Dropdowns

A **dropdown** allows users to select options from a list. It can be:

- **Single-select**: Only one item can be chosen (e.g., Country list).
- **Multi-select**: Multiple items can be selected (e.g., Favorite colors).

1. Selecting Options from Dropdowns

Playwright provides 4 simple ways to select options from a dropdown.

For Single-select Dropdowns (like #country):

By Visible Text

Select "India" by visible label shown to users.

```
await page.locator('#country').selectOption('India');
```

By Value Attribute

Select option using its value in HTML (e.g., <option value="uk">UK</option>).

```
await page.locator('#country').selectOption({ value: 'uk' });
```

By Label

Alternative way to use label property explicitly.

```
await page.locator('#country').selectOption({ label: 'India' });
```

By Index

Select option by its position (starting from 0).

```
await page.locator('#country').selectOption({ index: 3 });
```

For Multi-select Dropdowns (like #colors):

Use the same methods, but pass **arrays** to select multiple options.

Example: Select multiple colors using visible text:

```
await page.locator('#colors').selectOption(['Red', 'Green', 'Blue']);
```

2. Count of Options

You can check how many options are available in the dropdown.

Example:

```
const options = page.locator('#country > option');
await expect(options).toHaveCount(10);
```

📝 Useful to validate if all expected choices are loaded.

3. Check If a Specific Option Exists

Get all dropdown option texts and check if a certain item exists.

Example: Check if "Japan" is present

```
const optionsText = await page.locator('#country > option').allTextContents();
expect(optionsText).toContain('Japan');
```

📝 Great for testing if expected options are present.

4. Print All Dropdown Options

You can loop through the list and log each item.

Example:

```
const texts = await page.locator('#colors > option').allTextContents();
for (const text of texts) {
   console.log(text);
}
```

📝 Helpful to verify the dropdown content visually or in logs.

5. Check for Duplicate Options

Use a Set to detect if any options are repeated in the dropdown.

Example:

```
const options = await page.locator('#colors > option').allTextContents();
const set = new Set();
const duplicates = [];
for (const item of options) {
  if (set.has(item)) {
    duplicates.push(item);
  } else {
    set.add(item);
  }
}
console.log("Duplicate items:", duplicates);
```

📝 Good practice to ensure data quality in dropdowns.

6. Check If Dropdown Is Sorted Alphabetically

Compare the original list with a sorted version.

Example:

```
const options = await page.locator('#animals > option').allTextContents();
const original = [...options];
const sorted = [...options].sort();
expect(original).toEqual(sorted);
```

📝 Ensures dropdown values appear in expected order (A to Z).

The syntax **[...]** is called the **spread operator** in JavaScript/TypeScript.

When you see:

```
const originalList = [...options];
```

It means:
"Create a new array with the same elements as options."

Why it's used:

It **creates a shallow copy** of the array options. This is important because:

```
const sortedList = options.sort();  // ⚠️ This changes the original array!
```

If you sort the original array directly, you lose the original order. So instead, you do:

```
const originalList = [...options];     // save original order
const sortedList = [...options].sort(); // sorted version, without modifying the original
```