Using Ajax with jQuery

Overview

In this lab, you'll enhance a Web page so that it makes Ajax calls to a Web server. It doesn't really matter what the Web server technology is, so we've implemented a node web application. Other good candidates include ASP.NET MVC, PHP, Java servlets, etc.

To simplify Ajax interactions, you'll use jQuery to make the Ajax call. This is a lot cleaner (and more portable) than using the `XMLHttpRequest` object directly.

Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Understanding the project structure
2. Making an Ajax call
3. Processing the Ajax response
4. Additional suggestions

Familiarization

Start your editor and open the *Solution* project for this lab, to get an idea of what you're aiming for in this lab. Here are the details of the project:

- *Folder:*
  `\Solution\player-management`

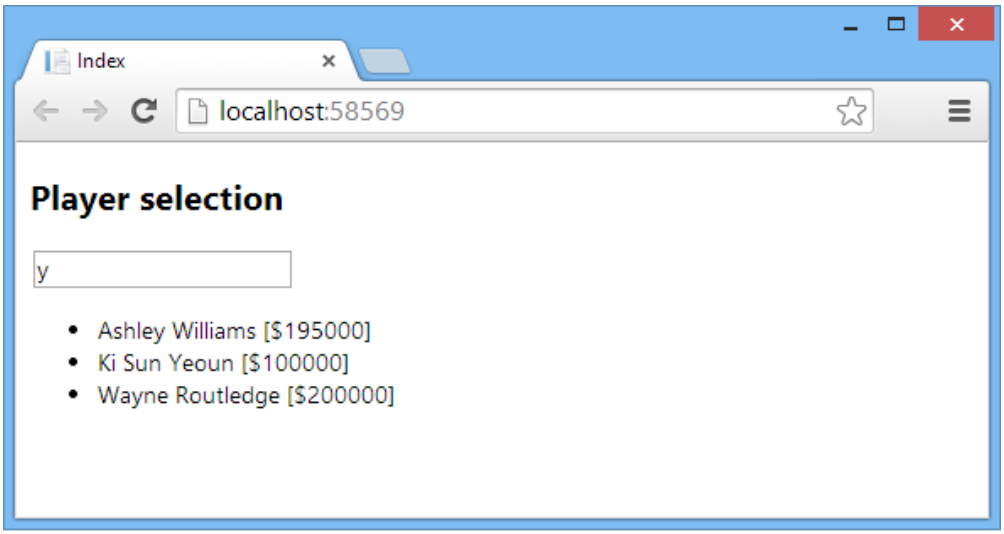
Build and run the application:

- Open command line in folder `\Solution\player-management`
- To install the dependencies, type: "npm install"
- To run the project, type: "npm start"
- Open the browser at the location that the server wrote on the command line (http://localhost:3000)

The home page appears, displaying an empty text box. The purpose of the Web page is to detect keystrokes one-by-one, and to send a request to the Web server to get a list of people whose names match what you've typed in so far.

Try it out:

- In the text box, type the letter `y`.
- The Web page detects this keystroke and sends the text "`y`" to the Web server.
- The Web server returns a list of players that contain a "`y`" in their name (each player has a first name, last name, and salary).
- The Web page receives the result and displays the matches in a bulleted list as follows:



- Type another letter in the text box, such as "`e`". This causes the Web page to send another Ajax request, this time containing the text "`ye`" (i.e. it sends the entire text, not just the

most recent keystroke). The Web server returns a list of players containing "`ye`" in their names. And so on ☺.

When you're happy with how all this works, close the browser window, return to your editor, and close the *Solution* project.

Exercise 1: Understanding the project structure

In your editor, open the *Start* project for this lab as follows:

- *Folder:*
  `\Start\player-management`

Build the application:

- Open command line in folder `\start\player-management`
- To install the dependencies, type: “npm install”

If you want to run the project later in the other exercises:

- To run the project, type: “npm start”
- Open the browser at the location that the server wrote on the command line (http://localhost:3000)

If you want to stop running the project later in the other exercises:

- Press ctrl+c and answer with Y(es).

Exercise 2: Making an Ajax call

In `.\public\index.html`, note the following points:

- The page has a text box where the user can enter a player's name. Each keystroke will trigger the `getMatchingPlayers()` JavaScript function, which is empty at the moment. You'll implement this function in this lab, to make an Ajax call to the Web server.
- The page also has a `<ul>` control named `ResultList`, ready to display a bulleted list of players info returned from the Web server.

As mentioned earlier, you're going to use jQuery to simplify Ajax interactions. Observe the folder already contains `jquery-1.10.2.js`. Add a script tag with the src property pointing to this jQuery file into your `index.html` page (in the `<head>` section).

Now add code in the `getMatchingPlayers()` JavaScript function, to make a simple Ajax call:

- First, display a diagnostic `alert()` popup to show the current text from the `Name` text box. This is a useful sanity check, to make sure the function is being called correctly and that it is getting the correct text from the text box.
- Empty the current contents of the `ResultList` control (you can use the jQuery `empty()` function to do this). This ensures that the list of matching names is emptied (and then re-populated) on every keystroke.
- Make an Ajax call via `$.ajax({ … })`, where the … represents a series of configuration options for the Ajax call. Replace the … with the following options for now:
  - `type: "GET"`
  - `url: "/players?q=" + $("#Name").val()`

  These configuration options represent an HTTP GET request to the URL `/players`. With the value of #Name input as the search text.
- After the Ajax call, display another diagnostic `alert()` popup just to confirm that the Ajax call was issued successfully (we'll show how to handle the returned data shortly).

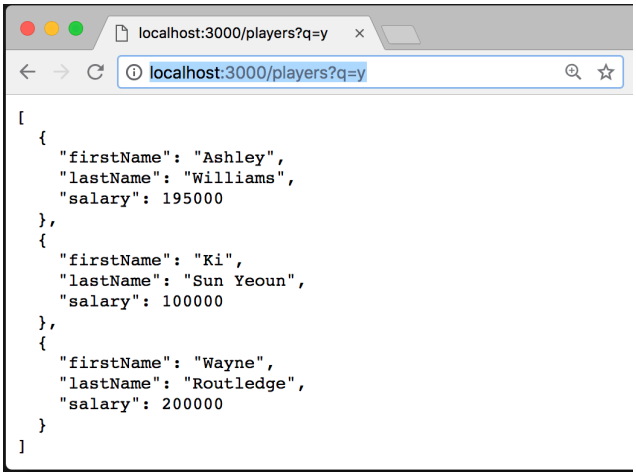
Run the Web application. In the text box, type a letter. Verify that a pair of `alert()` popups appear, showing that the keystroke was detected and that the Ajax call was issued successfully.

Exercise 3: Processing the Ajax response

The Ajax call you implemented in Exercise 2 causes the Web server to return data as a JSON result. To see what JSON data looks like, run your Web application again, and then modify the address in the browser address as follows (your port number may be different):

```
http://localhost:3000/players?q=y
```

This will query the players, which will return a collection of `Player` objects whose name contains "`y`" somewhere. The data is returned in JSON format, which looks like this in the browser window:



This is regular JavaScript object syntax for a collection of objects:

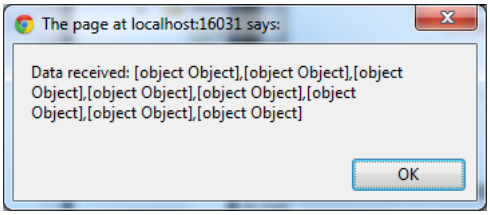
- The enclosing `[]` means "JavaScript array".
- Each item in the array is enclosed in `{}`, which means "JavaScript object".
- Inside each `{}` is a set of 3 `propertyName:propertyValue` pairs, to represent the `FirstName`, `LastName`, and `Salary` properties of each player.

Now that you understand the format of the data returned from the Web server, you can enhance your Ajax call to handle the returned data. Follow these steps:

- In your editor, re-open `\public\index.html` and locate the code that makes the Ajax call.
- In the `$.ajax({…})` call, set the `success` property to a call-back function as follows (this is obviously a simplified approach for now, but it's worthwhile working in baby steps with Ajax to make sure each step is working):

```
success: function (data) {
   alert("Data received: " + data);
}
```

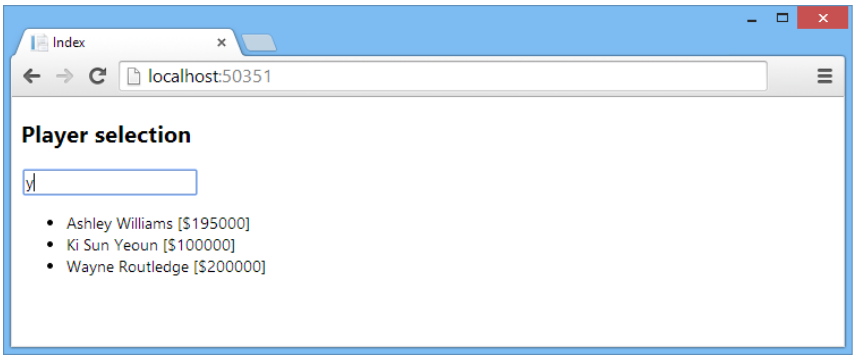
Run the Web application again. In the text box, type in a letter. Verify that you get an `alert()` popup such as the following, which verifies you've received an array of objects from the Web server:



Now enhance the `success` function so that it processes the `data` parameter properly. Bearing in mind that the `data` parameter is an array of player objects, here are some additional hints:

- Use the jQuery `$.each()` mechanism to process each item in the array.
- For each item, invoke a function to process it. The function should take two parameters (key and value). The key is the array index, and the value is the actual player object.
- The function should get the `ResultList` bulleted list and append an `<li>` element. The `<li>` element should display the `firstName`, `lastName`, and `salary` properties of the player object.

Run the Web application. In the text box, type in a letter. Verify that the Web page displays a list of matching players. For example:


Index
localhost:50351
Player selection
y
Ashley Williams [$195000]
Ki Sun Yeoun [$100000]
Wayne Routledge [$200000]

Exercise 4 (If time permits): Additional suggestions

- Handle potential errors in your Ajax call.
- Add `mouseover` and `mouseout` event-handlers for all `<li>` items within the `<ul>`, so that the `<li>` items change red when the user hovers over them (and back to black afterwards). For example: