Mocking Tauri APIs

When writing your frontend tests, having a "fake" Tauri environment to simulate windows or intercept IPC calls is common, so-called *mocking*. The `@tauri-apps/api/mocks` module provides some helpful tools to make this easier for you:

CAUTION

Remember to clear mocks after each test run to undo mock state changes between runs! See `clearMocks()` docs for more info.

IPC Requests

Most commonly, you want to intercept IPC requests; this can be helpful in a variety of situations:

- Ensure the correct backend calls are made
- Simulate different results from backend functions

Tauri provides the mockIPC function to intercept IPC requests. You can find more about the specific API in detail here.

NOTE

The following examples use Vitest, but you can use any other frontend testing library such as jest.

```
import { beforeAll, expect, test } from "vitest";
import { randomFillSync } from "crypto";

import { mockIPC } from "@tauri-apps/api/mocks";
import { invoke } from "@tauri-apps/api/tauri";

// jsdom doesn't come with a WebCrypto implementation
beforeAll(() => {
  Object.defineProperty(window, 'crypto', {
    value: {
      // @ts-ignore
      getRandomValues: (buffer) => {
        return randomFillSync(buffer);
      },
    },
  });
});


test("invoke simple", async () => {
  mockIPC((cmd, args) => {
    // simulated rust command called "add" that just adds two numbers
    if(cmd === "add") {
      return (args.a as number) + (args.b as number);
    }
  });
});
```

Sometimes you want to track more information about an IPC call; how many times was the command invoked? Was it invoked at all? You can use `mockIPC()` with other spying and mocking tools to test this:

```
import { beforeAll, expect, test, vi } from "vitest";
import { randomFillSync } from "crypto";

import { mockIPC } from "@tauri-apps/api/mocks";
import { invoke } from "@tauri-apps/api/tauri";

// jsdom doesn't come with a WebCrypto implementation
beforeAll(() => {
  Object.defineProperty(window, 'crypto', {
    value: {
      // @ts-ignore
      getRandomValues: (buffer) => {
        return randomFillSync(buffer);
      },
    },
  });
});


test("invoke", async () => {
  mockIPC((cmd, args) => {
    // simulated rust command called "add" that just adds two numbers
    if(cmd === "add") {
      return (args.a as number) + (args.b as number);
    }
  });

  // we can use the spying tools provided by vitest to track the mocked function
  const spy = vi.spyOn(window, "__TAURI_IPC__");
  expect(invoke("add", { a: 12, b: 15 })).resolves.toBe(27);
  expect(spy).toHaveBeenCalled();
});
```

To mock IPC requests to a sidecar or shell command you need to grab the ID of the event handler when `spawn()` or `execute()` is called and use this ID to emit events the backend would send back:

```
mockIPC(async (cmd, args) => {
  if (args.message.cmd === 'execute') {
    const eventCallbackId = `_${args.message.onEventFn}`;
    const eventEmitter = window[eventCallbackId];

    // 'Stdout' event can be called multiple times
    eventEmitter({
      event: 'Stdout',
      payload: 'some data sent from the process',
    });

    // 'Terminated' event must be called at the end to resolve the promise
    eventEmitter({
      event: 'Terminated',
      payload: {
        code: 0,
        signal: 'kill',
      },
    });
  }
});
```

Windows

Sometimes you have window-specific code (a splash screen window, for example), so you need to simulate different windows. You can use the `mockWindows()` method to create fake window labels. The first string identifies the "current" window (i.e., the window your JavaScript believes itself in), and all other strings are treated as additional windows.

NOTE

`mockWindows()` only fakes the existence of windows but no window properties. To simulate window properties, you need to intercept the correct calls using `mockIPC()`

```
import { beforeAll, expect, test } from 'vitest';
import { randomFillSync } from 'crypto';

import { mockWindows } from '@tauri-apps/api/mocks';

// jsdom doesn't come with a WebCrypto implementation
beforeAll(() => {
  Object.defineProperty(window, 'crypto', {
    value: {
      // @ts-ignore
      getRandomValues: (buffer) => {
        return randomFillSync(buffer);
      },
    },
  });
});

test('invoke', async () => {
  mockWindows('main', 'second', 'third');

  const { getCurrent, getAll } = await import('@tauri-apps/api/window');

  expect(getCurrent()).toHaveProperty('label', 'main');
  expect(getAll().map((w) => w.label)).toEqual(['main', 'second', 'third']);
});
```

Edit this page

*Last updated on **Apr 28, 2023***