# CSSE 477 – Milestone 2 – Application Server

You have, by now, a very good understanding of how a web server works. As some of have already noticed, the web server did not follow the Open-Closed Principle (OCP). It was not open to extension and all of the changes were made through direct modification of the code base given to you. We will change that in this milestone with few other cool improvements.

## Requirements

I would like to encourage creativity in this project, so I am **not** putting a lot of constraints in the requirements. Here are the base requirements for this milestone:

### Web Application Framework Requirement

You will extend Milestone 1 to have a plugin support similar to the one you practiced in Homework 6. Right now, GET, POST, PUT, and DELETE are all handled in a very specific way, i.e. to perform file operations. We would like to generalize that using web application hooks as you will see in **Assignment 9** – i.e., using Servlets. You will implement an extension mechanism that allows user to implement a Servlet-like mechanism to handle the following type of HTTP requests:

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

Note that you must provide a mechanism to **either create or access** an already created HTTP response object in the plugin class. The user plugin will fill extra headers and body in the response object as a part of handling these request types.

**Creative Consideration:** One Servlet-like interface/class that has methods to handle all of these request types (GET, POST, PUT, DELETE) versus separate class for each of these request types. Automated creation of response objects versus plugin code (provided by user) creating necessary response objects.

### Server Platform Requirement

**P-1: Dynamic Loading**

Note that your server must constantly monitor a dedicated application directory for an extension plugin. As soon as a plugin is dropped in the directory, the server must load it up as an extension and start using it.

**Creative Consideration:** How do you handle the case when the same extension plugin is dropped again with some changes?

### Extension Requirement

**E-1: Root Context and Configurable Route**

Each plugin must have a unique root path. This is the root context of the plugin. I will explain about root context with an example in the next few paragraphs.

A plugin may implement several servlet-like interfaces. Each servlet must also be associated to a configurable URI in the plugin. Assuming that a Servlet is mapped to a URI, when a request arrives at that URI, the host application will delegate the request to that Servlet. There are several ways to implement this mechanism. One simple approach would be to maintain a route configuration file in the plugin. The content of the configuration file would look like the following:

| **Request Type** | **Relative URI** | **Servlet Class** |
|---|---|---|
| GET | /MyGetHandlerServlet | edu.rosehulman.extension.MyGetHandlerServlet |
| POST | /PostServlet | edu.rosehulman.extension.MyPostServlet |
| PUT | /LoadedServlet | edu.rosehulman.extension.LoadedServlet |
| … | | |

Let's execute a use case to understand the flow of control from browser to the user supplied plugin that handles it:

1. Assume that a user types in the following URL in her browser: http://localhost:8080/MyPlugin/MyGetHandlerServlet.
2. The browser generates a **GET** request to get the **/MyPlugin/MyGetHandlerServlet** resource from the server.
3. In this example, the context root of the plugin is: **/MyPlugin** and the relative URI for the servlet is **/MyGetHandlerServlet**.
4. Using the context root (**/MyPlugin**), the host application (server) knows it should forward the request to the **MyPlugin** plugin.
5. When the plugin receives the request, using the relative URI (**/MyGetHandlerServlet**), it knows the request should be further forwarded to the **edu.rosehulman.extension.MyGetHandlerServlet** class.
6. As a result, an appropriate method (e.g. **doGet()**)or something similar that you come up with will be called to process the request in the **MyGetHandlerServlet** class.

**Creative Consideration**: What happens when two plugins use the same context root? What happens if two servlets within a plugin share the same relative URI for the same type of request? How do you separate static resources (e.g. index.html, image.jpg, etc.) from a dynamic one such as servlets?

## Deliverables

**Report (doc or pdf)**
Your report must include the following items:

1. You will update the MS1 report. Please include a **Change History** section to reflect changes in this version.
2. Updated architecture diagram.
3. Updated detailed design and a brief explanation.
4. Feature listing and assignment (Who did what)
5. Test report of the new features. It is expected that you will have several plugins implemented to test different aspect of all of the proposed features. **GET and POST must be tested using a web browser in addition to other tools.**
6. A section that lists improvements that you can think of but haven't tried yet.
7. Code documentation - either provide reasonable amount of in-line comments or write self-documenting code (http://en.wikipedia.org/wiki/Self-documenting).

**Code (zip)**

Bundle the project and turn-it in on Moodle.

## Timeline

**Thursday, Oct 22, 2015, Class Time** – Please verify your design with your instructor. You must bring a UML class diagram for discussion during class.

**Monday, Oct 26, 2015, 8:00 am** – Please turn in your report (pdf) and the source code (zip) files on Moodle. You will show your project work to your instructor during the class.

## Resources

Tutorial on Watching a Directory for Changes: http://docs.oracle.com/javase/tutorial/essential/io/notification.html.
How a Java Servlet Works: http://www.onjava.com/pub/a/onjava/2003/05/14/java_webserver.html.