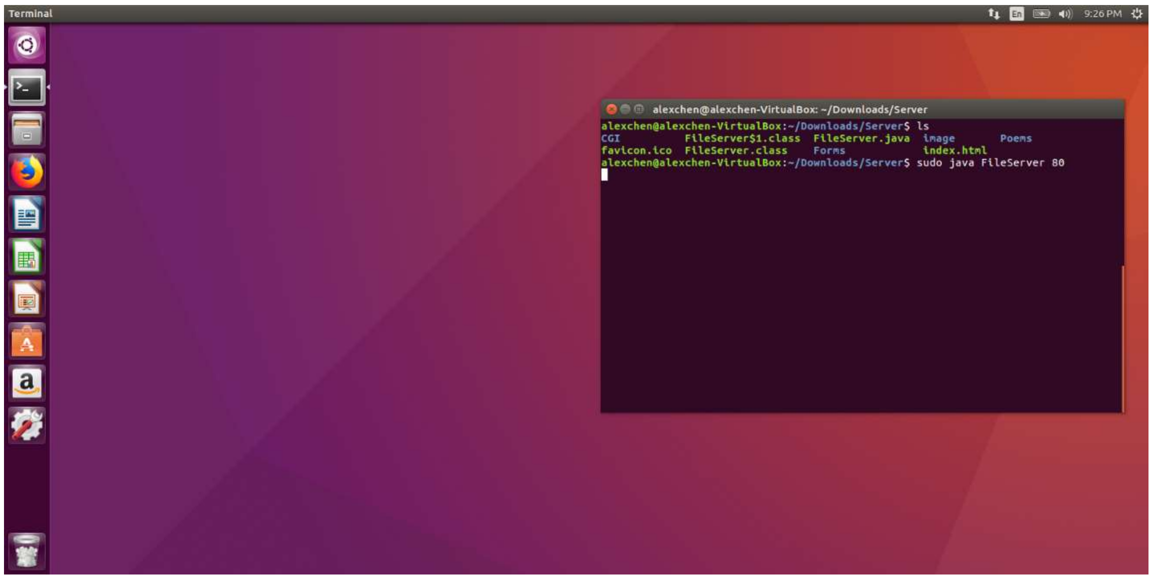
CS560 Project 1 Report

John Till and Peiyong Chen

Overview

The goal of this project is to implement an HTTP server which supports directory listing, static files, and CGI scripts. To motivate the server, we decided to host a collection of poetry. We use Java and its standard library to implement the server. The main testing environment is on Linux (a Ubuntu virtual machine). The server has also been tested on Window 10, which supports all features except for the CGI script because it is implemented by a Linux shell script. In this report, we will show how we run the code and how we implemented the required features. The Java code "FileServer.java" and hosted server files are included in our submission.

To test the server, compile the server code with "javac FileServer.java", then run the server with "java FileServer 80". By running the code, a server socket is linked to the port 80, as shown in Figure 1. You can connect to the server in a web browser on the same machine by entering "localhost" in the address bar.



```
alexchen@alexchen-VirtualBox:~/Downloads/Server$ ls
CGI          FileServer$1.class  FileServer.java  image        Poems
favicon.ico  FileServer.class    Forms            index.html
alexchen@alexchen-VirtualBox:~/Downloads/Server$ sudo java FileServer 80
```

Figure 1

Design and Results

Feature 1: Single connection mode and multiple connection mode (parallel with multiple threads)

The code to handle connections uses Java ServerSocket and Socket objects as shown in the Java Socket Tutorial found on the course schedule [1]. A ServerSocket object is created once in the lifetime of the server, and each client connect has a Socket object created by calling ".accept()" on the ServerSocket object. Whenever a Socket is created, we call a custom method "resolveClientConnection(Socket clientSocket)" to handle the connection. Thus the multithread feature is trivial- when the server is in multi-thread mode it calls “resolveClientConnection” in a new Java Thread instead of the main one. The steps we describe here are implemented in the following main function:

```
public static void main(String[] args) throws IOException {

	//Default to single-thread mode if only a port number is provided
	boolean multithread_mode = false;

	if (args.length == 2) {
		//Allow a second command line argument for multithread mode
		multithread_mode = Boolean.parseBoolean(args[1]);
	}else if(args.length != 1){
		System.err.println("Usage: java FileServer <port> <multithread 
mode>");
		System.exit(1);
	}

	int portNumber = Integer.parseInt(args[0]);
	ServerSocket serverSocket = new ServerSocket(portNumber);

	//Build directories of static files in html once on startup
	generateHtmlDirectories("/Poems", "");

	boolean keep_running = true;
	while(keep_running){
		Socket clientSocket = serverSocket.accept();

		if( !multithread_mode ){
			//Resolve the connection in the main thread
			resolveClientConnection(clientSocket);
		}else{
			//Fork a new thread and continue to wait for connections
			Thread t1 = new Thread(new Runnable() {
				public void run() {
					resolveClientConnection(clientSocket);
				}
			});
			t1.start();
		}
	}
}
```

We have run the server in both modes without incident.

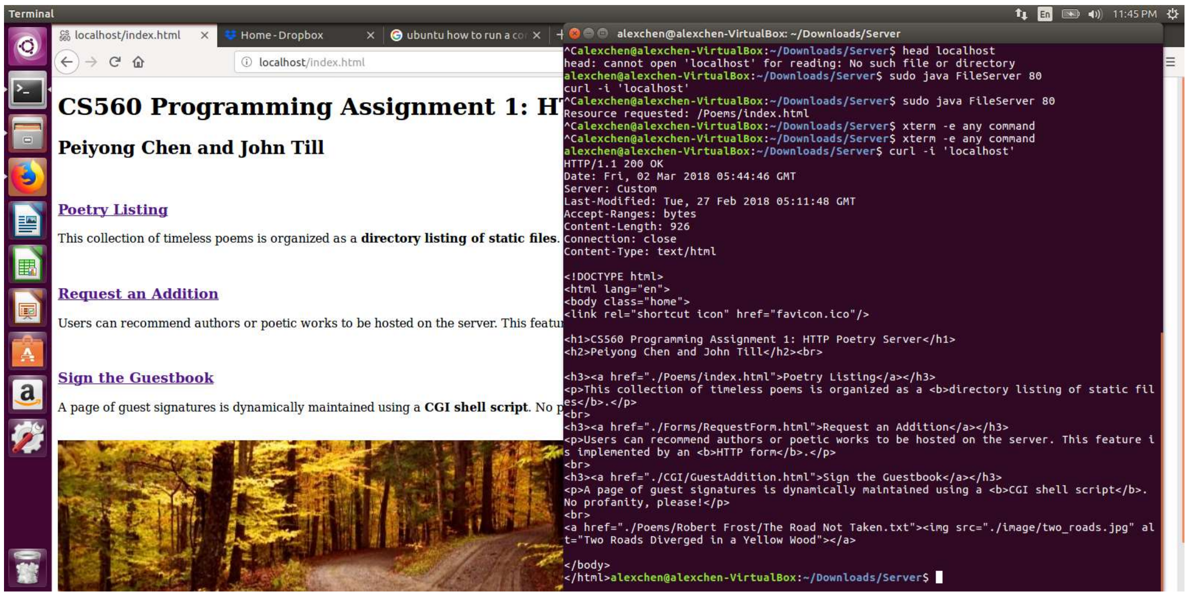
Feature 2: HTTP GET requests with query and header parsing

We relied on basic String manipulations to implement GET requests and response messages. The server uses a Scanner to check that the HTTP message starts with the word “GET,” and sends a “Method Not Allowed” error response for any other types of client messages. If the type of message is “GET”, the server scans the next word to find which resource is being requested. The server handles several cases, which include:

- The resource requested is the server homepage, “/”
- The resource requested is a form submission, “/Forms/request_form? … ”
- The resource requested is a CGI script, “/CGI/guest_addition? … ”
- The resource requested is a static file which exists in the server subdirectories, and has a whitelisted extension type

Based on the case the server may perform some intermediate work, and eventually will send a response.

As shown in Figure 2, after a client sends a request, we can see the HTTP requests and headers in the console.


Terminal
localhost/index.html
Home - Dropbox
ubuntu how to run a co
localhost/index.html
CS560 Programming Assignment 1: HT
Peiyong Chen and John Till
Poetry Listing
This collection of timeless poems is organized as a directory listing of static files.
Request an Addition
Users can recommend authors or poetic works to be hosted on the server. This featu
Sign the Guestbook
A page of guest signatures is dynamically maintained using a CGI shell script. No p
alexchen@alexchen-VirtualBox: ~/Downloads/Server
11:45 PM

```
^Calexchen@alexchen-VirtualBox:~/Downloads/Server$ head localhost
head: cannot open 'localhost' for reading: No such file or directory
alexchen@alexchen-VirtualBox:~/Downloads/Server$ sudo java FileServer 80
curl -i 'localhost'
^Calexchen@alexchen-VirtualBox:~/Downloads/Server$ sudo java FileServer 80
Resource requested: /Poems/index.html
^Calexchen@alexchen-VirtualBox:~/Downloads/Server$ xterm -e any command
^Calexchen@alexchen-VirtualBox:~/Downloads/Server$ xterm -e any command
alexchen@alexchen-VirtualBox:~/Downloads/Server$ curl -i 'localhost'
HTTP/1.1 200 OK
Date: Fri, 02 Mar 2018 05:44:46 GMT
Server: Custom
Last-Modified: Tue, 27 Feb 2018 05:11:48 GMT
Accept-Ranges: bytes
Content-Length: 926
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<body class="home">
<link rel="shortcut icon" href="favicon.ico"/>

<h1>CS560 Programming Assignment 1: HTTP Poetry Server</h1>
<h2>Peiyong Chen and John Till</h2><br>

<h3><a href="./Poems/index.html">Poetry Listing</a></h3>
<p>This collection of timeless poems is organized as a <b>directory listing of static fil
es</b>.</p>
<br>
<h3><a href="./Forms/RequestForm.html">Request an Addition</a></h3>
<p>Users can recommend authors or poetic works to be hosted on the server. This feature i
s implemented by an <b>HTTP form</b>.</p>
<br>
<h3><a href="./CGI/GuestAddition.html">Sign the Guestbook</a></h3>
<p>A page of guest signatures is dynamically maintained using a <b>CGI shell script</b>.
No profanity, please!</p>
<br>
<a href="./Poems/Robert Frost/The Road Not Taken.txt"><img src="./image/two_roads.jpg" al
t="Two Roads Diverged in a Yellow Wood"></a>

</body>
</html>alexchen@alexchen-VirtualBox:~/Downloads/Server$
```

Figure 2

The files in our local server are listed in Figure 3. Then we go to the Firefox browser and type in “localhost” to send out a request to connect the server. In our browser we can view the content “/index.html” which is sent back from the server after the request from client (us). The browser also fetches any images referenced in the HTML file, as well as the “favicon.ico” to display in the tab. The browser view is shown in Figure 4.

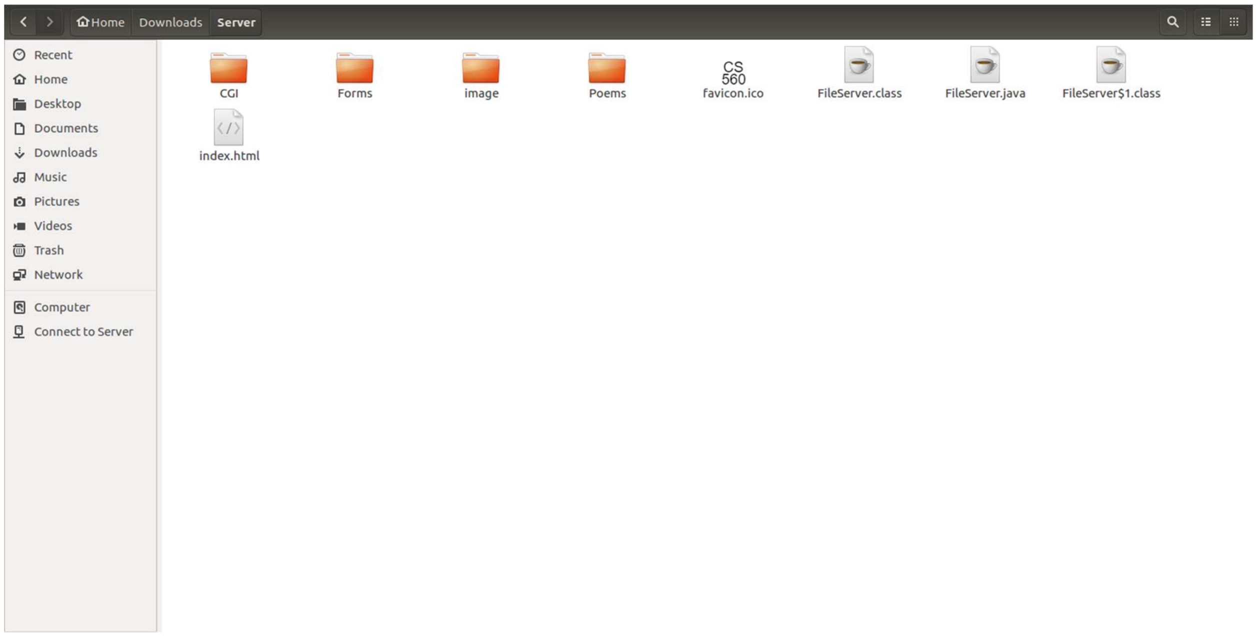
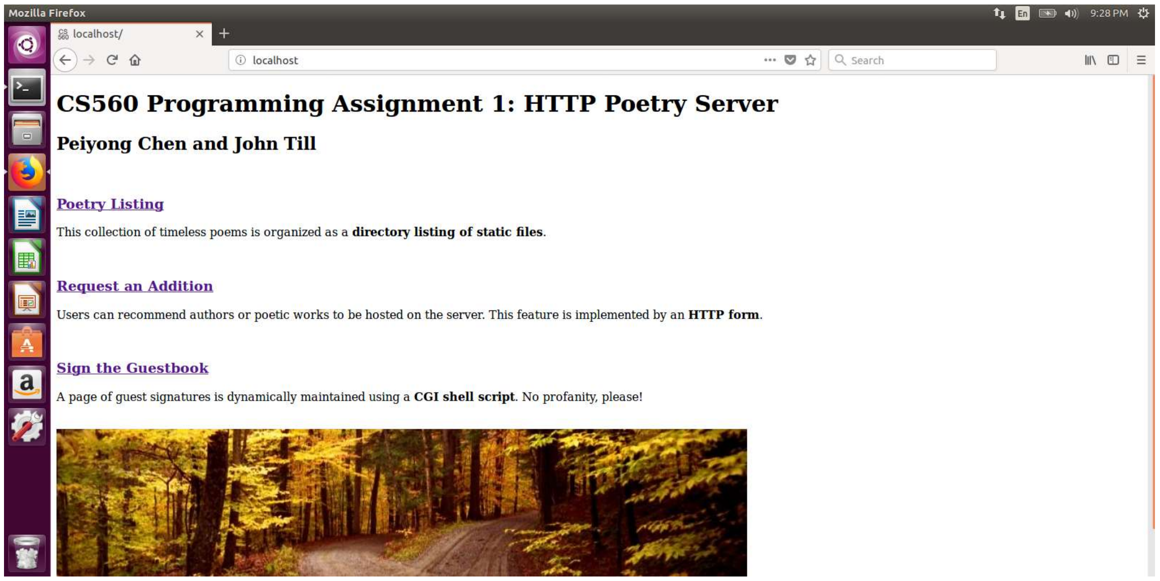


Figure 3



CS560 Programming Assignment 1: HTTP Poetry Server

Peiyong Chen and John Till

Poetry Listing

This collection of timeless poems is organized as a **directory listing of static files**.

Request an Addition

Users can recommend authors or poetic works to be hosted on the server. This feature is implemented by an **HTTP form**.

Sign the Guestbook

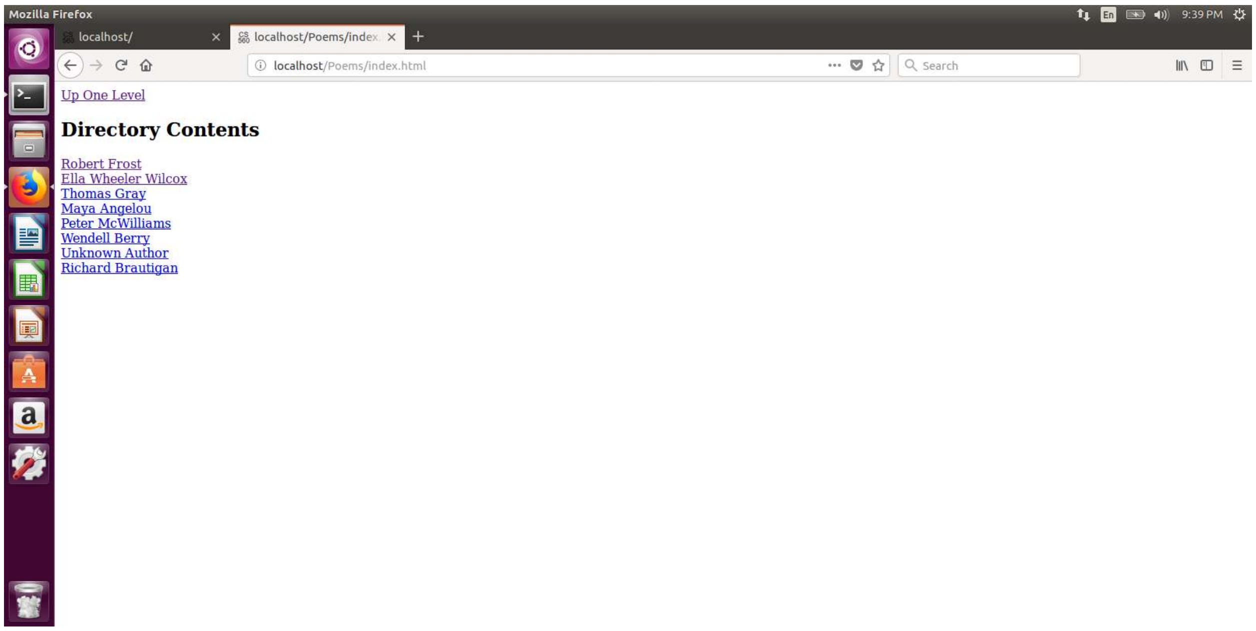
A page of guest signatures is dynamically maintained using a **CGI shell script**. No profanity, please!

Figure 4

Feature 3: Automatic directory listing

On server startup, the contents of "/Poems" are recursively scanned and recorded in "index.html" files which can be sent to clients. This allows clients to browse through our directory structure.

As the client, if we left click the link "Poetry Listing" from the main page, we can go to Directory Contents, which will list all the poetry documents of the local server, as shown in Figure 5.



Up One Level

Directory Contents

Robert Frost
Ella Wheeler Wilcox
Thomas Gray
Maya Angelou
Peter McWilliams
Wendell Berry
Unknown Author
Richard Brautigan

Figure 5

Feature 4: Static file transport

We implement a method

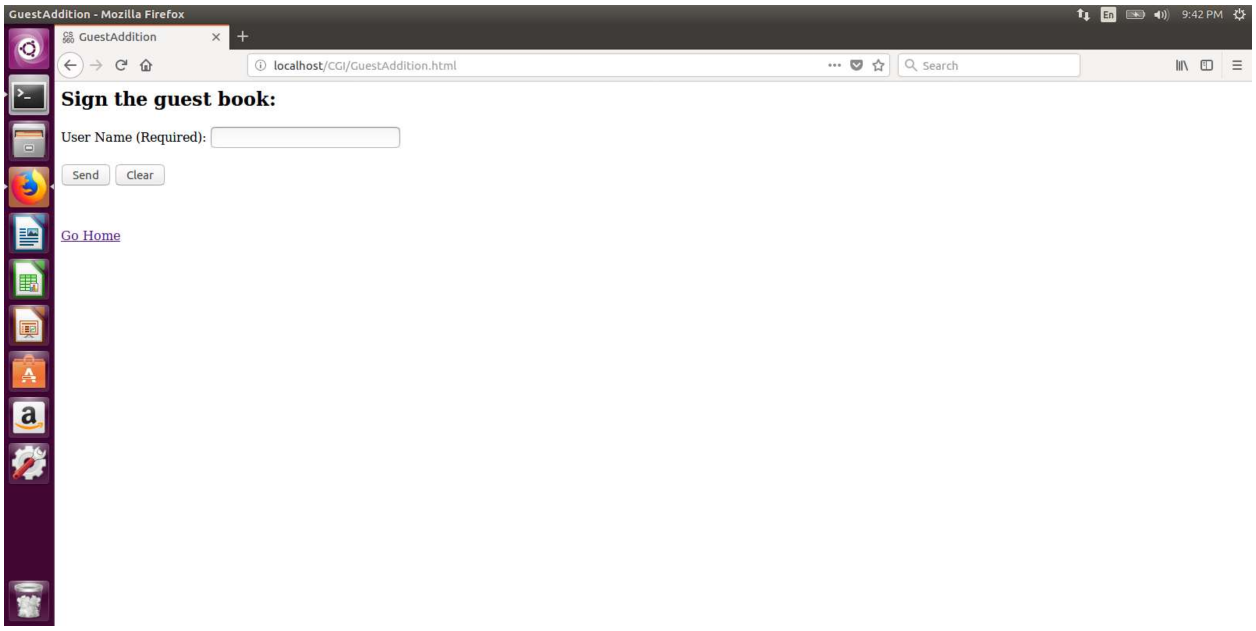
“convertResourceToResponse(String resource, PrintWriter print_out, OutputStream byte_out)”

to convert the specified resource into an HTTP response. Resource types must be in a whitelist of “txt”, “jpg”, “ico”, and “html” files. This means a client cannot request “FileServer.java” and see our source code. If the request resource exists and is in the whitelist, an HTTP response header and the resource are sent to the client using the output streams.

Feature 5: Basic CGI support by running a sample CGI script on the server side

The CGI script implements a guest book feature for the site. A single name is received by an HTTP form. This name is the input for a shell script which appends to name to a local file “guestbook.txt” then shows the whole guestbook in a dynamically generated HTML response.

Screenshots are shown in Figure 6-9. If we type in the “User Name”, such as “Alexchen”, the name “Alexchen” is attached to the tail of the “Guest Book” list. If we further type in the “Alexchen1”, “Alexchen1” will be attached to the tail of the list. The “Guest Book” file is located in the Server. By this we write our input to a local file.


GuestAddition - Mozilla Firefox
GuestAddition
localhost/CGI/GuestAddition.html
Search
9:42 PM

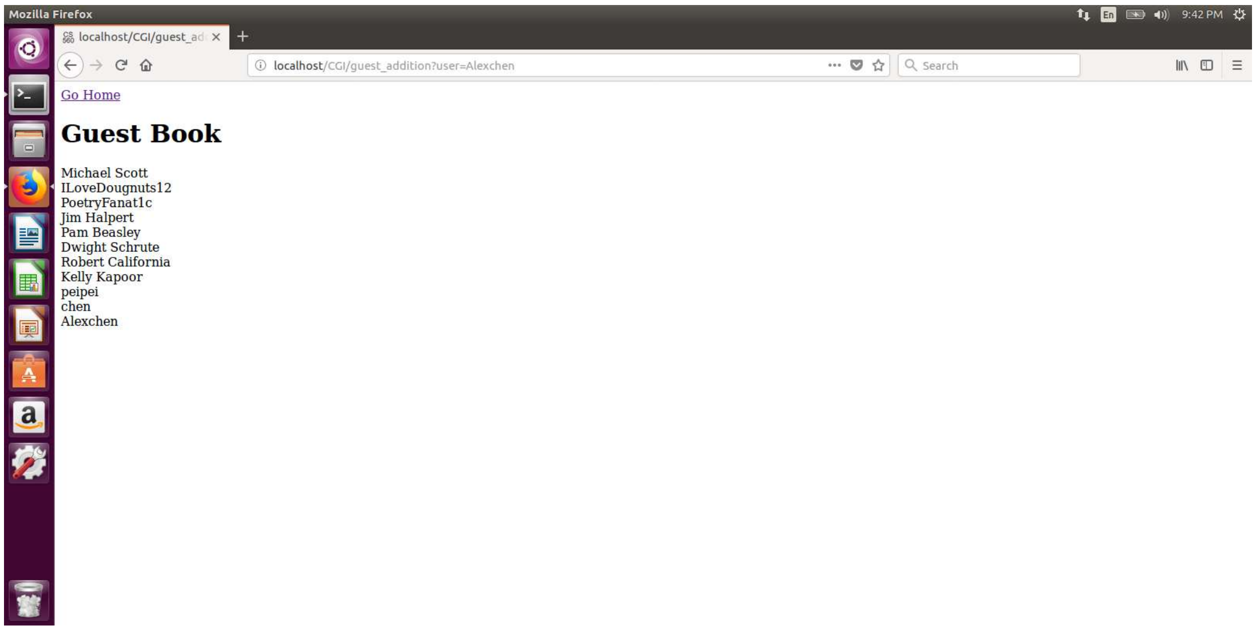
Sign the guest book:

User Name (Required):

Send Clear

Go Home

Figure 6

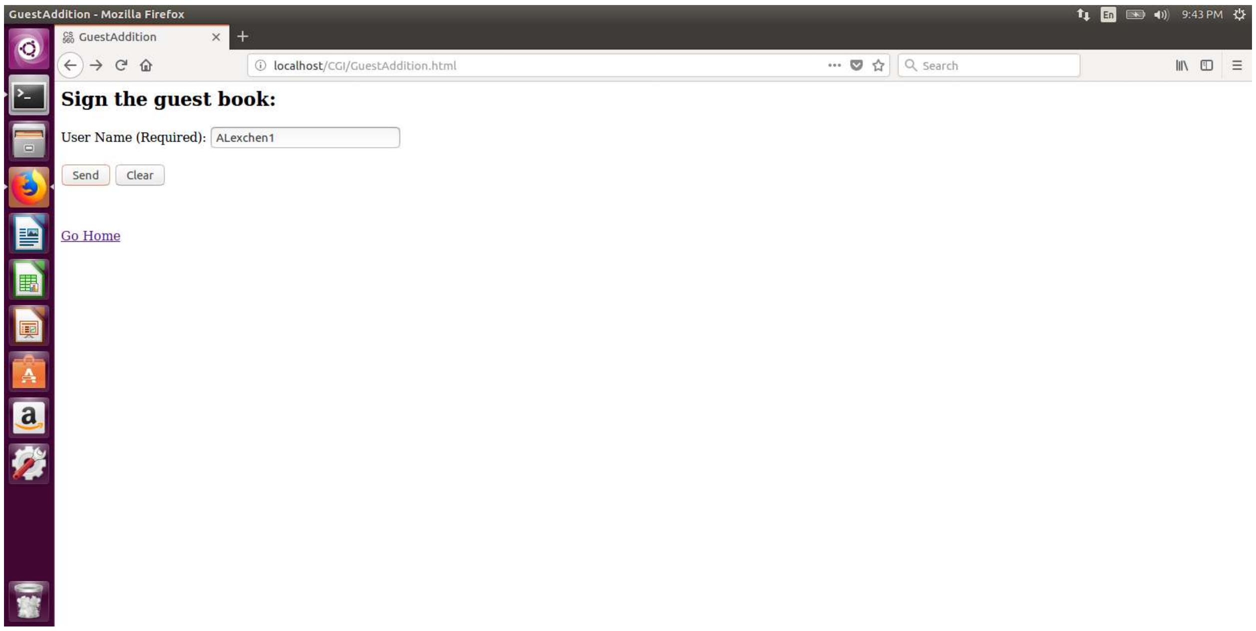


Go Home

Guest Book

Michael Scott
ILoveDougnuts12
PoetryFanat1c
Jim Halpert
Pam Beasley
Dwight Schrute
Robert California
Kelly Kapoor
peipei
chen
Alexchen

Figure 7


GuestAddition - Mozilla Firefox
GuestAddition
localhost/CGI/GuestAddition.html
Search
9:43 PM

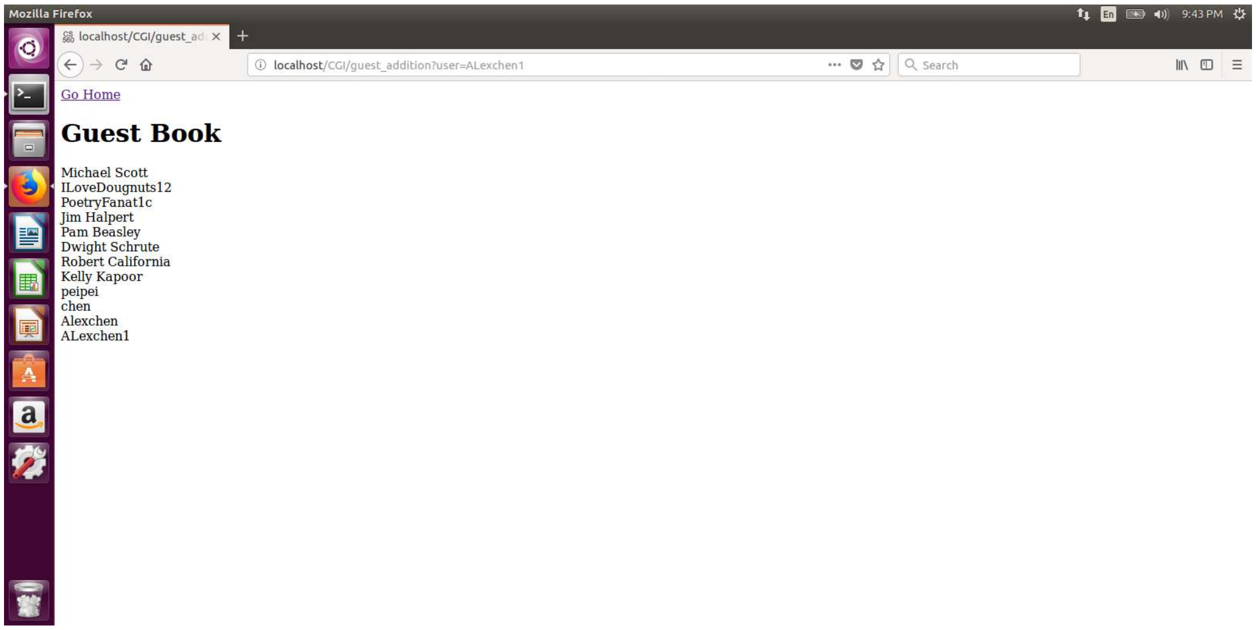
Sign the guest book:

User Name (Required): ALexchen1

Send Clear

Go Home

Figure 8



Go Home

Guest Book

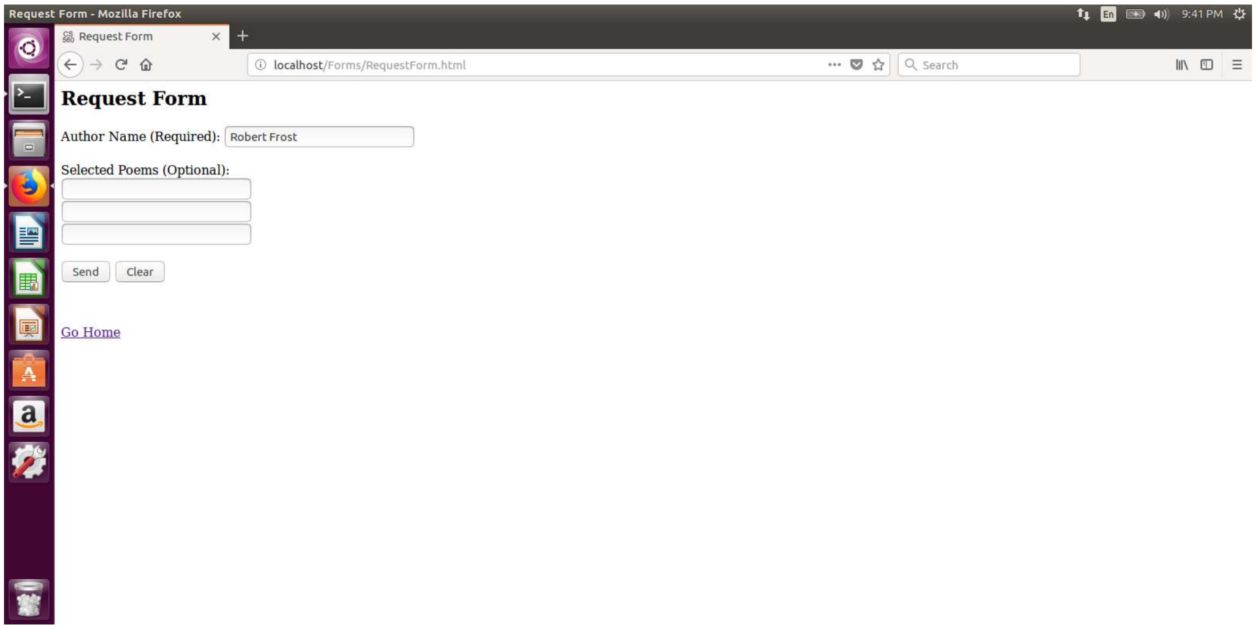
```
Michael Scott
ILoveDougnuts12
PoetryFanat1c
Jim Halpert
Pam Beasley
Dwight Schrute
Robert California
Kelly Kapoor
peipei
chen
Alexchen
ALexchen1
```

Figure 9

Extra credit: Form submission with results written into a local file

We implemented another form so that users may request additional poetic works to be added to our collection. This is implemented by an HTML form, and the results are appended to a local file "Suggestions.txt." The client receives a response of a static confirmation page.

If we click the "Request an addition", we will go the page which is implemented by a HTML form. We can input the author's name and get a confirmation response. For example, if we type in "Robert Frost" and enter, the server will send a response "Thanks for your input". Both are shown in Figure 10 & 11.


Request Form - Mozilla Firefox

localhost/Forms/RequestForm.html

Request Form

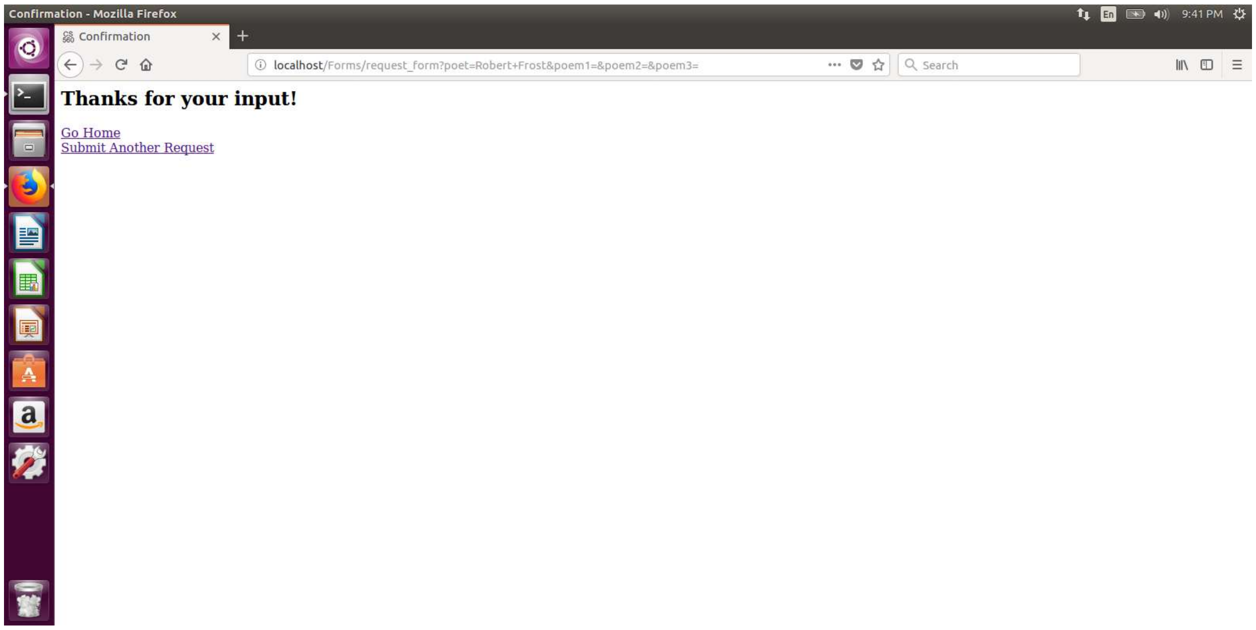
Author Name (Required): Robert Frost

Selected Poems (Optional):

Send Clear

Go Home

Figure 10


Confirmation - Mozilla Firefox

localhost/Forms/request_form?poet=Robert+Frost&poem1=&poem2=&poem3=

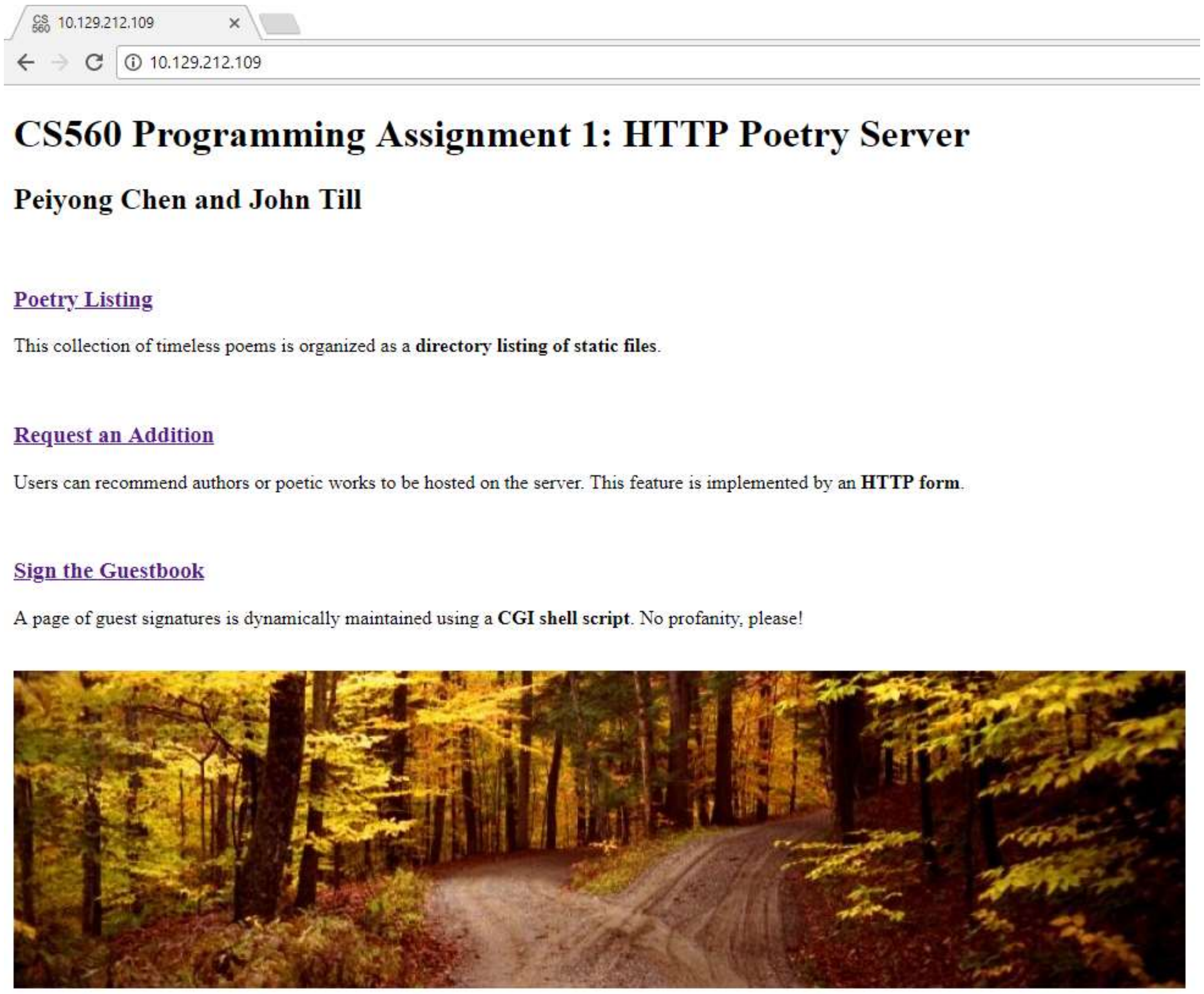
Thanks for your input!

Go Home
Submit Another Request

Figure 11

Further Testing: Network connection

We also connected to the server from a different machine on the "ut-open" network by specifying the server's ip-address. This test was performed in Google's Chrome web browser, and the result was successful. The test result is shown in Figure 12.



10.129.212.109

CS560 Programming Assignment 1: HTTP Poetry Server

Peiyong Chen and John Till

Poetry Listing

This collection of timeless poems is organized as a **directory listing of static files**.

Request an Addition

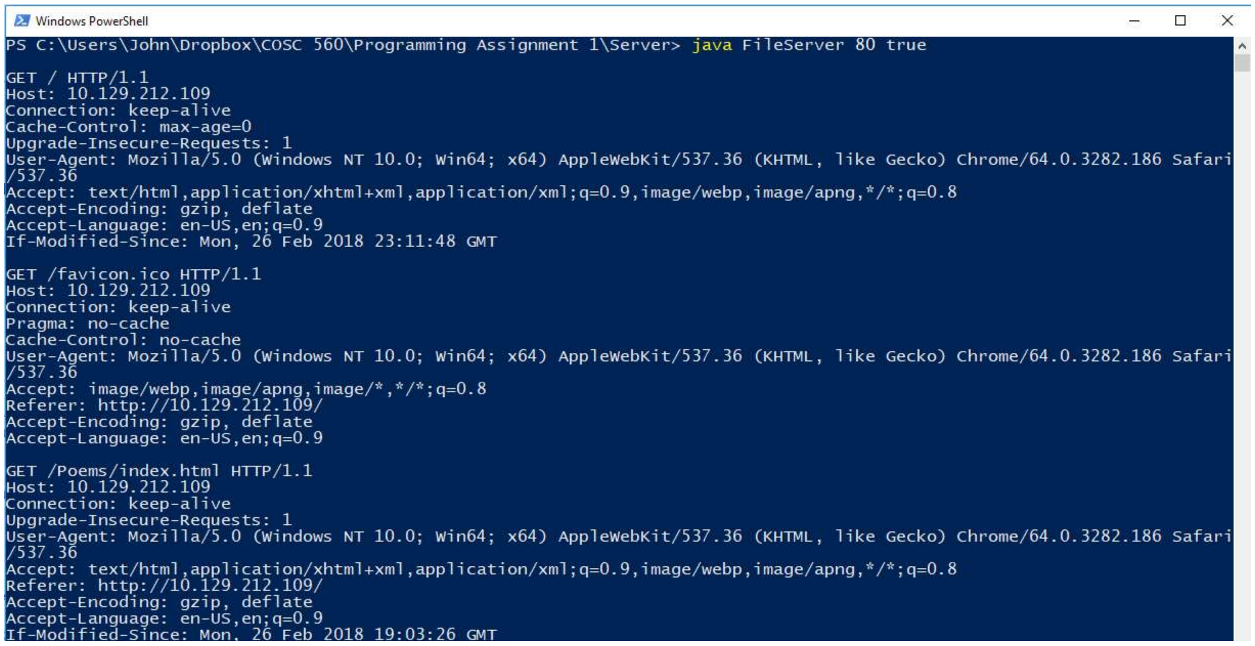
Users can recommend authors or poetic works to be hosted on the server. This feature is implemented by an **HTTP form**.

Sign the Guestbook

A page of guest signatures is dynamically maintained using a **CGI shell script**. No profanity, please!

Figure 12

The server prints the client messages to a console (shown in Figure 13):



```
PS C:\Users\John\Dropbox\COSC 560\Programming Assignment 1\Server> java FileServer 80 true

GET / HTTP/1.1
Host: 10.129.212.109
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari
/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Mon, 26 Feb 2018 23:11:48 GMT

GET /favicon.ico HTTP/1.1
Host: 10.129.212.109
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari
/537.36
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: http://10.129.212.109/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GET /Poems/index.html HTTP/1.1
Host: 10.129.212.109
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari
/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://10.129.212.109/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Mon, 26 Feb 2018 19:03:26 GMT
```

Figure 13

Works Cited

[1] “Lesson: All About Sockets.” Web address: docs.oracle.com/javase/tutorial/networking/sockets.