CS2003 Practical Web2: Directory Lister

Deadline: 25 November 2020
Credits: 30% of coursework mark
MMS is the definitive source for deadline and credit details.

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. Please read this entire document carefully before you start as there are important specification details all the way through it.
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.
- Use of AJAX for client-server communication.

1 Requirements

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

Making use of the hints given in Section 5, you are required to develop a web application to perform the following tasks:

(1) 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.

- Your server-side program should be stored in a file called `dir_list_server.js` and it should take a single command line argument (the root directory).
- We provide a directory called `test1Dir`, you can use this directory as the root during your testing.

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

(3) 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.

(4) 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.

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

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

(7) 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.

(8) Allow a search for files at the server based on a sub-string search.

For client-server communication you should use the protocol specified in Section 6, with minor modifications to allow a new 'search' sub-type for 'request' and 'response'.

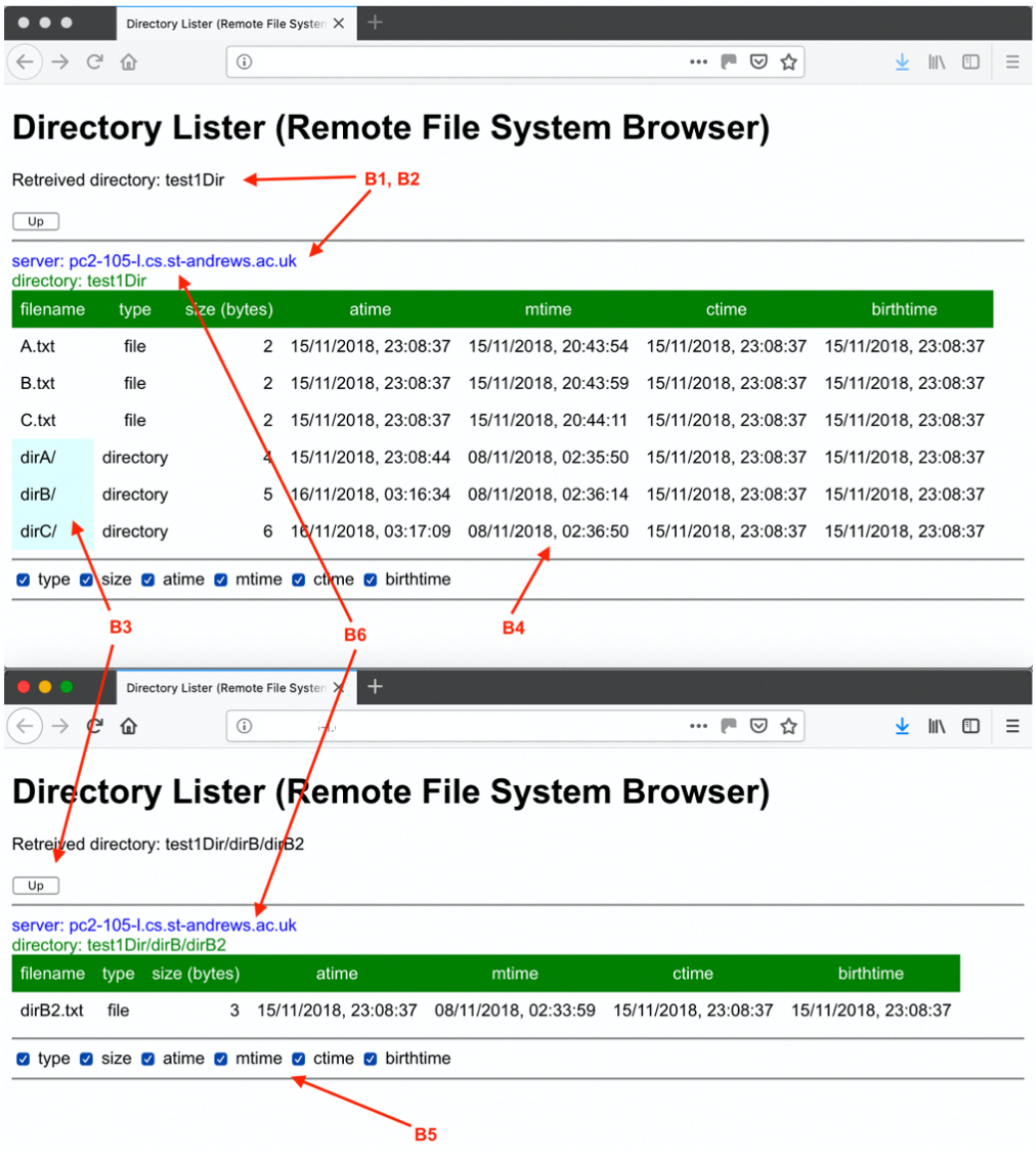

Figure 1: Two Firefox browsers simultaneously connected to the same server.

2 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 requirements 1 to 6 are visible (labelled as B1-6 in diagram).

3 Submission

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

- **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 another user's School host server, run the server-side component and access the web application through a web browser (assuming correct nginx proxy setup).
- Your marker will run the following command to test your application:
  `node dir_list_server.js test1Dir`
- To enable this, make sure you use the unix uid of the active user as the port number and set an appropriate nginx proxy to access your application. You can use process.getuid() in node.js to find the active user's uid. The marker will set up a similar proxy to test your application. Your application should work without any modifications, otherwise you will be penalised in marking.

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 except Express. 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.

a) **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 requirements listed above.
   - Suitable, simple, diagrams, as required, showing the operation of your program.
   - An indication of how you tested your application, including screenshots of the web pages showing your application working.

4. A note on collaboration

I encourage you to discuss the assignment with other members of the class. However, the code you develop for your solution should be your own, individual work, and your report should be written by you alone.

5. Hints

- As a first step create a basic node application and set up the nginx proxy.
- If you decide to use Express, install it by running `npm install express` (both on your development machine and on the host server).
- Write HTML, JS, and CSS files as required for the client-side of your application. These should be stored in static files and served by the node.js application.
- Use the `__dirname` variable in node.js when constructing paths. This variable points to the directory of your server script.
- Create a route called '/api'. This route should accept post requests whose body is JSON and produce response encoded in JSON. The protocol for the input and output of this protocol is given in Section 6.
- Take the root directory as a command line argument to your node.js program. You shouldn't need any other command line arguments.
- Use console.log statements generously in both the server and the client code to inspect the operation of your application, especially during development.
- A JavaScript function called getFileInfo is given to you on StudRes, in a file called lib.js. Feel free to use this when preparing the data for the directory listing.
- When making an AJAX request (using XMLHttpRequest or fetch), do not forget to set the method (post) and the appropriate headers to set the content type to JSON.
- Make use of earlier example files on JSON, AJAX and node.js. You may also need to review material from earlier weeks. Clearly denote the source of any code snippets you get from the examples.
- I recommend you use Firefox, the markers will also use Firefox.

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":"dirA"}
```

The server responds with:

```
{
 "response": "dirinfo",
 "info": {
    "server": "lyrane",
    "directoryname": "dirA",
    "files": {
       "A.txt": {
          "type": "file",
          "size": 2,
          "atime": "11/10/2020, 3:21:23 PM",
          "mtime": "11/11/2019, 5:49:19 PM",
          "ctime": "11/10/2020, 3:21:23 PM",
          "birthtime": "1/1/1970, 1:00:00 AM",
          "filename": "A.txt"
       },
       "dirA1": {
          "type": "directory",
          "size": 3,
```

```
            "atime": "11/10/2020, 3:21:23 PM",
            "mtime": "11/11/2019, 5:49:19 PM",
            "ctime": "11/10/2020, 3:21:23 PM",
            "birthtime": "1/1/1970, 1:00:00 AM",
            "filename": "dirA1"
        }
    }
  }
}
```


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 listing for directory test1Dir/dirA on server lyrane (one of the host servers). You should log in to your own host server (ssh <username>.host.cs.st-andrews.ac.uk). The directory has two files, one called A.txt, which is a standard file, and one called, dirA1, which is a directory. The size is in bytes. The various times shown (access time, modification time, creation time, and birth time) are displayed by calling toLocaleString("en-GB").

6.4 Observations

Note that the protocol is stateless: a single request has a single response.
The protocol can easily be extended by adding more request/response subtypes. I do not prescribe what the protocol should look like for the search functionality, you should extend the protocol appropriately and explain your design in the report.

Marking

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/