TypeScript

Introduction

Playwright supports TypeScript out of the box. You just write tests in TypeScript, and Playwright will read them, transform to JavaScript and run.

Note that Playwright does not check the types and will run tests even if there are non-critical TypeScript compilation errors. We recommend you run TypeScript compiler alongside Playwright. For example on GitHub actions:

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    ...
    - name: Run type checks
      run: npx tsc -p tsconfig.json --noEmit
    - name: Run Playwright tests
      run: npx playwright test
```

For local development, you can run `tsc` in watch mode like this:

```
npx tsc -p tsconfig.json --noEmit -w
```

tsconfig.json

Playwright will pick up `tsconfig.json` for each source file it loads. Note that Playwright **only supports** the following tsconfig options: `allowJs`, `baseUrl`, `paths` and `references`.

We recommend setting up a separate `tsconfig.json` in the tests directory so that you can change some preferences specifically for the tests. Here is an example directory structure.

```
src/
    source.ts

tests/
    tsconfig.json  # test-specific tsconfig
    example.spec.ts

tsconfig.json  # generic tsconfig for all typescript sources
```

```
playwright.config.ts
```

tsconfig path mapping

Playwright supports path mapping declared in the `tsconfig.json`. Make sure that `baseUrl` is also set.

Here is an example `tsconfig.json` that works with Playwright:

tsconfig.json
```
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@myhelper/*": ["packages/myhelper/*"] // This mapping is relative to "baseUrl".
    }
  }
}
```

You can now import using the mapped paths:

example.spec.ts
```
import { test, expect } from '@playwright/test';
import { username, password } from '@myhelper/credentials';

test('example', async ({ page }) => {
  await page.getByLabel('User Name').fill(username);
  await page.getByLabel('Password').fill(password);
});
```

tsconfig resolution

By default, Playwright will look up a closest tsconfig for each imported file by going up the directory structure and looking for `tsconfig.json` or `jsconfig.json`. This way, you can create a `tests/tsconfig.json` file that will be used only for your tests and Playwright will pick it up automatically.

```
# Playwright will choose tsconfig automatically
npx playwright test
```

Alternatively, you can specify a single tsconfig file to use in the command line, and Playwright will use it for all imported files, not only test files.

```
# Pass a specific tsconfig
npx playwright test --tsconfig=tsconfig.test.json
```

Manually compile tests with TypeScript

Sometimes, Playwright Test will not be able to transform your TypeScript code correctly, for example when you are using experimental or very recent features of TypeScript, usually configured in `tsconfig.json`.

In this case, you can perform your own TypeScript compilation before sending the tests to Playwright.

First add a `tsconfig.json` file inside the tests directory:

```
{
    "compilerOptions": {
        "target": "ESNext",
        "module": "commonjs",
        "moduleResolution": "Node",
        "sourceMap": true,
        "outDir": "../tests-out",
    }
}
```

In `package.json`, add two scripts:

```
{
  "scripts": {
    "pretest": "tsc --incremental -p tests/tsconfig.json",
    "test": "playwright test -c tests-out"
  }
}
```

The `pretest` script runs typescript on the tests. `test` will run the tests that have been generated to the `tests-out` directory. The `-c` argument configures the test runner to look for tests inside the `tests-out` directory.

Then `npm run test` will build the tests and run them.

Previous
Timeouts

Next
UI Mode

- Introduction
- tsconfig.json
  - tsconfig path mapping
  - tsconfig resolution
- Manually compile tests with TypeScript

Learn

- Getting started
- Playwright Training
- Learn Videos
- Feature Videos

Community

- Stack Overflow
- Discord
- Twitter
- LinkedIn

More

- GitHub
- YouTube
- Blog
- Ambassadors

Copyright © 2024 Microsoft