Multiple Page WebTable Handling in Playwright

What is Pagination?

Pagination in web tables involves splitting data into multiple pages. You can navigate between these pages using buttons or links, usually labeled with page numbers or next/previous controls.

Example DOM Structure:

```
<table id='heroesTable'>
  <thead>
    <tr><th>S.No</th><th>Hero</th><th>Movie</th></tr>
  </thead>
  <tbody>
    <tr><td>1</td><td>Iron Man</td><td>Iron Man (2008)</td></tr>
    <tr><td>2</td><td>Captain America</td><td>The First Avenger
(2011)</td></tr>
    <tr><td>3</td><td>Thor</td><td>Thor (2011)</td></tr>
    <tr><td>4</td><td>Hulk</td><td>The Incredible Hulk (2008)</td></tr>
    <tr><td>5</td><td>Black Widow</td><td>Iron Man 2 (2010)</td></tr>
    <tr><td>6</td><td>Hawkeye</td><td>Thor (2011)</td></tr>
  </tbody>
</table>

<table id='villainsTable'>
  <thead>
    <tr><th>S.No</th><th>Villain</th><th>Movie</th></tr>
  </thead>
  <tbody>
    <tr><td>1</td><td>Loki</td><td>Thor (2011)</td></tr>
    <tr><td>2</td><td>Thanos</td><td>Avengers: Endgame (2019)</td></tr>
    <tr><td>3</td><td>Ultron</td><td>Avengers: Age of Ultron
(2015)</td></tr>
    <tr><td>4</td><td>Killmonger</td><td>Black Panther (2018)</td></tr>
    <tr><td>5</td><td>Red Skull</td><td>Captain America (2011)</td></tr>
    <tr><td>6</td><td>Vulture</td><td>Spider-Man: Homecoming
(2017)</td></tr>
  </tbody>
</table>
```

```
<table id='teamsTable'>
  <thead>
    <tr><th>S.No</th><th>Team</th><th>Movie</th></tr>
  </thead>
  <tbody>
    <tr><td>1</td><td>Avengers</td><td>The Avengers (2012)</td></tr>
    <tr><td>2</td><td>Guardians of the Galaxy</td><td>Guardians of the
Galaxy (2014)</td></tr>
    <tr><td>3</td><td>Revengers</td><td>Thor: Ragnarok (2017)</td></tr>
    <tr><td>4</td><td>S.H.I.E.L.D.</td><td>Captain America: The Winter
Soldier (2014)</td></tr>
    <tr><td>5</td><td>Wakandan Army</td><td>Black Panther (2018)</td></tr>
    <tr><td>6</td><td>Dora Milaje</td><td>Black Panther (2018)</td></tr>
  </tbody>
</table>

<div class='pagination'>
    <button id='prev'>Previous</button>
    <button id='next'>Next</button>
</div>
```

Step-by-Step Script Explanation

Step 1: Capture the Table and Pagination Controls

Here, we capture the table elements and pagination buttons to interact with them.

```
// Define an array of table IDs that we will process
const tables = ['#heroesTable', '#villainsTable', '#teamsTable'];

// Capture the 'Next' and 'Previous' pagination buttons
const nextButton = await page.locator('#next');
const prevButton = await page.locator('#prev');
```

Explanation:

- We store the table IDs for later use when selecting each table element.
- We also capture the "Next" and "Previous" buttons that will be used to navigate between pages.

Step 2: Count Rows and Columns

We extract the total number of rows and columns for each table on the current page.

```
// Loop through each table to extract rows and columns information
for (const tableId of tables) {
    // Select the table using the table ID
    const paginationTable = await page.locator(tableId);
```

```
// Count the columns in the table header
    const columns = await paginationTable.locator('thead tr th');
    console.log(`Table ID: ${tableId}`);
    console.log('Total number of columns:', await columns.count());

    // Count the rows in the table body
    const rows = await paginationTable.locator('tbody tr');
    console.log('Total number of rows on current page:', await
rows.count());
}
```

Explanation:

- We loop through each table, select it, and count the number of columns by looking at the header (`thead`).
- We also count the number of rows displayed on the current page of each table by selecting the table body (`tbody`).

Step 3: Navigate Pages and Collect Data

In this step, we define a function that navigates through pages and collects all rows from the table. The function will click the "Next" button to load additional pages.

```
// Function to collect all rows by navigating through the pagination
async function collectAllRows(page, tableLocator, nextButtonLocator) {
    let allData = []; // Array to store the row data from all pages
    let pageIndex = 1; // To keep track of the page number

    // Continue until the 'Next' button is not visible or clickable
    do {
        // Select all the rows on the current page
        const rows = await tableLocator.locator('tbody tr');

        // Loop through each row and extract the data from each cell
        for (let i = 0; i < await rows.count(); i++) {
            const rowData = await
rows.nth(i).locator('td').allInnerTexts();
            allData.push(rowData); // Add the row data to the allData array
        }

        pageIndex++; // Move to the next page
        // Click the 'Next' button to load the next page of rows
    } while (await nextButtonLocator.isVisible() && await
nextButtonLocator.click());

    // Return all the collected data after all pages have been processed
    return allData;
}
```

Explanation:

- The `collectAllRows` function loops through all pages and collects data from the rows of each page.
- It uses the "Next" button to navigate through the pages.

- For each page, it collects the rows from the table and stores them in the `allData` array.

Step 4: Test Script

Finally, we implement a test script that iterates through all tables, collects rows, and verifies the pagination functionality.

```
import { test, expect } from '@playwright/test';

test('pagination handling for multiple tables', async ({ page }) => {
    // Navigate to the page containing the tables
    await page.goto('https://..');

    // Loop through each table and collect the rows
    for (const tableId of tables) {
        const paginationTable = await page.locator(tableId); // Select the
current table
        const allRows = await collectAllRows(page, paginationTable,
nextButton); // Collect all rows

        // You can process the rows here (e.g., store them, assert data,
etc.)
        // The collected rows are in the 'allRows' variable
    }
});
```

Explanation:

- The `test` function runs the test that navigates to the page and iterates over each table.
- It collects all rows using the `collectAllRows` function.
- You can process or assert the data as needed.

Summary of Key Points:

1. **Table Selection**: We capture the table elements using their IDs and store them in an array for iteration.
2. **Pagination Handling**: We dynamically handle pagination by clicking the "Next" button and collecting rows from all pages.
3. **Data Collection**: For each page, we collect row data from the table, store it, and return the data once all pages have been processed.
4. **Test Execution**: The test iterates through all the tables, collects data, and can be extended to perform further processing or assertions on the data.