**Assignment 8 – WhatABook, Part 2**

# Setup

Each week you will be asked to create a new folder under web-330 following a naming convention of **<week>-number**. If we are on week two, the folder name should be **week-2**. All files associated with the weekly assignment will be added to the appropriate folder. All programs must be linked in the appropriate landing page. Projects will be linked under the

“Projects” section of the index.html landing page.

The document title of all HTML files in this course must say “WEB 330 – Enterprise JavaScript

II.” And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and 231. As part of your submission, be sure to include screenshots of the results from the validation tests (HTML and CSS validators).

**User interface styling and formatting requirements are located in the Web 330 HTML, CSS, and JavaScript Requirements document.**

HTML: **<yourLastName>-whatabook2.html** CSS: **<yourLastName>-whatabook2.css** JS: **http-client.js**

# Grading Reminders

A. (rubric) All code sources (.html, .css, .js) are measured against

1. Code functionality: Does it work? Does it meet requirements?
2. Adherence to standards and conventions. Are you using the appropriate data types, including proper indention, are variables named appropriate (variable x is an example of poor naming conventions), is there an appropriate use of whitespace, is the code organized, and are semicolons being used to terminate code sentences?
3. Efficiency: Use of language features. Are you practicing DRY (Don’t-RepeatYourself?), are you leveraging built-in language features where appropriate, and are you using classes/functions to reduce code clutter?
4. Documentation: Code is maintainable by others
   1. Code comments are present in all blocks of code, written as full sentences, free of grammatical errors, and function/class parameters and data types have been identified.
   2. Code attribution is present in all files and authorship is clearly annotated.
5. Error trapping/handling. Are there errors in the program? Is there evidence of coding best practices to reduce user errors?
6. Assignment Specific Compliance. Does the delivered solution follow the instructions, as they are written? Does the output match what was provided in the screenshots (including spaces, styling, etc.)?

# Required Modifications

* Cite any sources in your opening programmer’s comment
* Link the appropriate CSS, JavaScript, and Google fonts
* http-client.js

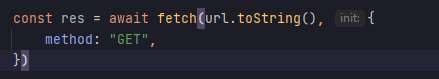
Additional JavaScript Requirements a. Create a class named HttpClient

b. In the body of the class create an async function named get with two parameters: url and params. Set the params parameter to a default empty string.

Additional JavaScript Requirements

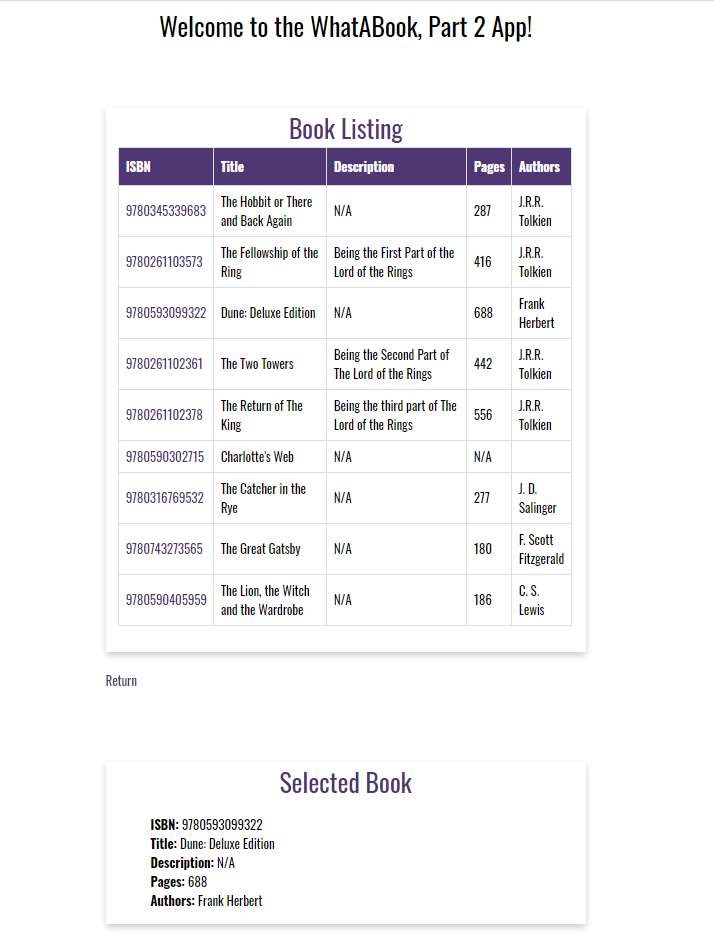
* 1. Instantiate a new URL object, supplying it the url parameter and assign the results to the url parameter: **url = new URL(url)**
  2. Instantiate a new URLSearchParams object, supplying it the params parameter and assigning it to the url.search property.
  3. Create an object literal named res and using the fetch() API passing in the url.toString() variable and specific the request as a GET request.
  4. Return the res object literal as JSON.

Exhibit A. fetch API call



* 1. Export the class.

# Exhibit B. User Interface (final solution)

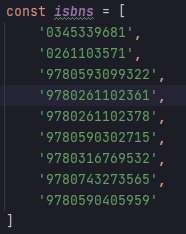


1. The final solution in Exhibit B is identical to the final solution in Assign\_7. This includes the functionality and HTML code. The main difference is how we implement the process. In assignment seven we read the data from an XML file. In this week’s assignment we will be making an API call to [https://openlibrary.org/api/books.](https://openlibrary.org/api/books) To this end, I will not be discussing the styling or HTML structure because you should have a working solution from assignment seven.

# <yourLastName>-whatabook2.html

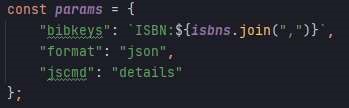
1. ~~Add an import statement for the HttpClient class.~~
2. ~~Instantiate a new HttpClient object and assign it to a new variable named http.~~
3. Create a variable named isbns and assign it an array of ISBN numbers (see below)

# Exhibit C. ISBNs



1. ~~Create an object literal named params with the following key/value pairs~~

**Exhibit D.**



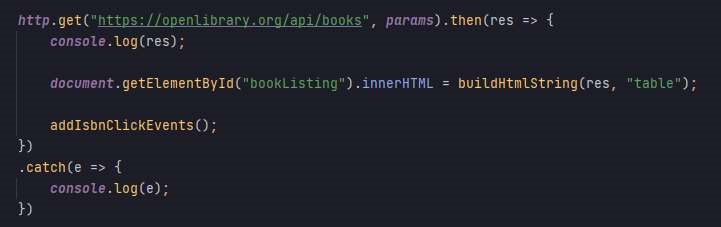
1. Call the http.get function and supply it with the open library URL and the params objects.

# Additional JavaScript Requirements

1. ~~Add a then clause using arrow functions with a res object~~
2. ~~In the body of the call, call the buildHtmlString() function supplying it with the res object~~

~~and the string value “table” and bind the results to the bookListing div.~~

**Exhibit E.**



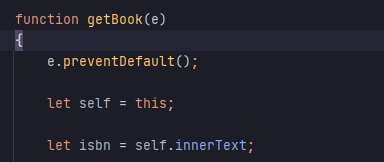
1. ~~Call the addIsbnClickEvents() function~~
2. ~~Add a catch() statement with an arrow function for error and write the error to the console using the console.log() API.~~

f. Create a function named getBook() that accepts an event object

# Additional JavaScript Requirements

1. ~~Create a variable named self and assign it the value this~~

**Exhibit F.**



1. ~~Create a variable named isbn and assign it the innerText of the self variable.~~
2. Build a new params object literal (see Exhibit D.)
3. Call the http.get function supplying it the open library URL and params object literal.
4. Add a then clause and in the body call the buildHtmlString function supplying it the res object and “ul” string value and bind the results to the selectedBook div.
5. Add a catch() clause for the error object and write the error to the console using the console.log() API.

g. Create a function named buildHtmlString with two parameters: res and format.

# Additional JavaScript Requirements

1. Create a variable named tableString and supply it with an HTML string for a table header.
2. ~~Create a variable named ulString and set it to an empty string.~~
3. Using a for…in statement, iterate over the res parameter and append the objects to the ulString and tableString variables. See Exhibit G. Please note, the code snippet in Exhibit G is a partial view of the solution. You will need to write the remaining pieces of code.

**Exhibit G.**



1. ~~Outside of the for…in loop, close the HTML tableString~~
2. Add an if statement that checks the format variable. If the format variable is a “table” return the tableString variable. Otherwise, return the ulString variable.