# Modules

## Overview

In this lab you'll refactor the good-old “product suggestions” application so that the code is spread across several modules. Remember, an ES6++ "module" is a JavaScript file that exports or imports other modules.

## Source folders

* C:/JsDeepDive/Labs/Student/Annex-Modules
* C:/JsDeepDive/Labs/Solutions/Annex-Modules

## 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. Familiarization
2. Modularizing the code
3. Running the application
4. (If Time Permits) Additional suggestions

## Exercise 1: Familiarization

Open a Command Prompt window, go to the C:/JsDeepDive folder, and run the following command to start the Babel transpiler:

npx gulp

Now go to the *student* folder and take a look at the various files. This should all look familiar by now. We've made some minor changes, but nothing seismic. Make sure you understand all the code in es6scripts/script.js before you go any further.

Open index.html in a browser window and take the Web page for a test drive, to remind yourself how it all works.

## Exercise 2: Modularizing the code

The code in es6scripts/script.js is quite monolithic – it contains a jumble of class definitions, global variables, event-handler functions, and general UI functions.

Split script.js into a series of smaller modules. Here's what we suggest, but it's up to you:

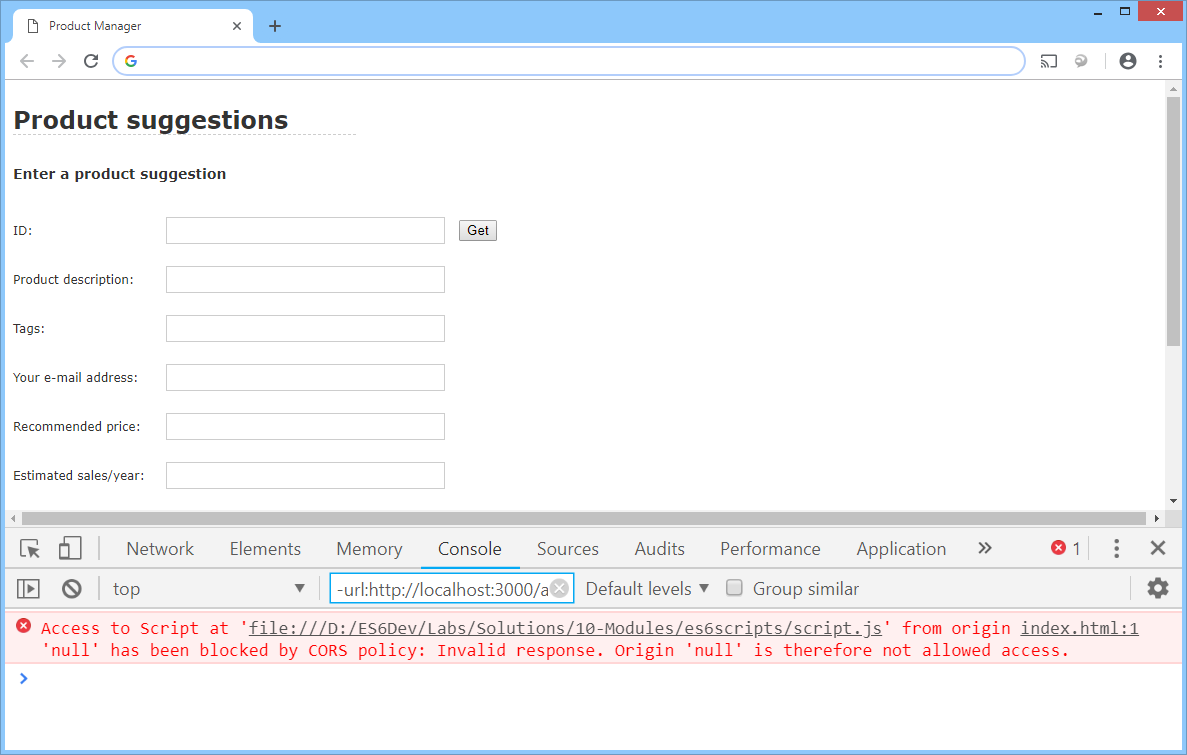
* util.js – Defines the Util class.
* product.js – Defines the Product class.
* userinterface.js – Defines functions to display/read content on the UI.
* script.js – Top-level script file, hooks up and implements event-handler functions.

Think carefully about what classes/functions/variables you want to export from each module, and make each module imports the things in needs.

Note: There's also a change you need to make in index.html. In the <script> tag that imports script.js, add type="module". This tells the browser to treat the file as an ES6 module, rather than as a vanilla script file.

## Exercise 3: Running the application

Try opening index.html in a browser. It won't work. Take a look in Chrome Developer Tools (F12) and you'll see an error in the Console window, indicating CORS requests are not supported.



CORS is "cross-origin requests", and generally it's a good thing – it allows a Web app to send HTTP requests to different origins (e.g. domains) than that from whence it came.

ES6++ modules don't work for files delivered from the file system – you must serve files from a web server. The easiest way to do that is via the NPM serve utility. Follow these steps:

* Open another Command Prompt window and go the *student* folder for this lab.
* Run npx serve.
* This starts a Web server, listening on port 5000 by default.

Go back to the browser and navigate to http://localhost:5000/index.html. This should now work correctly. Verify your application still works after your modularization efforts!

## Exercise 4 (If time permits): Additional suggestions

Explore the various syntax options for exporting and importing artefacts across modules. See what happens if you to try to access an artefact that hasn't been exported by a module.