JavaScript OO Programming

Overview

In this lab, you'll refactor the "product manager" Web application from the previous lab so that it uses a fully object-oriented `Product` object.

The `Product` object will hold a description, email, price, projected sales, and timestamp. You will also define a method to format this information as a string. If time permits, you'll implement another method to determine if a product's description matches a regular expression.

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. Defining a `Product` object
2. Adding methods to the `Product` object
3. Refactoring the Web application to use the `Product` object
4. Supporting regular expressions

Server

HTML files can be opened from the file system, but for security reasons browsers will limit the possibilities. It is better to open a server in the project folders. Here are two ways to do that.

Visual Studio Code

1. Open Visual Studio Code
2. Choose menu "File | Open…" and open the root folder of the course material. After opening you should see a folder called ".vscode" at the top of the tree in the Explorer on the left.
3. Choose menu "Tasks | Run Task…" and choose task "npm: install".
4. Select/Open a file in the folder that you want to be the root of your server. This will generally be the homepage of your app.
5. Choose menu "Tasks | Run Build Task…" or choose the shortcut.
6. To close the server again, choose menu "Tasks | Terminate Task…".

Command line

1. Globally install "live-server" with command "npm install -g live-server" (see: https://www.npmjs.com/package/live-server)
2. Open the command line (Windows) or terminal (macOS) in the folder that you want to be the root of your server. This will generally be the homepage of your app.

3. Run “live-server” to open the server. Choose Ctrl+c to close the server again.

Getting started with the Student project

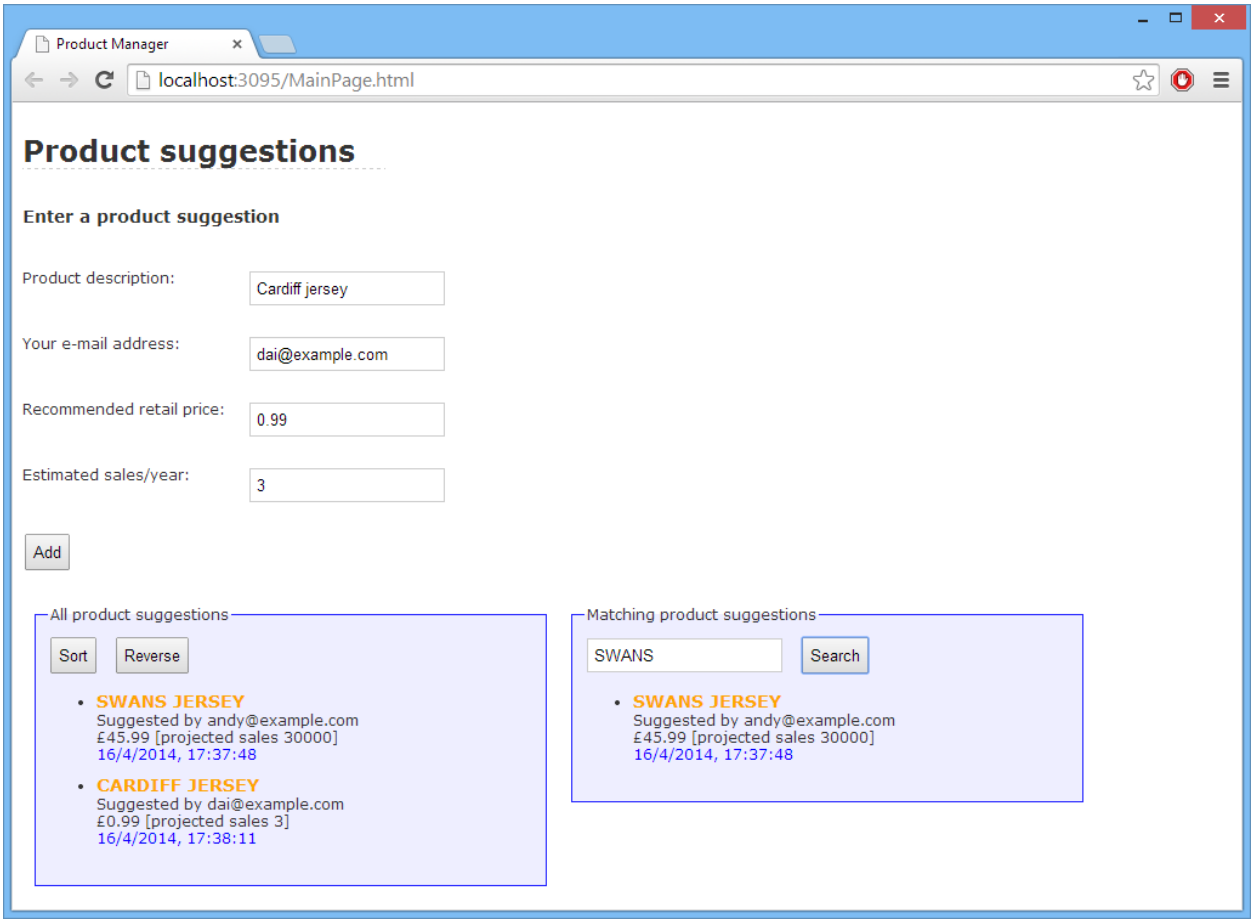
In Visual Studio, open the *Start* folder for this lab as follows:

- *Folder:*
  `Start`

Take a look at the following files to remind yourself how the application works (this is the same as the solution code to the previous lab):

- `MainPage.html` is the fully-complete Web page for this Web application.
- `ProductSuggestionsFunctions.js` contains all the initial JavaScript code for this Web application. At the moment, it uses a string to represent each product. In this lab, you'll refactor this code to use a `Product` object instead.

Run the Web application and familiarize yourself with its capabilities.



Exercise 1: Defining a Product object

The best way to define a new object is by implementing a constructor function. Therefore, open `ProductSuggestionsFunctions.js` and define a constructor function as follows:

- Declare a global variable named `Product`.
- Assign the `Product` variable a constructor function. The constructor function should take 4 parameters representing the following product-related info:
    - description
    - user's email
    - recommended retail price
    - estimated sales per year
- Inside the function body, assign the following properties to "this" object:
    - `description`
    - `email`
    - `price`
    - `sales`
    - `ts` (this is the timestamp of creation –assign the current date/time)

Exercise 2: Adding methods to the Product object

Now it's time to add functionality to `Product`. To do this, set the `Product.prototype` property so that it includes a `format` function as follows:

- The function should create a formatted string that contains the product's description, email, price, and sales (you can get most of this code from the existing `doAdd()` function – you might need to make some tweaks).
- The function should then append the formatted timestamp (again, you can get most of this code from the existing `doAdd()` function, with tweaks).
- The function should then return this concatenated text.

Exercise 3: Refactoring the Web application to use the Product object

In this exercise, you'll refactor your code so that it uses the `Product` object.

- First, refactor `doAdd()` so that it creates a `Product` object to contain the information entered by the user, and then add the `Product` object (rather than a formatted string) in the global `allProducts` array. Why is it better for the array to hold `Product` objects rather than formatted strings?

- Next, refactor `displayProducts()` so that it makes use of the `Product` object's `format()` method to return a formatted string representation of each product, ready to be displayed on the form.

- Finally, refactor `doSort()` so that it sorts the array of `Product` objects by description.

  The default behavior of the array `sort()` function is to call `toString()` on each object in the array, to compare items lexicographically. Therefore, a simple solution would be for you to implement a `toString()` method in the `Product` prototype that just returned the product's description. The array `sort()` function would then call `toString()` repeatedly to sort `Product` objects by description.

  Another (more flexible approach) is to supply a function to the array `sort()` function, to tell it how to compare two `Product` objects explicitly. Here's what we suggest for your `doSort()` function:

```
function doSort() {
    allProducts.sort(function (p1, p2) {
        if (p1.description < p2.description)
            return -1;
        else if (p1.description > p2.description)
            return +1;
        else
            return 0;
    });
    displayProducts(allProducts, "allProductsList");
}
```

Run the Web application. Add several product suggestions and verify the Web page displays them correctly. Also verify that the *Sort* and *Reverse* buttons still work correctly.

Exercise 4 (If time permits): Supporting regular expressions

Refactor your Web application so that it supports regular expression tests on `Product` objects. The recommended policy now is to just test the product's `description` property (rather than testing entire formatted product suggestion strings as before, with the possibility of all the embedded HTML markup such as `<font>` and `<bold>` getting in the way).