CS2003 Practical Web2 - Directory Lister

Saleem Bhatti

13 November 2018

Deadline: 28 November 2018 (please see MMS)

Description

Your task is to build a web interface to a remote file space, to allow directory contents to be listed with file details. The client and user interface will be provided by a web-browser, and the application server will be a node.js server.

Technologies you will need to use:

- HTML5 for web pages.
- CSS3 for presentation of web pages.
- JavaScript for client-side processing.
- JavaScript for server-side processing with node.js.

---

1 Basic Requirements

Using the files given (see Section 5, below) you are required to change and extend the code as necessary to perform the following tasks:

(B1) Allow the contents of a remote directory – a logical root – to be viewed. It should not be possible to browse to the parent directory of this root.

(B2) For the remote server that is accessed, the server name and the directory being accessed should be shown in the user interface.

(B3) Allow any subdirectories below the root directory to be navigated and viewed, one directory at a time. Navigation should also be allowed back up to the parent directories, and ultimately to the root.

(B4) For any directory, file details should be listed, including file name, type, size, and the times for access, modification, and creation of files. These should be presented in a sensible layout, such as a table-like presentation.

(B5) Allow the user to select which file details are visible.

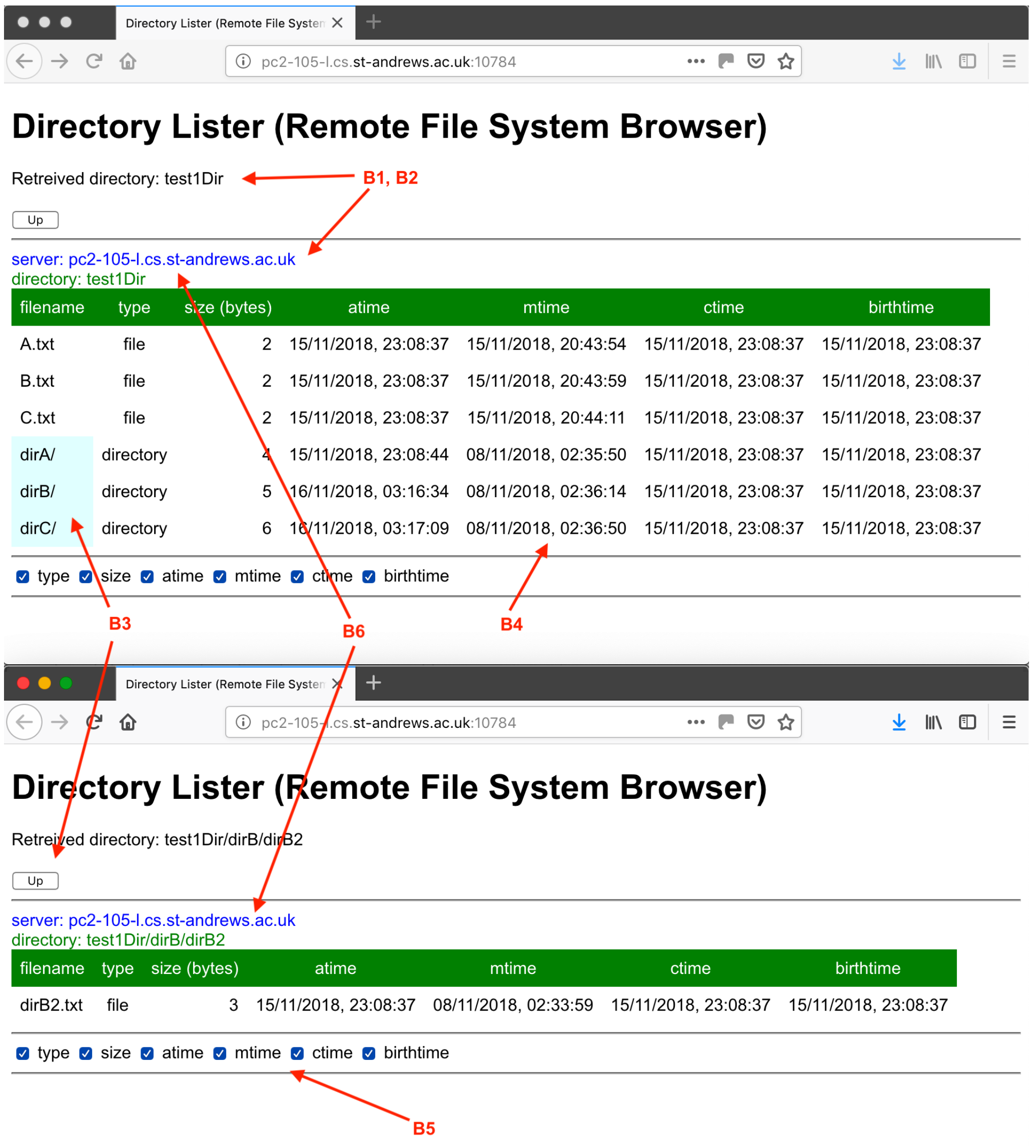
(B6) Allow multiple clients to browse the same remote file-space via the same server, simultaneously, and independently of each other.

(B7) Your communication protocol must conform with the specification shown in Section 6. Communication between client and server should be using WebSockets.

An incremental approach is recommended — make sure that each part works before going on to the next part. (Hint: Make a safe copy of whatever you have got working so far, so that you always have a working solution to submit, while you are developing the next part of your solution.)

1.1 Possible layout of user interface in browser

Figure 1 shows a possible layout that you can use as a starting point for your own design. Note in the figure that the red text and lines indicate where implementation of *Basic Requirements* B1 to B6 are visible.


Directory Lister (Remote File System
pc2-105-l.cs.st-andrews.ac.uk:10784
Up
B1, B2
B3
B6
B4
B5

Directory Lister (Remote File System Browser)

Retreived directory: test1Dir

server: pc2-105-l.cs.st-andrews.ac.uk
directory: test1Dir

filename	type	size (bytes)	atime	mtime	ctime	birthtime
A.txt	file	2	15/11/2018, 23:08:37	15/11/2018, 20:43:54	15/11/2018, 23:08:37	15/11/2018, 23:08:37
B.txt	file	2	15/11/2018, 23:08:37	15/11/2018, 20:43:59	15/11/2018, 23:08:37	15/11/2018, 23:08:37
C.txt	file	2	15/11/2018, 23:08:37	15/11/2018, 20:44:11	15/11/2018, 23:08:37	15/11/2018, 23:08:37
dirA/	directory	4	15/11/2018, 23:08:44	08/11/2018, 02:35:50	15/11/2018, 23:08:37	15/11/2018, 23:08:37
dirB/	directory	5	16/11/2018, 03:16:34	08/11/2018, 02:36:14	15/11/2018, 23:08:37	15/11/2018, 23:08:37
dirC/	directory	6	16/11/2018, 03:17:09	08/11/2018, 02:36:50	15/11/2018, 23:08:37	15/11/2018, 23:08:37

☑ type ☑ size ☑ atime ☑ mtime ☑ ctime ☑ birthtime

Directory Lister (Remote File System Browser)

Retreived directory: test1Dir/dirB/dirB2

server: pc2-105-l.cs.st-andrews.ac.uk
directory: test1Dir/dirB/dirB2

filename	type	size (bytes)	atime	mtime	ctime	birthtime
dirB2.txt	file	3	15/11/2018, 23:08:37	08/11/2018, 02:33:59	15/11/2018, 23:08:37	15/11/2018, 23:08:37

☑ type ☑ size ☑ atime ☑ mtime ☑ ctime ☑ birthtime

Figure 1: Two Firefox browsers simultaneously connected to the same server.

2 Additional Requirements

Please make sure that you have a working submission fulfilling the Basic Requirements before you attempt any of the Additional Requirements listed below.

Below are some suggestions for additional functionality that you can chose to implement in your submission. (Difficulty is indicated in brackets.)

(A1) In your report, discuss the security and privacy issues with your application, especially with respect to file access. *(Easy.)*

(A2) Allow the client-side file listings to be sorted in forward or reverse order for any of the columns, e.g. by filename, by size, modification time, etc. *(Easy to Medium.)*

(A3) Allow a search for files at the server based on a sub-string search. *(Medium to Hard.)*

This will require modifications to:

- to the server, to perform the search and collate the results to send back to the server.
- to the client, so that it can display search results and navigate to the files that match the search.

However, it should be possible to re-use much of the protocol specification already defined, with minor modifications to allow a new 'search' request, and the return of search results.

This becomes a very difficult requirement if you combine it with A4 (below), for searching across servers.

(A4) Allow a server program to discover other servers, and for those to be accessible to the client for navigation also. *(Hard.)*

This will require modifications:

- to your server, which will need to act as the access point to the other instances of servers.
- to the client, so that it knows of the existence of other servers.
- to the communication protocol between client and server, as well as new communication protocols for discovery and server-to-server communication.

(A5) Allow file downloads from the server of any selected file. *(Hard.)*

(A6) Allow file uploads to the server. *(Hard.)*

3 Submission

Your MMS submission should be a single `.zip`, or `.tar.gz` file containing:

(a) Any HTML, CSS, and JavaScript files. A single directory, called `cs2003-web2` (which may have subdirectories for resources test directories and files), with all the source code needed to run your application. Your submission should have server-side and client-side code. It should be possible to take your submitted directory, copy it to School filespace, and for it to be accessible via the School linux lab machines.

Please note the following important items:

- You must not use any external / third-party libraries or source code (whether HTML, CSS3, or JavaScript) in your solution. For *Additional Requirements*, you may need to use additional node.js modules, but you should make clear what they are.
- Please do not use absolute path-names for files or URLs in your submission. You should not need to access any external resources from your code. So, all your HTML, CSS3, and JavaScript files (as well as any other files, such as test files) must all be in the single directory that you submit.
- If the marker cannot run the code you have submitted on the Linux workstations in the lab, and access it via the School web-server from Firefox web-browser, then you will be penalised in marking.

(b) Report. A single PDF file as your report, with the information listed below. Where appropriate, try to concentrate on why you did something (design decisions) rather than just explaining what the code does. In your report, please include:

- A simple guide to running your application, including if you have used any additional node.js modules.
- A summary of the operation of your application indicating the level of completeness with respect to the *Basic Requirements* and *Additional Requirements* listed above.

- Suitable, simple, diagrams, as required, showing the operation of your program, especially if you have designed and implemented any of the *Additional Requirements*.
- An indication of how you tested your application, including screenshots of the web pages showing your application working. If you have designed and implemented any of the *Additional Requirements*, then testing and evidence of that functionality should also be provided.

4 A note on collaboration

I encourage you to discuss the assignment with other members of the class. This may be especially useful in examining the code that has been given out as your starting point, but also for discussing both the *Basic Requirements* and the *Additional Requirements*. However, the code you develop for your solution should be your own, individual work, and your report should be written by you alone.

5 Getting started

On studres, you will find a directory called `cs2003-web2` in `studres/CS2003/Practicals/Web2/`. This contains code that you should modify and extend directly for your solution. The files you will find are:

- *dir_list.css*: a basic CSS file.
- *dir_list_json.js*: a JavaScript file that is used for server-side generation of content, including getting file information, as well as encoding / decoding JSON objects for network communication. Some functions may also be useful at the client-side.
- *dir_list_example.html*: a HTML file that is generated from *dir_list_example.js* with node.js. It demonstrates the use of the server-side JavaScript code in *dir_list_json.js*.
- *dir_list_example.js*: a server-side JavaScript file that demonstrates the use of *dir_list_json.js*.
- *dir_list_json.js*: a file to use with node.js to test the collection and processing of file information from a directory.

You should copy the `cs2003-web2` directory from studres into your own filespace (it does not have to be your web filespace). Once you have copied the files, access them through the Firefox web-browser.

The Firefox web-browser is likley to be much more stable with WebSockets than Chrome is. I recommend you use Firefox onteh Lab machines, and the markers will also use Firefox.

You should be able to complete the *Basic Requirements* by modifying and/or extending these files and some of the example files from `CS2003/Examples/wk09/`, but you may need to review material from earlier weeks also.

In order to fulfil some of the *Additional Requirements*, you may need to create additional JavaScript files (and perhaps HTML and CSS files also), as well as modify and extend the files listed above, and the protocol specification.

Marking

Having completed the *Basic Requirements*, to gain a mark above 16, you will need to complete some of the *Additional Requirements* listed above. In your report, highlight where you have attempted any of the *Additional Requirements*. However, please first build a working solution meeting all the *Basic Requirements*, before trying anything more complex.

Note that attempting one or more of the items listed as Additional Requirements does not guarantee you a mark above 16. Your mark will depend on the overall quality of your submission, both the code and the report.

The submission will be marked on the University's common reporting scale according to the mark descriptors in the Student Handbook at:

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

Please ensure you are following the relevant guidelines on Good Academic Practice as outlined at:
https://www.st-andrews.ac.uk/students/rules/academicpractice/

6 Protocol specification

The protocol consists of text messages using JSON data structures that have been encoded to text strings using `JSON.stringify()`, and then decoded back to JSON objects using `JSON.parse()`.

6.1 Message structure

The *message structure* is:

```
{ <type> : <subtype>, <name> : <value> }
```

The `<type>` and `<subtype>` are a name-value pair, as are `<name>` and `<value>`. `<type>`, `<subtype>` and `<name>` are simple strings, but `<value>` could be complex JSON object that has been encoded for transmission.

6.2 The "request" message format

The `"request"` message type currently has one sub-type only, `"dirinfo"`, which, in turn, only has one name-value pair, with name `"dirpath"`. This is used by the client to ask for a directory listing. An example message might look like this:

```
{"request":"dirinfo","dirpath":"/"}
```

This is a request from the client to the server for the directory information for the path `"/"`. Note that when the server receives this, the 'root' directory is the logical root as provided by the server, and *not* the root of the actual file-system on the machine. This is used by the client because it may not know the real name of the root directory for the server.

6.3 The "response" message format.

The `"response"` message type currently has one sub-type only, `"dirinfo"`, which, in turn, only has one name-value pair associated with it, with name `"info"`. This is used by the server to respond to a request for a directory listing. An example message exchange might be as follows.
The client might send:

```
{"request":"dirinfo","dirpath":"test1Dir/dirA"}
```

The server responds with:

```
{
    "response":"dirinfo",
    "info":
    {
        "server":"pc2-105-l.cs.st-andrews.ac.uk",
        "directoryname":"test1Dir/dirA",
        "files":
        {
            "A.txt":
            {
                "filename":"A.txt",
                "type":"file",
                "size":2,
                "atime":1542323317000,
                "mtime":1541644351000,
                "ctime":1542323317000,
                "birthtime":1542323317000
            },
            "dirA1":
            {
                "filename":"dirA1",
                "type":"directory",
                "size":3,
                "atime":1542323324000,
                "mtime":1541644550000,
                "ctime":1542323317000,
                "birthtime":1542323317000
            }
        }
    }
}
```

Note that in this case, I have added whitespace, including line breaks, so that you can more easily see the structure of the data: in reality, this would be a single stream with no extraneous whitespace.

This shows the directory listing for directory `test1Dir/dirA` on server `pc2-105-l.cs.st-andrews.ac.uk`. The directory has two files, one called `A.txt`, which is a standard file, and one called, `dirA1`, whcih is a directory. The `size` is in bytes. The various times shown (access time, modification time, creation time, and birth time) are given in milliseconds since 10 January 1970. However, you will find code in file `dir_list_json.js` that converts these time values to a more readable form.

6.4 Observations

If you run the script `dir_list_json_test.js`, and examine the code, you can see how these data structures are generated.

Note that the protocol is stateless: a single request has a single response.

The protocol can easily be extended, and the code in `dir_list_json.js` can be re-used.