# Collections

## Overview

In this lab you'll work with a slightly modified version of the “product suggestions” application. The application will allow users to perform full create/read/update/delete operations on products. Each product will have a unique ID, and you'll use a Map to enable fast lookup of products based on their ID.

If time permits, you'll also implement "tag cloud" functionality in the web app. The user will be able to specify tags for a product when adding it, and can then look up all products that have a particular tag. You'll use a Set to avoid duplicate tags in the tag cloud.

## Source folders

* C:/JsDeepDive/Labs/Student/07-Collections
* C:/JsDeepDive/Labs/Solutions/07-Collections

## Roadmap

There are 7 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. Familiarization with the solution application
2. Understanding the student starter code
3. Refactoring the Web application to use a Map
4. Getting a product
5. Updating a product
6. Deleting a product
7. (If Time Permits) Implementing tag cloud functionality

## Exercise 1: Familiarization with the solution application

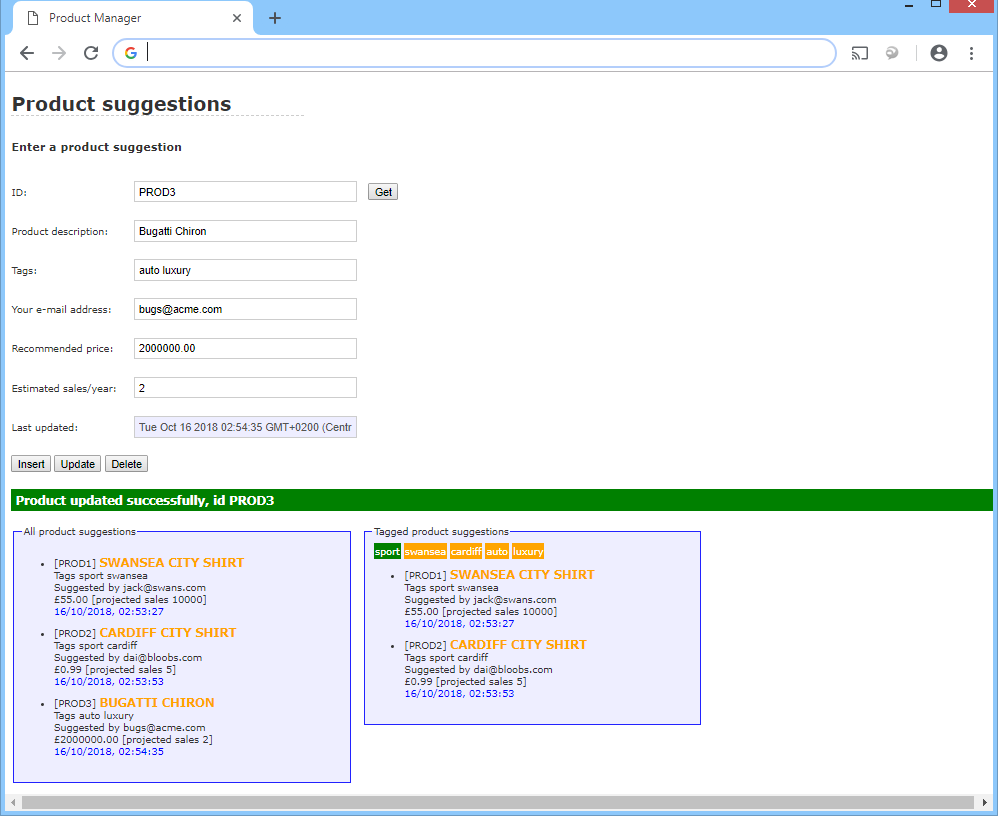
Open a Command Prompt window, go to the C:/JsDeepDive folder, and run this command:

npx gulp

Go to the **solution** folderfor this lab and open index.html in the browser. Spend some time playing with the app, to understand how it works. The upper part of the web page allows the user to insert, update, and delete products. Each product has an automatically-generated ID (e.g. PROD1, PROD2, etc.). You can get a product by its ID, if you know it. Also note you can specify tags for a product, e.g. in the screenshot below we've entered the tags *auto luxury* for the Bugatti Chiron.

The lower part of the web page contains two panels:

* The *All product suggestions* panel on the left-hand side displays all the products. This panel is redisplayed whenever the user inserts, updates, or deletes a product.
* The *Tagged product suggestions* panel on the right-hand side displays all the tags for all the products (no duplicates). These are actually hyperlinks. When the user clicks on a tag hyperlink, the panel displays all the products that have that tag.



## Exercise 2: Understanding the student starter code

Switch to the **student** folderand take a few minutes to understand index.html in a text editor. This file is already complete – it contains all the HTML markup for the entire lab.

Now take a look at the JavaScript code in es6scripts/script.js. Note the following points to get started:

* The global nextId variable simulates a static variable for the Product class. Every time a Product object is created, it's assigned a unique id based on nextId. (Note that nextId is incremented every time a Product is created).
* The global allProducts variable is an array at the moment. You'll change this to a Map in the next exercise, and tweak the various bits of code that use it (so that the code uses the Map API, rather than array techniques).
* The Product class has been enhanced with some extra properties (id is a generated product id, tags is a string such as *"auto luxury"*, and ts is the timestamp when the product was created or last modified).

Now take a look at the doInsert() function. Note the following points:

* doInsert() begins with a call to readProductDetail(), to read product details entered by the user. The readProductDetail() function has an id parameter, which defaults to undefined (this indicates we don't know the product id yet). The readProductDetail() function returns a new Product object initialized with the data from the user.
* Back in doInsert(), we insert the new Product into the allProducts array. You'll refactor this code in the next exercise, to insert the Product into a Map instead.
* At the end of doInsert(), we call displayProducts() to display the products in the *All product suggestions* panel. We also call displayTags() to display the tags in the *Tagged product suggestions* panel. Both of these functions receive an array – bear this in mind in the next exercise, when allProducts will be a Map not an array…

Also note the following "CRUD" functions aren’t implemented at all yet. You'll implement these functions later, to get/update/delete a product based on its id in the Map:

* doGet()
* doUpdate()
* doDelete()

**Exercise 3: Refactoring the Web application to use a Map**

Refactor the JavaScript code so that allProducts is a Map, rather than an array (it's much easier to perform lookups on a Map rather than an array). Each entry in the Map will contain a key/value pair as follows:

* The key will be a Product id.
* The value will be the associated Product object.

You'll also need to make a few quick changes in doInsert():

* Locate the statement that pushes a product into allProducts as an array. Change this statement to insert the product into allProducts as a Map.
* Locate the call to displayProducts(). This function must receive an array, so tweak the function call to pass in just the values (i.e. not the keys) of allProducts.
* Locate the call to displayTags() and tweak it in the same way.

Save your JavaScript file and make sure it transpiles successfully. Then refresh the Web page in the browser and try inserting some products. Verify they are inserted and displayed correctly.

**Exercise 4: Getting a product**

Implement the doGet() function to get a product from allProducts, based on the id entered by the user.

* If the id is valid, call displayProductDetail() and pass the gotten product as a parameter. Also display a success message via the setHtml() method as follows:
  + For the 1st parameter, pass the CSS selector '#messageArea'
  + For the 2nd parameter, pass a suitable success message
  + For the 3rd message, pass the CSS class name 'valid'
* If the id is invalid, call displayProductDetail() with an undefined parameter, to clear all the text boxes on the web page. Also display an error message via the setHtml() method as follows:
  + For the 1st parameter, pass the CSS selector '#messageArea'
  + For the 2nd parameter, pass a suitable error message
  + For the 3rd message, pass the CSS class name 'error'

Save your JavaScript file and make sure it transpiles successfully. Then refresh the Web page in the browser, insert some products, and then try to get them via the *Get* button. Verify the Web app behaves properly for valid and invalid product IDs.

**Exercise 5: Updating a product**

Implement the doUpdate() function to update an existing product in allProducts, based on data entered by the user.

The first step is to get the id entered by the user. If the id is valid:

* Call readProductDetail() to read the product details entered by the user. Pass the id as a parameter. You'll get back a Product object containing the specified id and the updated product details from the user.
* Update the existing entry in allProducts, so the id now refers to the updated Product object.
* Display a suitable success message, via setHtml().
* Call displayProductDetail() to display the updated product on the web page.
* Call displayProducts() and displayTags(), to redisplay the *All product suggestions* panel and the *Tagged product suggestions* panel respectively.

If the id is invalid, do this instead:

* Display a suitable error message, via setHtml().
* Call displayProductDetail() with an undefined parameter, to clear all the text boxes on the web page, to avoid any possible confusion for the user.

Save your JavaScript file and make sure it transpiles successfully. Then refresh the Web page in the browser, insert some products, and then try to update some via the *Update* button. Verify the Web app behaves properly for valid and invalid product IDs.

**Exercise 6: Deleting a product**

Implement the doDelete() function to delete an existing product in allProducts, based on the id entered the user. No more clues for this one – you can probably figure out what you need to do yourself by now ☺.

**Exercise 7 (If time permits): Implementing tag cloud functionality**

When the user inserts a new product, he/she can specify a series of tags to describe the product. This allows users to select products based on their tag.

Take another look at the **solution** application and remind yourself how the *Tagged product suggestions* panel works. Notice the tags are unique – if multiple products have the same tag, the tags aren’t repeated. You can click on a tag to see all products that have that tag.

Implement this functionality in the **student** application, with all the bells and whistles.