Assignment 2: Evaluation Criteria

**1. Code Quality (30%)**

* **Modularity & Reusability:**
  + Organize your test code into separate files/modules based on functionality (e.g., sortingTests.js, checkoutTests.js, etc.).
  + Create helper functions for repetitive actions (like login, adding items to the cart) to avoid code duplication.
* **Readability & Best Practices:**
  + Use descriptive variable and function names (e.g., verifySortingOrder, addItemToCart).
  + Follow consistent formatting (indentation, spacing, etc.) and style guides (like ESLint) to maintain code quality.
  + Comment on complex logic or any non-obvious decisions to improve understanding.
* **Error Handling:**
  + Implement try-catch blocks around critical sections of your tests to gracefully handle unexpected errors.
  + Log useful error messages to help diagnose issues during test failures.

**2. Test Coverage (30%)**

* **Required Scenarios:**
  + Ensure you cover all specified scenarios:
    - **Sorting Order (Z-A):** Verify items are sorted correctly in descending order.
    - **Price Order (High-Low):** Verify items are sorted correctly by price in descending order.
    - **Checkout Journey:** Validate that multiple items can be added to the cart and the checkout process works as expected.
* **Assertions:**
  + Use comprehensive assertions to validate expected outcomes (e.g., check the order of items, total prices).
  + Consider using libraries like chai for better assertion styles.
* **Edge Cases:**
  + Test scenarios where no items are present, or items with identical names/prices are displayed.
  + Check for UI behavior when adding out-of-stock items to the cart.

**3. Bonus Features (20%)**

* **Visual Testing:**
  + Implement visual regression testing using tools like Selenium with visual-regression options.
  + Capture screenshots of important pages before and after actions to compare layouts.
* **Accessibility Testing:**
  + Use tools like axe-core or pa11y to test for common accessibility issues.
  + Check for issues like missing alt attributes, color contrast, and ARIA roles.

**4. GitHub Repository (10%)**

* **Structure & Commit History:**
  + Use a clear directory structure (e.g., tests/, utils/, reports/).
  + Make meaningful commits with clear messages (e.g., Add tests for sorting functionality).
* **README.md:**
  + Include comprehensive setup and execution instructions.
  + Add sections for test coverage and how to interpret the results.

**5. Test Execution & Reporting (10%)**

* **Headed/Headless Execution:**
  + Record both types of execution using screen capture software. Make sure to highlight important steps in the headed version.
* **Reports & Logs:**
  + Ensure your Playwright tests generate detailed reports (can use the built-in reporter or customize your own).
  + Include logs in your repository that capture the test execution details.

**6. Initiative (Optional - 5%)**

* **Extra Efforts:**
  + Consider implementing CI/CD using GitHub Actions or similar tools to automatically run tests on pull requests.
  + Explore other enhancements, like integrating notifications for test results or using Docker for environment consistency.

**Final Checklist:**

* Review the final implementation against the evaluation criteria to ensure all aspects are covered.
* Test your entire suite locally to confirm all scenarios execute successfully.
* Make sure all files are pushed to the GitHub repository before submission.