Java Web Server

Project Documentation

Team:

web-server-albert-and-zac

Students:

Alberto Fernandez Saucedo & Zac Henney

Github Repository:

https://github.com/sfsu-csc-667-fall-2018/web-server-albert-and-zac.git

Introduction & Overview:

The purpose of the Web Server project is to write a Java web server in a team of two that is able to process a subset of the Hyper Text Transfer Protocols (HTTP). The project was allocated some loose milestones with required specifications. The 8 specifications are as follows:

1. Web server must be able to read and store standard configuration files as required to respond to client requests.

This specification requires that the web server be capable of defining its behavior via two common file format files ***httpd.conf*** and ***mime.types***. The files are located in the ***conf*** folder within the ***server root*** directory which also contains the WebServer Class (main class). The files also help to determine the types of requests the server can respond to.

2. The server must be able to parse HTTP request.

The server responds to the following request methods, **GET**, **HEAD**, **POST**, **PUT**, and **DELETE**.
The format for these methods is strictly adhered to and required for proper server response.

3. The server must be able to generate and send HTTP responses.

The server generates the required responses codes, **200**, **201**, **204, 304, 400, 401, 403, 404,** and **500** with headers **Date**, **Server, Content-Type** and **Content-Length**.

4. The server must be multithreaded and capable of responding to multiple requests.

The server utilizes multithreading by extending Thread in the Worker class. The server is capable of processing simultaneous requests.

5. Server must execute server side processes and handle server side scripts.

Missing requirement; server does not implement nor handle server side scripts.

6. Server must support simple authentication.

The server is capable of handling the 401/403 authentication workflow. The server recognizes the file in the AccessFileName directive contained in the httpd.conf file.

7. Server must support simple caching.

Utilizing the *Last-Modified* header the server is capable of generating the 304 response.

8. Server must be capable of logging.

The server logs to a file in the common log format at the specified location *LogFile* in httpd.conf.

The project was developed on the NetBeans IDE 8.2 and NetBeans Platform, using the Java 8 release. Testing was conducted with the API Development Environment, Postman v6.3.0, and Mozilla Firefox browser. The provided public html page was also used for testing.
For testing we primarily relied on Postman and the provided test website that we slightly modified.

200 Ok: Load website in browser (127.0.0.1:8080). Ensure that all images load.

201 Created: Create a PUT request (PUT 127.0.0.1:8080/test.txt) through Postman. Ensure that file is created and 201 Created response returned. Please note that only a file will work. Directory creation was an oversight and not implemented. Pre-existing directory will work.

304 Not Modified: Not implemented.

401 Unauthorized: Perform a GET request to a restricted path (127.0.0.1:8080/protected/) through Postman and ensure that a 401 response is returned.

403 Not Found: Through the test website click view details under 401/403. Attempt to login in random (incorrect) credentials. A 403 response should appear.

404 Not Found: Perform a GET request (127.0.0.1:8080/index.php) through Postman. Ensure that a 404 response is returned

To ensure that threads were being created we used the Thread Test within the test website. The majority of our testing relied on ensuring the website ran properly.

Issues

During implementation we ran into an issue when parsing file paths for htaccess detection. We found (JRob) that the leading / was being removed from the path and as a result the htaccess file was never detected. This was caused by using Java's Path substring method. To fix this we switched to using Java's Path getParent method.

Command line instructions to compile:

In the command prompt, go to the directory where the .java files are stored. Create a directory build. Run the java compilation from the command line:
javac –d ./build *.java
In your build directory should be the class tree, move to the build directory and run
jar cvf WebServer.jar *

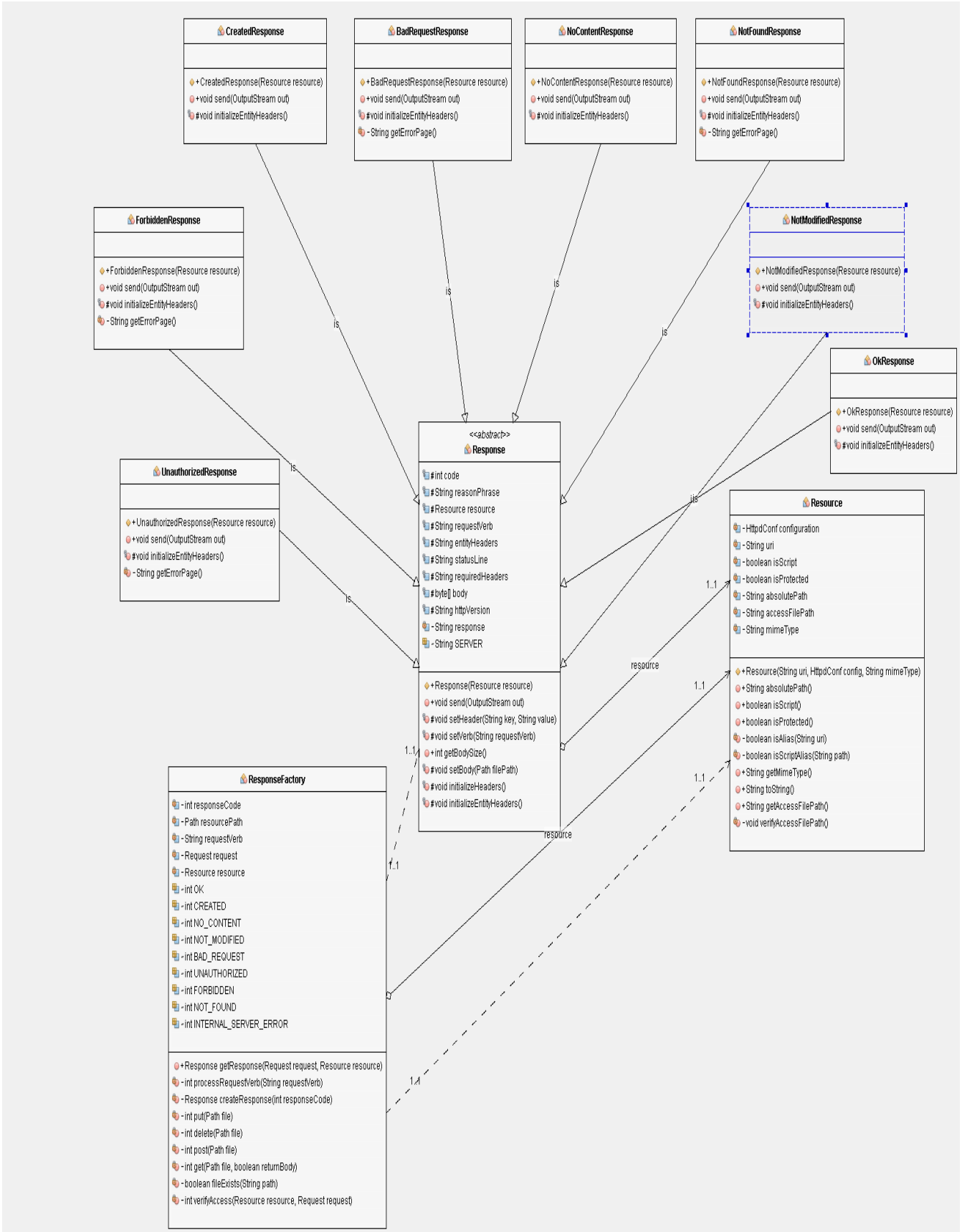
Assumptions:

1. The provided Web Server Project Class Diagram Proposal (v1.2) was roughly accurate and would provide most necessary classes to completely develop the Java Web Server application.
2. The provided workflow diagram (v2.1) demonstrates an accurate work flow of the behavior for the Java Web Server application.
3. Students are expected to have higher than novice level expertise with the Java programming language.
4. Students are expected to spend time researching and understanding HTTP specifications and standards.
5. Software tools necessary to implement and test software are freely accessible to students.
6. Code samples and snippets provided for reference are accurate and error free.

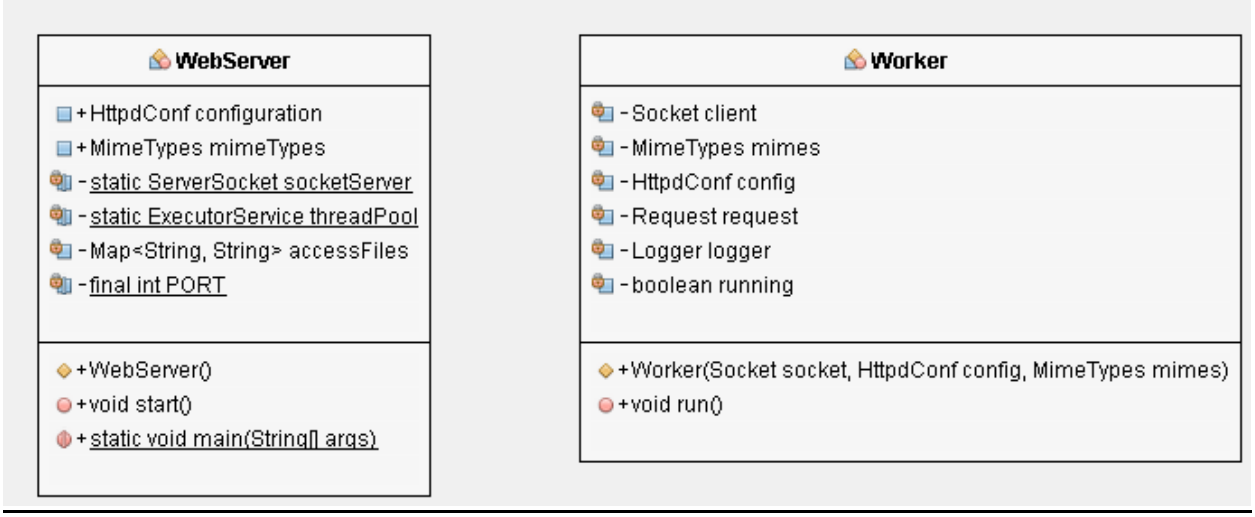
Implementation:

The entry point to the Java web server is the main method in WebServer.java. The server listens to port 8080 by default. WebServer is responsible for setting its configuration via the httpd.conf and MIME.types files located in conf folder located in the server's root directory. The HttpdConf and MimeTypes classes are used by the server to load the configurations and both utilize HashMaps to store their respectful headers. Each is also responsible for the parsing of their respected files. Both HttpdConf and MimeTypes extend ConfigurationReader, a class used to read a file and provide scanning logic, to help in the parsing. The server utilizes the

ExecutorService interface to manage a fixed thread pool of 10 threads, allowing the server to handle multiple simultaneous requests. The Worker class implements Runnable and is passed to the fixed thread pool for execution. The Worker expects to be passed a Socket, HttpdConf, and MimeType objects. The Socket is used with Worker to facilitate communication between the client and server. When WebServer receives a request, the Request class is responsible for handling the parsing and extracting the properly formatted HTTP request's information. Worker also utilizes a ResponseFactory to dynamically create a response to a request. All Responses extend from an abstract Response class. The Response class provides both abstract and implemented methods. The ResponseFactory verifies if a resource is protected and if the resource exists and then based on the appropriate response code, generates the appropriate Response to be sent to the client. See figure 1 for the Response relationship structure. Once a request is sent the Logger class handles the logging of the response.


CreatedResponse
+CreatedResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
BadRequestResponse
+BadRequestResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
- String getErrorPage()
NoContentResponse
+NoContentResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
NotFoundResponse
+NotFoundResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
- String getErrorPage()
ForbiddenResponse
+ForbiddenResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
- String getErrorPage()
NotModifiedResponse
+NotModifiedResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
OkResponse
+OkResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
is
<<abstract>>
Response
#int code
#String reasonPhrase
#Resource resource
#String requestVerb
#String entityHeaders
#String statusLine
#String requiredHeaders
#byte[] body
#String httpVersion
- String response
- String SERVER
+Response(Resource resource)
+void send(OutputStream out)
#void setHeader(String key, String value)
#void setVerb(String requestVerb)
+int getBodySize()
#void setBody(Path filePath)
#void initializeHeaders()
#void initializeEntityHeaders()
UnauthorizedResponse
+UnauthorizedResponse(Resource resource)
+void send(OutputStream out)
#void initializeEntityHeaders()
- String getErrorPage()
Resource
- HttpdConf configuration
- String uri
- boolean isScript
- boolean isProtected
- String absolutePath
- String accessFilePath
- String mimeType
+Resource(String uri, HttpdConf config, String mimeType)
+String absolutePath()
+boolean isScript()
+boolean isProtected()
- boolean isAlias(String uri)
- boolean isScriptAlias(String path)
+String getMimeType()
+String toString()
+String getAccessFilePath()
- void verifyAccessFilePath()
1..1
resource
ResponseFactory
- int responseCode
- Path resourcePath
- String requestVerb
- Request request
- Resource resource
- int OK
- int CREATED
- int NO_CONTENT
- int NOT_MODIFIED
- int BAD_REQUEST
- int UNAUTHORIZED
- int FORBIDDEN
- int NOT_FOUND
- int INTERNAL_SERVER_ERROR
+Response getResponse(Request request, Resource resource)
- int processRequestVerb(String requestVerb)
- Response createResponse(int responseCode)
- int put(Path file)
- int delete(Path file)
- int post(Path file)
- int get(Path file, boolean returnBody)
- boolean fileExists(String path)
- int verifyAccess(Resource resource, Request request)

WebServer and Worker Class



Results & Conclusions:

This project provided us with many challenges but overall allowed us to grow into better Computer Science students. Because of the scope of the Web Server project and all its moving parts we learned the importance of code organization and writing maintainable code. Using properly named variables, methods and class names allowed us to better navigate our own code. We learn how to implement the Factory pattern via the ResponseFactory. We also learned the importance of planning out the project before you start coding. One of the main challenges was taking such a big project with many moving parts and learning to break it down into more manageable smaller parts. The learning curve for the project was high as we had no experience implementing an http server from the bottom up. Overall the project was beneficial to our continued growth as students and software developers.