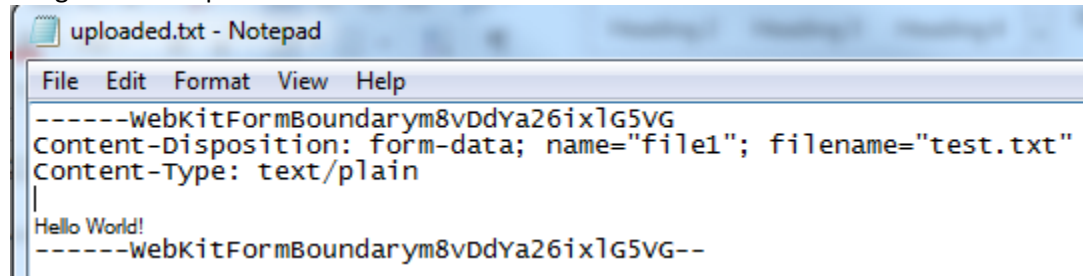
Image:



Test 3: Post made with Postman replacing file

File: uploaded.txt

Image Prior to Upload:



Expected Behavior: File Contents are replaced.

Actual Behavior: File Contents are replaced.

Image of Request:

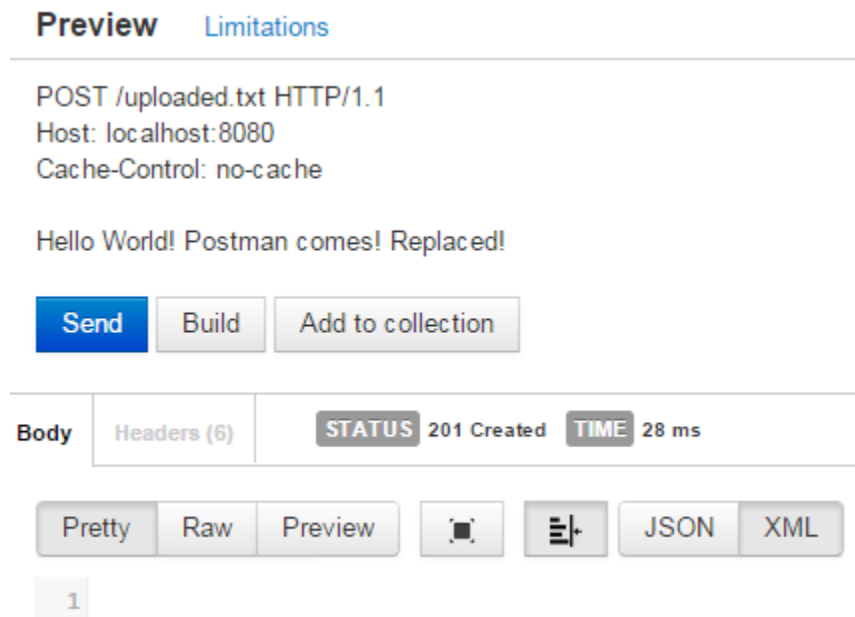
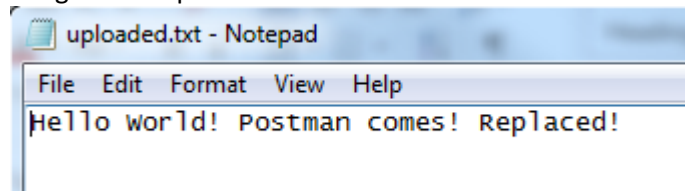


Image After Upload:



Put Request New File

Test 1: Postman with Put request and new file

File: new_put.txt

Expected Behavior: File is created and 201 Created is returned.

Actual Behavior: File is created and 201 Created is returned.

Request Image:

Preview [Limitations](#)

PUT /new_put.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Put! Postman!

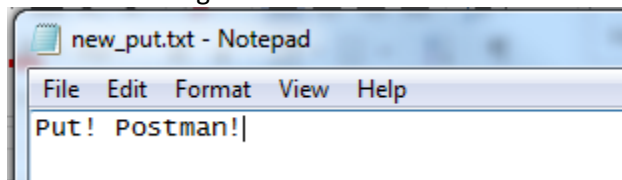
Send Build Add to collection

Body Headers (6) **STATUS** 201 Created **TIME** 56 ms

Pretty Raw Preview   JSON XML

1

File Image:



Put Request with Existing File

Test 1: Postman with existing file.

File: new_put.txt

Expected Behavior: File is appended with contents of request and 202 Accepted is returned.

Actual Behavior: File is appended with contents of request and 202 Accepted is returned.

Request Image:

Preview [Limitations](#)

PUT /new_put.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

New Line!

Send

Build

Add to collection

Body

Headers (4)

STATUS 202 Accepted

TIME 45 ms

Pretty

Raw

Preview

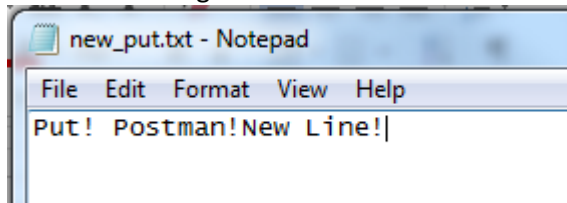


JSON

XML

1

File Image:



Delete Request on existing file

Test 1: Delete the created file new_put.txt

Expected Behavior: File is no longer in the web directory and 410 Gone is returned.

Actual Behavior: File is no longer in directory and 410 Gone is returned.

Request Image:

Preview [Limitations](#)

DELETE /new_put.txt HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache

Send

Build

Add to collection

Body

Headers (4)

STATUS 404 Not Found **TIME** 28 ms

Pretty

Raw

Preview

JSON







XML

Directory Image:

Documents library

web

Arrange by: Folder

Name	Date modified
 uploaded.txt	4/26/2015 3:50 PM
 new.txt	4/26/2015 3:47 PM
 post.html	4/26/2015 3:06 PM
 rmiclient.jar	3/25/2015 11:25 PM
 rmiserver.jar	3/25/2015 11:20 PM
 index.html	11/5/2013 10:00 PM

DELETE request with missing file

Test 1: Postman missing file

Expected Behavior: 404 Not Found is returned

Actual Behavior: 404 Not Found is returned

Image:

Preview [Limitations](#)

DELETE /new_put.txt HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache

Send

Build

Add to collection

body

Headers (4)

STATUS

404 Not Found

TIME

33 ms

Pretty

Raw

Preview



JSON

XML

1