API and AJAX

The JavaScript code for the examples below is in GitHub: https://github.com/kurmasz/CIS658_JavaScript

API

A Rails app can return data in formats other than HTML.

1. In your blog app, open `authors_controller.rb` and re-write the `index` method as follows:

   ```
   def index
      @authors = Author.all
      respond_to do |format|
        format.html {render :index}
        format.json {render :index, status: :ok}
        format.xml {render xml: @authors.as_json}
     end
   end
   ```

2. Visit this route and add `.xml` to the end of the URL (e.g., `http://myBlog.com/authors.xml`).

   Notice that the server returns the list of authors as an XML document. The XML tags are a bit awkward (e.g., "objects" and "object"); but, you could fix this by overriding the `to_xml` method for Authors.

3. Visit this route and add `.json` to the end of the URL (e.g., `http://myBlog.com/authors.json`)

   This time the server returns the list of authors as a JSON object. JSON serves a similar purpose to XML; but, is simplier. (See https://www.w3schools.com/js/js_json_xml.asp for a comparison.)

Ajax

Web clients can use JavaSript to access this data and update the page without having to do a refresh.

Note: By default web browsers do not allow cross-origin Javascript: AJAX calls must be made to the same server the web page came from. Thus to run this example you must either:

- Put the sample code on the same server as your Rails app
- Use Safari and go do "Developer => Disable Cross-Origin Restrictions
- Launch Chrome with cross-origin disabled: https://stackoverflow.com/questions/3102819/disable-same-

origin-policy-in-chrome (Be sure to exit *all* chrome processes first.)

1. Load `loadAuthors.html` and press the button.

   This `.html` file is simply an empty list followed by a button. When the button is pressed, the JavaScript makes an AJAX call to the API and retreives the list of authors as XML.

2. Open `loadAuthorsXML.js` and notice the `(function() }` construct at the beginning.

   It is bad form to pollute the global name space. In order to keep our methods out of the global name space, we place them inside an anonymous function. The `})()` construct at the end of the file calls this anonymous function as soon as it is defined

3. Examine the code from the bottom up (beginning with the `document.addEventListener` block).

   This block adds a listener to the DOM that is called whenever the state of the DOM changes. The document has three states: `loading`, `interactive` and `complete`. When the document indicates it is in the `interactive` state, we know the DOM is complete and it is safe to search it for elements. (Calls to `getElement...` methods executed while the document is still in the `loading` state may miss elements that have not yet been constructed and added to the DOM. If you ever have problems with DOM elements not being found, double-check that you aren't searching the document before the DOM is complete.)

   Documents in the `interactive` state, but not yet complete have a complete DOM; but, may still be loading / processing resources such as stylesheets and images. See https://developer.mozilla.org/en-US/docs/Web/API/Document/readyState for details.

4. Examine the `loadAuthors` method.

   This method makes a "raw" AJAX call. Most developers use jQuery or some other library that wraps this raw call. These libraries typically (a) automatically check `readystate` and (b) provide better scaffolding for adding error handlers.

   Notice that the `send` method is asynchronous: It returns immediately. When the response comes back from the server, the `onreadystatechange` method is called. (Yes, there are ways to make synchronous AJAX calls; but, it is not good practice.) See https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState for a description of the various ready states.

5. Intentionally misspell "authors" and notice the console messages.

   Misspelling "authors" generates a 404 result. Notice that `onerror` is *not* called in this case (because the server returned a usable result -- just not the result we hoped for).

6. Intentionally misspell the domain name and notice this console messages.

   In this case, `onerror` is called. Notice that `onreadystatechange` is also called. (This is why it is important to check the returned status!)

7. Examine the data returned by the AJAX call. Specifically, look at the console output generated by these two lines:

   ```
   console.log("Complete AJAX object:");
   console.log(ajax);
   ```

   AJAX stands for "Asynchronous Javascript and XML". It is designed with XML in mind and, therefore, provides some shortcuts when using XML. Look specifically at `ajax.responseXML` in the Developer Tools Console. Notice that it looks similar to the web page's main `document`. JavaScript automatically parses the returned XML and builds a DOM for it.

8. Examine `xmlToAuthors`

   This method uses the standard DOM methods to extract data from the XML and build an array containg data for each author.

9. Examine `insertAuthors`

   This method creates an `<li>` element with the author information and inserts it into the DOM. Note the use of `createElement` and `createTextNode`.

XSL / XSLT

XSL stands for E**X**tensible **S**tylesheet **L**anguage. Think of it as "CSS for XML).

- XSLT stands for XSL Transformations
- Uses XSL to transform one XML document into another
- https://www.w3schools.com/xml/xsl_languages.asp

This sample XSLT document transforms the XML produced by Rails into XML that looks a lot like HTML:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <ul id='rootOfList'>
    <xsl:for-each select="objects/object">
      <li>
        <xsl:value-of select="lname"/>,
        <xsl:value-of select="fname"/>
        Email:  <xsl:value-of select="email"/>
      </li>
    </xsl:foreach>
  </ul>
</xsl:template>
```

1. Place this document in your blog app's `assets` folder. Call it `author.xsl`
2. Rewrite `AuthorsController#index`

   ```
   def index
     @authors = Author.all

     # Apply XSLT
     document = Nokogiri::XML(@authors.as_json.to_xml)
     template = Nokogiri::XSLT(File.read('app/assets/author.xsl'));
     transformed_document = template.transform(document)

     # dumps the transformed document to the console so you can see the effects
     puts "---\nBefore"
     puts document
     puts "---\nAfter"
     puts transformed_document

     respond_to do |format|
       format.html {render :index}
       format.json {render :index, status: :ok}
       # format.xml {render xml: @authors.as_json}
       format.xml {render xml: transformed_document}
    end
   end
   ```

3. Visit the `authors.xml` page and examine the results of the transformation: The resulting XML is also valid HTML.
4. Modify `loadAuthors.html` to use `loadAuthorsXLS.js`

5. Examine `loadAuthorsXLS.js`

   This file is nearly identical to `loadAuthorsXML.js`. The main differences are that (a) we no longer need `xmlToAuthors` (the XSLT already performed the transformation), and (b) `insertAuthors` is much simpler: We need only insert the HTML into the DOM. (This mention could reasonably be written using one long line of code.)

JSON

1. Modify `loadAuthors.html` to use `loadAuthorsJSON.js`
2. Examine `loadAuthorsJSON.js`

   This file is nearly identical to `loadAuthorsXML.js`. The main difference is that we no longer need `xmlToAuthors` (the `JSON.parse` converts the AJAX response to an array of JavaScript objects).