DWA_08 Discussion Questions

In this module you will continue with your “Book Connect” codebase, and further iterate on your abstractions. You will be required to create an encapsulated abstraction of the book preview by means of a single factory function. If you are up for it you can also encapsulate other aspects of the app into their own abstractions.

To prepare for your session with your coach, please answer the following questions. Then download this document as a PDF and include it in the repository with your code.

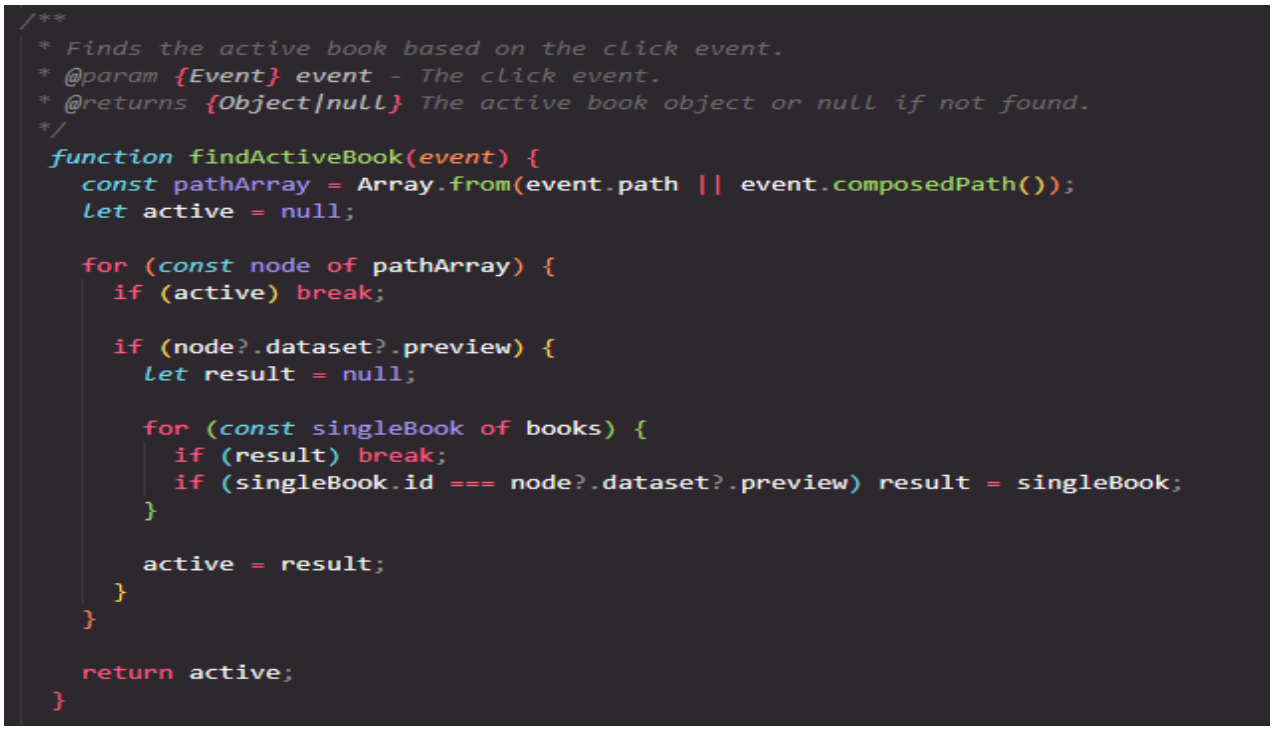
---

1. What parts of encapsulating your logic were easy?

Function Encapsulation: By defining functions separately, you can isolate specific blocks of code and assign them a clear purpose. Functions can maintain their own local variables, minimizing the risk of variable name clashes and other side effects. This approach simplifies the organization and management of my code.

Lexical Scoping and Closures: Lexical scoping allows functions to access variables defined in their outer scope. Combined with closures (functions that remember the environment in which they were created) this feature enables the encapsulation and preservation of state within a function. You can create private variables that are only accessible within the function's scope, thereby hiding internal implementation details and providing a clear interface for interacting with the code.

The **findActiveBook** function operates within its own local scope



```
/**
 * Finds the active book based on the click event.
 * @param {Event} event - The click event.
 * @returns {Object|null} The active book object or null if not found.
 */
function findActiveBook(event) {
  const pathArray = Array.from(event.path || event.composedPath());
  let active = null;

  for (const node of pathArray) {
    if (active) break;

    if (node?.dataset?.preview) {
      let result = null;

      for (const singleBook of books) {
        if (result) break;
        if (singleBook.id === node?.dataset?.preview) result = singleBook;
      }

      active = result;
    }
  }

  return active;
}
```

---

2. What parts of encapsulating your logic were hard?

Shared Global Scope: In this scope, variables and functions are accessible throughout the entire application. This can make it challenging to encapsulate code and prevent unintended access or modification of internal logic. Therefore, modularization and avoiding global variables are best practices that promote a more organized, maintainable, and reliable codebase. These practices help reduce issues related to internal logic by providing clear boundaries and encapsulation, making the code easier to understand and manage.

For example, the **openDataListActive** function is defined in the global scope, making it accessible from anywhere within the JavaScript code.

---

3. Is abstracting the book preview a good or bad idea? Why?

Abstracting the book preview functionality into a separate module or component is a good idea. This promotes reusability by encapsulating the logic within a distinct module. By doing so, you can maintain and update the preview functionality more easily without affecting other parts of the application.
