



WebdriverIO

! EXAMPLE APPLICATION

This [WebdriverIO](#) guide expects you to have already gone through the [example Application setup](#) to follow step-by-step. The general information may still be helpful otherwise.

This WebDriver testing example will use [WebdriverIO](#), and its testing suite. It is expected to have Node.js already installed, along with `npm` or `yarn` although the [finished example project](#) uses `yarn`.

Create a Directory for the Tests

Let's create a space to write these tests in our project. We will be using a nested directory for this example project as we will later also go over other frameworks, but typically you only need to use one. Create the directory we will use with `mkdir -p webdriver/webdriverio`. The rest of this guide assumes you are inside the `webdriver/webdriverio` directory.

Initializing a WebdriverIO Project

We will be using a pre-existing `package.json` to bootstrap this test suite because we have already chosen specific [WebdriverIO](#) config options and want to showcase a simple working solution. The bottom of this section has a collapsed guide on setting it up from scratch.

`package.json`:

```
{
  "name": "webdriverio",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "test": "wdio run wdio.conf.js"
  },
  "dependencies": {
    "@wdio/cli": "^7.9.1"
  },
  "devDependencies": {
    "@wdio/local-runner": "^7.9.1",
```

```
"@wdio/mocha-framework": "^7.9.1",
"@wdio/spec-reporter": "^7.9.0"
}
}
```

We have a script that runs a [WebdriverIO](#) config as a test suite exposed as the `test` command. We also have various dependencies added by the `@wdio/cli` command when we first set it up. In short, these dependencies are for the most simple setup using a local WebDriver runner, [Mocha](#) as the test framework, and a simple Spec Reporter.

► Click me if you want to see how to set a project up from scratch

Config

You may have noticed that the `test` script in our `package.json` mentions a file `wdio.conf.js`. That's the [WebdriverIO](#) config file which controls most aspects of our testing suite.

`wdio.conf.js`:

```
const os = require('os')
const path = require('path')
const { spawn, spawnSync } = require('child_process')

// keep track of the `tauri-driver` child process
let tauriDriver

exports.config = {
  specs: ['./test/specs/**/*.js'],
  maxInstances: 1,
  capabilities: [
    {
      maxInstances: 1,
      'tauri:options': {
        application: '../..target/release/hello-tauri-webdriver',
      },
    },
  ],
  reporters: ['spec'],
  framework: 'mocha',
  mochaOpts: {
    ui: 'bdd',
    timeout: 60000,
  },
}
```

```
// ensure the rust project is built since we expect this binary to exist for the webdriver
sessions
onPrepare: () => spawnSync('cargo', ['build', '--release']),

// ensure we are running `tauri-driver` before the session starts so that we can proxy the
webdriver requests
beforeSession: () =>
  (tauriDriver = spawn(
    path.resolve(os.homedir(), '.cargo', 'bin', 'tauri-driver'),
    [],
    { stdio: [null, process.stdout, process.stderr] }
  )),

// clean up the `tauri-driver` process we spawned at the start of the session
afterSession: () => tauriDriver.kill(),
}
```

If you are interested in the properties on the `exports.config` object, I [suggest reading the documentation](#). For non-WDIO specific items, there are comments explaining why we are running commands in `onPrepare`, `beforeSession`, and `afterSession`. We also have our specs set to `./test/specs/**/*.js`, so let's create a spec now.

Spec

A spec contains the code that is testing your actual application. The test runner will load these specs and automatically run them as it sees fit. Let's create our spec now in the directory we specified.

`test/specs/example.e2e.js`:

```
// calculates the luma from a hex color `#abcdef`
function luma(hex) {
  if (hex.startsWith('#')) {
    hex = hex.substring(1)
  }

  const rgb = parseInt(hex, 16)
  const r = (rgb >> 16) & 0xff
  const g = (rgb >> 8) & 0xff
  const b = (rgb >> 0) & 0xff
  return 0.2126 * r + 0.7152 * g + 0.0722 * b
}

describe('Hello Tauri', () => {
  it('should be cordial', async () => {
```

```

const header = await $('body > h1')
const text = await header.getText()
expect(text).toMatch(/^hHello/)
})

it('should be excited', async () => {
  const header = await $('body > h1')
  const text = await header.getText()
  expect(text).toMatch(/!$/)
})

it('should be easy on the eyes', async () => {
  const body = await $('body')
  const backgroundColor = await body.getCSSProperty('background-color')
  expect(luma(backgroundColor.parsed.hex)).toBeLessThan(100)
})
})

```

The `luma` function on top is just a helper function for one of our tests and is not related to the actual testing of the application. If you are familiar with other testing frameworks, you may notice similar functions being exposed that are used, such as `describe`, `it`, and `expect`. The other APIs, such as items like `$` and its exposed methods, are covered by the [WebdriverIO API docs](#).

Running the Test Suite

Now that we are all set up with config and a spec let's run it!

npm **Yarn**

```
$ yarn test
```

We should see output the following output:

```

→ webdriverio git:(main) X yarn test
yarn run v1.22.11
$ wdio run wdio.conf.js

```

```
Execution of 1 workers started at 2021-08-17T08:06:10.279Z
```

```

[0-0] RUNNING in undefined - /test/specs/example.e2e.js
[0-0] PASSED in undefined - /test/specs/example.e2e.js

```

"spec" Reporter:


```
-----  
[wry 0.12.1 linux #0-0] Running: wry (v0.12.1) on linux  
[wry 0.12.1 linux #0-0] Session ID: 81e0107b-4d38-4eed-9b10-ee80ca47bb83  
[wry 0.12.1 linux #0-0]  
[wry 0.12.1 linux #0-0] » /test/specs/example.e2e.js  
[wry 0.12.1 linux #0-0] Hello Tauri  
[wry 0.12.1 linux #0-0]     ✓ should be cordial  
[wry 0.12.1 linux #0-0]     ✓ should be excited  
[wry 0.12.1 linux #0-0]     ✓ should be easy on the eyes  
[wry 0.12.1 linux #0-0]  
[wry 0.12.1 linux #0-0] 3 passing (244ms)
```

Spec Files: 1 passed, 1 total (100% completed) in 00:00:01

Done in 1.98s.

We see the Spec Reporter tell us that all 3 tests from the `test/specs/example.e2e.js` file, along with the final report `Spec Files: 1 passed, 1 total (100% completed) in 00:00:01`.

Using the [WebdriverIO](#) test suite, we just easily enabled e2e testing for our Tauri application from just a few lines of configuration and a single command to run it! Even better, we didn't have to modify the application at all.

 [Edit this page](#)

Last updated on Feb 20, 2023