**Computer Science Department**
**San Francisco State University**
**CSC 667/867**
**Spring 2022**

# Web Server Project

## Due Date
Monday, February 21, at midnight.

## Overview
The purpose of this project is to write a web server application capable of a subset of the HyperText Transfer Protocol.

This is a two person team project. You must get started **immediately**.

## Submission
Submission must occur via github repository. You may create your repository here: https://classroom.github.com/a/HpNdvn9W.

Your assignment will be submitted using github. Only the "main" branch of your repository will be graded. Late submission is determined by the last commit time on the `main` branch. You are required to submit a documentation **PDF** named `documentation.pdf` in a `documentation` folder at the root of your project.

The pdf should contain the following information:
* Full names of team members
* A ***link*** to the github repository
* A copy of the rubric with each item checked off that was completed (feel free to provide a ***suggested*** total you deserve based on completion)
* Project discussion:
    * Brief description of architecture (pictures are required here)
    * Problems you encountered during implementation, and how you solved them
    * A discussion of what was difficult, and why
    * A description of your test plan (if you can't prove that it works, you shouldn't get 100%). (Tip: use Postman and telnet for testing the functionality of your server.)

Please take care to properly format your code so that it is easily readable by a human. Presentation counts!

**Grading Policy**
As a reminder, any student found cheating on their project will receive a zero, and be reported to the department for possible disciplinary action. Any sharing of github repositories outside of your team will be considered cheating. For this assignment, the standard late policy will apply.

**Specification**
Generally, this project can be divided into the following milestones:
1. Read, and store, standard configuration files for use in responding to client requests
2. Parse HTTP Requests
3. Generate and send HTTP Responses (this involves many possible code paths, and is probably the most significant implementation step)
4. Respond to multiple simultaneous requests through the use of threads
5. Execute server side processes to handle server side scripts
6. Support simple authentication
7. Support simple caching
8. Logging

**This is how the project will be graded:**
1. Your repository will be cloned
2. I will cd into the repository directory
3. I will execute the following commands:
   3.1. rm -rf conf
   3.2. find . -name "*.class" -type f -delete
   3.3. find . -name "*.jar" -type f -delete
   3.4. cp myConf ./conf
   3.5. javac WebServer.java
   3.6. java WebServer

**1. Read Standard Server Configuration**
Web Servers typically read in a number of configuration files that define the behavior of the server, as well as the types of requests that the server can respond to. Your web server must read two common file formats (the files have been copied from an older Apache Web Server application): `httpd.conf`, and `mime.types`. These files must be located in a folder named ***`conf`***, which will be in the server root directory (along with the main class for your server, which must be named **`WebServer`**).

**httpd.conf**
A file that provides configuration options for the server in a key value format, where the first entry on each line is the configuration option, and the second is the configuration value. The following table lists the `httpd.conf` keys your web server must support, along with a brief description of the value.

| Key | Value |
|---|---|
| **Listen** | Port number that the server will listen on for incoming requests. If this directive is not provided, the server should use the port 8080. |
| **DocumentRoot** | Absolute path to the root of the document tree |
| **LogFile** | Absolute path to the file where logs should be written to |

| Key | Value |
|---|---|
| **Alias** | Two values: the first value is a symbolic path, the second value is the absolute path that the symbolic path resolves to |
| **ScriptAlias** | Two values: the first value is a symbolic path to a script directory, the second value is the absolute path that the symbolic path resolves to, from which scripts will be executed |
| **AccessFile** | The name of the file that is used to determine whether or not a directory tree requires authentication for access (default is .htaccess) |
| **DirectoryIndex** | One or more filenames to be used as the resource name in the event that a file is not provided explicitly in the request. If this directive is not provided, the server should use “index.html”. |

**mime.types**
A file that provides a mapping between file extension and mime type. The server uses this file in order to properly set the `Content-Type` header in the HTTP Response. The format of this file is any number of lines, where each line contains the following information:

*mime-type file-extension*

Note that each line could contain any number of file extensions for the given mime type (including zero!). You should use the file provided for you on iLearn.

**Note:**
When parsing either of these files, empty lines, and lines beginning with the comment character, #, should be ignored.

**2. Parse HTTP Requests**
Your server must appropriately respond to the following request methods: **GET, HEAD, POST, PUT, and DELETE**.

The general format of an HTTP Request will be covered in class, and is well documented online.

**3. Generate and Send HTTP Responses**
Your server must be able to generate all of the following response codes: **200, 201, 204, 304, 400, 401, 403, 404, 500**. The following headers must be returned in every response: **Date** and **Server**. The value for the Server header should be the last names of both students on the team. Note that there are additional headers that must be returned according to the protocol for each of the possible responses (including **Content-Type** and **Content-Length**).

The general format of an HTTP Response will be covered in class, and is well documented online.

**4. Respond to Multiple Requests Simultaneously (Threaded)**
Your server must be able to handle simultaneous requests. We will discuss strategies for implementing and testing this in class.

**5. Execute Server Scripts**

Your server must be able to execute resources that have been requested in a ScriptAlias'ed directory. For every header encountered in a request, your server should convert that header to an environment variable to be sent to the script. The script should receive the body of PUT or POST methods via stdin.

Refer to the CGI lecture for details, but recall that:

* The script is responsible for setting the **Content-Length** and **Content-Type** headers.
* The script may add additional headers (so no new line should be output by the server when serving these files).
* The server is responsible for providing environment variables to the script, including SERVER_PROTOCOL, QUERY_STRING, and all of the headers received from the client, with HTTP_ prepended.

Note that the testing scripts provided in the sample website expect that you have a perl interpreter installed. For Windows users, you will need to install perl and update the she-bang to point to the directory the perl interpreter was installed.

**6. Support Simple Authentication**

Your server must be able to handle the 401/403 authentication workflow. Permission to access a given resource will be determined by the presence of the file specified by the `AccessFileName` directive in the `httpd.conf` file anywhere in the directory tree of the requested resource. This file contains directives in key value pairs that specify authentication information for a given set of resources. The following table lists the `.htaccess` keys your web server must support, along with a brief description of the value.

| Key | Value |
|---|---|
| **AuthUserFile** | Absolute path to the location of the file containing the username and password pairs, in .htpassword format |
| **AuthType** | The only value you will be required to support is 'Basic' |
| **AuthName** | The name that will be displayed in the authentication window provided by clients |
| **Require** | Specifies the user or group that can access a resource. valid-user is a special value that indicates that any user listed in the AuthUserFile can access the resource |

Note that the authentication workflow requires additional headers (`WWW-Authenticate`, `Authorization`).

The AuthUserFile contains lines of key value pairs separated by a colon. The key is the username, and the value is the encrypted password, as in this example: **`jrob:{SHA} cRDtpNCeBiql5KOQsKVyrA0sAiA=`**. We will be using SHA encryption. Note that the password string in the file contains the **`{SHA}`** header - this must be removed before comparison as it is not a part of the encrypted password itself.

The client will provide a header containing the Base64 encoded username and password, separated by a colon. The **java.util.Base64** class will come in handy for decoding this key value pair. You will also need the **java.security.MessageDigest** class. (A skeleton has been provided for your to help with password comparison - see the `Htpassword.java` file on iLearn.)

### 7. Support Simple Caching

Your server must be able to appropriately generate the 304 response. Note that this requires additional headers (`Last-Modified`).

### 8. Logging

Your server must log to standard out, and append to a log file (location specified in `httpd.conf`), in the common log format: **https://en.wikipedia.org/wiki/Common_Log_Format**.

### Grading Rubric

| Category | Description | | |
|---|---|---|---|
| **Code Quality** | Code is clean, well formatted (appropriate white space and indentation) | 5 | |
| | Classes, methods, and variables are meaningfully named (no comments exist to explain functionality - the identifiers serve that purpose) | 5 | |
| | Methods are small and serve a single purpose | 3 | |
| | Code is well organized into a meaningful file structure | 2 | **15** |
| **Documentation** | A PDF is submitted that contains: | 3 | |
| | Full names of team members | 3 | |
| | A link to github repository | 3 | |
| | A copy of this rubric with each item checked off that was completed (feel free to provide a suggested total you deserve based on completion) | 1 | |
| | Brief description of architecture (pictures are handy here, but do not re-submit the pictures I provided) | 5 | |
| | Problems you encountered during implementation, and how you solved them | 5 | |
| | A discussion of what was difficult, and why | 5 | |
| | A thorough description of your test plan (if you can't prove that it works, you shouldn't get 100%) | 5 | **30** |
| **Functionality - Server** | Starts up and listens on correct port | 3 | |

| Category | Description | | |
|---|---|---|---|
| | Logs in the common log format to stdout and log file | 2 | |
| | Multithreading | 5 | **10** |
| **Functionality - Responses** | 200 | 2 | |
| | 201 | 2 | |
| | 204 | 2 | |
| | 400 | 2 | |
| | 401 | 2 | |
| | 403 | 2 | |
| | 404 | 2 | |
| | 500 | 2 | |
| | Required headers present (Server, Date) | 1 | |
| | Response specific headers present as needed (Content-Length, Content-Type) | 2 | |
| | Simple caching (HEAD with If-Modified-Since results in 304 with Last-Modified header, Last-Modified header sent) | 1 | |
| | Response body correctly sent | 3 | **23** |
| **Functionality - Mime Types** | Appropriate mime type returned based on file extension (defaults to text/text if not found in mime.types) | 2 | **2** |
| **Functionality - Config** | Correct index file used (defaults to index.html) | 1 | |
| | Correct htaccess file used | 1 | |
| | Correct document root used | 1 | |
| | Aliases working (will be mutually exclusive) | 3 | |
| | Script Aliases working (will be mutually exclusive) | 3 | |
| | Correct port used (defaults to 8080) | 1 | |
| | Correct log file used | 1 | **11** |
| **CGI** | Correctly executes and responds | 4 | |
| | Receives correct environment variables | 3 | |
| | Connects request body to standard input of cgi process | 2 | **9** |