WebdriverIO framework with JavaScript

```
npm init
(npm init wdio  .)
(npm init wdio@latest .)
(npm init wdio@latest ./path/to/new/project)
(npm init wdio@latest . -- --yes)
code .
Pwd -> present working directory in Mac
npx wdio --version # prints e.g. `8.13.10`
npm install @wdio/cli
npx wdio config                                          # run configuration wizard
npm.cmd pkg set scripts.wdio="wdio run ./wdio.conf.ts"

npm install
npm i ts-node --save-dev
npm i wdio-chromedriver-service --save-dev
npm i chromedriver --save-dev
npm i @wdio/cli@latest --save-dev
npm i typescript --save-dev
npm install @types/webdriverio --save-dev
npm install @wdio/mocha-framework --save-dev
npm install @wdio/jasmine-framework --save-dev
npm install @wdio/cucumber-framework --save-dev
npm install --save-dev chai
npm i deepmerge --save-dev
npm install @wdio/allure-reporter --save-dev
sudo npm install -g allure-commandline
allure generate allure-results && allure open

const { expect } = require("expect-webdriverio")

PACKAGE.JSON:-
"devDependencies": {
   "@wdio/allure-reporter": "^9.1.3",
   "@wdio/cli": "^9.2.1",
   "@wdio/cucumber-framework": "^9.1.3",
   "@wdio/local-runner": "^9.2.1",
   "@wdio/spec-reporter": "^9.1.3",
   "chromedriver": "^129.0.4",
   "ts-node": "^10.9.2",
   "typescript": "^5.6.3",
   "wdio-chromedriver-service": "^8.1.1"
 }

TSCONFIG.JSON:-
"compilerOptions": {
```

"types": ["node", "@wdio/globals/types", "webdriverio/async", "expect-webdriverio", "@wdio/cucumber-framework"]
}

#You can start your test suite by using the run command and pointing to the WebdriverIO config that you just created:
npx wdio run ./wdio.conf.ts
(npx wdio wdio.conf.ts)

Three framework options available in WebdriverIO

1. Mocha
2. Cucumber
3. Jasmine

WebdriverIO Runner has built-in support for Mocha, Jasmine, and Cucumber.js. You can also integrate it with 3rd-party open-source frameworks, such as Serenity/JS.

What is WebdriverIO?

WebdriverIO allows you to automate any application written with modern web frameworks such as React, Angular, Polymeror Vue.js as well as native mobile applications for Android and iOS.

WebdriverIO built on Node.js engine and uses JavaScript to code the Automation
WebdriverIO uses Selenium under wood. All the great things about selenium are available in WebdriverIO with
additional advantage of exclusive assertions for Test Validations

WebdriverIO uses additional features and is a wrapper on top of selenium to automate web applications
WebdriverIO uses additional features and is a wrapper on top of Appium to automate native mobile applications

>npm init wdio@latest .

webdriverio - jasmine, mocha, chai -> testing frameworks for javascript code
webdriverio code is wrapped under any of the test framework
webdriverio is a browser automation framework
JavaScript is Asynchronous

What is Asynchronous?

No sequence of execution of scripts

Each and every steps in javascript returns Promise which is an object
`Promise is having 3 status.`

```
a) Resolved
b) Pending
c) Rejected
```

This Promise will inform whether the step is resolved, pending or rejected.
Until Promise of a step is resolved control will not execute next steps.
Inorder to have a sequential execution, we have to wait until that Promise object has resolved status. Then we can confirm this step execution is fully completed.
To make synchronous sequential execution, we have to wait till Promise is resolved for which we can write a keyword called 'await'.
We need to use 'await' everywhere wherever we have webdriverio code.
Whenever we are using 'await' in your block, we have to treat and tell that function to use keyword 'async' which is a modifier

if we miss 'await', below is the output
```
Promise { <pending> }
```

To 'allow remote automation' in safari browser, run below command in terminal
```
safaridriver --enable
```

Chai assertions:

https://www.chaijs.com/
https://www.chaijs.com/guide/styles/
```
>npm install chai –save-dev
```

- Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.
- Chai provides additional helper assertion methods to webdriverIO.
- There will be conflict of using webdriverio expect & chai expect. Hence we need to manually import the chai library as below in our webdriverio code. Thereby we will have clarity of 'expect' is from webdriverio or chai

```
const expectchai = require('chai').expect
```
OR
```
import { expect as chaiExpect } from 'chai'
```

We need to use below code
```
chaiExpect(await dropdown.getValue()).to.eql('stud')
```
Instead of
```
expect(await dropdown.getValue()).to.equal('stud')
```

3. Data driven testing from Json files
4. Effective utilisation of Capabilities to run tests in parallel/ different browsers
5. Running selective tests using Mocha Grep options
6. Importance of Bail and Base URL Options in Configuration File
7. Creating Test Suites and controlling the execution with Command Line parameters
8. Building customised wdio configuration file to run tests with different options and environment
9. Retry mechanism in Mocha framework to run flaky tests
10. Generate rich HTML allure reports for test execution
11. Implement Pre and Post hooks to capture screenshots on Test failures
12. Integrate the complete WebdriverIO framework into Jenkins for Continuous Integration

Javascript getter:

```
const student = {

    // data property
    firstName: 'Monica',

    // accessor property(getter)
    get getName() {
        return this.firstName;
    }
};

// accessing data property
console.log(student.firstName); // Monica

// accessing getter methods
console.log(student.getName); // Monica

// trying to access as a method
console.log(student.getName()); // error
```

To export the class containing locators & actions and make it available outside the class using getters

```
class LoginPage {
   get username() {
      return $("input[name='username']")
   }
   get password() {
      return $("//input[@name='password']")
```

Screenshots:
Using below line we can capture screenshot in webdriverio and there is no condition like test should pass or fail in this case.
```
await browser.saveScreenshot("screenshot.png")
```

Scroll to element:
```
 await $("#mousehover").scrollIntoView()
```

Hover on to element:
```
 await $("#mousehover").moveTo()
```

LinkText & PartialLinkText selector in webdriverio
```
<a href="#top">Top</a>
```
Use $(“= Top”) as below:
```
await $("=Top").click()
```

SDET Unicorns by Dilpreet Johal

Example
HTML:
```
<a class="nav-link btn btn-primary"> Checkout ( 0 )
            <span class="sr-only">(current)</span>
 </a>
```
LinkText:
```
$("= Checkout ( 0 ) “).waitForExist()
```
PartialLinkText:
```
$(“*=Checkout”).waitForExist()
```

Promise.all()
It is nothing but array of Promises.
Currently Promise.all() not required

Previous:
```
const sumOfProducts = await Promise.all(await
productPrices.map(async (productPrice) => parseInt((await
productPrice.getText()).split('.')[1].trim()).reduce((acc,
price) => acc + price, 0)))
```

Current:
```
(await productPrices.map(async (productPrice) =>
parseInt((await productPrice.getText()).split('.')
[1].trim())).reduce((acc, price) => acc + price, 0))
```

WebdriverIO Framework Development
1. Implement Page object design pattern for the tests
2. Parameterize the test cases using Mocha Framework

```
    }
    get signIn() {
        return $("#signInBtn")
    }
    get alert() {
        return $(".alert-danger")
    }

    async Login(username, password) {
        await this.username.setValue(username)
        await this.password.setValue(password)
        await this.signIn.click()
    }
}

export default new LoginPage()
```

Previous:
a. Export from one .js file
`module.exports = new LoginPage()`
b. Import from other .js file
`const loginpage = require('../pageobjects/loginPage')`
C. To make require statement as supported in webdriverio, add below statement in package.json file
`"type": "commonjs"`

Current:
a. Export from one .js file
`export default new LoginPage()`
b. Import from other .js file
`import loginpage from "../pageobjects/loginPage"`
c. To load an ES module or to support export token, set "type": "module" in the package.json
`"type": "module"`

Parameterization & Data Driven Test from Json files

- loginTest.json file containing 2 sets of data having username & password

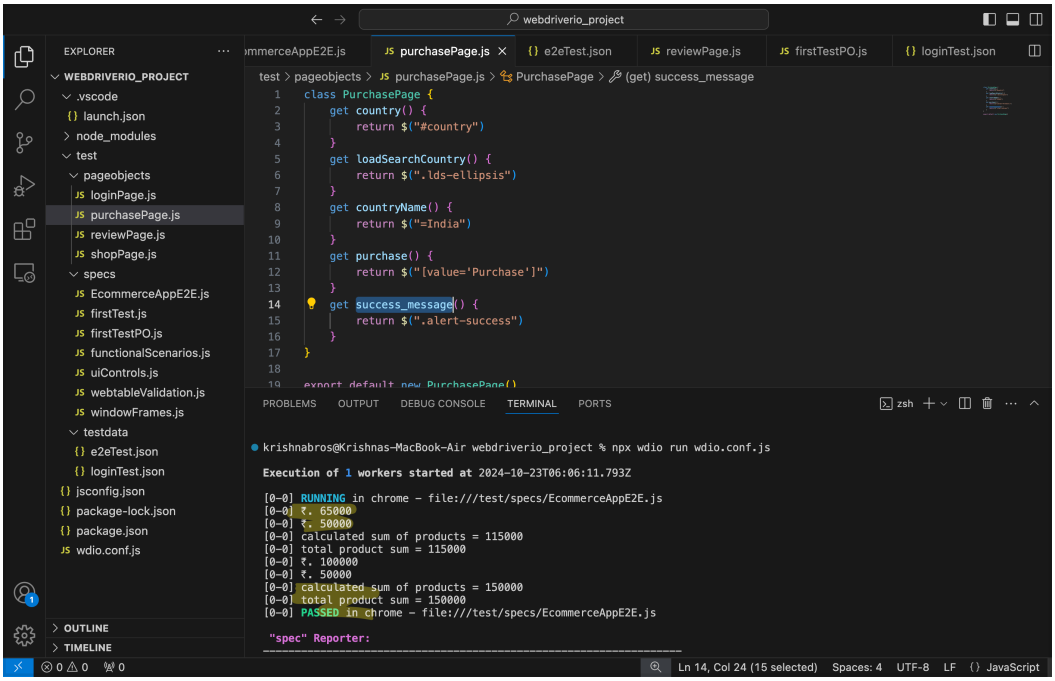
```
[
    {
        "username": "rahul",
        "password": "l113"
    },
    {
        "username": "rahul12",
        "password": "l1113@$"
    }
]
```

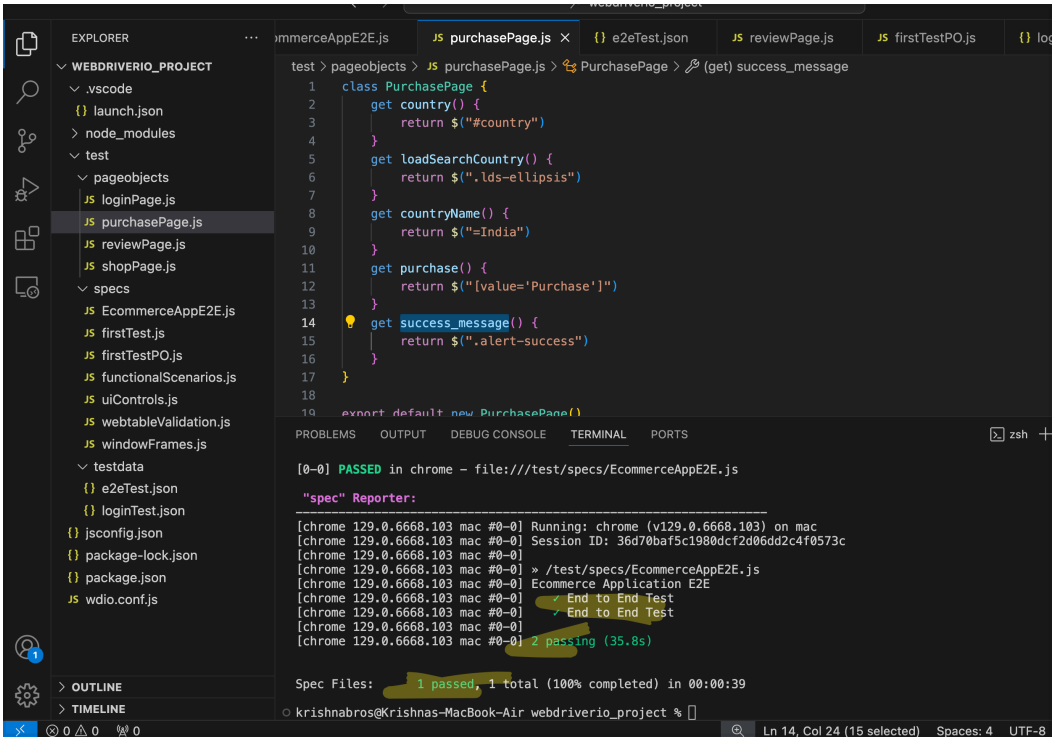
- Read JSON format and convert it into string format

```
import fs from "fs"
let credentials = JSON.parse(fs.readFileSync('test/testdata/
loginTest.json'))
```

- Usage in code:

```
describe('Ecommerce Application', async () => {
  credentials.forEach(({username, password}) => {
    it('Login fail page', async () => {
        await loginpage.Login(username, password)
     })
    })
 })
```





Capabilities:

Ref:

Capabilities | WebdriverIO
webdriver.io

Browser Specific Capability Extensions

- goog:chromeOptions: Chromedriver extensions, only applicable for testing in Chrome
- moz:firefoxOptions: Geckodriver extensions, only applicable for testing in Firefox
- ms:edgeOptions: EdgeOptions for specifying the environment when using EdgeDriver for testing Chromium Edge

Browser Specific Driver Options

In order to propagate options to the driver you can use the following custom capabilities:

- Chrome or Chromium: wdio:chromedriverOptions
- Firefox: wdio:geckodriverOptions
- Microsoft Egde: wdio:edgedriverOptions
- Safari: wdio:safaridriverOptions

To set up path to chromedriver in wdio.conf.js file

```
{
    browserName: 'chrome', // or 'chromium'
```

```
    'wdio:chromedriverOptions': {
        binary: '/path/to/chromedriver'
    }
}
```

Run Browser Headless

- Chrome

```
{
    browserName: 'chrome',   // or 'chromium'
    'goog:chromeOptions': {
        args: ['headless', 'disable-gpu']
    }
}
```

- Firefox

```
    browserName: 'firefox',
    'moz:firefoxOptions': {
        args: ['-headless']
    }
```

- Edge

```
browserName: 'msedge',
    'ms:edgeOptions': {
        args: ['--headless']
    }
```

- Safari

```
It seems that Safari doesn't support running in headless
mode.
```

Automate Different Browser Channels

1.Chrome

- When testing on Chrome, WebdriverIO will automatically download the desired browser version and driver for you based on the defined browserVersion, e.g.:

```
{
    browserName: 'chrome', // or 'chromium'
    browserVersion: '116' // or '116.0.5845.96', 'stable',
'latest', 'dev', 'canary', 'beta'
}
```

- If you like to test a manually downloaded browser, you can provide a binary path to the browser via:

```
{
    browserName: 'chrome',  // or 'chromium'
    'goog:chromeOptions': {
```

```
        binary: '/Applications/Google\ Chrome\ Canary.app/
Contents/MacOS/Google\ Chrome\ Canary'
    }
}
```

- Additionally, if you like to use a manually downloaded driver, you can provide a binary path to the driver via:

```
{
    browserName: 'chrome', // or 'chromium'
    'wdio:chromedriverOptions': {
        binary: '/path/to/chromdriver'
    }
}
```

2.Firefox

- When testing on Firefox, WebdriverIO will automatically download the desired browser version and driver for you based on the defined browserVersion, e.g.:

```
{
    browserName: 'firefox',
    browserVersion: '119.0a1' // or 'latest'
}
```

- If you like to test a manually downloaded version you can provide a binary path to the browser via:

```
{
    browserName: 'firefox',
    'moz:firefoxOptions': {
        binary: '/Applications/Firefox\ Nightly.app/
Contents/MacOS/firefox'
    }
}
```

- Additionally, if you like to use a manually downloaded driver, you can provide a binary path to the driver via:

```
{
    browserName: 'firefox',
    'wdio:geckodriverOptions': {
        binary: '/path/to/geckodriver'
    }
}
```

3.Safari

- When testing on Safari, make sure you have the Safari Technology Preview installed on your machine. You can point WebdriverIO to that version via:

```
{
    browserName: 'safari technology preview'
```

```
}
```

To run in maximised mode of browser

```
capabilities: [{
        maxInstances: 2,
        browserName: 'chrome',
        'goog:chromeOptions': {
                args: ['--start-maximized','headless',
'disable-gpu']
            },
        'wdio:chromedriverOptions': {
                binary : "/Users/krishnabros/Documents/
chromedriver/chromedriver129"
            }
    }],
```

To run specific spec file in specific browser

- We have **specs** field to mention specific spec/ test file to run in specific browser

```
{
        maxInstances: 2,
        browserName: 'firefox',
        specs: [
            'test/specs/firstTest.js',
            'test/specs/uiControls.js'
        ],
        'moz:firefoxOptions': {
            //args: ['-headless']
        },
```

Parallel execution of spec files/ test cases:

- Using maxInstances field we can control the number of browser to open and run spec files/ test cases at a time

```
maxInstances: 3
```

- As projects grow, inevitably more and more integration tests are added. This increases build time and slows productivity.
- To prevent this, you should run your tests in parallel. WebdriverIO already tests each spec (or *feature file* in Cucumber) in parallel within a single session. In general, try to test only a single feature per spec file. Try to not have too many or too few tests in one file. (However, there is no golden rule here.)
- Once your tests have several spec files, you should start running your tests concurrently. To do so, adjust the maxInstances property in your config file. WebdriverIO allows you to run your tests with maximum concurrency—meaning that no matter how many files and tests you have, they can all run in parallel. (This is still subject to certain limits, like your computer's CPU, concurrency restrictions, etc.)

- Let's say you have 3 different capabilities (Chrome, Firefox, and Safari) and you have set maxInstances to 1. The WDIO test runner will spawn 3 processes. Therefore, if you have 10 spec files and you set maxInstances to 10, *all* spec files will be tested simultaneously, and 30 processes will be spawned.
- You can define the maxInstances property globally to set the attribute for all browsers.
- If you run your own WebDriver grid, you may (for example) have more capacity for one browser than another. In that case, you can *limit* the maxInstances in your capability object:

```
// wdio.conf.js
export const config = {
    // ...
    // set maxInstance for all browser
    maxInstances: 10,
    // ...
    capabilities: [{
        browserName: 'firefox'
    }, {
        // maxInstances can get overwritten per capability. So if you have an in-house WebDriver
        // grid with only 5 firefox instance available you can make sure that not more than
        // 5 instance gets started at a time.
        browserName: 'chrome'
    }],
    // ...
}
```

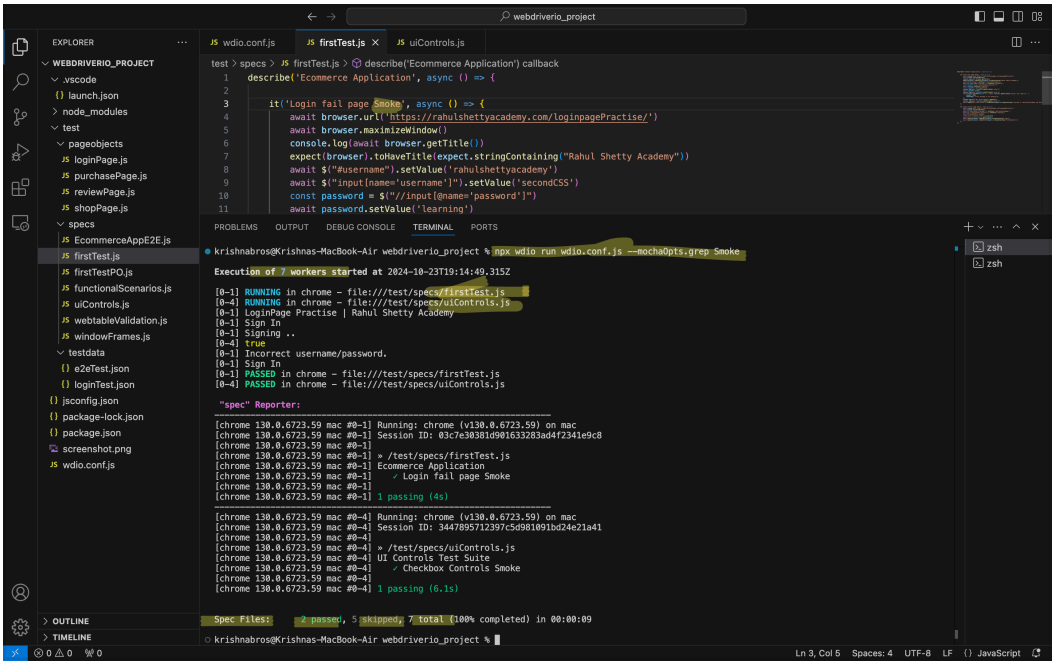
Grouping test cases for execution using Mocha Grep Options

- **Smoke** - we can add this 'Smoke' text at the end of 'it' block and this will run that group of test cases at execution time

`Mocha Grep Options` helps in selecting and controlling specific test cases of Spec files to run at run time

- Use below command to run 'Smoke' test cases (or 'it' blocks) of spec files at runtime with the help of Mocha Grep Options

```
npx wdio run wdio.conf.js --mochaOpts.grep Smoke
```



Bail -> Stop testing after failure

- If you want your test run to stop after a specific number of test failures, use bail. (It defaults to 0, which runs all tests no matter what.)
- Note: A test in this context are all tests within a single spec file (when using Mocha or Jasmine) or all steps within a feature file (when using Cucumber).
- If you want to control the bail behavior within tests of a single test file, take a look at the available framework options.
- With the bail option, you can tell WebdriverIO to stop testing after any test fails.
- This is helpful with large test suites when you already know that your build will break, but you want to avoid the lengthy wait of a full testing run.
- The bail option expects a number, which specifies how many test failures can occur before WebDriver stop the entire testing run. The default is 0, meaning that it always runs all tests specs it can find.

```
Type: Number
Default: 0 (don't bail; run all tests)
```

Example->

```
bail: 0   -> means execution donot stop until completed even if any there are any number of failed test cases
```

Or

```
bail: 3   -> execution abruptly stops if minimum 3 test fails -> thought behind it is to check application is phishy
```

BaseURL

Shorten **url** command calls by setting a base URL.

- If your **url** parameter starts with /, then baseUrl is prepended (except the baseUrl path, if it has one).
- If your **url** parameter starts without a scheme or / (like some/path), then the full baseUrl is prepended directly.
- baseUrl is attached at runtime to browser url
- baseUrl helps in avoiding the hard coding of url

```
baseUrl: 'https://rahulshettyacademy.com/'
```

Previous:

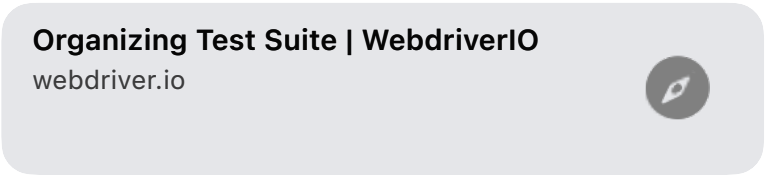
```
await browser.url('https://rahulshettyacademy.com/
loginpagePractise/')
```

Now:

```
baseUrl: 'https://rahulshettyacademy.com/'
await browser.url('/loginpagePractise/')
```

Grouping Test Specs In Suites

Ref: https://webdriver.io/docs/organizingsuites/



Suites: You can group test specs in suites and run single specific suites instead of all of them.

```
// wdio.conf.js
export const config = {
    // define all tests
    specs: ['./test/specs/**/*.spec.js'],
    // ...
    // define specific suites
    suites: {
        login: [
            './test/specs/login.success.spec.js',
            './test/specs/login.failure.spec.js'
        ],
        otherFeature: [
            // ...
        ]
    },
    // ...
}
```

Now, if you want to only run a single suite, you can pass the suite name as a CLI argument:

```
wdio wdio.conf.js --suite login
```

Or, run multiple suites at once:

```
wdio wdio.conf.js --suite login --suite otherFeature
```

Grouping Test Specs To Run Sequentially

To group tests to run in a single instance, define them as an array within the specs definition.

```
    "specs": [
        [
            "./test/specs/test_login.js",
            "./test/specs/test_product_order.js",
            "./test/specs/test_checkout.js"
        ],
        "./test/specs/test_b*.js",
    ],
```

In the example above, the tests 'test_login.js', 'test_product_order.js' and 'test_checkout.js' will be run sequentially in a single instance and each of the "test_b*" tests will run concurrently in individual instances.

Run Selected Tests

In some cases, you may wish to only execute a single test (or subset of tests) of your suites.

With the **--spec** parameter, you can specify which *suite* (Mocha, Jasmine) or *feature* (Cucumber) should be run. The path is resolved relative from your current working directory.

For example, to run only your login test:

```
wdio wdio.conf.js --spec ./test/specs/e2e/login.js
```

Or run multiple specs at once:

```
wdio wdio.conf.js --spec ./test/specs/signup.js --spec ./
test/specs/forgot-password.js
```

If the **--spec** value does not point to a particular spec file, it is instead used to filter the spec filenames defined in your configuration.

To run all specs with the word “dialog” in the spec file names, you could use:

```
wdio wdio.conf.js --spec dialog
```

Exclude Selected Tests

When needed, if you need to exclude particular spec file(s) from a run, you can use the **--exclude** parameter (Mocha, Jasmine) or feature (Cucumber).

For example, to exclude your login test from the test run:

```
wdio wdio.conf.js --exclude ./test/specs/e2e/login.js
```

Or, exclude multiple spec files:

```
wdio wdio.conf.js --exclude ./test/specs/signup.js --
exclude ./test/specs/forgot-password.js
```

Or, exclude a spec file when filtering using a suite:

```
wdio wdio.conf.js --suite login --exclude ./test/specs/e2e/
```

```
login.js
```

If the --exclude value does not point to a particular spec file, it is instead used to filter the spec filenames defined in your configuration.
To exclude all specs with the word “dialog” in the spec file names, you could use:
```
wdio wdio.conf.js --exclude dialog
```

When the **--exclude** option is provided, it will override any patterns defined by the config or capability level's exclude parameter.

Run Suites and Test Specs
Run an entire suite along with individual specs.
```
wdio wdio.conf.js --suite login --spec ./test/specs/
signup.js
```

Running Specific Tests with MochaOpts
You can also filter which specific **suite|describe** and/or **it|test** you want to run by passing a mocha specific argument: **--mochaOpts.grep** to the wdio CLI.
```
wdio wdio.conf.js --mochaOpts.grep myText
wdio wdio.conf.js --mochaOpts.grep "Text with spaces"
```

Note: Mocha will filter the tests after the WDIO test runner creates the instances, so you might see several instances being spawned but not actually executed.

Exclude Specific Tests with MochaOpts
You can also filter which specific **suite|describe** and/or **it|test** you want to exclude by passing a mocha specific argument: **--mochaOpts.invert** to the wdio CLI. **--mochaOpts.invert** performs opposite of **--mochaOpts.grep**
```
wdio wdio.conf.js --mochaOpts.grep "string|regex" --
mochaOpts.invert
wdio wdio.conf.js --spec ./test/specs/e2e/login.js --
mochaOpts.grep "string|regex" --mochaOpts.invert
```

Note: Mocha will filter the tests after the WDIO test runner creates the instances, so you might see several instances being spawned but not actually executed.

Using capability-defined spec patterns
```
{
    specs: ['tests/general/**/*.js']
}
```

Inherit From Main Config File

- If you run your test suite in multiple environments (e.g., dev and integration) it may help to use multiple configuration files to keep things manageable.
- Similar to the page object concept, the first thing you'll need is a main config file. It contains all configurations you share across environments.
- Then create another config file for each environment, and supplement the the main config with the environment-specific ones:

```
// wdio.dev.config.js
import { deepmerge } from 'deepmerge-ts'
import wdioConf from './wdio.conf.js'

// have main config file as default but overwrite
environment specific information
export const config = deepmerge(wdioConf.config, {
    capabilities: [
        // more caps defined here
        // ...
    ],

    // run tests on sauce instead locally
    user: process.env.SAUCE_USERNAME,
    key: process.env.SAUCE_ACCESS_KEY,
    services: ['sauce']
}, { clone: false })

// add an additional reporter
config.reporters.push('allure')
```

Run test with Mocha Opts in test runner:

- We can run specific group of test cases using mocha grep options either by command line parameters or directly from test runner

(wdio.conf.js) file

```
mochaOpts: {
        ui: 'bdd',
        timeout: 60000
    },
```

- Add grep: "sanity" to run sanity test cases or 'it' blocks

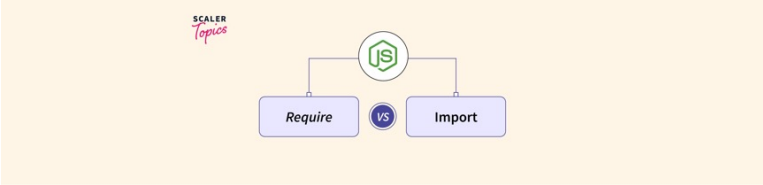
```
mochaOpts: {
        ui: 'bdd',
        timeout: 60000,
        grep: "sanity"
    },
```

History commands:
1072 safaridriver --enable
1167 npx wdio run wdio.conf.js
1169 npx wdio run wdio.conf.js --mochaOpts.grep Smoke
1171 npx wdio run wdio.conf.js --mochaOpts.grep Smoke
1178 npx wdio run wdio.conf.js --suite debitCard
1180 npx wdio run wdio.conf.js --suite creditCard
1182 npx wdio run wdio.conf.js --spec test/specs/EcommerceAppE2E.js

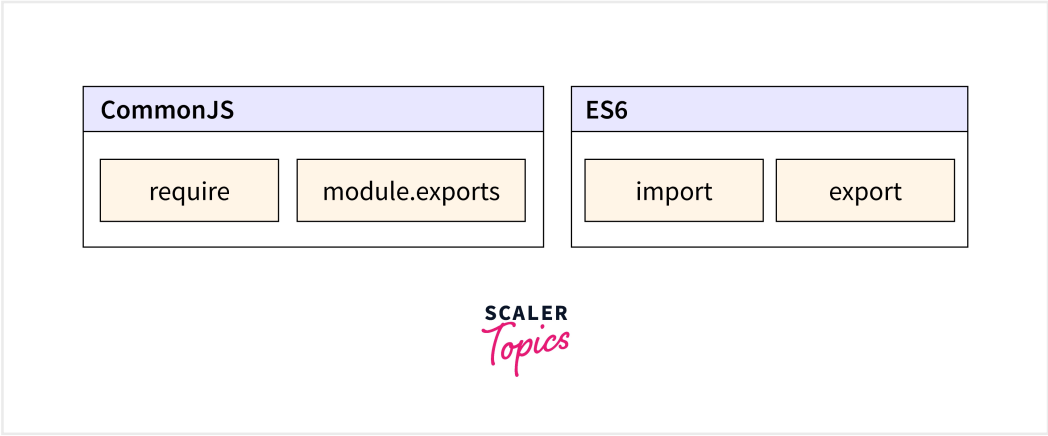
```
npx wdio run wdio.conf.js --exclude test/specs/
EcommerceAppE2E.js
npx wdio run wdio.conf.js --spec test/specs/
EcommerceAppE2E.js  --exclude test/specs/EcommerceAppE2E.js
```

Node.js Require vs Import

Ref: https://www.scaler.com/topics/nodejs/require-vs-import-nodejs/



Node.js Require vs Import - Scaler Topics
scaler.com



Rerun suites in Mocha

Ref: https://webdriver.io/docs/retry/

Retry Flaky Tests | WebdriverIO
webdriver.io

Since version 3 of Mocha, you can rerun whole test suites (everything inside an describe block). If you use Mocha you should favor this retry mechanism instead of the WebdriverIO implementation that only allows you to rerun certain test blocks (everything within an it block). In order to use the this.retries() method, the suite block describe must use an unbound function **function(){} instead of a fat arrow function () => {},** as described in Mocha docs. Using Mocha you can also set a retry count for all specs using mochaOpts.retries in your wdio.conf.js.
Here is an example:

```
describe('retries', function () {
    // Retry all tests in this suite up to 4 times
    this.retries(4)

    beforeEach(async () => {
        await browser.url('http://www.yahoo.com')
    })

    it('should succeed on the 3rd try', async function () {
        // Specify this test to only retry up to 2 times
        this.retries(2)
        console.log('run')
        await expect($('.foo')).toBeDisplayed()
```

```
    })
})
```

Rerun single tests in Jasmine or Mocha

To rerun a certain test block you can just apply the number of reruns as last parameter after the test block function:

- **MochaJasmine**

```
describe('my flaky app', () => {
    /**
     * spec that runs max 4 times (1 actual run + 3 reruns)
     */
    it('should rerun a test at least 3 times', async
function () {
        console.log(this.wdioRetries) // returns number of
retries
        // ...
    }, 3)
})
```

Real Example:

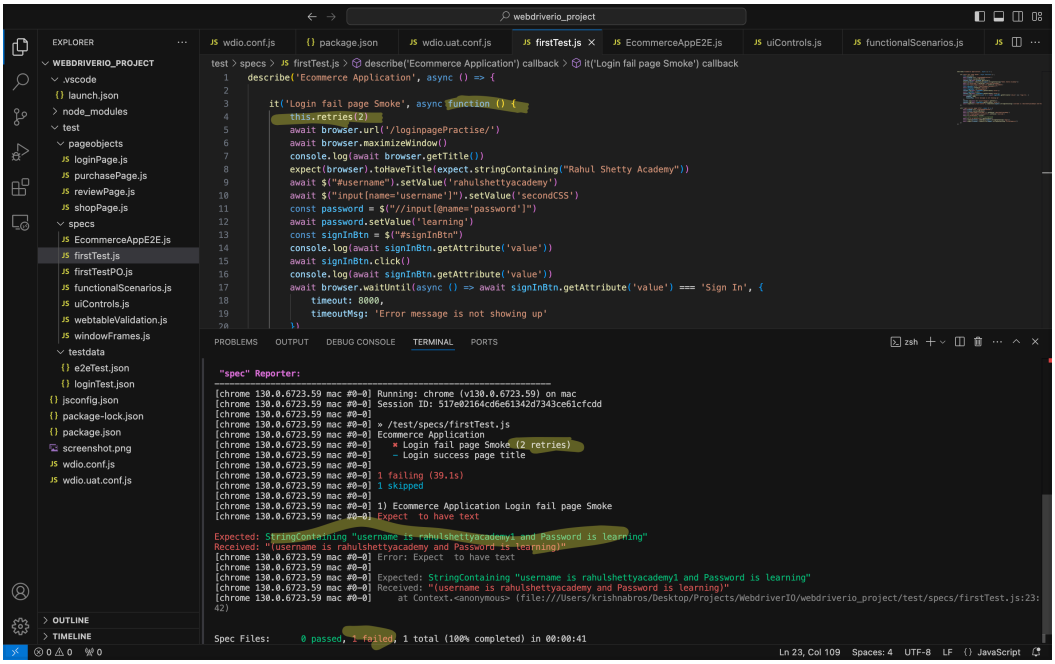
```
describe('Ecommerce Application', async () => {

    it('Login fail page Smoke', async function () {
        this.retries(2)
        await browser.url('/loginpagePractise/')
        await browser.maximizeWindow()
        console.log(await browser.getTitle())

expect(browser).toHaveTitle(expect.stringContaining("Rahul
Shetty Academy"))
        await $("#username").setValue('rahulshettyacademy')
        await $
("input[name='username']").setValue('secondCSS')
        const password = $("//input[@name='password']")
        await password.setValue('learning')
        const signInBtn = $("#signInBtn")
        console.log(await signInBtn.getAttribute('value'))
        await signInBtn.click()
        console.log(await signInBtn.getAttribute('value'))
        await browser.waitUntil(async () => await
signInBtn.getAttribute('value') === 'Sign In', {
            timeout: 8000,
            timeoutMsg: 'Error message is not showing up'
        })
        console.log(await $(".alert-danger").getText())
        console.log(await signInBtn.getAttribute('value'))
```

```
        await expect($("p.text-
center")).toHaveText(expect.stringContaining('username is
rahulshettyacademy1 and Password is learning'))
    })

})
```



Rerunning in Cucumber

Rerun full suites in Cucumber

For cucumber >=6 you can provide the retry configuration option along with a retryTagFilter optional parameter to have all or some of your failing scenarios get additional retries until succeeded. For this feature to work you need to set the scenarioLevelReporter to true.

Rerun Step Definitions in Cucumber

To define a rerun rate for a certain step definitions just apply a retry option to it, like:

```
export default function () {
    /**
     * step definition that runs max 3 times (1 actual run +
2 reruns)
     */
    this.Given(/^some step definition$/, { wrapperOptions:
{ retry: 2 } }, async () => {
        // ...
    })
    // ...
})
```

Reruns can only be defined in your step definitions file, never in your feature file.

Run a specific test multiple times

This is to help prevent flaky tests from being introduced in a codebase. By adding the --**repeat** cli option it will run the specified specs or suites N times. When using this cli flag the --spec or --suite flag must also be specified.

```
# This will run the example.e2e.js spec 5 times
npx wdio run ./wdio.conf.js --spec example.e2e.js --repeat 5
```

Scripts in package.json:

Add below scripts in package.json file

```
 "scripts": {
    "wdio": "wdio run ./wdio.conf.js",
    "creditCardTest": "npx wdio run wdio.conf.js --suite
creditCard",
    "QARegression": "npx wdio run wdio.conf.js",
    "UATRegression": "npx wdio run wdio.uat.conf.js",
    "debitCardTest": "npx wdio run wdio.conf.js --suite
debitCard"
  }
```

CLI:

```
npm run creditCardTest
npm run QARegression
npm run UATRegression
```

Allure Reporter

```
npm install @wdio/allure-reporter --save-dev
```

Configuration

Configure the output directory in your wdio.conf.js file:

```
export const config = {
    // ...
    reporters: [['allure', {
        outputDir: 'allure-results',
        disableWebdriverStepsReporting: true,
        disableWebdriverScreenshotsReporting: false,
    }]],
    // ...
}
```

- **outputDir** defaults to ./allure-results. After a test run is complete, you will find that this directory has been populated with an .**xml** file for each spec, plus a number of **.txt** and .**png** files and other attachments.

Add Screenshots

Screenshots can be attached to the report by using
the takeScreenshot function from WebDriverIO in the afterTest hook for Mocha

and Jasmine or afterStep hook for Cucumber. First
set disableWebdriverScreenshotsReporting: false in reporter options, then add in afterStep hook:

Mocha / Jasmine

wdio.conf.js

```
afterTest: async function(test, context, { error, result,
duration, passed, retries }) {
    if (error) {
        await browser.takeScreenshot();
    }
}
```

Step1: Download allure dependencies
Step2: Prepare configuration to create report directory
Step3: Add screenshot method to afterTest hook

Hooks

https://webdriver.io/docs/configuration/#hooks

Displaying the allure report

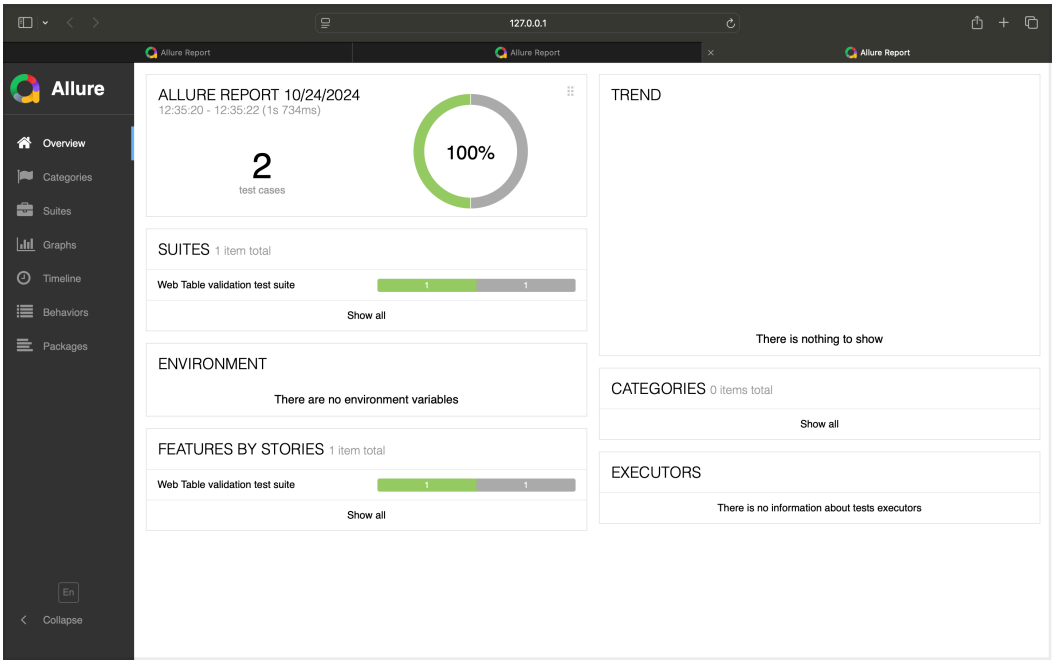
Install the Allure command-line tool, and process the results directory:

```
allure generate [allure_output_dir] && allure open
```

Example:

```
allure generate allure-results --clean && allure open
```

This will generate a report (by default in ./allure-report), and open it in your browser.



You can also auto-generate the report by using the Allure command line tool programmatically. To do so install the package in your project by:
`sudo npm install -g allure-commandline`

Allure clean history

```
allure generate --clean --output your-result-folder
Ex:
allure generate --clean --output allure-report

allure generate allure-results --clean && allure open

"scripts": {
    "wdio": "wdio run ./wdio.conf.js",
    "creditCardTest": "npx wdio run wdio.conf.js --suite
creditCard",
    "QARegression": "npx wdio run wdio.conf.js",
    "UATRegression": "npx wdio run wdio.uat.conf.js",
    "debitCardTest": "npx wdio run wdio.conf.js --suite
debitCard",
    "generateReport": "allure generate allure-results --
clean && allure open"
  }
```

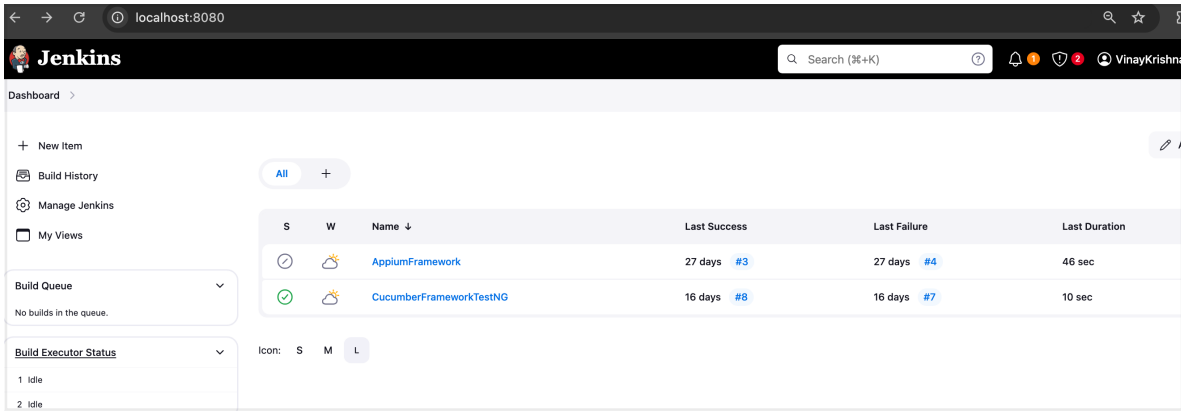
Jenkins job set up

To start jenkins server in local machine

- Download jenkins.war file
- Go to the location of jenkins.war
- Open the terminal

- Run the below command to start jenkins

```
java –jar jenkins.war –httpPort=9090
```

- Open the chrome browser
- Hit the url http://localhost:8080/
- Enter username & password (already setup earlier)to sign in to jenkins dashboard
  username: admin
  password: admin
- 

Configure Jenkins job

1. Add new item
2. Enter an item name as '**WebdriverIO**'
3. Select an item type as '**Freestyle project'**
4. Click on **Advanced CTA** under General tab
5. Select **'This project is parameterised'** checkbox under General tab
6. Click on **Add Parameter** dropdown under General tab
7. Select '**Choice Parameter'** option from dropdown
   (a) Enter **name** of choice parameter as '**Script**'
   (b) Enter **choices** as
   creditCardTest
   QARegression
   UATRegression
   debitCardTest
8. Select '**Use custom workspace**' checkbox under General tab
9. Enter **Directory** as '**/Users/krishnabros/Desktop/Projects/WebdriverIO/webdriverio_project**'
10. Select **Source Code Management** radio button as '**None**'
11. Click on **Add build steps** dropdown under Build Environment tab
12. (a) Select '**Execute shell**' option from dropdown if we are running test code in **Mac/ linux** machine
    (b) Select '**Execute Windows batch command**' option from dropdown if we are running test code in **Windows** machine
13. Enter command inside the **Build-steps**
    (a) Execute shell

npm run "$Script"

(b) Execute Windows batch command

npm run “%Script%”

14. Click on **Save** CTA

Run Jenkins Job

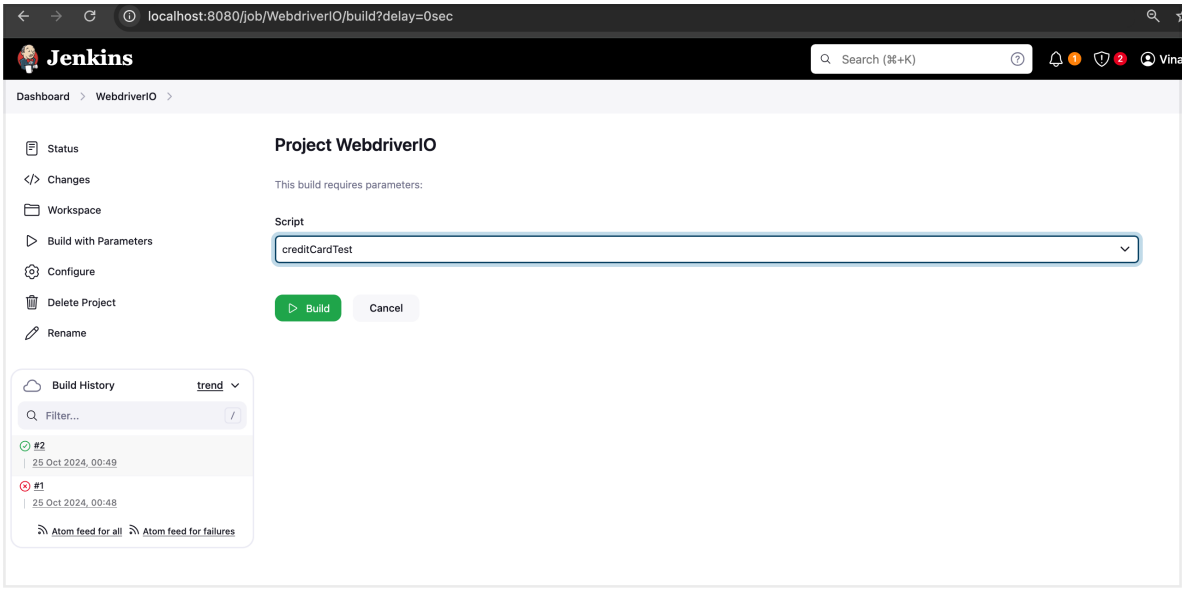
- Run the below command to start jenkins from terminal

```
java –jar /Users/krishnabros/Desktop/Projects/jenkins.war
–httpPort=9090
```

- Open the browser and navigate to below url to launch Jenkins dashboard

`http://localhost:8080/`

- Enter username & password to sign in to Jenkins dashboard
- Open the Job configured
- Select the script parameters
- Click on Build CTA to run the Job



Jenkins integration with Allure reporter

Ref: https://allurereport.org/docs/integrations-jenkins/

Allure Report Docs – Jenkins integration

allurereport.org

With the Allure Report plugin for Jenkins, you can add an “Allure Report” step to your build configuration, so that Jenkins will generate a test report automatically for each build.

Installation and configuration

To enable Allure support in your Jenkins installation, do the following under the

administrator account:

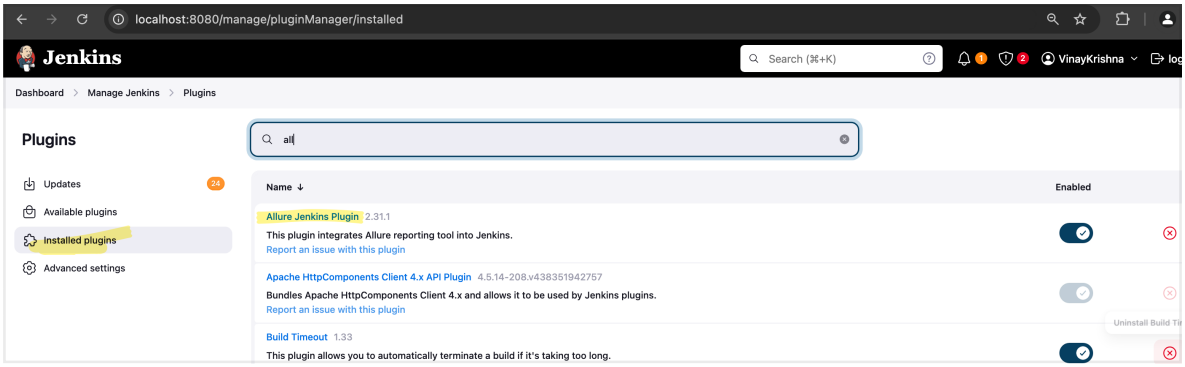
1. install the plugin,
2. install the global tool.

1. Install the plugin

The recommended method of installing the Allure Report plugin is via the Jenkins web interface. However, if your Jenkins configuration does not have access to the internet, you can install the plugin by uploading an HPI file to Jenkins manually. The selected installation method does not affect the functionality of the plugin.

To install the plugin from the internet directly:

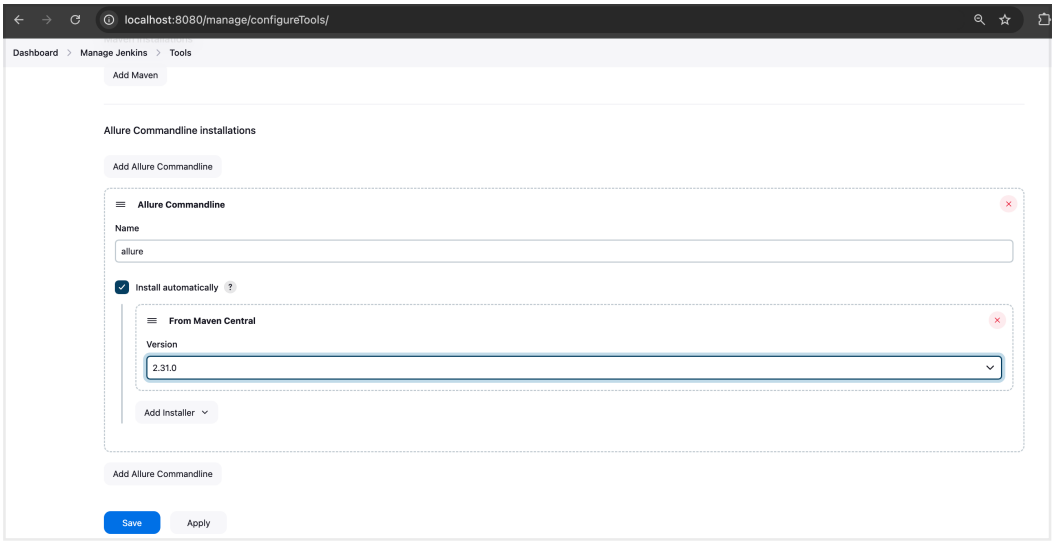
1. In the Jenkins web interface, go to **Manage Jenkins → Manage Plugins → Available plugins**.
2. Using the search box, find the "Allure" plugin. Check the checkbox next to the plugin.
3. Click **Install without restart**.
   On the **Download progress** page, wait until each status is "Success".



2.Install the global tool

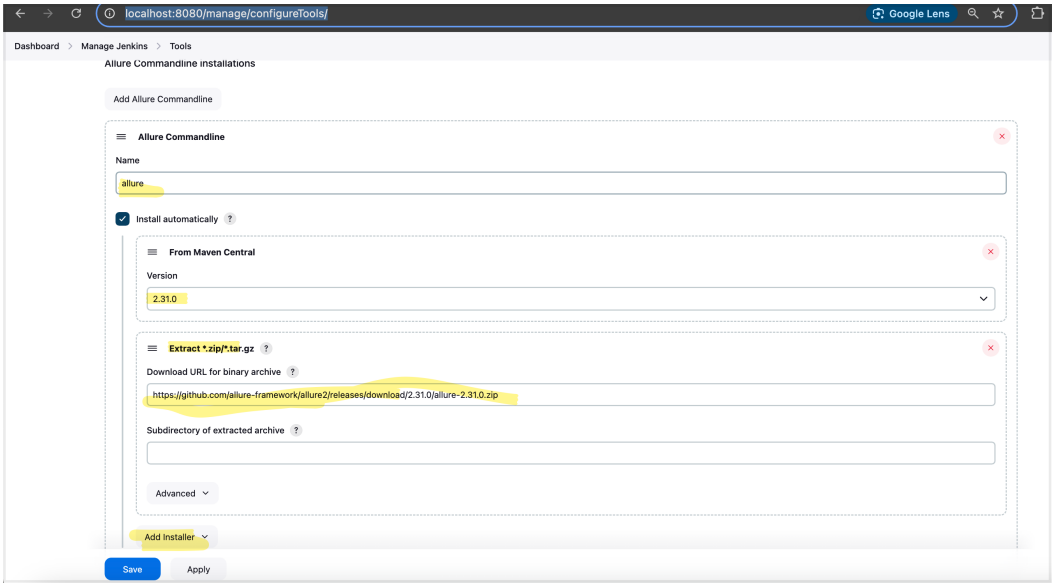
To install the tool from the internet directly:

1. In the Jenkins web interface, go to **Manage Jenkins → Global Tool Configuration**.
2. Under the **Allure Commandline** section, click **Add Allure Commandline**.
   Make sure that the **Install automatically** checkbox is checked, and the options block for **From Maven Central** is shown.
3. Fill in the fields:
   - **Name** — a name to help you recognize this version of the tool, e.g., "2.24.0".
   - **Version** — the release of Allure Report to install. The latest version is recommended.
4. Click **Save**.



To install the tool from a custom URL:

1. In the allure-commandline directory at Maven Central, find the latest ZIP or TAR.GZ archive. Copy it to a web server that is available to the Jenkins server.
2. In the Jenkins web interface, go to **Manage Jenkins → Global Tool Configuration**.
3. Under the **Allure Commandline** section, click **Add Allure Commandline**.
   Make sure that the **Install automatically** checkbox is checked.
4. Click **X** to remove the **From Maven Central** block.
5. Click **Add installer → Extract *.zip/*.tar.gz**..
6. Fill in the fields:
   - **Name** — a name to help you recognize this version of the tool, e.g., "2.24.0".
   - **Download URL for binary archive** — the URL from which the utility will be downloaded.
     (you will find the allure binary to download from below link https://github.com/allure-framework/allure2/releases/tag/2.31.0) as https://github.com/allure-framework/allure2/releases/download/2.31.0/allure-2.31.0.zip
   - **Subdirectory of extracted archive** — leave empty.
7. Click **Save**.



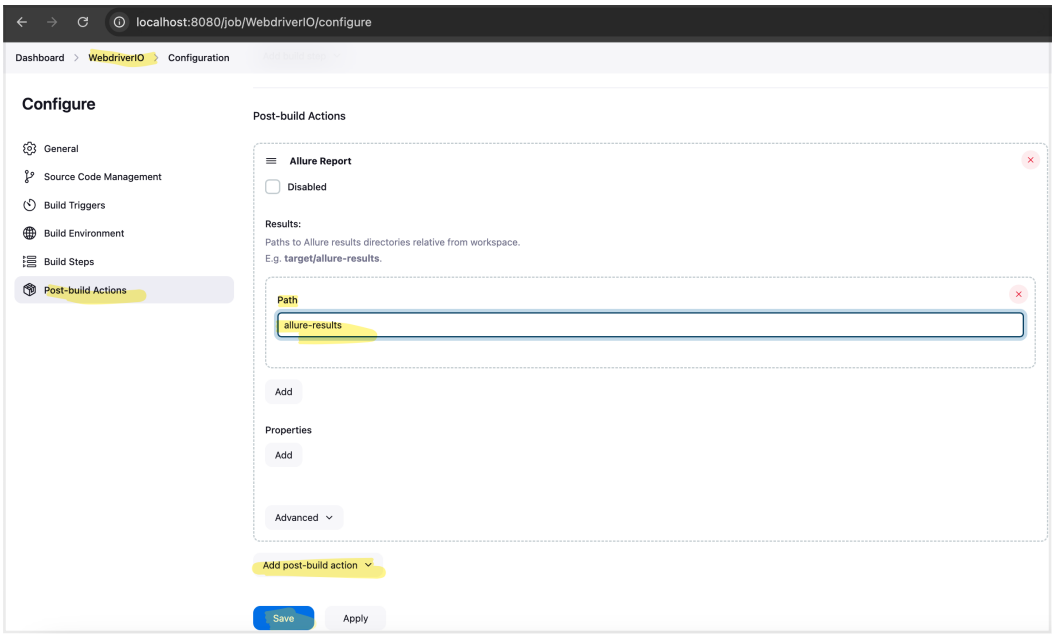
Using Allure in a freestyle project

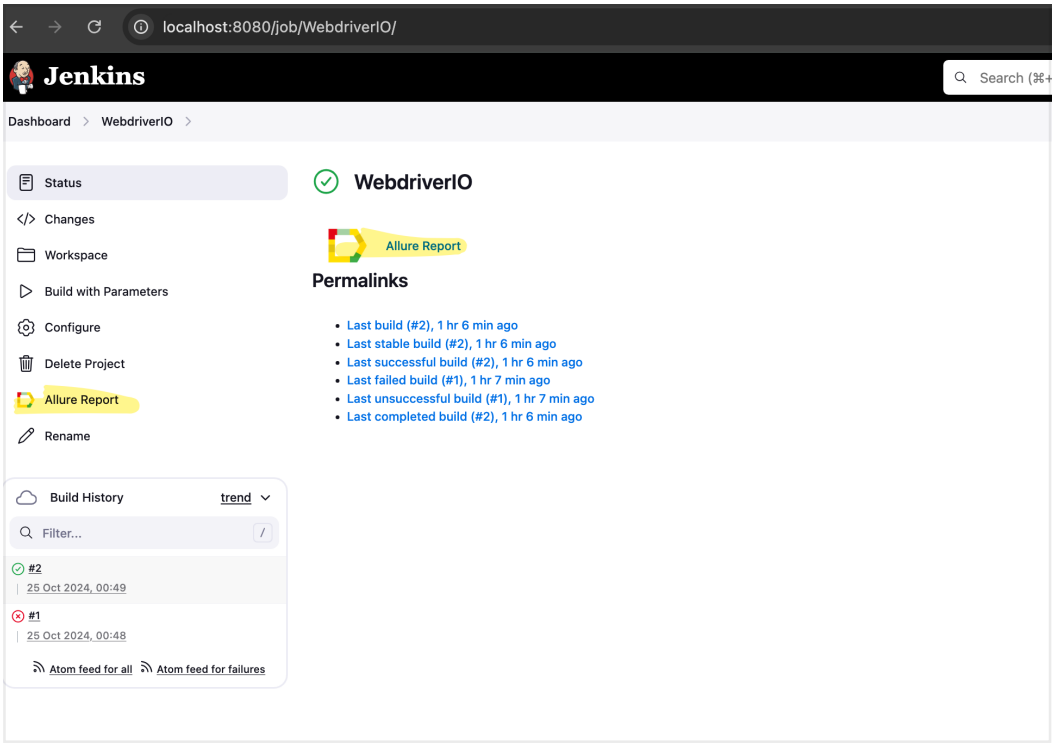
In Jenkins, a “freestyle project” is a build configuration for which you add and edit steps via the web interface. With the Allure Report plugin, the web interface gets an option to add a post-build action for building test reports.

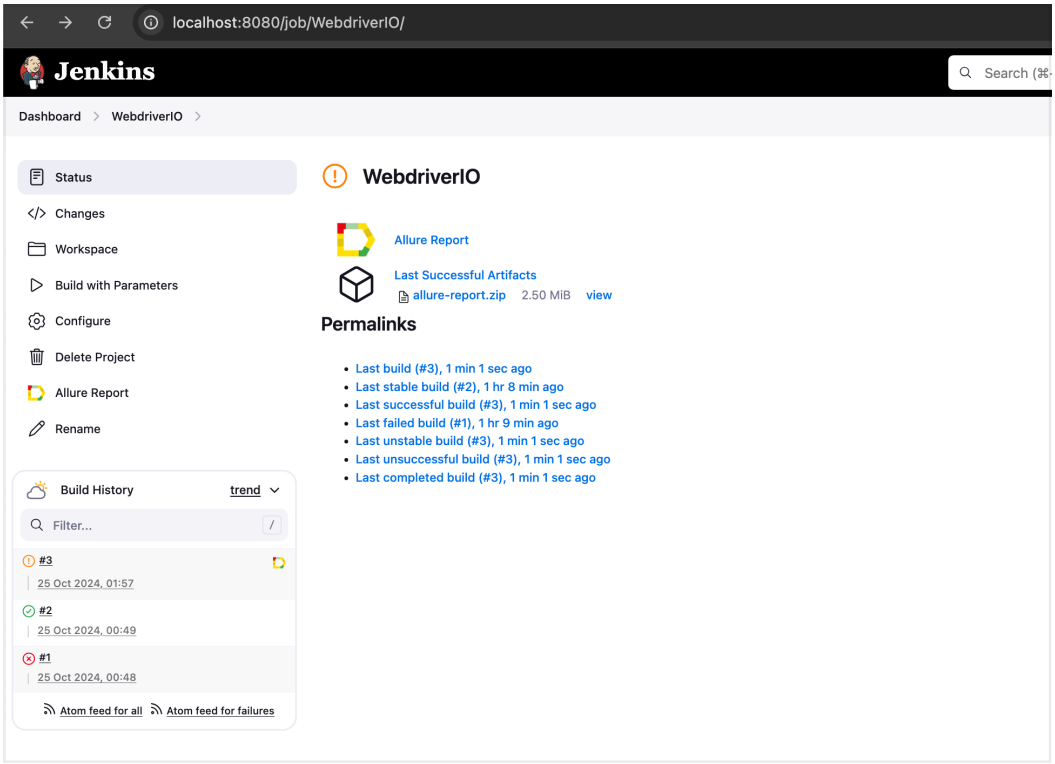
1. In the Jenkins web interface, select a job you want to enable Allure Report for.
2. In the menu on the left, click **Configure**.
   INFO
   Before continuing, make sure that the build configuration:
   - contains a step that runs the project's tests,
   - has the Allure adapter enabled for its test framework.
3. Under the **Post-build Actions** section, click **Add post-build action → Allure Report**.
4. In the **Results → Path** field, specify the path to the test results directory (see How it works).
   TIP
   If you have multiple build steps generating test results into multiple directories, use the **Add** button to specify more paths.

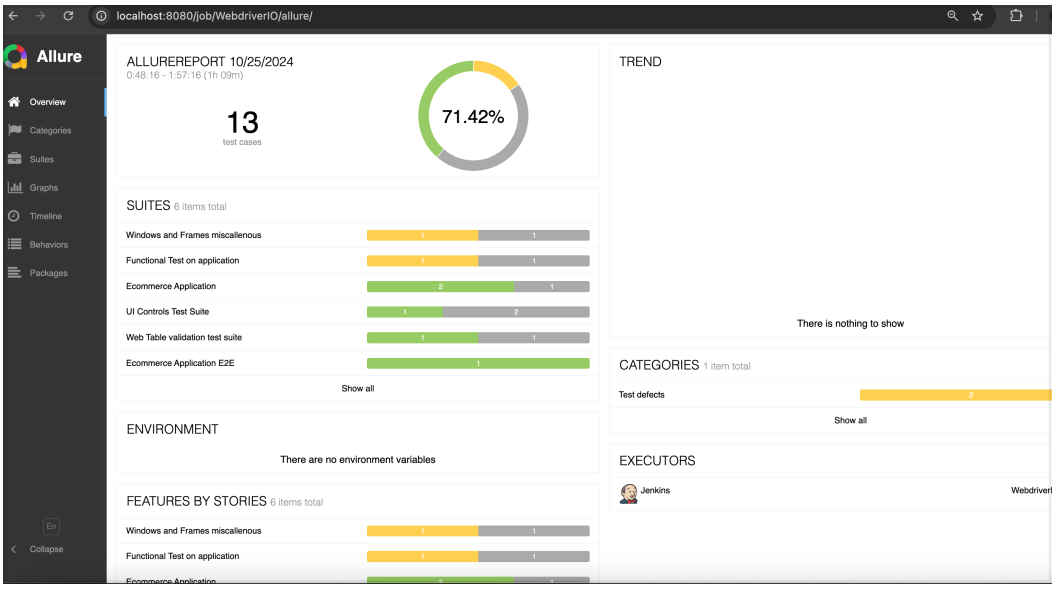
5. If you have more than one version of the Allure global tool installed (see Install the global tool), click **Advanced** and make sure that the proper version is selected in **Commandline**.
6. Click **Save**.

After the configuration, you can click **Build Now** to run the build configuration.


localhost:8080/job/WebdriverIO/configure
Dashboard
WebdriverIO
Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Post-build Actions
Allure Report
Disabled
Results:
Paths to Allure results directories relative from workspace.
E.g. target/allure-results.
Path
allure-results
Add
Properties
Add
Advanced
Add post-build action
Save
Apply


localhost:8080/job/WebdriverIO/
Jenkins
Search (⌘+
Dashboard
WebdriverIO
Status
Changes
Workspace
Build with Parameters
Configure
Delete Project
Allure Report
Rename
Build History
trend
Filter...
#2
25 Oct 2024, 00:49
#1
25 Oct 2024, 00:48
Atom feed for all
Atom feed for failures
WebdriverIO
Allure Report
Permalinks
Last build (#2), 1 hr 6 min ago
Last stable build (#2), 1 hr 6 min ago
Last successful build (#2), 1 hr 6 min ago
Last failed build (#1), 1 hr 7 min ago
Last unsuccessful build (#1), 1 hr 7 min ago
Last completed build (#2), 1 hr 6 min ago


localhost:8080/job/WebdriverIO/
Jenkins
Search (⌘
Dashboard
WebdriverIO
Status
Changes
Workspace
Build with Parameters
Configure
Delete Project
Allure Report
Rename
WebdriverIO
Allure Report
Last Successful Artifacts
allure-report.zip 2.50 MiB view
Permalinks
Last build (#3), 1 min 1 sec ago
Last stable build (#2), 1 hr 8 min ago
Last successful build (#3), 1 min 1 sec ago
Last failed build (#1), 1 hr 9 min ago
Last unstable build (#3), 1 min 1 sec ago
Last unsuccessful build (#3), 1 min 1 sec ago
Last completed build (#3), 1 min 1 sec ago
Build History
trend
Filter...
#3
25 Oct 2024, 01:57
#2
25 Oct 2024, 00:49
#1
25 Oct 2024, 00:48
Atom feed for all
Atom feed for failures


localhost:8080/job/WebdriverIO/allure/
Allure
Overview
Categories
Suites
Graphs
Timeline
Behaviors
Packages
Collapse
ALLUREREPORT 10/25/2024
0:48:16 - 1:57:16 (1h 09m)
13
test cases
71.42%
TREND
There is nothing to show
SUITES 6 items total
Windows and Frames miscallenous
Functional Test on application
Ecommerce Application
UI Controls Test Suite
Web Table validation test suite
Ecommerce Application E2E
Show all
CATEGORIES 1 item total
Test defects
Show all
ENVIRONMENT
There are no environment variables
EXECUTORS
Jenkins
FEATURES BY STORIES 6 items total
Windows and Frames miscallenous
Functional Test on application


Jenkins
Search (⌘+K)
VinayKrishna
log out
Dashboard
WebdriverIO
Status
Changes
Workspace
Build with Parameters
Configure
Delete Project
Allure Report
Rename
Build History
trend
Filter...
#4
25 Oct 2024, 01:59
#3
25 Oct 2024, 01:57
#2
25 Oct 2024, 00:49
#1
25 Oct 2024, 00:48
Atom feed for all
Atom feed for failures
WebdriverIO
Add description
Allure Report
Last Successful Artifacts
allure-report.zip 2.50 MiB view
Permalinks
Last build (#4), 41 sec ago
Last stable build (#2), 1 hr 10 min ago
Last successful build (#4), 41 sec ago
Last failed build (#1), 1 hr 11 min ago
Last unstable build (#4), 41 sec ago
Last unsuccessful build (#4), 41 sec ago
Last completed build (#4), 41 sec ago
Allure history trend
count
12
10
8
6
4
2
0
#1
#4