

This test verifies that the **SecurityOverrideList** component correctly renders when it receives data from the API.

1. **Mock API Response**: The test uses mockEndpoint to return predefined mockData, ensuring that no real API calls are made.
2. **Component Rendering**: The TestComponent is rendered, which contains the SecurityOverrideList.
3. **Waiting for UI Update**: The waitFor function ensures that the component has finished fetching data before making assertions.
4. **Assertions**:
   * Checks if the **grid wrapper (.ag-root-wrapper)** exists in the document.
   * Verifies that **43 header cells** (.ag-header-cell) are rendered, confirming correct column structure.

A computer screen with text

Description automatically generated

This test ensures that the **SecurityOverrideList** component properly handles cases where the API returns no data (undefined).

1. **Mock API Response**: The test sets up mockEndpoint to return undefined, simulating a scenario where the backend does not provide data.
2. **Component Rendering**: The TestComponent is rendered, which contains SecurityOverrideList.
3. **Waiting for UI Update**: The waitFor function ensures the component has finished processing before making assertions.
4. **Assertions**:
   * Checks that the **grid wrapper (.ag-root-wrapper)** **exists**, confirming that the component still renders.
   * Ensures that **no data rows (.ag-row) are present**, validating that the component correctly handles empty responses.

A screen shot of a computer program

Description automatically generated

This test ensures that the **SecurityOverrideList** component correctly displays a **loading spinner** while data is being fetched from the API.

1. **Mock API Response**: The test sets up mockEndpoint to return mockData, simulating a real API request.
2. **Component Rendering**: The TestComponent is rendered, which contains the SecurityOverrideList.
3. **Waiting for UI Update**: The waitFor function ensures that assertions are executed after the component has completed rendering.
4. **Assertion**:
   * Checks that the **loading spinner (.nexus-spinner-loading-spinner-small) is not in the document**, verifying that it disappears after the data has been loaded.

A computer screen with text

Description automatically generated

This test ensures that the **SecurityOverrideList** component correctly handles cases where the API returns **malformed or unexpected data**.

1. **Mock API Response**:
   * The test sets up mockEndpoint to return a response with an **incorrect data structure** ([{ id: 20 }]).
   * The status is set to 200, meaning the API request itself is successful, but the data format is not as expected.
2. **Component Rendering**:
   * The TestComponent is rendered, which contains the SecurityOverrideList.
3. **Waiting for UI Update**:
   * The waitFor function ensures that assertions are executed only after the component has finished rendering.
4. **Assertion**:
   * **Checks if the grid (.nexus-grid) is present in the document**. If this test passes, it indicates that the grid is still rendering despite incorrect data.

A screen shot of a computer program

Description automatically generated

This test verifies that the **SecurityOverrideList** component correctly displays a **message** when the API returns an empty dataset.

1. **Mock API Response**:
   * The test sets up mockEndpoint to return an **empty array ([])**.
   * The status code is 200, meaning the request was successful, but no data is available.
2. **Component Rendering**:
   * The TestComponent is rendered, which includes the SecurityOverrideList.
3. **Waiting for UI Update**:
   * The waitFor function ensures assertions are executed after the component has finished rendering.
4. **Assertion**:
   * The test checks whether an **empty data message (.nexus-no-data) is not present in the document**.
   * This assertion might be incorrect, as the expected behavior should be to **ensure the message appears** when there is no data.

A computer screen with text on it

Description automatically generated

A computer screen with text on it

Description automatically generated

This test verifies that the **SecurityOverrideList** component refreshes its data when the **refresh button is clicked**.

1. **Mock API Response**:
   * The test sets up mockEndpoint to return refreshedData, simulating new data after a refresh.
   * The API endpoint /crd.api.alpha.benchmark/hub/v1/security-overrides?page=0&limit=100 is mocked with updated data.
2. **Component Rendering**:
   * The TestComponent is rendered, which includes the SecurityOverrideList.
3. **Finding the Refresh Button**:
   * The test waits until the **refresh button (.bi-arrow-repeat) is available** in the document.
   * Uses vi.waitUntil to ensure the button is detected within a specified timeout (500ms).
4. **Assertions Before Clicking Refresh**:
   * Ensures the **refresh button is in the document**
   * **Triggering the Refresh Action**, Simulates a **click event** on the refresh button.
   * Ensures the **updated data grid (.nexus-grid) is displayed** after the refresh action

A computer screen shot of a code

Description automatically generated

This test ensures that the **SecurityOverrideList** component displays the correct **column headers** in the data grid based on the defined column definitions (colDefs).

**Steps Involved:**

1. **Mock API Response**:
   * The test sets up mockEndpoint to return mockData with a successful 200 status.
   * This simulates the API providing the expected data for the grid.
2. **Component Rendering**:
   * The TestComponent is rendered, which includes the SecurityOverrideList.
3. **Waiting for UI Update**:
   * The waitFor function ensures that the component has fully rendered before making assertions.
   * The test fetches all elements with the class .nexus-grid-header-cell, which represents the **column headers** in the grid.
4. **Defining Expected Columns**:
   * A list of expected column names (expectedColumns) is defined to match the expected headers.
5. **Assertions**
   * The test iterates through each column header and verifies that its text content matches the expected column name.
   * Each column header's text should match the expected column name

A screen shot of a computer code

Description automatically generated

This test ensures that the **SecurityOverrideList** component correctly handles an **invalid API response** by showing an error when the server returns a failure status (500).

**Steps Involved:**

1. **Mock API Response**:
   * The test sets up mockEndpoint to return a 500 status code, simulating a **server failure**.
   * This checks how the component behaves when the backend fails.
2. **Component Rendering**:
   * The TestComponent is rendered, which includes the SecurityOverrideList.
3. **Waiting for UI Update**:
   * The test waits for the **refresh button (.bi-arrow-repeat)** to be available using vi.waitUntil.
   * A **click event** is fired on the refresh button to trigger a data fetch attempt.
4. **Assertions**:
   * The test **checks if the grid (.nexus-grid) is still in the document** despite the API error.
   * The component handles the server error **gracefully**.
   * It **prevents incorrect data from rendering** or shows an error message.

A screenshot of a computer

Description automatically generated

**1. Overall Test Coverage Summary**

Our project has **90.57% statement coverage**, meaning nearly all executable code is tested. **Branch coverage is at 82.28%**, which means some conditional paths are untested. **Function coverage is at 80.81%**, indicating that around 20% of functions still need test cases. While these numbers are strong, we can improve by increasing function and branch coverage, particularly in complex logic and edge cases.

**2. Key Areas for Improvement**

The **shared-components module has the lowest function coverage at 62.5%**, suggesting that several component methods, event handlers, or UI interactions are not directly tested. Additionally, the **services/hub module has only 72.72% branch coverage**, meaning some conditions in API handling logic might not be covered. To improve, we should **add more tests for error handling, alternate data flows, and missing UI interactions**.

**3. Next Steps & Action Plan**

While our overall test coverage is strong, we should **focus on untested functions and missing branch cases** to ensure complete validation. Adding **unit tests for UI interactions, API edge cases, and failure scenarios** will enhance reliability. Increasing test coverage will not only improve software stability but also ensure **better maintainability and confidence in future releases**.