

Guide: Playwright

Introduction to Playwright for the ["JavaScript Applications" course@SoftUni](#).

1. Installation

Playwright enables fast, reliable and capable automation across all browsers.

- 1) First, we need to start the server:

```
node server.js
```

- 2) Write in terminal **npm init -y** to create **package.json**:

```
{
  "name": "Lab",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- 3) Use **npm** to install Playwright in your Node.js project:

```
npm install --save-dev playwright-chromium
```

```
{
  "name": "Lab",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "playwright-chromium": "^1.12.3"
  }
}
```

4) Install **chai** and **mocha** in your project:

```
npm install --save-dev chai
```

```
npm install --save-dev mocha
```

5) To **start** tests, write in terminal:

```
mocha {test file's name}.js
```

2. Executing the Tests

Before running the test suite, make sure a web server is operational, and the application can be found at the root of its network address. To start the included dev-server, open a terminal in the folder containing **package.json** and execute:

```
npm run start
```

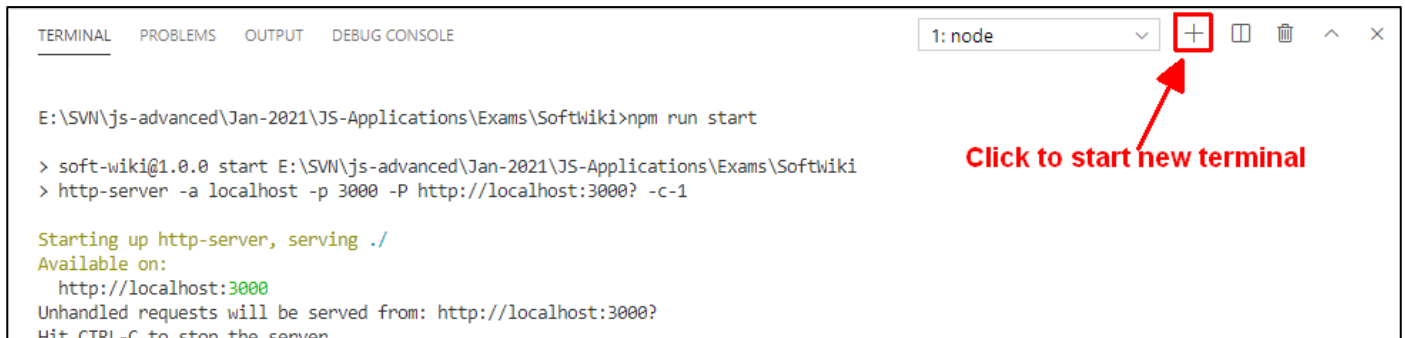
To work your **package.json** must look like the example:

```
{
  "name": "{name}",
  "version": "1.0.0",
  "description": "",
  "main": "index.html",
  "scripts": {
    "test": "mocha tests",
    "start": "http-server -a localhost -p 3000 -P http://localhost:3000? -c-1"
  }
}
```

This is a one-time operation unless you terminate the server at any point. It can be restarted with the same command as above.

To execute the tests, open a new terminal (do not close the terminal, running the web server instance) in the folder containing **package.json** and execute:

```
npm run test
```



```
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
1: node
E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki>npm run start
> soft-wiki@1.0.0 start E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki
> http-server -a localhost -p 3000 -P http://localhost:3000? -c-1

Starting up http-server, serving ./
Available on:
  http://localhost:3000
Unhandled requests will be served from: http://localhost:3000?
Hit CTRL-C to stop the server
```

Test results will be displayed in the terminal, along with detailed information about encountered problems. You can perform this operation as many times as it is necessary by re-running the above command.

3. Debugging Your Solution

If a test fails, you can view detailed information about the requirements that were not met by your application. Open the file **test.js** in the folder **tests** and navigate to the desired section as described below.

This first step will not be necessary if you are using the included web server. Make sure the application host is set correctly:

```
5 const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
6 const interval = 300;
7 const timeout = 6000;
8 const DEBUG = false;
9 const slowMo = 500;
```

The value for **host** must be the address where your application is being served. Make sure that entering this address in a regular internet browser shows your application.

To make just a single test run, instead of the full suite (useful when debugging a single failing test), find the test and append **.only** after the **it** reference:

```
62 it.only('register makes correct API call [ 5 Points ]', async () => {
63   const data = mockData.users[0];
64   const { post } = await createHandler(endpoints.register, { post: data });
65 }
```

On slower machines, some of the tests may require more time to complete. You can instruct the tests to run more slowly by slightly increasing the values for **interval** and **timeout**:

```
5 const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
6 const interval = 300;
7 const timeout = 6000;
8 const DEBUG = false;
9 const slowMo = 500;
```

Note that **interval** values greater than 500 and **timeout** values greater than 10000 are not recommended.

If this doesn't make the test pass, set the value of **DEBUG** to **true** and run the tests again – this will launch a browser instance and allow you to see what is being tested, what the test sees and where it fails (or hangs):

```
5 const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
6 const interval = 300;
7 const timeout = 6000;
8 const DEBUG = true;
9 const slowMo = 500;
```

If the actions are happening too fast, you can increase the value of **slowMo**. If the browser hangs, you can just close it and abort any remaining tests by focusing the terminal window and pressing **[Ctrl+C]** followed by the letter "y" and **[Enter]**.

The final thing to look for is the exact row where the test fails:

```
1) E2E tests
   Catalog [ 20 Points ]
     show details [ 5 Points ]:

AssertionError: expected true to be false
+ expected - actual

-true
+false

at Context.<anonymous> (tests\e2e.test.js:229:79)
```

Test failed at row 229

4. Example

- We have tests for 01. Accordion – Lab.

The main structure must look like:

```
const { chromium } = require('playwright-chromium');
const { expect } = require('chai');

let browser, page;

describe('E2E tests', function () {
  this.timeout(6000);
  before(async () => {
    browser = await chromium.launch({ handleless: false, slowMo: 500 });
  });
  after(async () => {
    await browser.close();
  });
  beforeEach(async () => {
    page = await browser.newPage();
  });
  afterEach(async () => {
    await page.close();
  });
})
```

This is a **sample structure** of what **testing groups** should look like. Don't forgets to **require chai** in the JavaScript file.

```
it('load static page', async () => {
  await page.goto('http://127.0.0.1:5500/01.%20Accordion/index.html');

  const content = await page.textContent('.accordion .head span');
  expect(content).to.contains('Scalable Vector Graphics');
});

it('toggles content', async () => {
  await page.goto('http://127.0.0.1:5500/01.%20Accordion/index.html');

  await page.click('#main>.accordion:first-child >> text=More');
  await page.waitForSelector('#main>.accordion:first-child >> .extra p');
  const visible = await page.isVisible('#main>.accordion:first-child >> .extra p');
  expect(visible).to.be.true;
});

it('toggles content', async () => {
  await page.goto('http://127.0.0.1:5500/01.%20Accordion/index.html');

  await page.click('#main>.accordion:first-child >> text=More');
  await page.waitForSelector('#main>.accordion:first-child >> .extra p');
  await page.click('#main>.accordion:first-child >> text=Less');

  const visible = await page.isVisible('#main>.accordion:first-child >> .extra p');
  expect(visible).to.be.false;
});
```

- After that, write in terminal and press Enter:

```
mocha test.js
```

```
E2E tests
  ✓ load static page (2530ms)
  ✓ toggles content (441ms)
  ✓ toggles content (533ms)

3 passing (5s)
```