Files Web Service Uploads
Cpt S 422 Homework Assignment
by Evan Olds

Assignment Instructions:

Read all instructions *carefully* before you write any code.

In this assignment, you will extend the files web service that you created in an earlier assignment by adding support for uploading files. This will be done by implementing the "PUT" method. You will need some JavaScript code to put on the page to allow uploads, but this will be given to you.

Add Upload HTML and JavaScript to the File List Pages

From the previous assignment where you implemented the files web service, you will have a function to build the HTML for a page that lists files and directories. This function should be called "BuildDirHTML" or something similar. You likely started this function by initializing a StringBuilder and then appending HTML strings appropriately for links. Use the code below to add in the necessary JavaScript at the beginning of your file list pages. Also, as you'll see from this code, there is an "m_allowUploads" member variable. Add this bool member variable to your FilesWebService class and initialize it to true by default.

```
// Build an HTML file listing
var sb = new StringBuilder("<html>");

// We'll need a bit of script if uploading is allowed
if (m_allowUploads)
{
    sb.AppendLine(
@"<script>
function selectedFileChanged(fileInput, urlPrefix)
{
    document.getElementById('uploadHdr').innerText = 'Uploading ' + fileInput.files[0].name + '...';

    // Need XMLHttpRequest to do the upload
    if (!window.XMLHttpRequest)
    {
        alert('Your browser does not support XMLHttpRequest. Please update your browser.');
        return;
    }

    // Hide the file selection controls while we upload
    var uploadControl = document.getElementById('uploader');
    if (uploadControl)
    {
        uploadControl.style.visibility = 'hidden';
    }

    // Build a URL for the request
    if (urlPrefix.lastIndexOf('/') != urlPrefix.length - 1)
    {
        urlPrefix += '/';
    }
```

```
    var uploadURL = urlPrefix + fileInput.files[0].name;

    // Create the service request object
    var req = new XMLHttpRequest();
    req.open('PUT', uploadURL);
    req.onreadystatechange = function()
    {
        document.getElementById('uploadHdr').innerText = 'Upload (request status == ' + req.status + ')';
        // Un-comment the line below and comment-out the line above if you want the page to
        // refresh after the upload
        //location.reload();
    };
    req.send(fileInput.files[0]);
}
</script>
");
}
```

After that, you will put the code from the earlier assignment that produces links for files and directories. At the end of the HTML, before adding the closing "</html>", add the HTML for the file uploader. This is given for you below. The GetHREF function call refers to a utility function that gets the hyperlink reference for the directory that you're producing the listing for. You may have had something like this from the earlier assignment. If not, you'll need to add it. It produces the HREF of the form: "/files/dir1/dirUserIsInNow". This is then inserted into the JavaScript so that it knows how to build the proper upload URI.

```
// If uploading is allowed, put the uploader at the bottom
if (m_allowUploads)
{
    sb.AppendFormat(
        "<hr><h3 id='uploadHdr'>Upload</h3><br>" +
        "<input id=\"uploader\" type='file' " +
        "onchange='selectedFileChanged(this,\"{0}\")' /><hr>",
        GetHREF(directory, true));
}
sb.Append("</html>");
```

Implement the PUT Method Handler for Uploads

In your FilesWebService code, implement a handler that first checks the method to see if it is "PUT". If your core server code denied non-GET requests, then you'll need to extend it to allow "PUT" as well. If you see this method when handling a request, it will be a file upload as opposed to download.

The JavaScript code you have been given will generate a PUT request where the entire body is the file data. So, once you've got the code in place to create an open the proper file for a PUT request, you will then just read all the data from the body of the request and write it to the file. Write a regular "200 OK" response if the upload completes successfully. This result goes to the JavaScript and you will not see it in the browser (although the simple JavaScript you've been given does display the status code from the upload). Make sure the additional requirements are also met:

- Don't allow overwriting existing files
- Allow all (non-existing-file-overwriting) uploads provided the file name doesn't contain reserved characters (/ and \)
- Files can be uploaded to any directory that the service shares
- Files must be uploaded into the current directory that the user is seeing the listing for within the browser (in other words you can't just have every upload just always put the file in the root directory)
- There is no limit on file sizes or content
- You only need to support uploading 1 file at a time (1 per page refresh, as described below)
- You do NOT need to support cross-origin requests or the OPTIONS method, which will have been discussed in class

The JavaScript code that you've been given is minimal and primitive. After one file upload, you'll have to refresh the page if you wish to do another, since it hides the file chooser on the page after the first upload.

As usual, you need to come up with your own test cases. If you write code for your tests you do NOT need to include it with your submission. Your tests are a means of ensuring proper functionality of your code. Be sure to test files of various sizes and include relevant edge cases like files of size 0, and files of size > int.MaxValue. Beyond that, do whatever testing you feel is necessary to ensure that you've met the assignment requirements.

Final Remarks:

This assignment relies heavily on previous assignments. If your code for the files web service had problems, now is the time to fix them. You will not be given solutions for previous assignments just because you didn't do well on them. So please do not ask for this. Also, remember to do your own work. As of the time of this writing (Fall 2016), this has been the first class ever where I've failed multiple people already and reported them to the office of student conduct for cheating on prior assignments. Make sure you do your own work so that you do not end up like these students.

You need to demonstrate that you can implement this project up to this point to be considered to have the required skillset to pass the course. For those that have done well up to this point, this assignment will not be very difficult. For those that have NOT done well, you need to get your code from previous assignments fixed so that you can implement the upload feature. If code from your previous submissions was so broken that you cannot build on top of it or fix it in order to get this assignment done, then you will not have demonstrated the necessary skills for the course. If you need to seek help in order to build the necessary skillset, then make sure you do so during this week!