CSSE432 Web Server Project

Final Report

Team 3
Robbie Cooper
Grant Smith
Caleb Post

Server Implementation

Language

The server that we implemented was done using Java. This let us create a platform independent server that we could run and test on any operating system with Java installed. It provided us with a platform that all three of us were relatively familiar with and with a platform that abstracted out much of the socket connection setup so that we could deal with the issues of a web server and not with the issues of socket connections.

Sockets

There are two different types of sockets used in our implementation. These are both supplied in the Java.net library. The first is a `ServerSocket` which we create and keep open at the port specified at runtime when the server is started. The `ServerSocket` automatically provides the functionality of staying alive and waiting for connections incoming to its port. Connections that are made are then made into a new `Socket` connection that we use for all content requested by that client. This satisfies the requirement that we have persistent connections.

Each time a client connects to the server and is granted a socket, the server spawns a new thread to deal with requests from that client. This lets our server handle multiple clients connected at once simultaneously requesting information from the server.

Headers

The headers that our server responds to are the Host, User-Agent, Connection, Accept, Cookie and Accept-Language headers. It responds to these with the Date, Server, Content-Length, Connection, and Content-Type headers.

Host

If the user-agent does not specify a Host header in their request, as per the RFC, our server returns a 400 Bad Request.

User-Agent

The user agent header field is used in our server as a point of identifying what client is asking for information. It would be used to restrict content, but all the tools that we have to test our server far exceed the capabilities of our server and so we do not need to restrict access to content based on the request.

Connection

The connection header is used in our server to determine whether a connection should be kept alive or closed. If a client specifies that they want a connection type `close` then after their

request is fielded we then close the socket that their request was handled on. Otherwise we leave the socket open for further requests until the socket times out.

Cookie

Cookie header is handled as part of our persistent state. The client passes any cookie(s) that they have using this header and then we deal with them on the server side as part of persistent state.

Accept

The accept header is handled in our server so that if a client requests content that is determined by our server to not be acceptable by the client then we respond using a 406 Not Acceptable error code.

Accept-Language

The Accept-Language header is handle by the server to provide the client with an appropriate, localized version of a file, if one exists. The client will send a language code to the server and the server will take the requested, default file name, along with the language code, and compare them against the available localized versions of the requested file. If such a file exists, the server modifies the request to reflect the client's language preference and proceeds as normal. Otherwise, the English version of the file is served to the client by default.

Response Codes

Our server can respond with the following HTTP response codes in these situations.

200 OK - This is the response when a valid request is made to the server.
301 Moved Permanently - This is the response made by our server when a request is made to a file which is within a permanently moved directory list stored in a file on the server in the moved.txt file.
400 Bad Request - This is the response made by our server when a header field is corrupted in structure or the Host header field is missing in a request.
403 Forbidden - This is the response made by our server when an unauthorized user attempts to request content. An unauthorized user is defined by a user whose IP is not present in the .htacess file
404 Not Found - This is the response made by our server when content requested by client is not found on the server.
406 Not Acceptable - This is the response made by our server when content requested by client is not accepted by the client as defined by their Accept header in their request.
500 Internal Server Error - This is the response when the server has an error that was caused by a bug in our code. We also specifically throw this error on a POST request if the `-failPostTo500` debug flag is specified at startup of the server.
501 Not Implemented - This is the response when the request method is not supported by the server. The server only supports GET, HEAD, and POST.

High-Level Architecture - Connection Handling Flow

The server listens on port 8000 (by default) for all incoming TCP connections on port 80. When a connection is made, a new thread is created to handle it. This way, multiple connections can be handled at once. All of this happens in the `ConnectionAccepter` class.

Next, a `ConnectionHandler` runs. This procedure opens the socket for reading and passes the raw input into the Request class for parsing. The request class returns a Request object with structured data. The type of the instance is a subclass of Request that reflects the HTTP method used to make the request (`GetRequest`, `PostRequest`, `HeadRequest`, etc).

Once `ConnectionHandler` has the `Request` object, it passes to the `Response` class, which generates the appropriate response object. Like the requests, the instance type of the response is a subclass of `Response`, but reflects the kind of response returned. If the response was not idempotent (such as the response for POST), the action is taken during creation of this instance.

Finally, `ConnectionHandler` writes the response to the socket and then closes it if keep-alive is off. Otherwise, it continues listening to the same socket for more requests.

Errors and irregular control flow

If there is a problem parsing the `Request`, the `Request` throws an exception depending on what the problem was. The `ConnectionHandler` catches the exception and writes the appropriate 400 or 500 response. Also, exceptions are used to trigger 300 responses despite not actually being problems.

If there is a problem generating the `Response`, the `Response` class throws exceptions which are handled by `ConnectionHandler` in a similar way, producing the appropriate 500 response.

Stateful Connections

Statefulness is implemented in our server through the use of cookies. Since there is no authentication directly in our web server we use the cookies to log the user's history of requests on the server.

At first request to the server Set-Cookie is specified by our server with a new cookie and a value of 1. This represents the user agent having requested one file from the server. For each successive request this cookie is then sent to the server and incremented to reflect the increased number of items requested by the client.

Security Component

The security component in our server is in the form of a software implemented firewall. We cross reference any request from a client to our server with the .htaccess file. If their IP is not

specified in the .htaccess file then any requests from that client are ignored by our server. They are served a 403 Forbidden response and their connection to the server is closed.

In addition to this, access to the .htaccess file is restricted. No client can access this file, even if they are an allowed user through the firewall implementation.

Command Line Startup Options

-debug : Turns on debugging output
-failPostTo500 : Causes post to fail with a 500 response for testing purposes
-root ROOT : Sets the document root to the directory specified by ROOT
-p PORT : Causes the server to listen on the port number PORT