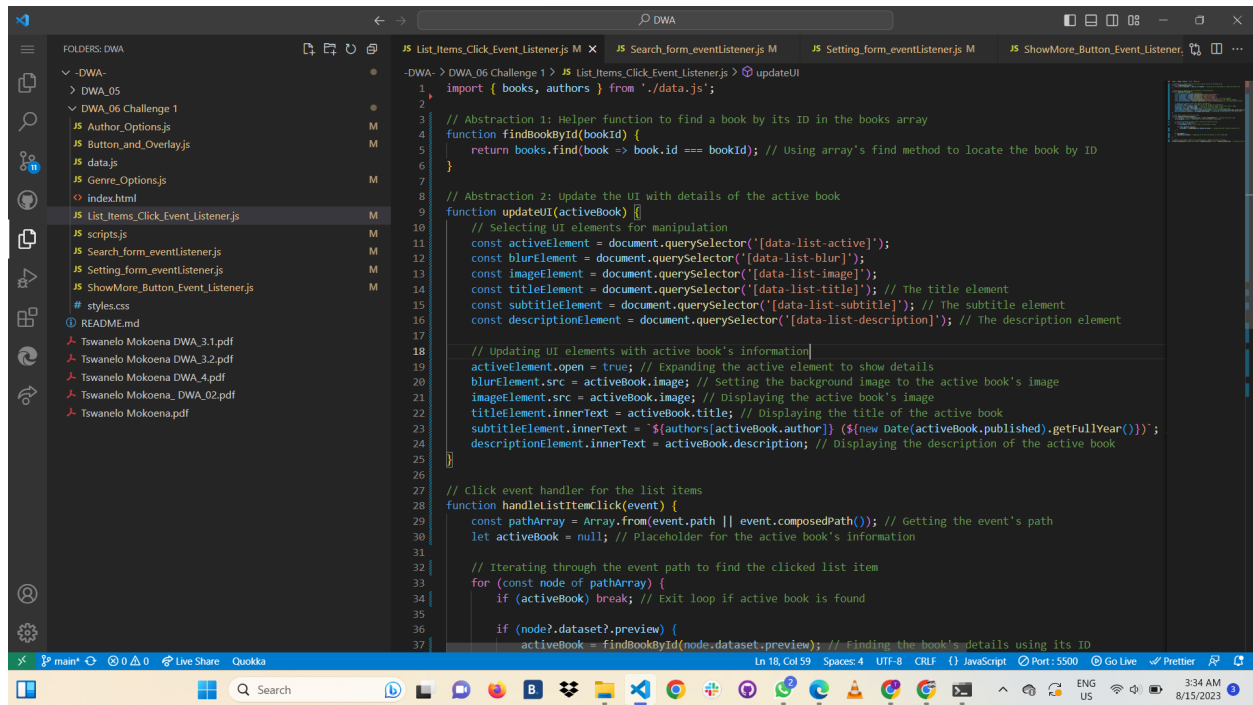


DWA_07.4 Knowledge Check_DWA7

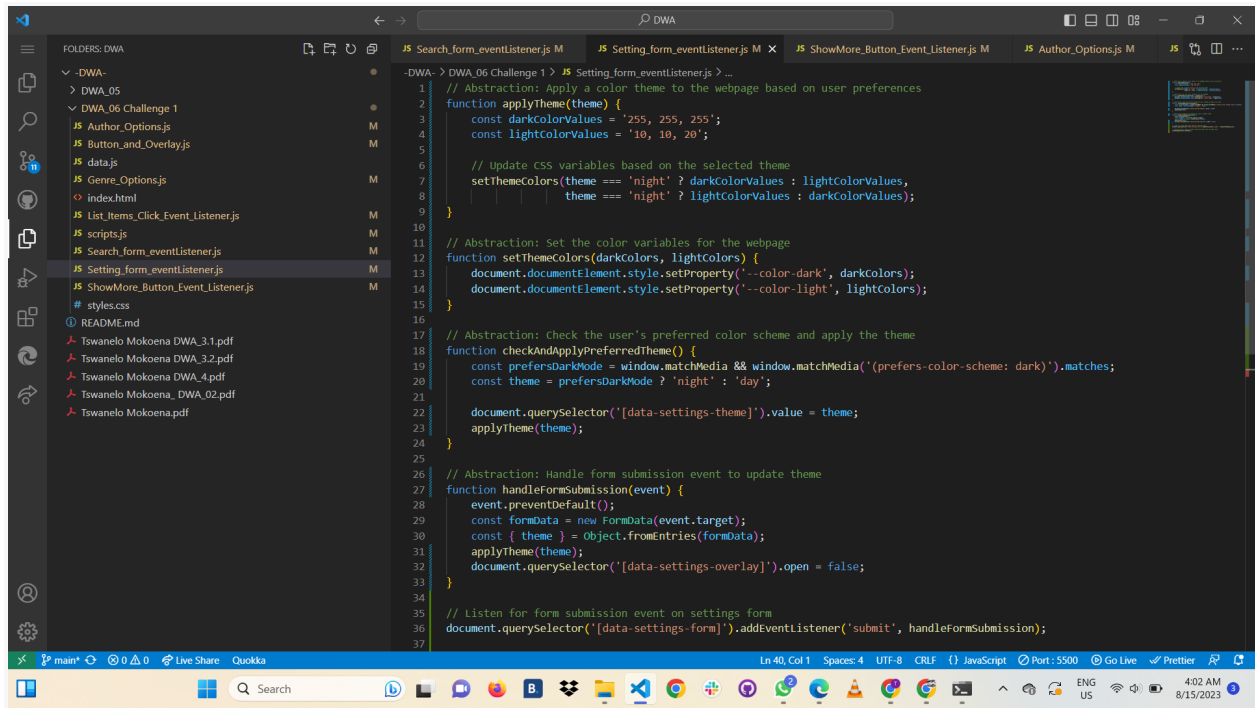
1. Which were the three best abstractions, and why?



```
1 import { books, authors } from './data.js';
2
3 // Abstraction 1: Helper function to find a book by its ID in the books array
4 function findBookById(bookId) {
5   return books.find(book => book.id === bookId); // Using array's find method to locate the book by ID
6 }
7
8 // Abstraction 2: Update the UI with details of the active book
9 function updateUI(activeBook) {
10   // Selecting UI elements for manipulation
11   const activeElement = document.querySelector('[data-list-active]');
12   const blurElement = document.querySelector('[data-list-blur]');
13   const imageElement = document.querySelector('[data-list-image]');
14   const titleElement = document.querySelector('[data-list-title]'); // The title element
15   const subtitleElement = document.querySelector('[data-list-subtitle]'); // The subtitle element
16   const descriptionElement = document.querySelector('[data-list-description]'); // The description element
17
18   // Updating UI elements with active book's information
19   activeElement.open = true; // Expanding the active element to show details
20   blurElement.src = activeBook.image; // Setting the background image to the active book's image
21   imageElement.src = activeBook.image; // Displaying the active book's image
22   titleElement.innerText = activeBook.title; // Displaying the title of the active book
23   subtitleElement.innerText = `${authors[activeBook.author]} (${new Date(activeBook.published).getFullYear()})`;
24   descriptionElement.innerText = activeBook.description; // Displaying the description of the active book
25 }
26
27 // Click event handler for the list items
28 function handleItemClick(event) {
29   const pathArray = Array.from(event.path || event.composedPath()); // Getting the event's path
30   let activeBook = null; // Placeholder for the active book's information
31
32   // Iterating through the event path to find the clicked list item
33   for (const node of pathArray) {
34     if (activeBook) break; // Exit loop if active book is found
35
36     if (node?.dataset?.preview) {
37       activeBook = findBookById(node.dataset.preview); // Finding the book's details using its ID
```

Abstraction 1: The "**findBookById**" function does something special. It helps to find a book using its ID in the group of books. This special function makes the main part of the code look nicer and easier to understand. It also divides the different jobs of the code, making it easier to work on and keep in good shape.

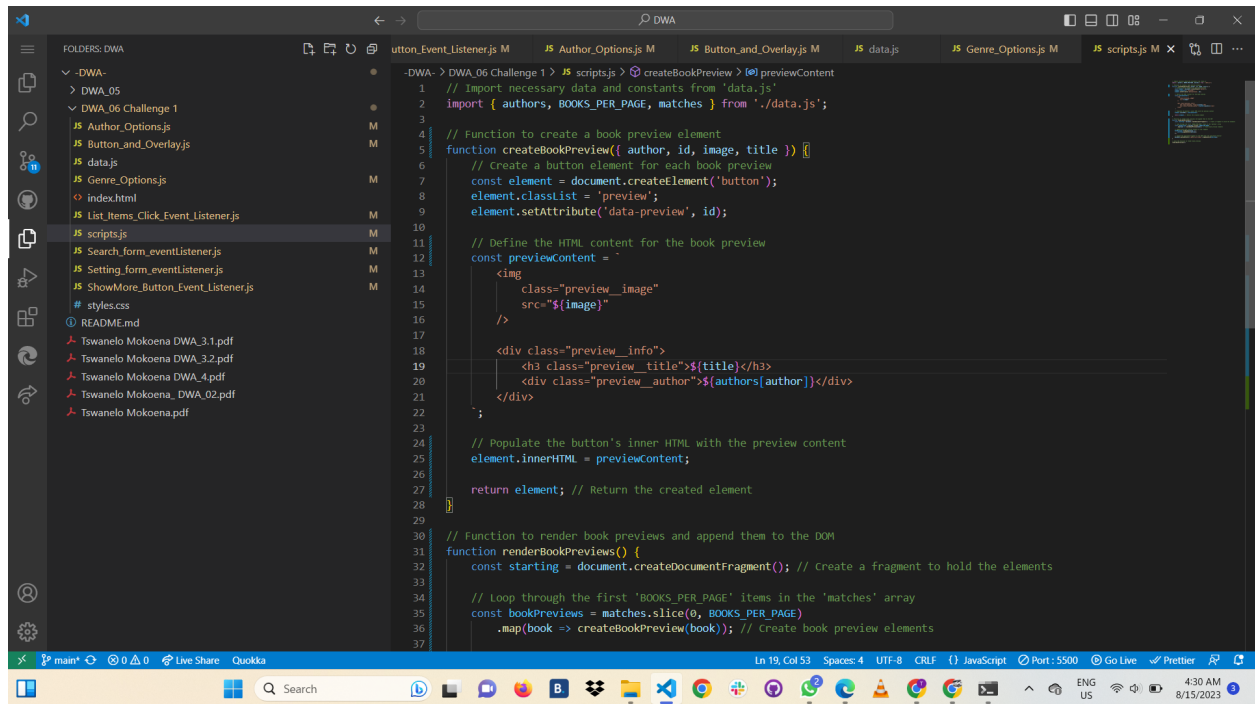
Abstraction 2: The "**updateUI**" function does something important. It puts information about a book that's being used right now into the parts you see on the screen. By putting this action into its own special function, the main set of instructions looks neater and is simpler to figure out. This also helps use the same code again later if we need to show similar updates in different parts of the app.



```
1 // Abstraction: Apply a color theme to the webpage based on user preferences
2 function applyTheme(theme) {
3   const darkColorValues = '255, 255, 255';
4   const lightColorValues = '10, 10, 20';
5
6   // Update CSS variables based on the selected theme
7   setThemeColors(theme === 'night' ? darkColorValues : lightColorValues,
8   theme === 'night' ? lightColorValues : darkColorValues);
9 }
10
11 // Abstraction: Set the color variables for the webpage
12 function setThemeColors(darkColors, lightColors) {
13   document.documentElement.style.setProperty('--color-dark', darkColors);
14   document.documentElement.style.setProperty('--color-light', lightColors);
15 }
16
17 // Abstraction: Check the user's preferred color scheme and apply the theme
18 function checkAndApplyPreferredTheme() {
19   const prefersDarkMode = window.matchMedia('(prefers-color-scheme: dark)').matches;
20   const theme = prefersDarkMode ? 'night' : 'day';
21
22   document.querySelector('[data-settings-theme]').value = theme;
23   applyTheme(theme);
24 }
25
26 // Abstraction: Handle form submission event to update theme
27 function handleFormSubmission(event) {
28   event.preventDefault();
29   const formData = new FormData(event.target);
30   const { theme } = Object.fromEntries(formData);
31   applyTheme(theme);
32   document.querySelector('[data-settings-overlay]').open = false;
33 }
34
35 // Listen for form submission event on settings form
36 document.querySelector('[data-settings-form]').addEventListener('submit', handleFormSubmission);
37
```

Using functions to group similar tasks together in the code makes it easier to understand. This way, the code is easier to read and maintain, and it helps to keep different tasks separate from each other.

2. Which were the three worst abstractions, and why?



```
1 // import necessary data and constants from 'data.js'
2 import { authors, BOOKS_PER_PAGE, matches } from './data.js';
3
4 // Function to create a book preview element
5 function createBookPreview({ author, id, image, title }) {
6   // Create a button element for each book preview
7   const element = document.createElement('button');
8   element.classList = 'preview';
9   element.setAttribute('data-preview', id);
10
11   // Define the HTML content for the book preview
12   const previewContent = `
13     
17
18     <div class="preview_info">
19       <h3 class="preview_title">${title}</h3>
20       <div class="preview_author">${authors[author]}</div>
21     </div>
22   `;
23
24   // Populate the button's inner HTML with the preview content
25   element.innerHTML = previewContent;
26
27   return element; // Return the created element
28
29
30 // Function to render book previews and append them to the DOM
31 function renderBookPreviews() {
32   const starting = document.createDocumentFragment(); // Create a fragment to hold the elements
33
34   // Loop through the first 'BOOKS_PER_PAGE' items in the 'matches' array
35   const bookPreviews = matches.slice(0, BOOKS_PER_PAGE);
36   .map(book => createBookPreview(book)); // Create book preview elements
37 }
```

Making Code Easier: Simplifying DOM Changes The function `renderBookPreviews` does two things: it creates book preview pieces and adds them to the webpage. This can be clearer if we split these tasks. If we do this, the code can be simpler to understand and test.

Fixed Selection in Code: Be Careful In the function `renderBookPreviews`, we have a fixed selector `[data-list-items]`. This finds where the book previews should go on the page. But if the webpage's structure changes, this code might not work. To avoid future problems, it's better to give the selector as a special word when using the function.

Hidden Link to 'data.js' Module: The code directly takes certain things (like `authors`, `BOOKS_PER_PAGE`, `matches`) from the 'data.js' module. This is fine for small projects, but it makes the code very connected to the data. If the data changes, the code might

stop working. It could be better to hide the data behind a special tool that the code can talk to. This would also help with testing and let us switch data sources more easily.

3. How can The three worst abstractions be improved via SOLID principles.

Single Responsibility Principle (SRP): The function "createBookPreview" does two things: it makes the structure of a web page and adds details about a book to it. It's better to do these things separately.

Open/Closed Principle (OCP): The current code can't easily add new features without changing existing parts. We can make it easier to add new things by adding a special layer.

Dependency Inversion Principle (DIP): The current code relies directly on information about authors and a fixed number for books per page. It's better to give this information to the code from the outside, instead of using fixed values.

These changes use SOLID ideas to make my code easier to work with. SOLID is like advice to make my code more organized and flexible. It's not a strict rule, and I can decide how much to use based on my program.
