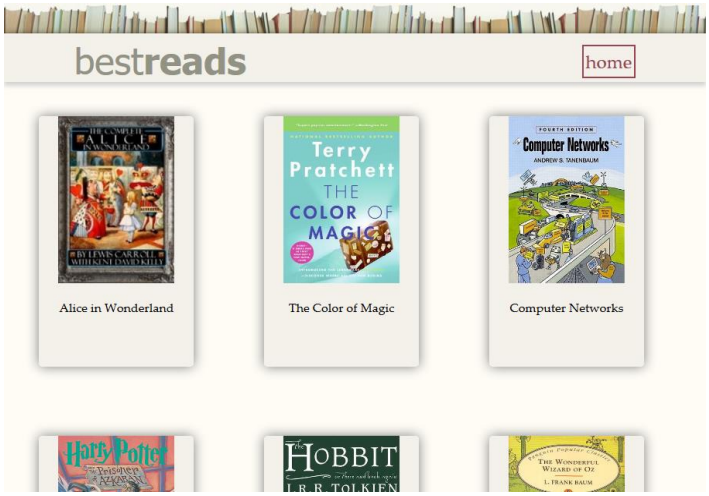
University of Arizon, CSc 337
Homework Assignment 8: Book Review

This assignment is about making a web service and using Ajax to retrieve data from it. You will write a NodeJS service that will generate a variety of data about different books depending on the parameters it is sent. You will also write Javascript code to make requests to this service and inject the responses into a page. Turn in these files:

- **bestreads_service.js**, the NodeJS service that will supply the book data
- **bestreads.js**, the Javascript that will request the information from **bestreads_service.js** and inject it into **bestreads.html**

bestreads.html and **bestreads.css** are provided for you. **You may not modify them**.



When the page loads it should display the images and titles of each book that you have data for and the `singlebook div` should be hidden. If the user clicks a book cover or title all of the books and titles should be removed from the page and the `singlebook div` should be shown. You should send a request to the server for data for this book and display its cover image, title, author, rating, description and reviews.

From the course web site you will download a **.ZIP archive** of input files for many books, described on the next page. Unzip this file into the same directory as your HW8 files, so that the files will be located in relative paths such as **books/harrypotter/info.txt** and **books/wizardofoz/cover.jpg**. Your code should assume these are the relative paths to use.

Web Service Details:

Your **bestreads_service.js** service will provide different data based upon a couple *query parameters* named `mode` and `title` that are passed from your Javascript to the page in its URL. The value of `mode` should be `description`, `info`, `reviews` or `books` depending on which information you want. The value of `title` should be a string representing the single book to display. The browser will request your page with a URL such as the following:

http://localhost:3000?mode=description&title=harrypotter

Your NodeJS code can store these parameters into variables using code such as the following:

```
var book = req.query.title;
```

All of your NodeJS code should use these parameters' values, and you should never hard-code particular book names. Your code may assume that the browser has properly supplied these parameters and has given them valid values. You may assume that the book exists and has a corresponding folder of valid input data and images. You do not have to handle the case of a missing or empty `mode` or `title` parameter, a value that contains bad characters, a value of a book that is not found, etc.

Based upon the mode variable's value, your service must output that data. All book data is in a `books` directory. Each book is stored in a directory named the same as your query parameter in that `books` directory. For example, the book `harrypotter` stores its files in a folder named **books/harrypotter/** . You may assume that the files exist and are valid at all times.

The behavior of each mode is described below:

- **mode=info:** The title parameter must also be passed with this mode. Your service should output the contents of **info.txt**, a file with three lines of information about the book: its **title**, **author**, and **number of stars**, as JSON. For example:

```
{"title":"Harry Potter and the Prisoner of Azkaban",
 "author":"by J.K. Rowling, Mary GrandPre(Illustrator)",
 "stars":"4.5"}
```

- **mode=description:** The title parameter must also be passed with this mode. Your service should output the entire contents of **description.txt**, a file with information about the book, as plain text.
- **mode=reviews:** The title parameter must also be passed with this mode. Output JSON for all of the reviews for the book. All of the reviews should be in a list called "reviews". Each review should contain a reviewer name, number of stars and review text. The reviews are stored in files called **review1.txt**, **review2.txt**, etc. Each file contains one review for each book which is exactly three lines: The reviewer's name, the number of stars they gave the book and their review. If a book has 10 or more reviews, the names will be e.g. **review01.txt**, ... So don't hard-code file names like `"review1.txt"`; look for all files that begin with `"review"` and end with `".txt"`. Example output:

```
{"reviews": [{"name": "Wil Wheaton", "stars": 4,
              "review": "I'm beginning to wonder if there will ever be a
                        Defense Against The Dark Arts teacher who is just a
                        teacher."},
              ...
             ]
}
```

- **mode=books:** Outputs JSON containing the titles and folder names for each of the books that you have data for. The JSON for each book should be stored as a list element in a list accessed with the name `books`. Each book should contain a `title` and a `folder`. If a new folder of book data is added to your books folder your program should automatically adjust to include it the next time your service is called. Example output:

```
{"books": [{"title": "Harry Potter and the Prisoner of Azkaban",
            "folder": "harrypotter"},
           {"title": "The Wizard of Oz",
            "folder": "wizardofoz"},
           ...
          ]
}
```

Javascript Details:

Your **bestreads.js** will use Ajax fetch to request data from your NodeJS service and insert it into **bestreads.html**. Here is the functionality your page should have:

- When the page loads it should request all of the books (`mode=books`) from the web service. It should display each of these books by adding the image of the book's cover and the book's title (in a paragraph) to a `div` and adding that `div` to the `allbooks` div already on the page. The `singlebook div` should be hidden.
- If the `home` button on the upper right is clicked it should do the same thing that the page does when it loads. Even if the button is pressed multiple times very quickly, only one listing for each book should appear on the page.
- When a user clicks on a book cover or title of a book the `allbooks div` should be emptied out and the `singlebook div` should be shown. You should then request the info, description and reviews for that book from the server. The title, author and stars gotten from the info request should be inserted into the elements with ids matching their names. The description can be directly inserted into the paragraph with the id `description`. The reviews should be inserted into the `div` with the id `reviews`. Output the reviewer name and number of stars in an `h3` and the review text in a paragraph. For example the code in the `allbooks div` might be:

```
<h3>Wil Wheaton<span>4</span></h3>
<p>I'm beginning to wonder if there will ever be a Defense Against The Dark Arts teacher who is just a teacher.</p>
<h3>Zoe<span>5</span></h3>
<p>Yup yup yup I love this book</p>
<h3>Kiki<span>5</span></h3>
<p>Literally one of the best books I've ever read. I was chained to it for two days. </p>
```

Development Strategy and Hints:

NodeJS code is difficult to **debug** if you have errors. Write the program **incrementally**, adding small pieces of code to a working page, and not advancing until you have tested the new code. The following functions may be helpful:

Implementation and Grading:

For full credit, your JS code should follow the rules listed in the **Style Guide** on the class web site. Your page's output must successfully pass the W3C **HTML validator**. (Not the NodeJS service but your **bestreads.html** after you have inserted data you have gotten back from the service in it. To validate your page, view the page, then choose View Source in your browser and copy/paste it into the validator.) Your JavaScript code should pass our **JSLint** tool with no errors. Your .js file must run in **JavaScript strict mode** by putting `"use strict";` at the top.

We will also grade the **style of your NodeJS and JS code**. Do not use the `global` keyword. Use indentation/spacing, and avoid long lines over 100 characters. Avoid redundant code, and use parameters and return values properly. Make extra effort to minimize **redundant code**. Capture common operations as functions to keep code size and complexity from growing. For full credit your code must have **at least one function** in each language to remove redundancy and/or capture structure.

Make sure to test your code on all available books from the ZIP file. You may want to think about other edge cases, such as what your page should do if the book has only a single review, etc. You should not have code that depends on particular books or uses `if/else` statements to see which book to display.

Use the **"module pattern"** shown in class to wrap your JS code in an anonymous function. **No global variables or symbols**, nor "module-global" vars, are allowed in JS on this assignment; values should be declared at the most local scope possible. If a constant value is used, you may declare it as a module-global "constant" variable `IN_UPPER_CASE`.

Fetch data using **Ajax fetch**. Be careful to avoid redundancy in your Ajax code; if you do similar Ajax operations in many places, make a helper function(s). Process JSON data using `JSON.parse`.

Another grading focus is **commenting**. Put a descriptive header (name, course, date, description) at the top of your code and comment each non-trivial section of NodeJS and JS code explaining the purpose of that code.

Format your JS and NodeJS code similarly to the examples from class. Properly use whitespace and indentation.

Please do not place a solution to this assignment online on a publicly accessible web site.