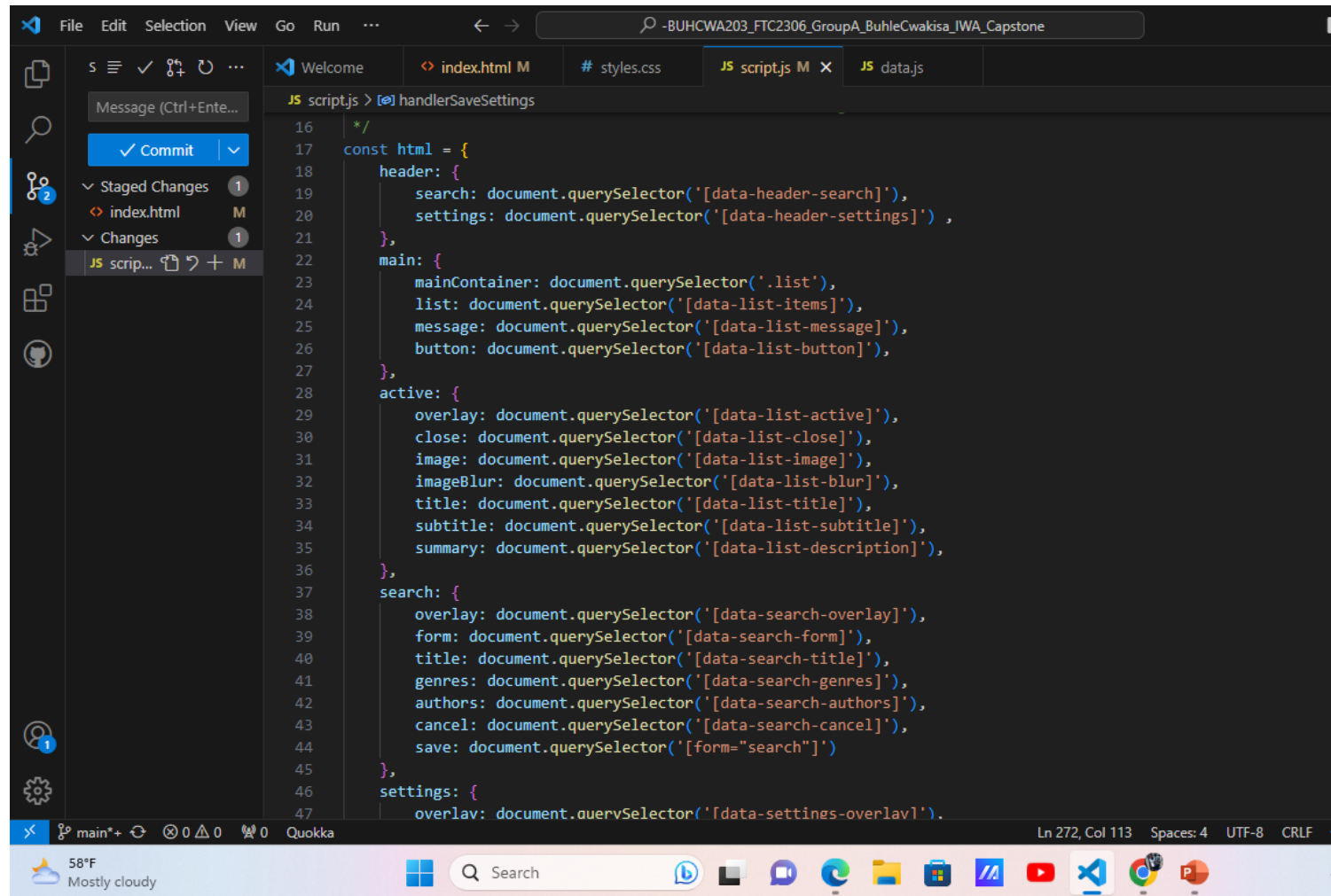




# BOOK CONNECT

IWA 19 CAPSTONE

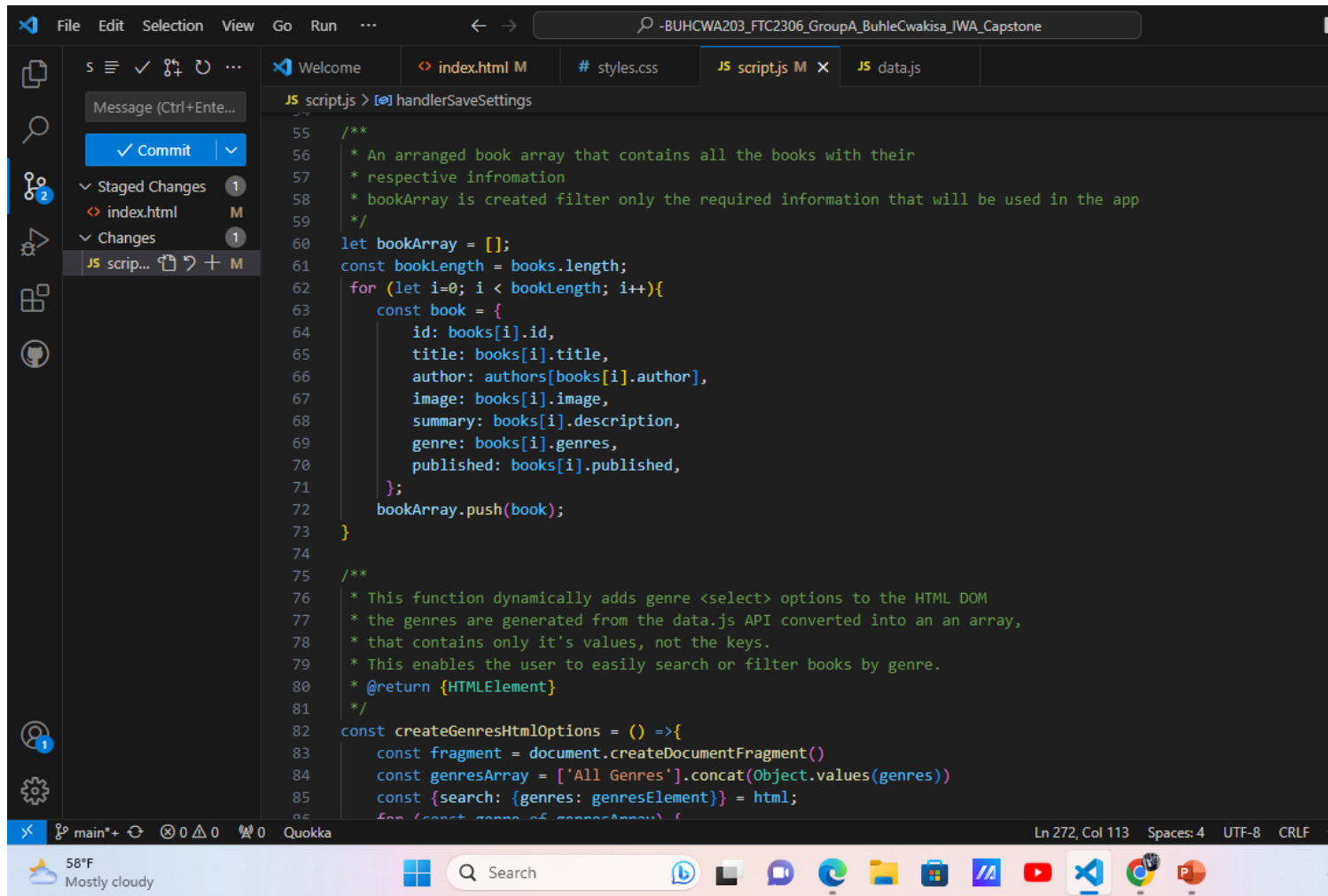




```
16  */
17  const html = {
18    header: {
19      search: document.querySelector('[data-header-search]'),
20      settings: document.querySelector('[data-header-settings]'),
21    },
22    main: {
23      mainContainer: document.querySelector('.list'),
24      list: document.querySelector('[data-list-items]'),
25      message: document.querySelector('[data-list-message]'),
26      button: document.querySelector('[data-list-button]'),
27    },
28    active: {
29      overlay: document.querySelector('[data-list-active]'),
30      close: document.querySelector('[data-list-close]'),
31      image: document.querySelector('[data-list-image]'),
32      imageBlur: document.querySelector('[data-list-blur]'),
33      title: document.querySelector('[data-list-title]'),
34      subtitle: document.querySelector('[data-list-subtitle]'),
35      summary: document.querySelector('[data-list-description]'),
36    },
37    search: {
38      overlay: document.querySelector('[data-search-overlay]'),
39      form: document.querySelector('[data-search-form]'),
40      title: document.querySelector('[data-search-title]'),
41      genres: document.querySelector('[data-search-genres]'),
42      authors: document.querySelector('[data-search-authors]'),
43      cancel: document.querySelector('[data-search-cancel]'),
44      save: document.querySelector('[form="search"]'),
45    },
46    settings: {
47      overlay: document.querySelector('[data-settings-overlay]'),
```

A object literal that contains references to all the HTML elements referenced through the operation of the app either upon initialisation or while its running (via event listeners). This ensure that all UI elements can be accessed and seen in a structured manner in a single data structure.

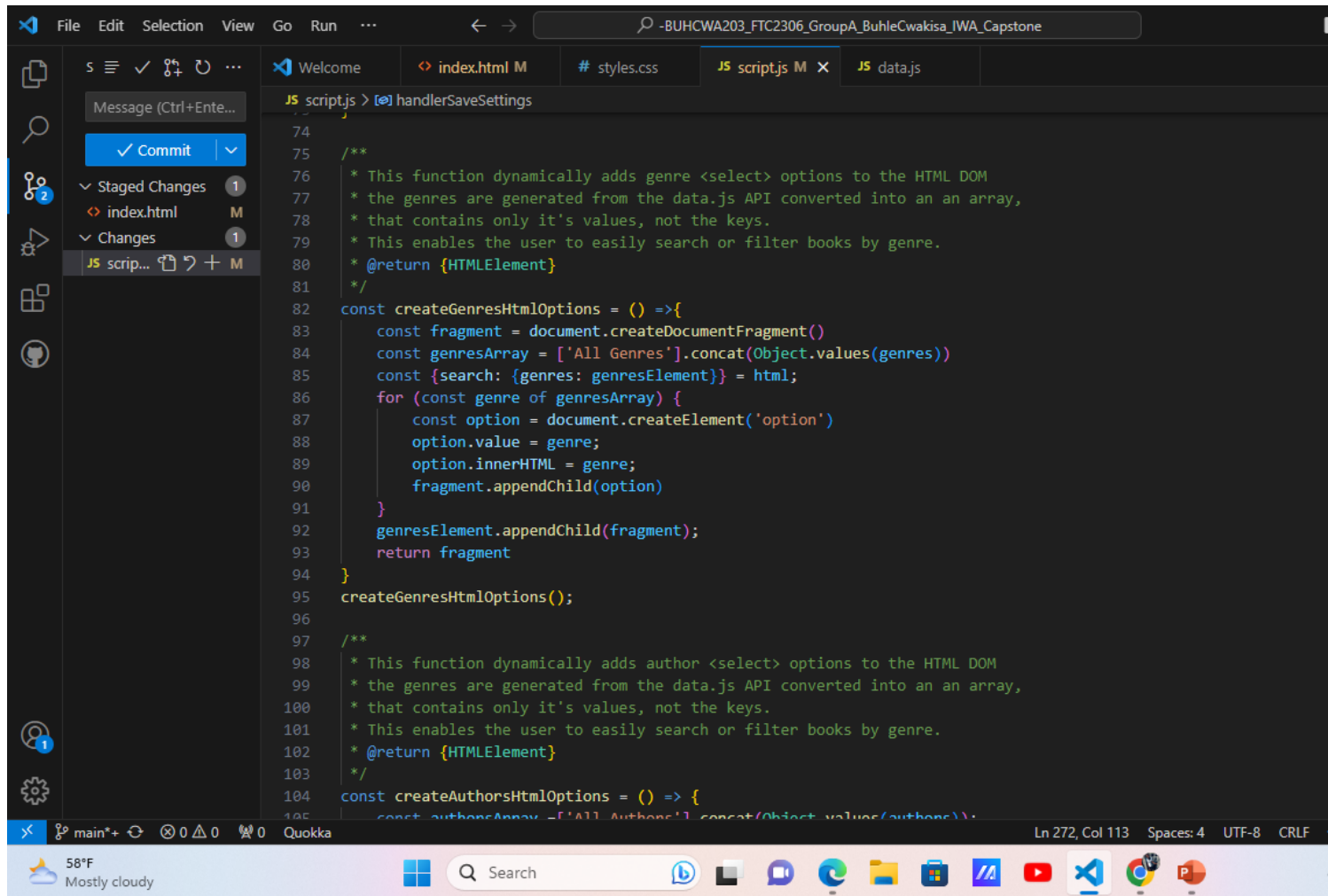
An arranged book array that contains all the books with their respective information bookArray is created filter only the required information that will be used in the app



The screenshot shows a Visual Studio Code editor window with the following details:

- File Explorer (Left):** Shows 'Staged Changes' with 'index.html' and 'Changes' with 'JS scrip...'. A 'Commit' button is visible.
- Editor Tabs:** 'index.html M', '# styles.css', 'JS script.js M', and 'JS data.js'.
- Editor Content:** The 'JS script.js' file is open, showing the 'handlerSaveSettings' function. The code includes comments and logic for creating a 'bookArray' and a 'createGenresHtmlOptions' function.
- Code Snippet:**

```
55 /**
56  * An arranged book array that contains all the books with their
57  * respective information
58  * bookArray is created filter only the required information that will be used in the app
59  */
60 let bookArray = [];
61 const bookLength = books.length;
62 for (let i=0; i < bookLength; i++){
63   const book = {
64     id: books[i].id,
65     title: books[i].title,
66     author: authors[books[i].author],
67     image: books[i].image,
68     summary: books[i].description,
69     genre: books[i].genres,
70     published: books[i].published,
71   };
72   bookArray.push(book);
73 }
74
75 /**
76  * This function dynamically adds genre <select> options to the HTML DOM
77  * the genres are generated from the data.js API converted into an an array,
78  * that contains only it's values, not the keys.
79  * This enables the user to easily search or filter books by genre.
80  * @return {HTMLElement}
81  */
82 const createGenresHtmlOptions = () =>{
83   const fragment = document.createDocumentFragment()
84   const genresArray = ['All Genres'].concat(Object.values(genres))
85   const {search: {genres: genresElement}} = html;
86   for (const genre of genresArray) {
```
- Status Bar (Bottom):** Shows 'main\*+', '0 0 0', 'Quokka', 'Ln 272, Col 113', 'Spaces: 4', 'UTF-8', and 'CRLF'.
- System Tray (Bottom):** Displays weather (58°F, Mostly cloudy), search, and various application icons.

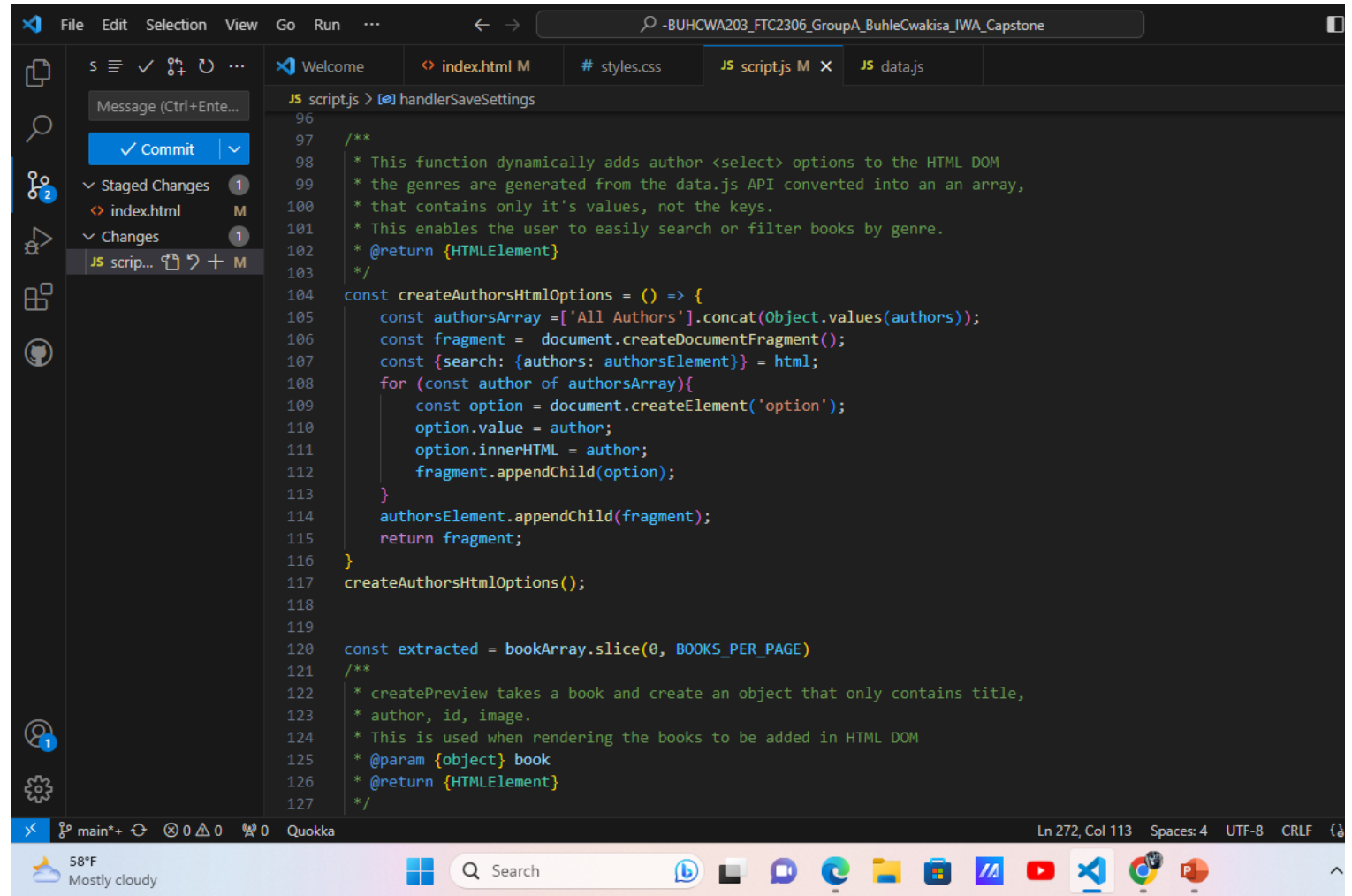


This function dynamically adds genre `<select>` options to the HTML DOM

the genres are generated from the data.js API converted into an array, that contains only it's values, not the keys.

This enables the user to easily search or filter books by genre.

This function dynamically adds author <select> options to the HTML DOM the genres are generated from the data.js API converted into an an array, that contains only it's values, not the keys. This enables the user to easily search or filter books by genre.

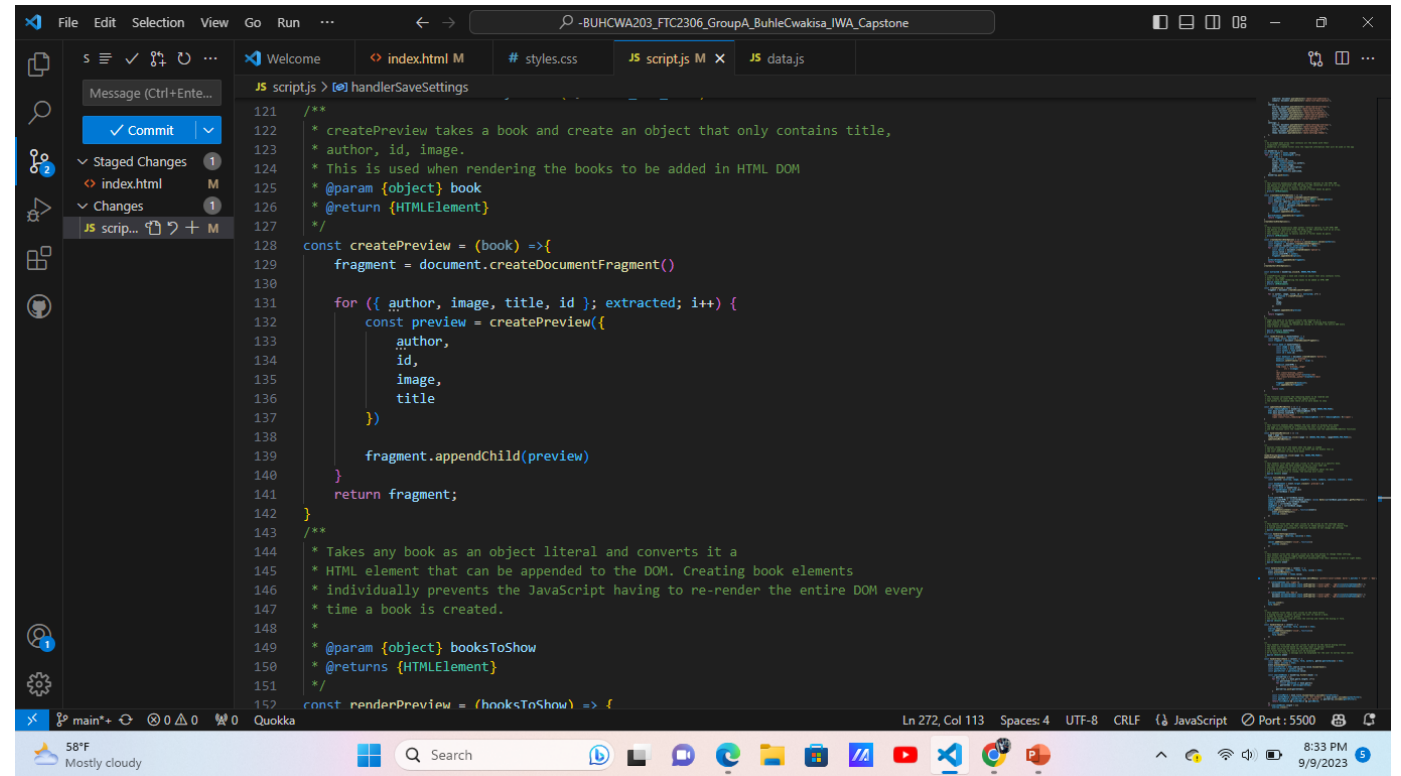


```
File Edit Selection View Go Run ... -BUHCWA203_FTC2306_GroupA_BuhleCwakisa_IWA_Capstone
JS script.js M # styles.css JS script.js M JS data.js
Message (Ctrl+Enter)
Commit
Staged Changes 1
index.html M
Changes 1
JS scrip... + M
96
97 /**
98  * This function dynamically adds author <select> options to the HTML DOM
99  * the genres are generated from the data.js API converted into an an array,
100  * that contains only it's values, not the keys.
101  * This enables the user to easily search or filter books by genre.
102  * @return {HTMLElement}
103  */
104 const createAuthorsHtmlOptions = () => {
105   const authorsArray = ['All Authors'].concat(Object.values(authors));
106   const fragment = document.createDocumentFragment();
107   const {search: {authors: authorsElement}} = html;
108   for (const author of authorsArray){
109     const option = document.createElement('option');
110     option.value = author;
111     option.innerHTML = author;
112     fragment.appendChild(option);
113   }
114   authorsElement.appendChild(fragment);
115   return fragment;
116 }
117 createAuthorsHtmlOptions();
118
119
120 const extracted = bookArray.slice(0, BOOKS_PER_PAGE)
121 /**
122  * createPreview takes a book and create an object that only contains title,
123  * author, id, image.
124  * This is used when rendering the books to be added in HTML DOM
125  * @param {object} book
126  * @return {HTMLElement}
127  */
```

Ln 272, Col 113 Spaces: 4 UTF-8 CRLF

58°F Mostly cloudy Search

the createPreview  
takes a book and  
create an object  
that only contains  
title,  
author, id, image.  
This is used when  
rendering the  
books to be added  
in HTML DOM

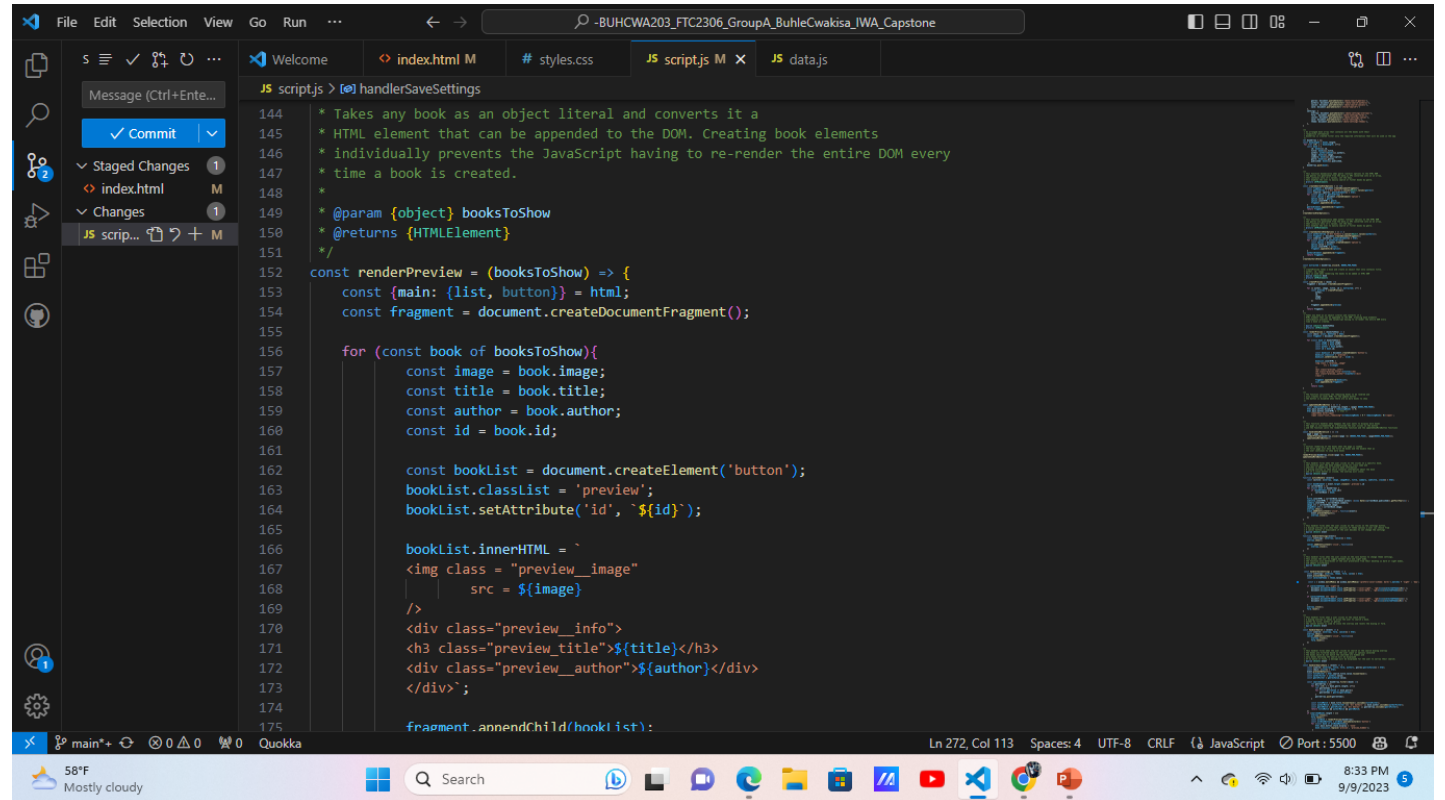


The screenshot shows a Visual Studio Code editor window with the following details:

- File Explorer (Left):** Shows a project structure with 'index.html' and 'JS script.js' under 'Staged Changes'.
- Editor (Center):** Displays the 'handlerSaveSettings' file in 'JS script.js'. The code defines a `createPreview` function that takes a `book` object and returns an `HTMLFragment`. It uses `document.createDocumentFragment()` and `fragment.appendChild()` to build the preview. A `renderPreview` function is also partially visible at the bottom.
- Code Snippets:**

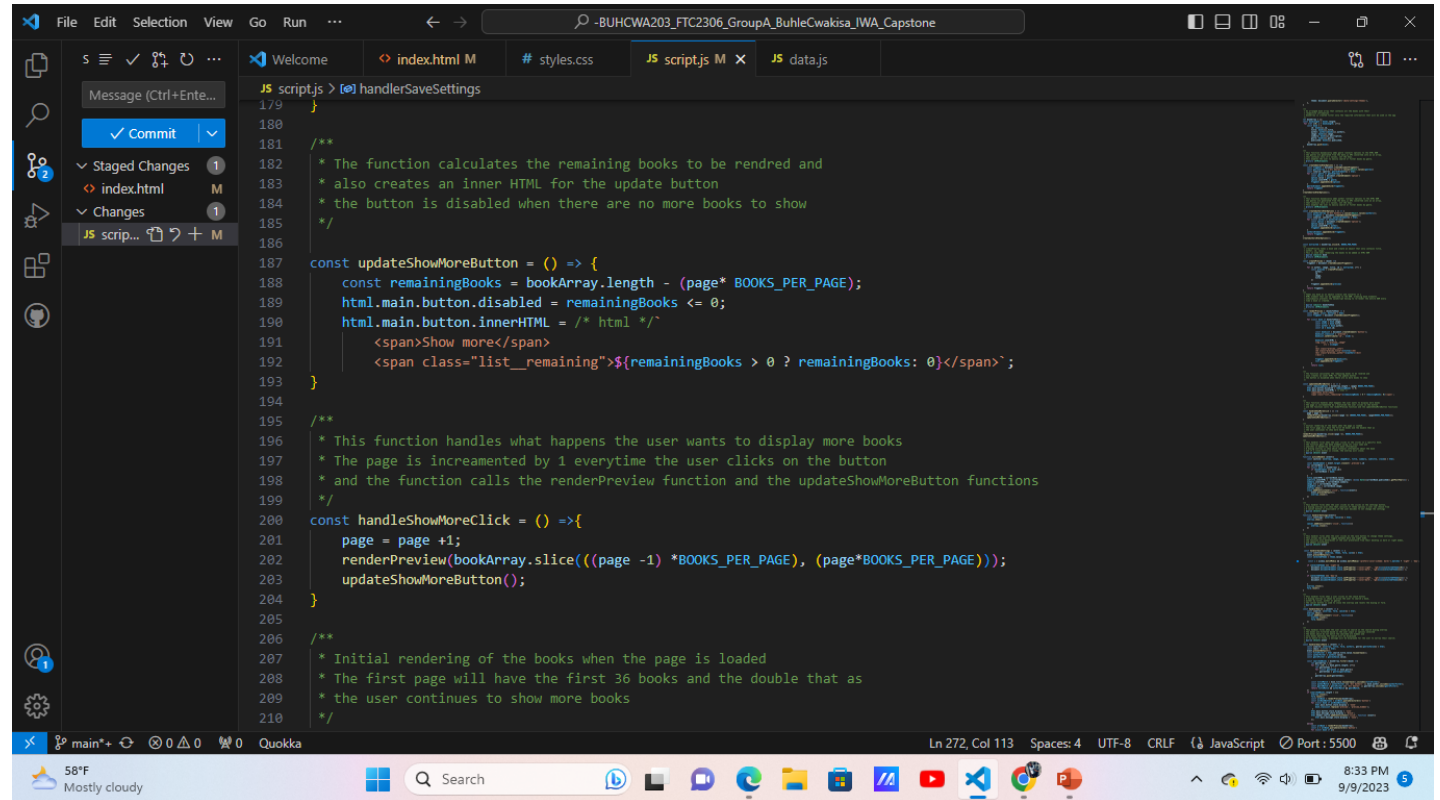
```
121 /**
122  * createPreview takes a book and create an object that only contains title,
123  * author, id, image.
124  * This is used when rendering the books to be added in HTML DOM
125  * @param {object} book
126  * @return {HTMLFragment}
127  */
128 const createPreview = (book) =>{
129   fragment = document.createDocumentFragment()
130
131   for ({ author, image, title, id }; extracted; i++) {
132     const preview = createPreview({
133       author,
134       id,
135       image,
136       title
137     })
138     fragment.appendChild(preview)
139   }
140   return fragment;
141 }
142 /**
143  * Takes any book as an object literal and converts it a
144  * HTML element that can be appended to the DOM. Creating book elements
145  * individually prevents the JavaScript having to re-render the entire DOM every
146  * time a book is created.
147  *
148  * @param {object} booksToShow
149  * @returns {HTMLFragment}
150  */
151 const renderPreview = (booksToShow) => {
```
- Bottom Bar:** Shows the status bar with 'Ln 272, Col 113', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', and 'Port: 5500'. The system tray at the bottom indicates '58°F Mostly cloudy' and the time '8:33 PM 9/9/2023'.

Takes any book as an object literal and converts it a HTML element that can be appended to the DOM. Creating book elements individually prevents the JavaScript having to re-render the entire DOM every time a book is created.



```
144 * Takes any book as an object literal and converts it a
145 * HTML element that can be appended to the DOM. Creating book elements
146 * individually prevents the JavaScript having to re-render the entire DOM every
147 * time a book is created.
148 *
149 * @param {object} booksToShow
150 * @returns {HTMLElement}
151 */
152 const renderPreview = (booksToShow) => {
153   const {main: {list, button}} = html;
154   const fragment = document.createDocumentFragment();
155
156   for (const book of booksToShow){
157     const image = book.image;
158     const title = book.title;
159     const author = book.author;
160     const id = book.id;
161
162     const bookList = document.createElement('button');
163     bookList.classList = 'preview';
164     bookList.setAttribute('id', `${id}`);
165
166     bookList.innerHTML = `
167     <img class = "preview_image"
168       src = ${image}
169     />
170     <div class="preview_info">
171       <h3 class="preview_title">${title}</h3>
172       <div class="preview_author">${author}</div>
173     </div>`;
174
175     fragment.appendChild(bookList);
```

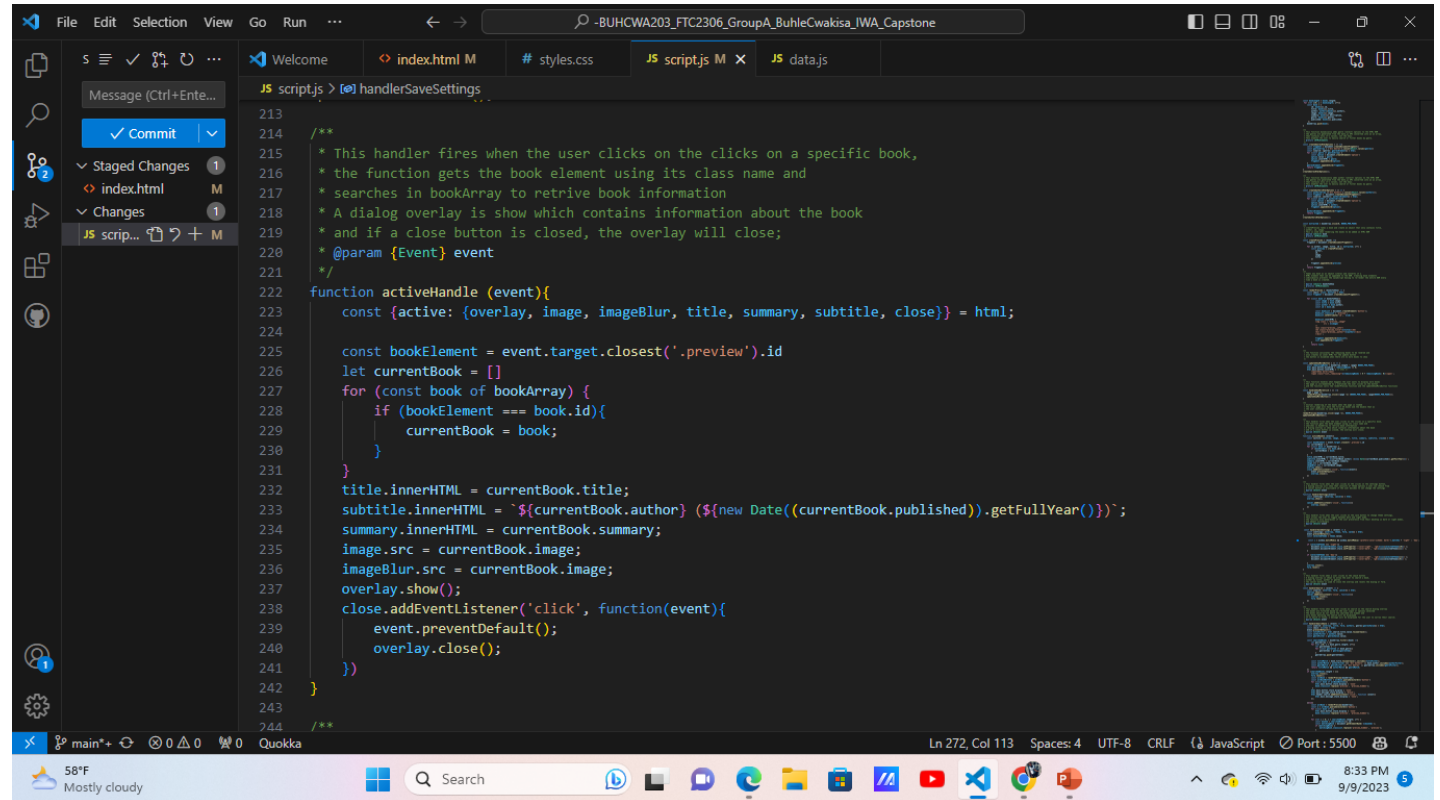
The function calculates the remaining books to be rendered and also creates an inner HTML for the update button the button is disabled when there are no more books to show



```
JS script.js > handlerSaveSettings
179 }
180
181 /**
182  * The function calculates the remaining books to be rendered and
183  * also creates an inner HTML for the update button
184  * the button is disabled when there are no more books to show
185  */
186
187 const updateShowMoreButton = () => {
188   const remainingBooks = bookArray.length - (page * BOOKS_PER_PAGE);
189   html.main.button.disabled = remainingBooks <= 0;
190   html.main.button.innerHTML = /* html */
191     <span>Show more</span>
192     <span class="list_remaining">${remainingBooks > 0 ? remainingBooks: 0}</span>;
193 }
194
195 /**
196  * This function handles what happens the user wants to display more books
197  * The page is incremented by 1 everytime the user clicks on the button
198  * and the function calls the renderPreview function and the updateShowMoreButton functions
199  */
200 const handleShowMoreClick = () =>{
201   page = page +1;
202   renderPreview(bookArray.slice(((page -1) *BOOKS_PER_PAGE), (page*BOOKS_PER_PAGE)));
203   updateShowMoreButton();
204 }
205
206 /**
207  * Initial rendering of the books when the page is loaded
208  * The first page will have the first 36 books and the double that as
209  * the user continues to show more books
210  */
```

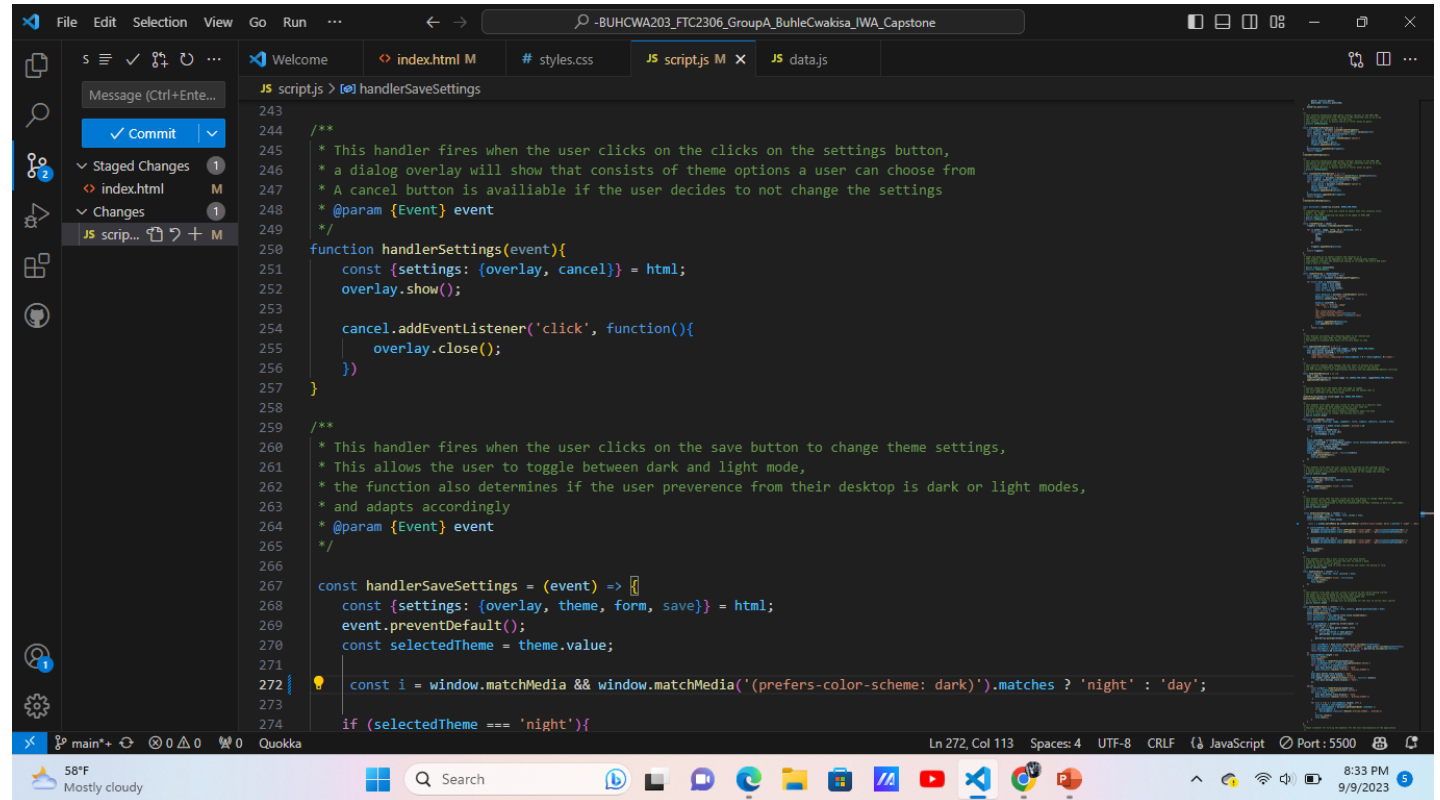


This handler fires when the user clicks on the clicks on a specific book,  
the function gets the book element using its class name and searches in bookArray to retrieve book information  
A dialog overlay is show which contains information about the book  
and if a close button is closed, the overlay will close



```
213
214
215  /**
216   * This handler fires when the user clicks on the clicks on a specific book,
217   * the function gets the book element using its class name and
218   * searches in bookArray to retrieve book information
219   * A dialog overlay is show which contains information about the book
220   * and if a close button is closed, the overlay will close;
221   * @param {Event} event
222   */
223  function activeHandle (event){
224    const {active: {overlay, image, imageBlur, title, summary, subtitle, close}} = html;
225
226    const bookElement = event.target.closest('.preview').id
227    let currentBook = []
228    for (const book of bookArray) {
229      if (bookElement === book.id){
230        currentBook = book;
231      }
232    }
233    title.innerHTML = currentBook.title;
234    subtitle.innerHTML = `${currentBook.author} (${new Date((currentBook.published)).getFullYear()})`;
235    summary.innerHTML = currentBook.summary;
236    image.src = currentBook.image;
237    imageBlur.src = currentBook.image;
238    overlay.show();
239    close.addEventListener('click', function(event){
240      event.preventDefault();
241      overlay.close();
242    })
243  }
244  /**
```

This handler fires when the user clicks on the clicks on the settings button, a dialog overlay will show that consists of theme options a user can choose from  
A cancel button is available if the user decides to not change the settings

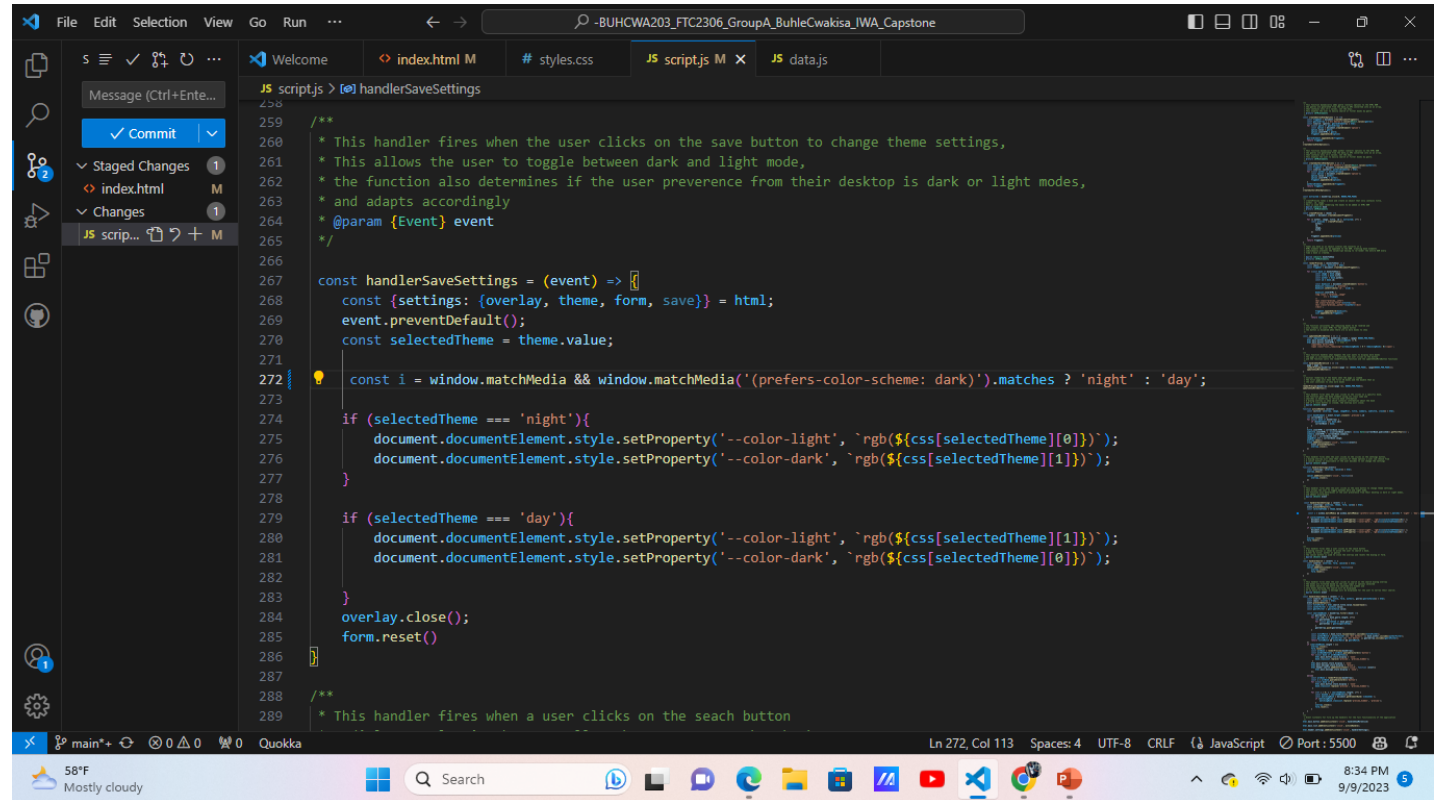


The screenshot shows a VS Code editor window with the following details:

- File Explorer (Left):** Shows a project structure with 'index.html' and 'JS script.js' under 'Staged Changes' and 'Changes'.
- Editor (Center):** Displays the 'handlerSaveSettings' function in 'script.js'. The code includes comments and logic for handling theme settings, including a cancel button and a save button.
- Code Snippets:**

```
243  
244  
245 /**  
246  * This handler fires when the user clicks on the clicks on the settings button,  
247  * a dialog overlay will show that consists of theme options a user can choose from  
248  * A cancel button is available if the user decides to not change the settings  
249  * @param {Event} event  
250  */  
251 function handlerSettings(event){  
252   const {settings: {overlay, cancel}} = html;  
253   overlay.show();  
254  
255   cancel.addEventListener('click', function(){  
256     overlay.close();  
257   })  
258 }  
259  
260 /**  
261  * This handler fires when the user clicks on the save button to change theme settings,  
262  * This allows the user to toggle between dark and light mode,  
263  * the function also determines if the user preference from their desktop is dark or light modes,  
264  * and adapts accordingly  
265  * @param {Event} event  
266  */  
267 const handlerSaveSettings = (event) => {  
268   const {settings: {overlay, theme, form, save}} = html;  
269   event.preventDefault();  
270   const selectedTheme = theme.value;  
271  
272   const i = window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches ? 'night' : 'day';  
273  
274   if (selectedTheme === 'night'){
```
- Status Bar (Bottom):** Shows 'Ln 272, Col 113', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', and 'Port: 5500'.

This handler fires when the user clicks on the save button to change theme settings, This allows the user to toggle between dark and light mode, the function also determines if the user preference from their desktop is dark or light modes, and adapts accordingly

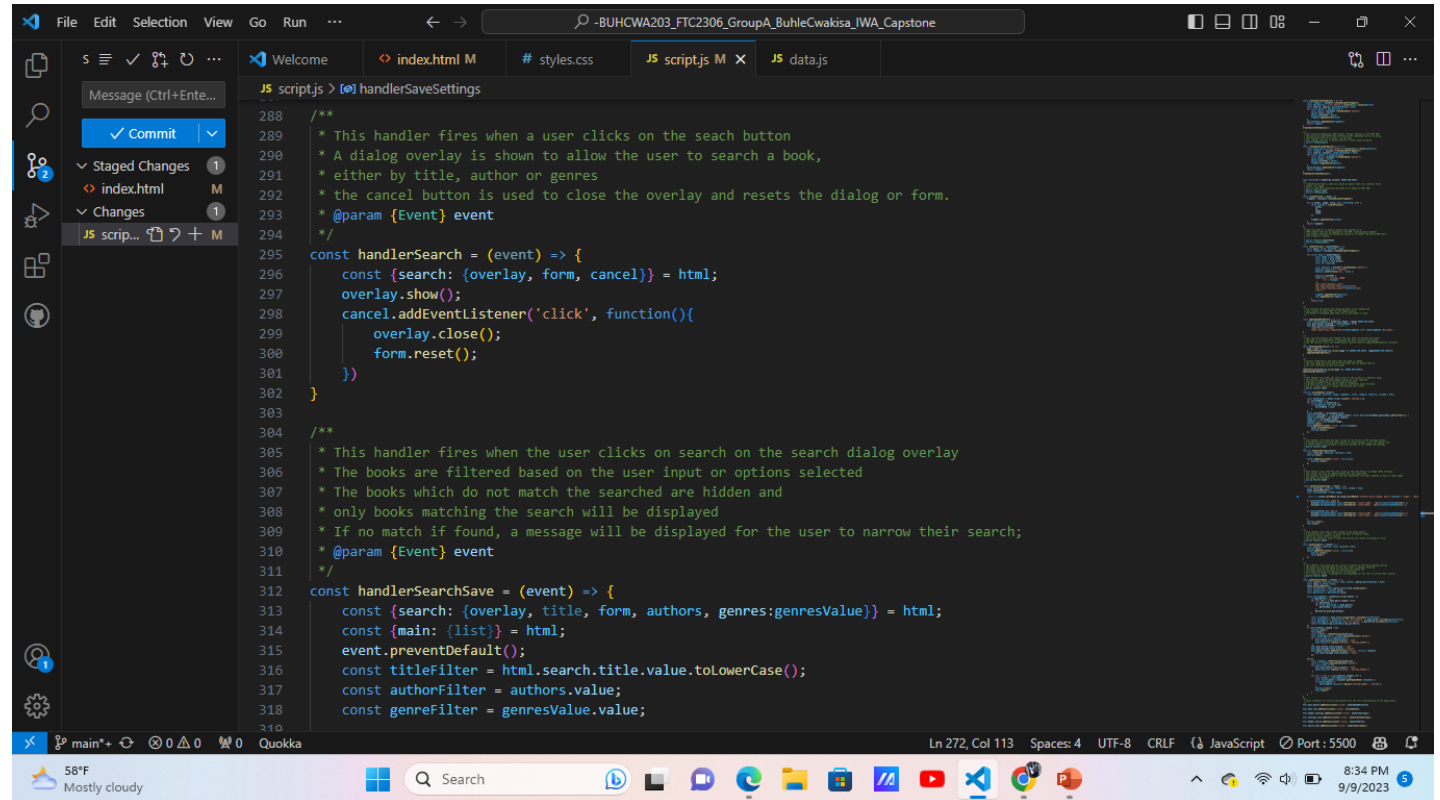


The screenshot shows a Visual Studio Code editor window with the following details:

- File Explorer (Left):** Shows a project structure with 'index.html' and 'script.js' under a 'JS' folder.
- Source Control (Left):** Shows 'Staged Changes' and 'Changes' for 'index.html' and 'script.js'.
- Editor (Center):** Displays the 'script.js' file with the following code:

```
259 /**  
260  * This handler fires when the user clicks on the save button to change theme settings,  
261  * This allows the user to toggle between dark and light mode,  
262  * the function also determines if the user preference from their desktop is dark or light modes,  
263  * and adapts accordingly  
264  * @param {Event} event  
265  */  
266  
267 const handlerSaveSettings = (event) => {  
268   const {settings: {overlay, theme, form, save}} = html;  
269   event.preventDefault();  
270   const selectedTheme = theme.value;  
271  
272   const i = window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches ? 'night' : 'day';  
273  
274   if (selectedTheme === 'night'){  
275     document.documentElement.style.setProperty('--color-light', `rgb(${css[selectedTheme][0]})`);  
276     document.documentElement.style.setProperty('--color-dark', `rgb(${css[selectedTheme][1]})`);  
277   }  
278  
279   if (selectedTheme === 'day'){  
280     document.documentElement.style.setProperty('--color-light', `rgb(${css[selectedTheme][1]})`);  
281     document.documentElement.style.setProperty('--color-dark', `rgb(${css[selectedTheme][0]})`);  
282   }  
283   overlay.close();  
284   form.reset();  
285 }  
286  
287 /**  
288  * This handler fires when a user clicks on the search button  
289  */
```
- Bottom Bar:** Shows the status bar with 'Ln 272, Col 113', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', and 'Port: 5500'.

This handler fires when a user clicks on the search button  
A dialog overlay is shown to allow the user to search a book,  
either by title, author or genres  
the cancel button is used to close the overlay and resets the  
dialog or form.



The screenshot shows a VS Code editor window with the following details:

- File Explorer (Left):** Shows a project structure with files like `index.html`, `styles.css`, `script.js`, and `data.js`. The `script.js` file is selected.
- Editor (Center):** Displays the `script.js` file. The code is as follows:

```
288 /**
289  * This handler fires when a user clicks on the search button
290  * A dialog overlay is shown to allow the user to search a book,
291  * either by title, author or genres
292  * the cancel button is used to close the overlay and resets the dialog or form.
293  * @param {Event} event
294  */
295 const handlerSearch = (event) => {
296   const {search: {overlay, form, cancel}} = html;
297   overlay.show();
298   cancel.addEventListener('click', function(){
299     overlay.close();
300     form.reset();
301   })
302 }
303
304 /**
305  * This handler fires when the user clicks on search on the search dialog overlay
306  * The books are filtered based on the user input or options selected
307  * The books which do not match the searched are hidden and
308  * only books matching the search will be displayed
309  * If no match is found, a message will be displayed for the user to narrow their search;
310  * @param {Event} event
311  */
312 const handlerSearchSave = (event) => {
313   const {search: {overlay, title, form, authors, genres:genresValue}} = html;
314   const {main: {list}} = html;
315   event.preventDefault();
316   const titleFilter = html.search.title.value.toLowerCase();
317   const authorFilter = authors.value;
318   const genreFilter = genresValue.value;
```
- Output Console (Bottom):** Shows the status bar with "Ln 272, Col 113", "Spaces: 4", "UTF-8", "CRLF", "JavaScript", and "Port: 5500".

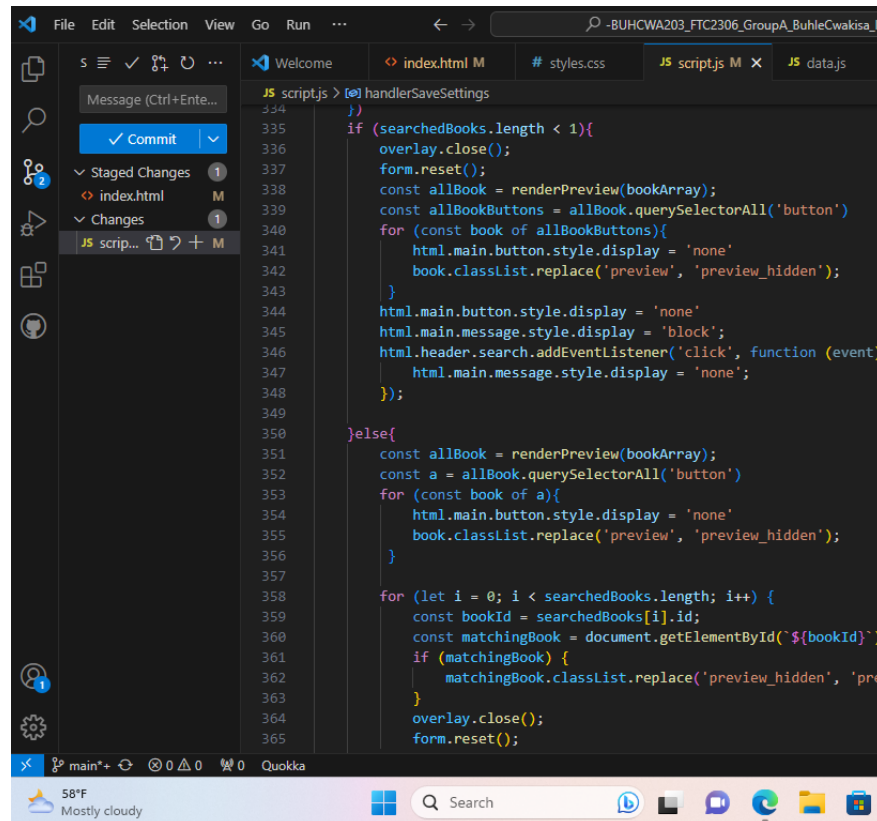
This handler fires when the user clicks on search on the search dialog overlay

The books are filtered based on the user input or options selected

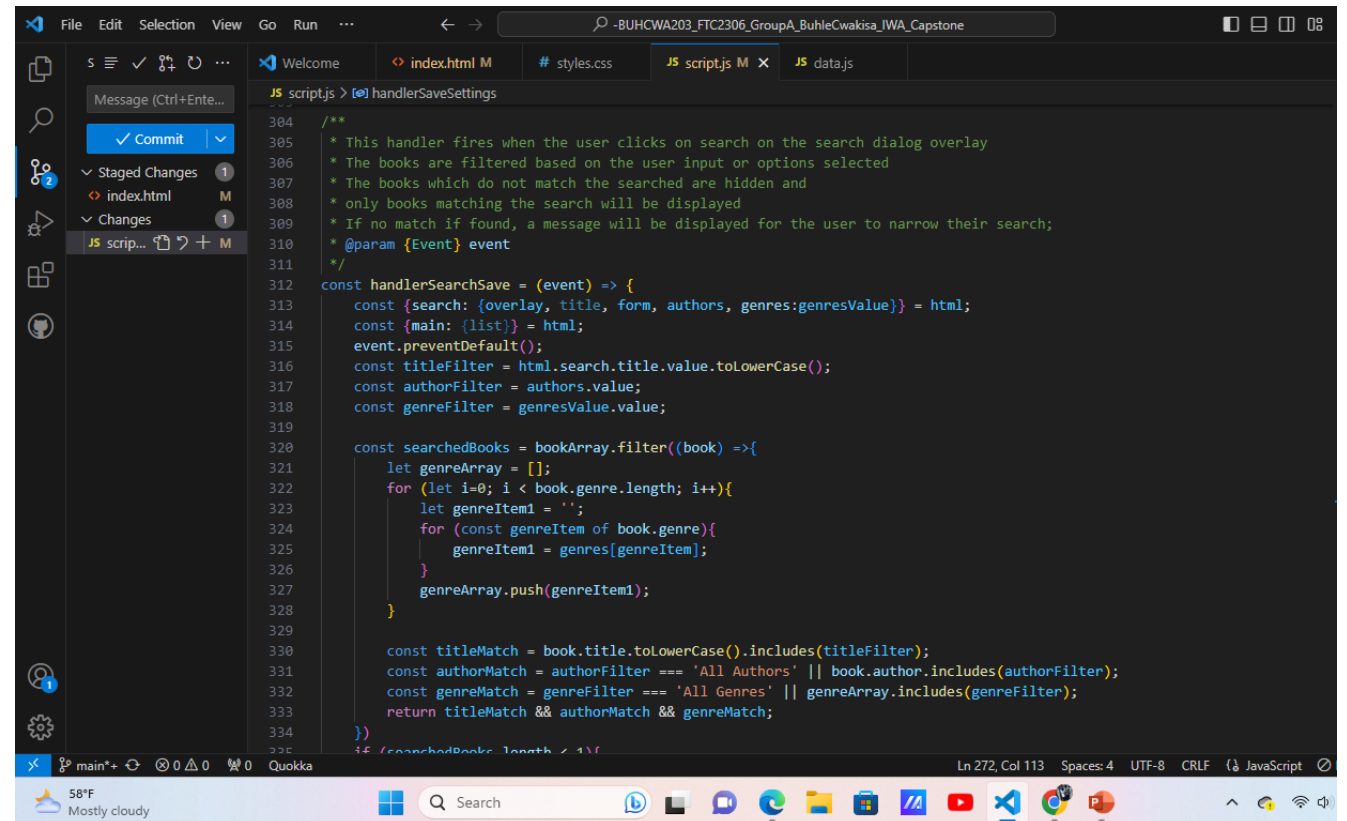
The books which do not match the searched are hidden and

only books matching the search will be displayed

If no match is found, a message will be displayed for the user to narrow their search;

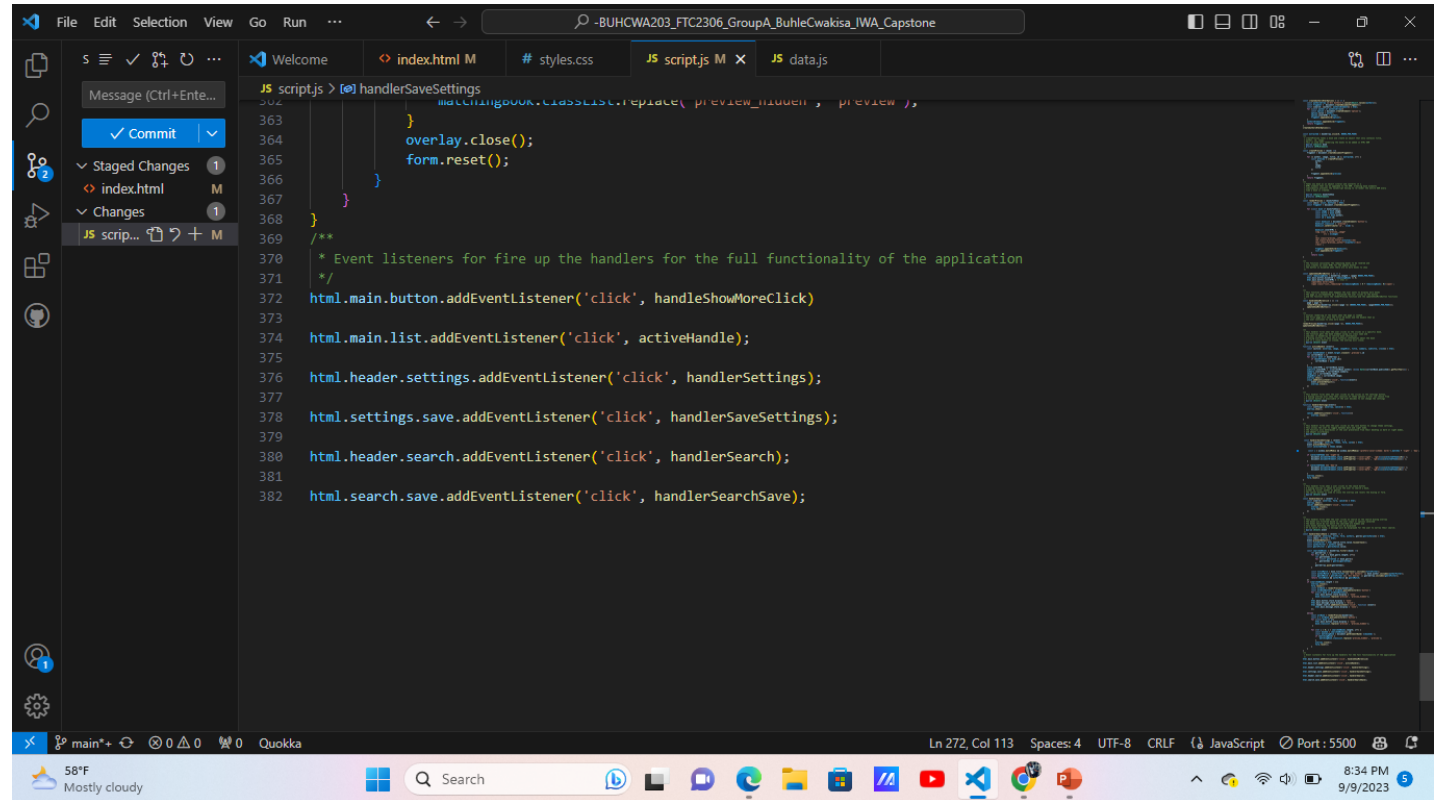


```
334 }
335 if (searchedBooks.length < 1){
336   overlay.close();
337   form.reset();
338   const allBook = renderPreview(bookArray);
339   const allBookButtons = allBook.querySelectorAll('button')
340   for (const book of allBookButtons){
341     html.main.button.style.display = 'none'
342     book.classList.replace('preview', 'preview_hidden');
343   }
344   html.main.button.style.display = 'none'
345   html.main.message.style.display = 'block';
346   html.header.search.addEventListener('click', function (event) {
347     html.main.message.style.display = 'none';
348   });
349 }else{
350   const allBook = renderPreview(bookArray);
351   const a = allBook.querySelectorAll('button')
352   for (const book of a){
353     html.main.button.style.display = 'none'
354     book.classList.replace('preview', 'preview_hidden');
355   }
356   for (let i = 0; i < searchedBooks.length; i++) {
357     const bookId = searchedBooks[i].id;
358     const matchingBook = document.getElementById(`${bookId}`);
359     if (matchingBook) {
360       matchingBook.classList.replace('preview_hidden', 'pre
361     }
362     overlay.close();
363     form.reset();
364   }
365 }
```



```
304 /**
305  * This handler fires when the user clicks on search on the search dialog overlay
306  * The books are filtered based on the user input or options selected
307  * The books which do not match the searched are hidden and
308  * only books matching the search will be displayed
309  * If no match is found, a message will be displayed for the user to narrow their search;
310  * @param {Event} event
311  */
312 const handlerSearchSave = (event) => {
313   const {search: {overlay, title, form, authors, genres:genresValue}} = html;
314   const {main: {list}} = html;
315   event.preventDefault();
316   const titleFilter = html.search.title.value.toLowerCase();
317   const authorFilter = authors.value;
318   const genreFilter = genresValue.value;
319
320   const searchedBooks = bookArray.filter((book) =>{
321     let genreArray = [];
322     for (let i=0; i < book.genre.length; i++){
323       let genreItem1 = '';
324       for (const genreItem of book.genre){
325         genreItem1 = genres[genreItem];
326       }
327       genreArray.push(genreItem1);
328     }
329
330     const titleMatch = book.title.toLowerCase().includes(titleFilter);
331     const authorMatch = authorFilter === 'All Authors' || book.author.includes(authorFilter);
332     const genreMatch = genreFilter === 'All Genres' || genreArray.includes(genreFilter);
333     return titleMatch && authorMatch && genreMatch;
334   })
335   if (searchedBooks.length < 1){
```

Event listeners  
to fire up the  
handlers for the  
full functionality  
of the  
application



The screenshot shows a Visual Studio Code editor window with a dark theme. The file explorer on the left shows a project structure with files like index.html, styles.css, script.js, and data.js. The main editor area displays a JavaScript file (script.js) with the following code:

```
362
363
364     }
365     overlay.close();
366     form.reset();
367 }
368
369 /**
370  * Event listeners for fire up the handlers for the full functionality of the application
371  */
372 html.main.button.addEventListener('click', handleShowMoreClick)
373
374 html.main.list.addEventListener('click', activeHandle);
375
376 html.header.settings.addEventListener('click', handlerSettings);
377
378 html.settings.save.addEventListener('click', handlerSaveSettings);
379
380 html.header.search.addEventListener('click', handlerSearch);
381
382 html.search.save.addEventListener('click', handlerSearchSave);
```

The status bar at the bottom indicates the current file is 'main.js', the cursor is at line 272, column 113, and the file is encoded in UTF-8 with CRLF line endings. The system tray at the bottom shows the date and time as 8:34 PM on 9/9/2023.