Client-Server File Browser
v0.5 (some clarifying details to be added in a few days)
Cpt S 489

Read all instructions carefully before writing any code. Do all work individually!

In this assignment, you will create both client-side and server-side JavaScript applications. The server-side content will use Node.js. If you have not done so already, download and install Node.js as the first step.

Server

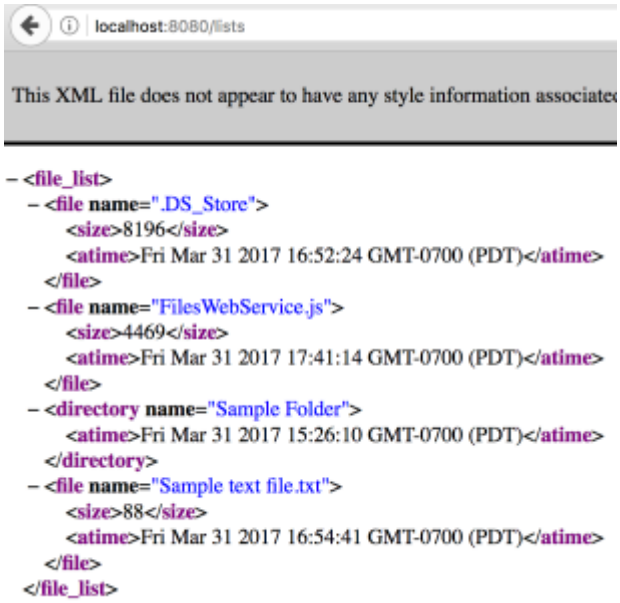
The server-side Node.js application will host a directory of shared files from the server. This is achieved through 2 simple services:

1. Providing descriptive listings of files to the client. These lists will either be XML or JSON strings.
2. Provide the ability to download content of shared files from the server.

As was discussed in class, there are a few options for how the server can provide these services. It could be the case that the URL starts with a string that distinguishes the request as either wanting file listings/information vs. file content. For example, all list-oriented URLs might start with "/lists" and all file-download URLs might start with "/files". A second option is that the server could determine, based on the URL, whether it's a file or directory and then send either a list or file contents.

Example of second option: Assume server is sharing a directory that has subdirectory "dir1" and file "abc.txt" contained within. The URL "/dir1" is known to refer to a directory, so the server should send back a file listing. The URL "/abc.txt" is known to refer to a file, so the server should send back the file contents.

Choose one of the two schemes and include information in a comment at the top of the server JavaScript code about which one you used. List a URL that can be navigated to when the server-side script is running that will show either the XML or JSON (whichever you chose to implement) in the browser window. This will not ultimately be how the service is used, since there will be client-side code that gets the data from this service, but it must be the case that you can see the serialized data from the service with a simple request in the browser. Below is a sample image of what a simple XML file listing might look like when browsing to localhost:8080/lists.



Client:

Make an HTML file with script that uses XMLHttpRequest to get file lists from the server and dynamically generates content on the page to show this list to the user. The user never navigates away from this page. The ability to browse through all the shared content, including subdirectories, happens without ever reloading the page. The JavaScript on the page will do requests as needed to get lists of files and directories and display them to the user.

The look and feel of the client-side file browser should be similar to the basic file explorer in Windows or finder in MacOS. Provide a view with columns for name and size (in bytes) at a minimum. Additional details can be shown if you like. Make sure directories are visually distinguished from files. Highlight them in a different color, bold their text, or provide some other visual cue.

Each file must be a link and when you click on it you open/download the file. Each directory must also be a link and when you click on it you see files in that directory. They can either expand out beneath the directory in the page or the entire list can be refreshed.

Implementation Notes:

To allow XMLHttpRequest to get data from your Node.js server, you may need to add the header to the response:

"Access-Control-Allow-Origin": "*"

So when starting a response with response.writeHead, it may look something like:

response.writeHead(200, { "Content-Type": contentType, "Access-Control-Allow-Origin": "*" });

In that example, the variable contentType is expected to be defined and set to a content type appropriate for the type of file being sent.