AOBSEL894-BCL2302-Chanique

IWA-19 Final Capstone Presentation

As a Junior software Developer the first thing that caught my attention had to be the syntax errors

Followed by

The main reasons as to why the code was not running as it should.(main problems)

Here are all the problems i identified

1. In line 1, matches is assigned the value of books, but books is not defined anywhere in the code
2. In lines 2-3, page is initialized with a value of 1, but i is later used in the code without being defined first.
3. In lines 4-5, there are two if statements that throw errors if certain conditions are not met, but the variables used in these statements (books and range) are not defined anywhere in the code.
4. In lines 6-13, the day and night variables are defined as objects, but they contain string values that should be arrays of numbers.
5. In lines 14-25, the extracted variable is assigned a slice of books, but books is not defined anywhere in the code, and the for loop that follows uses a destructuring syntax that is not valid.

Continuation…

1. In lines 26-41, the genres variable is initialized as a document fragment, but the subsequent code that creates and appends option elements to it is incomplete and contains several errors.
2. In lines 42-56, the authors variable is initialized as a document fragment, but the subsequent code that creates and appends option elements to it is incomplete and contains several errors.
3. In lines 57-59, the data-settings-theme variable is used in an expression that compares it to the result of window.matchMedia, but it is not clear what data-settings-theme represents or what its value is.
4. In lines 60-63, the v variable is assigned a value based on the result of window.matchMedia, but the syntax used to assign this value (? 'night' | 'day') is not valid.
5. In lines 64-65, documentElement is used, but it is not defined anywhere in the code.

There were certainly a lot of problems…

1. In lines 66-74, the `data-list-button` variable is assigned a string value, but it should be assigned a reference to a DOM element, and the subsequent code that sets its `disabled` and `innerHTML` properties contains several errors.
2. In lines 75-76, the `data-search-cancel` and `data-settings-cancel` variables are used in expressions that check their `open` properties, but it is not clear what these properties represent or how they are set.
3. In lines 77-82, the `data-settings-form` variable is used in an expression that invokes a `submit` action, but it is not clear what this action does or what the `actions.settings.submit` value represents.
4. In lines 83-98, the `data-list-button` variable is used in an expression that attaches a `click` event listener, but the subsequent code that is executed when the button is clicked contains several errors and is incomplete.

Tough times never last…

1. In lines 99-103, the data-header-search variable is used in an expression that attaches a click event listener, but the subsequent code that is executed when the button is clicked contains several errors and is incomplete.
2. In lines 104-133, the data-search-form variable is used in an expression that attaches a click event listener, but the subsequent code that is executed when the form is submitted contains several errors and is incomplete.
3. In lines 134-140, the data-settings-overlay variable is used in an expression that attaches a submit event listener, but the subsequent code that is executed when the form is submitted contains several

Now, after going through the problems i thought of the solutions, here is what i am recommending we do on order to get our code running and meet the user’s expectations

What do I recommend?

I say let’s just meet the user’s expectations…

As a user, I want to view a list of book previews, by title and author, so that I can discover new books to read.

1. As a user, 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.
2. As a user, I want to have the option of reading a summary of the book so that I can decide whether I want to read it.
3. As a user, 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.
4. As a user, I want to find books based on specific text phrases so that I don’t need to remember the entire title of a book.
5. As a user, I want to filter books by author so that I can find books to read by authors that I enjoy.
6. As a user, I want to filter books by genre so that I can find books to read in genres that I enjoy.
7. As a user, I want to toggle between dark and light modes so that I can use the app comfortably at night.

On second thoughts, let’s get technical.

```
BOOKS_PER_PAGE = 36;
> authors = {···
}

> genres = {···
}

> books = [···
]
```

```
export const BOOKS_PER_PAGE = 36;
> export const authors = {···
}

> export const genres = {···
}

> export const books = [···
]
```

firstly..

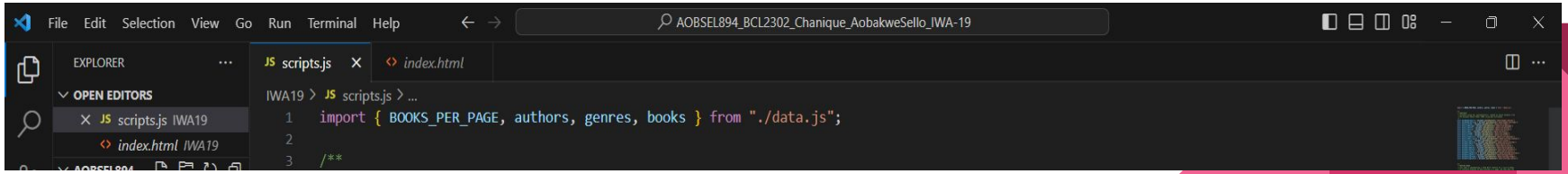
Let us declare the variables using a global constant, then export the objects and the array that consist of a nested object inside to the scripts.js module

why? - Because we need them to run the code in scripts.js

Secondly

Import the objects and arrays in the scripts.js

why? - Because we want to make them possible in scripts.js module



Let us define all the constants

```
/**
 * CONSTANTS
 * Constants using the `querySelector()` method to select elements from
 * the Document Object Model (DOM) using data attributes
 */
const dataHeaderSearch = document.querySelector('[data-header-search]')
const dataHeaderSettings = document.querySelector('[data-header-settings]')
const dataListItems = document.querySelector('[data-list-items]')
const dataListMessage = document.querySelector('[data-list-message]')
const dataListButton = document.querySelector('[data-list-button]')
const dataListActive = document.querySelector('[data-list-active]')
const dataListBlur = document.querySelector('[data-list-blur]')
const dataListImage = document.querySelector('[data-list-image]')
const dataListTitle = document.querySelector('[data-list-title]')
const dataListSubtitle = document.querySelector('[data-list-subtitle]')
const dataListDescription = document.querySelector('[data-list-description]')
const dataListClose = document.querySelector('[data-list-close]')
const dataSearchOverlay = document.querySelector('[data-search-overlay]')
const dataSearchForm = document.querySelector('[data-search-form]')
const dataSearchTitle = document.querySelector('[data-search-title]')
const dataSearchGenres = document.querySelector('[data-search-genres]')
const dataSearchAuthors = document.querySelector('[data-search-authors]')
const dataSearchCancel = document.querySelector('[data-search-cancel]')
const dataSettingsOverlay = document.querySelector('[data-settings-overlay]')
const dataSettingsForm = document.querySelector('[data-settings-form]')
const dataSettingsTheme = document.querySelector('[data-settings-theme]')
const dataSettingsCancel = document.querySelector('[data-settings-cancel]')
```

This code initializes a set of constants by using the `querySelector()` method to select elements from the Document Object Model (DOM) of a web page. These constants are identified by their corresponding data attributes in square brackets (`[data-attribute]`). The constants represent various elements on the web page, such as search bar, settings button, list items, message, button, active list, blurred list, list image, list title, list subtitle, list description, close button, search overlay, search form, search title, search genres, search authors, settings overlay, settings form, settings theme, and settings cancel button.

By selecting these elements using their data attributes, the code can access and manipulate them in the JavaScript code. For example, the code can add event listeners to the search button and settings button, or change the style or content of the search form or settings form based on user interaction.

What the user wants the user gets.

```
/**
 * PREVIEW BOOKS
 * This code is implementing a "Show More" feature for a list of books.
 * It initially displays 36 book previews on a page, and then when the
 * user clicks "Show More" button, it displays the next 36 books
 */

let matches = books
let page = 1;     //used to keep track of current page of book reviews being displayed
const range = [0, BOOKS_PER_PAGE]

if (!books && !Array.isArray(books)) {
    throw new Error('Source required')
}

if (!range && range.length === 2) {
    throw new Error('Range must be an array with two numbers')
}
```

File Edit Selection View Go Run Terminal Help

AOBSEL894_BCL2302_Chanique_AobakweSello_IWA-19

EXPLORER

OPEN EDITORS

scripts.js IWA19 M

index.html IWA19

AOBSEL894_BCL2302_CHANIQ...

.vscode

IWA19

meta

.gitignore

data.js

index.html

scripts.js M

styles.css

README.md

OUTLINE

TIMELINE

scripts.js M | index.html

IWA19 > scripts.js > dataListButton.addEventListener('click') callback > newBookFragment

```
}

/**
 * The createPreview() function takes a book preview object and returns
 * a button element containing the book preview information in HTML form
 */
function createPreview(preview) {
    const { author: authorId, id, image, title } = preview

    const showPreview = document.createElement('button')
    showPreview.classList = 'preview'
    showPreview.setAttribute('data-preview', id)

    showPreview.innerHTML = /* html */ `
        <img
            class="preview__image"
            src="${image}"
        />

        <div class="preview__info">
            <h3 class="preview__title">${title}</h3>
            <div class="preview__author">${authors[authorId]}</div>
        </div>
    `

    return showPreview
}

const bookFragment = document.createDocumentFragment()

const startIndex = (page - 1) * BOOKS_PER_PAGE
const endIndex = startIndex + BOOKS_PER_PAGE

const bookExtracted = books.slice(startIndex, endIndex)

/**
 * This loop iterates over the book previews to display on the current page,
 * creates a book preview button using the createPreview function, and
```

main* ⊗ 0 ⚠ 0 Quokka

Ln 106, Col 62 Spaces: 2 UTF-8 LF {} JavaScript Port : 5501

17°C Partly sunny

Search

ENG US

10:04 2023/05/08

Explaining the code

This code is implementing a "Show More" feature for a list of books. It displays 36 book previews on a page and when the user clicks the "Show More" button, it displays the next 36 books.

The code initializes the matches array to the books array passed in. It also sets the page variable to 1 and range to a constant array [0, BOOKS_PER_PAGE], which represents the range of indices to be displayed on the current page.

The code then checks if books is a valid array, and if range is an array with two numbers; it throws an error if either condition is not met.

The createPreview() function takes a book preview object and returns a button element containing the book preview information in HTML form. It uses destructuring assignment to extract the author ID, image, title, and other information from the preview object, and then creates a button element with the extracted information.

The code creates a document fragment to hold the book previews, and then calculates the start and end indices for the books to be displayed on the current page. It extracts the books within that range using the slice() method, and assigns them to the bookExtracted variable.

The createPreview() function is then called on each book preview object in bookExtracted, and the resulting button elements are appended to the document fragment.

Finally, the document fragment is added to the DOM to display the book previews on the page. When the user clicks the "Show More" button, the page variable is incremented, and the process is repeated to display the next set of book previews.

File Edit Selection View Go Run Terminal Help

AOBSEL894_BCL2302_Chanique_AobakweSello_IWA-19

EXPLORER

OPEN EDITORS

- scripts.js IWA19 M
- index.html IWA19

AOBSEL894_BCL2302_CHANIQ...

- .vscode
- IWA19
  - meta
  - .gitignore
  - data.js
  - index.html
  - scripts.js M
  - styles.css
- README.md

OUTLINE

TIMELINE

scripts.js M | index.html

IWA19 > scripts.js > dataListButton.addEventListener('click') callback

```javascript
/**
 * This loop iterates over the book previews to display on the current page,
 * creates a book preview button using the createPreview function, and
 * appends the button to the bookFragment container
 */
for (const preview of bookExtracted) {
    const showPreview = createPreview(preview)
    bookFragment.appendChild(showPreview)
}

dataListItems.appendChild(bookFragment)

/**
 * This sets up a click event listener for the "Show More" button. When clicked,
 * the code executes the logic to display the next set of book previews.
 */
dataListButton.addEventListener('click', () => {
    page++;

    const newStartIndex = (page - 1) * BOOKS_PER_PAGE
    const newEndIndex = newStartIndex + BOOKS_PER_PAGE

    const newBookExtracted = books.slice(newStartIndex, newEndIndex)

    const newBookFragment = document.createDocumentFragment()

    for (const preview of newBookExtracted) {
        const showPreview = createPreview(preview)
        newBookFragment.appendChild(showPreview)
    }

    dataListItems.appendChild(newBookFragment);

    const remaining = matches.length - page * BOOKS_PER_PAGE;
    dataListButton.innerHTML = /* HTML */ `
      <span>Show more</span>
      <span class="list__remaining">(${remaining > 0 ? remaining : 0})</span>
```

main* | 0 0 | Quokka | Ln 117, Col 29 | Spaces: 2 | UTF-8 | LF | {} JavaScript | Port : 5501

17°C Partly sunny | Search | ENG US | 10:09 2023/05/08

What was happening in the previous code

This code sets up a click event listener for a "Show More" button and defines the logic for displaying the next set of book previews.

When the button is clicked, the code increments the page number by one and calculates the start and end index of the next set of books to display. It then extracts the books within that range from the `books` array and creates a document fragment containing the previews for those books.

The code then appends the fragment to the list of book previews and updates the text of the "Show More" button to display the remaining number of books left to be displayed. If there are no books remaining to be displayed, the button is disabled.

The `BOOKS_PER_PAGE` constant is used to determine the number of books to display per page, and `matches.length` is used to determine the total number of books available for display.

The code also contains an initial assignment to `dataListButton.innerHTML` which sets the text of the "Show More" button before it is clicked for the first time. This initial text is the same as the text that is updated with the remaining number of books when the button is clicked.

```
/**
 * BOOK SUMMARY
 */



// When dataListItems is clicked, it shows a modal by invoking showModal() on dataListActive.
dataListItems.addEventListener('click', (event) => {
    dataListActive.showModal()
    let pathArray = Array.from(event.path || event.composedPath())
    let active;

    for (const node of pathArray) {
      if (active) break;
      const id = node?.dataset?.preview

      for (const singleBook of books) {
        if (singleBook.id === id) {
          active = singleBook
          break;
        }
      }
    }

    if (!active) return;
    dataListImage.src = active.image;
    dataListBlur.src = active.image;
    dataListTitle.textContent = active.title;
    dataListSubtitle.textContent = `${authors[active.author]} (${new Date(active.published).getFullYear()})`
    dataListDescription.textContent = active.description;
})


//When dataListClose is clicked, it closes the modal by invoking close() on dataListActive.
dataListClose.addEventListener('click', () => {
    dataListActive.close()
})
```

IWA19 > JS scripts.js > ...

```
})

//When dataSearchCancel is clicked, it closes modal by invoking close() on dataSearch
dataSearchCancel.addEventListener('click', () => {
    dataSearchOverlay.close()
})

const genresFragment = document.createDocumentFragment()
const genreElement = document.createElement('option')
genreElement.value = 'any'
genreElement.innerText = 'All Genres'
genresFragment.appendChild(genreElement)

for (const [id] of Object.entries(genres)) {
    const genreElement = document.createElement('option')
    genreElement.value = id
    genreElement.innerText = genres[id]
    genresFragment.appendChild(genreElement)
}

dataSearchGenres.appendChild(genresFragment)

const authorsFragment = document.createDocumentFragment()
const authorsElement = document.createElement('option')
authorsElement.value = 'any'
authorsElement.innerText = 'All Authors'
authorsFragment.appendChild(authorsElement)

for (const [id] of Object.entries(authors)) {
    const authorsElement = document.createElement('option')
    authorsElement.value = id
    authorsElement.innerText = authors[id]
    authorsFragment.appendChild(authorsElement)
}

dataSearchAuthors.appendChild(authorsFragment)
```

Search

Explaining the code…

The first block of code listens for a click event on a dataListItems element. When this event is triggered, it displays a modal by invoking the showModal() function on a dataListActive element. It then retrieves the data-preview attribute from the clicked element and uses it to find the corresponding book object from an array of book objects called "books". Once it finds the book object, it updates the content of the modal with information about the book, including the book's image, title, author, publication date, and description.

The second block of code creates dropdown menus for genres and authors. It creates an option element for each genre and author in an object containing genre and author information, and appends these option elements to a document fragment. The fragment is then appended to the appropriate dropdown menu element in the HTML. The "any" option is also included at the beginning of each dropdown menu, allowing users to select all genres or authors.

The final block of code listens for click events on a dataHeaderSearch element. When this event is triggered, it displays a modal by invoking the showModal() function on a dataSearchOverlay element. The modal contains a search form for users to input search terms, select a genre and/or author, and submit the search. The dataSearchCancel element allows users to close the search modal by invoking the close() function on the dataSearchOverlay element.

File Edit Selection View Go Run Terminal Help

AOBSEL894_BCL2302_Chanique_AobakweSello_IWA-19

EXPLORER

OPEN EDITORS

- scripts.js IWA19 M
- README.md

AOBSEL894_BCL2302_CHANIQ...

- .vscode
- IWA19
  - meta
  - .gitignore
  - data.js
  - index.html
  - scripts.js M
  - styles.css
  - README.md

OUTLINE

TIMELINE

scripts.js M | README.md

IWA19 > scripts.js > ...

```
    }

    /**
     * If the search returns results, they are displayed using a function called createPreviewsFragment,
     * and the result count is displayed in a button. If there are no results and the user has provided
     * search criteria, a message is displayed indicating that there were no results.
     */
    if (result.length > 0) {
      const resultFragment = createPreview(result);
      dataListItems.replaceChildren(resultFragment);
      dataListButton.innerHTML = /* html */ `
        <span>Show more</span>
        <span class="list__remaining"> (0)</span>
      `;
      dataListButton.disabled = true;
    //   showPreview();
    } else if (filters.title.trim() && filters.author !== 'any' && filters.genre !== 'any') {
      const firstElementChild = dataListMessage;
      dataListItems.innerHTML = '';
      dataListItems.replaceChildren(firstElementChild);
      dataListMessage.style.display = 'block';
      dataListButton.innerHTML = /* html */ `
        <span>Show more</span>
        <span class="list__remaining"> (0)</span>
      `;
      dataListButton.disabled = true;
    }

    dataSearchOverlay.close();
    dataSearchForm.reset();
    window.scrollTo({ top: 0, behavior: 'smooth' });
})
```

main* ⊗ 0 ⚠ 0 Quokka

Ln 279, Col 3 Spaces: 2 UTF-8 LF {} JavaScript Port : 5501

19°C Mostly cloudy

Search

ENG US

10:32 2023/05/08

File Edit Selection View Go Run Terminal Help

AOBSEL894_BCL2302_Chanique_AobakweSello_IWA-19

EXPLORER

OPEN EDITORS

- scripts.js IWA19 M
- *README.md*

AOBSEL894_BCL2302_CHANIQ...

- .vscode
- IWA19
  - meta
  - .gitignore
  - data.js
  - index.html
  - scripts.js M
  - styles.css
- README.md

OUTLINE

TIMELINE

scripts.js M | *README.md*

IWA19 > scripts.js > ...

```javascript

dataSettingsCancel.addEventListener('click', () => {
    dataSettingsOverlay.close()
})

//The css object defines two themes, 'day' and 'night'
const css = {
    day : ['255, 255, 255', '10, 10, 20'],
    night: ['10, 10, 20', '255, 255, 255']
}

//The value of the dataSettingsTheme input is determined based on whether the user's preferred color scheme is dark or not.
dataSettingsTheme.value = window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches ? 'night' : 'day'
let v = window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches ? 'night' : 'day'

//
//  This code sets up for a submit event listener. When the form is submitted, the selected
//  object is created by converting the form data to an object using Object.fromEntries().
//  Depending on the theme selected, the --color-light and --color-dark CSS variables are
//  updated with the corresponding light and dark color values from the css object
//
dataSettingsForm.addEventListener('submit', (event) => {
    event.preventDefault()
    const formSubmit = new FormData(event.target)
    const selected = Object.fromEntries(formSubmit)

    if (selected.theme === 'night') {
        document.documentElement.style.setProperty('--color-light', css[selected.theme][0])
        document.documentElement.style.setProperty('--color-dark', css[selected.theme][1])
    } else if (selected.theme === 'day') {
        document.documentElement.style.setProperty('--color-light', css[selected.theme][0])
        document.documentElement.style.setProperty('--color-dark', css[selected.theme][1])
    }

    dataSettingsOverlay.close()
})
```

main* ⊗ 0 ⚠ 0 Quokka | Ln 278, Col 3 | Spaces: 2 | UTF-8 | LF | {} JavaScript | Port : 5501

19°C Mostly cloudy | Search | ENG US | 10:40 2023/05/08

Day and night, selecting themes

1. The first two event listeners handle the opening and closing of a modal dialog box (`dataSettingsOverlay`) when the user clicks on the header settings button or the cancel button in the dialog box.
2. The `css` object defines two themes, "day" and "night", and each theme is defined by two RGB color values. The first value is for the light or background color, and the second value is for the dark or text color.
3. The next two lines of code use `window.matchMedia()` to check whether the user has their preferred color scheme set to dark mode. If so, the `dataSettingsTheme` input is set to "night", and the `v` variable is also set to "night".
4. The `dataSettingsForm` event listener handles the submit event that occurs when the user selects a theme and clicks the "Save" button in the dialog box. The `FormData` constructor is used to create a new object from the form data, and the `Object.fromEntries()` method is used to convert this object into an object with key-value pairs.
5. The selected theme is obtained from the `selected` object, and then the `document.documentElement.style.setProperty()` method is used to set the `--color-light` and `--color-dark` CSS variables with the corresponding values from the `css` object depending on the theme selected.
6. Finally, the `dataSettingsOverlay` is closed after the user has selected a theme.

And that, is all from me.Aobakwe Sello. Thank you.