

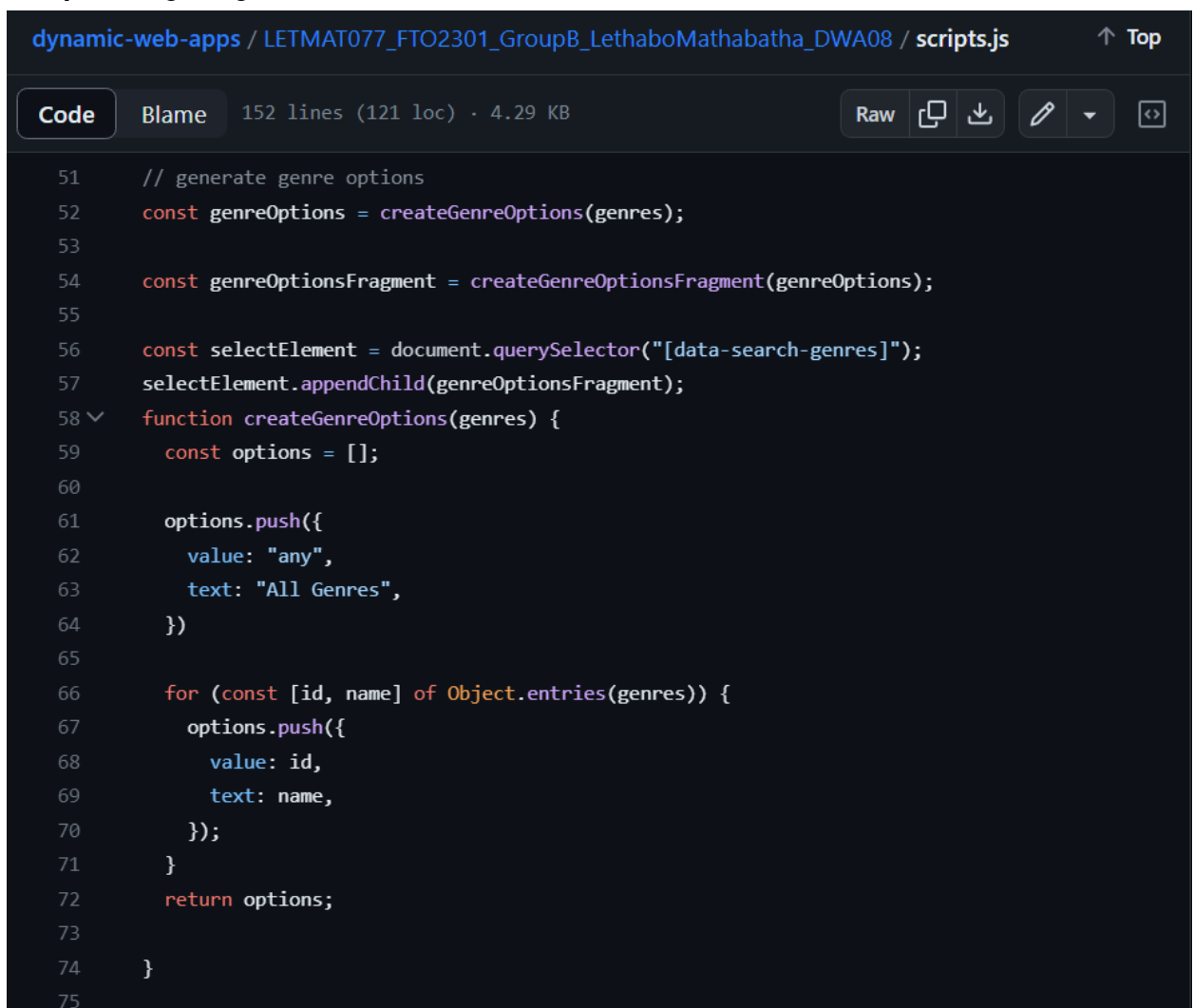
DWA_08 Discussion Questions

In this module you will continue with your “Book Connect” codebase, and further iterate on your abstractions. You will be required to create an encapsulated abstraction of the book preview by means of a single factory function. If you are up for it you can also encapsulate other aspects of the app into their own abstractions.

To prepare for your session with your coach, please answer the following questions. Then download this document as a PDF and include it in the repository with your code.

1. What parts of encapsulating your logic were easy?

- **Encapsulating the genre code**



The screenshot shows a code editor interface with a dark theme. The top bar displays the file path: `dynamic-web-apps / LETMAT077_FTO2301_GroupB_LethaboMathabatha_DWA08 / scripts.js`. Below the path, there are tabs for `Code` and `Blame`, and a status bar indicating `152 lines (121 loc) · 4.29 KB`. To the right of the status bar are icons for `Raw`, `Copy`, `Download`, `Edit`, and `Toggle Syntax Highlighting`. The main area contains JavaScript code for generating genre options. The code starts with a comment `// generate genre options` and defines `genreOptions` using `createGenreOptions(genres)`. It then defines `genreOptionsFragment` using `createGenreOptionsFragment(genreOptions)`. A `selectElement` is selected using `document.querySelector("[data-search-genres]")` and `genreOptionsFragment` is appended to it. A function `createGenreOptions(genres)` is defined, which initializes an `options` array, pushes a default option for "All Genres", and then iterates over the `genres` object to push specific genre options. The function returns the `options` array.

```
51 // generate genre options
52 const genreOptions = createGenreOptions(genres);
53
54 const genreOptionsFragment = createGenreOptionsFragment(genreOptions);
55
56 const selectElement = document.querySelector("[data-search-genres]");
57 selectElement.appendChild(genreOptionsFragment);
58 ✓ function createGenreOptions(genres) {
59     const options = [];
60
61     options.push({
62         value: "any",
63         text: "All Genres",
64     })
65
66     for (const [id, name] of Object.entries(genres)) {
67         options.push({
68             value: id,
69             text: name,
70         });
71     }
72     return options;
73 }
74
75
```

- Encapsulating the author generation code

```
dynamic-web-apps / LETMAT077_FTO2301_GroupB_LethaboMathabatha_DWA08 / scripts.js ↑ Top

Code Blame 152 lines (121 loc) · 4.29 KB Raw Copy Download Edit Dropdown Close

75
76 // generate author options
77 const authorOptions = generateAuthorOptions(authors);
78 const authorOptionsFragment = createAuthorOptionsFragment(authorOptions);
79 document.querySelector("[data-search-authors]").appendChild(authorOptionsFragment);
80
81 /**
82  * Creates author options based on an array of authors and returns an array of option objects
83  * @param {Object} authors - An object with author information.
84  * @return {Array} An array of option objects.
85  */
86 ✓ function generateAuthorOptions(authors) {
87     const options = [];
88
89     options.push({
90         value: "any",
91         text: "All Authors",
92     });
93
94     for (const [id, name] of Object.entries(authors)) {
95         options.push({
96             value: id,
97             text: name,
98         });
99     }
100
101     return options;
102 }
103
```

2. What parts of encapsulating your logic were hard?

```
dynamic-web-apps / LETMAT077_FTO2301_GroupB_LethaboMathabatha_DWA08 / modules / helper.js ↑ Top
Code Blame 305 lines (244 loc) · 8.93 KB Raw Copy Download Edit View Source

7   * @return {DocumentFragment} A document fragment containing a list of buttons representing the starting books.
8   */
9   export function createBookGenerator() {
10    const getStartingBooks = (matches, BOOKS_PER_PAGE, authors) => {
11      const starting = document.createDocumentFragment();
12
13      for (const { author, id, image, title } of matches.slice(0, BOOKS_PER_PAGE)) {
14        const element = document.createElement('button');
15        element.classList = 'preview';
16        element.setAttribute('data-preview', id);
17
18        element.innerHTML = `
19          
23
24          <div class="preview__info">
25            <h3 class="preview__title">${title}</h3>
26            <div class="preview__author">${authors[author]}</div>
27          </div>
28        `;
29
30        starting.appendChild(element);
31      }
32
33      return starting;
34    };
35
36    return {
37      getStartingBooks,
38    };
39  }
```

3. Is abstracting the book preview a good or bad idea? Why?

Abstracting the book preview is a good idea considering these factors that arise from abstraction:

- Code reusability: allows for the creation of reusable components or functions that can be used in multiple places
 - Consistency: ensures that the presentation and behavior of the preview are consistent across the application
 - Maintainability: the separation of logic from the rest of the code makes the codebase more modular and thus easier to maintain
-