# Getting Started with HTML, CSS, and JavaScript

## Overview

In this lab, you’ll review a complete Web application to get up to speed quickly with common HTML, CSS, and JavaScript features. We’ll explain the details as we go along, so you can understand how it all fits together.

## Source folders

* <Lab\_HOME>/labs/GettingStarted
* <LAB\_HOME>/Solutions/GettingStarted

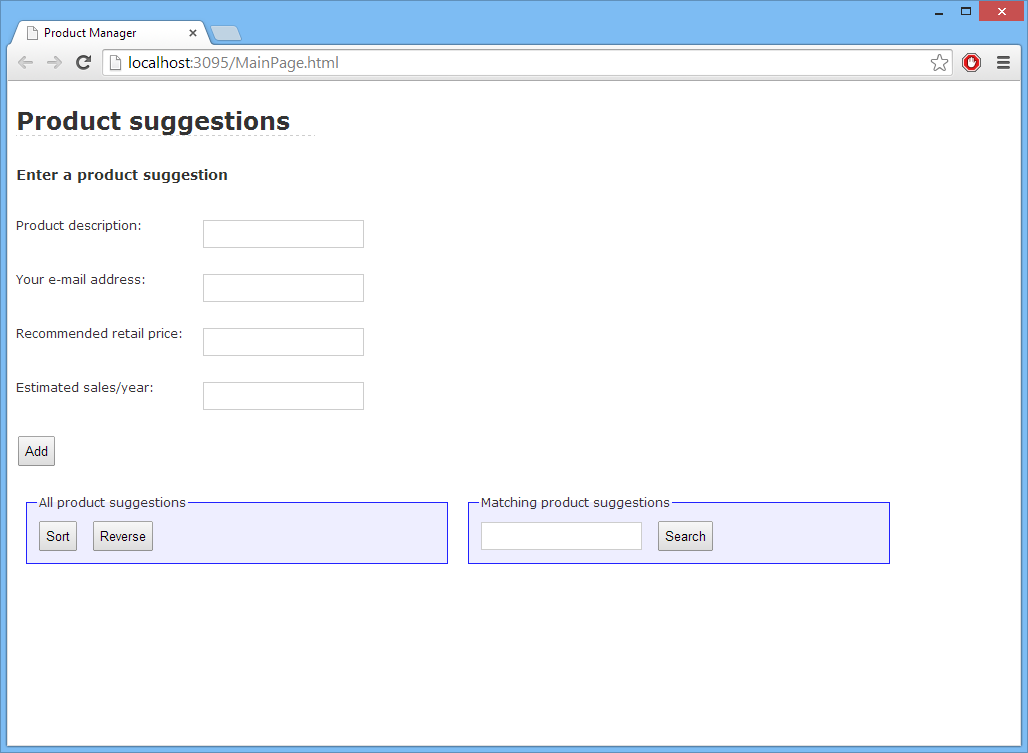
## Roadmap

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. Working with strings and dates
3. Working with arrays
4. Working with regular expressions (if time permits)

## Familiarization

The project contains a single HTML page named MainPage.html, which is the start-up page in this Web application. Run the Web application (*Ctrl+F5*). The web page appears as follows:

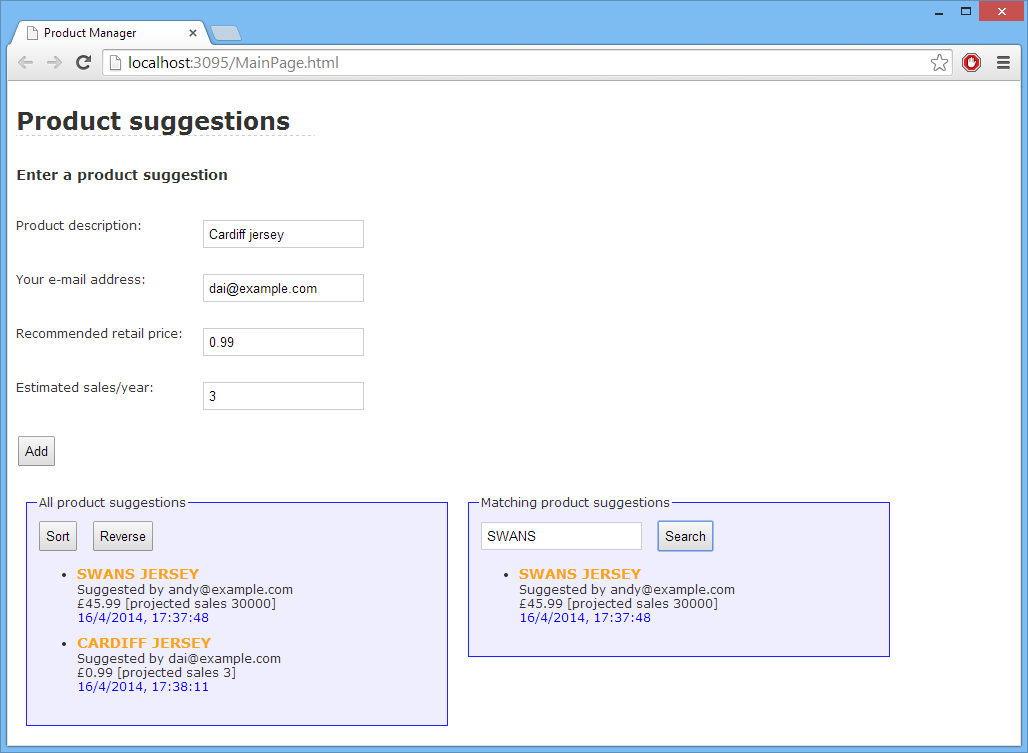


The web page allows the user to suggest new products that a company might add to its product range. The user can enter a description of the product, their email address, a recommended retail price for the new product, and an estimate of the number of units that might be sold in a year.

When the user clicks *Add*, the Web page adds the product suggestion details (as a formatted string) to an array behind the scenes. The contents of the array are displayed in the *All product suggestions* panel in the lower-left of the Web page. The user can also sort and reverse the array, via the *Sort* and *Reverse* buttons.

On the lower-right of the Web page, the *Matching product suggestions* panel allows the user to search for products that match a regular expression pattern entered by the user. The matching records are displayed as a list in that panel.

The following screenshot shows how the Web page might appear after adding several product suggestions and doing a search:



## Play around with the Web page until you’re happy with how it all works, then close the browser. The following exercises explain how that all worked ☺.Exercise 1: Understanding the project structure

Open MainPage.html and observe the following points:

* The <head> links in a predefined style sheet named Stylesheet.css. Take a look at this style sheet if you like. For more info about CSS styles, see Appendix A.
* The <head> includes a script file named ProductSuggestionsFunctions.js. Take a look at this script file. The script file contains various functions that we’ll explain during this lab. For more info about essential JavaScript syntax, see Appendix B.
* Back in MainPage.html, the <body> defines all the HTML markup. Note the following points:
  + When the user clicks the *Add* button, it causes the doAdd() function to be called. This function returns false, to prevent the form from submitting itself to the Web server.
  + The first <fieldset> element displays all the product suggestions (the product suggestions will actually be displayed in the <span> whose id is allProductsList). The <fieldset> also contains *Sort* and *Reverse* buttons, which invoke the doSort() and doReverse() functions respectively.
  + The second <fieldset> element allows the user to search for product suggestions that match a regular expression. The results will be displayed in the <span> whose id is matchingProductsList.

## Exercise 2: Working with strings and dates

Open the ProductSuggestionsFunctions.js script file. Note that it kicks off by defining a global variable named allProducts – this is an empty array.

Now locate the doAdd() function. The purpose of this function is to get the values entered by the user in all the text boxes, build a formatted string, and then add it to an array.

The function begins by getting the values entered in all the text boxes:

* To get the value of a text box, we use code such as the following:

var *elemValue* = document.getElementById(*"elemId"*).value;

For example, to get the value of the *description* text box (whose ID is description):

var *description* = document.getElementById("description").value;

* The *email* field is retrieved as a simple string.
* The *price* field is converted into a floating-point number with 2 decimal places, by using JavaScript’s parseFloat() function.
* The *sales* field is converted into an integer, via JavaScript’s parseInt() function.

Next, the code creates a string variable to format these values. Here’s an explanation of the string handling code (for more info about the standard String object in JavaScript, see Appendix C):

* We use the toUpperCase() function on the standard String object to convert the product description to uppercase.
* We use the big(), bold(), and fontcolor() functions on the String object to add HTML formatting to the string (i.e. to set the font size, font weight, and font colour).

Next we append the current timestamp to the string, by using functions on the standard Date object (see Appendix C for more details about Date):

* We get the current date and time, simply by creating a new Date object.
* We use various Date functions to get the date (i.e. day), month, full year, hours, minutes, and seconds.
* We use the pad() helper function (at the bottom of the script file) to ensure that 2-digit time fields (hours, minutes, seconds) are displayed with a leading zero if necessary.

This is how the details appear when displayed:



**Exercise 3: Working with arrays**

In this exercise you’ll examine the array-handling code, which adds each product suggestion to an array and displays the array items as a bulleted list on the Web page.

*Aside: In this lab, the array will contain simple HTML strings. In the next lab, you’ll refactor the Web application so that it uses fully object-oriented Product objects instead, which is a much more elegant approach.*

To get started, note again that we declared a global variable named allProducts to hold an empty array. Then, near the end of doAdd(), we push the current product suggestion into the array and then call displayProducts() to display all the products suggestions. Note that the displayProducts() function takes two parameters:

* The array to display. We want to display all the product suggestions, so we pass in allProducts here.
* The id of the target element where we want to display the array data. We pass in “allProductsList” here (this is the id of the <span> where the full product list will appear).

Now locate displayProducts(). The function displays all the items in the specified array as a bulleted list, in the specified target element. Note the following points:

* We build up a formatted string of HTML as follows:

<ul>  
 <li> … </li>  
 <li> … </li>  
 <li> … </li>  
</ul>

* We loop through the array, and wrap each array item in a <li> … </li>.
* To assign the HTML to the target element, we use the following code:

var targetElement = document.getElementById(targetElementName);

targetElement.innerHTML = str;

Now locate the doSort() function. This function sorts the array and redisplays the sorted array data.

Finally locate the doReverse() function. This function reverses the array and redisplays the reversed array data.

## Exercise 4 (If time permits): Working with regular expressions

Regular expressions are great for pattern matching. We use regular expressions in doSearch() to search for all product suggestions that match a regular expression entered by the user. Note the following points in the doSearch().

The user will enter the regular expression in the searchString text box. We get the value of this text box and create a new RegExp object based on this string, as follows:

var searchString = document.getElementById("searchString").value;

var re = new RegExp(searchString);

* Next, we create a new array ready to hold all the matching product suggestions.
* We loop through the array of all product suggestions (allProducts) and test whether each item matches the regular expression. If it does, we add it to our “matches” array.
* After the loop, we display the “matches” array in the “matchingProductsList” target element (we use the displayProducts() function to do this).

Run the Web application to remind yourself how this works. Add some product suggestions, and then try out the search functionality. You should see a result such as the following:

