## How load information from a .CSV file

This time I will be describing how to load information from a CSV file. This is very helpful when we need to enter large amount of data, and we don’t want make our users to type all that information, is much easier provide the functionality to load that information from the file where it was originally stored, in this case we will be loading it from a .CSV file.

For accomplish this we need the HTML code in our .JSP, the JavaScript code to make validations in the client, make the request and then handle the response, and finally the controller that receive the Multipart File, extract the data and send it back to the view.

**HTML Code:**

The HTML code I used was just this:

<**div>**  
 <**div**>  
 <**form** enctype=**"multipart/form-data"** method=**"post"** name=**"uploadForm"** id=**"uploadForm"** action=**"callblocking/uploadcsv"**>  
 <**fieldset**>  
 <**legend**>Upload phone numbers</**legend**>  
 <**input** type=**"file"** name=**"csvFile"** id=**"csvFile"**>  
 </**fieldset**>  
 </**form**>  
 </**div**>  
 <**footer**>  
 <**button** id=**"cancelUploadPhonesBtn"**>Cancel</**button**>  
 <**button** id=**"uploadPhonesBtn"**>Update</**button**>  
 </**footer**>  
</**div**>

The key components here are the form with the enctype=**"multipart/form-data"** and the input file field, they will allow us to look for the file in our local file system and send it to the server, but in our case we will customize that “sending” process a little bit in the Java Script, as we will see in the next section.

Also we have the button **uploadPhonesBtn** which will trigger the action once we want to upload the file previously selected.

**Java Script Code:**

The Java Script code is structured as classes, to make the code more organized and following the Object Oriented principle:

Here we have the Java Script class ImportPhoneNumbersManager, which has as only “public” function *import*, which will be called when click the upload button, this function validate the file and send it to the server using the “internal” functions validateFile and uploadFile respectively.

Here we also have kind of internal class FileUploadHandlers, where we will group the different handlers to be used in the uploadFile function.

**var** ImportPhoneNumbersManager = (**function**() {

**return** {  
 **import**: **function**() {  
 **var** inputFile = document.getElementById(**'csvFile'**);

**if** (!validateFile(inputFile)) **return false**;  
   
 **var** form = document.forms.namedItem(**"uploadForm"**);  
uploadFile(inputFile, form);   
 }  
 }

**function** uploadFile(inputFile, form) {  
 console.log(**'FormData obj present'**);  
 **var** fd = **new** FormData();  
 fd.append(**'csvFile'**, inputFile.files[0]);  
  
 $.ajax({  
  *url: form.action,*   
 type: **"POST"**,  
 data: fd,  
 processData: **false**, *// tell jQuery not to process the data* contentType: **false**, *// tell jQuery not to set contentType* dataType: **"json"**,  
 beforeSend: FileUploadHandlers.beforeSend,  
 complete: FileUploadHandlers.complete,  
 success: FileUploadHandlers.success,  
 fail: FileUploadHandlers.error  
 });  
 }

**function** validateFile(inputFile) {  
 console.log(**'Validating html 5 file upload.'**);  
 **if** (inputFile.files.length <= 0) {  
 alert(**'No file selected.'**);  
 **return false**;  
 }  
 **var** file = inputFile.files[0];  
 **var** filter = /^(text\/csv|application\/vnd.ms-excel)$/i;  
 **var** fileSize = file.size/1024;  
 **var** maxSize = 32; *//kb* **if** (!filter.test(file.type)) {  
 alert(**'Selected file is not csv'**);  
 **return false**;  
 }  
 **else if** (fileSize > maxSize) {  
 alert(**'Selected file size should be no greater than '** + maxSize + **' KB.'**);  
 **return false**;  
 }  
  
 **return true**;  
 }

**var** FileUploadHandlers = (**function**() {  
  
 **return** {  
 beforeSend: **function**() {  
 console.log( **"uploading..."** );  
 },  
 complete: **function**() {  
 console.log( **"Finished loading data"** );  
 },  
 success: **function** (result, status) {  
 **var** output = **''**;  
 jQuery.each(result, **function**(index, value) {  
 **if**(**this**.trim() != **''**){  
 output += **this** + **'\n'**;  
 }  
 });  
 $(**'#phoneNumberTextArea'**).val(output);  
  
 $(**'#cancelUploadPhonesBtn'**).click();  
 },  
 error: **function** (data, status, e) {  
 console.log( **"Request failed: "** + e );  
 }  
 }  
 })();  
  
   
  
   
  
  
  
})();

Then we are ready to attach the button to its action:

$(**'#uploadPhonesBtn'**).click( **function**(e) {  
 e.preventDefault();  
 ImportPhoneNumbersManager.import();  
});

Notice that we call first the event preventDefault to stop the default behavior of the button inside the form and then we call the function import, inside theImportPhoneNumbersManager class**.**

**Server Side**

And here is our server side code, which is a method in a controller, which will receive a multipart file via POST, then call an auxiliary method to extract the information and return the data to the view.

@RequestMapping(value = **"/addPhoneNumbers/uploadcsv"**, method = RequestMethod.***POST***)  
@ResponseBody List importFile(@RequestParam(**"csvFile"**) MultipartFile file) {  
 **try** {  
 **return** *extractCsv*(transferToTmpFile(file))  
 } **catch** (IOException e) {  
 **return** [errors:**"**$e.message**"**] //This is Groovy code  
 }  
}

**private** File transferToTmpFile(MultipartFile file) {  
 File csv = File.*createTempFile*(**"phoneNum"**, **"csv"**)  
 file.transferTo(csv)  
 **return** csv  
}

**private static** List extractCsv(File file) {  
 String line = **""**;

List<String> list = **new** ArrayList<String>();

**try(**BufferedReader br = **new** FileReader(file)**)**{

**while** ((line = br.readLine()) != **null**) {  
 list.addAll(line.split(“,”))  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return** list;  
}

To read the .CSV file we take every line, and split the content by comma, to get the information of every column. Then store the elements in a list and return it.

**Conclusions**

And that’s it, this is all we need to load content from a .CSV file, with some customization in the client side for validations and process the data coming from the server.

I hope this help you, feel free to ask any question, correction or suggestion, other readers and me will appreciate it.

See you in the next hacking…