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

# **Change History**

## **MS2 – Plugin Support**

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.

## **MS3 – Applying Improvement Tactics for Server Attributes**

At this point our server can handle/improve the following scenarios:

* Sending a 408 Response Timeout and disconnecting the socket when the server is hanging
* Handling when an incorrect plugin is dropped into the plugin folder without the server crashing
* Handling numerous requests
* Scheduling multiple requests in a queue by content-length
* Handling DDos Attacks
* Preventing blacklisted IP’s from connecting to the server

The only new addition to our class structure was the Response408Timeout class that would generate a response if a response is not generated within 10 seconds of attempting to read the request. All other changes were internal to the classes; the details of which can be found in our tactic implementation specifications.

## **MS4 – Developing an Application to Run on the Server**

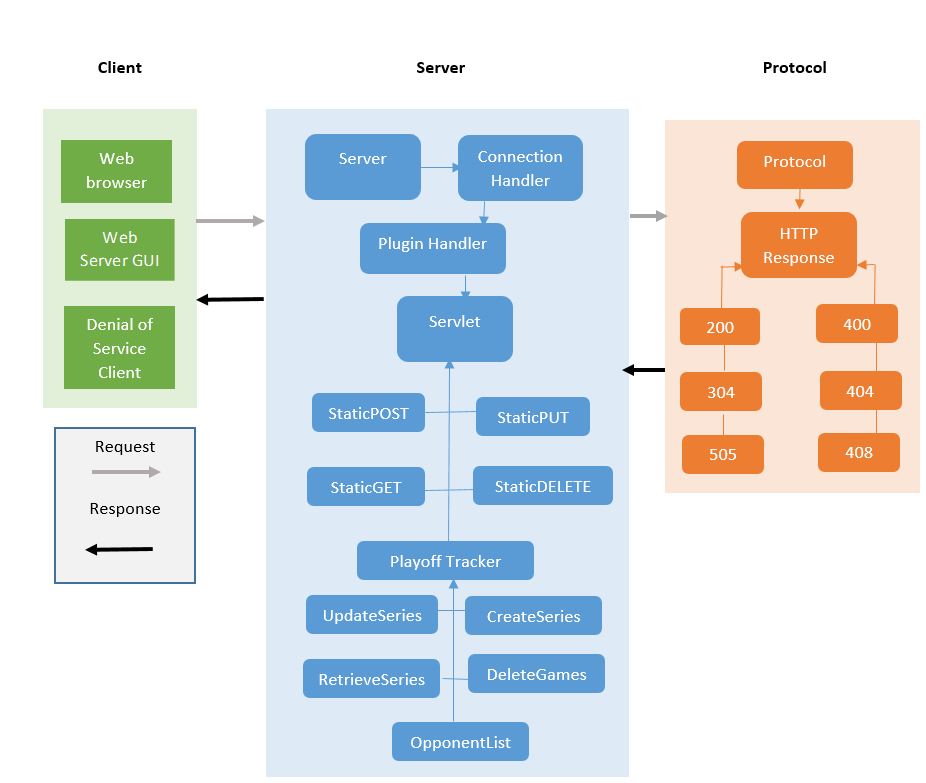
For this milestone we created a Playoff Tracker plugin that had the following servlets:

* CreateSeries (POST)
* DeleteGames (DELETE)
* OpponentList (GET)
* RetrieveSeries (GET)
* UpdateSeries (PUT)

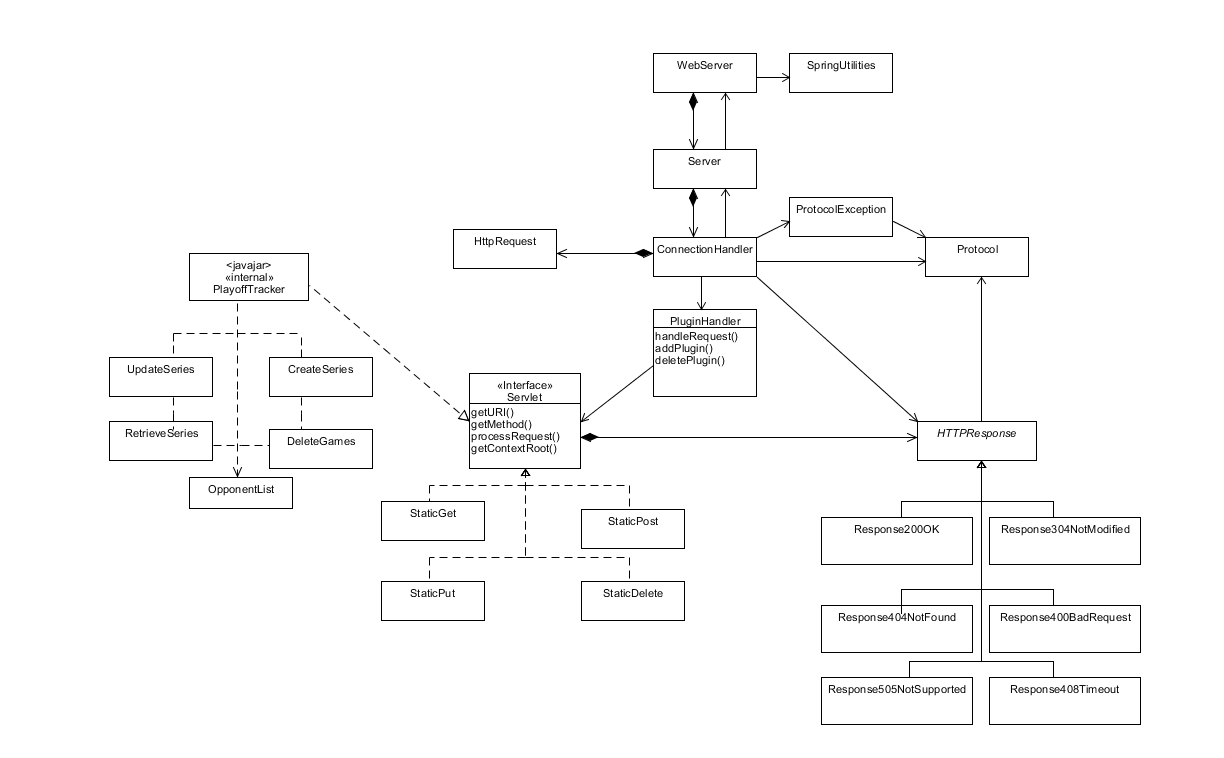
The plugin was added to the server’s plugin directory. To the servers web folder we added the html pages needed to serve all of these requests, as well as a js file with the ajax calls needed, and a css file to make the page look nicer. Other than, no other classes or functionality was added to the server for this milestone. The architecture diagram and class diagram in the following sections have been updated to reflect the addition of the Playoff Tracker Plugin.

# **Architecture and Design**

## **Updated Architecture Diagram**



## **Updated and Detailed Design**



### **Brief Explanation**

The only addition to the UML diagram this time around was adding the Playoff Tracker plugin to demonstrate the servlets it has that we used during this milestone that implement our Servlet interface.

# **Tactics/Feature Listing**

Anything with a \* next to it indicates we worked on that feature together

## **Angelica Rodriguez**

## **MS1**

* F1 – Redesign and Refactor\*
* F2 – Basic support for the POST request\*

### **MS2**

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

### **MS3**

* P1 – Handling Numerous Requests
* P2 – Scheduling Events
* S2 – Blacklisted IP Connection

### **MS4**

* F1 – Retrieving a list of opponents\*
* F2 – Retrieving series results
* F3 – Creating a series schedule\*
* General design for plugin

## **John Krasich**

## **MS1**

* F1 – Redesign and Refactor\*
* F2 – Basic support for the POST request\*
* F3 – Basic support for the PUT request
* F4 – Basic support for the DELETE request

### **MS2**

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

### **MS3**

* A1 – Request Timeout
* A2 – Incorrect Plugin Drop
* S1 – Handling DDoS Attacks

### **MS4**

* F2 – Retrieving series results\*
* F3 – Creating a series schedule\*
* F4 – Editing a series schedule
* F5 – Deleting a game from a series

# **Architectural Evaluation and Improvements**

**???????**

# **2015 NBA Playoff Tracker for the Chicago Bulls**

The Playoff Tracker is an application that allows the user to follow the Chicago Bull’s NBA playoff run, keeping track of opponents, scores, and upcoming games.

## **API Design**

**F1 – Retrieving a list of opponents**

**Method**: GET

**URI**: /PlayoffTracker/OpponentList

**Request** **Body**:

<none>

**Response** **Body**:

{

“code”: 200,

“message”: “Ok”,

“payload”:

[

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h2>2015 Playoff Opponent List</h2>

<br>

<table class=*"customTable"*>

<thead>

<tr>

<th>Round</th>

<th>Team</th>

</tr>

</thead>

<tbody>

<tr>

<td>1</td>

<td id=*"r1"*>Opponent 1</td>

</tr>

<tr class=*even*>

<td>2</td>

<td id=*"r2"*>Opponent 2</td>

</tr>

….

]

}

**Development Status**: DONE

**F2 – Retrieving a series results**

**Method**: GET

**URI**: /PlayoffTracker/RetrieveSeries/r1

**Request Body**:

<none>

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

“payload”:

[

<!DOCTYPE html>

<html>

<head>

<body>

<h1>Series Schedule </h1>

<h2>Opponent: Milwaukee Bucks</h2>

<table class=*'table'*>

<thead>

<tr>

<th>Game</th>

<th>Date</th>

<th>Location</th>

<th>Result</th>

<th>Score</th>

</tr>

</thead>

<tbody>

<tr>

<td>1</td>

<td>4/18/15</td>

<td>Away</td>

<td>W</td>

<td>91-103</td>

</tr>

<tr class=*"even"*>

<td>2</td>

<td>4/20/15</td>

<td>Away</td>

<td>W</td>

<td>82-91</td>

</tr>

….

]

}

**Development Status**: DONE

**F3 – Creating a series schedule**

**Method**: POST

**URI**: /PlayoffTracker/CreateSeries/r2

**Request Body**:

{

opponent=Cavaliers

date= 5/4/15

location=Away

date=5/6/15

location=Away

…

}

**Response Body**:

{

“code”: 200,

“message”: “OK”,

“payload”:

[

<!DOCTYPE html>

<html>

<head>

<body>

<h1>Series Schedule </h1>

<h2>Opponent: Cavaliers</h2>

<table class=*'table'*>

<thead>

<tr>

<th>Game</th>

<th>Date</th>

<th>Location</th>

<th>Result</th>

<th>Score</th>

</tr>

</thead>

<tbody>

<tr>

<td>1</td>

<td>5/4/15</td>

<td>Away</td>

<td></td>

<td></td>

</tr>

<tr>

<td>2</td>

<td>5/6/15</td>

<td>Away</td>

<td></td>

<td></td>

</tr>

<tr>

]

}

**Development Status**: DONE

**F4 – Editing a series schedule**

**Method**: PUT

**URI**: /PlayoffTracker/UpdateSeries/r2

**Request Body**:

{

{“id”: 1, “result”: “win”, “score”: “99-92”},

{“id”: 2, “result”: “loss”, “score”: “106-91”},

…

}

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

“payload”:

[

{“id”: 1, “opponent”: “Cavaliers”, “date”: “5/4”, “location”: “Away”

“result”: “win”, “score”: “99-92”},

{“id”: 2, “opponent”: “Cavaliers”, “date”: “5/6”, “location”: “Away”

“result”: “loss”, “score”: “106-91”},

…

]

}

**Development Status**: TODO

**F5 – Deleting a game from the series schedule**

**Method**: DELETE

**URI**: /PlayoffTracker/DeleteGames/r2

**Request Body**:

{

{“id”: 6},

{“id”: 7}

}

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

“payload”: <none>

}

**Development Status**: TODO

# **Future Improvements**

As was mentioned before, having the servlets loaded via configuration file might have been easier, especially this time around when there were changes that needed to be made to the servlets every time we changed our mind about how it should work. Another improvement is that for our put and posts, instead of having the response be in plain text, we could have returned JSON or XML and made use of one of the parsers that worked with Java. That probably would have made the building of the response much easier. As with all web applications, we could have definitely added some input validation for all the fields that the user has to enter when creating a schedule, or updating one.