**Setup:-**

[1] Install Visual Studio Code if not already installed.

[2] Open Visual Studio Code

[3] Create a new folder in Windows explorer where you want your test café solution to be created. Name folder-

test-café-testing

[4] In Visual Studio Code open Integrated Terminal using – CTRL + ‘

[5] Navigate to where your test café solution folder is

[6] Type –

*npm init*

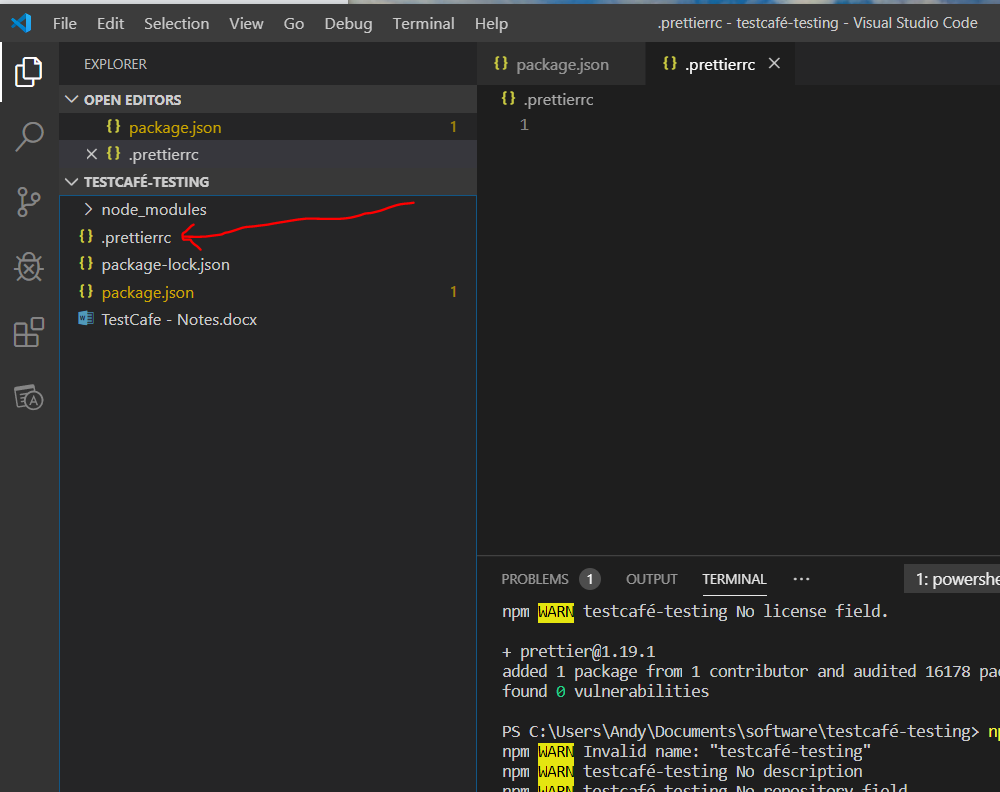
[7] Install TestCafe locally by typing –

*npm install testcafe*

[8] Now install Prettier:-

npm install prettier

[9] Create a file in the base of your folder called - .prettier. See blow:-



[10] Add this to .prettier file:-

{

    "singleQuote": true,

    "useTabs": true,

    "tabWidth": 2,

    "bracketSpacing": true,

    "arrowParens": "avoid",

    "trailingComma": "es5"

}

[11] Create a folder called:-

tests

then create a file in the folder called:-

basics.test.js

Add the following code to ‘basic.test.js’:-

import { Selector } from 'testcafe';

// prettier-ignore

fixture `Getting started with Testcafe`

    .page `https://devexpress.github.io/testcafe/example/`

test('My first testcafe test', async t => {

    // here goes testcafe code

    await t.typeText('#developer-name', "John")

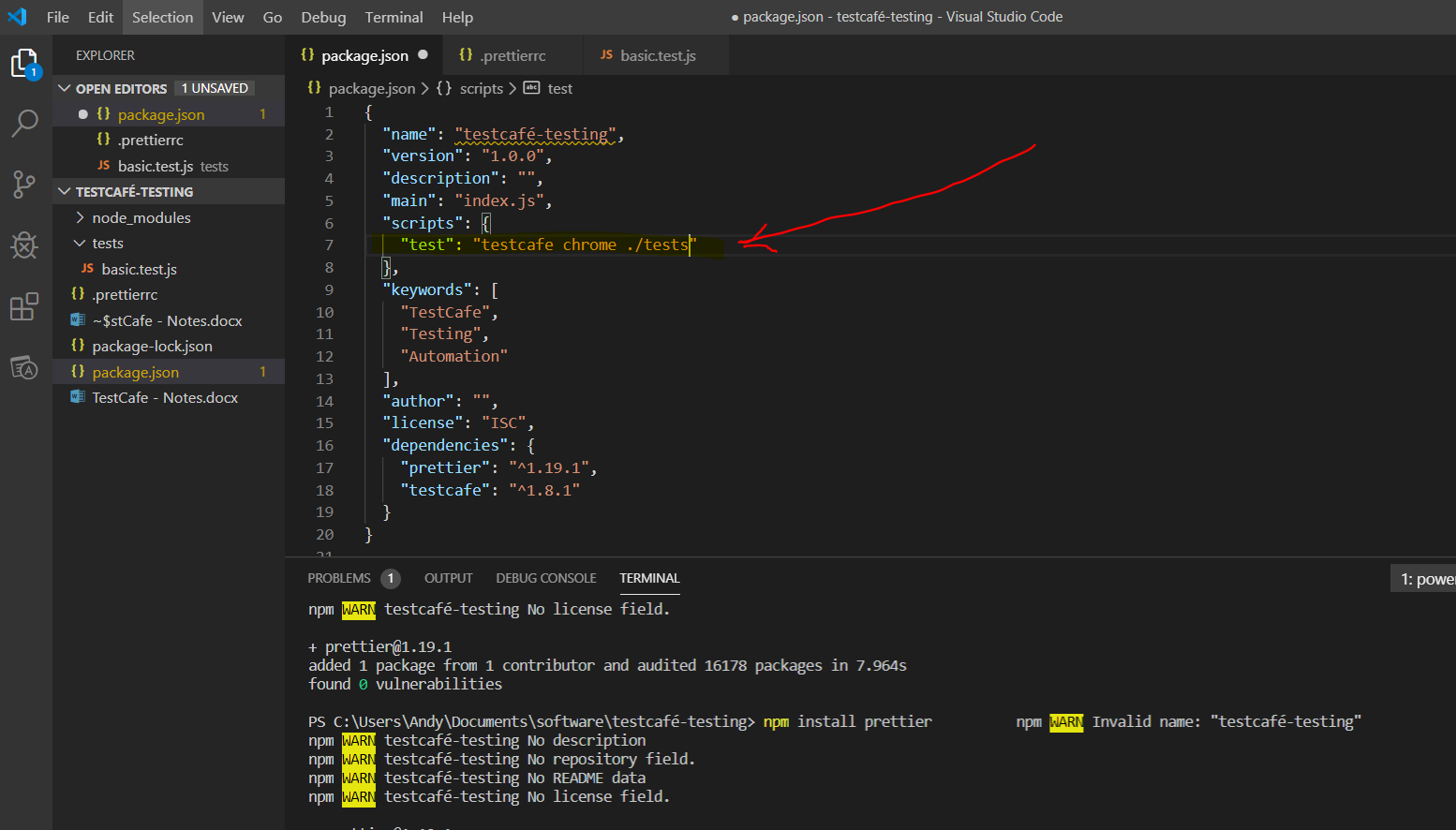
    await t.click('#submit-button')

})

[12] Use this web site for info on Testcafe:-

<https://devexpress.github.io/testcafe/example/>

[13] To run your tests you need to go to your ‘package.json’ file and enter the following:-



{

  "name": "testcafe-testing",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test:chrome": "testcafe chrome ./tests",

    "test:safari": "testcafe safari ./tests",

    "test:firefox": "testcafe firefox ./tests"

  },

  "keywords": [

    "TestCafe",

    "Testing",

    "Automation"

  ],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "prettier": "^1.19.1",

    "testcafe": "^1.8.1"

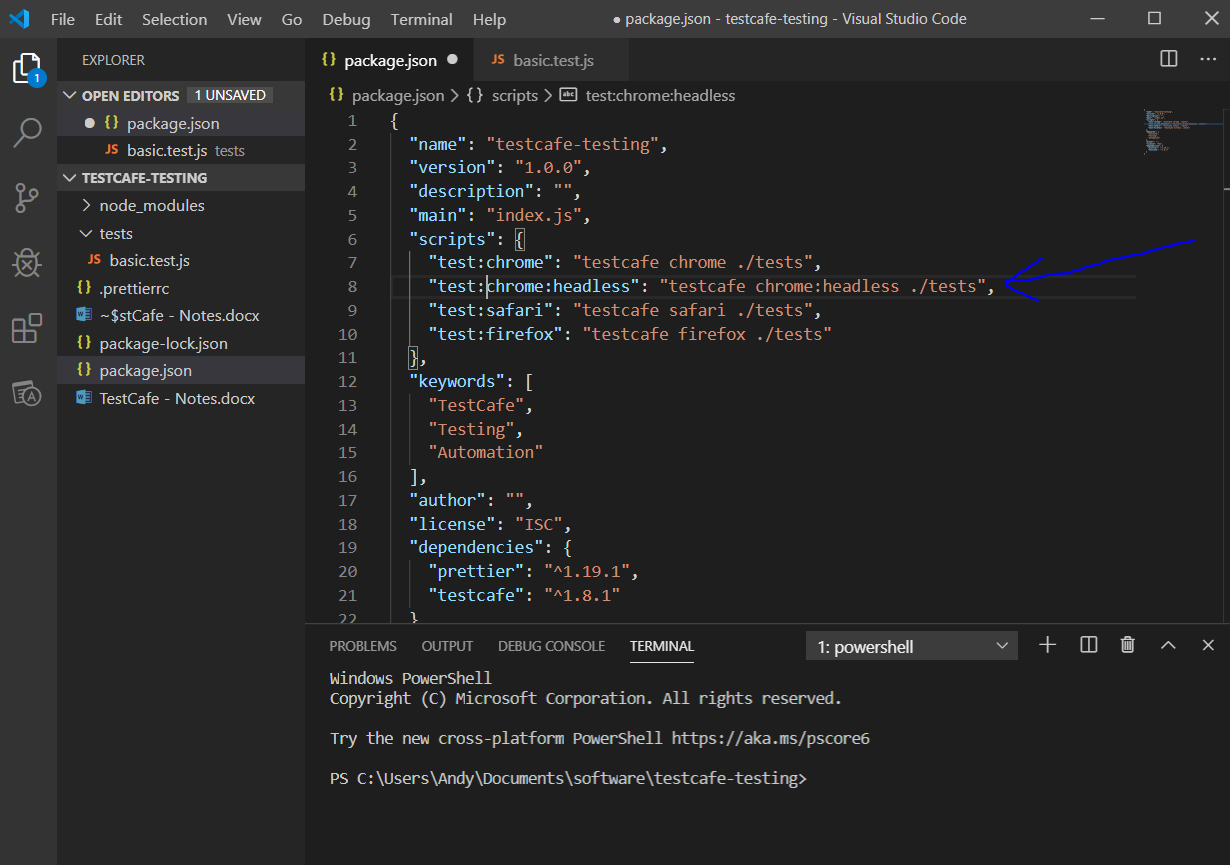
  }

}

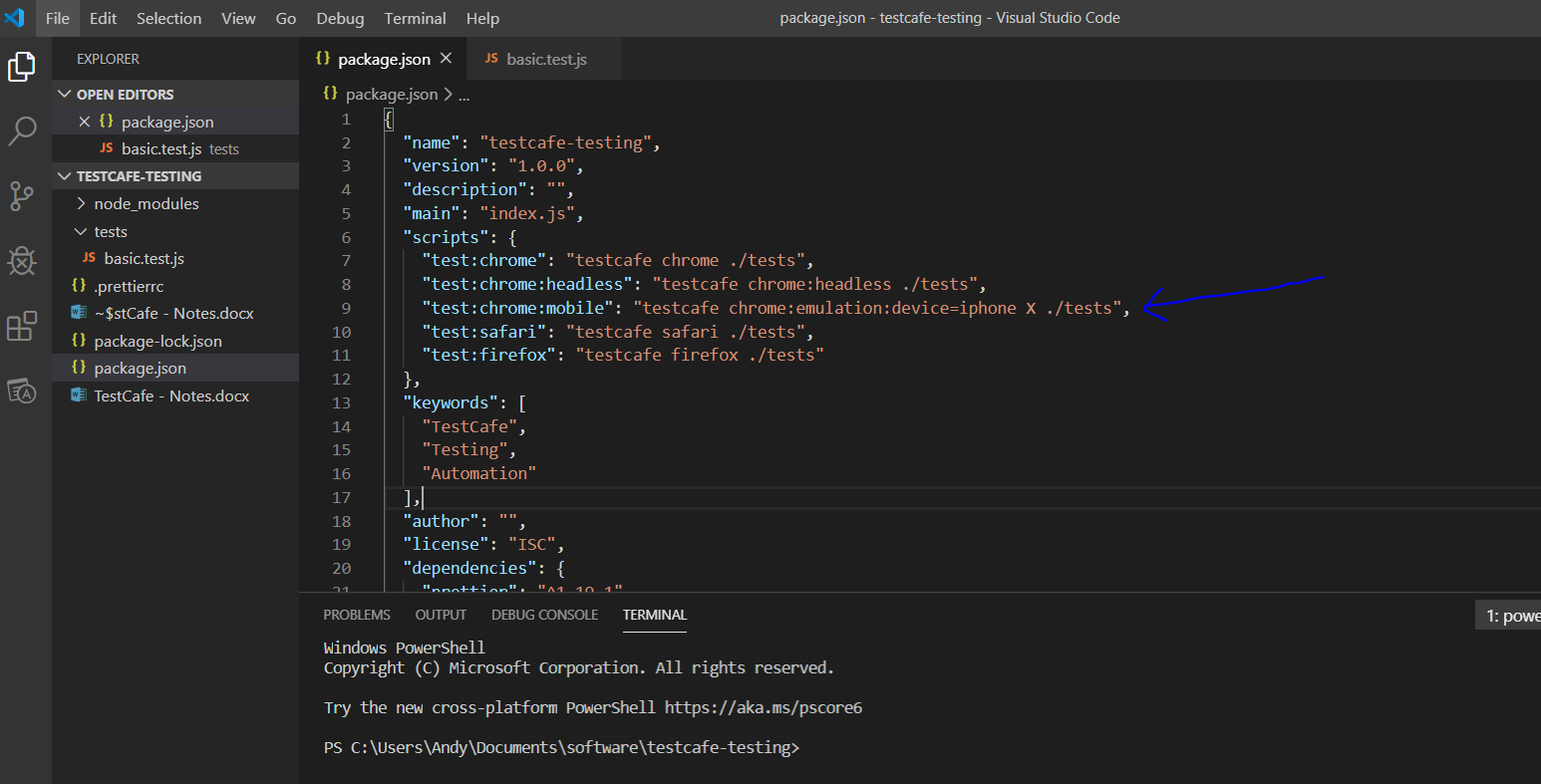
Now to run your tests type the following into the integrated terminal in visual studio code:-

npm run test:chrome

[14] Run tests in headless mode. With this no browser is open. Add the following line to the package.json file:-



[15] Add code to package.json file for a mobile device:-



To this type –

npm run test:chrome:mobile

[16] Take a screen shot when a test fails:-

Create a new file in the root of your project called:-

.testcaferc.json

Add the following to the .testcaferc.json file:-

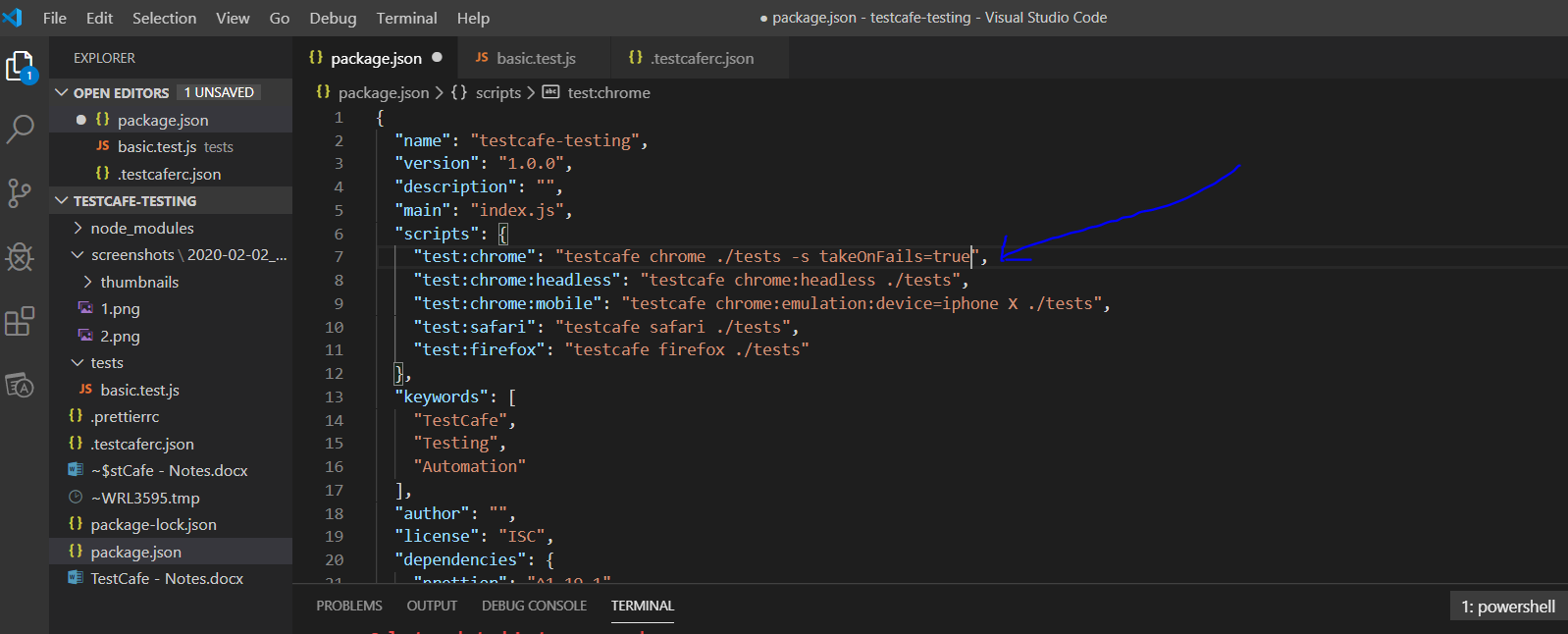
{

    "screenshotPath": "./screenshots",

    "screenshotPathPattern": "${DATE}\_${TIME}/${FIXTURE}.png"

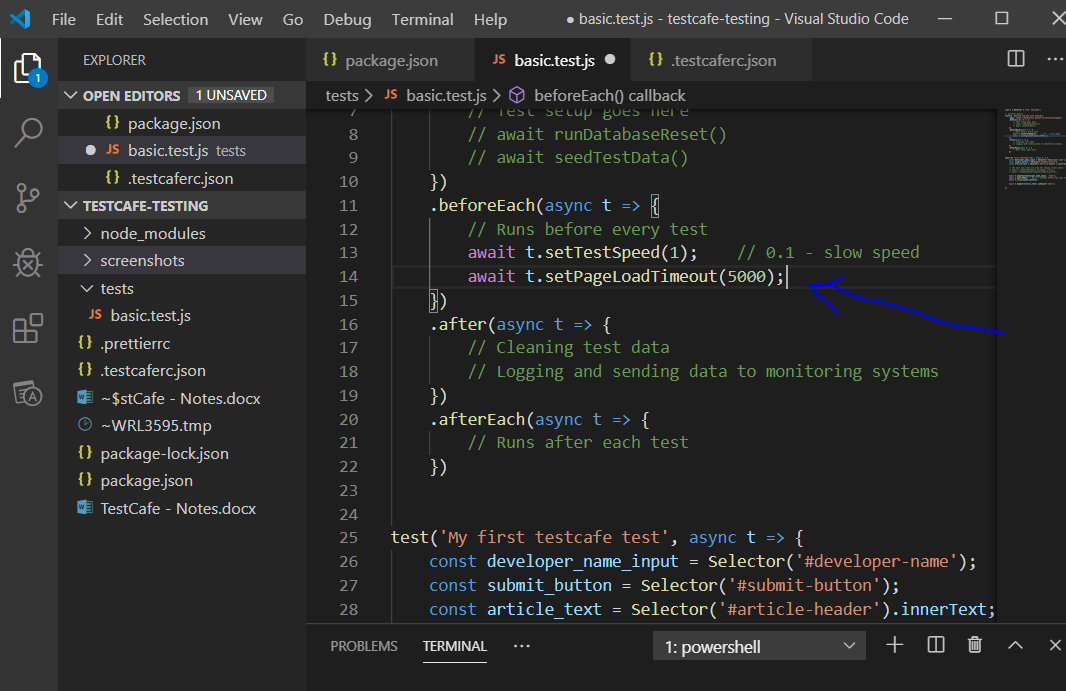
}

Add the following to the package.json file:-

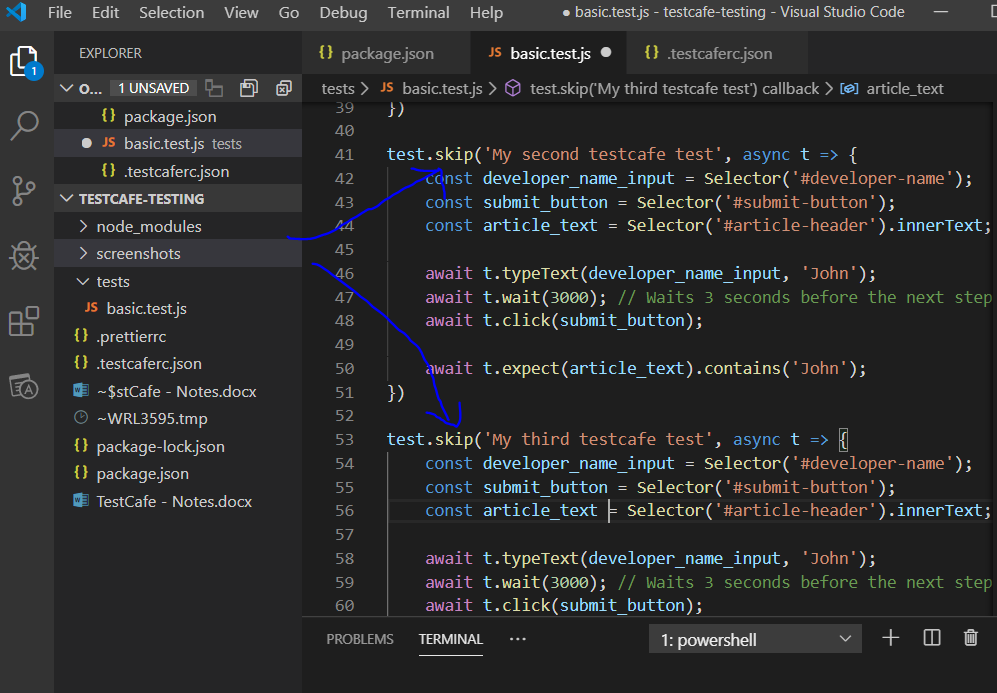


"test:chrome": "testcafe chrome ./tests -s takeOnFails=true",

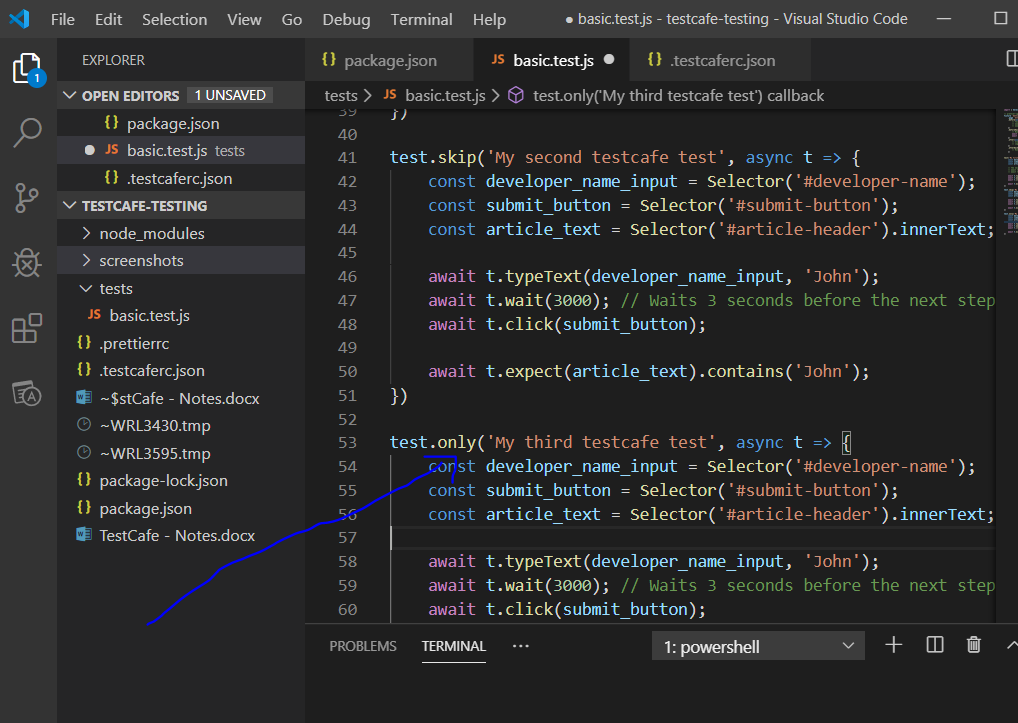
[17] Set page timeout:-



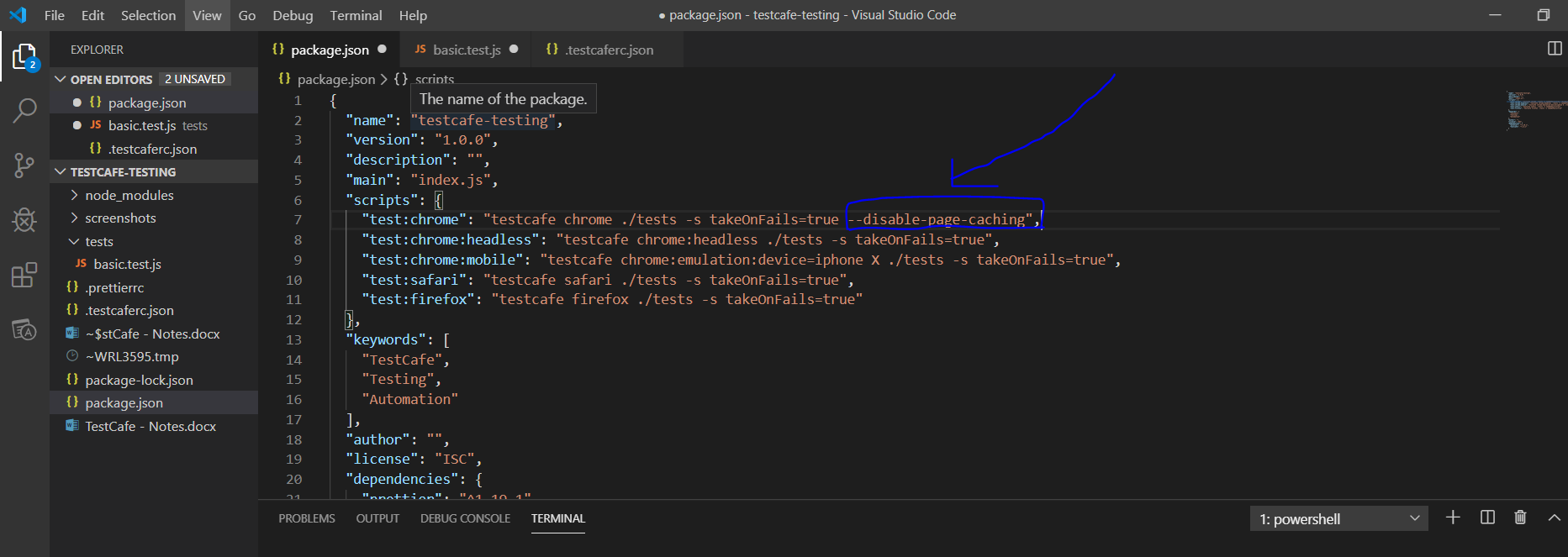
[18] If you have more than one test and you want to not runn all tests then use - .skip:-



To run only one test you can mark a test as .only:-



[19] Disable page caching by setting the following (see for the chrome browser):-



[20] Testcafe API functions:-

// Testcafe API function:

    // Click

    await t.click('select', { option });

    // Double Click

    await t.doubleClick('selector', { option });

    // Right Click

    await t.rightClick('selector', { option });

    // Drag Element

    await t.drag('selector', 200, 0, {option });

    // Hover

    await t.hover('selector', { option });

    // Select Text

    await t.selectText('selector');

    // Type Text

    await t.typeText('selector', 'write text');

    // Press Key

    await t.pressKey('backspace');

    // Navigate

    await t.navigateTo('url');

    // Take screenshot

    await t.takeScreenshot()

    await t.takeElementScreenshot()

    // Resize Window

    await t.resizeWindow(800, 600);

[21] Assertions:-

// Testcafe Assertions:-

    // Deep Equal - must match on type and value

    await t.expect('foo').eql('foo', 'message', options);

    // Not Deep Equal - must NOT match on type and value

    await t.expect('foo').notEql('foo');

    // OK - tests where the result between the expect brackets is TRUE

    await t.expect(true).ok();

    // Not OK  - tests where the result between the expect brackets is FALSE

    await t.expect(true).notOk();

    // Contains - checks if the thing between the expect() brackets contains 'o'. If it does then it passes

    await t.expect('foo').contains('o');

    // Not Contains - checks if the thing between the expect() brackets does not contain 'o'. If it does NOT then it passes

    await t.expect('foo').notContains('o');

    // Greater Than. If the value between the expect() bracket is greater than 5, then the test passes

    await t.expect(10).gt(5);

    // Greater Than or Equal to. If the value between the expect() bracket is greater than or equal to 10, then the test passes

    await t.expect(10).gte(10);

    // Less Than. If the value between the expect() bracket is less than 10, then the test passes

    await t.expect(5).lt(10);

    // Less Than or Equal to. If the value between the expect() bracket is less than or equal to 10, then the test passes

    await t.expect(10).lte(10);

    // Within - If the value between the expect() brackets is between 1 and 100, then the test passes

    await t.expect(10).within(1, 100);

    // Not Within - If the value between the expect() brackets is not between 5 and 50, then the test passes

    await t.expect(10).notWithin(5, 50);

[22] End To End Testing. Using the following web application:-

<http://zero.webappsecurity.com/index.html>

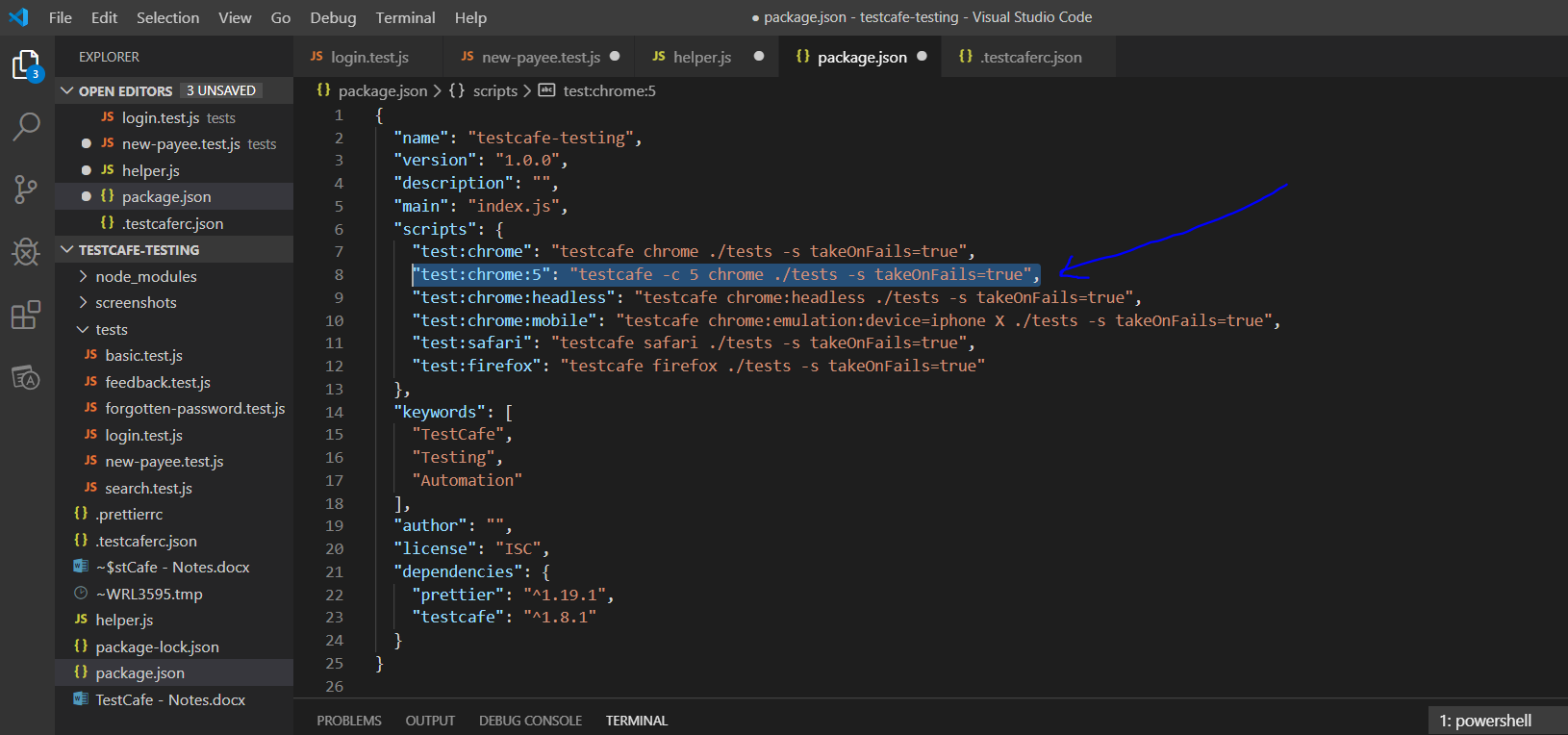
Login to application:-

[23] All parallel testing:-

Go to package.json file

To spin up 5 browser consecutively add the following line of code:-

"test:chrome:5": "testcafe -c 5 chrome ./tests -s takeOnFails=true",



[24] Execute tests headless: -

Go to package.json and add the following line:-

"test:chrome:headless:5":"testcafe -c 5 chrome:headless ./tests -s takeonFails=true",

[25] Fire up multiple different types of browsers:-

Go to package.json and add the following line:-

"test:multiple": "testcafe chrome,firefox ./tests -s takeOnFails=true"

[26] Test Café does not use XPath selectors, therefore you need to install the following package:-

npm install --save-dev xpath-to-ccs

Add the following code:-

import xPathToCss from 'xpath-to-css';

    // XPATH TO CSS EXAMPLE

    const xPath = `'//div[@id="foo"[2]/span[@class="bar"]//a[contains(@class, "baz")]//img[1]'`;

    const css = xPathToCss(xPath);

    console.log(css);

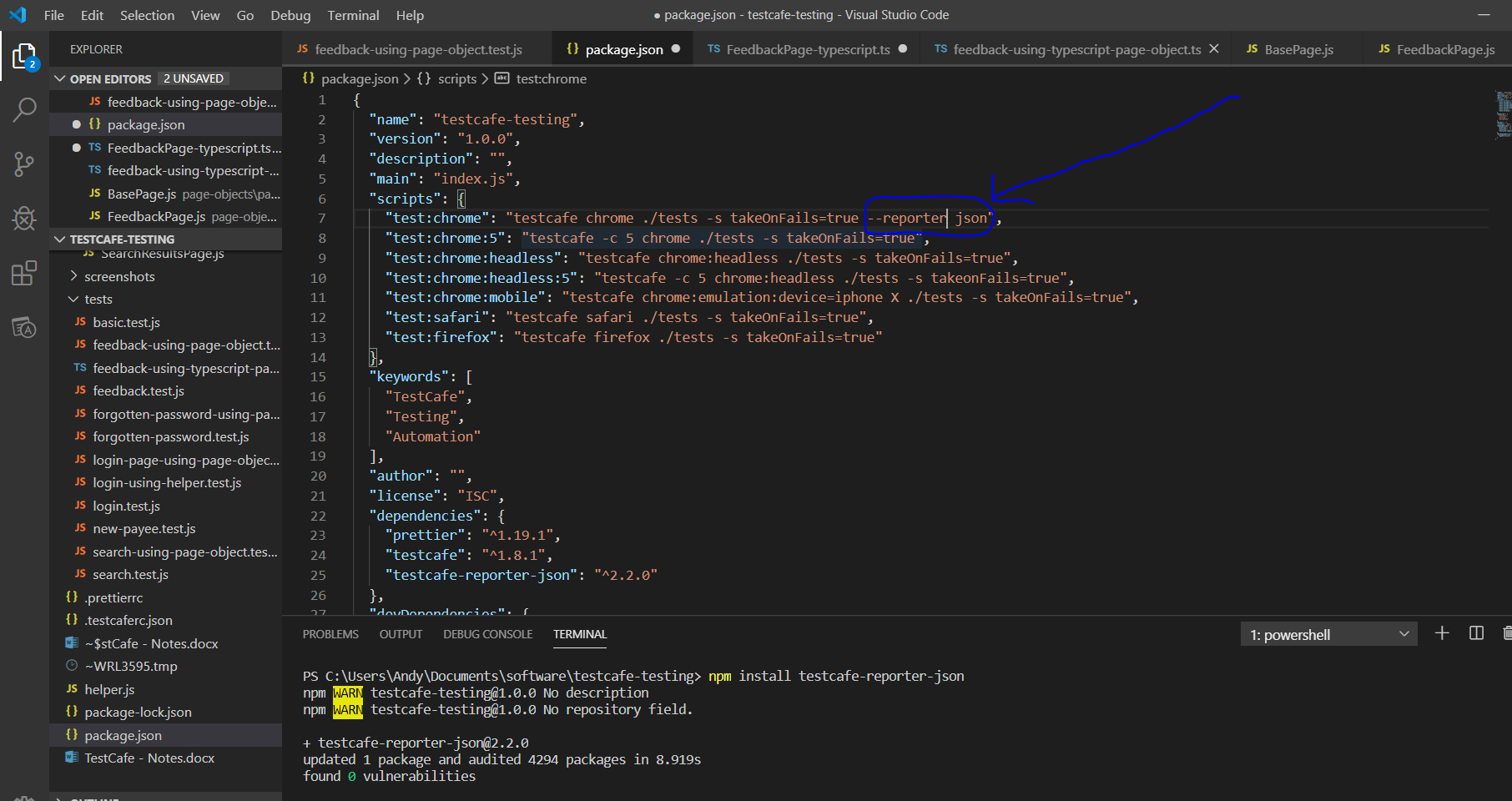
[27] Install Testcafe report packages:-

**testcafe-reporter-json**

Open integrated terminal and type the following:-

npm install testcafe-reporter-json

Then add the following to the ‘package.json’ file:-



When running your tests by typing:-

npm run test:chrome

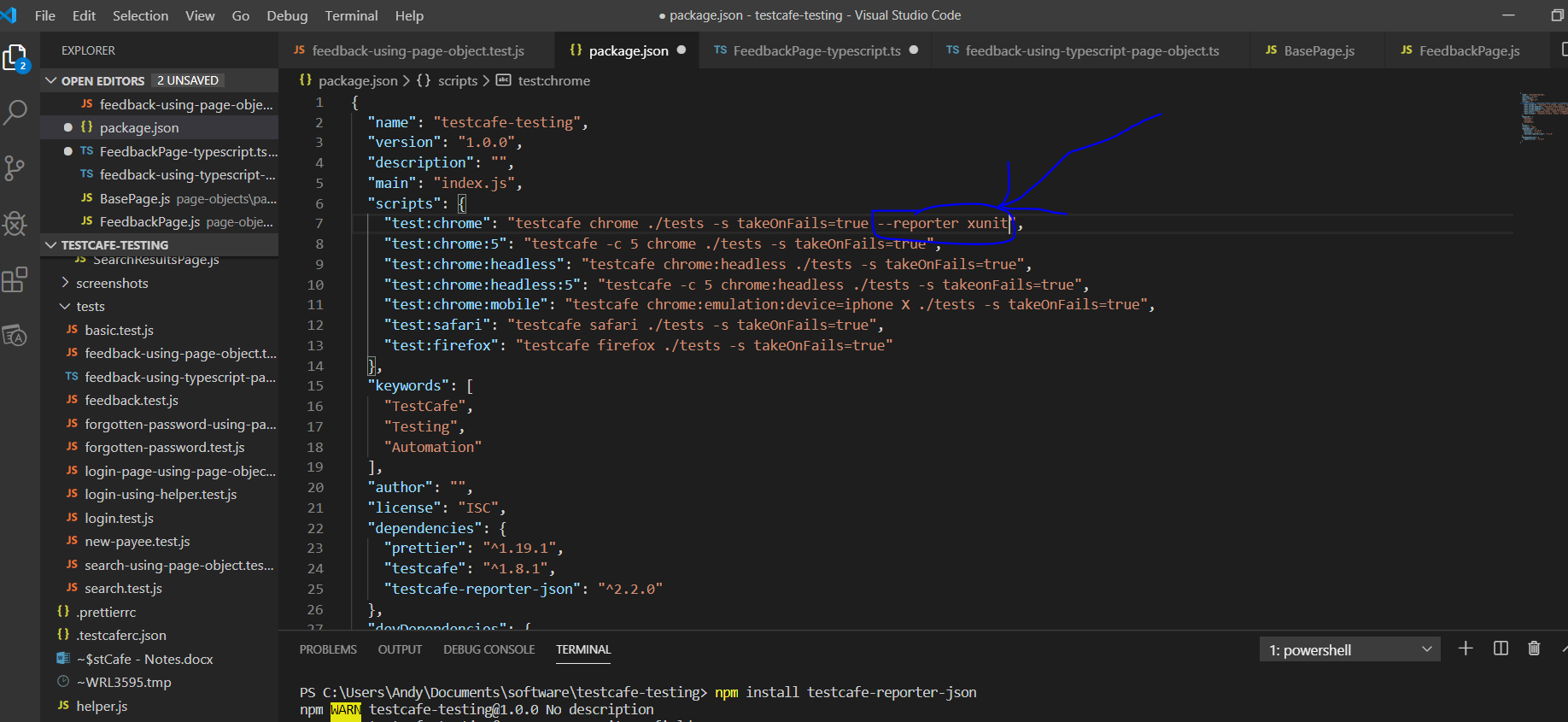
You will see the output of running the tests in the integrated terminal window.

**testcafe-reporter-xunit**

Open integrated terminal and type the following:-

npm install testcafe-reporter-xunit

Then add the following to the ‘package.json’ file:-



Run tests by typing the following into the terminal window:-

npm run test:chrome

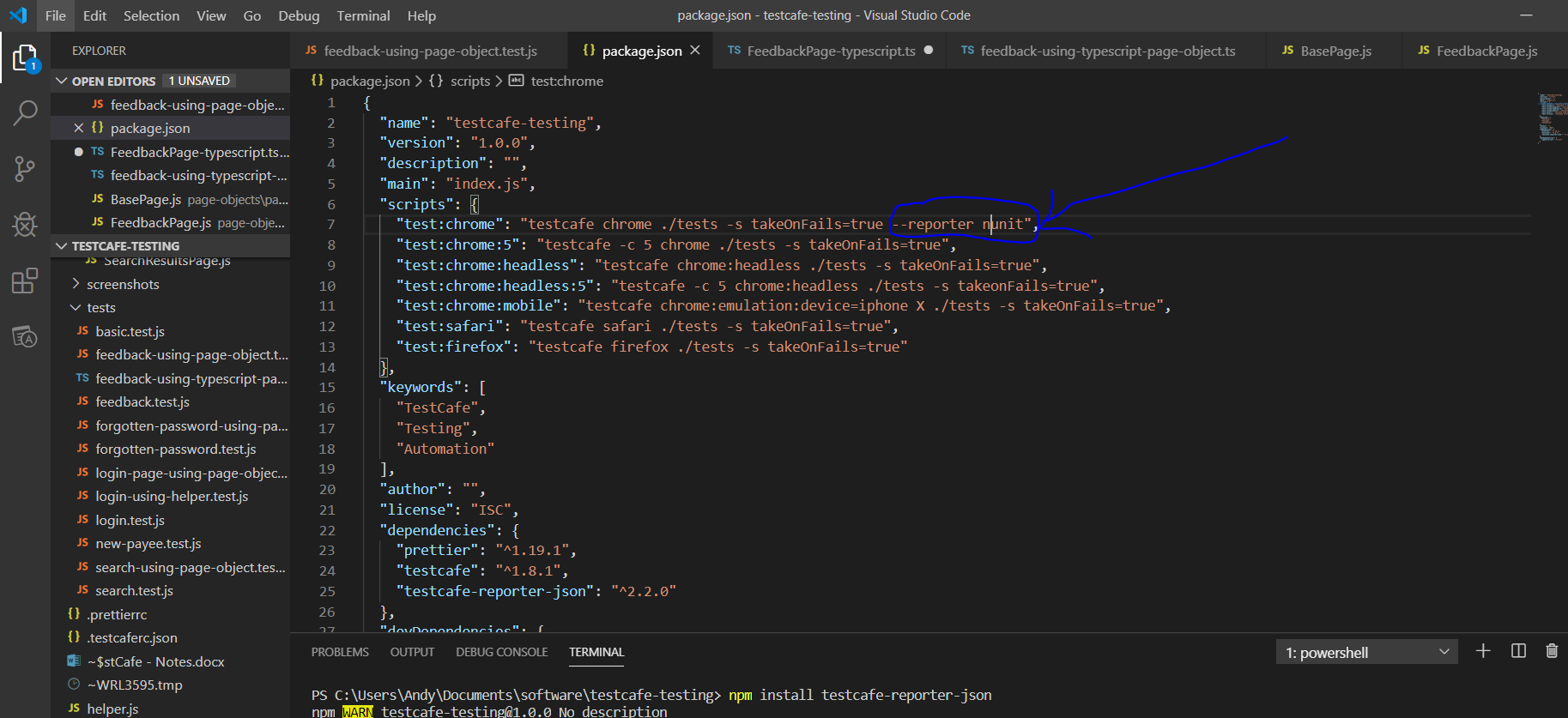
This creates xml formatted test results in the integrated terminal window.

**testcafe-reporter-nunit**

Open integrated terminal and type the following:-

npm install testcafe-reporter-nunit

Then add the following to the ‘package.json’ file:-

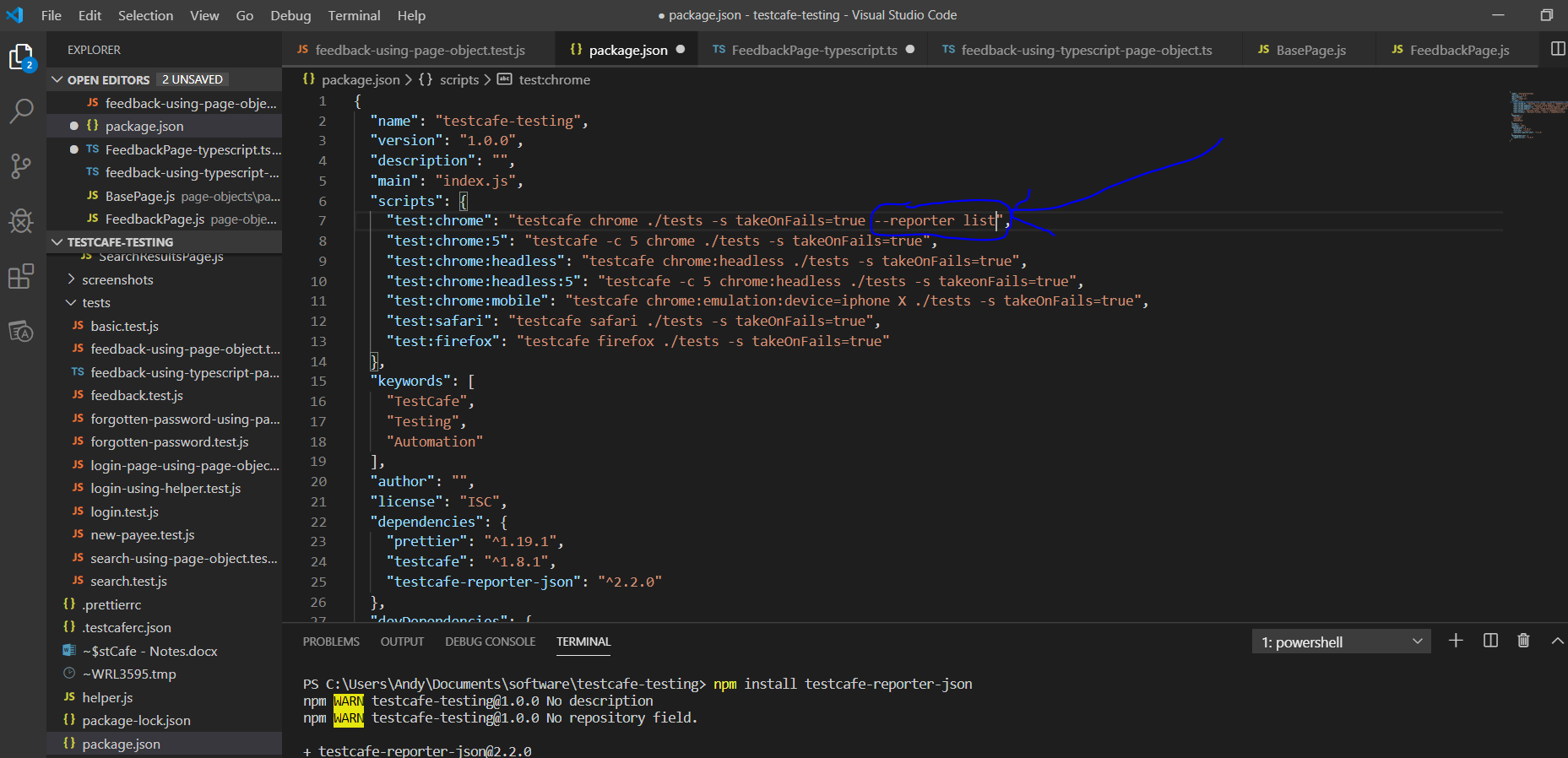


**testcafe-reporter-list**

Open integrated terminal and type the following:-

npm install testcafe-reporter-list

Then add the following to the ‘package.json’ file:-

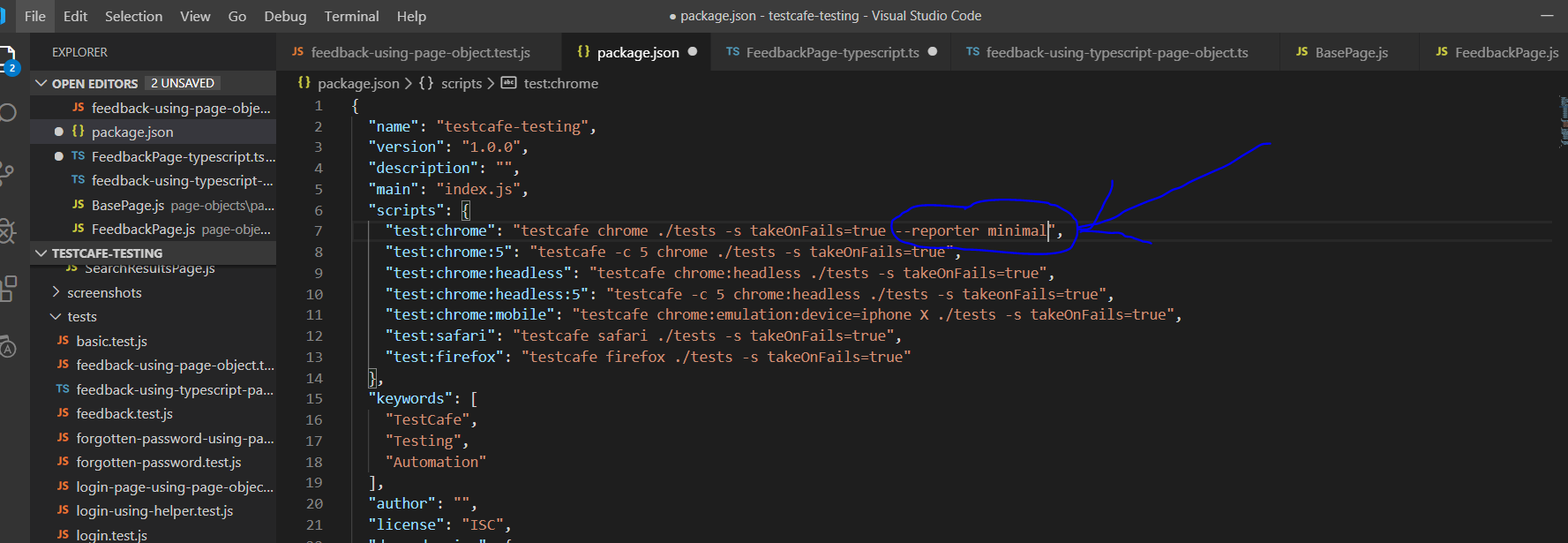


**testcafe-reporter-minimal**

Open integrated terminal and type the following:-

npm install testcafe-reporter-minimal

Then add the following to the ‘package.json’ file:-



[29] Add ids to html elements. If possible use this (data-test-id=””):-

<form data-test-id=”login-form”>

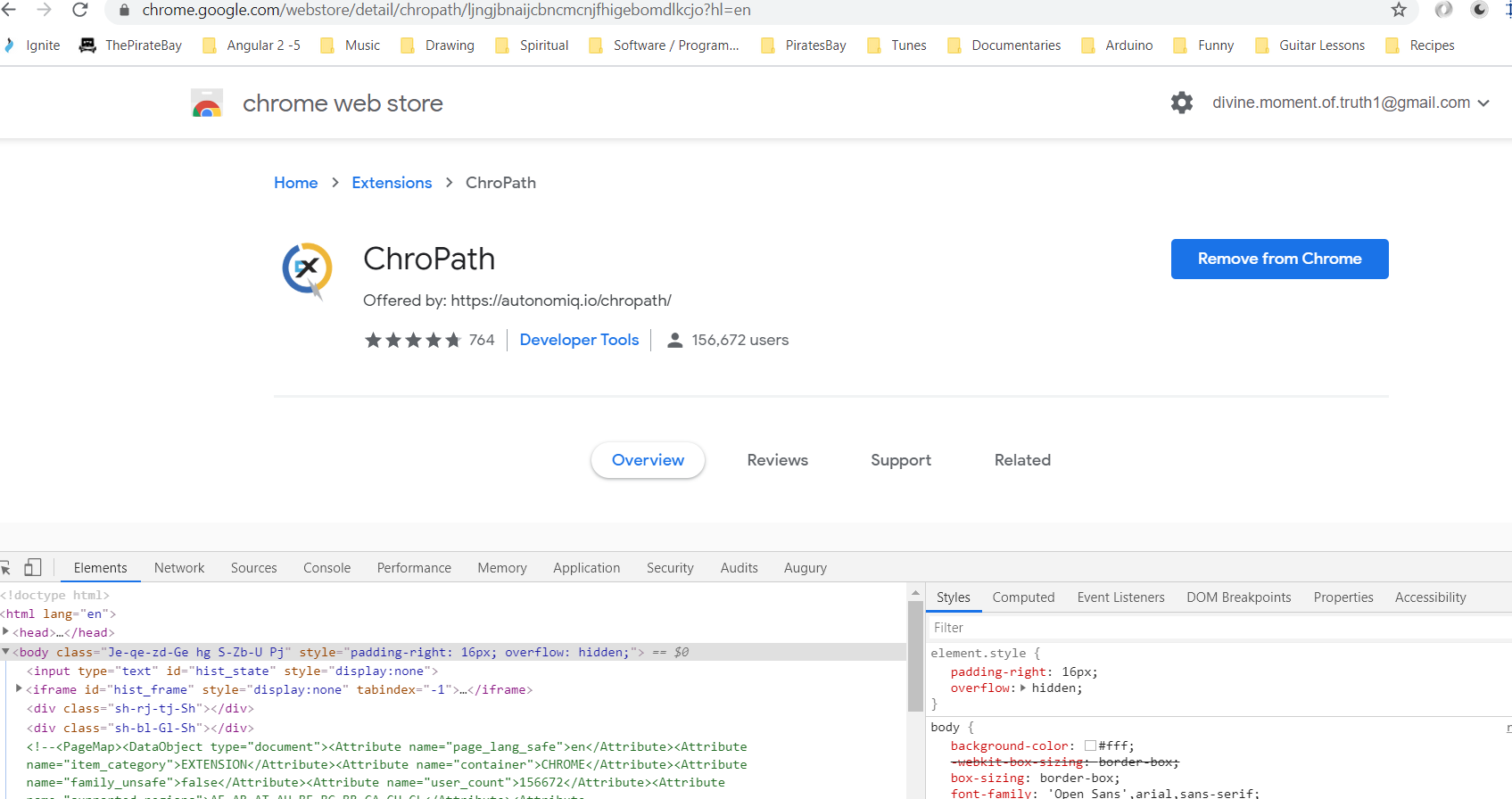
<button data-test-id=”btn-menu” class=”btn-large”>Menu</button>

<button data-test-id=”btn-maps” class=”btn-large”>Maps</button>

<button data-test-id=”btn-user” class=”btn-large”>Users</button>

</form>

[30] Download the ChroPath extension for your browser:-



Download Mocha

Download Chai

Download the following mobile testing:-

BrowserStack

Download the following for free CI/CD pipelines:-

Circleci - Best for small scale projects

Travis-ci - Best when using open source programs that will be tested in different environments

Jenkins -