Lab Exercise handling-errors

This is a lab exercise on developing secure software. For more information, see the introduction to the labs.

1. Task

Please change the code below to leverage the exception handling mechanism and improve its robustness.

2. Background

In this exercise, we'll demonstrate how to enhance the robustness of a JavaScript function by transitioning from a return code mechanism to exception handling. The primary goal is to understand how to handle errors more effectively, improving the readability and maintainability of the code.

Initially, the code uses a return code mechanism to handle potential errors. The `divide` function returns an object indicating whether the operation was successful or if an error occurred. The caller must then check the success property and handle the result or error accordingly. This method works but can clutter the code with repetitive error-checking logic and what's worse, it's easy for callers to ignore the error return code, which can result in surprising failures and security vulnerabilities.

3. Task Information

Please change the code below to leverage the exception handling mechanism and improve its robustness.

By throwing an error when an invalid operation is detected (in this case, division by zero), we separate the normal flow of the code from the error-handling logic. The caller can use a try...catch block to handle any exceptions that might be thrown, making the code cleaner and more robust.

To complete this task:

1. Locate the function in your code that uses return codes to indicate success or failure. In our case, the function is `divide` .
2. Modify the function to throw an error when an invalid operation is detected. In our case, we throw an error when the parameter `b` is zero.
3. Set the error message to "Division by zero is forbidden".
4. Update the success path to return the result of the division operation.

5. Modify the calling code to use a try block to wrap the call to the `divide` function.
6. Within the try block, log the result of the division operation if no error is thrown.
7. Add a catch block to handle any errors that might be thrown by the `divide` function.
8. For both the success and error paths, log the appropriate message to console or to error.

Use the “hint” and “give up” buttons if necessary.

4. Interactive Lab (COMPLETE!)

Please change the code below so the `divide` function and the calling code use exception handling instead of a return code mechanism. The `divide` function should throw an error when division by zero is attempted. The calling code should use a try...catch block to handle the error, stgore the result in a constant named `result`, catch any error into a variable named `error`, and display an appropriate message.

```
// Implement a simple division method that returns the result of a / b
function divide(a, b) {
  if (b === 0) {
    throw new Error("Division by zero is forbidden");
  } else {
    return  a / b ;
  }
}

// This code calls the divide function and logs the result or error
try{
  const result = divide(10, 2);
  console.log("Result:",result);
} catch(err) {
  console.error("Error:", err.message);
}
```

Hint Reset Give up

This lab was developed by Avishay Balter at Microsoft.

Completed 2025-04-03T22:34:29.758Z 962b2a31-c973-4549-906b-fb06ca6dd584 063fqgz0t474lb (GA)