InstallationPlaywright Test was created specifically to accommodate the needs of end-to-end testing. Playwright supports all modern rendering engines including Chromium, WebKit, and Firefox. Test on Windows, Linux, and macOS, locally or on CI, headless or headed with native mobile emulation of Google Chrome for Android and Mobile Safari.You will learnHow to install PlaywrightWhat's InstalledHow to run the example testHow to open the HTML test reportInstalling Playwright Get started by installing Playwright using npm or yarn. Alternatively you can also get started and run your tests using the VS Code Extension.npmyarnpnpmnpm init playwright@latestyarn create playwrightpnpm dlx create-playwrightRun the install command and select the following to get started:Choose between TypeScript or JavaScript (default is TypeScript)Name of your Tests folder (default is tests or e2e if you already have a tests folder in your project)Add a GitHub Actions workflow to easily run tests on CIInstall Playwright browsers (default is true)What's Installed Playwright will download the browsers needed as well as create the following files.playwright.config.tspackage.jsonpackage-lock.jsontests/ example.spec.tstests-examples/ demo-todo-app.spec.tsThe playwright.config is where you can add configuration for Playwright including modifying which browsers you would like to run Playwright on. If you are running tests inside an already existing project then dependencies will be added directly to your package.json.The tests folder contains a basic example test to help you get started with testing. For a more detailed example check out the tests-examples folder which contains tests written to test a todo app.Running the Example Test By default tests will be run on all 3 browsers, chromium, firefox and webkit using 3 workers. This can be configured in the playwright.config file. Tests are run in headless mode meaning no browser will open up when running the tests. Results of the tests and test logs will be shown in the terminal.npx playwright testSee our doc on Running Tests to learn more about running tests in headed mode, running multiple tests, running specific tests etc.Run your tests with UI Mode for a better developer experience with time travel debugging, watch mode and more.npx playwright test --uiHTML Test Reports Once your test has finished running a HTML Reporter will have been created which shows you a full report of your tests allowing you to filter the report by browsers, passed tests, failed tests, skipped tests and flaky tests. You can click on each test and explore the test's errors as well as each step of the test. By default, the HTML report is opened automatically if some of the tests failed.npx playwright show-reportSystem requirements Node.js 16+Windows 10+, Windows Server 2016+ or Windows Subsystem for Linux (WSL).MacOS 12 Monterey or MacOS 13 Ventura.Debian 11, Ubuntu 20.04 or Ubuntu 22.04.What's next Write tests using web first assertions, page fixtures and locatorsRun single test, multiple tests, headed modeGenerate tests with CodegenSee a trace of your tests

Writing testsPlaywright tests are simple, theyperform actions, andassert the state against expectations.There is no need to wait for anything prior to performing an action: Playwright automatically waits for the wide range of actionability checks to pass prior to performing each action.There is also no need to deal with the race conditions when performing the checks - Playwright assertions are designed in a way that they describe the expectations that need to be eventually met.That's it! These design choices allow Playwright users to forget about flaky timeouts and racy checks in their tests altogether.You will learnHow to write the first testHow to perform actionsHow to use assertionsHow tests run in isolationHow to use test hooksFirst test Take a look at the following example to see how to write a test.tests/example.spec.tsimport { test, expect } from '@playwright/test';test('has title', async ({ page }) => { await page.goto('https://playwright.dev/'); // Expect a title "to contain" a substring. await expect(page).toHaveTitle(/Playwright/);});test('get started link', async ({ page }) => { await page.goto('https://playwright.dev/'); // Click the get started link. await page.getByRole('link', { name: 'Get started' }).click(); // Expects the URL to contain intro. await expect(page).toHaveURL(/.*intro/);});noteAdd // @ts-check at the start of each test file when using JavaScript in VS Code to get automatic type checking.Actions Navigation Most of the tests will start with navigating page to the URL. After that, test will be able to interact with the page elements.await page.goto('https://playwright.dev/');Playwright will wait for page to reach the load state prior to moving forward. Learn more about the page.goto() options.Interactions Performing actions starts with locating the elements. Playwright uses Locators API for that. Locators represent a way to find element(s) on the page at any moment, learn more about the different types of locators available. Playwright will wait for the element to be actionable prior to performing the action, so there is no need to wait for it to become available.// Create a locator.const getStarted = page.getByRole('link', { name: 'Get started' });// Click it.await getStarted.click();In most cases, it'll be written in one line:await page.getByRole('link', { name: 'Get started' }).click();Basic actions This is the list of the most popular Playwright actions. Note that there are many more, so make sure to check the Locator API section to learn more about them.ActionDescriptionlocator.check()Check the input checkboxlocator.click()Click the elementlocator.uncheck()Uncheck the input checkboxlocator.hover()Hover mouse over the elementlocator.fill()Fill the form field (fast)locator.focus()Focus the elementlocator.press()Press single keylocator.setInputFiles()Pick files to uploadlocator.selectOption()Select option in the drop downlocator.type()Type text character by character (slow)Assertions Playwright includes test assertions in the form of expect function. To make an assertion, call expect(value) and choose a matcher that reflects the expectation.There are many generic matchers like toEqual, toContain, toBeTruthy that can be used to assert any conditions.expect(success).toBeTruthy();Playwright also includes async matchers that will wait until the expected condition is met. Using these matchers allows making the tests non-flaky and resilient. For example, this code will wait until the page gets the title containing "Playwright":await expect(page).toHaveTitle(/Playwright/);Here is the list of the most popular async assertions. Note that there are many more to get familiar with:AssertionDescriptionexpect(locator).toBeChecked()Checkbox is checkedexpect(locator).toBeEnabled()Control is enabledexpect(locator).toBeVisible()Element is visibleexpect(locator).toContainText()Element contains textexpect(locator).toHaveAttribute()Element has attributeexpect(locator).toHaveCount()List of elements has given lengthexpect(locator).toHaveText()Element matches textexpect(locator).toHaveValue()Input element has valueexpect(page).toHaveTitle()Page has titleexpect(page).toHaveURL()Page has URLexpect(page).toHaveScreenshot()Page has screenshotTest Isolation Playwright Test is based on the concept of test fixtures such as the built in page fixture, which is

passed into your test. Pages are isolated between tests due to the Browser Context, which is equivalent to a brand new browser profile, where every test gets a fresh environment, even when multiple tests run in a single Browser.tests/example.spec.tstest('basic test', async ({ page }) => { // ...});Using Test Hooks You can use various test hooks such as test.describe to declare a group of tests and test.beforeEach and test.afterEach which are executed before/after each test. Other hooks include the test.beforeAll and test.afterAll which are executed once per worker before/after all tests.tests/example.spec.tsimport { test, expect } from '@playwright/test';test.describe('navigation', () => { test.beforeEach(async ({ page }) => {   // Go to the starting url before each test.   await page.goto('https://playwright.dev/'); }); test('main navigation', async ({ page }) => {   // Assertions use the expect API.   await expect(page).toHaveURL('https://playwright.dev/'); });});What's Next Run single test, multiple tests, headed modeGenerate tests with CodegenSee a trace of your tests
Running testsYou can run a single test, a set of tests or all tests. Tests can be run on one browser or multiple browsers. By default tests are run in a headless manner meaning no browser window will be opened while running the tests and results will be seen in the terminal.You will learnHow to run tests from the command lineHow to debug testsHow to open the HTML test reporterRun tests in UI Mode Run your tests with UI Mode for a better developer experience with time travel debugging, watch mode and more.npx playwright test --uiCommand Line Running all testsnpx playwright testRunning a single test filenpx playwright test landing-page.spec.tsRun a set of test filesnpx playwright test tests/todo-page/ tests/landing-page/Run files that have landing or login in the file namenpx playwright test landing loginRun the test with the titlenpx playwright test -g "add a todo item"Running tests in headed modenpx playwright test landing-page.spec.ts --headedRunning tests on a specific projectnpx playwright test landing-page.ts --project=chromiumDebugging Tests Since Playwright runs in Node.js, you can debug it with your debugger of choice e.g. using console.log or inside your IDE or directly in VS Code with the VS Code Extension. Playwright comes with the Playwright Inspector which allows you to step through Playwright API calls, see their debug logs and explore locators.Debugging all tests:npx playwright test --debugDebugging one test file:npx playwright test example.spec.ts --debugDebugging a test from the line number where the test(.. is defined:npx playwright test example.spec.ts:10 --debugCheck out our debugging guide to learn more about the Playwright Inspector as well as debugging with Browser Developer tools.Test Reports The HTML Reporter shows you a full report of your tests allowing you to filter the report by browsers, passed tests, failed tests, skipped tests and flaky tests. By default, the HTML report is opened automatically if some of the tests failed.npx playwright show-reportYou can click on each test and explore the tests errors as well as each step of the test.What's Next Generate tests with CodegenSee a trace of your tests
Test generatorPlaywright comes with the ability to generate tests out of the box and is a great way to quickly get started with testing. It will open two windows, a browser window where you interact with the website you wish to test and the Playwright Inspector window where you can record your tests, copy the tests, clear your tests as well as change the language of your tests.You will learnHow to record a testHow to generate locators Your browser does not support the video tag.Running Codegen Use the codegen command to run the test generator followed by the URL of the website you

want to generate tests for. The URL is optional and you can always run the command without it and then add the URL directly into the browser window instead.npx playwright codegen demo.playwright.dev/todomvcRecording a test Run codegen and perform actions in the browser. Playwright will generate the code for the user interactions. Codegen will look at the rendered page and figure out the recommended locator, prioritizing role, text and test id locators. If the generator identifies multiple elements matching the locator, it will improve the locator to make it resilient and uniquely identify the target element, therefore eliminating and reducing test(s) failing and flaking due to locators.Generating locators You can generate locators with the test generator. Press the 'Record' button to stop the recording and the 'Pick Locator' button will appear.Click on the 'Pick Locator' button and then hover over elements in the browser window to see the locator highlighted underneath each element. To choose a locator click on the element you would like to locate and the code for that locator will appear in the field next to the Pick Locator button.You can then edit the locator in this field to fine tune it or use the copy button to copy it and paste it into your code. Emulation You can also generate tests using emulation so as to generate a test for a specific viewport, device, color scheme, as well as emulate the geolocation, language or timezone. The test generator can also generate a test while preserving authenticated state. Check out the Test Generator guide to learn more.What's Next See a trace of your tests
Trace viewerPlaywright Trace Viewer is a GUI tool that lets you explore recorded Playwright traces of your tests meaning you can go back and forward through each action of your test and visually see what was happening during each action.You will learnHow to record a traceHow to open the HTML reportHow to open and view the traceRecording a Trace By default the playwright.config file will contain the configuration needed to create a trace.zip file for each test. Traces are setup to run on-first-retry meaning they will be run on the first retry of a failed test. Also retries are set to 2 when running on CI and 0 locally. This means the traces will be recorded on the first retry of a failed test but not on the first run and not on the second retry.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ retries: process.env.CI ? 2 : 0, // set to 2 when running on CI // ... use: {   trace: 'on-first-retry', // record traces on first retry of each test },});To learn more about available options to record a trace check out our detailed guide on Trace Viewer.Traces are normally run in a Continuous Integration(CI) environment as locally you can use debugging methods to debug tests. However should you want to run traces locally you can force tracing to be on with --trace on.npx playwright test --trace onnoteThe trace-on flag was introduced in Playwright v1.25. Check your package.json to make sure you have at least this version of Playwright installed.Opening the HTML Report The HTML report shows you a report of all your tests that have been ran and on which browsers as well as how long they took. Tests can be filtered by passed tests, failed, flakey or skipped tests. You can also search for a particular test. Clicking on a test will open the detailed view where you can see more information on your tests such as the errors, the test steps and the trace.npx playwright show-reportIn the HTML report click on the trace icon next to the test name file name to directly open the trace for the required test.You can also click open the detailed view of the test and scroll down to the 'Traces' tab and open the trace by clicking on the trace screenshot.To learn more about reporters check out our detailed guide on reporters including the HTML

Reporter.Viewing the Trace View traces of your test by clicking through each action or hovering using the timeline and see the state of the page before and after the action. Inspect the log, source and network during each step of the test. The trace viewer creates a DOM snapshot so you can fully interact with it, open devtools etc.To learn more about traces check out our detailed guide on Trace Viewer.
UI ModeUI Mode let's you explore, run and debug tests with a time travel experience complete with watch mode. All test files are loaded into the testing sidebar where you can expand each file and describe block to individually run, view, watch and debug each test. Filter tests by text or @tag or by passed, failed and skipped tests as well as by projects as set in your playwright.config file. See a full trace of your tests and hover back and forward over each action to see what was happening during each step and pop out the DOM snapshot to a separate window for a better debugging experience.Running tests in UI Mode To open UI mode, run the following command:npx playwright test --uiFiltering tests Filter tests by text or @tag or by passed, failed or skipped tests. You can also filter by projects as set in your playwright.config file. If you are using project dependencies make sure to run your setup tests first before running the tests that depend on them. The UI mode will not take into consideration the setup tests and therefore you will have to manually run them first.Running your tests Once you launch UI Mode you will see a list of all your test files. You can run all your tests by clicking the triangle icon in the sidebar. You can also run a single test file, a block of tests or a single test by hovering over the name and clicking on the triangle next to it. Viewing test traces Traces are shown for each test that has been run, so to see the trace, click on one of the test names. Note that you won't see any trace results if you click on the name of the test file or the name of a describe block.Actions and metadata In the Actions tab you can see what locator was used for every action and how long each one took to run. Hover over each action of your test and visually see the change in the DOM snapshot. Go back and forward in time and click an action to inspect and debug. Use the Before and After tabs to visually see what happened before and after the action. Next to the Actions tab you will find the Metadata tab which will show you more information on your test such as the Browser, viewport size, test duration and more.Source, console, log and network As you hover over each action of your test the source code for the test is highlighted below. Click on the source tab to see the source code for the entire test. Click on the console tab to see the console logs for each action. Click on the log tab to see the logs for each action. Click on the network tab to see the network logs for each action.Attachments The "Attachments" tab allows you to explore attachments. If you're doing visual regression testing, you'll be able to compare screenshots by examining the image diff, the actual image and the expected image. When you click on the expected image you can use the slider to slide one image over the other so you can easily see the differences in your screenshots.Pop out and inspect the DOM Pop out the DOM snapshot into it's own window for a better debugging experience by clicking on the pop out icon above the DOM snapshot. From there you can open the browser DevTools and inspect the HTML, CSS, Console etc. Go back to UI Mode and click on another action and pop that one out to easily compare the two side by side or debug each individually.Timeline view At the top of the trace you can see a timeline view of each action of your test. Hover back and forth to see an image snapshot for each action.Pick locator Click on the pick locator button and hover over the

DOM snapshot to see the locator for each element highlighted as you hover. Click on an element to save the locator into the pick locator field. You can then copy the locator and paste it into your test.Watch mode Next to the name of each test in the sidebar you will find an eye icon. Clicking on the icon will activate watch mode which will re-run the test when you make changes to it. You can watch a number of tests at the same time be clicking the eye icon next to each one or all tests by clicking the eye icon at the top of the sidebar. If you are using VS Code then you can easily open your test by clicking on the file icon next to the eye icon. This will open your test in VS Code right at the line of code that you clicked on.Docker & GitHub Codespaces For Docker and GitHub Codespaces environments, you can run UI mode in the browser. In order for an endpoint to be accessible outside of the container, it needs to be bound to the 0.0.0.0 interface:npx playwright test --ui-host=0.0.0.0In the case of GitHub Codespaces, the port gets forwarded automatically, so you can open UI mode in the browser by clicking on the link in the terminal.To have a static port, you can pass the --ui-port flag:npx playwright test --ui-port=8080 --ui-host=0.0.0.0noteBe aware that when specifying the --ui-host=0.0.0.0 flag, UI Mode with your traces, the passwords and secrets is accessible from other machines inside your network. In the case of GitHub Codespaces, the ports are only accessible from your account by default.
CI GitHub ActionsWhen installing Playwright you are given the option to add a GitHub Actions. This creates a playwright.yml file inside a .github/workflows folder containing everything you need so that your tests run on each push and pull request into the main/ master branch.What you will learn:GitHub ActionsCreate a Repo and Push to GitHubOpening the WorkflowsViewing Test LogsHTML ReportDownloading the HTML ReportViewing the HTML ReportViewing the TraceWhat's NextGitHub Actions Tests will run on push or pull request on branches main/master. The workflow will install all dependencies, install Playwright and then run the tests. It will also create the HTML report.name: Playwright Testson: push: branches: [main, master] pull_request: branches: [main, master]jobs: test: timeout-minutes: 60   runs-on: ubuntu-latest   steps: - uses: actions/checkout@v3   - uses: actions/setup-node@v3   with: node-version: 18   - name: Install dependencies   run: npm ci   - name: Install Playwright Browsers   run: npx playwright install --with-deps   - name: Run Playwright tests   run: npx playwright test   - uses: actions/upload-artifact@v3   if: always()   with: name: playwright-report   path: playwright-report/   retention-days: 30Create a Repo and Push to GitHub Create a repo on GitHub and create a new repository or push an existing repository. Follow the instructions on GitHub and don't forget to initialize a git repository using the git init command so you can add, commit and push your code.Opening the Workflows Click on the Actions tab to see the workflows. Here you will see if your tests have passed or failed.On Pull Requests you can also click on the Details link in the PR status check.Viewing Test Logs Clicking on the workflow run will show you the all the actions that GitHub performed and clicking on Run Playwright tests will show the error messages, what was expected and what was received as well as the call log.HTML Report The HTML Report shows you a full report of your tests. You can filter the report by browsers, passed tests, failed tests, skipped tests and flaky tests.Downloading the HTML Report In the Artifacts section click on the playwright-report to download your report in the format of a zip file.Viewing the HTML Report Locally opening the report will not work as expected as

you need a web server in order for everything to work correctly. First, extract the zip, preferably in a folder that already has Playwright installed. Using the command line change into the directory where the report is and use npx playwright show-report followed by the name of the extracted folder. This will serve up the report and enable you to view it in your browser.npx playwright show-report name-of-my-extracted-playwright-reportTo learn more about reports check out our detailed guide on HTML ReporterViewing the Trace Once you have served the report using npx playwright show-report, click on the trace icon next to the test's file name as seen in the image above. You can then view the trace of your tests and inspect each action to try to find out why the tests are failing.To learn more about traces check out our detailed guide on Trace Viewer.To learn more about running tests on CI check out our detailed guide on Continuous Integration.What's Next Learn how to use LocatorsLearn how to perform ActionsLearn how to write Assertions
Getting started - VS CodePlaywright Test was created specifically to accommodate the needs of end-to-end testing. Playwright supports all modern rendering engines including Chromium, WebKit, and Firefox. Test on Windows, Linux, and macOS, locally or on CI, headless or headed with native mobile emulation of Google Chrome for Android and Mobile Safari. Get started by installing Playwright and generating a test to see it in action. Alternatively you can also get started and run your tests using the CLI.Installation Install the VS Code extension from the marketplace or from the extensions tab in VS Code.Once installed, open the command panel and type:Install PlaywrightSelect Test: Install Playwright and Choose the browsers you would like to run your tests on. These can be later configured in the playwright.config file. You can also choose if you would like to have a GitHub Actions setup to run your tests on CI.Running Tests You can run a single test by clicking the green triangle next to your test block to run your test. Playwright will run through each line of the test and when it finishes you will see a green tick next to your test block as well as the time it took to run the test.Run Tests and Show Browsers You can also run your tests and show the browsers by selecting the option Show Browsers in the testing sidebar. Then when you click the green triangle to run your test the browser will open and you will visually see it run through your test. Leave this selected if you want browsers open for all your tests or uncheck it if you prefer your tests to run in headless mode with no browser open.Use the Close all browsers button to close all browsers.View and Run All Tests View all tests in the testing sidebar and extend the tests by clicking on each test. Tests that have not been run will not have the green check next to them. Run all tests by clicking on the white triangle as you hover over the tests in the testing sidebar.Run Tests on Specific Browsers The VS Code test runner runs your tests on the default browser of Chrome. To run on other/multiple browsers click the play button's dropdown and choose another profile or modify the default profile by clicking Select Default Profile and select the browsers you wish to run your tests on.Choose various or all profiles to run tests on multiple profiles. These profiles are read from the playwright.config file. To add more profiles such as a mobile profile, first add it to your config file and it will then be available here.Debugging Tests With the VS Code extension you can debug your tests right in VS Code see error messages, create breakpoints and live debug your tests.Error Messages If your test fails VS Code will show you error messages right in the editor showing what was expected, what was received as well as a complete call

log.Live Debugging You can debug your test live in VS Code. After running a test with the Show Browser option checked, click on any of the locators in VS Code and it will be highlighted in the Browser window. Playwright will highlight it if it exists and show you if there is more than one resultYou can also edit the locators in VS Code and Playwright will show you the changes live in the browser window.Run in Debug Mode To set a breakpoint click next to the line number where you want the breakpoint to be until a red dot appears. Run the tests in debug mode by right clicking on the line next to the test you want to run. A browser window will open and the test will run and pause at where the breakpoint is set. You can step through the tests, pause the test and rerun the tests from the menu in VS Code.Debug in different Browsers By default debugging is done using the Chromium profile. You can debug your tests on different browsers by right clicking on the debug icon in the testing sidebar and clicking on the 'Select Default Profile' option from the dropdown.Then choose the test profile you would like to use for debugging your tests. Each time you run your test in debug mode it will use the profile you selected. You can run tests in debug mode by right clicking the line number where your test is and selecting 'Debug Test' from the menu.To learn more about debugging, see Debugging in Visual Studio Code.Generating Tests CodeGen will auto generate your tests for you as you perform actions in the browser and is a great way to quickly get started. The viewport for the browser window is set to a specific width and height. See the configuration guide to change the viewport or emulate different environments.Record a New Test To record a test click on the Record new button from the Testing sidebar. This will create a test-1.spec.ts file as well as open up a browser window. In the browser go to the URL you wish to test and start clicking around. Playwright will record your actions and generate a test for you. Once you are done recording click the cancel button or close the browser window. You can then inspect your test-1.spec.ts file and see your generated test. Your browser does not support the video tag.Record at Cursor To record from a specific point in your test file click the Record at cursor button from the Testing sidebar. This generates actions into the existing test at the current cursor position. You can run the test, position the cursor at the end of the test and continue generating the test.Picking a Locator Pick a locator and copy it into your test file by clicking the Pick locator button form the testing sidebar. Then in the browser click the element you require and it will now show up in the Pick locator box in VS Code. Press 'enter' on your keyboard to copy the locator into the clipboard and then paste anywhere in your code. Or press 'escape' if you want to cancel.Playwright will look at your page and figure out the best locator, prioritizing role, text and test id locators. If the generator finds multiple elements matching the locator, it will improve the locator to make it resilient and uniquely identify the target element, so you don't have to worry about failing tests due to locators.What's next Write tests using web first assertions, page fixtures and locatorsSee test reportsSee a trace of your tests Release notesVersion 1.36 Ø<ßÝþ Summer maintenance release.Browser Versions Chromium 115.0.5790.75Mozilla Firefox 115.0WebKit 17.0This version was also tested against the following stable channels:Google Chrome 114Microsoft Edge 114Version 1.35 Highlights UI mode is now available in VSCode Playwright extension via a new "Show trace viewer" button:UI mode and trace viewer mark network requests handled with page.route() and browserContext.route() handlers, as well as those issued via the API testing:New option maskColor for methods page.screenshot(), locator.screenshot(), expect(page).toHaveScreenshot() and expect(locator).toHaveScreenshot() to change default masking color:await page.goto('https://playwright.dev');await expect(page).toHaveScreenshot({ mask: [page.locator('img')], maskColor: '#00FF00', // green});New uninstall CLI command to uninstall browser binaries:$ npx playwright uninstall # remove browsers installed by this installation$ npx playwright uninstall --all # remove all ever-install Playwright browsersBoth UI mode and trace viewer now could be opened in a browser tab:$ npx playwright test --ui-port 0 # open UI mode in a tab on a random port$ npx playwright show-trace --port 0 # open trace viewer in tab on a random port& þ Breaking changes playwright-core binary got renamed from playwright to playwright-core. So if you use playwright-core CLI, make sure to update the name:$ npx playwright-core install # the new way to install browsers when using playwright-coreThis change does not affect @playwright/test and playwright package users.Browser Versions Chromium 115.0.5790.13Mozilla Firefox 113.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 114Microsoft Edge 114Version 1.34 Highlights UI Mode now shows steps, fixtures and attachments: New property testProject.teardown to specify a project that needs to run after this and all dependent projects have finished. Teardown is useful to cleanup any resources acquired by this project.A common pattern would be a setup dependency with a corresponding teardown:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ projects: [{   name: 'setup',   testMatch: /global.setup\.ts/,   teardown: 'teardown',   },   {   name: 'teardown',   testMatch: /global.teardown\.ts/,   },   {   name: 'chromium',   use: devices['Desktop Chrome'],   dependencies: ['setup'],   },   {   name: 'firefox',   use: devices['Desktop Firefox'],   dependencies: ['setup'],   },   {   name: 'webkit',   use: devices['Desktop Safari'],   dependencies: ['setup'],   },],});New method expect.configure to create pre-configured expect instance with its own defaults such as timeout and soft.const slowExpect = expect.configure({ timeout: 10000 });await slowExpect(locator).toHaveText('Submit');// Always do soft assertions.const softExpect = expect.configure({ soft: true });New options stderr and stdout in testConfig.webServer to configure output handling:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ // Run your local dev server before starting the tests webServer: {   command: 'npm run start',   url: 'http://127.0.0.1:3000',   reuseExistingServer: !process.env.CI,   stdout: 'pipe',   stderr: 'pipe', },});New locator.and() to create a locator that matches both locators.const button = page.getByRole('button').and(page.getByTitle('Subscribe'));New events browserContext.on('console') and browserContext.on('dialog') to subscribe to any dialogs and console messages from any page from the given browser context. Use the new methods consoleMessage.page() and dialog.page() to pin-point event source.& þ Breaking changes npx playwright test no longer works if you install both playwright and @playwright/test. There's no need to install both, since you can always import browser automation APIs from @playwright/test directly:automation.tsimport { chromium, firefox, webkit } from '@playwright/test';/* ... */Node.js 14 is no longer supported since it reached its end-of-life on April 30, 2023.Browser Versions Chromium 114.0.5735.26Mozilla Firefox 113.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 113Microsoft Edge 113Version 1.33 Locators Update Use locator.or() to create a locator that matches either of the two locators.

Consider a scenario where you'd like to click on a "New email" button, but sometimes a security settings dialog shows up instead. In this case, you can wait for either a "New email" button, or a dialog and act accordingly:const newEmail = page.getByRole('button', { name: 'New email' });const dialog = page.getByText('Confirm security settings');await expect(newEmail.or(dialog)).toBeVisible();if (await dialog.isVisible()) await page.getByRole('button', { name: 'Dismiss' }).click();await newEmail.click();Use new options hasNot and hasNotText in locator.filter() to find elements that do not match certain conditions.const rowLocator = page.locator('tr');await rowLocator   .filter({ hasNotText: 'text in column 1' })   .filter({ hasNot: page.getByRole('button', { name: 'column 2 button' }) })   .screenshot();Use new web-first assertion expect(locator).toBeAttached() to ensure that the element is present in the page's DOM. Do not confuse with the expect(locator).toBeVisible() that ensures that element is both attached & visible.New APIs locator.or()New option hasNot in locator.filter()New option hasNotText in locator.filter()expect(locator).toBeAttached()New option timeout in route.fetch()reporter.onExit()& þ Breaking change The mcr.microsoft.com/playwright:v1.33.0 now serves a Playwright image based on Ubuntu Jammy. To use the focal-based image, please use mcr.microsoft.com/playwright:v1.33.0-focal instead.Browser Versions Chromium 113.0.5672.53Mozilla Firefox 112.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 112Microsoft Edge 112Version 1.32 Introducing UI Mode (preview) New UI Mode lets you explore, run and debug tests. Comes with a built-in watch mode.Engage with a new flag --ui:npx playwright test --uiNew APIs New options updateMode and updateContent in page.routeFromHAR() and browserContext.routeFromHAR().Chaining existing locator objects, see locator docs for details.New property testInfo.testId.New option name in method tracing.startChunk().& þ Breaking change in component tests Note: component tests only, does not affect end-to-end tests.@playwright/experimental-ct-react now supports React 18 only.If you're running component tests with React 16 or 17, please replace @playwright/experimental-ct-react with @playwright/experimental-ct-react17.Browser Versions Chromium 112.0.5615.29Mozilla Firefox 111.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 111Microsoft Edge 111Version 1.31 New APIs New property testProject.dependencies to configure dependencies between projects.Using dependencies allows global setup to produce traces and other artifacts, see the setup steps in the test report and more.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ projects: [{   name: 'setup',   testMatch: /global.setup\.ts/,   },   {   name: 'chromium',   use: devices['Desktop Chrome'],   dependencies: ['setup'],   },   {   name: 'firefox',   use: devices['Desktop Firefox'],   dependencies: ['setup'],   },   {   name: 'webkit',   use: devices['Desktop Safari'],   dependencies: ['setup'],   },],});New assertion expect(locator).toBeInViewport() ensures that locator points to an element that intersects viewport, according to the intersection observer API.const button = page.getByRole('button');// Make sure at least some part of element intersects viewport.await expect(button).toBeInViewport();// Make sure element is fully outside of viewport.await expect(button).not.toBeInViewport();// Make sure that at least half of the element intersects viewport.await expect(button).toBeInViewport({ ratio: 0.5 });Miscellaneous DOM snapshots in trace viewer can be now opened in a separate window.New method defineConfig to be used in playwright.config.New option Route.fetch.maxRedirects for method route.fetch().Playwright now supports Debian 11 arm64.Official docker images now include Node 18 instead of Node 16.& þ Breaking change in component tests Note: component tests only, does not affect end-to-end tests.playwright-ct.config configuration file for component testing now requires calling defineConfig.// Beforeimport { type PlaywrightTestConfig, devices } from '@playwright/experimental-ct-react';const config: PlaywrightTestConfig = { // ... config goes here ...};export default config;Replace config variable definition with defineConfig call:// Afterimport { defineConfig, devices } from '@playwright/experimental-ct-react';export default defineConfig({ // ... config goes here ...});Browser Versions Chromium 111.0.5563.19Mozilla Firefox 109.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 110Microsoft Edge 110Version 1.30 Browser Versions Chromium 110.0.5481.38Mozilla Firefox 108.0.2WebKit 16.4This version was also tested against the following stable channels:Google Chrome 109Microsoft Edge 109Version 1.29 New APIs New method route.fetch() and new option json for route.fulfill():await page.route('**/api/settings', async route => { // Fetch original settings. const response = await route.fetch(); // Force settings theme to a predefined value. const json = await response.json(); json.theme = 'Solorized'; // Fulfill with modified data. await route.fulfill({ json });});New method locator.all() to iterate over all matching elements:// Check all checkboxes!const checkboxes = page.getByRole('checkbox');for (const checkbox of await checkboxes.all()) await checkbox.check();locator.selectOption() matches now by value or label:<select multiple> <option value="red">Red</div> <option value="green">Green</div> <option value="blue">Blue</div></select>await element.selectOption('Red');Retry blocks of code until all assertions pass:await expect(async () => { const response = await page.request.get('https://api.example.com'); await expect(response).toBeOK();}).toPass();Read more in our documentation.Automatically capture full page screenshot on test failure:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   screenshot: {   mode: 'only-on-failure',   fullPage: true,   } }});Miscellaneous Playwright Test now respects jsconfig.json.New options args and proxy for androidDevice.launchBrowser().Option postData in method route.continue() now supports Serializable values.Browser Versions Chromium 109.0.5414.46Mozilla Firefox 107.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 108Microsoft Edge 108Version 1.28 Playwright Tools Record at Cursor in VSCode. You can run the test, position the cursor at the end of the test and continue generating the test.Live Locators in VSCode. You can hover and edit locators in VSCode to get them highlighted in the opened browser.Live Locators in CodeGen. Generate a locator for any element on the page using "Explore" tool.Codegen and Trace Viewer Dark Theme. Automatically picked up from operating system settings.Test Runner Configure retries and test timeout for a file or a test with test.describe.configure().// Each test in the file will be retried twice and have a timeout of 20 seconds.test.describe.configure({ retries: 2, timeout: 20_000 });test('runs first', async ({ page }) => {});test('runs second', async ({ page }) => {});Use testProject.snapshotPathTemplate and testConfig.snapshotPathTemplate to configure a template controlling location of snapshots generated by expect(page).toHaveScreenshot() and expect(snapshot).toMatchSnapshot().playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ testDir: './tests', snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',});New APIs locator.blur()locator.clear()android.launchServer() and android.connect()androidDevice.on('close')Browser Versions Chromium 108.0.5359.29Mozilla Firefox 106.0WebKit 16.4This version was also tested against the following stable channels:Google Chrome 107Microsoft Edge 107Version 1.27 Locators With these new APIs writing locators is a joy:page.getByText() to locate by text content.page.getByRole() to locate by ARIA role, ARIA attributes and accessible name.page.getByLabel() to locate a form control by associated label's text.page.getByTestId() to locate an element based on its data-testid attribute (other attribute can be configured).page.getByPlaceholder() to locate an input by placeholder.page.getByAltText() to locate an element, usually image, by its text alternative.page.getByTitle() to locate an element by its title.await page.getByLabel('User Name').fill('John');await page.getByLabel('Password').fill('secret-password');await page.getByRole('button', { name: 'Sign in' }).click();await expect(page.getByText('Welcome, John!')).toBeVisible();All the same methods are also available on Locator, FrameLocator and Frame classes.Other highlights workers option in the playwright.config.ts now accepts a percentage string to use some of the available CPUs. You can also pass it in the command line:npx playwright test --workers=20%New options host and port for the html reporter.import { defineConfig } from '@playwright/test';export default defineConfig({ reporter: [['html', { host: 'localhost', port: '9223' }]],});New field FullConfig.configFile is available to test reporters, specifying the path to the config file if any.As announced in v1.25, Ubuntu 18 will not be supported as of Dec 2022. In addition to that, there will be no WebKit updates on Ubuntu 18 starting from the next Playwright release.Behavior Changes expect(locator).toHaveAttribute() with an empty value does not match missing attribute anymore. For example, the following snippet will succeed when button does not have a disabled attribute.await expect(page.getByRole('button')).toHaveAttribute('disabled', '');Command line options --grep and --grep-invert previously incorrectly ignored grep and grepInvert options specified in the config. Now all of them are applied together.Browser Versions Chromium 107.0.5304.18Mozilla Firefox 105.0.1WebKit 16.0This version was also tested against the following stable channels:Google Chrome 106Microsoft Edge 106Version 1.26 Assertions New option enabled for expect(locator).toBeEnabled().expect(locator).toHaveText() now pierces open shadow roots.New option editable for expect(locator).toBeEditable().New option visible for expect(locator).toBeVisible().Other highlights New option maxRedirects for apiRequestContext.get() and others to limit redirect count.New command-line flag --pass-with-no-tests that allows the test suite to pass when no files are found.New command-line flag --ignore-snapshots to skip snapshot expectations, such as expect(value).toMatchSnapshot() and expect(page).toHaveScreenshot().Behavior Change A bunch of Playwright APIs already support the waitUntil: 'domcontentloaded' option. For example:await page.goto('https://playwright.dev', { waitUntil: 'domcontentloaded',});Prior to 1.26, this would wait for all iframes to fire the DOMContentLoaded event.To align with web specification, the 'domcontentloaded' value only waits for the target frame to fire the 'DOMContentLoaded' event. Use waitUntil: 'load' to wait for all iframes.Browser Versions Chromium 106.0.5249.30Mozilla Firefox 104.0WebKit 16.0This version was also tested against the following stable channels:Google Chrome 105Microsoft Edge 105Version 1.25 VSCode Extension Watch your tests running live & keep devtools open.Pick selector.Record new test from current page state.Test Runner test.step() now returns the value of the step function:test('should work', async ({ page }) => { const pageTitle = await test.step('get title', async () => {   await page.goto('https://playwright.dev');   return await page.title(); }); console.log(pageTitle);});Added test.describe.fixme().New 'interrupted' test status.Enable tracing via CLI flag: npx playwright test --trace=on.Announcements Ø<ß• We now ship Ubuntu 22.04 Jammy Jellyfish docker image: mcr.microsoft.com/ playwright:v1.34.0-jammy.Ø>Þ¦ This is the last release with macOS 10.15 support (deprecated as of 1.21).Ø>Þ¦ This is the last release with Node.js 12 support, we recommend upgrading to Node.js LTS (16).& þ Ubuntu 18 is now deprecated and will not be supported as of Dec 2022.Browser Versions Chromium 105.0.5195.19Mozilla Firefox 103.0WebKit 16.0This version was also tested against the following stable channels:Google Chrome 104Microsoft Edge 104Version 1.24 Ø<ß Multiple Web Servers in playwright.config.ts Launch multiple web servers, databases, or other processes by passing an array of configurations:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ webServer: [{   command: 'npm run start',   url: 'http://127.0.0.1:3000',   timeout: 120 * 1000,   reuseExistingServer: !process.env.CI,   },   {   command: 'npm run backend',   url: 'http://127.0.0.1:3333',   timeout: 120 * 1000,   reuseExistingServer: !process.env.CI,   }], use: {   baseURL: 'http://localhost:3000/', },});Ø=Ü Debian 11 Bullseye Support Playwright now supports Debian 11 Bullseye on x86_64 for Chromium, Firefox and WebKit. Let us know if you encounter any issues!Linux support looks like this:Ubuntu 20.04Ubuntu 22.04Debian 11Chromium' ' ' WebKit' ' ' Firefox' ' ' Ø=Ýuþ Anonymous Describe It is now possible to call test.describe() to create suites without a title. This is useful for giving a group of tests a common option with test.use().test.describe(() => { test.use({ colorScheme: 'dark' }); test('one', async ({ page }) => {   // ... }); test('two', async ({ page }) => {   // ... });});Ø>Ýé Component Tests Update Playwright 1.24 Component Tests introduce beforeMount and afterMount hooks. Use these to configure your app for tests.For example, this could be used to setup App router in Vue.js:src/component.spec.tsimport { test } from '@playwright/experimental-ct-vue';import { Component } from './mycomponent';test('should work', async ({ mount }) => { const component = await mount(Component, {   hooksConfig: {   /* anything to configure your app */   } });});playwright/index.tsimport { router } from '../router';import { beforeMount } from '@playwright/experimental-ct-vue/hooks';beforeMount(async ({ app, hooksConfig }) => { app.use(router);});A similar configuration in Next.js would look like this:src/component.spec.jsximport { test } from '@playwright/experimental-ct-react';import { Component } from './mycomponent';test('should work', async ({ mount }) => { const component = await mount(<Component></Component>, {   // Pass mock value from test into `beforeMount`.   hooksConfig: {   router: {   query: { page: 1, per_page: 10 },   asPath: '/posts'   }   } });});playwright/index.jsimport router from 'next/router';import { beforeMount } from '@playwright/experimental-ct-react/hooks';beforeMount(async ({ hooksConfig }) => { // Before mount, redefine useRouter to return mock value from test. router.useRouter = () => hooksConfig.router;});Version

1.23 Network Replay Now you can record network traffic into a HAR file and re-use this traffic in your tests.To record network into HAR file:npx playwright open --save-har=github.har.zip https://github.com/microsoftAlternatively, you can record HAR programmatically:const context = await browser.newContext({ recordHar: { path: 'github.har.zip' }});// ... do stuff ...await context.close();Use the new methods page.routeFromHAR() or browserContext.routeFromHAR() to serve matching responses from the HAR file:await context.routeFromHAR('github.har.zip');Read more in our documentation.Advanced Routing You can now use route.fallback() to defer routing to other handlers.Consider the following example:// Remove a header from all requests.test.beforeEach(async ({ page }) => { await page.route('**/*', async route => {   const headers = await route.request().allHeaders();   delete headers['if-none-match'];   route.fallback({ headers }); });});test('should work', async ({ page }) => { await page.route('**/*', route => {   if (route.request().resourceType() === 'image')   route.abort();   else   route.fallback(); });});Note that the new methods page.routeFromHAR() and browserContext.routeFromHAR() also participate in routing and could be deferred to.Web-First Assertions Update New method expect(locator).toHaveValues() that asserts all selected values of <select multiple> element.Methods expect(locator).toContainText() and expect(locator).toHaveText() now accept ignoreCase option.Component Tests Update Support for Vue2 via the @playwright/experimental-ct-vue2 package.Support for component tests for create-react-app with components in .js files.Read more about component testing with Playwright.Miscellaneous If there's a service worker that's in your way, you can now easily disable it with a new context option serviceWorkers:playwright.config.tsexport default { use: {   serviceWorkers: 'block', }};Using .zip path for recordHar context option automatically zips the resulting HAR:const context = await browser.newContext({ recordHar: {   path: 'github.har.zip', }});If you intend to edit HAR by hand, consider using the "minimal" HAR recording mode that only records information that is essential for replaying:const context = await browser.newContext({ recordHar: {   path: 'github.har',   mode: 'minimal', }});Playwright now runs on Ubuntu 22 amd64 and Ubuntu 22 arm64. We also publish new docker image mcr.microsoft.com/playwright:v1.34.0-jammy.& þ Breaking Changes & þ   WebServer is now considered "ready" if request to the specified url has any of the following HTTP status codes:200-299300-399 (new)400, 401, 402, 403 (new)Version 1.22 Highlights Components Testing (preview)Playwright Test can now test your React, Vue.js or Svelte components. You can use all the features of Playwright Test (such as parallelization, emulation & debugging) while running components in real browsers.Here is what a typical component test looks like:App.spec.tsximport { test, expect } from '@playwright/experimental-ct-react';import App from './App';// Let's test component in a dark scheme!test.use({ colorScheme: 'dark' });test('should render', async ({ mount }) => { const component = await mount(<App></App>); // As with any Playwright test, assert locator text. await expect(component).toContainText('React'); // Or do a screenshot Ø=Þ€ await expect(component).toHaveScreenshot(); // Or use any Playwright method await component.click();});Read more in our documentation.Role selectors that allow selecting elements by their ARIA role, ARIA attributes and accessible name.// Click a button with accessible name "log in"await page.locator('role=button[name="log in"]').click();Read more in our documentation.New locator.filter() API to filter an existing locatorconst buttons = page.locator('role=button');// ...const submitButton = buttons.filter({ hasText: 'Submit' });await submitButton.click();New web-first assertions expect(page).toHaveScreenshot() and expect(locator).toHaveScreenshot() that wait for screenshot stabilization and enhances test reliability.The new assertions has screenshot-specific defaults, such as:disables animationsuses CSS scale optionawait page.goto('https://playwright.dev');await expect(page).toHaveScreenshot();The new expect(page).toHaveScreenshot() saves screenshots at the same location as expect(snapshot).toMatchSnapshot().Version 1.21 Highlights New role selectors that allow selecting elements by their ARIA role, ARIA attributes and accessible name.// Click a button with accessible name "log in"await page.locator('role=button[name="log in"]').click();Read more in our documentation.New scale option in page.screenshot() for smaller sized screenshots.New caret option in page.screenshot() to control text caret. Defaults to "hide".New method expect.poll to wait for an arbitrary condition:// Poll the method until it returns an expected result.await expect.poll(async () => { const response = await page.request.get('https://api.example.com'); return response.status();}).toBe(200);expect.poll supports most synchronous matchers, like .toBe(), .toContain(), etc. Read more in our documentation.Behavior Changes ESM support when running TypeScript tests is now enabled by default. The PLAYWRIGHT_EXPERIMENTAL_TS_ESM env variable is no longer required.The mcr.microsoft.com/playwright docker image no longer contains Python. Please use mcr.microsoft.com/playwright/python as a Playwright-ready docker image with pre-installed Python.Playwright now supports large file uploads (100s of MBs) via locator.setInputFiles() API.Browser Versions Chromium 101.0.4951.26Mozilla Firefox 98.0.2WebKit 15.4This version was also tested against the following stable channels:Google Chrome 100Microsoft Edge 100Version 1.20 Highlights New options for methods page.screenshot(), locator.screenshot() and elementHandle.screenshot():Option animations: "disabled" rewinds all CSS animations and transitions to a consistent stateOption mask: Locator[] masks given elements, overlaying them with pink #FF00FF boxes.expect().toMatchSnapshot() now supports anonymous snapshots: when snapshot name is missing, Playwright Test will generate one automatically:expect('Web is Awesome <3').toMatchSnapshot();New maxDiffPixels and maxDiffPixelRatio options for fine-grained screenshot comparison using expect().toMatchSnapshot():expect(await page.screenshot()).toMatchSnapshot({ maxDiffPixels: 27, // allow no more than 27 different pixels.});It is most convenient to specify maxDiffPixels or maxDiffPixelRatio once in testConfig.expect.Playwright Test now adds testConfig.fullyParallel mode. By default, Playwright Test parallelizes between files. In fully parallel mode, tests inside a single file are also run in parallel. You can also use --fully-parallel command line flag.playwright.config.tsexport default { fullyParallel: true,};testProject.grep and testProject.grepInvert are now configurable per project. For example, you can now configure smoke tests project using grep:playwright.config.tsexport default { projects: [{   name: 'smoke tests',   grep: /@smoke/,   },],};Trace Viewer now shows API testing requests.locator.highlight() visually reveals element(s) for easier debugging.Announcements We now ship a designated Python docker image mcr.microsoft.com/playwright/python. Please switch over to it if you use Python. This is the last release that includes Python inside our javascript mcr.microsoft.com/playwright docker image.v1.20 is the last release to receive WebKit update for macOS 10.15 Catalina. Please update MacOS to keep using latest & greatest WebKit!Browser Versions Chromium 101.0.4921.0Mozilla Firefox 97.0.1WebKit 15.4This version was also tested against the following stable channels:Google Chrome 99Microsoft Edge 99Version 1.19 Playwright Test Update Playwright Test v1.19 now supports soft assertions. Failed soft assertionsdo not terminate test execution, but mark the test as failed.// Make a few checks that will not stop the test when failed...await expect.soft(page.locator('#status')).toHaveText('Success');await expect.soft(page.locator('#eta')).toHaveText('1 day');// ... and continue the test to check more things.await page.locator('#next-page').click();await expect.soft(page.locator('#title')).toHaveText('Make another order');Read more in our documentationYou can now specify a custom error message as a second argument to the expect and expect.soft functions, for example:await expect(page.locator('text=Name'), 'should be logged in').toBeVisible();The error would look like this: Error: should be logged in   Call log: - expect.toBeVisible with timeout 5000ms   - waiting for "getByText('Name')"   2 | 3 | test('example test', async({ page }) => {   > 4 | await expect(page.locator('text=Name'), 'should be logged in').toBeVisible(); | ^   5 | });   6 |Read more in our documentationBy default, tests in a single file are run in order. If you have many independent tests in a single file, you can now run them in parallel with test.describe.configure().Other Updates Locator now supports a has option that makes sure it contains another locator inside:await page.locator('article', { has: page.locator('.highlight'),}).click();Read more in locator documentationNew locator.page()page.screenshot() and locator.screenshot() now automatically hide blinking caretPlaywright Codegen now generates locators and frame locatorsNew option url in testConfig.webServer to ensure your web server is ready before running the testsNew testInfo.errors and testResult.errors that contain all failed assertions and soft assertions.Potentially breaking change in Playwright Test Global Setup It is unlikely that this change will affect you, no action is required if your tests keep running as they did.We've noticed that in rare cases, the set of tests to be executed was configured in the global setup by means of the environment variables. We also noticed some applications that were post processing the reporters' output in the global teardown. If you are doing one of the two, learn moreBrowser Versions Chromium 100.0.4863.0Mozilla Firefox 96.0.1WebKit 15.4This version was also tested against the following stable channels:Google Chrome 98Microsoft Edge 98Version 1.18 Locator Improvements locator.dragTo()expect(locator).toBeChecked({ checked })Each locator can now be optionally filtered by the text it contains:await page.locator('li', { hasText: 'my item' }).locator('button').click();Read more in locator documentationTesting API improvements expect(response).toBeOK()testInfo.attach()test.info()Improved TypeScript Support Playwright Test now respects tsconfig.json's baseUrl and paths, so you can use aliasesThere is a new environment variable PW_EXPERIMENTAL_TS_ESM that allows importing ESM modules in your TS code, without the need for the compile step. Don't forget the .js suffix when you are importing your esm modules. Run your tests as follows:npm i --save-dev @playwright/test@1.18.0-rc1PW_EXPERIMENTAL_TS_ESM=1 npx playwright testCreate Playwright The npm init playwright command is now generally available for your use:# Run from your project's root directorynpm init playwright@latest# Or create a new projectnpm init playwright@latest new-projectThis will create a Playwright Test configuration file, optionally add examples, a GitHub Action workflow and a first test example.spec.ts.New APIs & changes new testCase.repeatEachIndex APIacceptDownloads option now defaults to trueBreaking change: custom config options Custom config options are a convenient way to parametrize projects with different values. Learn more in this guide.Previously, any fixture introduced through test.extend() could be overridden in the testProject.use config section. For example,// WRONG: THIS SNIPPET DOES NOT WORK SINCE v1.18.// fixtures.jsconst test = base.extend({ myParameter: 'default',});// playwright.config.jsmodule.exports = { use: {   myParameter: 'value', },};The proper way to make a fixture parametrized in the config file is to specify option: true when defining the fixture. For example,// CORRECT: THIS SNIPPET WORKS SINCE v1.18.// fixtures.jsconst test = base.extend({ // Fixtures marked as "option: true" will get a value specified in the config, // or fallback to the default value. myParameter: ['default', { option: true }],});// playwright.config.jsmodule.exports = { use: {   myParameter: 'value', },};Browser Versions Chromium 99.0.4812.0Mozilla Firefox 95.0WebKit 15.4This version was also tested against the following stable channels:Google Chrome 97Microsoft Edge 97Version 1.17 Frame Locators Playwright 1.17 introduces frame locators - a locator to the iframe on the page. Frame locators capture the logic sufficient to retrieve the iframe and then locate elements in that iframe. Frame locators are strict by default, will wait for iframe to appear and can be used in Web-First assertions.Frame locators can be created with either page.frameLocator() or locator.frameLocator() method.const locator = page.frameLocator('#my-iframe').locator('text=Submit');await locator.click();Read more at our documentation.Trace Viewer Update Playwright Trace Viewer is now available online at https://trace.playwright.dev! Just drag-and-drop your trace.zip file to inspect its contents.NOTE: trace files are not uploaded anywhere; trace.playwright.dev is a progressive web application that processes traces locally.Playwright Test traces now include sources by default (these could be turned off with tracing option)Trace Viewer now shows test nameNew trace metadata tab with browser detailsSnapshots now have URL barHTML Report Update HTML report now supports dynamic filteringReport is now a single static HTML file that could be sent by e-mail or as a slack attachment.Ubuntu ARM64 support + more Playwright now supports Ubuntu 20.04 ARM64. You can now run Playwright tests inside Docker on Apple M1 and on Raspberry Pi.You can now use Playwright to install stable version of Edge on Linux:npx playwright install msedgeNew APIs Tracing now supports a 'title' optionPage navigations support a new 'commit' waiting optionHTML reporter got new configuration optionstestConfig.snapshotDir optiontestInfo.parallelIndextestInfo.titlePathtestOptions.trace has new optionsexpect.toMatchSnapshot supports subdirectoriesreporter.printsToStdio()Version 1.16 Ø<ß- Playwright Test API Testing Playwright 1.16 introduces new API Testing that lets you send requests to the server directly from Node.js! Now you can:test your server APIprepare server side state before visiting the web application in a testvalidate server side post-conditions after running some actions in the browserTo do a request on behalf of Playwright's Page, use new page.request API:import { test, expect } from '@playwright/test';test('context fetch', async ({ page }) => { // Do a GET request on

behalf of page const response = await page.request.get('http://example.com/
foo.json'); // ...});To do a stand-alone request from node.js to an API endpoint, use new
request fixture:import { test, expect } from '@playwright/test';test('context fetch', async
({ request }) => { // Do a GET request on behalf of page const response = await
request.get('http://example.com/foo.json'); // ...});Read more about it in our API testing
guide.Response Interception It is now possible to do response interception by
combining API Testing with request interception.For example, we can blur all the
images on the page:import { test, expect } from '@playwright/test';import jimp from
'jimp'; // image processing librarytest('response interception', async ({ page }) =>
{ await page.route('**/*.jpeg', async route => {   const response = await
page._request.fetch(route.request());   const image = await jimp.read(await
response.body());   await image.blur(5);   route.fulfill({   response,   body: await
image.getBufferAsync('image/jpeg'),   }); }); const response = await page.goto('https://
playwright.dev'); expect(response.status()).toBe(200);});Read more about response
interception.New HTML reporter Try it out new HTML reporter with either --
reporter=html or a reporter entry in playwright.config.ts file:$ npx playwright test --
reporter=htmlThe HTML reporter has all the information about tests and their failures,
including surfacing trace and image artifacts.Read more about our reporters.Ø<ß- Playwright
Library locator.waitFor Wait for a locator to resolve to a single element with a given state.
Defaults to the state: 'visible'.Comes especially handy when working with lists:import
{ test, expect } from '@playwright/test';test('context fetch', async ({ page }) => { const
completeness = page.locator('text=Success'); await completeness.waitFor();
expect(await page.screenshot()).toMatchSnapshot('screen.png');});Read more about
locator.waitFor().Docker support for Arm64 Playwright Docker image is now published
for Arm64 so it can be used on Apple Silicon.Read more about Docker integration.Ø<ß-
Playwright Trace Viewer web-first assertions inside trace viewerrun trace viewer with
npx playwright show-trace and drop trace files to the trace viewer PWAAPI testing is
integrated with trace viewerbetter visual attribution of action targetsRead more about
Trace Viewer.Browser Versions Chromium 97.0.4666.0Mozilla Firefox 93.0WebKit
15.4This version of Playwright was also tested against the following stable
channels:Google Chrome 94Microsoft Edge 94Version 1.15 Ø<ß- Playwright Library Ø=Ý±þ Mouse
Wheel By using Page.mouse.wheel you are now able to scroll vertically or horizontally.Ø=ÜÜ
New Headers API Previously it was not possible to get multiple header values of a
response. This is now possible and additional helper functions are
available:Request.allHeaders()Request.headersArray()Request.headerValue(name:
string)Response.allHeaders()Response.headersArray()Response.headerValue(name:
string)Response.headerValues(name: string)Ø<ß Forced-Colors emulation Its now possible
to emulate the forced-colors CSS media feature by passing it in the context options or
calling Page.emulateMedia().New APIs Page.route() accepts new times option to
specify how many times this route should be matched.Page.setChecked(selector:
string, checked: boolean) and Locator.setChecked(selector: string, checked: boolean)
was introduced to set the checked state of a checkbox.Request.sizes() Returns
resource size information for given http request.BrowserContext.tracing.startChunk() -
Start a new trace chunk.BrowserContext.tracing.stopChunk() - Stops a new trace chunk.
Ø<ß- Playwright Test Ø>Ý test.parallel() run tests in the same file in parallel
test.describe.parallel('group', () => { test('runs in parallel 1', async ({ page }) => { });

test('runs in parallel 2', async ({ page }) => { });});By default, tests in a single file are run in order. If you have many independent tests in a single file, you can now run them in parallel with test.describe.parallel(title, callback).Ø=Þà Add --debug CLI flag By using npx playwright test --debug it will enable the Playwright Inspector for you to debug your tests.Browser Versions Chromium 96.0.4641.0Mozilla Firefox 92.0WebKit 15.0Version 1.14 Ø<ß- Playwright Library &¡þ New "strict" mode Selector ambiguity is a common problem in automation testing. "strict" mode ensures that your selector points to a single element and throws otherwise.Pass strict: true into your action calls to opt in.// This will throw if you have more than one button!await page.click('button', { strict: true });Ø=ÜÍ New Locators API Locator represents a view to the element(s) on the page. It captures the logic sufficient to retrieve the element at any given moment.The difference between the Locator and ElementHandle is that the latter points to a particular element, while Locator captures the logic of how to retrieve that element.Also, locators are "strict" by default!const locator = page.locator('button');await locator.click();Learn more in the documentation.Ø>Ýé Experimental React and Vue selector engines React and Vue selectors allow selecting elements by its component name and/or property values. The syntax is very similar to attribute selectors and supports all attribute selector operators.await page.locator('_react=SubmitButton[enabled=true]').click();await page.locator('_vue=submit-button[enabled=true]').click();Learn more in the react selectors documentation and the vue selectors documentation.'(New nth and visible selector engines nth selector engine is equivalent to the :nth-match pseudo class, but could be combined with other selector engines.visible selector engine is equivalent to the :visible pseudo class, but could be combined with other selector engines.// select the first button among all buttonsawait button.click('button >> nth=0');// or if you are using locators, you can use first(), nth() and last()await page.locator('button').first().click();// click a visible buttonawait button.click('button >> visible=true');Ø<ß- Playwright Test ' Web-First Assertions expect now supports lots of new web-first assertions.Consider the following example:await expect(page.locator('.status')).toHaveText('Submitted');Playwright Test will be re-testing the node with the selector .status until fetched Node has the "Submitted" text. It will be re-fetching the node and checking it over and over, until the condition is met or until the timeout is reached. You can either pass this timeout or configure it once via the testProject.expect value in test config.By default, the timeout for assertions is not set, so it'll wait forever, until the whole test times out.List of all new assertions:expect(locator).toBeChecked()expect(locator).toBeDisabled()expect(locator).toBeEditable()expect(locator).toBeEmpty()expect(locator).toBeEnabled()expect(locator).toBeFocused()expect(locator).toBeHidden()expect(locator).toBeVisible()expect(locator).toContainText(text, options?)expect(locator).toHaveAttribute(name, value)expect(locator).toHaveClass(expected)expect(locator).toHaveCount(count)expect(locator).toHaveCSS(name, value)expect(locator).toHaveId(id)expect(locator).toHaveJSProperty(name, value)expect(locator).toHaveText(expected, options)expect(page).toHaveTitle(title)expect(page).toHaveURL(url)expect(locator).toHaveValue(value)&Ó Serial mode with describe.serial Declares a group of tests that should always be run serially. If one of the tests fails, all subsequent tests are skipped. All tests in a group are retried together.test.describe.serial('group', () => { test('runs first', async ({ page }) => { /* ... */ }); test('runs second', async ({ page }) => { /* ... */ });});Learn more in the documentation.Ø=Ü> Steps API with test.step Split long tests into multiple steps using test.step() API:import { test, expect } from '@playwright/test';test('test', async ({ page }) => { await test.step('Log in', async () => {   // ... }); await test.step('news feed', async () => {   // ... });});Step information is exposed in reporters API.Ø<ß Launch web server before running tests To launch a server during the tests, use the webServer option in the configuration file. The server will wait for a given url to be available before running the tests, and the url will be passed over to Playwright as a baseURL when creating a context.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ webServer: {   command: 'npm run start', // command to launch   url: 'http://127.0.0.1:3000', // url to await for   timeout: 120 * 1000,   reuseExistingServer: !process.env.CI, },});Learn more in the documentation.Browser Versions Chromium 94.0.4595.0Mozilla Firefox 91.0WebKit 15.0Version 1.13 Playwright Test &¡þ Introducing Reporter API which is already used to create an Allure Playwright reporter.&úþ New baseURL fixture to support relative paths in tests.Playwright Ø=Ý– Programmatic drag-and-drop support via the page.dragAndDrop() API.Ø=Ý Enhanced HAR with body sizes for requests and responses. Use via recordHar option in browser.newContext().Tools Playwright Trace Viewer now shows parameters, returned values and console.log() calls.Playwright Inspector can generate Playwright Test tests.New and Overhauled Guides IntroAuthenticationChrome ExtensionsPlaywright Test AnnotationsPlaywright Test ConfigurationPlaywright Test FixturesBrowser Versions Chromium 93.0.4576.0Mozilla Firefox 90.0WebKit 14.2New Playwright APIs new baseURL option in browser.newContext() and browser.newPage()response.securityDetails() and response.serverAddr()page.dragAndDrop() and frame.dragAndDrop()download.cancel()page.inputValue(), frame.inputValue() and elementHandle.inputValue()new force option in page.fill(), frame.fill(), and elementHandle.fill()new force option in page.selectOption(), frame.selectOption(), and elementHandle.selectOption()Version 1.12 &¡þ Introducing Playwright Test Playwright Test is a new test runner built from scratch by Playwright team specifically to accommodate end-to-end testing needs:Run tests across all browsers.Execute tests in parallel.Enjoy context isolation and sensible defaults out of the box.Capture videos, screenshots and other artifacts on failure.Integrate your POMs as extensible fixtures.Installation:npm i -D @playwright/testSimple test tests/foo.spec.ts:import { test, expect } from '@playwright/test';test('basic test', async ({ page }) => { await page.goto('https://playwright.dev/'); const name = await page.innerText('.navbar__title'); expect(name).toBe('Playwright');});Running:npx playwright testØ=ÜI Read more in Playwright Test documentation.Ø>Ýß &Bþ Introducing Playwright Trace Viewer Playwright Trace Viewer is a new GUI tool that helps exploring recorded Playwright traces after the script ran. Playwright traces let you examine:page DOM before and after each Playwright actionpage rendering before and after each Playwright actionbrowser network during script executionTraces are recorded using the new browserContext.tracing API:const browser = await chromium.launch();const context = await browser.newContext();// Start tracing before creating / navigating a page.await context.tracing.start({ screenshots: true, snapshots: true });const page = await context.newPage();await page.goto('https://playwright.dev');// Stop tracing and export it into a zip archive.await context.tracing.stop({ path: 'trace.zip' });Traces are examined later with the Playwright CLI:npx playwright show-trace trace.zipThat will open the following GUI:Ø=ÜI Read more in trace viewer documentation.Browser Versions Chromium 93.0.4530.0Mozilla Firefox 89.0WebKit 14.2This version of Playwright was also tested against the following stable channels:Google Chrome 91Microsoft Edge 91New APIs reducedMotion option in page.emulateMedia(), browserType.launchPersistentContext(), browser.newContext() and browser.newPage()browserContext.on('request')browserContext.on('requestfailed')browserContext.on('requestfinished')browserContext.on('response')tracesDir option in browserType.launch() and browserType.launchPersistentContext()new browserContext.tracing API namespacenew download.page() methodVersion 1.11 Ø<ß¥ New video: Playwright: A New Test Automation Framework for the Modern Web (slides)We talked about PlaywrightShowed engineering work behind the scenesDid live demos with new features '(Special thanks to applitools for hosting the event and inviting us!Browser Versions Chromium 92.0.4498.0Mozilla Firefox 89.0b6WebKit 14.2New APIs support for async predicates across the API in methods such as page.waitForRequest() and othersnew emulation devices: Galaxy S8, Galaxy S9+, Galaxy Tab S4, Pixel 3, Pixel 4new methods:page.waitForURL() to await navigations to URLvideo.delete() and video.saveAs() to manage screen recordingnew options:screen option in the browser.newContext() method to emulate window.screen dimensionsposition option in page.check() and page.uncheck() methodstrial option to dry-run actions in page.check(), page.uncheck(), page.click(), page.dblclick(), page.hover() and page.tap()Version 1.10 Playwright for Java v1.10 is now stable!Run Playwright against Google Chrome and Microsoft Edge stable channels with the new channels API.Chromium screenshots are fast on Mac & Windows.Bundled Browser Versions Chromium 90.0.4430.0Mozilla Firefox 87.0b10WebKit 14.2This version of Playwright was also tested against the following stable channels:Google Chrome 89Microsoft Edge 89New APIs browserType.launch() now accepts the new 'channel' option. Read more in our documentation.Version 1.9 Playwright Inspector is a new GUI tool to author and debug your tests.Line-by-line debugging of your Playwright scripts, with play, pause and step-through.Author new scripts by recording user actions.Generate element selectors for your script by hovering over elements.Set the PWDEBUG=1 environment variable to launch the InspectorPause script execution with page.pause() in headed mode. Pausing the page launches Playwright Inspector for debugging.New has-text pseudo-class for CSS selectors. :has-text("example") matches any element containing "example" somewhere inside, possibly in a child or a descendant element. See more examples.Page dialogs are now auto-dismissed during execution, unless a listener for dialog event is configured. Learn more about this.Playwright for Python is now stable with an idiomatic snake case API and pre-built Docker image to run tests in CI/CD.Browser Versions Chromium 90.0.4421.0Mozilla Firefox 86.0b10WebKit 14.1New APIs page.pause().Version 1.8 Selecting elements based on layout with :left-of(), :right-of(), :above() and :below().Playwright now includes command line interface, former playwright-cli.npx playwright --helppage.selectOption() now waits for the options to be present.New methods to assert element state like page.isEditable().New APIs elementHandle.isChecked().elementHandle.isDisabled().elementHandle.isEditable().elementHandle.isEnabled().elementHandle.isHidden().elementHandle.isVisible().page.isChecked().page.isDisabled().page.isEditable().page.isEnabled().page.isHidden().page.isVisible().New option 'editable' in elementHandle.waitForElementState().Browser Versions Chromium 90.0.4392.0Mozilla Firefox 85.0b5WebKit 14.1Version 1.7 New Java SDK: Playwright for

Java is now on par with JavaScript, Python and .NET bindings.Browser storage API: New convenience APIs to save and load browser storage state (cookies, local storage) to simplify automation scenarios with authentication.New CSS selectors: We heard your feedback for more flexible selectors and have revamped the selectors implementation. Playwright 1.7 introduces new CSS extensions and there's more coming soon.New website: The docs website at playwright.dev has been updated and is now built with Docusaurus.Support for Apple Silicon: Playwright browser binaries for WebKit and Chromium are now built for Apple Silicon.New APIs browserContext.storageState() to get current state for later reuse.storageState option in browser.newContext() and browser.newPage() to setup browser context state.Browser Versions Chromium 89.0.4344.0Mozilla Firefox 84.0b9WebKit 14.1
Canary releasesPlaywright for Node.js has a canary releases system.It permits you to test new unreleased features instead of waiting for a full release. They get released daily on the next NPM tag of Playwright.It is a good way to give feedback to maintainers, ensuring the newly implemented feature works as intended.note Using a canary release in production might seem risky, but in practice, it's not. A canary release passes all automated tests and is used to test e.g. the HTML report, Trace Viewer, or Playwright Inspector with end-to-end tests. npm install -D @playwright/test@nextNext npm Dist Tag For any code-related commit on main, the continuous integration will publish a daily canary release under the @next npm dist tag.You can see on npm the current dist tags:latest: stable releasesnext: next releases, published dailybeta: after a release-branch was cut, usually a week before a stable release each commit gets published under this tagUsing a Canary Release npm install -D @playwright/test@nextDocumentation The stable and the next documentation is published on playwright.dev. To see the next documentation, press Shift on the keyboard 5 times.
Test configurationPlaywright has many options to configure how your tests are run. You can specify these options in the configuration file. Note that test runner options are top-level, do not put them into the use section.Basic Configuration Here are some of the most common configuration options.import { defineConfig, devices } from '@playwright/test';export default defineConfig({ // Look for test files in the "tests" directory, relative to this configuration file. testDir: 'tests', // Run all tests in parallel. fullyParallel: true, // Fail the build on CI if you accidentally left test.only in the source code. forbidOnly: !!process.env.CI, // Retry on CI only. retries: process.env.CI ? 2 : 0, // Opt out of parallel tests on CI. workers: process.env.CI ? 1 : undefined, // Reporter to use reporter: 'html', use: {   // Base URL to use in actions like `await page.goto('/')`.   baseURL: 'http://127.0.0.1:3000',   // Collect trace when retrying the failed test.   trace: 'on-first-retry', }, // Configure projects for major browsers. projects: [{   name: 'chromium',   use: { ...devices['Desktop Chrome'] },   },], // Run your local dev server before starting the tests. webServer: {   command: 'npm run start',   url: 'http://127.0.0.1:3000',   reuseExistingServer: !process.env.CI, },});OptionDescriptiontestConfig.forbidOnlyWhether to exit with an error if any tests are marked as test.only. Useful on CI.testConfig.fullyParallelhave all tests in all files to run in parallel. See /Parallelism and sharding for more details.testConfig.projectsRun tests in multiple configurations or on multiple browserstestConfig.reporterReporter to use. See Test Reporters to learn more about

which reporters are available.testConfig.retriesThe maximum number of retry attempts per test. See Test Retries to learn more about retries.testConfig.testDirDirectory with the test files.testConfig.useOptions with use{}testConfig.webServerTo launch a server during the tests, use the webServer optiontestConfig.workersThe maximum number of concurrent worker processes to use for parallelizing tests. Can also be set as percentage of logical CPU cores, e.g. '50%'.. See /Parallelism and sharding for more details.Filtering Tests Filter tests by glob patterns or regular expressions.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ // Glob patterns or regular expressions to ignore test files. testIgnore: '*test-assets', // Glob patterns or regular expressions that match test files. testMatch: '*todo-tests/*.spec.ts',});OptionDescriptiontestConfig.testIgnoreGlob patterns or regular expressions that should be ignored when looking for the test files. For example, '*test-assets'testConfig.testMatchGlob patterns or regular expressions that match test files. For example, '*todo-tests/*.spec.ts'. By default, Playwright runs .*(test|spec).(js|ts|mjs) files.Advanced Configuration playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ // Folder for test artifacts such as screenshots, videos, traces, etc. outputDir: 'test-results', // path to the global setup files. globalSetup: require.resolve('./global-setup'), // path to the global teardown files. globalTeardown: require.resolve('./global-teardown'), // Each test is given 30 seconds. timeout: 30000,});OptionDescriptiontestConfig.globalSetupPath to the global setup file. This file will be required and run before all the tests. It must export a single function.testConfig.globalTeardownPath to the global teardown file. This file will be required and run after all the tests. It must export a single function.testConfig.outputDirFolder for test artifacts such as screenshots, videos, traces, etc.testConfig.timeoutPlaywright enforces a timeout for each test, 30 seconds by default. Time spent by the test function, fixtures, beforeEach and afterEach hooks is included in the test timeout.Expect Options Configuration for the expect assertion library.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ expect: {   // Maximum time expect() should wait for the condition to be met.   timeout: 5000,   toHaveScreenshot: {   // An acceptable amount of pixels that could be different, unset by default.   maxDiffPixels: 10,   },   toMatchSnapshot: {   // An acceptable ratio of pixels that are different to the total amount of pixels, between 0 and 1.   maxDiffPixelRatio: 0.1,   }, },});OptionDescriptiontestConfig.expectWeb first assertions like expect(locator).toHaveText() have a separate timeout of 5 seconds by default. This is the maximum time the expect() should wait for the condition to be met. Learn more about test and expect timeouts and how to set them for a single test.expect(page).toHaveScreenshot()Configuration for the expect(locator).toHaveScreeshot() method.expect(snapshot).toMatchSnapshot()Configuration for the expect(locator).toMatchSnapshot() method.Add custom matchers using expect.extend You can extend Playwright assertions by providing custom matchers. These matchers will be available on the expect object.In this example we add a custom toBeWithinRange function in the configuration file. Custom matcher should return a message callback and a pass flag indicating whether the assertion passed.TypeScriptJavaScriptplaywright.config.tsimport { expect, defineConfig } from '@playwright/test';expect.extend({ toBeWithinRange(received: number, floor: number,

ceiling: number) {   const pass = received >= floor && received <= ceiling;   if (pass) {   return {   message: () => 'passed',   pass: true,   };   } else {   return {   message: () => 'failed',   pass: false,   };   } },});export default defineConfig({});playwright.config.tsconst { expect, defineConfig } = require('@playwright/test');expect.extend({ toBeWithinRange(received, floor, ceiling) {   const pass = received >= floor && received <= ceiling;   if (pass) {   return {   message: () => 'passed',   pass: true,   };   } else {   return {   message: () => 'failed',   pass: false,   };   } },});module.exports = defineConfig({});Now we can use toBeWithinRange in the test.example.spec.tsimport { test, expect } from '@playwright/test';test('numeric ranges', () => { expect(100).toBeWithinRange(90, 110); expect(101).not.toBeWithinRange(0, 100);});noteDo not confuse Playwright's expect with the expect library. The latter is not fully integrated with Playwright test runner, so make sure to use Playwright's own expect.For TypeScript, also add the following to your global.d.ts. If it does not exist, you need to create it inside your repository. Make sure that your global.d.ts gets included inside your tsconfig.json via the include or compilerOptions.typeRoots option so that your IDE will pick it up.You don't need it for JavaScript.global.d.tsexport {};declare global { namespace PlaywrightTest {   interface Matchers<R, T> {   toBeWithinRange(a: number, b: number): R;   } }}
Test use optionsIn addition to configuring the test runner you can also configure Emulation, Network and Recording for the Browser or BrowserContext. These options are passed to the use: {} object in the Playwright config.Basic Options Set the base URL and storage state for all tests:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Base URL to use in actions like `await page.goto('/')`.   baseURL: 'http://127.0.0.1:3000',   // Populates context with given storage state.   storageState: 'state.json', },});OptionDescriptiontestOptions.baseURLBase URL used for all pages in the context. Allows navigating by using just the path, for example page.goto('/settings').testOptions.storageStatePopulates context with given storage state. Useful for easy authentication, learn more.Emulation Options With Playwright you can emulate a real device such as a mobile phone or tablet. See our guide on projects for more info on emulating devices. You can also emulate the "geolocation", "locale" and "timezone" for all tests or for a specific test as well as set the "permissions" to show notifications or change the "colorScheme". See our Emulation guide to learn more.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Emulates `'prefers-colors-scheme'` media feature.   colorScheme: 'dark',   // Context geolocation.   geolocation: { longitude: 12.492507, latitude: 41.889938 },   // Emulates the user locale.   locale: 'en-GB',   // Grants specified permissions to the browser context.   permissions: ['geolocation'],   // Emulates the user timezone.   timezoneId: 'Europe/Paris',   // Viewport used for all pages in the context.   viewport: { width: 1280, height: 720 }, },});OptionDescriptiontestOptions.colorSchemeEmulates 'prefers-colors-scheme' media feature, supported values are 'light', 'dark', 'no-preference'testOptions.geolocationContext geolocation.testOptions.localeEmulates the user locale, for example en-GB, de-DE, etc.testOptions.permissionsA list of permissions to grant to all pages in the context.testOptions.timezoneIdChanges the

timezone of the context.testOptions.viewportViewport used for all pages in the context.Network Options Available options to configure networking:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Whether to automatically download all the attachments.   acceptDownloads: false,   // An object containing additional HTTP headers to be sent with every request.   extraHTTPHeaders: {   'X-My-Header': 'value',   },   // Credentials for HTTP authentication.   httpCredentials: {   username: 'user',   password: 'pass',   },   // Whether to ignore HTTPS errors during navigation.   ignoreHTTPSErrors: true,   // Whether to emulate network being offline.   offline: true,   // Proxy settings used for all pages in the test.   proxy: {   server: 'http://myproxy.com:3128',   bypass: 'localhost',   }, },});OptionDescriptiontestOptions.acceptDownloadsWhether to automatically download all the attachments, defaults to true. Learn more about working with downloads.testOptions.extraHTTPHeadersAn object containing additional HTTP headers to be sent with every request. All header values must be strings.testOptions.httpCredentialsCredentials for HTTP authentication.testOptions.ignoreHTTPSErrorsWhether to ignore HTTPS errors during navigation.testOptions.offlineWhether to emulate network being offline.testOptions.proxyProxy settings used for all pages in the test.noteYou don't have to configure anything to mock network requests. Just define a custom Route that mocks the network for a browser context. See our network mocking guide to learn more.Recording Options With Playwright you can capture screenshots, record videos as well as traces of your test. By default these are turned off but you can enable them by setting the screenshot, video and trace options in your playwright.config.js file. Trace files, screenshots and videos will appear in the test output directory, typically test-results.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Capture screenshot after each test failure.   screenshot: 'only-on-failure',   // Record trace only when retrying a test for the first time.   trace: 'on-first-retry',   // Record video only when retrying a test for the first time.   video: 'on-first-retry' },});OptionDescriptiontestOptions.screenshotCapture screenshots of your test. Options include 'off', 'on' and 'only-on-failure'testOptions.tracePlaywright can produce test traces while running the tests. Later on, you can view the trace and get detailed information about Playwright execution by opening Trace Viewer. Options include: 'off', 'on', 'retain-on-failure' and 'on-first-retry'testOptions.videoPlaywright can record videos for your tests. Options include: 'off', 'on', 'retain-on-failure' and 'on-first-retry'Other Options playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Maximum time each action such as `click()` can take. Defaults to 0 (no limit).   actionTimeout: 0,   // Name of the browser that runs tests. For example `chromium`, `firefox`, `webkit`.   browserName: 'chromium',   // Toggles bypassing Content-Security-Policy.   bypassCSP: true,   // Channel to use, for example "chrome", "chrome-beta", "msedge", "msedge-beta".   channel: 'chrome',   // Run browser in headless mode.   headless: false,   // Change the default data-testid attribute.   testIdAttribute: 'pw-test-id', },});OptionDescriptiontestOptions.actionTimeoutTimeout for each Playwright action in milliseconds. Defaults to 0 (no timeout). Learn more about timeouts and how to set them for a single test.testOptions.browserNameName of the browser that runs tests. Defaults to 'chromium'. Options include chromium, firefox, or

webkit.testOptions.bypassCSPToggles bypassing Content-Security-Policy. Useful when CSP includes the production origin. Defaults to false.testOptions.channelBrowser channel to use. Learn more about different browsers and channels.testOptions.headlessWhether to run the browser in headless mode meaning no browser is shown when running tests. Defaults to true.testOptions.testIdAttributeChanges the default data-testid attribute used by Playwright locators.More browser and context options Any options accepted by browserType.launch() or browser.newContext() can be put into launchOptions or contextOptions respectively in the use section.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   launchOptions: {   slowMo: 50,   }, },});However, most common ones like headless or viewport are available directly in the use section - see basic options, emulation or network.Explicit Context Creation and Option Inheritance If using the built-in browser fixture, calling browser.newContext() will create a context with options inherited from the config:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   userAgent: 'some custom ua',   viewport: { width: 100, height: 100 }, },});An example test illustrating the initial context options are set:import { test, expect } from '@playwright/test';test('should inherit use options on context when using built-in browser fixture', async ({ browser }) => { const context = await browser.newContext(); const page = await context.newPage(); expect(await page.evaluate(() => navigator.userAgent)).toBe('some custom ua'); expect(await page.evaluate(() => window.innerWidth)).toBe(100); await context.close();});Configuration Scopes You can configure Playwright globally, per project, or per test. For example, you can set the locale to be used globally by adding locale to the use option of the Playwright config, and then override it for a specific project using the project option in the config. You can also override it for a specific test by adding test.use({}) in the test file and passing in the options.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   locale: 'en-GB' },});You can override options for a specific project using the project option in the Playwright config.import { defineConfig, devices } from '@playwright/test';export default defineConfig({ projects: [{   name: 'chromium',   use: {   ...devices['Desktop Chrome'],   locale: 'de-DE',   },   },],});You can override options for a specific test file by using the test.use() method and passing in the options. For example to run tests with the French locale for a specific test:import { test, expect } from '@playwright/test';test.use({ locale: 'fr-FR' });test('example', async ({ page }) => { // ...});The same works inside a describe block. For example to run tests in a describe block with the French locale:import { test, expect } from '@playwright/test';test.describe('french language block', () => { test.use({ locale: 'fr-FR' }); test('example', async ({ page }) => {   // ... });});
Test use optionsIn addition to configuring the test runner you can also configure Emulation, Network and Recording for the Browser or BrowserContext. These options are passed to the use: {} object in the Playwright config.Basic Options Set the base URL and storage state for all tests:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Base URL to use in actions like `await page.goto('/')`.   baseURL: 'http://127.0.0.1:3000',   // Populates context with given storage state.   storageState:

'state.json', },});OptionDescriptiontestOptions.baseURLBase URL used for all pages in the context. Allows navigating by using just the path, for example page.goto('/ settings').testOptions.storageStatePopulates context with given storage state. Useful for easy authentication, learn more.Emulation Options With Playwright you can emulate a real device such as a mobile phone or tablet. See our guide on projects for more info on emulating devices. You can also emulate the "geolocation", "locale" and "timezone" for all tests or for a specific test as well as set the "permissions" to show notifications or change the "colorScheme". See our Emulation guide to learn more.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Emulates `'prefers-colors-scheme'` media feature.   colorScheme: 'dark',   // Context geolocation.   geolocation: { longitude: 12.492507, latitude: 41.889938 },   // Emulates the user locale.   locale: 'en-GB',   // Grants specified permissions to the browser context.   permissions: ['geolocation'],   // Emulates the user timezone.   timezoneId: 'Europe/Paris',   // Viewport used for all pages in the context.   viewport: { width: 1280, height: 720 }, },});OptionDescriptiontestOptions.colorSchemeEmulates 'prefers-colors-scheme' media feature, supported values are 'light', 'dark', 'no-preference'testOptions.geolocationContext geolocation.testOptions.localeEmulates the user locale, for example en-GB, de-DE, etc.testOptions.permissionsA list of permissions to grant to all pages in the context.testOptions.timezoneIdChanges the timezone of the context.testOptions.viewportViewport used for all pages in the context.Network Options Available options to configure networking:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Whether to automatically download all the attachments.   acceptDownloads: false,   // An object containing additional HTTP headers to be sent with every request.   extraHTTPHeaders: {   'X-My-Header': 'value',   },   // Credentials for HTTP authentication.   httpCredentials: {   username: 'user',   password: 'pass',   },   // Whether to ignore HTTPS errors during navigation.   ignoreHTTPSErrors: true,   // Whether to emulate network being offline.   offline: true,   // Proxy settings used for all pages in the test.   proxy: {   server: 'http:// myproxy.com:3128',   bypass: 'localhost',   }, },});OptionDescriptiontestOptions.acceptDownloadsWhether to automatically download all the attachments, defaults to true. Learn more about working with downloads.testOptions.extraHTTPHeadersAn object containing additional HTTP headers to be sent with every request. All header values must be strings.testOptions.httpCredentialsCredentials for HTTP authentication.testOptions.ignoreHTTPSErrorsWhether to ignore HTTPS errors during navigation.testOptions.offlineWhether to emulate network being offline.testOptions.proxyProxy settings used for all pages in the test.noteYou don't have to configure anything to mock network requests. Just define a custom Route that mocks the network for a browser context. See our network mocking guide to learn more.Recording Options With Playwright you can capture screenshots, record videos as well as traces of your test. By default these are turned off but you can enable them by setting the screenshot, video and trace options in your playwright.config.js file. Trace files, screenshots and videos will appear in the test output directory, typically test-results.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Capture screenshot after each test failure.   screenshot: 'only-on-failure',   // Record trace only when retrying a test for the first time.   trace: 'on-first-retry',   // Record video only when retrying a test for the first time.   video: 'on-first-retry' },});OptionDescriptiontestOptions.screenshotCapture screenshots of your test. Options include 'off', 'on' and 'only-on-failure'testOptions.tracePlaywright can produce test traces while running the tests. Later on, you can view the trace and get detailed information about Playwright execution by opening Trace Viewer. Options include: 'off', 'on', 'retain-on-failure' and 'on-first-retry'testOptions.videoPlaywright can record videos for your tests. Options include: 'off', 'on', 'retain-on-failure' and 'on-first-retry'Other Options playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   // Maximum time each action such as `click()` can take. Defaults to 0 (no limit).   actionTimeout: 0,   // Name of the browser that runs tests. For example `chromium`, `firefox`, `webkit`.   browserName: 'chromium',   // Toggles bypassing Content-Security-Policy.   bypassCSP: true,   // Channel to use, for example "chrome", "chrome-beta", "msedge", "msedge-beta".   channel: 'chrome',   // Run browser in headless mode.   headless: false,   // Change the default data-testid attribute.   testIdAttribute: 'pw-test-id', },});OptionDescriptiontestOptions.actionTimeoutTimeout for each Playwright action in milliseconds. Defaults to 0 (no timeout). Learn more about timeouts and how to set them for a single test.testOptions.browserNameName of the browser that runs tests. Defaults to 'chromium'. Options include chromium, firefox, or webkit.testOptions.bypassCSPToggles bypassing Content-Security-Policy. Useful when CSP includes the production origin. Defaults to false.testOptions.channelBrowser channel to use. Learn more about different browsers and channels.testOptions.headlessWhether to run the browser in headless mode meaning no browser is shown when running tests. Defaults to true.testOptions.testIdAttributeChanges the default data-testid attribute used by Playwright locators.More browser and context options Any options accepted by browserType.launch() or browser.newContext() can be put into launchOptions or contextOptions respectively in the use section.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   launchOptions: {   slowMo: 50,   }, },});However, most common ones like headless or viewport are available directly in the use section - see basic options, emulation or network.Explicit Context Creation and Option Inheritance If using the built-in browser fixture, calling browser.newContext() will create a context with options inherited from the config:playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   userAgent: 'some custom ua',   viewport: { width: 100, height: 100 }, },});An example test illustrating the initial context options are set:import { test, expect } from '@playwright/test';test('should inherit use options on context when using built-in browser fixture', async ({ browser }) => { const context = await browser.newContext(); const page = await context.newPage(); expect(await page.evaluate(() => navigator.userAgent)).toBe('some custom ua'); expect(await page.evaluate(() => window.innerWidth)).toBe(100); await context.close();});Configuration Scopes You can configure Playwright globally, per project, or per test. For example, you can set the locale to be used globally by adding locale to the use option of the Playwright config, and then override it for a specific project using the project option in the config. You can also override it for a specific test by adding test.use({}) in the test file and passing in the options.playwright.config.tsimport { defineConfig } from '@playwright/test';export default defineConfig({ use: {   locale: 'en-GB' },});You can override options for a specific project using the project option in the Playwright config.import { defineConfig, devices } from '@playwright/test';export default defineConfig({ projects: [{   name: 'chromium',   use: {   ...devices['Desktop Chrome'],   locale: 'de-DE',   },   },],});You can override options for a specific test file by using the test.use() method and passing in the options. For example to run tests with the French locale for a specific test:import { test, expect } from '@playwright/test';test.use({ locale: 'fr-FR' });test('example', async ({ page }) => { // ...});The same works inside a describe block. For example to run tests in a describe block with the French locale:import { test, expect } from '@playwright/test';test.describe('french language block', () => { test.use({ locale: 'fr-FR' }); test('example', async ({ page }) => {   // ... });});