WEB APPLICATION ARCHITECTURES

Ajax

Objectives

In this exercise you will learn how to access DOM elements using code

Reference material

This exercise is based on material from the “Ajax” chapter.

Overview

- In this lab you’ll exercise using Ajax in AngularJS projects.

Estimated duration

The estimated duration for this lab is 25 minutes.

Completed solution

There is a completed solution for this lab.

Step by step instructions

1. First of all, you will run a web server on your own computer. To do this:
   - Select the start menu, and type “cmd” to open a command window
   - Locate the WebService folder within the starters folder for this chapter
   - Type “cd” followed by the location of the WebService folder to change into that folder
     - Hint – you can drag the folder into the command window, and the location will be added to the current command
   - Type “node index” to run the web server
     - When finished (but not yet!), press Ctrl-C to end the web server
2. Next, open the Client folder in Code. Examine the starter. You will see that it is a very simple HTML page, with a button (“load data”) that currently does nothing.
3. Add a JavaScript file to your project, and in that file, create a simple AngularJS controller. Ensure that the controller’s function takes 2 parameters: $scope (as usual), and also $http. The controller should simply add a variable called “customers” to the $scope, which should be initialised to the value “null”. You will add more code to the controller soon.

4. Add a <script> tag that references this file. Add an ng-app attribute to the <body> of the HTML page. Wrap the contents of the body of the page in a <div> tag, and ensure the <div> has an ng-controller attribute that links it to the AngularJS controller.
5. Add an “ng-click” handler to the <input>, and create a function inside your controller which will handle the click. In this function, call $http(), passing a parameter with the following options:
    a. method:“GET”
    b. url:“http://localhost:8000/getcustomers”
6. Use a .then call to add a function that gets called when the $http Ajax call succeeds. This function should take a parameter called “results”. You can see examples of how to do all of this in the slides; note that the slides show $http being called from the main controller method, whereas you need to call it from the click event handler, so your code will be in a different place, but otherwise identical.

    In this function, store results.data (i.e. the data part of the results) in the $scope.customers variable that you created earlier.
7. In the HTML page, add a <div> below the <input>. Use ng-repeat to show the contents of the customers variable. You can choose which properties of the customers you want to show.

    Hint: the customers variable will be an array of objects. If you want to know the names of some of the properties, you might try pointing your web browser directly at the web service and looking at the Json it returns. You will see properties named CustomerID, CompanyName, and many others.

When you have time

8. Add another <div> below the results <div>. Set the contents of this <div> to “Loading…”.
9. Add ng-hide (or ng-show) attributes to the two divs, as follows:
    a. The results div should show whenever “customers” is not null. In the click event handler, you should set “customers” to be equal to null before starting the Ajax request
    b. The “loading…” div should be shown in the click event, and hidden in the results handler for the $http method. You will need an extra Boolean variable in your $scope to support this.

When you have even more time

10. See if you can use the same techniques to also show an error message if something goes wrong.

    Hints:

    - If your failure callback function has a parameter called “reason”, then “reason.data” may contain new lines, which won’t be rendered in the HTML. To fix this, you could create a CSS style, then use a class to apply that style to the relevant div. The style might look like this:

      ```
      div.white-space-pre-line {
          white-space: pre-line;
      }
      ```

    - You can test your error handling code by altering the Ajax request’s URL, so that it points to an invalid URL.