# Carly Heibel and Michael Royappa

Milestone 1

CSSE477

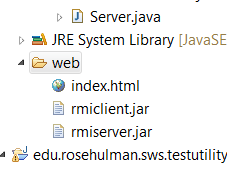
# Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Milestone** | **Description** | **Version** | **Contributor** | **Date** |
| M1 | Creation | 1.0 | Carly Heibel | 4/29/2015 |
| M1 | Screenshots from testing | 1.1 | Michael Royappa | 4/30/2015 |
| M2 | Adding content for milestone | 2.0 | Carly Heibel | 5/8/2015 |
| M3 | Adding plans for tactics | 3.0 | Carly and Michael | 5/8/2015 |
| M3 | Adding test data | 3.1 | Carly | 5/10/2015 |
|  |  |  |  |  |

Milestone 1 – Simple Web Server

# POST Request

### File Hierarchy before POST:



### POST Request:

POST /temp.txt HTTP/1.1

content-length: 463

referer: http://localhost:8080/upload.html

accept-language: en-US,en;q=0.8

origin: http://localhost:8080

host: localhost:8080

connection: keep-alive

content-type: multipart/form-data; boundary=----WebKitFormBoundaryEzjCwl5nt4xFk9AS

cache-control: max-age=0

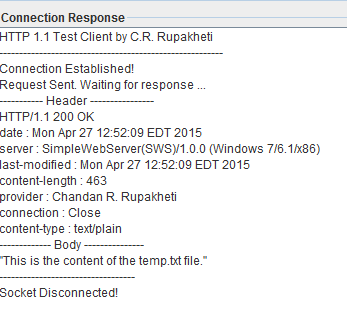
accept-encoding: gzip, deflate

accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

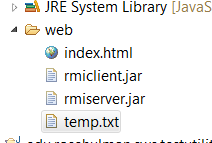
User-Agent: HttpTestClient/1.0

"This is the content of the temp.txt file."

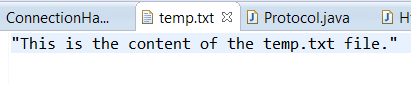
### Output of POST in TestClient:



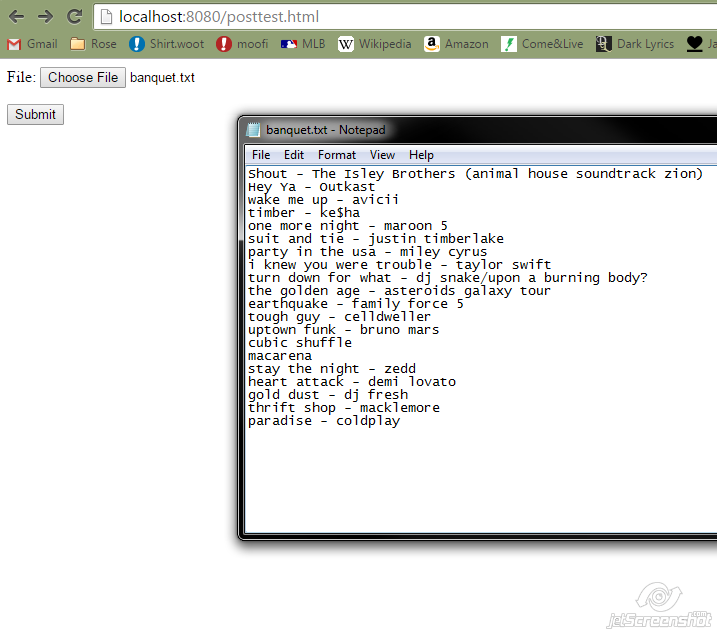
### File Hierarchy after running POST Request:



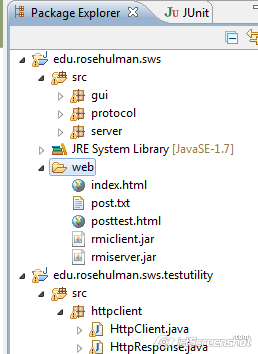
### File contents after successful POST:



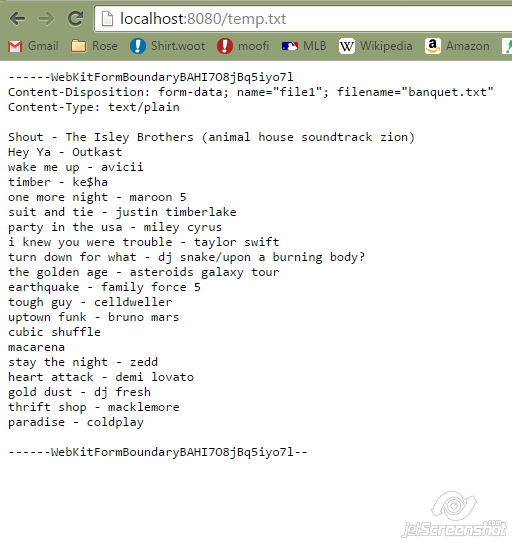
### File to be posted to /temp.txt in browser:

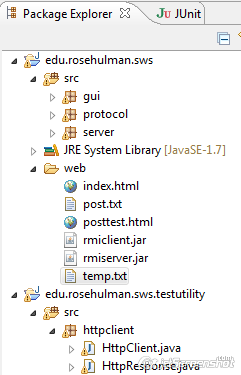


### Web directory before POST:



### File contents in browser and directory after clicking “Submit” and firing POST:





# PUT Request

### PUT Request:

PUT /temp.txt HTTP/1.1

content-length: 60

referer: http://localhost:8080/upload.html

accept-language: en-US,en;q=0.8

origin: http://localhost:8080

host: localhost:8080

connection: keep-alive

content-type: multipart/form-data; boundary=----WebKitFormBoundaryEzjCwl5nt4xFk9AS

cache-control: max-age=0

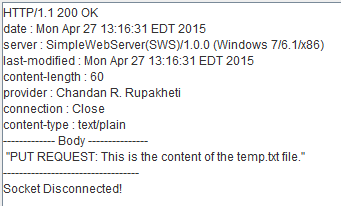
accept-encoding: gzip, deflate

accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

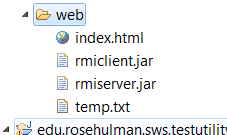
User-Agent: HttpTestClient/1.0

"PUT REQUEST: This is the content of the temp.txt file."

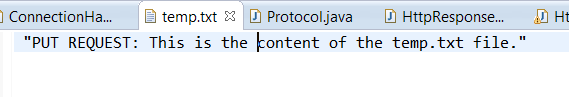
### Output of PUT Request in TestClient (deleted temp.txt, so it should re-create it):



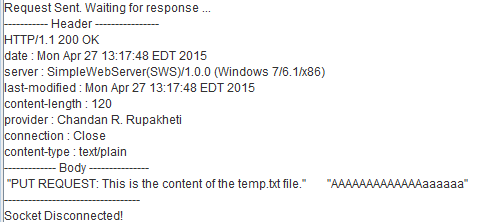
### File Hierarchy after running PUT with no existing temp.txt:

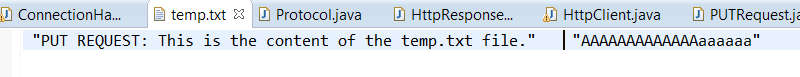


### Contents of new temp.txt:

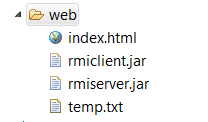


### PUT Request after file already exists:



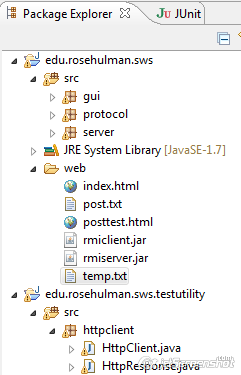


### File Structure doesn’t change:

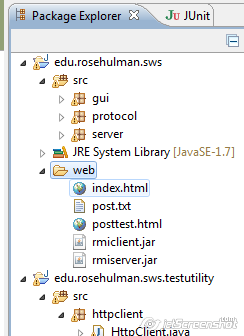


# DELETE Request

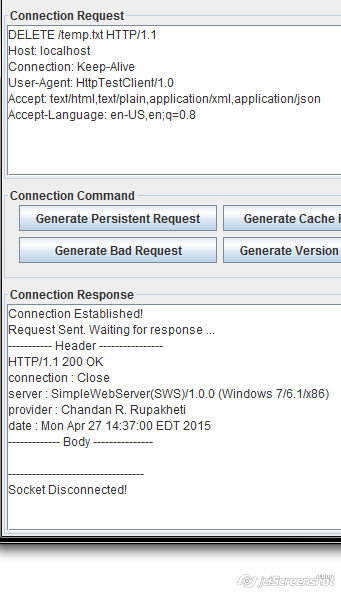
### Directory before DELETE:



### Directory after DELETE:



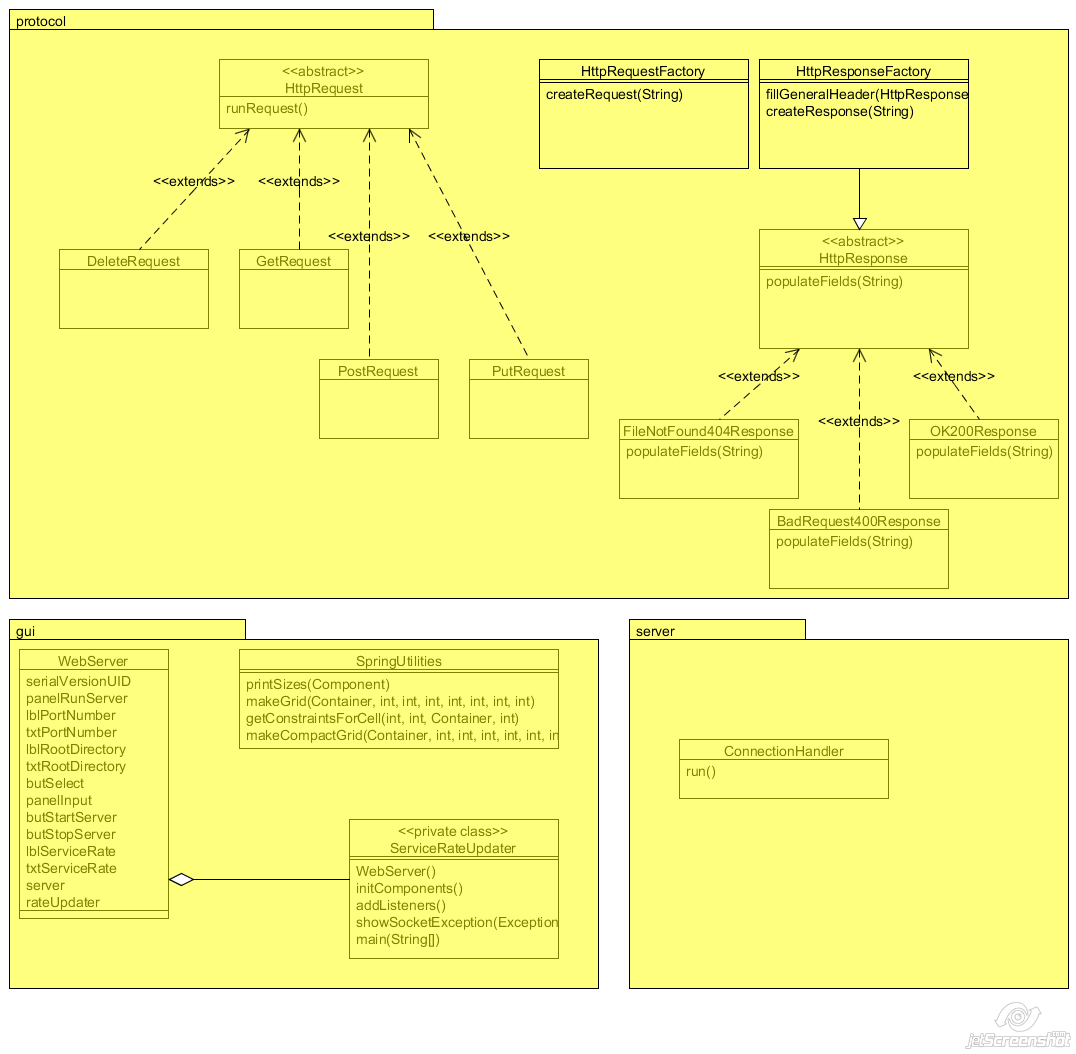
### Request and response text:



# Improvements

We would like to implement a HttpRequestFactory to streamline creation of HttpRequest objects. We would also like to implement some more elegant method of handling the many different response cases in GETRequest.java.

# Class Diagram



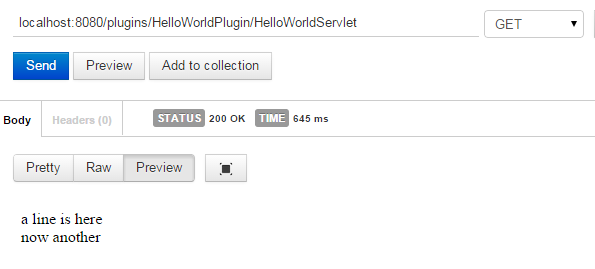
Milestone 2 – Application Server (Servlets)

# Feature Listing

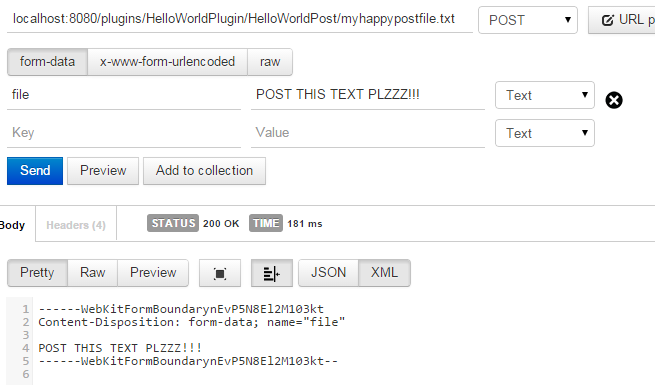
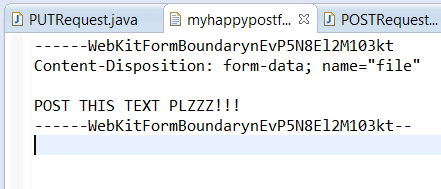
|  |  |
| --- | --- |
| **Feature** | **Contributor** |
| W-1: GET Requests | Carly and Michael (Pair programming) |
| W-2: POST Requests | Carly and Michael (Pair programming) |
| W-3: PUT Requests | Carly and Michael (Pair programming) |
| W-4: DELETE Requests | Carly and Michael (Pair programming) |
| P-1: Dynamic Loading | Carly and Michael (Pair programming) |
| E-1: Root Context and Configurable Route | Carly and Michael (Pair programming) |
| D-1: Test result | Carly |
| D-2 Future Improvements | Carly and Michael |
|  |  |

# Test Report

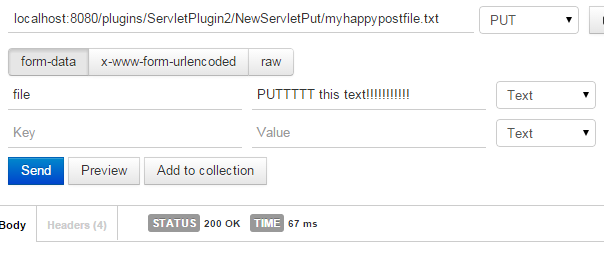
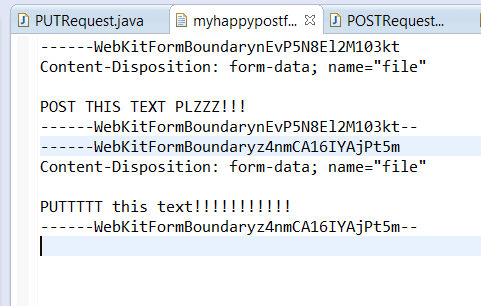
### GET Request



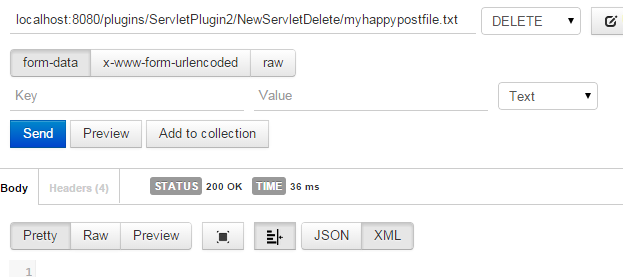
### POST Request

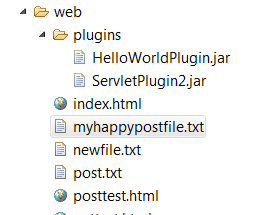
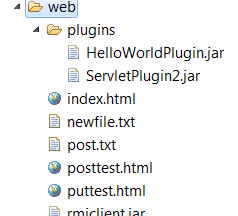


### PUT Request



### Delete Request

Before: Request: After:



# Future Improvements

* Sending file text in 200OK responses for PUT
* Separating out more specific errors to send more accurate error responses
* Adding more information to headers

Milestone 3 – Application Server (Tactics)

# Feature Listing

|  |  |
| --- | --- |
| **Feature** | **Contributor** |
| Tactics Plan | Carly and Michael (Pair brainstorming) |
| Testing and implementing improvements | Carly |
| Attempted to implement DDOS defenses | Michael |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Test Scenarios

### Availability

A1 - No Down Time

A1.1 – Scenario

|  |  |
| --- | --- |
| Source | User |
| Stimulus | Client Request |
| Environment | System has thrown an exception |
| Artifact | Web Server |
| Response | Response sent back with appropriate code |
| Response Measure | Server has no down time |

A1.2 – Test plan

1. Send server GET request with no filename in the URL

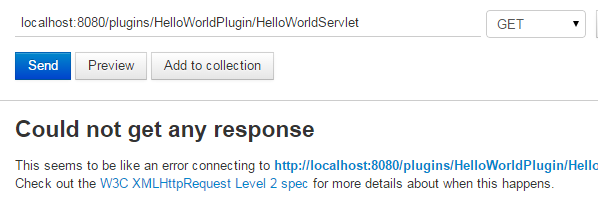
2. Send server POST request with too much in the URL

3. Send server request for servlet that does not exist

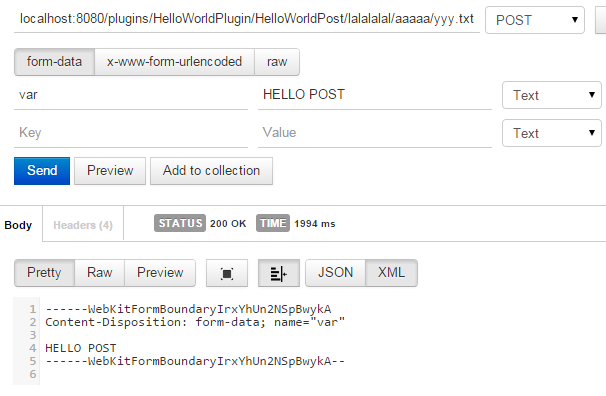
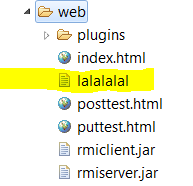
4. Send server request for plugin that does not exist

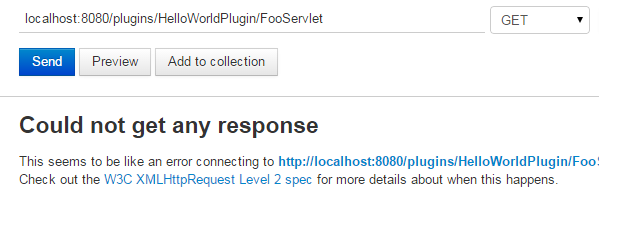
A1.3 – Baseline Test Results

1. Server failed to send a response back to the user (Array index out of bounds exception)

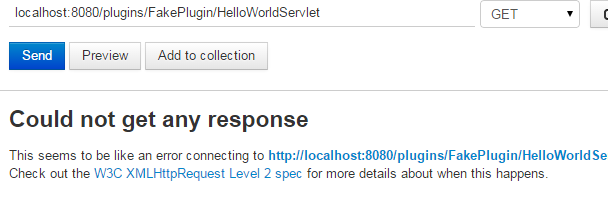


2. Server uses incorrect filename (lalalalal instead of yyy.txt) but does not crash:

3. Server failed to send a response back to the user (Array index out of bounds exception)

4. Server failed to send a response back to the user (Array index out of bounds exception)



A1.4 – Improvement Tactics

Add more try/catch statements to handle specific exceptions and return the most appropriate response based on which type of exception was thrown.

A1.5 – New Results & Conclusion

1. Server recognizes that it is a GET request, thus needing no filename, and successfully completes the request, returning a 200 OK response

2. Server sent proper response (400 Bad Request) back to the client

3. Server sent proper response (400 Bad Request) back to the client

4. Server sent proper response (400 Bad Request) back to the client

Conclusion: Implementing these additional requests (as well as 500 Internal Server Error) allow the server to still function as normal by responding to the client with a response packet rather than hanging up and not sending any response back to the client at all. These informative error-code requests allow the server not to take any down time from the user and keep the user informed.

A2 – DDOS Attack

A2.1 – Scenario

|  |  |
| --- | --- |
| Source | Denial of Service Launcher |
| Stimulus | Large number of requests per second |
| Environment | Under Normal Operations |
| Artifact | Web Server |
| Response | Ignore malicious requests |
| Response Measure | System continues running |

A2.2 – Test plan

1. Run Denial of Service Launcher with 200 requests/sec

A2.3 – Baseline Test Results

1.

A2.4 – Improvement Tactics

Limit repeated requests from each sender by using a blacklist

A2.5 – New Results & Conclusion

1.

Conclusion:

### Performance

P1 – Too Long to Run

P1.1 – Scenario

|  |  |
| --- | --- |
| Source | User/Script |
| Stimulus | Client Request |
| Environment | Servlet is processing a request |
| Artifact | Plugin-provided servlet |
| Response | 408 Request Timeout response sent back |
| Response Measure | No request will take up too much processing time |

P1.2 – Test plan

1. Create a servlet that sleeps for 30 sec and run it

P1.3 – Baseline Test Results

1. Servlet successfully runs for 30 sec (which is over our server’s cap of 10 seconds)

P1.4 – Improvement Tactics

Impose a timeout limit of 10 seconds for all requests. If this limit is reached, cancel the request and send back a 408 response

P1.5 – New Results & Conclusion

1. Servlet runs (via spawning it on a thread) until the time cap, and then it is terminated. If it was short enough, it completes and returns a 200 OK response. If it had to be cut short, it stops and returns a 408 Request Timeout response.

Conclusion: Using threading allows the servlet request to be terminated if it is taking up too much processing time. This allows other requests to be served without waiting for the long request to finish if it is not sharing the server.

P2 – DDOS Attack

P2.1 – Scenario – SEE A2.1

P2.2 – Test plan

1. Run Denial of Service Launcher with 200 requests/sec and record how many it handles

P2.3 – Baseline Test Results

1.

P2.4 – SEE A2.4

P2.5 – New Results & Conclusion

1.

Conclusion:

### Security

S1 – Large File Size

S1.1 – Scenario

|  |  |
| --- | --- |
| Source | User/Script |
| Stimulus | Put or Post Request |
| Environment | Under Normal Operations |
| Artifact | Web Server |
| Response | 413 Request Entity Too Large Response |
| Response Measure | Large file requests will be denied |

S1.2 – Test plan

1. Set memory limit for run configuration to 128MB and run a request trying to post/put file that is 129MB.

S1.3 – Baseline Test Results

1.

S1.4 – Improvement Tactics

Impose a file size limit of 128MB for all requests. If this limit is reached, cancel the request and send back a 413 response

S1.5 – New Results & Conclusion

1.

Conclusion:

S2 – DDOS Attack

S2.1 – Scenario – SEE A2.1

S2.2 – Test plan

1. Run Denial of Service Launcher with 200 requests/sec and record table of users and respective requests processed per second

S2.3 – Baseline Test Results

1.

S2.4 – SEE A2.4

S2.5 – New Results & Conclusion

1.

Conclusion:

# Future Improvements

* Print the stack trace of exceptions in the body of responses