Clock

Introduction

Accurately simulating time-dependent behavior is essential for verifying the correctness of applications. Utilizing Clock functionality allows developers to manipulate and control time within tests, enabling the precise validation of features such as rendering time, timeouts, scheduled tasks without the delays and variability of real-time execution.

The Clock API provides the following methods to control time:

- `setFixedTime`: Sets the fixed time for `Date.now()` and `new Date()`.
- `install`: initializes the clock and allows you to:
  - `pauseAt`: Pauses the time at a specific time.
  - `fastForward`: Fast forwards the time.
  - `runFor`: Runs the time for a specific duration.
  - `resume`: Resumes the time.
- `setSystemTime`: Sets the current system time.

The recommended approach is to use `setFixedTime` to set the time to a specific value. If that doesn't work for your use case, you can use `install` which allows you to pause time later on, fast forward it, tick it, etc. `setSystemTime` is only recommended for advanced use cases.

NOTE

page.clock overrides native global classes and functions related to time allowing them to be manually controlled: - `Date` - `setTimeout` - `clearTimeout` - `setInterval` - `clearInterval` - `requestAnimationFrame` - `cancelAnimationFrame` - `requestIdleCallback` - `cancelIdleCallback` - `performance` - `Event.timeStamp`

Test with predefined time

Often you only need to fake `Date.now` while keeping the timers going. That way the time flows naturally, but `Date.now` always returns a fixed value.

```
<div id="current-time" data-testid="current-time"></div>
<script>
  const renderTime = () => {
    document.getElementById('current-time').textContent =
```

```
        new Date().toLocaleString();
  };
  setInterval(renderTime, 1000);
</script>
await page.clock.setFixedTime(new Date('2024-02-02T10:00:00'));
await page.goto('http://localhost:3333');
await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:00:00 AM');

await page.clock.setFixedTime(new Date('2024-02-02T10:30:00'));
// We know that the page has a timer that updates the time every second.
await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:30:00 AM');
```

Consistent time and timers

Sometimes your timers depend on `Date.now` and are confused when the `Date.now` value does not change over time. In this case, you can install the clock and fast forward to the time of interest when testing.

```
<div id="current-time" data-testid="current-time"></div>
<script>
  const renderTime = () => {
    document.getElementById('current-time').textContent =
        new Date().toLocaleString();
  };
  setInterval(renderTime, 1000);
</script>
// Initialize clock with some time before the test time and let the page
load
// naturally. `Date.now` will progress as the timers fire.
await page.clock.install({ time: new Date('2024-02-02T08:00:00') });
await page.goto('http://localhost:3333');

// Pretend that the user closed the laptop lid and opened it again at 10am,
// Pause the time once reached that point.
await page.clock.pauseAt(new Date('2024-02-02T10:00:00'));

// Assert the page state.
await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:00:00 AM');

// Close the laptop lid again and open it at 10:30am.
await page.clock.fastForward('30:00');
await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:30:00 AM');
```

Test inactivity monitoring

Inactivity monitoring is a common feature in web applications that logs out users after a period of inactivity. Testing this feature can be tricky because you need to wait for a long time to see the effect. With the help of the clock, you can speed up time and test this feature quickly.

```
<div id="remaining-time" data-testid="remaining-time"></div>
<script>
```

```
  const endTime = Date.now() + 5 * 60_000;
  const renderTime = () => {
    const diffInSeconds = Math.round((endTime - Date.now()) / 1000);
    if (diffInSeconds <= 0) {
      document.getElementById('remaining-time').textContent =
        'You have been logged out due to inactivity.';
    } else {
      document.getElementById('remaining-time').textContent =
        `You will be logged out in ${diffInSeconds} seconds.`;
    }
    setTimeout(renderTime, 1000);
  };
  renderTime();
</script>
<button type="button">Interaction</button>
// Initial time does not matter for the test, so we can pick current time.
await page.clock.install();
await page.goto('http://localhost:3333');
// Interact with the page
await page.getByRole('button').click();

// Fast forward time 5 minutes as if the user did not do anything.
// Fast forward is like closing the laptop lid and opening it after 5
minutes.
// All the timers due will fire once immediately, as in the real browser.
await page.clock.fastForward('05:00');

// Check that the user was logged out automatically.
await expect(page.getByText('You have been logged out due to
inactivity.')).toBeVisible();
```

Tick through time manually, firing all the timers consistently

In rare cases, you may want to tick through time manually, firing all timers and animation frames in the process to achieve a fine-grained control over the passage of time.

```
<div id="current-time" data-testid="current-time"></div>
<script>
  const renderTime = () => {
    document.getElementById('current-time').textContent =
        new Date().toLocaleString();
  };
  setInterval(renderTime, 1000);
</script>
// Initialize clock with a specific time, let the page load naturally.
await page.clock.install({ time: new Date('2024-02-02T08:00:00') });
await page.goto('http://localhost:3333');

// Pause the time flow, stop the timers, you now have manual control
// over the page time.
await page.clock.pauseAt(new Date('2024-02-02T10:00:00'));
await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:00:00 AM');

// Tick through time manually, firing all timers in the process.
// In this case, time will be updated in the screen 2 times.
await page.clock.runFor(2000);
```

```
await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:00:02 AM');
```

Related Videos

Previous
Chrome extensions

Next
Components (experimental)

- Introduction
- Test with predefined time
- Consistent time and timers
- Test inactivity monitoring
- Tick through time manually, firing all the timers consistently
- Related Videos

Learn

- Getting started
- Playwright Training
- Learn Videos
- Feature Videos

Community

- Stack Overflow
- Discord
- Twitter
- LinkedIn

More

- GitHub
- YouTube
- Blog
- Ambassadors