FINAL CAPSTONE IWA19

1. I imported data from data.js

2.added variable let.

3.used const to declare a constant variable.

4.the code defines a function called `createPreview`. This function takes an object as a parameter, which should have properties like `author`, `id`, `image`, and `title`.

5.The function creates a new HTML element, which is a `button` element with the class `preview` and a `data-preview` attribute set to the book's ID

6.The function then sets the `innerHTML` of the button element to an HTML string that includes an `img` element and a `div` element.

7.The `img` element displays the book's cover image. The `div` element contains the book's title and author, which are retrieved from the `title`, `author`, and `authors` variables.Finally, the function returns the `element` object.

```
script.js > searchFilter.addEventListener('submit') callback
import { books, authors, BOOKS_PER_PAGE, genres } from "./data.js";

let page = 1;
const matches = books;

// Show more books on the list
function createPreview({ author, id, image, title }) {
  let element = document.createElement('button');
  element.classList = 'preview';
  element.setAttribute('data-preview', id);

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

      <div class="preview__info">
          <h3 class="preview__title">${title}</h3>
          <div class="preview__author">"${authors[author]}"</div>
      </div>
  `;
  return element;
}
```

1.ADDED A LET VARIABLE.document fragment is created to hold the list of book previews.
2.the `slice` method, which returns a new array with the selected elements.
3.A `for...of` loop is used to loop through the selected books and create a book preview for each one using the `createPreview` function. The book preview is then added to the fragment using the `appendChild` method.
4.The fragment is added to the page by selecting the element with the `data-list-items` attribute and appending the fragment to it.
5."Show more books" button is selected using the `querySelector` method and an event listener is added to it using the `addEventListener` method. When the button is clicked, a new subset of books is extracted based on the current page number and the `createPreview` function is used to create a book preview for each one. The previews are added to a new fragment and the fragment is appended to the page. The `showMore` variable is updated to keep track of the number of books that have been shown and the `disabled` attribute of the button is updated based on whether there are more books to show.

```
// Responsible for list of books
let fragment = document.createDocumentFragment();

const extracted = books.slice(0, 36);
for (const { author, title, image, id } of extracted) {
  const preview = createPreview({ author, id, image, title });
  fragment.appendChild(preview);
}

const dataListItems = document.querySelector("[data-list-items]");
dataListItems.appendChild(fragment);

const moreBooks = document.querySelector("[data-list-button]");
let showMore = page * BOOKS_PER_PAGE;

// show more books button
moreBooks.addEventListener("click", () => {
  const dataListItems = document.querySelector("[data-list-items]");
  const remaining = matches.slice(showMore, matches.length);
  const fragment = document.createDocumentFragment();

  for (const { author, title, image, id } of remaining) {
    const preview = createPreview({ author, id, image, title });
    fragment.appendChild(preview);
  }

  dataListItems.appendChild(fragment);
  showMore += remaining.length;
  moreBooks.disabled = !(matches.length - showMore > 0);

});
```

1. The `moreBooks` button is given a new HTML content that displays the remaining number of books to show when the button is clicked. This HTML is generated dynamically based on the number of books left to show.

2.When a book preview is clicked, an event listener is triggered. The function checks which book preview was clicked by iterating through an array of elements in the event's path. Once the active book is found, its details are displayed in an expandable overlay that appears on top of the page. The overlay displays the book's image, title, author, publication date, and description.

```
// Responsible for show more title
moreBooks.innerHTML = /* html */ `
  <span>Show more</span> (
  <span class="list__remaining">${
    matches.length - showMore > 0 ? matches.length - showMore : 0
  }</span> )
`;

// Handle preview click
document.querySelector("[data-list-items]").addEventListener("click", (event) => {
  const pathArray = Array.from(event.path || event.composedPath());
  let active;

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

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

  if (!active) return;
  document.querySelector("[data-list-active]").open = true;
  document.querySelector("[data-list-image]").setAttribute("src", active.image);
  document.querySelector( "[data-list-blur]" ).style.backgroundImage = `url('${active
  document.querySelector("[data-list-title]").textContent = active.title;
  document.querySelector( "[data-list-subtitle]").textContent = `${authors[active.aut
    (${new Date(  active.published ).getFullYear()})`;
  document.querySelector("[data-list-description]").textContent = active.description;
```

This section of the code handles the functionality of the close button in the book preview overlay. When the close button is clicked, an event listener triggers and sets the `open` attribute of the overlay's element to `false`, causing the overlay to disappear.

Additionally, this section handles the functionality of the search button in the page's header. When the search button is clicked, an event listener triggers and sets the `open` attribute of the search overlay element to `true`, causing the search overlay to appear. The search input field also receives focus, so the user can start typing their search query right away.

```
  //CLOSE BUTTON
    const closeButton = document.querySelector('[data-list-close]');
    closeButton.addEventListener('click', () => {
      document.querySelector('[data-list-active]').open = false;
    });

});

//Search modal show
document.querySelector('[data-header-search]').addEventListener('click', () => {
  document.querySelector('[data-search-overlay]').open = true ;
  data-search-title.focus();
})
```

```
 document.querySelector('[data-list-message]').innerHTML = ''
/ Get the form data using FormData
 const formData = new FormData(event.target)
  const title1 = formData.get('title');
  const genre1 = formData.get('genre');
  const author1 = formData.get('author');
/ Create an array to store filtered books
onst filteredBooks = [];
/ Loop through all books to filter based on form data
or (let i = 0; i < books.length; i++) {
 const book = books[i];
// If genre and author are not selected, filter by title only
 if (genre1 === 'any' && author1 === 'any') {
  if (book.title.toLowerCase().includes(title1.toLowerCase())){
   filteredBooks.push(book);
  }
 }
 // If genre is not selected, filter by title and author
 if (genre1 === 'any') {
```

prevent the default action of the fo
submission,

5.the code hides the book list by ch
`style.display.` code clears the mes
by setting the `innerHTML` property o
element that corresponds to the m
to an empty string.The element is i
the `[data-list-message]` attribute.

6.The fourth section of the code ge
data using the `FormData` constructo
`FormData` constructor is passed the
`event.target` object, which corresp
search form. The form data is then
the values of the `title`, `genre`, and
fields.

7.code creates an empty array calle
`filteredBooks`, loops through all th
the `books` array filters them based o
search criteria entered in the form

8.To filter the book list, the code cr
empty array called `filteredBooks`.
loops through each book in the `boo`
checks whether it meets the user's
criteria.9.If the user only enters a b
code filters the books by title only.
selects an author and/or a genre, th
filters the books based on the selec
and/or genre.The filtered books are
the `filteredBooks` array. If no boo

```
/ If genre is not selected, filter by title and author
f (genre1 === 'any') {
 if (book.title.toLowerCase().includes(title1.toLowerCase())
  && book.author === author1){
  filteredBooks.push(book);
 }
}
/ If title is not entered, filter by author and genre
if (title1 === '') {
 if (book.author === author1 && book.genres.includes(genre1)){
  filteredBooks.push(book);
 }
}
// if neither title nor author are selected, filter by genre only
if (title1 === '' && author1 === 'any' ) {
 if (book.genres.includes(genre1)){
  filteredBooks.push(book);
 }
}
// display message if no books match filters
if (filteredBooks.length > 0){
 document.querySelector('[data-list-message]').innerText = ''
 document.querySelector('[data-list-button]').disabled = true
 document.querySelector('[data-list-message]').style.marginTop =
  '-125px';
} else{
 document.querySelector('[data-list-message]').innerText =
  'No results found. Your filters might be too narrow.'
 document.querySelector('[data-list-button]').disabled = true
}
```

1.To filter the book list, the code creates an empty array called `filteredBooks`. It then loops through each book in the `books` array and checks whether it meets the user's search criteria.

2.If the user only enters a book title, the code filters the books by title only. If the user selects an author and/or a genre, the code filters the books based on the selected author and/or genre.

3.The filtered books are added to the `filteredBooks` array. If no books match the search criteria, the code displays a message indicating that no results were found.

1. selects an element with a class of "list__message" and sets its CSS display property to "block", making it visible.

2creates a new empty DocumentFragment, which is a lightweight document object that can hold other DOM elements.

3..loop iterates through each book in the `filteredBooks` array and creates a preview button for each one. The button's attributes and content are set using data attributes and interpolated template literals.

4.selects the same element as before (`list__message`) and appends the DocumentFragment `fragment2`, which contains all the preview buttons for the filtered books.

5.reset the search form and close the search overlay, presumably after the filtered books have been displayed.

```
// display filtered books
document.querySelector('[class="list__message"]').style.display = 'block'
// Create a document fragment to hold the filtered books
const fragment2 = document.createDocumentFragment()
// Loop through the filtered books and create a preview button for each one
    for (const {author ,image, title, id , description, published} of filteredBooks) {
        const preview = document.createElement('button')
        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.dataset.genre = genres
        // create preview button with the book information
        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>`
// append preview button to fragment
        fragment2.appendChild(preview)
      }
      // add filtered books to message area
          const booklist2 = document.querySelector('[class="list__message"]')
          booklist2.append(fragment2)
              document.querySelector('[data-search-form]').reset()
              document.querySelector("[data-search-overlay]").close()
          })
```

1. This code creates a drop-down menu for selecting book genres in a search feature the code creates an "All Genres" option with a value of "any" and appends it to the drop-down menu.

2.the code loops through the `genres` object, which contains genre IDs and names, and creates an option element for each genre. The option element's value is set to the genre ID and its inner text is set to the genre name. Each option element is then appended to the drop-down menu.

3.this code dynamically generates the genre options for the search feature based on the `genres` object.

```
// Drop down for genres

const dataSearchGenres = document.querySelector("[data-search-genres]");
const allGenresOption = document.createElement("option");
allGenresOption.value = "any";
allGenresOption.innerText = "All Genres";
dataSearchGenres.appendChild(allGenresOption);
for (const [id, names] of Object.entries(genres)) {
    const element = document.createElement("option");
    element.value = id;
    element.innerText = names;
    dataSearchGenres.appendChild(element);
}

for (const [id, names] of Object.entries(genres)) {
    const element = document.createElement("option");
    element.value = id;
    element.innerText = names;
    dataSearchGenres.appendChild(element);
}
```

1.I select the element with the attribute `data-search-authors` and assign it to the variable `dataSearchAuthors`.I create a new `option` element and assign it to the variable `allAuthorsOption`.I set the `value` attribute of the `allAuthorsOption` to `"any"`.I set the `innerText` property of the `allAuthorsOption` to `"All Authors` I append the `allAuthorsOption` to the `dataSearchAuthors` element.I loop through `authors`objectusing`Object.entries(authors)` For each author in the `authors` object, I create a new `option` element and assign it to the variable `element`.I set the `value` attribute of the `element` to the author's ID.We set the `innerText` property of the `element` to the author's name.i append the `element` to the `dataSearchAuthors` element.

2.select the element with the attribute `data-settings-form` .add an event listener for the `submit` event.prevent the default form submission behavior using

```
// Drop down for authors

const dataSearchAuthors = document.querySelector("[data-search-authors]");
const allAuthorsOption = document.createElement("option");
allAuthorsOption.value = "any";
allAuthorsOption.innerText = "All Authors";
dataSearchAuthors.appendChild(allAuthorsOption);
for (const [id, names] of Object.entries(authors)) {
    const element = document.createElement("option");
    element.value = id;
    element.innerText = names;
    dataSearchAuthors.appendChild(element);
}

document.querySelector('[data-settings-form]').addEventListener('submit', (event) => {
  event.preventDefault();
  actions.settings.submit();
  });

  // Closes the preview overlay
document.querySelector('[data-list-close]').addEventListener('click', () => {
  document.querySelector('[data-list-active]').open = false;
});
```

1. IT gets the settings button element and adds a click event listener to it. When clicked, it will show the theme overlay dialog box .Then gets the cancel button in the theme overlay dialog box and adds a click event listener to it. When clicked, it will close the theme overlay dialog box.and gets the theme select element and defines two objects for the color values of the day and night themes.

2. This code block sets up a theme mode for the application. It first gets the settings button and adds a click event listener to it. When the button is clicked, it displays a theme overlay dialog. The code then defines the color values for the day and night themes.

3.it gets the theme select element and the root element

```
//Theme mode

// Get the settings button and add a click event listener
const settingsBtn = document.querySelector('[data-header-settings]');
settingsBtn.addEventListener('click', (event) => {
  event.preventDefault();

  // Show the theme overlay dialog
  const themeOverlay = document.querySelector('[data-settings-overlay]');
  themeOverlay.showModal();

  const settingsCancelBtn = document.querySelector('[data-settings-cancel]');
    settingsCancelBtn.addEventListener('click', () => {
      const themeOverlay = document.querySelector('[data-settings-overlay]');
      themeOverlay.close();
    });
});

// Get the theme select element and the root element
const themeSelect = document.querySelector('[data-settings-theme]');
//const root = document.documentElement;

// Define the color values for the day and night themes
const css = {
  day: {
    dark: '10, 10, 20',
    light: '255, 255, 255',
  },
  night: {
    dark: '255, 255, 255',
    light: '10, 10, 20',
  }
}
```

1.The first line gets a reference to the form with an ID of 'settings'.

2.The next line, which is currently commented out, would get a reference to the theme select element.

3.An event listener is added to the form for the 'submit' event. When the form is submitted, the event listener's callback function is called

4.Inside the callback function, the value of the selected theme is retrieved from the 'themeSelect' element, which is currently not defined.

5.The color values for the selected theme are then applied to the document's root element using the 'setProperty' method of the 'style' property of the 'documentElement'.

6.The last two lines initialize the theme based on the user's OS theme preference. A media query is used to check if the user's preferred color scheme is dark, and the initial theme is set accordingly using the ‘setProperty’ method.

```
const form = document.getElementById('settings');
//const themeSelect = document.querySelector('[data-settings-theme]');

form.addEventListener('submit', (event) => {
  event.preventDefault();
  const theme = themeSelect.value;

  document.documentElement.style.setProperty('--color-dark', css[theme].dark);
  document.documentElement.style.setProperty('--color-light', css[theme].light);
});

// Initialize theme based on user's OS theme preference
const prefersDarkMode = window.matchMedia('(prefers-color-scheme: dark)').matches;
const initialTheme = prefersDarkMode ? 'night' : 'day';
document.documentElement.style.setProperty('--color-dark', css[initialTheme].dark);
document.documentElement.style.setProperty('--color-light', css[initialTheme].light);
```