CSCE 120: Learning To Code

Consuming Data I
Hacktivity 11.1

Introduction

Prior to engaging in this hacktivity, you should have completed all of the pre-class activities as outlined in this module. At the start of class, you will be randomly assigned a partner to work with on the entirety of this hacktivity as a *peer programming* activity. Your instructor will inform you of your partner for this class.

One of you will be the driver and the other will take on the navigator role. Recall that a driver is in charge of the keyboard and computer while the navigator is in charge of the handout and directing the activity. However, you are *both* responsible for contributing and discussing solutions. If you were a driver/navigator in the prior activity, switch roles for this activity.

1 Knowledge Check

With your partner, discuss and answer each of the following questions. Write your answers down on a separate sheet of paper or type them up in a plain text file.

1. Briefly explain what HTTP is and the client-server model
2. Briefly explain what Ajax is and how it can be used in a web app
3. Briefly explain what CORS is
4. What jQuery function facilitates an Ajax operation and what are its essential parameters?

2 Warm-up Exercises

Download the code we've provided from GitHub using the URL, https://github.com/cbourke/BulletinApp. Open the project in Light Table. Open the HTML and JavaScript files in the `exercises` folder. Evaluate the HTML page so that jQuery is loaded (do not connect to the Light Table UI).

2.1 Using Ajax

First, let's get some practice using ajax. The HTML file is provided simply to load the necessary jQuery files. In the JavaScript file, formulate and execute an `$.ajax()` function to connect to the following webservices. For these exercises, simply log the data to the console. Determine if the webservice supports CORS and/or json-p or neither. You can refer to the jQuery documentation on `$.ajax()` (http://api.jquery.com/jquery.ajax/) for how to use json-p.

1. http://cse.unl.edu/~cbourke/CSCE120/proxies/calc.php
   This simple webservice takes two parameters: `x` and `y` and returns basic calculations between the two. Does it support CORS? Does it support json-p?
2. https://www.reddit.com/r/woodworking/new.json
   This is a woodworking "subreddit" on the popular website, Reddit.
3. https://api.imgur.com/3/gallery.json
   This is an API for the image sharing site Imgur. Does it support CORS? Json-p? In either case, does it return data? Read the documentation (https://api.imgur.com/) and determine why or why not.

   As an alternative, try https://api.imgur.com/3/gallery/hot/viral/0.json instead.
4. Point your browser to http://apis.io/ which is an API search engine. Search for an API that may be of interest to you and attempt to connect to it. Does it support CORS? Json-P? Does it require an API key?

2.2 Importance of Asynchronicity

We have setup a webservice at the following URL: http://cse.unl.edu/~cbourke/CSCE120/proxies/delay.php, it doesn't do anything but *simulate* a long-running process by delaying its response by a certain number of seconds that you can specify by providing a `delay` parameter (numeric value indicating the number of seconds to delay the response, the default is 5 seconds).

1. Modify the `sayHello()` function (which is invoked when a user clicks the "Say Hello!" button) so that it makes a connection to this service.

2. Modify your code to give some visual feedback to the user by displaying a "loading" message or graphic (we have provided an animated loading GIF in the project under `../common/images/loading.gif`). Upon success, change this so that the loading graphic goes away and a success message is displayed to the user.
3. Once working, try different values for the delay parameter. While waiting for a response, interact with the "Action" elements (but do not interact with the "Delay" button). Are you able to interact with these things in the page? Why?
4. Now observe the consequences of a long-running *synchronous* action. Click the delay button, which calls a function to simply add 1 to a number over and over many times. Depending on your speed, you may need to increase/decrease the value of the variable `n` in the code to get a more manageable but observable delay. Can you interact with the other elements in the page while this executes? Discuss possible reasons for this difference with your partner.

3 Loading Data Asynchronously

We have provided a web page that loads now familiar enrollment data (see the `enrollment` folder) into a table. That is, it will once you've completed it.

1. Complete the `loadData()` function by making an ajax query to load the data in the `student.json` file.
2. Since you are loading a JSON data file, be sure to set the `dataType` property of your ajax call to `"json"` [1]
3. Load the data by processing each enrollment record and adding it as a row to the (initially empty) table
4. Add some flair:
   a) Display a loading graphic while the data is being loaded, making sure to remove the graphic once the data is successfully loaded
   b) Fade in the table data to give a visual cue to the user that the data was successfully loaded
5. A user has to click the "(Re)Load" button for the data to be loaded. Add some code to make the data load when the page is initially loaded.

[1] This is only necessary because you are loading a local file. In a true client/server situation as in the previous examples, the server response includes *headers* that indicate the type and formatting of the data it responds with. The `$.ajax()` function is able to use this to automatically determine the `dataType`.