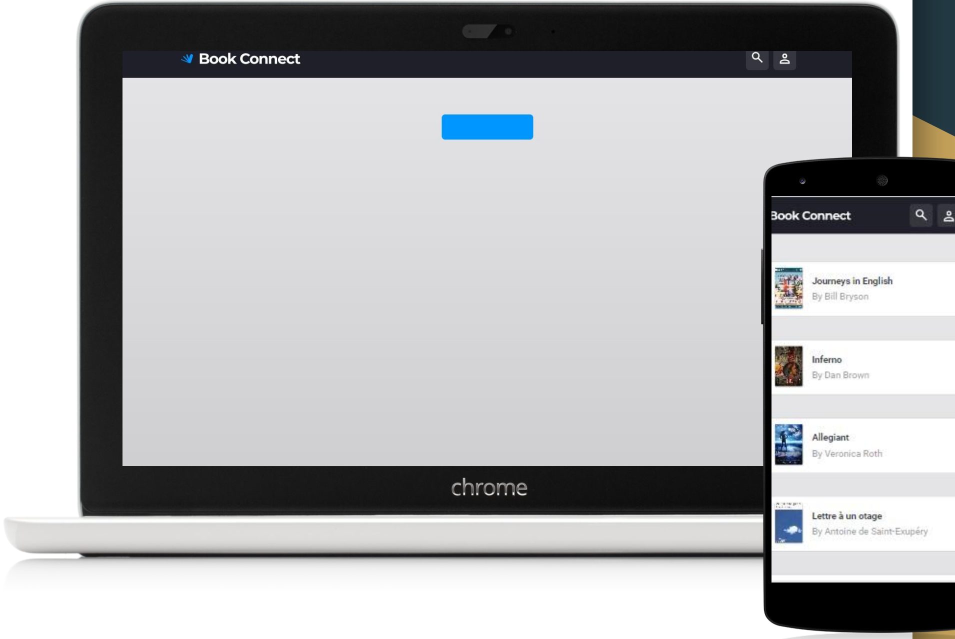
# Elvin Harris

## Final Capstone Project

ELVHAR045_SOZ2301_Elvin_Harris_IWA19



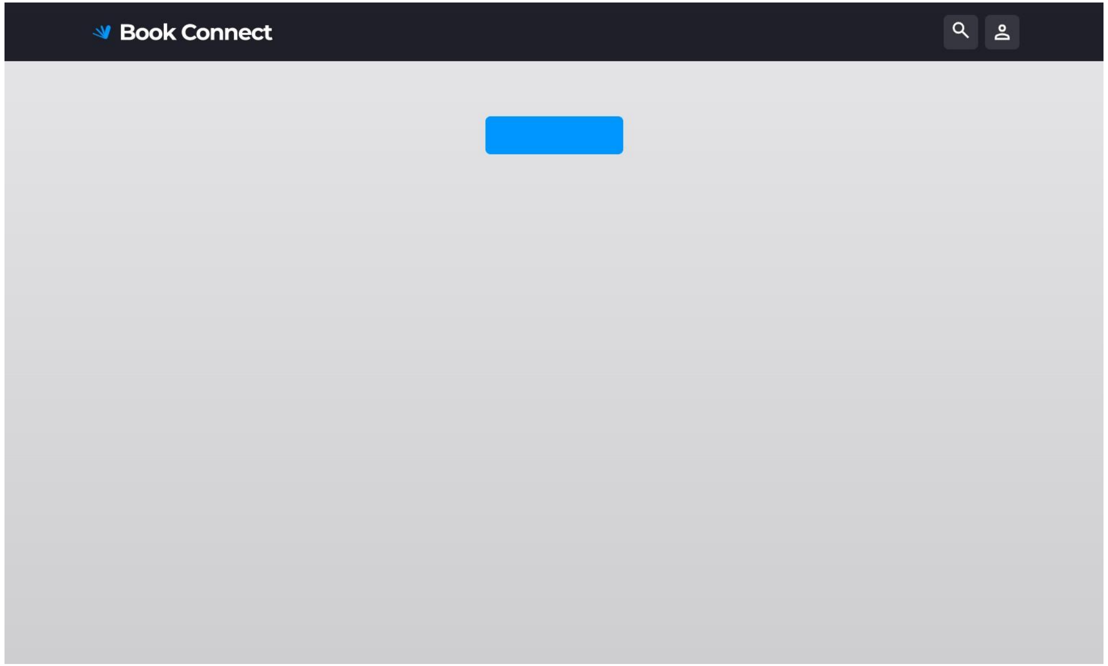
# Requirements

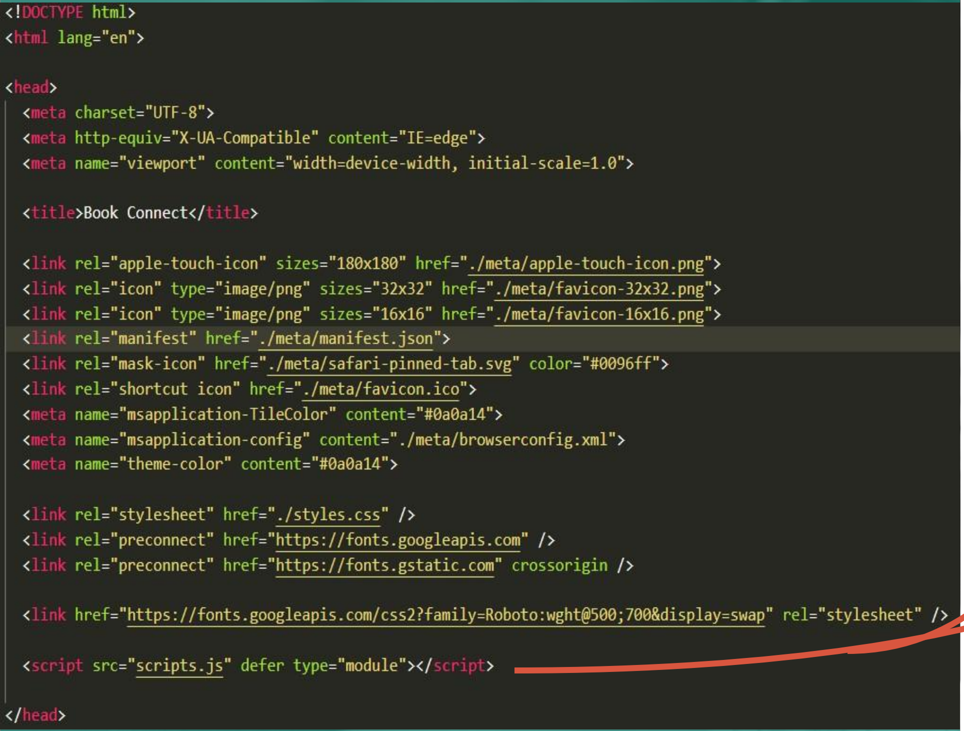
From user:

1. I want to view a list of book previews, by title and author, so that I can discover new books to read.
2. I want an image associated with all book previews so that I can recognize a book by the cover even if I forgot the name.
3. I want to have the option of reading a summary of the book so that I can decide whether I want to read it.
4. I want to have the option of seeing the date that a book was published so that I can determine how easy it is to obtain second-hand.
5. I want to find books based on specific text phrases so that I don’t need to remember the entire title of a book.
6. I want to filter books by author so that I can find books to read by authors that I enjoy.
7. I want to filter books by genre so that I can find books to read in genres that I enjoy.
8. I want to toggle between dark and light modes so that I can use the app comfortably at night.

## The Problem

Website doesn't display correct.





```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Book Connect</title>

  <link rel="apple-touch-icon" sizes="180x180" href="./meta/apple-touch-icon.png">
  <link rel="icon" type="image/png" sizes="32x32" href="./meta/favicon-32x32.png">
  <link rel="icon" type="image/png" sizes="16x16" href="./meta/favicon-16x16.png">
  <link rel="manifest" href="./meta/manifest.json">
  <link rel="mask-icon" href="./meta/safari-pinned-tab.svg" color="#0096ff">
  <link rel="shortcut icon" href="./meta/favicon.ico">
  <meta name="msapplication-TileColor" content="#0a0a14">
  <meta name="msapplication-config" content="./meta/browserconfig.xml">
  <meta name="theme-color" content="#0a0a14">

  <link rel="stylesheet" href="./styles.css" />
  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />

  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@500;700&display=swap" rel="stylesheet" />

  <script src="scripts.js" defer type="module"></script>

</head>
```

I removed the data.js file from the index.html file.

I made my file strict as well by making the ‘type=”module”’ and added defer to make sure my script only execute after my document has loaded.

I exported the data from the data.js file and imported it into the scripts.js file.

```javascript
import { books, authors, genres
} from "./data.js";
```

```javascript
export const BOOKS_PER_PAGE = 36;
export const authors = {
export const genres = {
export const books = [
```

Before:

```
matches = books
page = 1;
```

After:

```
const matches = books;
let page = 1;
let range = books.length;
```

I defined/fixed these variables. I've assigned the value of the "books" variable to the const "matches", which implies that "matches" will now reference the same array as "books".

The "page" variable is set to 1, and the "range" is set to the length of the "books" array.

## Before:

```
day = {
    dark: '10, 10, 20',
    light: '255, 255, 255',
}

night = {
    dark: '255, 255, 255',
    light: '10, 10, 20',
}
```

## After:

```
const day = {
  dark: '10, 10, 20',
  light: '255, 255, 255',
};

const night = {
  dark: '255, 255, 255',
  light: '10, 10, 20',
};
```

This code defines two objects, “day” and “night”, representing color themes for day and night. Each object has two properties: “dark” and “light”, storing the RGB values for dark and light colors in their respective themes.

## Select elements from the DOM

```
const dataSettingsTheme =
document.querySelector('[data-settings-theme]')
const saveButton = document.querySelector("body >
dialog:nth-child(5) > div > div >
button.overlay__button.overlay__button_primary")

saveButton.addEventListener('click', (event) => {
 event.preventDefault()
 if (dataSettingsTheme.value === 'day') {
document.querySelector('body').style.setProperty('--color-dark',
day.dark)

document.querySelector('body').style.setProperty('--color-light',
day.light)
if (typeof appOverlays !== 'undefined') {
appOverlays.settingsOverlay.close()}}
if (dataSettingsTheme.value === 'night') {
document.querySelector('body').style.setProperty('--color-dark',n
ight.dark)
document.querySelector('body').style.setProperty('--color-light',
night.light)
    if (typeof appOverlays !== 'undefined') {
      appOverlays.settingsOverlay.close()}}})
```

This code sets up an event listener for a save button, which is responsible for saving the user's theme selection. It performs the following actions:

- It selects the input element that allows the user to change themes using the "**dataSettingsTheme**" variable.
- It selects the save button using the "**saveButton**" variable.
- It adds an event listener to the save button, listening for a 'click' event.
- When the save button is clicked, it prevents the default behavior of the button (prevents form submission) using "**event.preventDefault()**".
- It checks the value of the `dataSettingsTheme`input element to determine which theme the user has selected.
- If the selected theme is 'day', it sets the dark and light colors for the body element to the values defined in the "**day**" object.
- If the selected theme is 'night', it sets the dark and light colors for the body element to the values defined in the "**night**" object.
- If the "**appOverLays**" object is defined, it closes the settings overlay.

In summary, this code sets up an event listener for a save button that saves the user's theme selection.

## Before:

```
const extracted =
books.slice(0, 36)
```

## After:

```
let startIndex = 0;
let endIndex = 36;
const extracted =
books.slice(startIndex, endIndex)
```

I've added more two variables, "**startIndex**" and "**endIndex**", with the values 0 and 36 respectively.

Then, it uses the **slice()** method on the "**books**" array to extract a portion of the array.

The extracted portion of the array is stored in the "**extracted**" variable, which will contain an array with elements from index 0 to index 35 (since the ending index is exclusive).

In summary, this code selects a portion of the "**books**" array specified by the "startIndex" and "endIndex" variables and stores it in the "extracted" variable.

**Book List**:

```
for (let i = 0; i < extracted.length; i++) {
//For each book, create a new "dl" element called "preview" and assign it
various data attributes based on the book's properties
  const preview = document.createElement('dl')
  preview.className = 'preview'
  preview.dataset.id = books[i].id
  preview.dataset.title = books[i].title
  preview.dataset.image = books[i].image
  preview.dataset.subtitle = `${authors[books[i].author]} (${(new
Date(books[i].published)).getFullYear()})`
  preview.dataset.description = books[i].description
  preview.dataset.genre = books[i].genres
  //Set the "innerHTML" of the "preview" element to a string of HTML code
that includes an image, the book title, and author name
  //Template literal(using backticks)
  preview.innerHTML = /*html*/`
      <div>
      <image class='preview__image' src="${books[i].image}" alt="book
pic"}/>
      </div>
      <div class='preview__info'>
      <dt class='preview__title'>${books[i].title}<dt>
      <dt class='preview__author'> By ${authors[books[i].author]}</dt>
      </div>`
  // add preview element to the document fragment
  //This will display all the book previews.
  fragment.appendChild(preview)
};
```

This code snippet performs a loop over the "**extracted**" array (assuming "**extracted**" is previously defined) using a "**for**" loop.

For each book in the "**extracted**" array, it creates a new `<dl>` element called "preview" using "document.createElement('dl')".

In summary, this code selects a portion of the "**books**" array specified by the "**startIndex**" and "**endIndex**" variables and stores it in the "**extracted**" variable.

## Settings(clicking theme button):

```
const settingButton =
document.querySelector("[data-header-settings]" )
settingButton.addEventListener('click', (event) =>
{

document.querySelector("[data-settings-overlay]" ).s
tyle.display = "block";
})

const settingCancel =
document.querySelector('[data-settings-cancel]' )
settingCancel.addEventListener('click', (event) =>
{

document.querySelector("[data-settings-overlay]" ).s
tyle.display = "none";
})
```

This JavaScript code sets up event listeners for a **settings button** and a **cancel button**. When the settings button is clicked, it displays the settings overlay by setting its CSS “**display**” property to "**block**". Conversely, when the cancel button is clicked, it hides the settings overlay by setting its CSS “**display**” property to "**none**".

**Display book details on preview pop-up and close preview.**

```
const detailsToggle = (event) => {
  const overlay1 = document.querySelector('[data-list-active]');
  const title = document.querySelector('[data-list-title]')
  const subtitle = document.querySelector('[data-list-subtitle]')
  const description = document.querySelector('[data-list-description]')
  const image1 = document.querySelector('[data-list-image]')
  const imageBlur = document.querySelector('[data-list-blur]')
  event.target.dataset.id ? overlay1.style.display = "block" : undefined;
event.target.dataset.description ? description.innerHTML =
event.target.dataset.description : undefined;
  event.target.dataset.subtitle ? subtitle.innerHTML = event.target.dataset.subtitle
: undefined;
  event.target.dataset.title ? title.innerHTML = event.target.dataset.title :
undefined;
  event.target.dataset.image ? image1.setAttribute('src',
event.target.dataset.image) : undefined;
  event.target.dataset.image ? imageBlur.setAttribute('src',
event.target.dataset.image) : undefined;
};

const detailsClose = document.querySelector('[data-list-close]')
detailsClose.addEventListener('click', (event) => {
document.querySelector("[data-list-active]").style.display = "none";
})
```

The "**detailsToggle**" function is triggered by an event and performs various actions based on the event target's data attributes.

It selects different elements from the HTML document, such as the title, subtitle, description, images, and an overlay element. If the event target has a specific data attribute, it updates the corresponding elements with the data value.

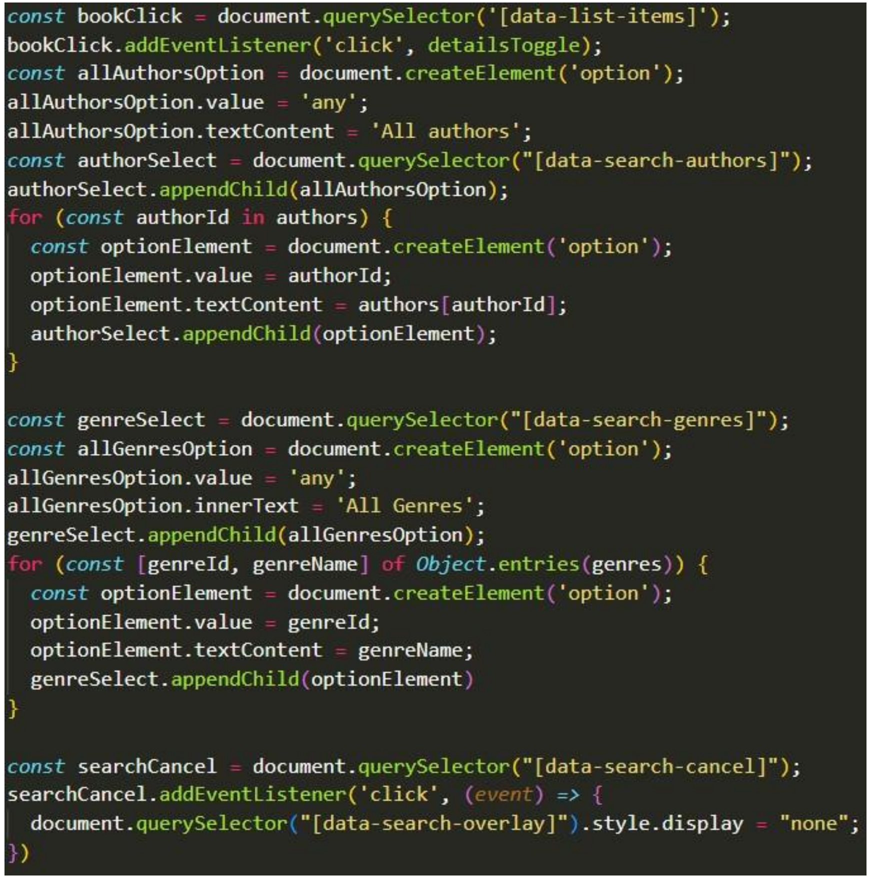
For example, it sets the **display property** of the **overlay element** to "**block**" if the **event target** has a **data-id attribute**.

If the event target does not have the expected data attributes, it does nothing.

The "**detailsClose**" function is triggered by a click event on an element with a specific data attribute (data-list-close). When the event occurs, it selects the overlay element and sets its display property to "none", effectively closing the preview pop-up.

In summary, this code sets up an event listener that triggers the **"detailsToggle"** function when an event occurs. The function updates elements on the page based on the event target's data attributes.

## Search (Authors and genres) and cancel search



```
const bookClick = document.querySelector('[data-list-items]');
bookClick.addEventListener('click', detailsToggle);
const allAuthorsOption = document.createElement('option');
allAuthorsOption.value = 'any';
allAuthorsOption.textContent = 'All authors';
const authorSelect = document.querySelector("[data-search-authors]");
authorSelect.appendChild(allAuthorsOption);
for (const authorId in authors) {
  const optionElement = document.createElement('option');
  optionElement.value = authorId;
  optionElement.textContent = authors[authorId];
  authorSelect.appendChild(optionElement);
}

const genreSelect = document.querySelector("[data-search-genres]");
const allGenresOption = document.createElement('option');
allGenresOption.value = 'any';
allGenresOption.innerText = 'All Genres';
genreSelect.appendChild(allGenresOption);
for (const [genreId, genreName] of Object.entries(genres)) {
  const optionElement = document.createElement('option');
  optionElement.value = genreId;
  optionElement.textContent = genreName;
  genreSelect.appendChild(optionElement)
}

const searchCancel = document.querySelector("[data-search-cancel]");
searchCancel.addEventListener('click', (event) => {
  document.querySelector("[data-search-overlay]").style.display = "none";
})
```

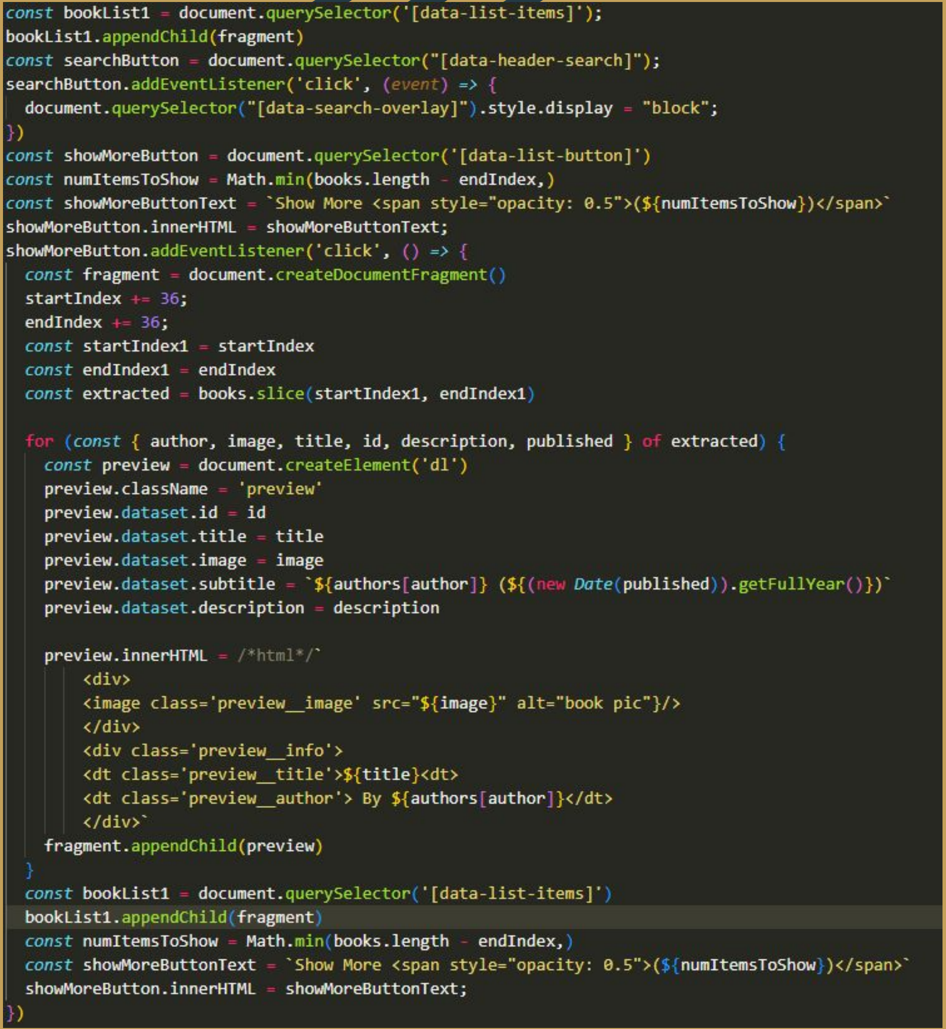
The code selects an element with the data attribute "**data-list-items**" and adds an event listener to it. When this element is clicked, it triggers the "**detailsToggle**" function.

Next, the code creates a new option element, sets its value and text content, and appends it to a select element with the data attribute "data-search-authors". It then iterates over each author in the "**authors**" object, creates option elements for each author, sets their values and text content based on the author's information, and appends them to the select element.

Similarly, the code creates a new option element for genres, sets its value and text content, and appends it to a select element with the data attribute "data-search-genres". It iterates over each genre in the `genres` object, creates option elements for each genre, sets their values and text content, and appends them to the select element.

Finally, the code selects an element with the data attribute "data-search-cancel" and adds an event listener to it. When this element is clicked, it sets the display property of an element with the data attribute "data-search-overlay" to "none", hiding the search overlay.

In summary, this code sets up event listeners for click events and dynamically creates and appends option elements to select elements based on the provided data.



```javascript
const bookList1 = document.querySelector('[data-list-items]');
bookList1.appendChild(fragment)
const searchButton = document.querySelector("[data-header-search]");
searchButton.addEventListener('click', (event) => {
  document.querySelector("[data-search-overlay]").style.display = "block";
})
const showMoreButton = document.querySelector('[data-list-button]')
const numItemsToShow = Math.min(books.length - endIndex,)
const showMoreButtonText = `Show More <span style="opacity: 0.5">(${numItemsToShow})</span>`
showMoreButton.innerHTML = showMoreButtonText;
showMoreButton.addEventListener('click', () => {
  const fragment = document.createDocumentFragment()
  startIndex += 36;
  endIndex += 36;
  const startIndex1 = startIndex
  const endIndex1 = endIndex
  const extracted = books.slice(startIndex1, endIndex1)

  for (const { author, image, title, id, description, published } of extracted) {
    const preview = document.createElement('dl')
    preview.className = 'preview'
    preview.dataset.id = id
    preview.dataset.title = title
    preview.dataset.image = image
    preview.dataset.subtitle = `${authors[author]} (${(new Date(published)).getFullYear()})`
    preview.dataset.description = description

    preview.innerHTML = /*html*/`
        <div>
        <image class='preview__image' src="${image}" alt="book pic"}/>
        </div>
        <div class='preview__info'>
        <dt class='preview__title'>${title}<dt>
        <dt class='preview__author'> By ${authors[author]}</dt>
        </div>`
    fragment.appendChild(preview)
  }
  const bookList1 = document.querySelector('[data-list-items]')
  bookList1.appendChild(fragment)
  const numItemsToShow = Math.min(books.length - endIndex,)
  const showMoreButtonText = `Show More <span style="opacity: 0.5">(${numItemsToShow})</span>`
  showMoreButton.innerHTML = showMoreButtonText;
})
```

This code first appends a "**fragment**" to the HTML element selected with the data attribute "**data-list-items**".

Next, it selects the search button HTML element with the data attribute "data-header-search" and adds an event listener to it. When clicked, it displays the search overlay.

Then, it updates the text of the "Show More" button to indicate how many more items will be displayed. The number of items is calculated based on the length of the "**books**" array and the "**endIndex**".

Lastly, it adds an event listener to the "Show More" button. When clicked, it creates a new "**fragment**" and updates the "**startIndex**" and "**endIndex**" variables. It extracts a subset of books based on the updated indices and generates preview elements for each book. The previews are appended to the "**bookList1**" element. The text of the "**Show More**" button is updated again to reflect the remaining number of items to be shown.

## Thank You!

Elvin Harris