Web Server Report  
Team RAJ – Angelica Rodriguez, John Krasich CSSE 477

Table of Contents

[**Milestone 1** 3](#_Toc418580294)

[**Architecture Diagram** 3](#_Toc418580295)

[**Detailed Design** 4](#_Toc418580309)

[**Further Improvements** 5](#_Toc418580310)

[**Test Report** 6](#_Toc418580311)

[**Testing Utility** 6](#_Toc418580312)

[**Web Browser** 10](#_Toc418580313)

[**Change History – MS2** 11](#_Toc418580314)

[**Updated Architecture Diagram** 11](#_Toc418580315)

[**Updated Detailed Design** 12](#_Toc418580331)

[**Brief Description** 12](#_Toc418580332)

[**Feature Listing & Assignment** 13](#_Toc418580333)

[**Test Report** 14](#_Toc418580334)

[**Plugin Addition** 14](#_Toc418580335)

[**POST** 15](#_Toc418580336)

[**GET** 16](#_Toc418580337)

[**PUT** 17](#_Toc418580338)

[**DELETE** 18](#_Toc418580339)

[**Future Improvements** 19](#_Toc418580340)

[**Change History – MS3** 20](#_Toc418580341)

[**Updated Architecture Diagram** 21](#_Toc418580342)

[**Updated Detailed Design** 22](#_Toc418580343)

[**Brief Description** 22](#_Toc418580344)

[**Tactics/Feature Listing** 23](#_Toc418580345)

[**Architectural Evaluation and Improvements** 24](#_Toc418580346)

[**Availability** 24](#_Toc418580347)

[**Performance** 24](#_Toc418580348)

[**Security** 24](#_Toc418580349)

[**Future Improvements** 26](#_Toc418580350)

# **Milestone 1**

## **Architecture Diagram**

**Protocol**

**Server**

**Client**

Protocol

Server

Web browser

Connection Handler

HTTP Response

Web Server GUI

HTTP Request

400

200

404

304

Request

Response

PUT

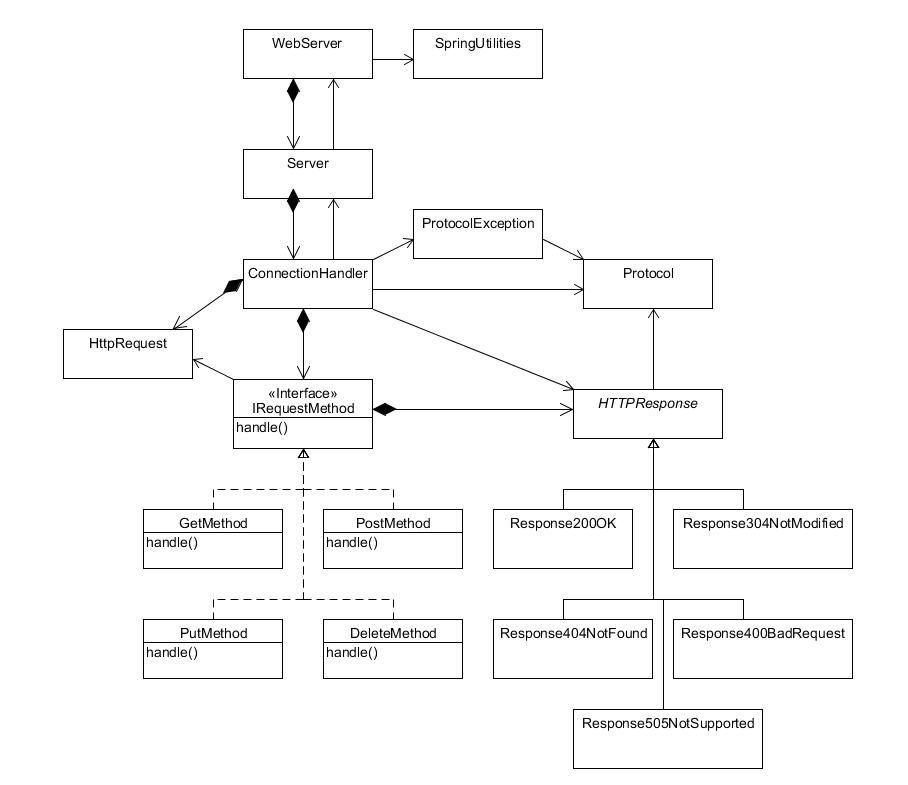
POST

505

GET

DELETE

## **Detailed Design**



Our refactoring of the web server utilized the following design patterns:

**Strategy Pattern** – The IRequestMethod interface allows for the various implementations of request handling to be completed in unique classes. This way, additional request handling can implanted with minimal changes to the ConnectionHandler class – simply add the new request to the ConnectionHandler’s map of request methods.

**Bridge Pattern** – The abstract HTTPResponse class is used by the ConnectionHandler to write the generated response back to the client. However, the responses vary depending on the response code. Using a bridge pattern, each different response’s implementation can be handled in separate classes without the ConnectionHandler needing to have any knowledge of how it is implemented.

## **Further Improvements**

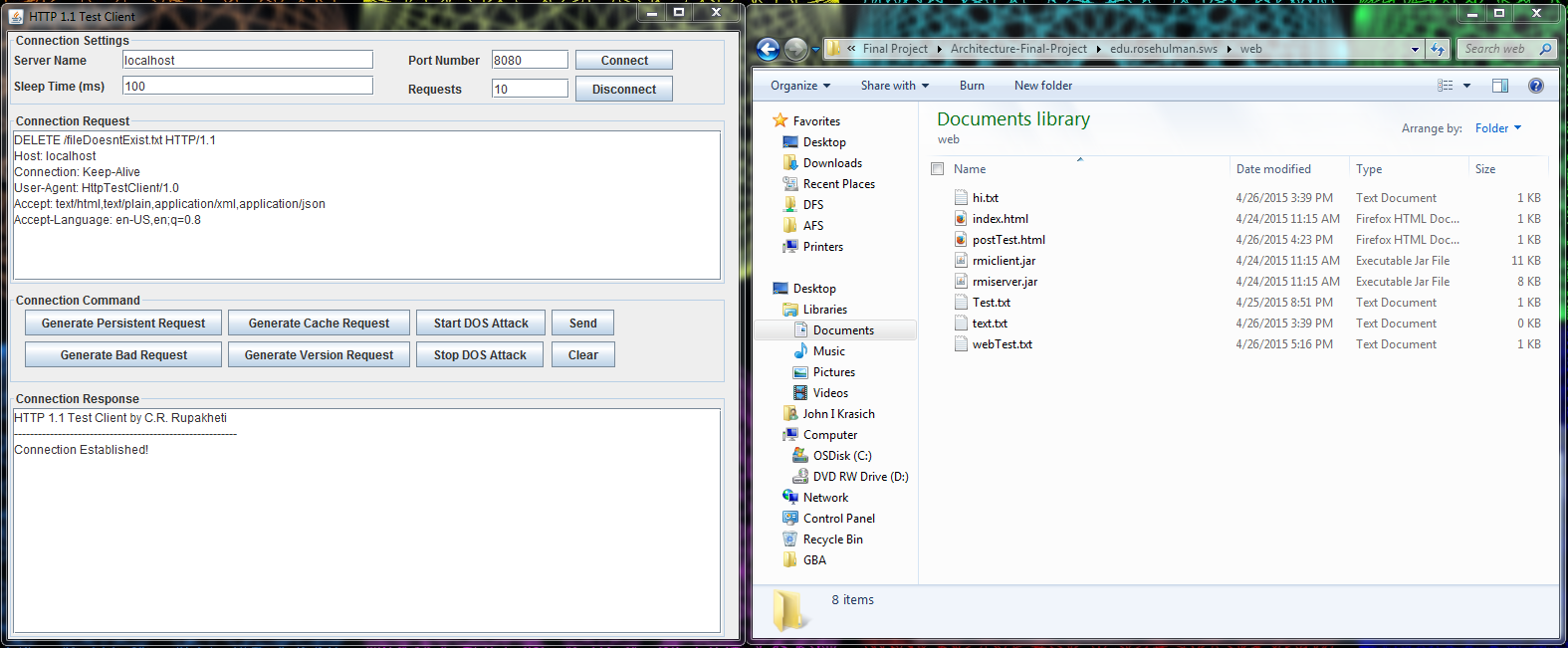
One area that can be further refactored would involve dividing the responsibilities of the ConnectionHandler class separately between requests and responses. The “run” method is rather long – breaking this up into different methods (or different classes) would make the code much more organized and easier to understand.

## **Test Report**

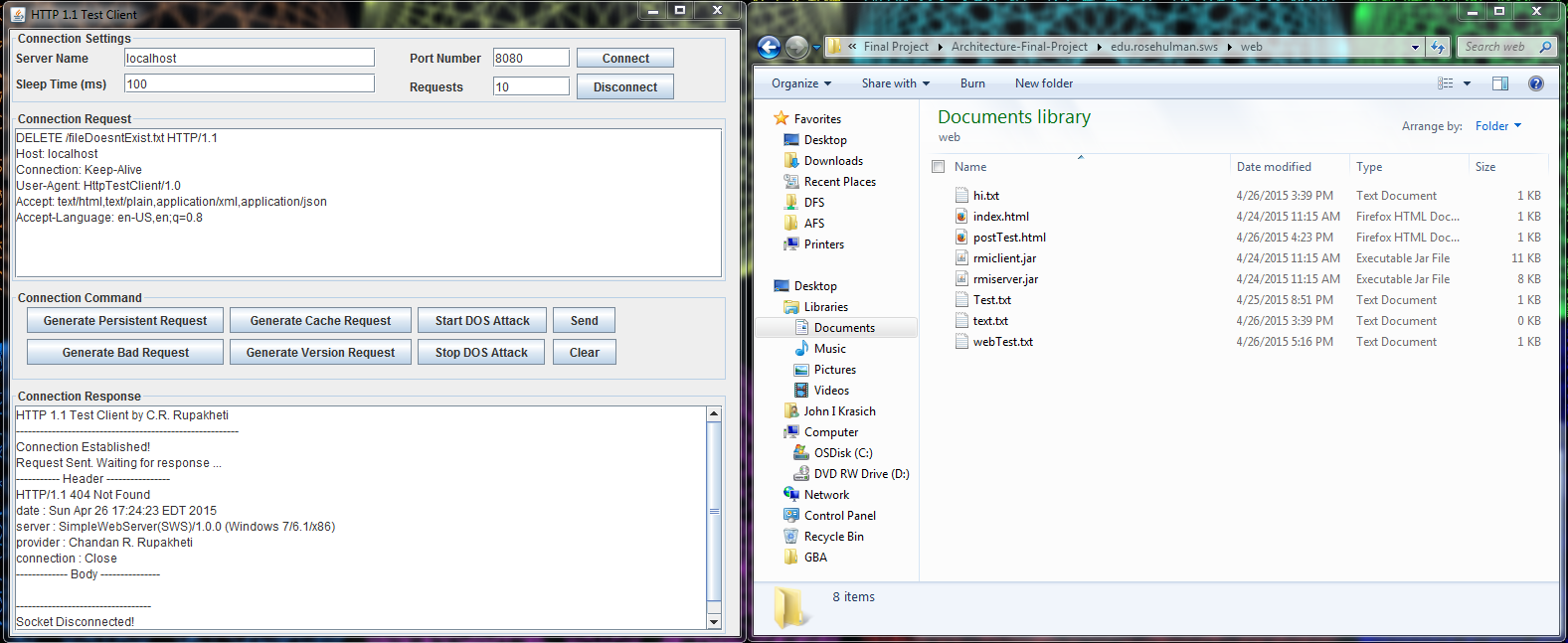
### **Testing Utility**

#### **DELETE of Non-Existent File**

Before:

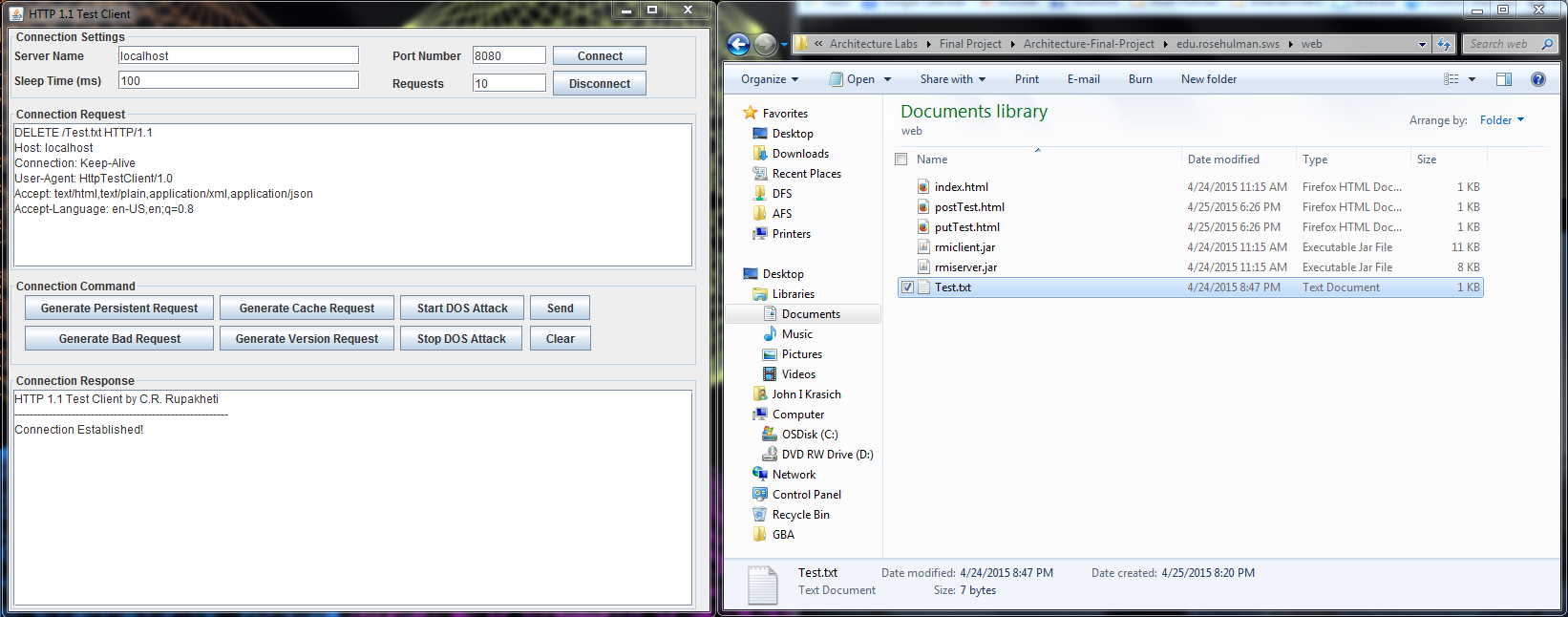


After: Response 404 Not Found



#### **DELETE**

Before:

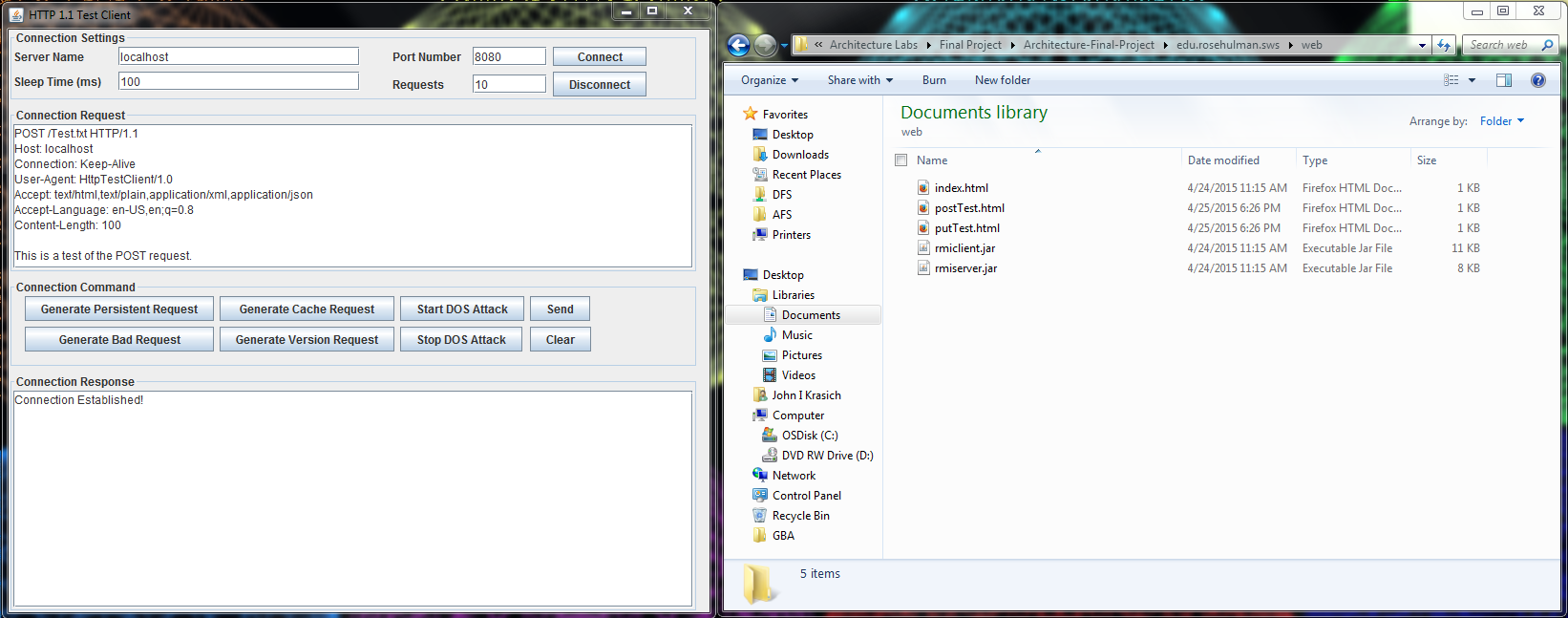


After: Response – 200 OK. File Test.txt has been deleted successfully

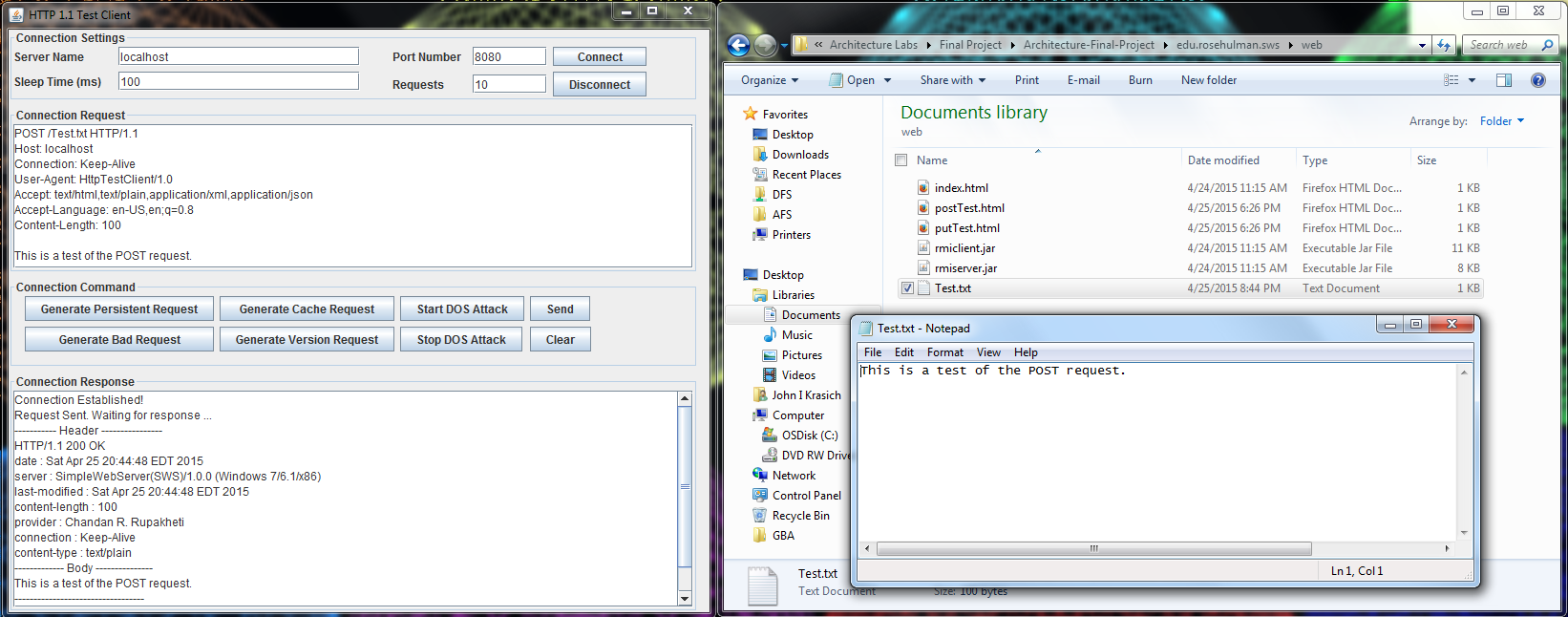


#### **POST**

Before:

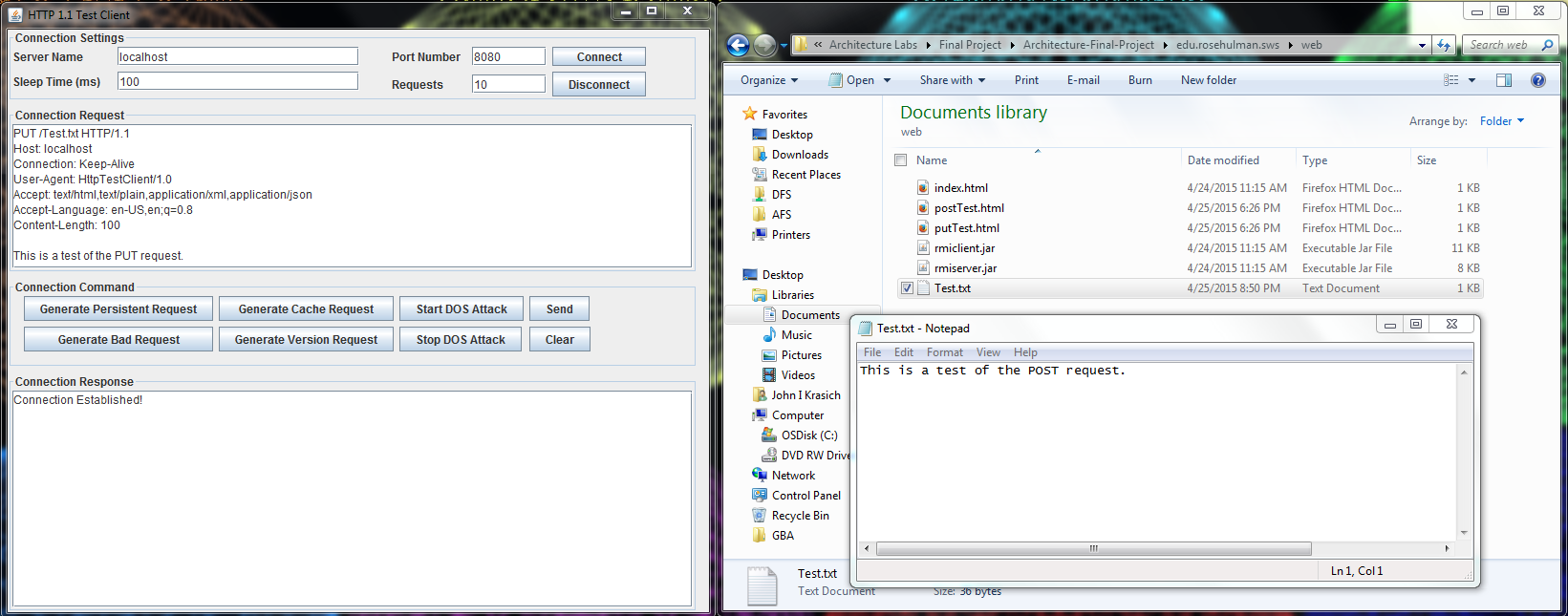


After: Response 200 OK. The file Test.txt has been created and filled with the body of the request.

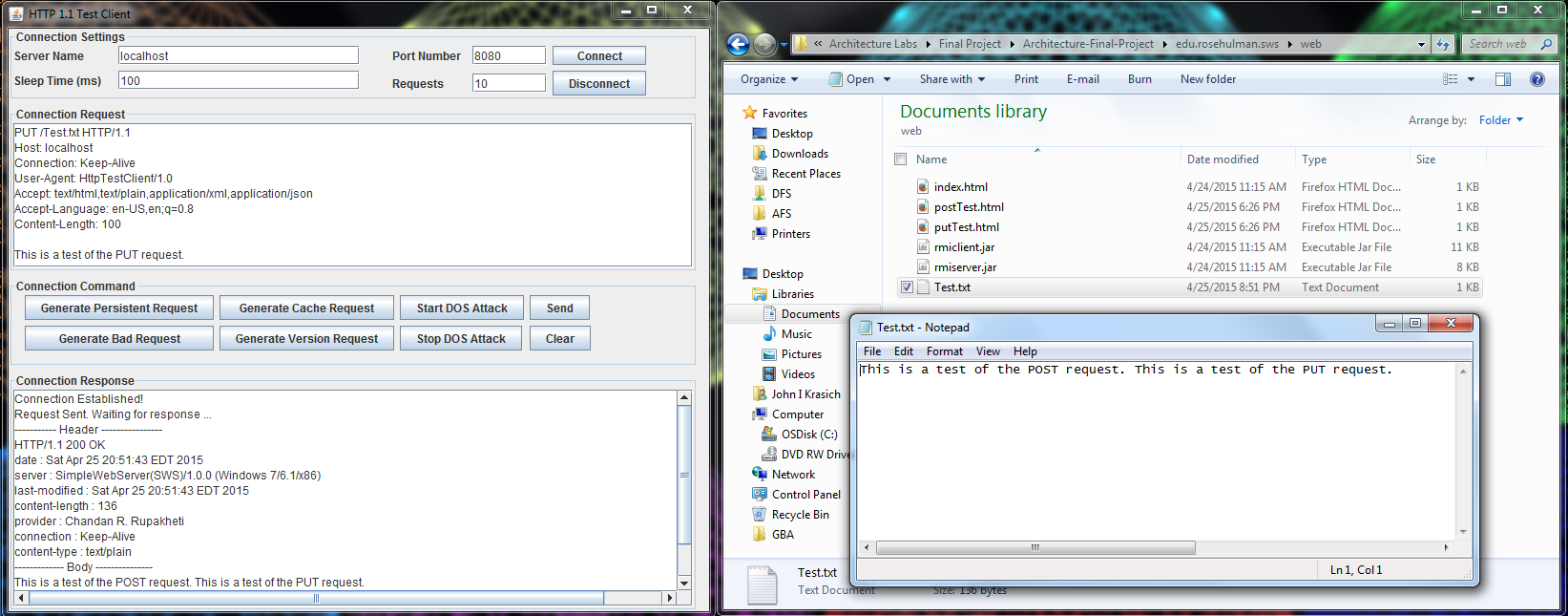


#### **PUT**

Before:



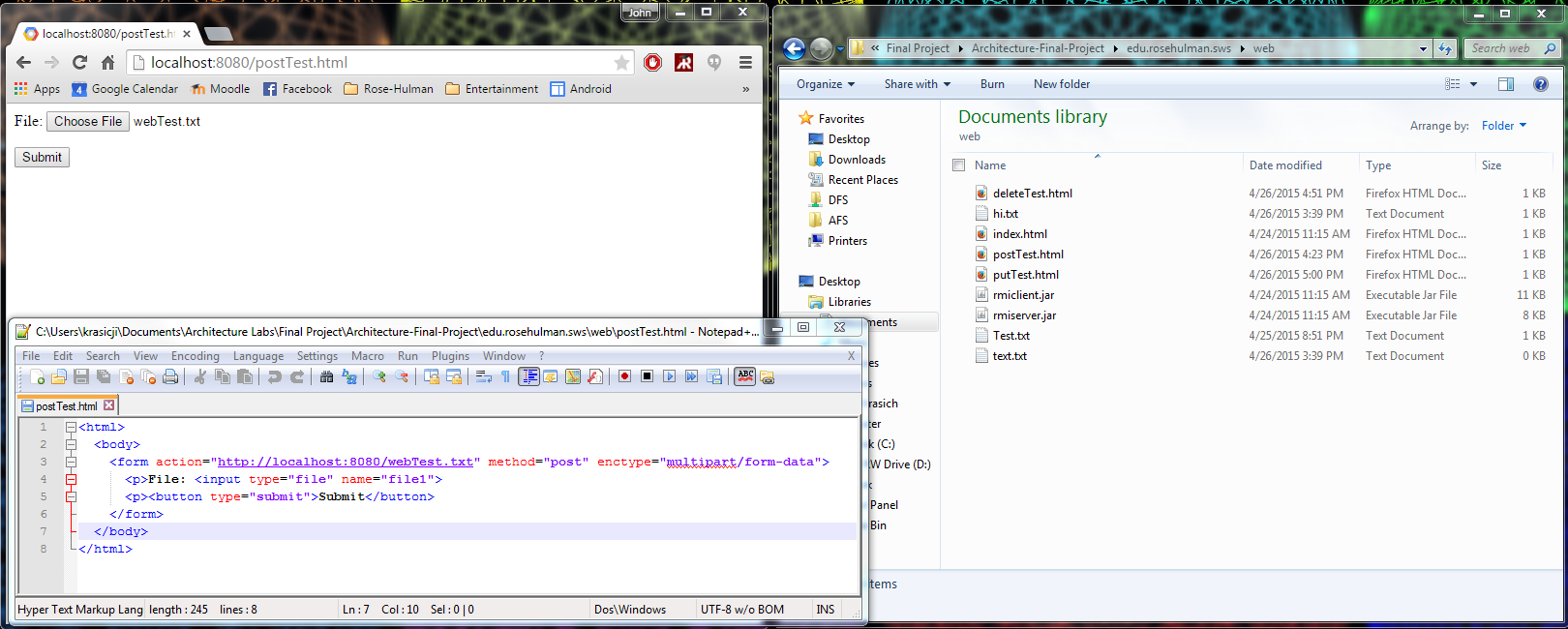
After: Response 200 OK. The body of the request was appended to the Test.txt file.

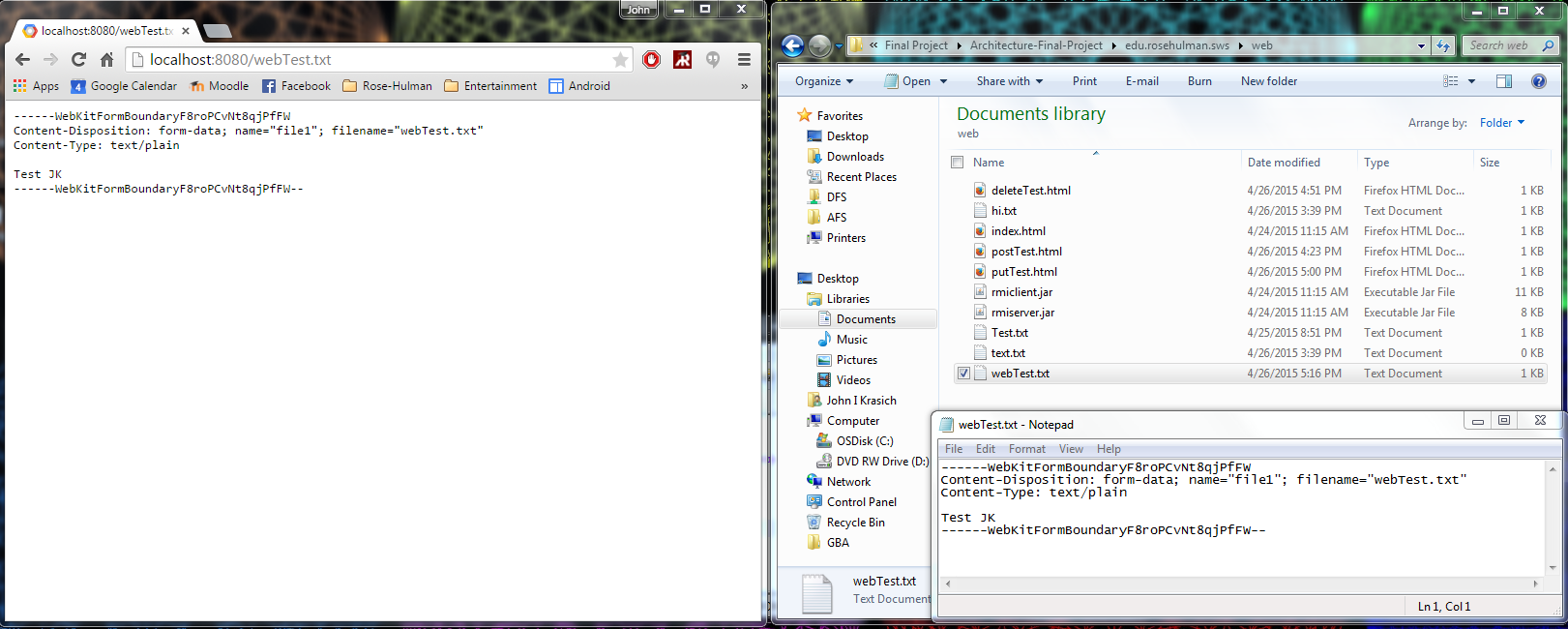


### **Web Browser**

#### **POST**

Before:



After: Response 200 OK – the text of the file was written into webTest.txt and returned as the body in the response.

# **Change History – MS2**

## **Updated Architecture Diagram**

**Server**

## 

**Protocol**

**Client**

Connection Handler

Server

Protocol

Web browser

HTTP Response

Plugin Handler

304

200

400

Servlet

Web Server GUI

404

StaticPUT

StaticPOST

Request

Response

505

StaticDELETE

StaticGET

UserServlet

## **Updated Detailed Design**

### **Brief Description**

The most significant changes made for this milestone were the addition of the PluginHandler class and Servlet Interface. The PluginHandler watches a Plugins directory for the addition of Jar files from which new servlets would be dynamically included into the web server. The ConnectionHandler communicates with the PluginHandler, passing along the request for the PlugHandler to process correctly. This is done through a HashMap, which relates the context root to a second HashMap that stores the servlets and their respective URIs. Any servlet must implement the Servlet interface, which contains information necessary for the PluginHandler as well as its custom request processing method. The basic GET, POST, PUT, and DELETE methods from MS1 became “static servlets” that will be run if no plugin is found for that kind of request.

## **Feature Listing & Assignment**

**Angelica Rodriguez**

* W-1: GET Requests
* W-2: POST Requests
* W-3: PUT Requests
* W-4: DELETE Requests

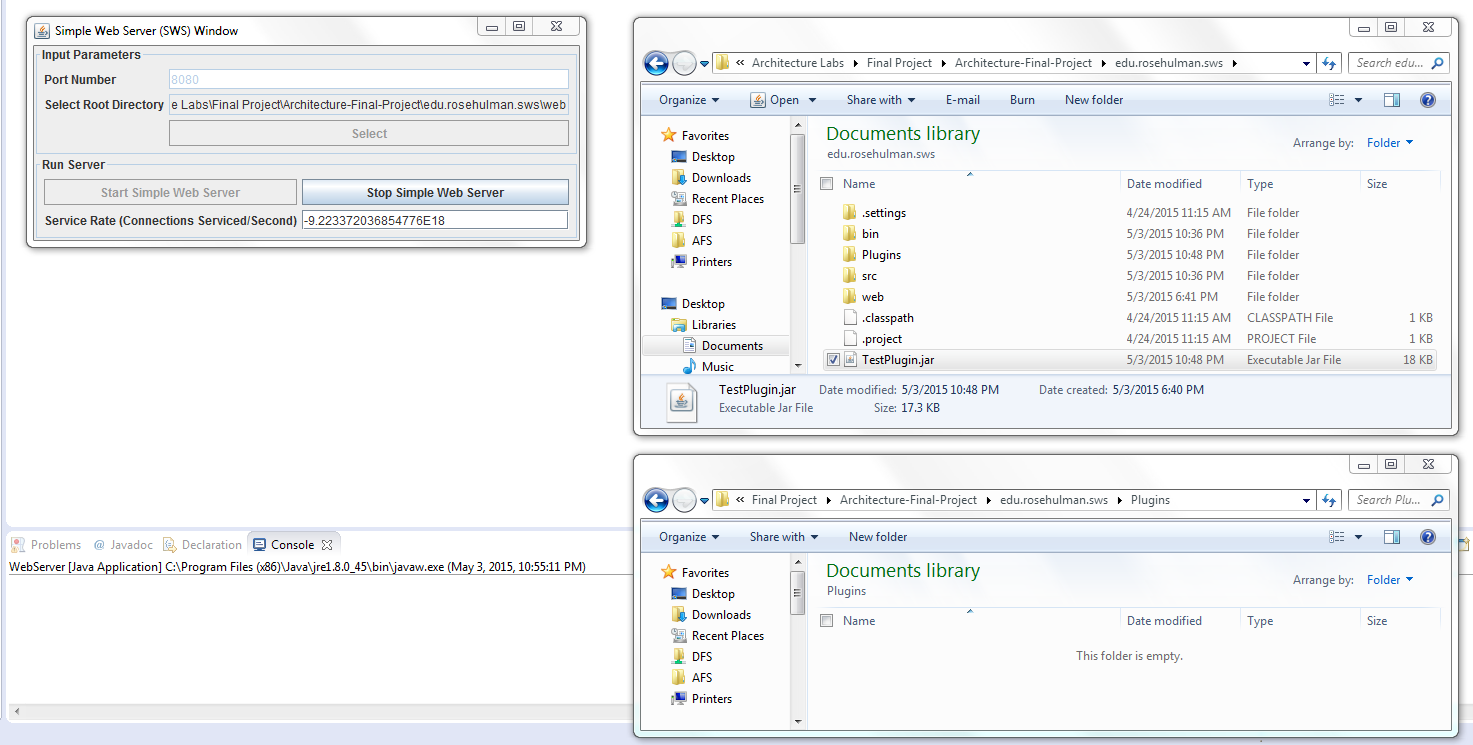
**John Krasich**

* P-1: Dynamic Loading
* E-1: Root Context and Configurable Route
* Test Report

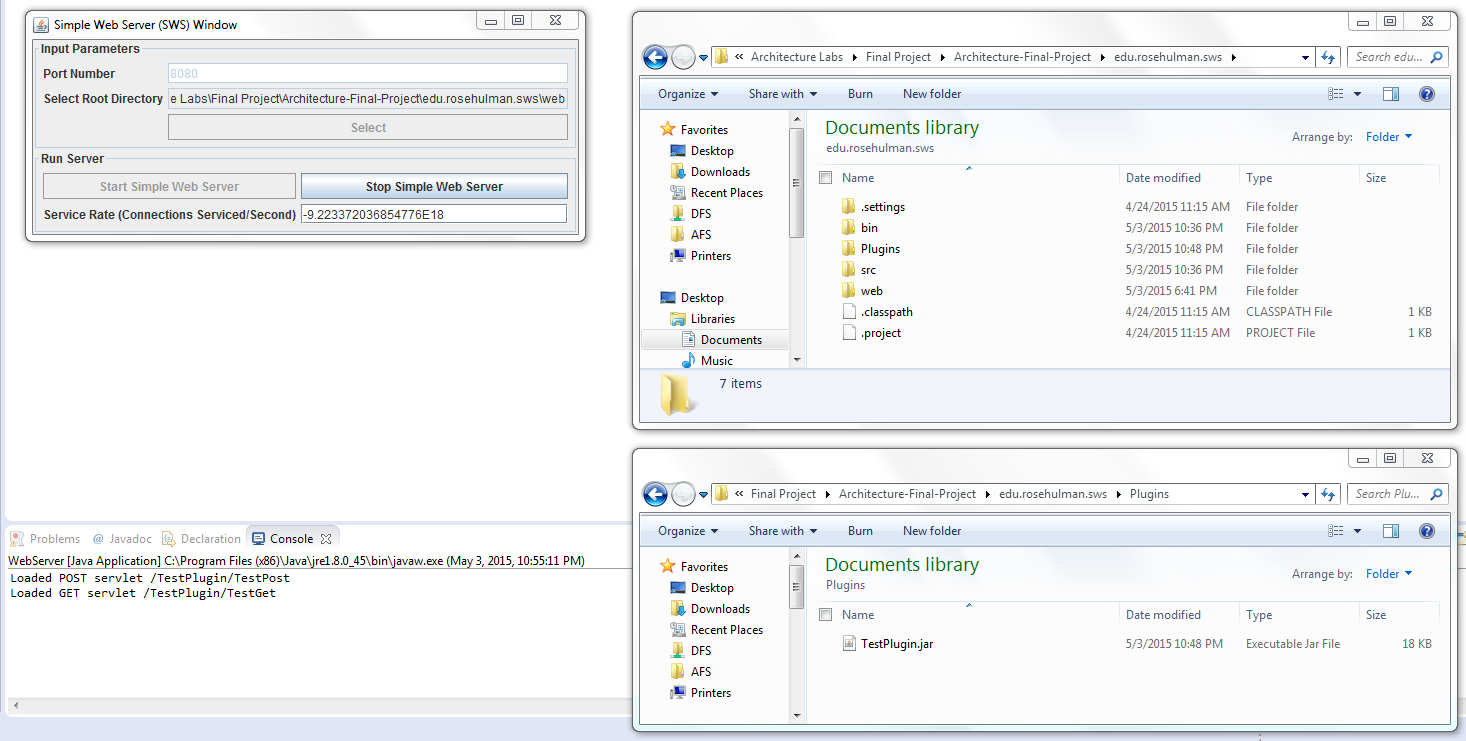
## **Test Report**

### **Plugin Addition**

Before:

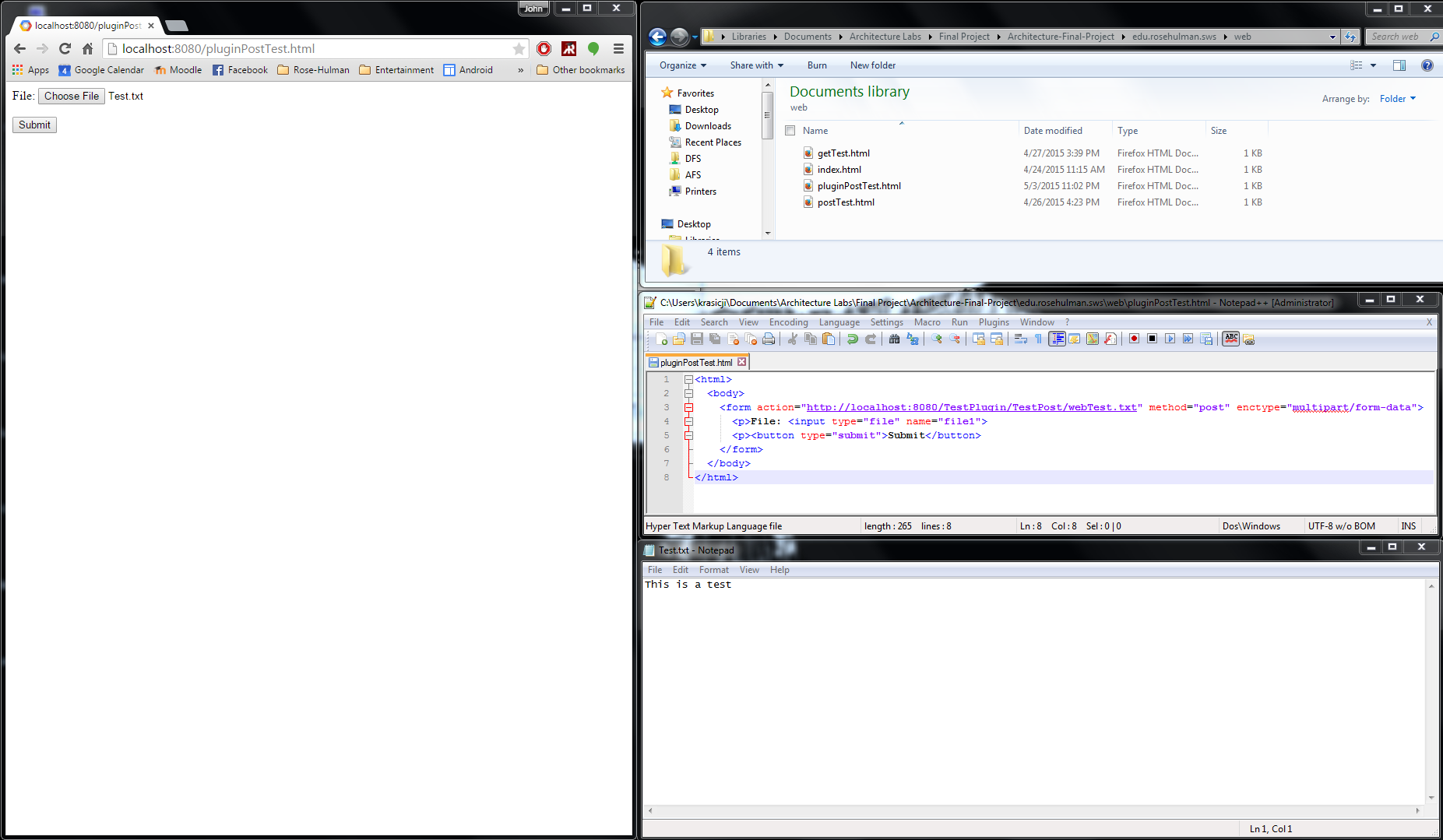


After: the plugin containing two servlets were dynamically loaded into the web server.

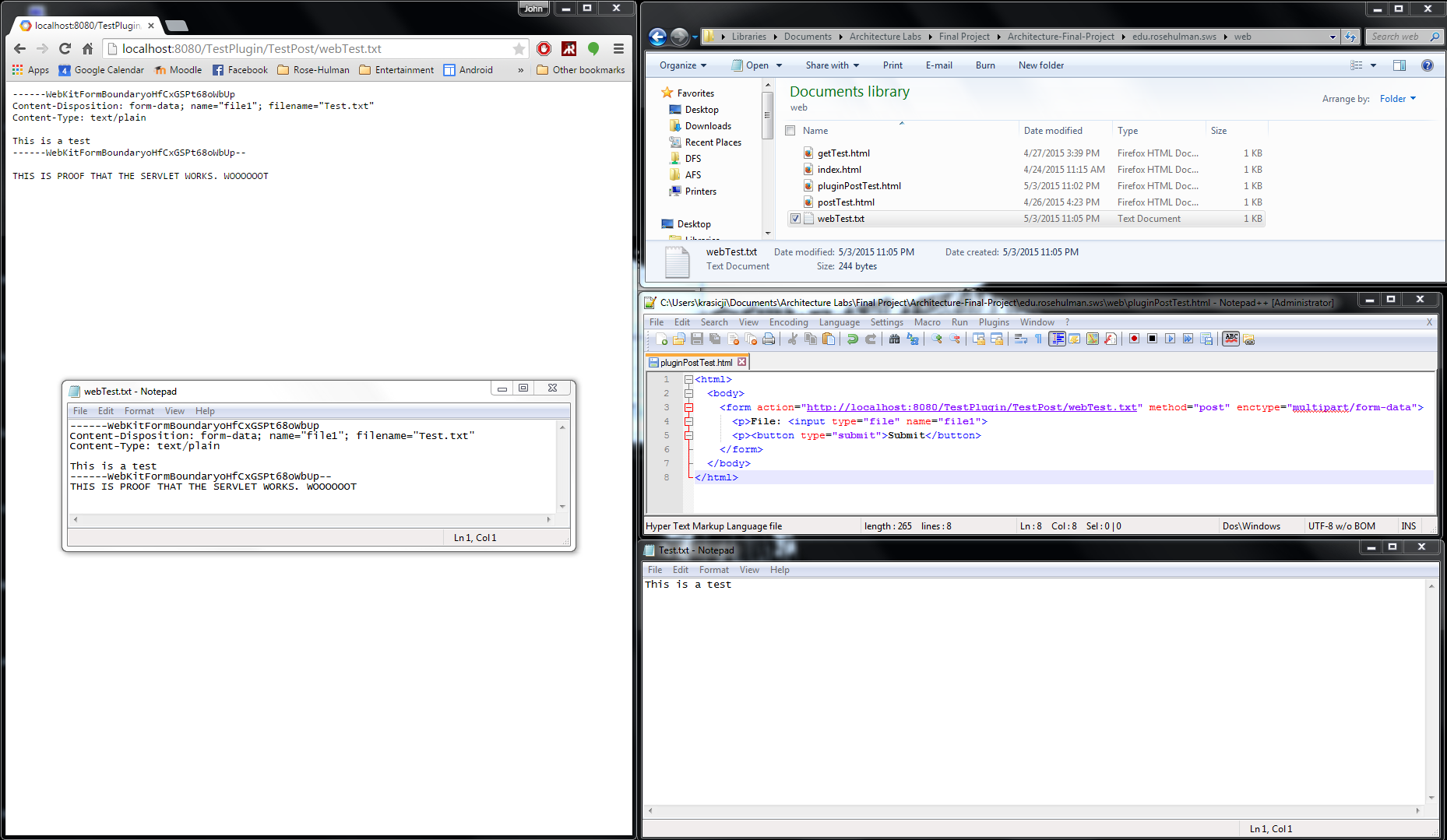


### **POST**

Before:

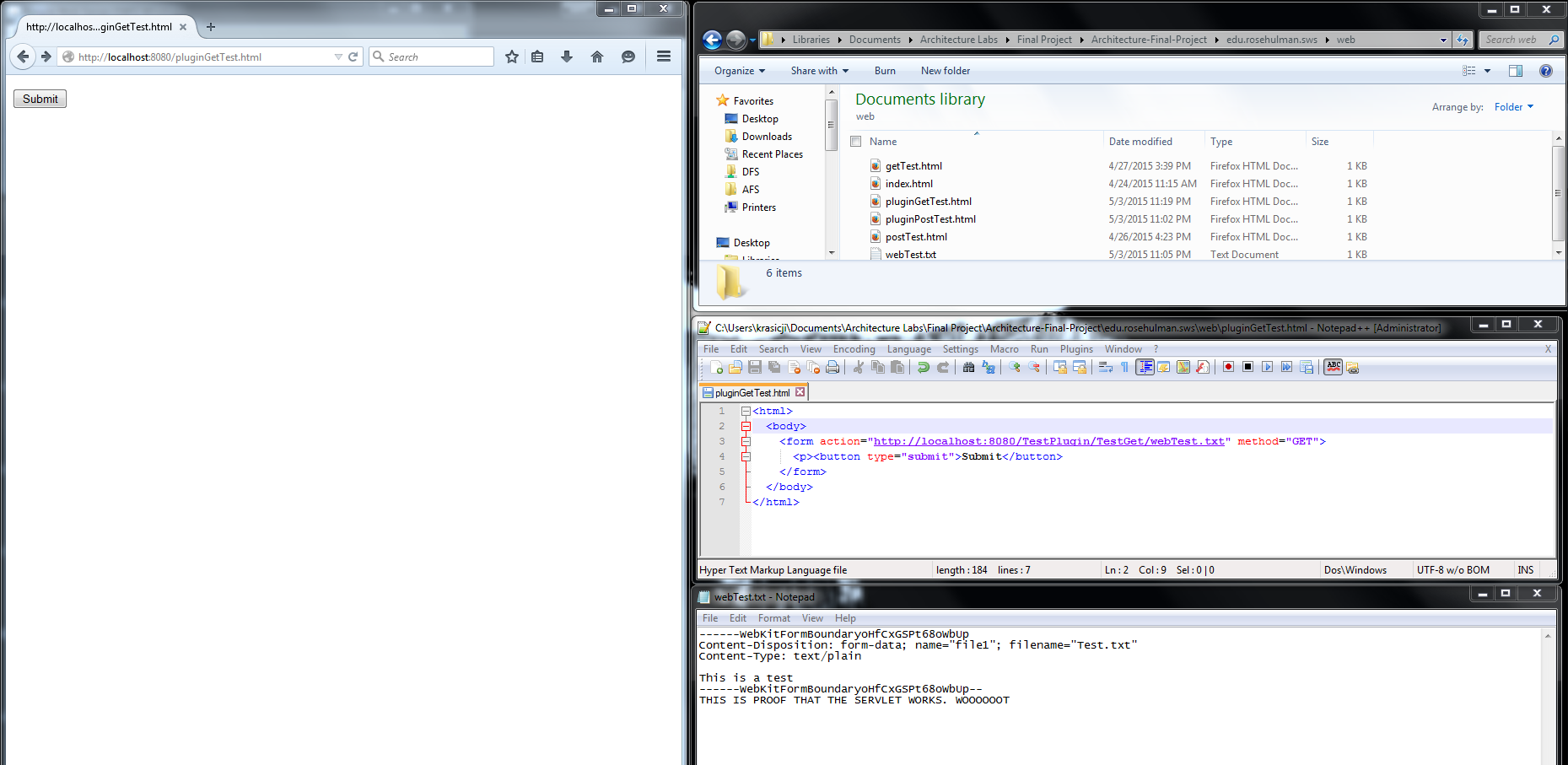


After: the servlet appended “THIS IS PROFF THAT THE SERVLET WORKS. WOOOOOT” to the file + body.

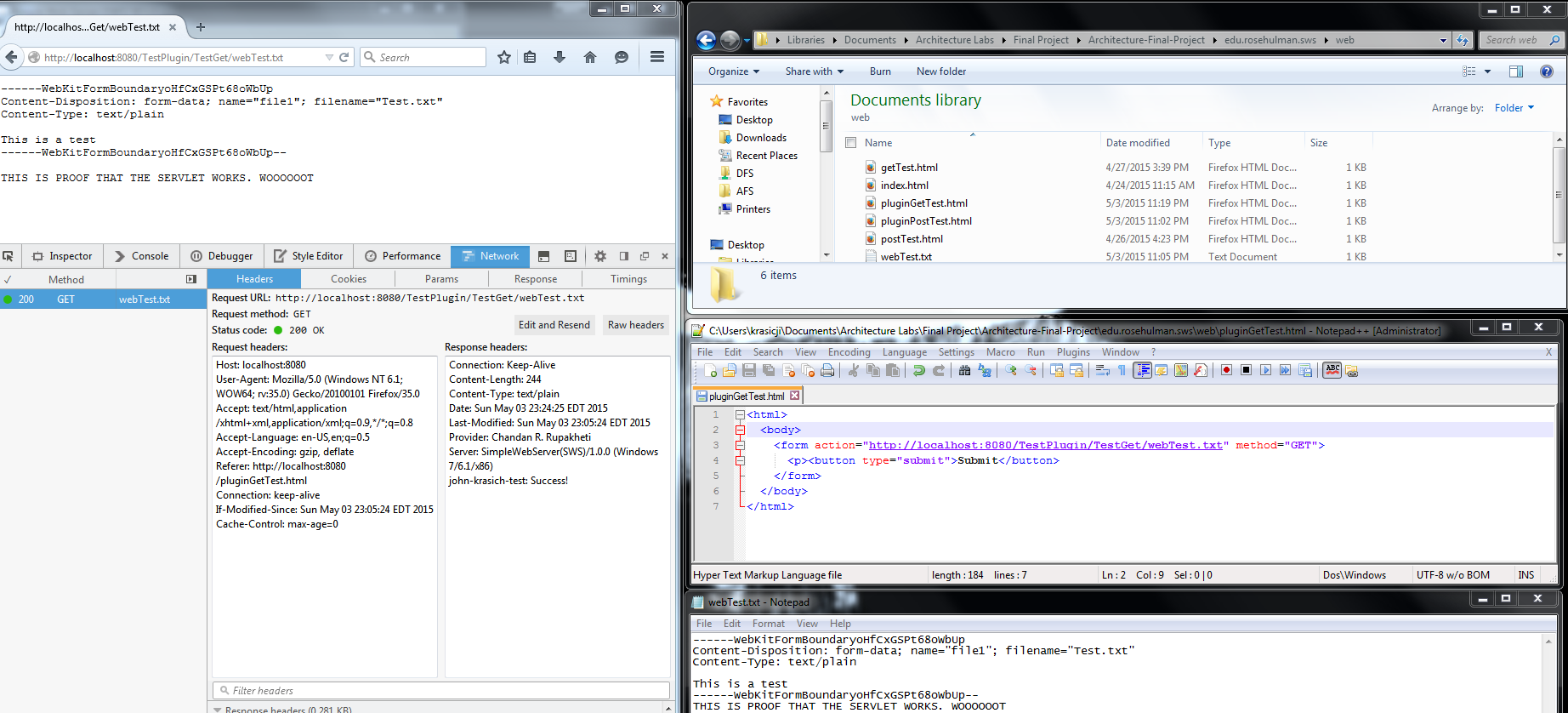


### **GET**

Before:

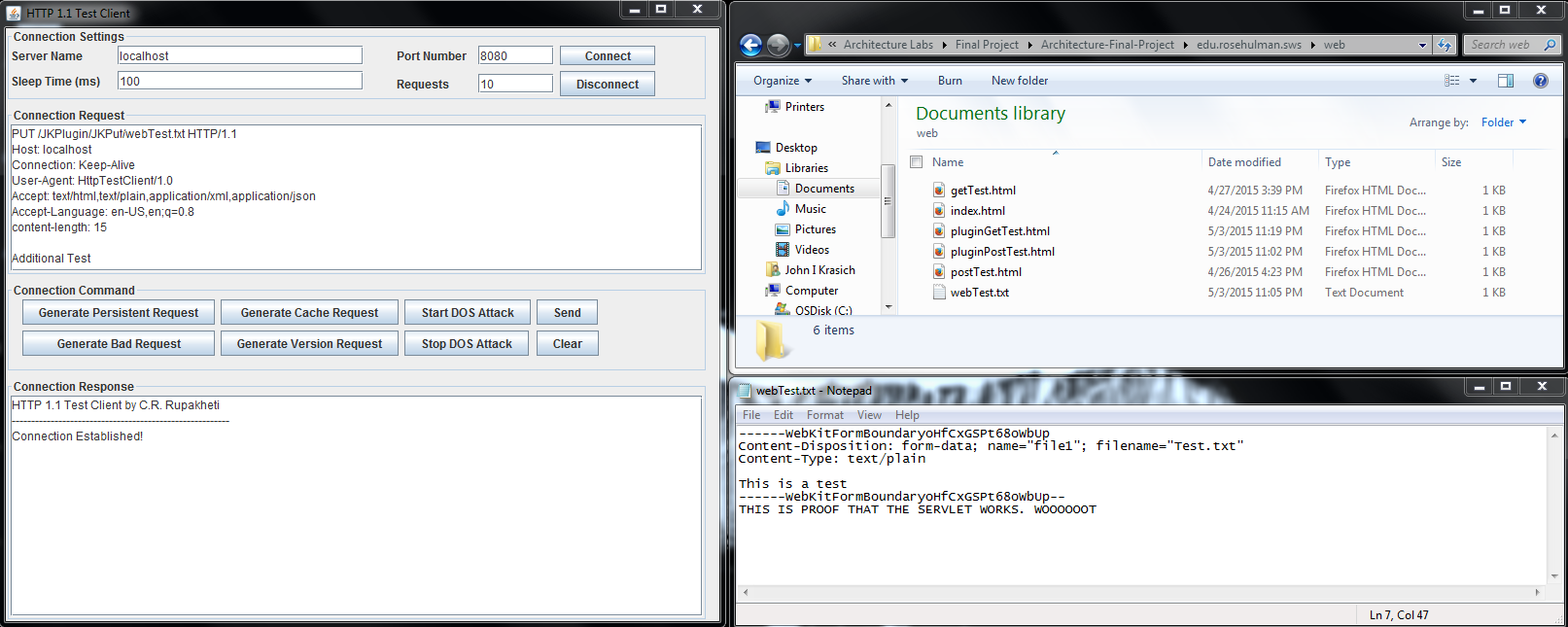


After: the servlet appended an additional header to the response.

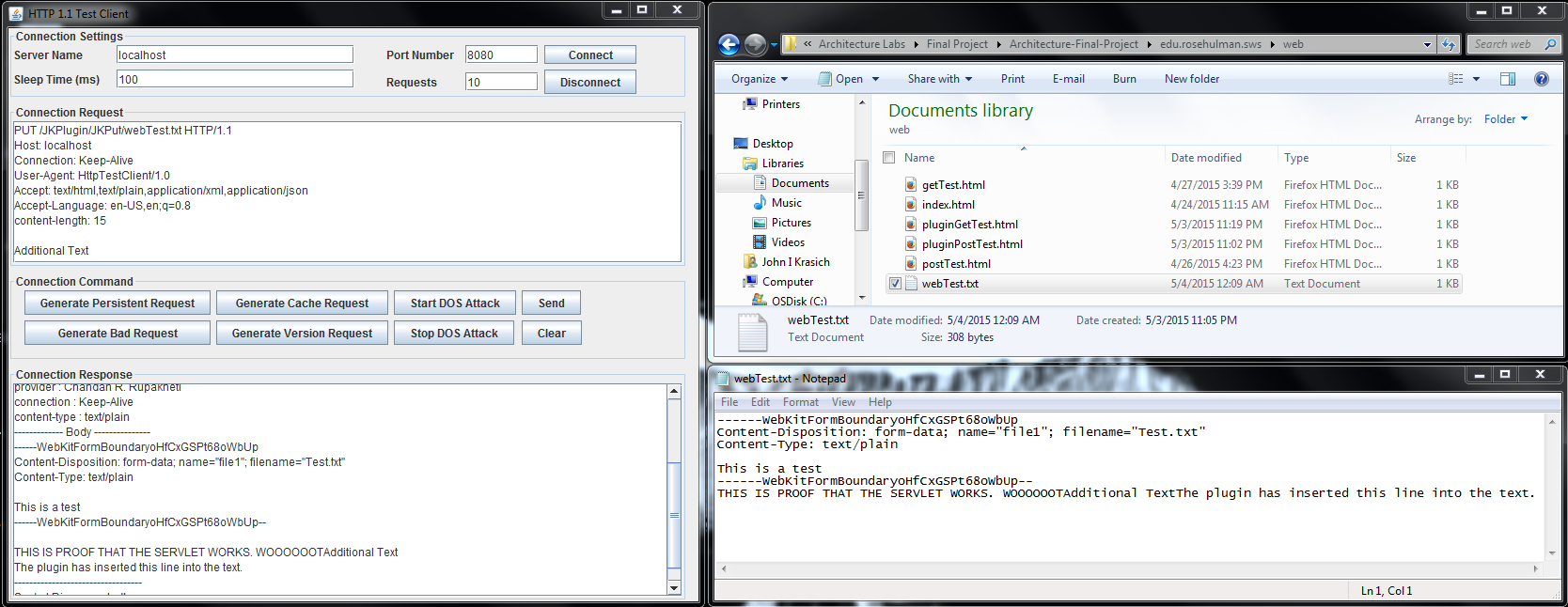


### **PUT**

Before:

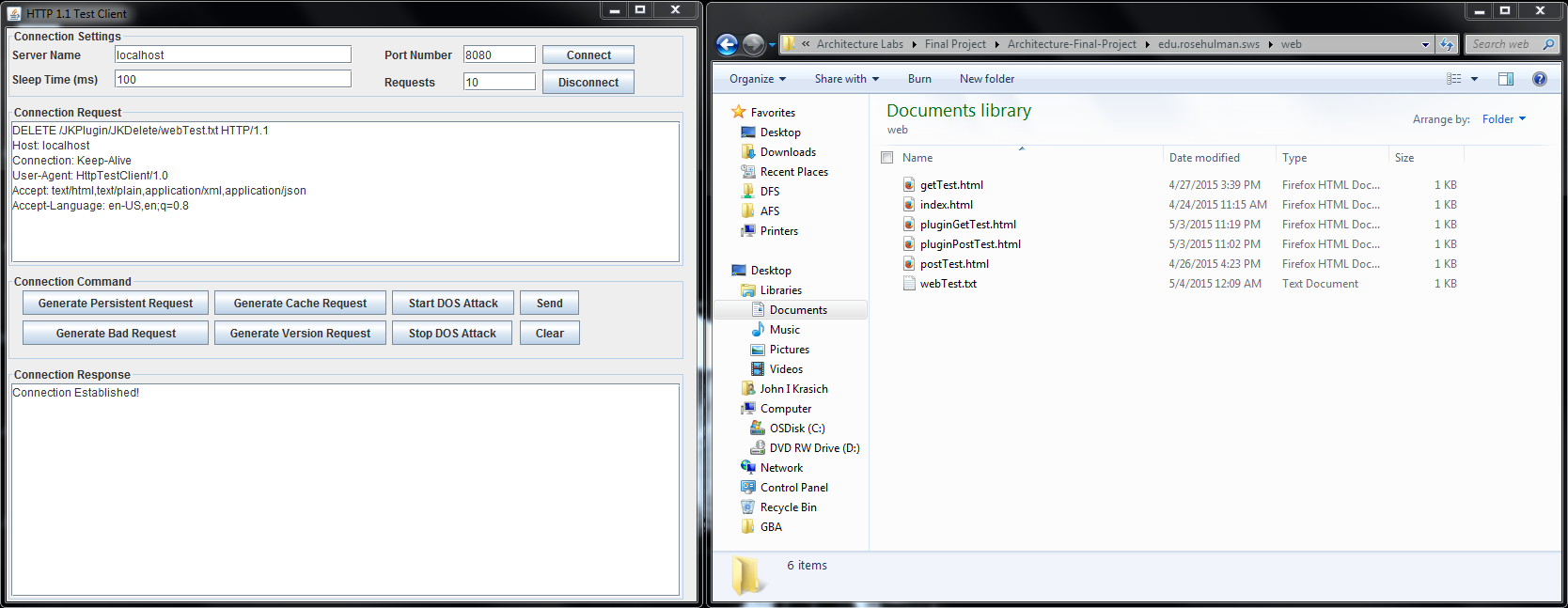


After: the servlet appended the extra text “The plugin has inserted this line into the text” into the file.

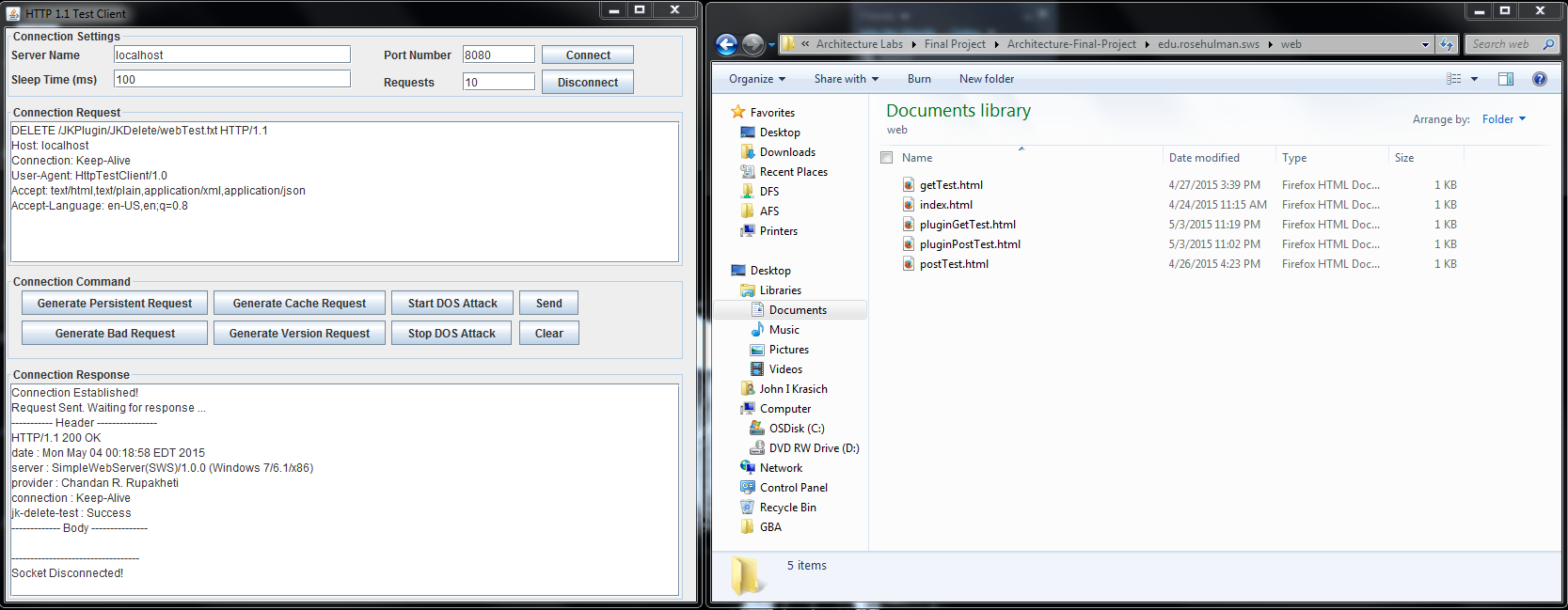


### **DELETE**

Before:



After: the servlet appended an extra header into the delete response.



## **Future Improvements**

One idea we did not get to try but wanted to was to have the users supply a configuration file for the servlets with the information necessary, rather than have them hardcode the request code and create the JAR file. This would be a significant improvement because it would allow it to be easily modifiable, since they could make changes dynamically.

# **Change History – MS3**

## **Updated Architecture Diagram**

## **Updated Detailed Design**

### **Brief Description**

## **Tactics/Feature Listing John – Availability tactics and DDoS attack Jelly – Scheduling, performance for multiple requests, and blacklisting**

## **Architectural Evaluation and Improvements**

### **Availability**

#### **A1 – Request Timeout**

* A1.1 Concrete Scenario
  + Source – User
  + Stimulus – User makes a malformed request
  + Artifact – Web Server
  + Environment – Normal operation
  + Response – 408 Response Timeout + disconnect socket
  + Response Measure – 10 seconds
* A1.2 Test Plan
  + We will use the test utility to send a GET request with no body (currently causes webserver to hang)
* A1.3 Baseline
  + Response must be given after 10 seconds following the request.
* A1.4 Improvement Tactics
  + We will create a Response408ResponseTimeout class, which will be called from the ConnnectionHandler class if a response is not generated within 10 seconds of attempting to read the request.
* A1.5 Conclusion

#### **A2 – Incorrect Plugin Drop**

* A2.1 Concrete Scenario
  + Source – Developer
  + Stimulus – Developer adds a new plugin to the webserver that has inconsistencies
  + Artifact – Web Server
  + Environment – Normal operation
  + Response – Developer should be notified and the plugin removed (ignored by server)
  + Response Measure – Web server does not crash, and does not implement the plugin’s servlets.
* A2.2 Test Plan
  + We will create a plugin with non-servlet classes and drop the plugin into the plugins folder.
* A2.3 Baseline
* A2.4 Improvement Tactics
  + We will add exception handling within our PluginHandler class to notify developers of the error. User’s attempting to navigate to the plugin’s context root will receive a 404 response.
* A2.5 Conclusion

### **Performance**

#### **P1 – Handling Numerous Requests**

* P1.1 Concrete Scenario
  + Source – Multiple Users
  + Stimulus – A large number of requests are being sent to the server simultaneously
  + Artifact – Web Server
  + Environment – Normal operation
  + Response – Web server should maintain 100 connections/second for an extended period of time without error
  + Response Measure – Web server runs for a minute at 100 connections/second without errors
* P1.2 Test Plan
  + We will send numerous requests to the webserver such that its connection rate is always at its peak to see how long the server can maintain the connections.
* P1.3 Baseline
* P1.4 Improvement Tactics
  + When the file is being processed, we will stop our timeout timer. It is assumed that while the file is being read (or body being written), there is no need to timeout the process simply because the file is large.
* P1.5 Conclusion

#### **P2 – Scheduling Events**

* P2.1 Concrete Scenario
  + Source – User(s)
  + Stimulus – Multiple requests of various sizes being sent simultaneously
  + Artifact – Web Server
  + Environment – Normal operation
  + Response – Web server should prioritize request based on size (with starvation prevention)
  + Response Measure – Speed at which requests are returned should be quicker than its original speed before scheduling.
* P2.2 Test Plan
  + We will bombard the server with many different requests of various sizes
* P2.3 Baseline
* P2.4 Improvement Tactics
  + We will create a queue of requests organized by content length and will process the requests in order when available.
* P2.5 Conclusion

### **Security**

#### **S1 – Handling DDoS Attacks**

* S1.1 Concrete Scenario
  + Source – The Denial of Service Launcher
  + Stimulus – The Denial of Service Launcher attacks the web server
  + Artifact – Web Server
  + Environment – Normal operation
  + Response – Web server should properly remove stale connections
  + Response Measure – Web server service rate limited to 100 connections/second
* S1.2 Test Plan
  + We will run the denial of service launcher and “attack” the web server
* S1.3 Baseline
* S1.4 Improvement Tactics
  + We will monitor the service rate to make sure the rate is in an acceptable range, as well as periodically (or as needed) remove stale connections
* S1.5 Conclusion

#### **S2 – Blacklisted IP Connection**

* S2.1 Concrete Scenario
  + Source – A blacklisted IP
  + Stimulus – A blacklisted IP connects to the web server
  + Artifact – Web Server
  + Environment – Normal operation
  + Response – Web server should deny the connection to the IP and log the event
  + Response Measure – The event is logged and no connection is made
* S2.2 Test Plan
  + We will add a known IP (such as Jelly’s computer) to the blacklist and attempt to access the webserver.
* S2.3 Baseline
* S2.4 Improvement Tactics
  + We will create a blacklist to check against when connections are attempted on the server.
* S2.5 Conclusion

## **Future Improvements**