DWA_07.4 Knowledge Check_DWA7

---

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

Modular Files (Modularity): Splitting the code into different files like bookListFunctionality, ThemeSettings, previewFunctionality, and others is a fantastic practice. It makes the codebase organized and easier to manage. Each file has a specific purpose, and it promotes reusability. Plus, if one part of the code needs changes, you know exactly where to go.

Functions: Creating functions for tasks like handling search form submissions and generating book previews abstracts away the complex details of those actions. This makes the code more readable and helps follow the Single Responsibility Principle. It's like having separate tools for different tasks – it's clear and focused.

Imported Modules: Importing functions and data from other files abstracts away the nitty-gritty implementation details. This is super helpful for larger projects, especially when different team members are working on different parts.

---

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

1. Event Listener Overabstraction:
In the BookListFunctionality file, there's a function handleDataListButtonClick that adds a click event listener to a specific element. While using event listeners is a good idea to make our app interactive, having too many layers of abstraction within this event listener can make things complex. The function's logic involves checking different nodes and elements, like previewId

and active, by looping through them. This complex loop within an event listener might make it harder to follow the flow of interactions and understand how the app behaves when a user clicks on something.

2. Too many Modules:
In the BookListFunctionality, we've created a function loadMoreBooks that handles loading more books and appending them to the page. While modular is generally a great practice, over-separating functionalities into too many files can lead to a fragmented codebase. For instance, we have a separate loadMoreBooks function and loadInitialBooks function to handle loading books. This division, while aiming for modularity, can sometimes lead to confusion about where to find specific functionalities. Combining related functions into a single module might help in keeping the codebase more coherent.

3. Data Duplication:
In the SearchFunctionality file, the function createAuthors creates options for authors in both the data list and the search dropdown. While this separation might seem logical, it can lead to duplicated data. If the authors' names or IDs change, we'd need to update them in two places. This data duplication can become a maintenance challenge, especially as the app grows.
A better approach could be centralizing the data in one place and using it wherever needed, reducing the chances of inconsistencies.

---

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

1. Event Listener Overabstraction:
Problem: The handleDataListButtonClick function in BookList file has complex logic within the event listener, making it hard to follow the flow of interactions.

Solution (SOLID Principle: Single Responsibility Principle):
Refactor the function to have a single responsibility: handling the click event. Move the complex logic involving previewId and active into separate functions. This will not only make the event listener more focused but also improve readability and maintainability.

2. Over-Modularization:
Problem: In the BookListFunctionality, the functions loadMoreBooks and loadInitialBooks are separated, potentially leading to a fragmented codebase.

Solution (SOLID Principle: Single Responsibility Principle and Open/Closed Principle):Combine related functions into a single module, keeping the functionalities cohesive. For instance, create a module that encapsulates both loading more books and initializing the page. This follows the Single Responsibility Principle and also leaves the module open for extension and closed for modification, as per the Open/Closed Principle.

3. Data Duplication:
Problem: The createAuthors function in SearchFunctionality duplicates author data for both the data list and search dropdown.

Solution (SOLID Principle: DRY - Don't Repeat Yourself):
Centralize the author data in a single location, possibly in the data.js file. This ensures that any changes made to the author data are reflected consistently across the app. Both the data list and search dropdown should reference the centralized author data. This adheres to the DRY principle, reducing duplication and maintenance challenges.
