**DataAccess.cs**

**1. Null Reference Exception**

* **Uninitialized Objects:** Objects like \_returnDapper might not be initialized before accessing their properties.
* **Null Query Results:** Query methods like Query or QueryAsync might return null, and ToList or property access is attempted without validation.
* **Missing Data in Stored Procedure:** Stored procedure may not return expected output, causing null values in the result set.

**2. ObjectDisposedException**

* **Premature Connection Closure:** The database connection is explicitly closed (via EnsureConnectionClosed()) before the query or reader operations complete.
* **Improper Use of GridReader:** Accessing GridReader from QueryMultiple after the connection is disposed.
* **Reader Consumption Issue:** Attempting to access query results after the reader is disposed.

**3. ArgumentNullException**

* **Null Parameters:** Parameters passed to a stored procedure, such as ClientId or JobTagIds, might be null.
* **Null Query Results:** Attempting LINQ operations like ToList, First, or Select on a null query result.
* **Missing or Malformed Input:** Stored procedure input parameters are not properly populated or validated.

**4. General Code Design Issues**

* **Explicit Connection Management:** The use of EnsureConnectionOpen and EnsureConnectionClosed risks leaving connections open or prematurely closing them, leading to inconsistent behavior.
* **Transaction Handling Gaps:** Missing or incomplete transaction commits/rollbacks might cause partial or invalid results.
* **Improper Output Handling:** Output parameters like @ReturnStatus or @ErrorCode might not be set by the stored procedure, leading to inconsistent behavior.

**DashboardFilter.razor.cs**  
**1. Null Reference Exceptions**

Reason: Uninitialized Properties

SessionStorage, EquipmentHttpClient, and other injected services might not be set or injected correctly in the component.

Example: If dependency injection is misconfigured or services are not registered.

Reason: Empty or Null Session Data

If session storage data (DashBoardFilterData) is not present or improperly stored, DashBaoardFilterData could be null, leading to property access issues.

Reason: Missing API Data

API calls (e.g., GetUserSettingsAsync, GetAllPositionDetails) might return null or empty results, and the code does not validate the response before processing.

**2. Invalid Operation Exceptions**

Reason: Concurrent Modification

Modifications to VesselDashBoardParameters (e.g., RemoveAll and Add) without thread-safety in a concurrent environment can cause exceptions.

Reason: Incorrect Task Synchronization

Improper async/await handling in methods like ShowDashBoardData or PopulateCombo can lead to race conditions or unawaited tasks.

**3. Deserialization or Parsing Errors**

Reason: Mismatched API Response

API responses might not match the expected model structure, leading to errors during deserialization (e.g., GetUserSettingsAsync parsing settingValueString).

Reason: Invalid Date Formats

The DateTime.TryParse might fail if settingValueString is in an unexpected format or culture.

**4. Performance Issues**

Reason: Parallel API Calls Without Coordination

Task.WhenAll is used for multiple API calls, but failures in any of these tasks can cause the method to fail without clear logging or recovery.

Reason: Frequent Session Storage Access

Multiple reads/writes to session storage (e.g., SessionStorage.GetItemAsync, SetItemAsync) can slow down performance in a high-frequency usage environment.

**5. Unhandled Exceptions**

Reason: Missing Error Handling in PopulateCombo

If any of the service calls fail, the generic Exception thrown in the catch block doesn’t log the actual error source or reason.

Reason: Exception in DashBoardParameter

Operations like RemoveAll or adding new parameters to VesselDashBoardParameters might fail silently if Values or HardCode is null/invalid.

**6. Logical Bugs**

Reason: Incorrect Filter Handling

Logic in DashBoardParameter and OnClear might leave VesselDashBoardParameters in an inconsistent state due to partial clearing or updating.

Reason: Potential Empty Date Handling

If FromDate or ToDate is null, downstream operations dependent on these values might behave unexpectedly.

Reason: Redundant Hard-Coded Values

Hard-coded values like "VSL", "GRP", "CAT" are repeated in multiple places, increasing maintenance overhead and risk of inconsistency.

**7. Resource Management Issues**

Reason: ResourceManager String Retrieval

If resource strings (e.g., rsAllPendingJobs) are missing or incorrectly defined in Resources, ResourceManager.GetString might return null.

**8. Usability and UX Concerns**

Reason: Lack of Feedback on Loader

LoaderVisible changes are not communicated effectively to the user if errors occur during asynchronous operations.

**JobClassHttpClient.cs**

**1. Null Reference Exceptions**

* **Reason: Missing Token**
  + \_tokenService.GetToken() might return null, and authorization is not set on \_httpClient.
  + Example: If the token service fails or is not properly implemented.
* **Reason: Missing Local Storage Data**
  + await \_localStorageService.GetItemAsync<List<GetJobClassModel>> might return null, and operations like Add or FindIndex on the resulting list could cause an error.
* **Reason: Missing API Response**
  + API calls (GetStreamAsync, PostAsync, etc.) may return empty or invalid responses that are not handled.

**2. Deserialization Errors**

* **Reason: API Schema Mismatch**
  + The API response might not match the structure of CreateJobClassResponse or other models used in deserialization.
  + Example: Missing or incorrectly named properties in the JSON response.
* **Reason: Case Sensitivity Issues**
  + If the API returns JSON with properties that don’t match the case of your model properties, deserialization may fail unless PropertyNameCaseInsensitive is enabled.
* **Reason: Invalid Data from Local Storage**
  + If the data stored in local storage doesn’t match the expected model (GetJobClassModel), deserialization will fail.

**3. Unhandled HTTP Errors**

* **Reason: API Error Responses**
  + If the API returns a non-success status code (e.g., 4xx or 5xx), the response is not properly handled, leading to potential runtime errors.
* **Reason: Missing Error Handling**
  + Methods like PostAsync, PutAsync, and DeleteAsync do not check the status code before proceeding with deserialization.

**4. Concurrency Issues**

* **Reason: Race Conditions**
  + Concurrent calls to LocalStorageService for the same keys may result in unexpected behavior or overwrites.
* **Reason: No Synchronization**
  + Multiple clients might access and modify the local storage or job class list without synchronization, causing inconsistencies.

**5. Validation Errors**

* **Reason: API Validation**
  + The API might return validation errors like "COMERR102" or "JCERR105" that are not logged or properly acted upon.
* **Reason: Missing Validation Check**
  + Some responses do not check for ValidationErrors before proceeding, which might cause incorrect or incomplete updates.

**6. Authorization Issues**

* **Reason: Missing Token**
  + The token is retrieved but not set in the Authorization header, leading to unauthorized API calls.
* **Reason: Expired Token**
  + If the token is expired, API requests might fail without a retry mechanism.

**7. Local Storage Expiration Issues**

* **Reason: Expired Data Not Refreshed**
  + If JobClassDataExpirationKey indicates expired data but the API call fails, the local storage remains outdated.
* **Reason: Missing Expiration Key**
  + The code assumes the expiration key exists, but it might not be set, leading to incorrect logic execution.

**8. Edge Cases Not Handled**

* **Reason: Empty Job Class List**
  + If jobclassExistingData or jobclasslist is empty, operations like FindIndex or RemoveAt might cause errors.
* **Reason: Missing API Endpoints**
  + The endpoints (/api/JobClass/) might not exist or are misconfigured, causing runtime errors.

**9. General Code Design Issues**

* **Reason: Inefficient Local Storage Access**
  + Frequent reads and writes to local storage may impact performance and lead to inconsistencies.
* **Reason: Redundant Code**
  + Repeated logic for token retrieval and setting headers could lead to maintenance overhead.