

Automated Visual Regression & Accessibility Testing

(made easy ✨)

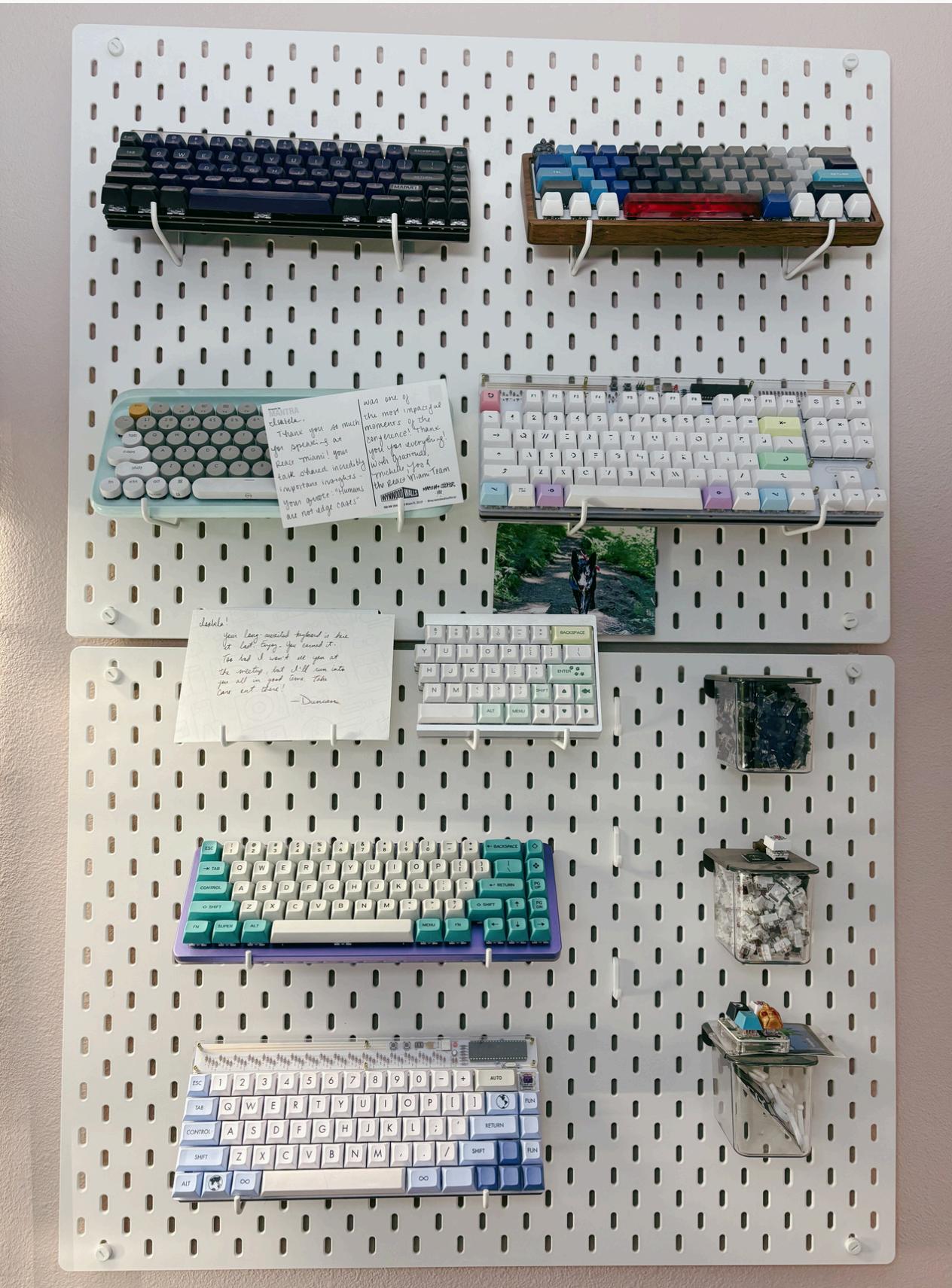
Hi, I'm Isabel

Lead software engineer @ Disney

Seattle







Automated visual regression & accessibility testing

What's in our toolbox?

Automated accessibility testing

Automated visual regression testing

Putting it all together in GitHub actions

What's in our toolbox?

Playwright

- Automation framework for end-to-end testing
- Cross browser, platform, and language
- Highly customizable
- Powerful tooling

What's in our toolbox?

Playwright

GitHub Actions

- CI/CD automation tool built into GitHub
- Event-driven execution

What's in our toolbox?

Playwright

GitHub Actions

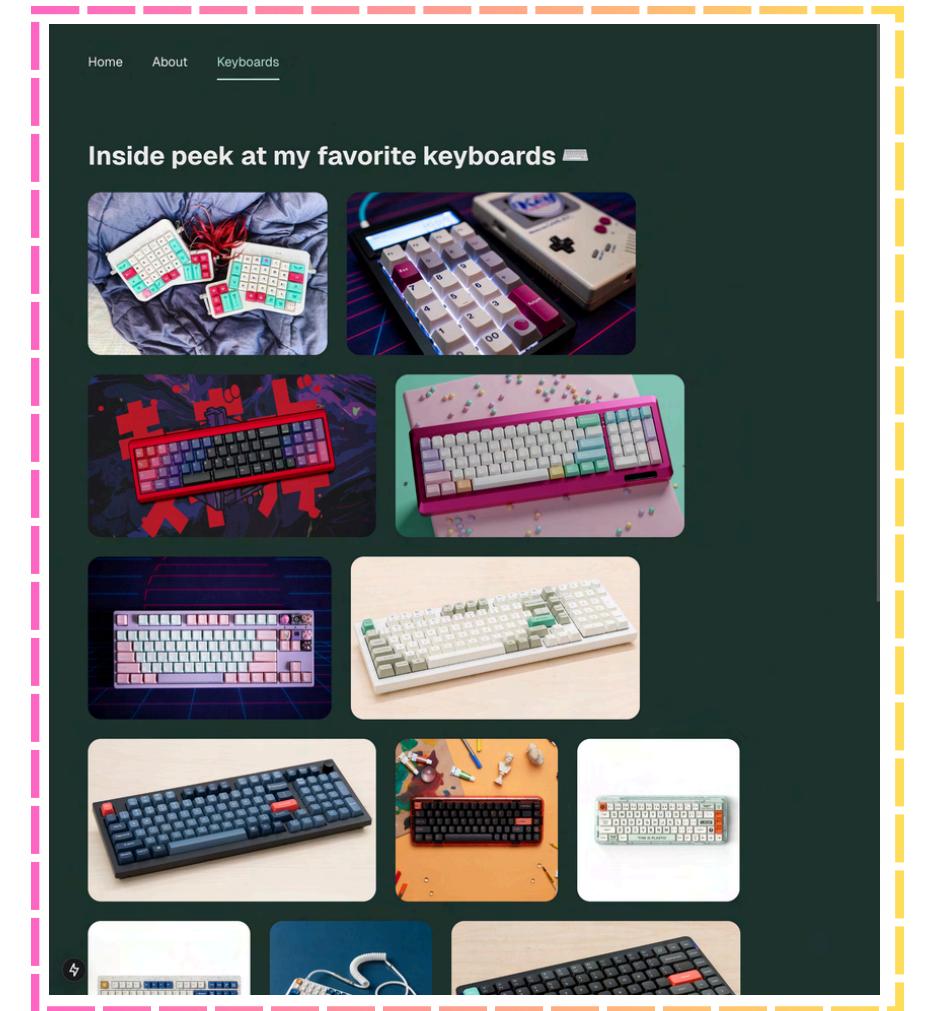
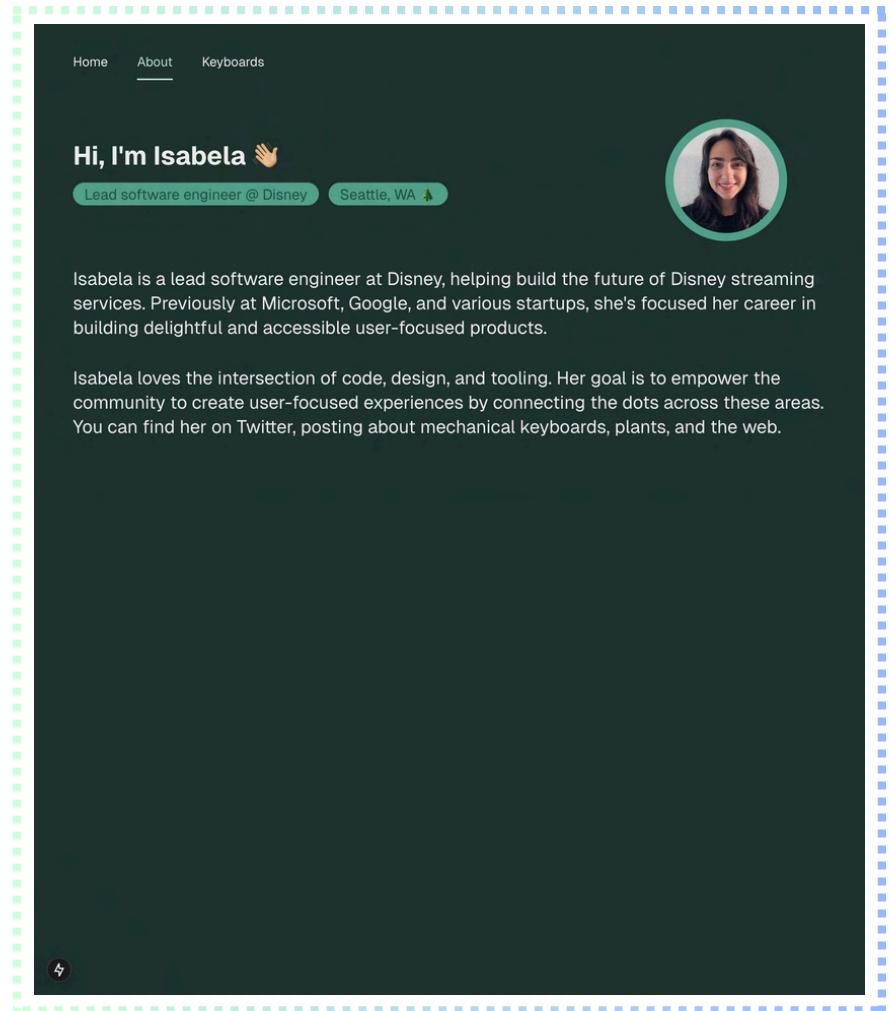
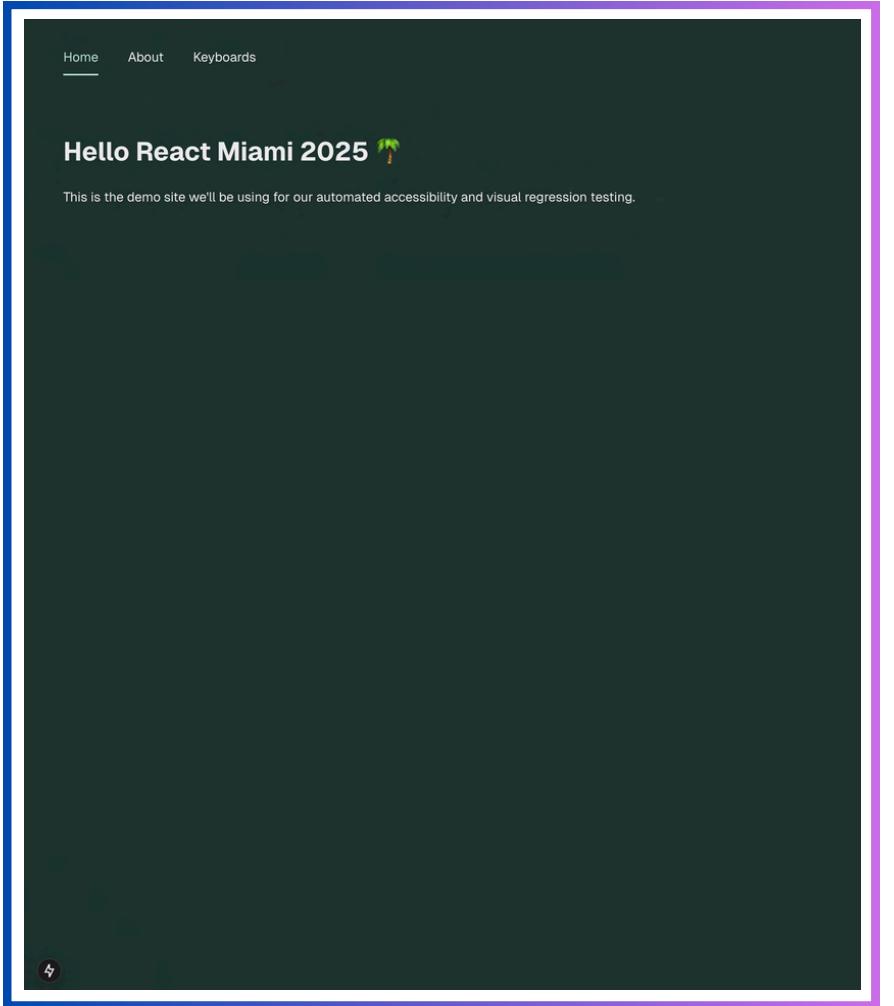
Docker

- Platform for building and running containers
- Ensures consistency across environments

Automated accessibility testing

Our demo NextJS app

```
✓ REACT-MIAMI-2025
> .github/workflows
> .next
> node_modules
> playwright-report
> public
└ src
  └ app
    └ about
      # page.module.css
      ☀ page.tsx
    └ keyboards
      # page.module.css
      ☀ page.tsx
    ★ favicon.ico
    # globals.css
    ☀ layout.tsx
    # page.module.css
    ☀ page.tsx
  > components
  > test-results
  > tests
  ⚡ .gitignore
  🛥 dockerfile
  JS eslint.config.mjs
  TS next-env.d.ts
  TS next.config.ts
  {} package-lock.json
  {} package.json
  TS playwright.config.ts
  ! pnpm-lock.yaml
  ⓘ README.md
  TS tsconfig.json
```

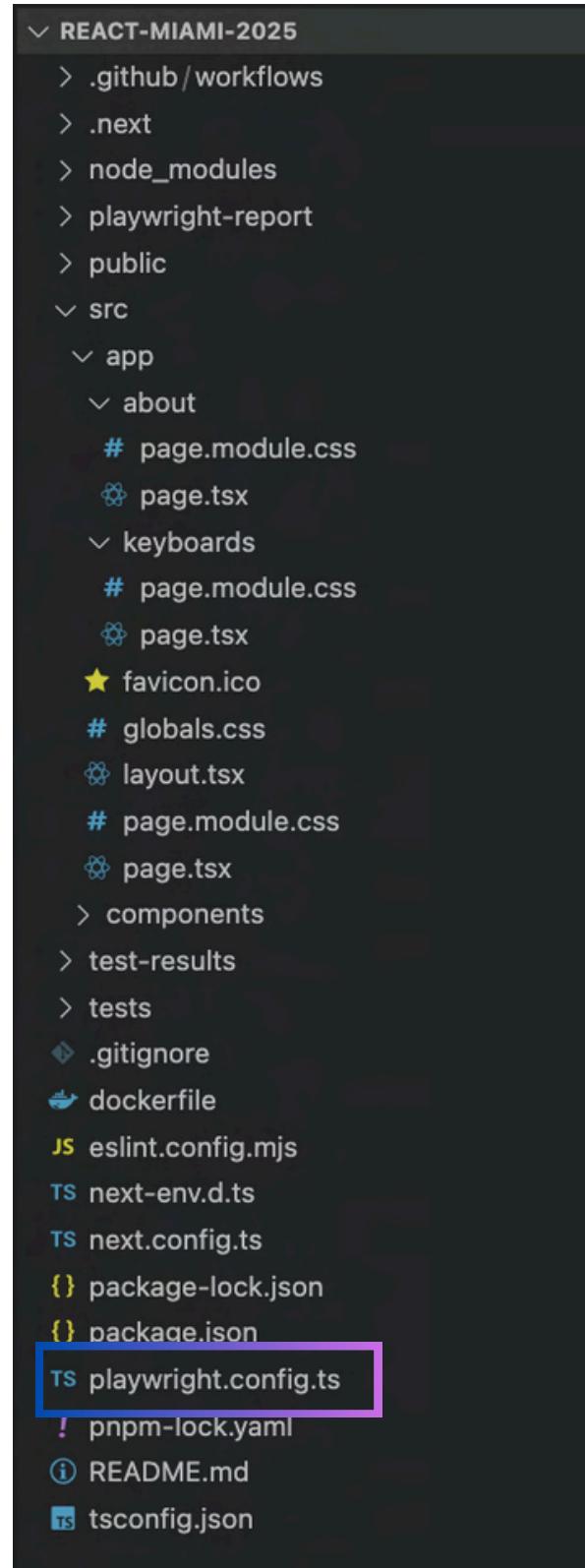


Automated accessibility testing

```
pnpm create playwright
```

- Installs Playwright and takes you through setup steps

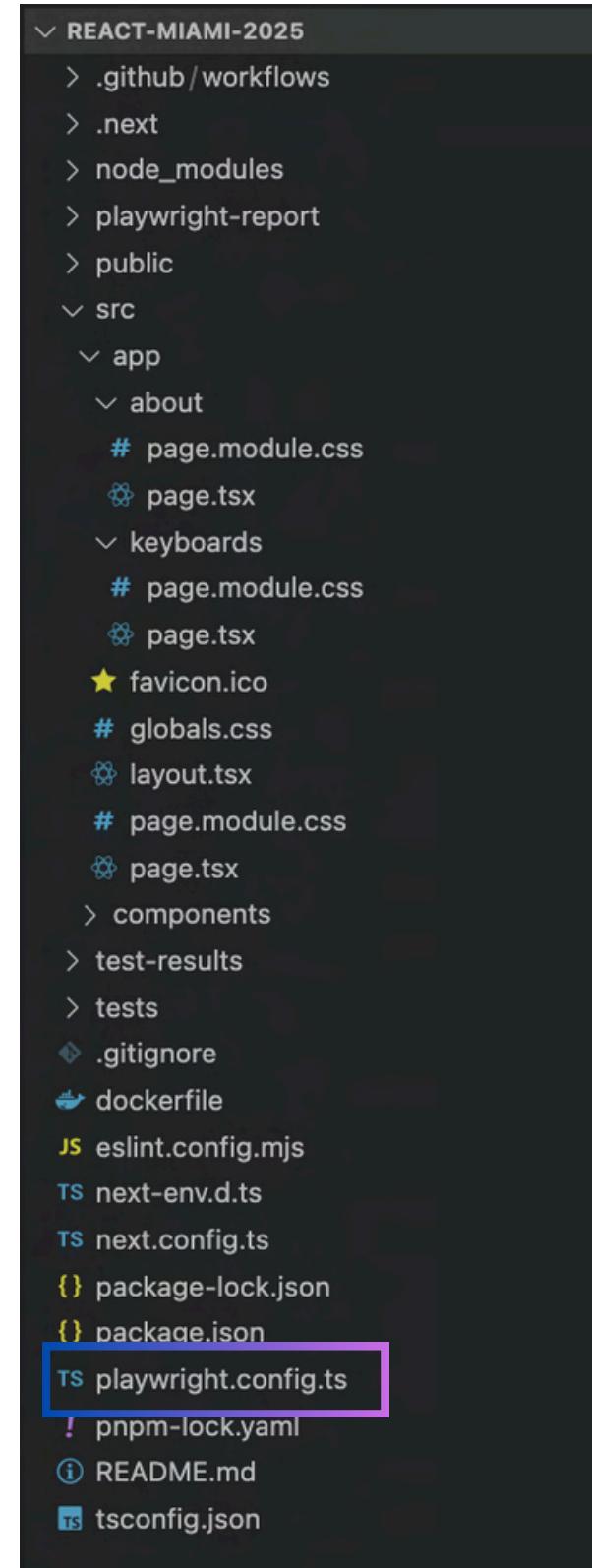
Automated accessibility testing



```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
  ],
  webServer: {
    command: process.env.CI ? 'npm start' : 'pnpm dev',
    url: 'http://localhost:3000',
    reuseExistingServer: !process.env.CI,
  },
});
```

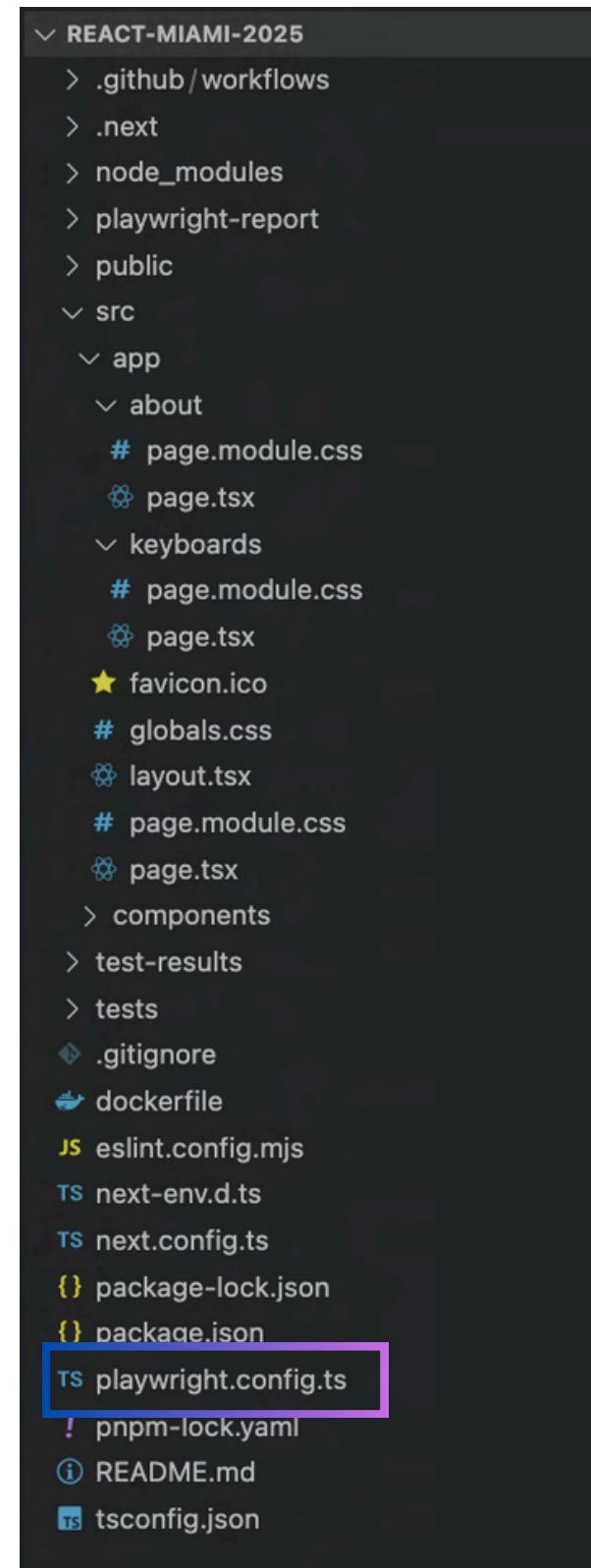
Automated accessibility testing



```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
  ],
  webServer: {
    command: process.env.CI ? 'npm start' : 'pnpm dev',
    url: 'http://localhost:3000',
    reuseExistingServer: !process.env.CI,
  },
});
```

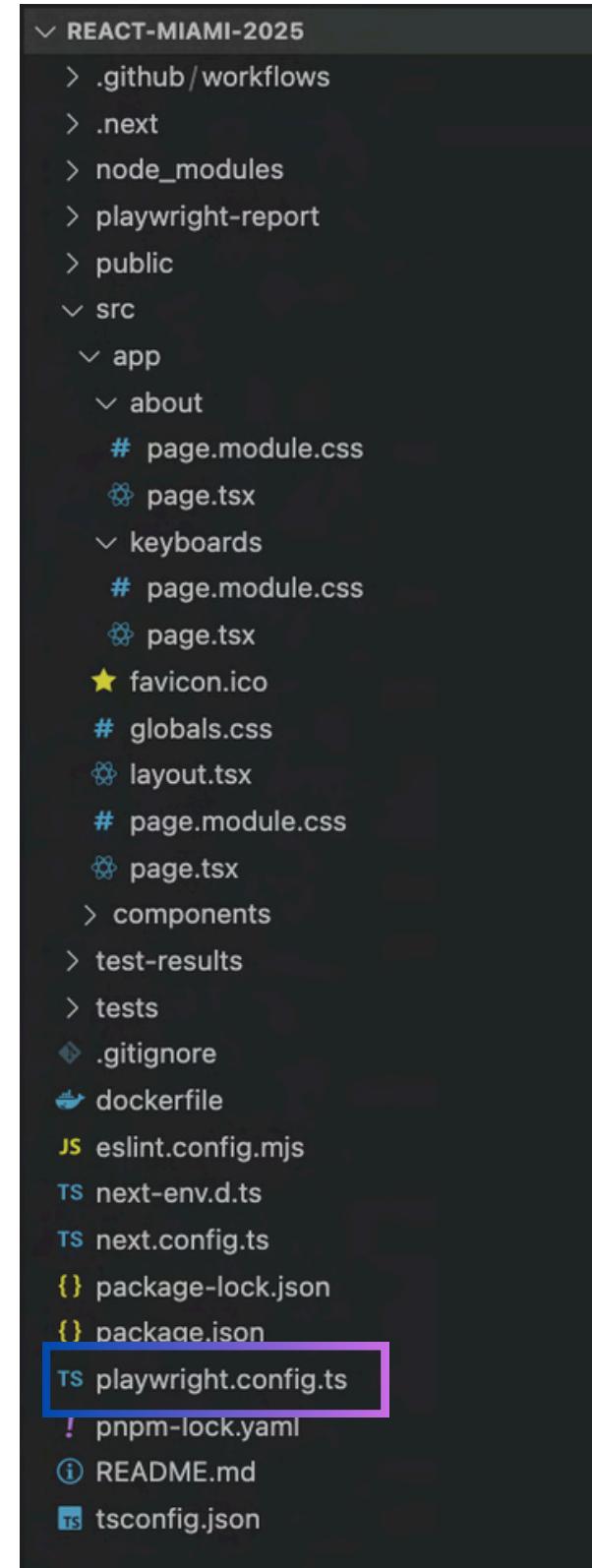
Automated accessibility testing



```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
  ],
  webServer: {
    command: process.env.CI ? 'npm start' : 'pnpm dev',
    url: 'http://localhost:3000',
    reuseExistingServer: !process.env.CI,
  },
});
```

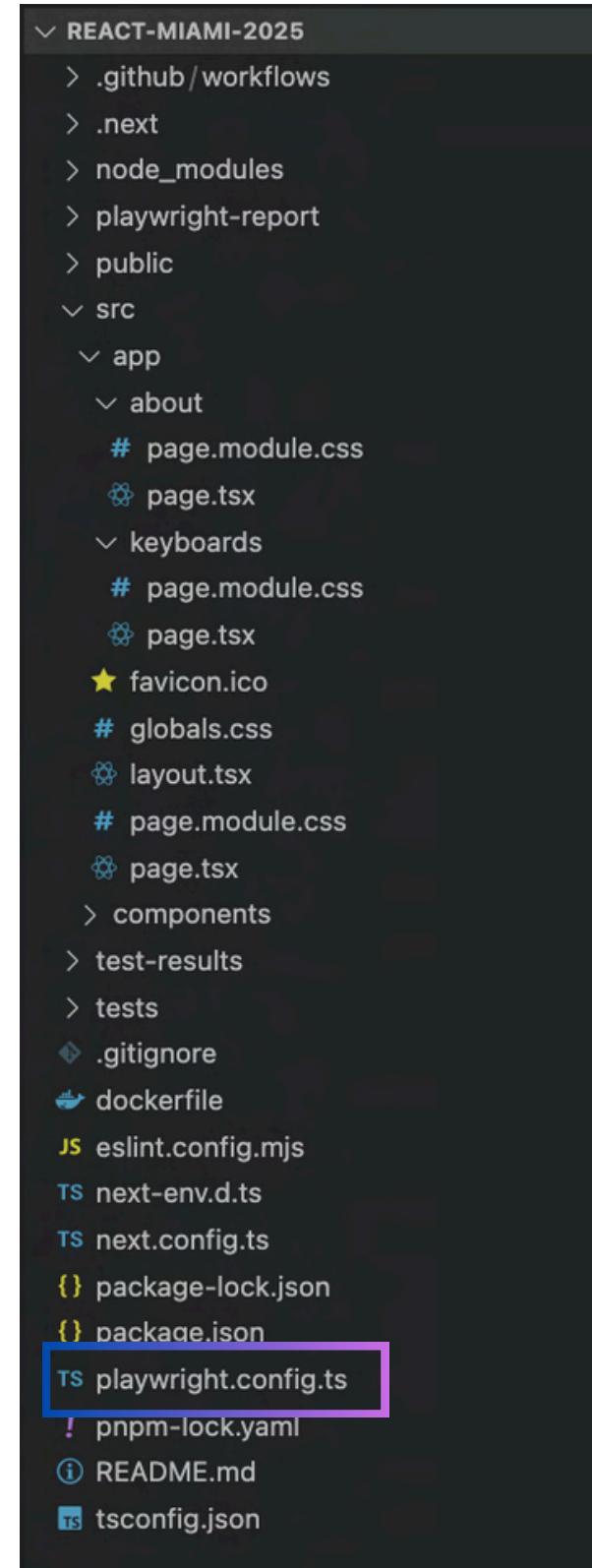
Automated accessibility testing



```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
  ],
  webServer: {
    command: process.env.CI ? 'npm start' : 'pnpm dev',
    url: 'http://localhost:3000',
    reuseExistingServer: !process.env.CI,
  },
});
```

Automated accessibility testing



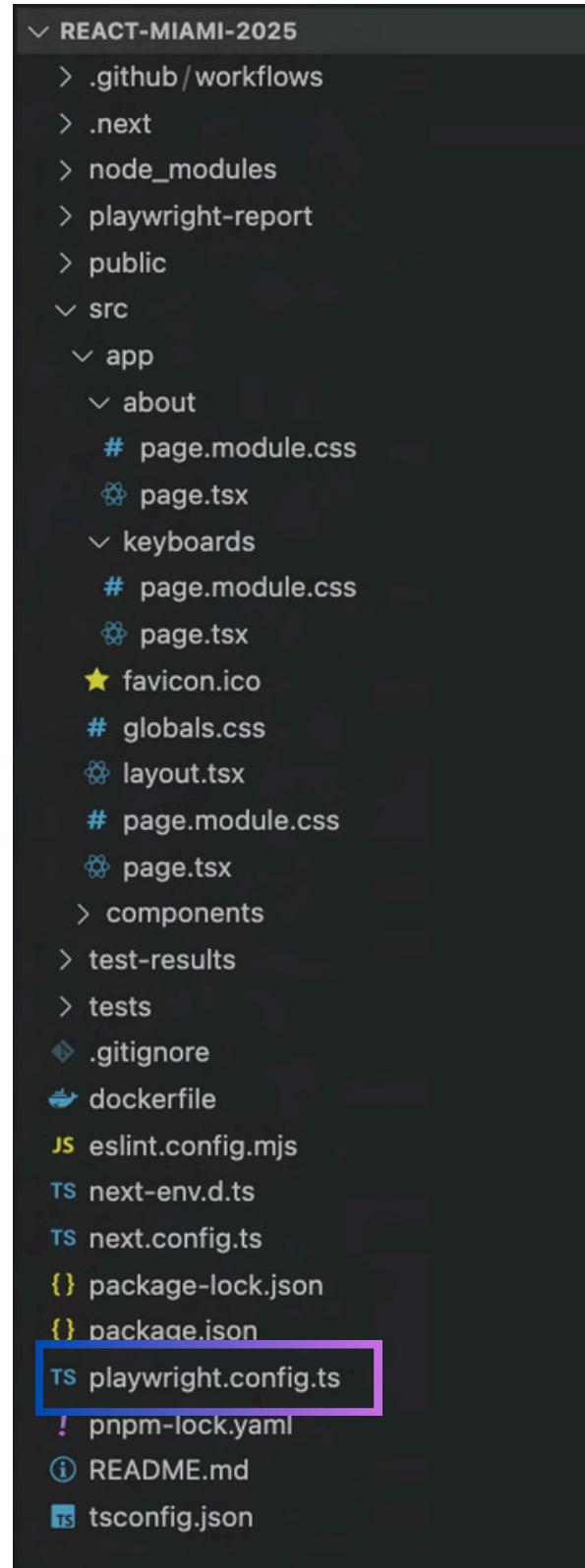
```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
});

projects: [
  {
    name: 'chromium',
    use: { ...devices['Desktop Chrome'] },
  },
],
};

webServer: {
  command: process.env.CI ? 'npm start' : 'pnpm dev',
  url: 'http://localhost:3000',
  reuseExistingServer: !process.env.CI,
},
);
```

Automated accessibility testing

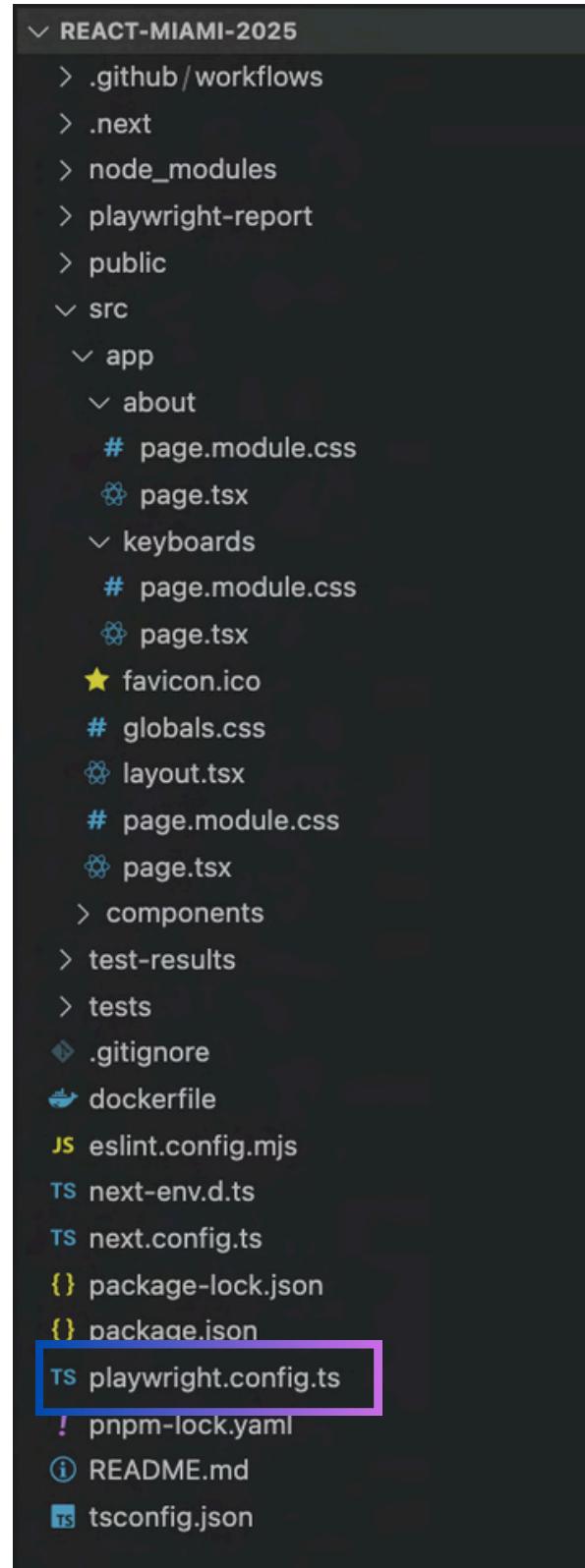


```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
  ],
};

webServer: {
  command: process.env.CI ? 'npm start' : 'pnpm dev',
  url: 'http://localhost:3000',
  reuseExistingServer: !process.env.CI,
},
});
```

Automated accessibility testing



```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['playwright-ctrf-json-reporter', { outputDir: 'playwright-report' }],
    ['list']
  ],
  snapshotPathTemplate: '{testDir}/__screenshots__/{testFilePath}/{arg}{ext}',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
  ],
  webServer: {
    command: process.env.CI ? 'npm start' : 'pnpm dev',
    url: 'http://localhost:3000',
    reuseExistingServer: !process.env.CI,
  },
});
```

Automated accessibility testing

```
REACT-MIAMI-2025
> .github
> .next
> node_modules
> playwright-report
> public
< src
  < app
    < about
      # page.module.css
      ✘ page.tsx
    < keyboards
      # page.module.css
      ✘ page.tsx
    ★ favicon.ico
    # globals.css
    ✘ layout.tsx
    # page.module.css
    ✘ page.tsx
  > components
  > test-results
< tests
  > screenshots
    TS a11y.spec.ts
  { generated-urls.json
  TS utils.spec.ts
  TS vrt.spec.ts
  .gitignore
  dockerfile
  JS eslint.config.mjs
  TS next-env.d.ts
  TS next.config.ts
  { package-lock.json
  { package.json
  TS playwright.config.ts
  ! pnpm-lock.yaml
  README.md
  TS tsconfig.json
```

```
import AxeBuilder from '@axe-core/playwright';
import { expect, test } from '@playwright/test';

/**
 * POC: Test a single page for accessibility.
 */
test('accessibility', async ({ page }) => {
  await page.goto('/keyboards');

  const accessibilityScanResults = await new AxeBuilder({ page }).analyze();

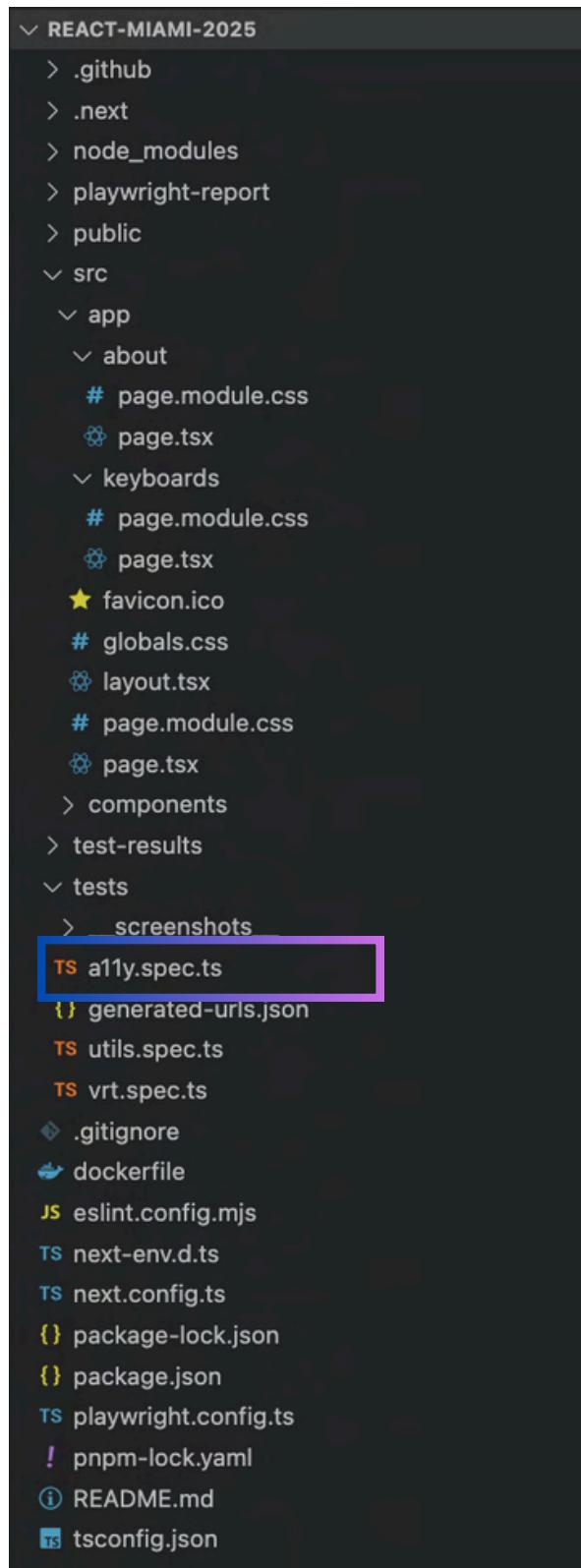
  expect(accessibilityScanResults.violations).toEqual([]);
});
```

Navigate to the route

Run the accessibility analyzer

Assert that there are no violations

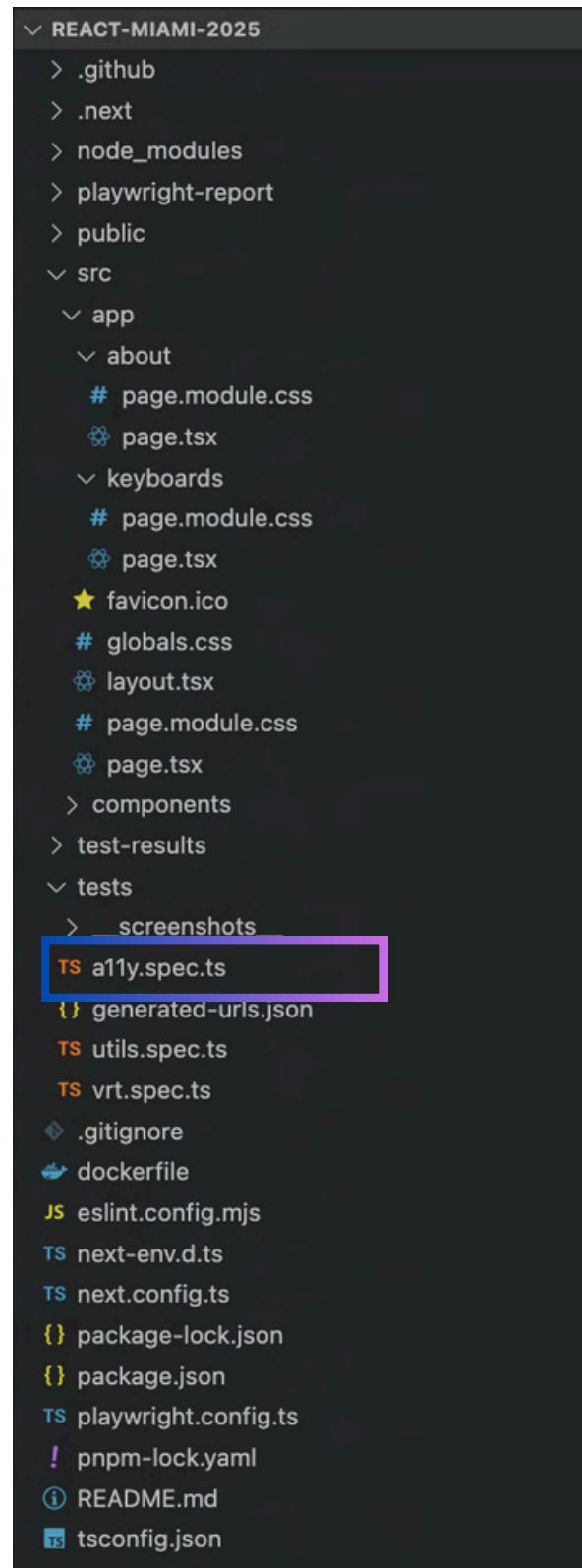
Automated accessibility testing



How do we scale this?

```
/**  
 * Approach #0: A single test.  
 * Navigate to the site and grab all the valid URLs in the <nav>.  
 * Then loop through those URLs and run a new test for each that tests the a11y.  
 *  
 * Error: Playwright Test did not expect test() to be called here.  
 */  
test('test all URLs in <nav>', async ({ page }) => {  
    await page.goto('/');  
  
    const urls = await page.$$eval('nav a', anchors => anchors.map(a => (a as HTMLAnchorElement).href));  
  
    // Iterate through each link and run a test for each  
    for (const url of urls) {  
        test(`#${url} tests`, async () => {  
            await page.goto(url);  
  
            const accessibilityScanResults = await new AxeBuilder({ page }).analyze();  
            expect(accessibilityScanResults.violations).toEqual([]);  
        });  
    }  
});
```

Automated accessibility testing

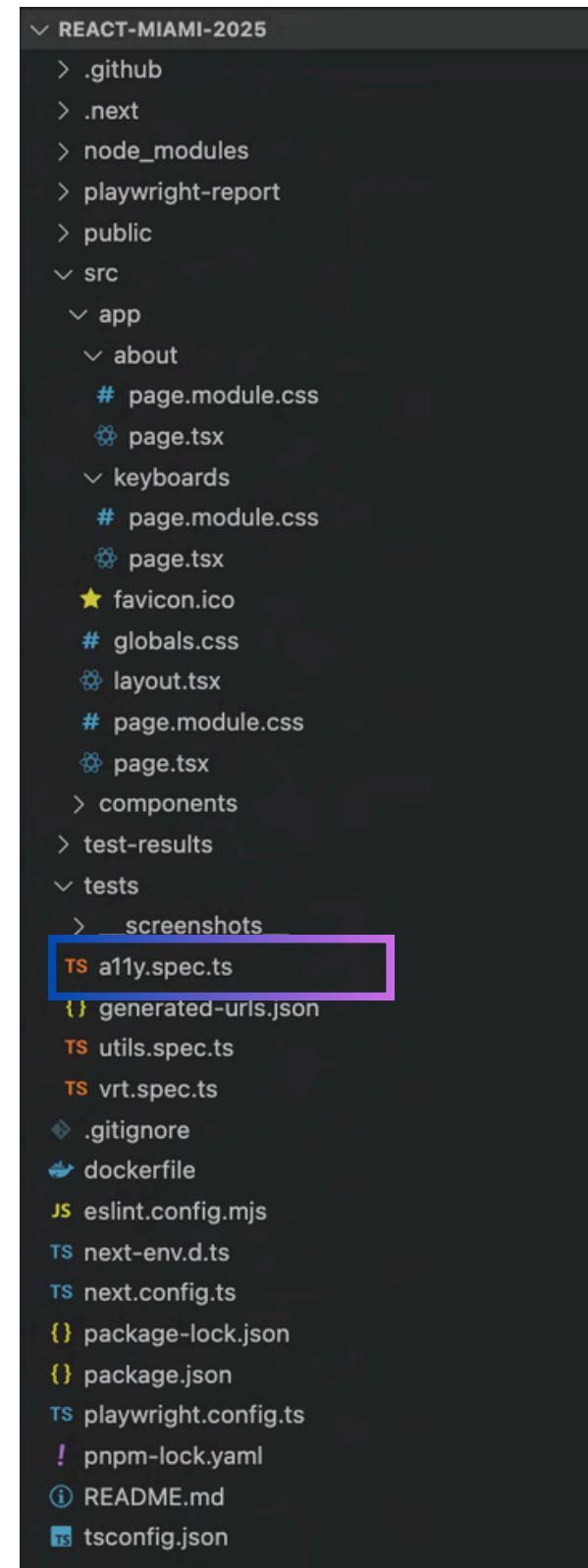


How do we scale this?

```
/**  
 * Approach #0: A single test.  
 * Navigate to the site and grab all the valid URLs in the  
 * Then loop through those URLs and run a new test for each  
 *  
 * Error: Playwright Test did not expect test() to be called  
 */  
test('test all URLs in <nav>', async ({ page }) => {  
    await page.goto('/');  
  
    const urls = await page.$$eval('nav a', anchors => anchors.map(a => (a as HTMLAnchorElement).href));  
  
    // Iterate through each link and run a test for each  
    for (const url of urls) {  
        test(`#${url} tests`, async () => {  
            await page.goto(url);  
  
            const accessibilityScanResults = await new AxeBuilder({ page }).analyze();  
            expect(accessibilityScanResults.violations).toEqual([]);  
        });  
    }  
});
```

This actually isn't possible to do because of how Playwright detects tests to be run.

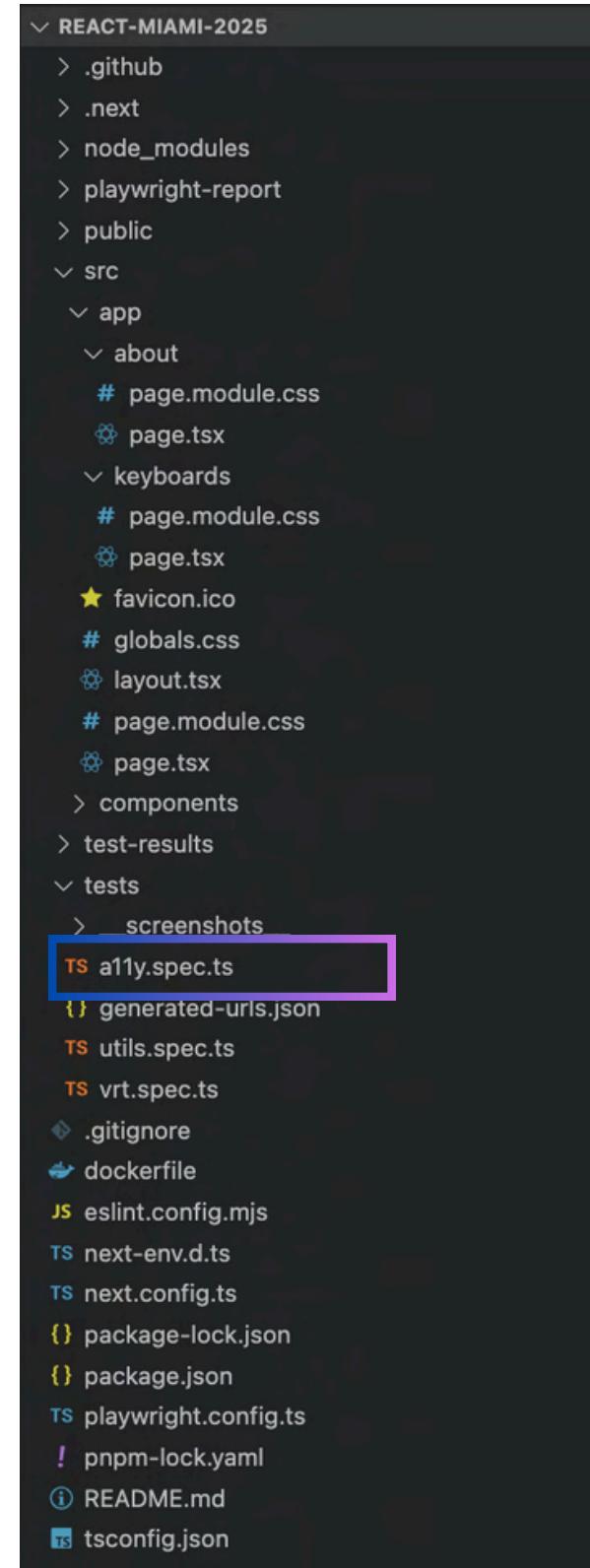
Automated accessibility testing



How do we scale this?

```
/**  
 * Approach #1: A single test.  
 * Before the test is run, we navigate to the site and grab all the valid URLs in the <nav>.  
 * Then for each URL, we run a test.step to run the accessibility test for that URL.  
 */  
let urls: string[] = [];  
test.beforeAll(async () => {  
    const browser = await chromium.launch();  
    const page = await browser.newPage();  
    await page.goto('/');  
  
    // Find all <a> tags inside the <nav> element  
    urls = await page.$$eval('nav a', anchors => anchors.map(a => (a as HTMLAnchorElement).href));  
});  
  
test('test all URLs in <nav>', async ({ page }) => {  
    // Iterate through each link and run a test for each  
    for (const url of urls) {  
        await test.step(`#${url} tests`, async () => {  
  
            await page.goto(url);  
  
            const accessibilityScanResults = await new AxeBuilder({ page }).analyze();  
            expect(accessibilityScanResults.violations).toEqual([]);  
        });  
    }  
});
```

Automated accessibility testing



How do we scale this?

```
/**  
 * Approach #1: A single test.  
 * Before the test is run, we navigate to the site and grab all the va  
 * Then for each URL, we run a test.step to run the accessibility test  
 */  
let urls: string[] = [];  
test.beforeAll(async () => {  
    const browser = await chromium.launch();  
    const page = await browser.newPage();  
    await page.goto('/');  
  
    // Find all <a> tags inside the <nav> element  
    urls = await page.$$eval('nav a', anchors => anchors.map(a => (a as  
});  
  
test('test all URLs in <nav>', async ({ page }) => {  
    // Iterate through each link and run a test for each  
    for (const url of urls) {  
        await test.step(`#${url} tests`, async () => {  
  
            await page.goto(url);  
  
            const accessibilityScanResults = await new AxeBuilder({ page })  
            expect(accessibilityScanResults.violations).toEqual([]);  
        });  
    }  
});
```

This is okay but...

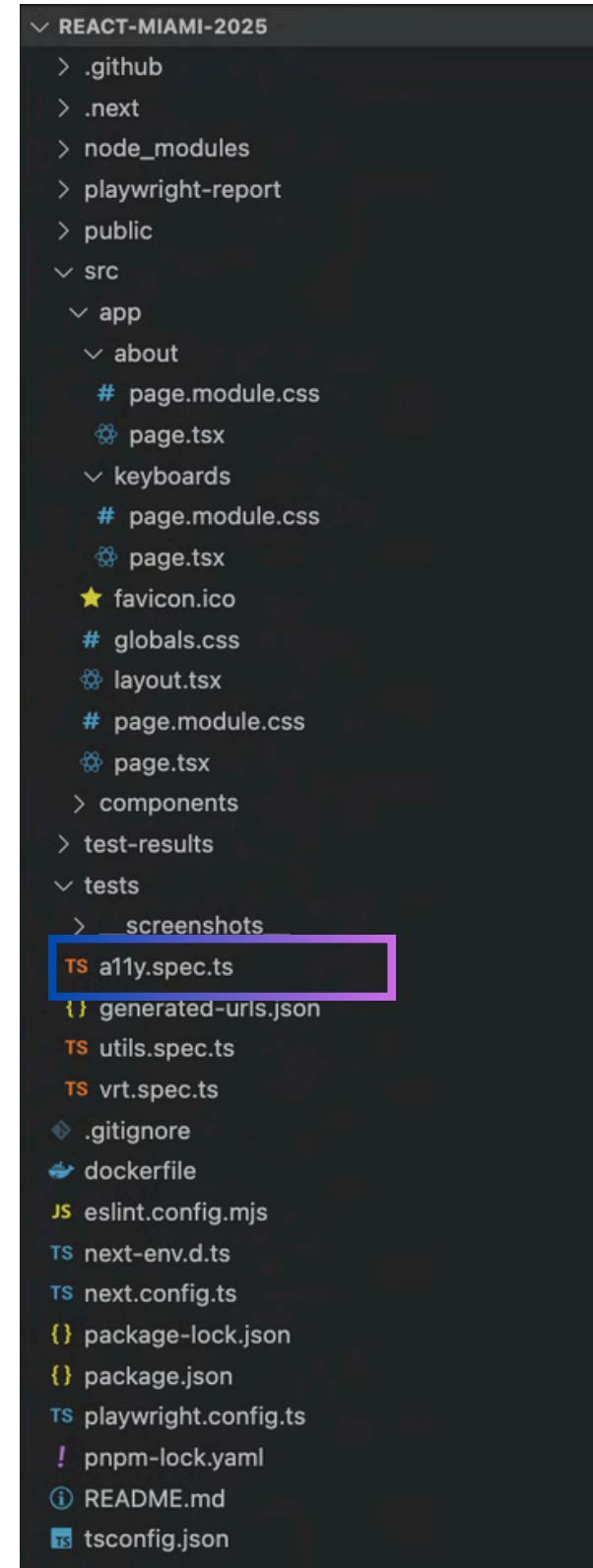
Pros:

- Run a single, self-contained test.

Cons:

- Test steps can't be run in parallel.
- Test won't display test.step() name, making debugging and validation difficult.

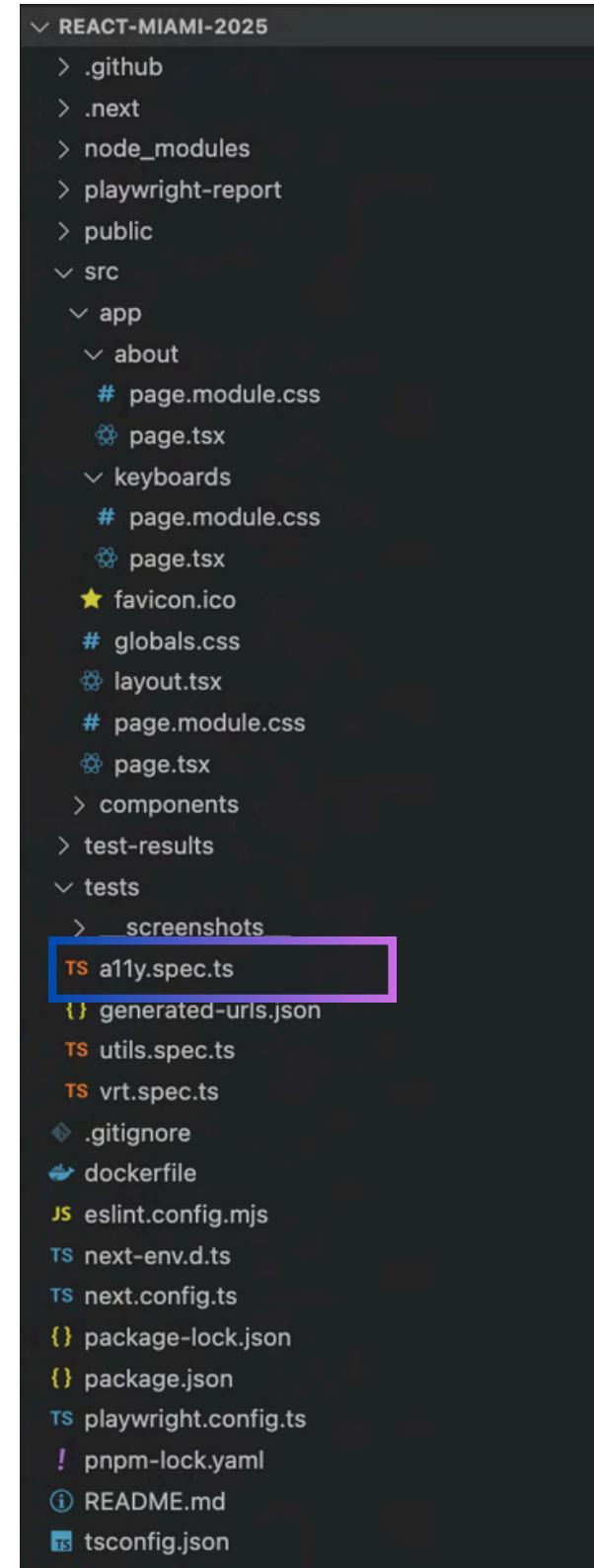
Automated accessibility testing



How do we scale this?

```
/**  
 * Approach #2: Multiple tests with pre-generation.  
 * Before we run the a1y tests, we run a separate test that generates all the URLs we want to test and  
 * saves them to a json file.  
 */  
const filePath = path.join(__dirname, 'generated-urls.json');  
const json = fs.readFileSync(filePath, {encoding: "utf-8"});  
const urls = JSON.parse(json);  
  
for (const url of urls) {  
  test(`Accessibility test: ${url}`, async ({ page }) => {  
    await page.goto(url);  
  
    const accessibilityScanResults = await new AxeBuilder({ page }).analyze();  
    expect(accessibilityScanResults.violations).toEqual([]);  
  });  
}
```

Automated accessibility testing



How do we scale this?

```
/**  
 * Approach #2: Multiple tests with pre-generation.  
 * Before we run the a1y tests, we run a separate test that genera  
 saves them to a json file.  
 */  
const filePath = path.join(__dirname, 'generated-urls.json');  
const json = fs.readFileSync(filePath, {encoding: "utf-8"});  
const urls = JSON.parse(json);  
  
for (const url of urls) {  
  test(`Accessibility test: ${url}`, async ({ page }) => {  
    await page.goto(url);  
  
    const accessibilityScanResults = await new AxeBuilder({ page })  
      .expect(accessibilityScanResults.violations).toEqual([]);  
  });  
}
```

This works pretty well

Pros:

- Tests run in parallel.
- Test names are displayed.
- Test reports are easy to read.

Cons:

- Need to run pre-generation test before running our actual tests.

Automated accessibility testing

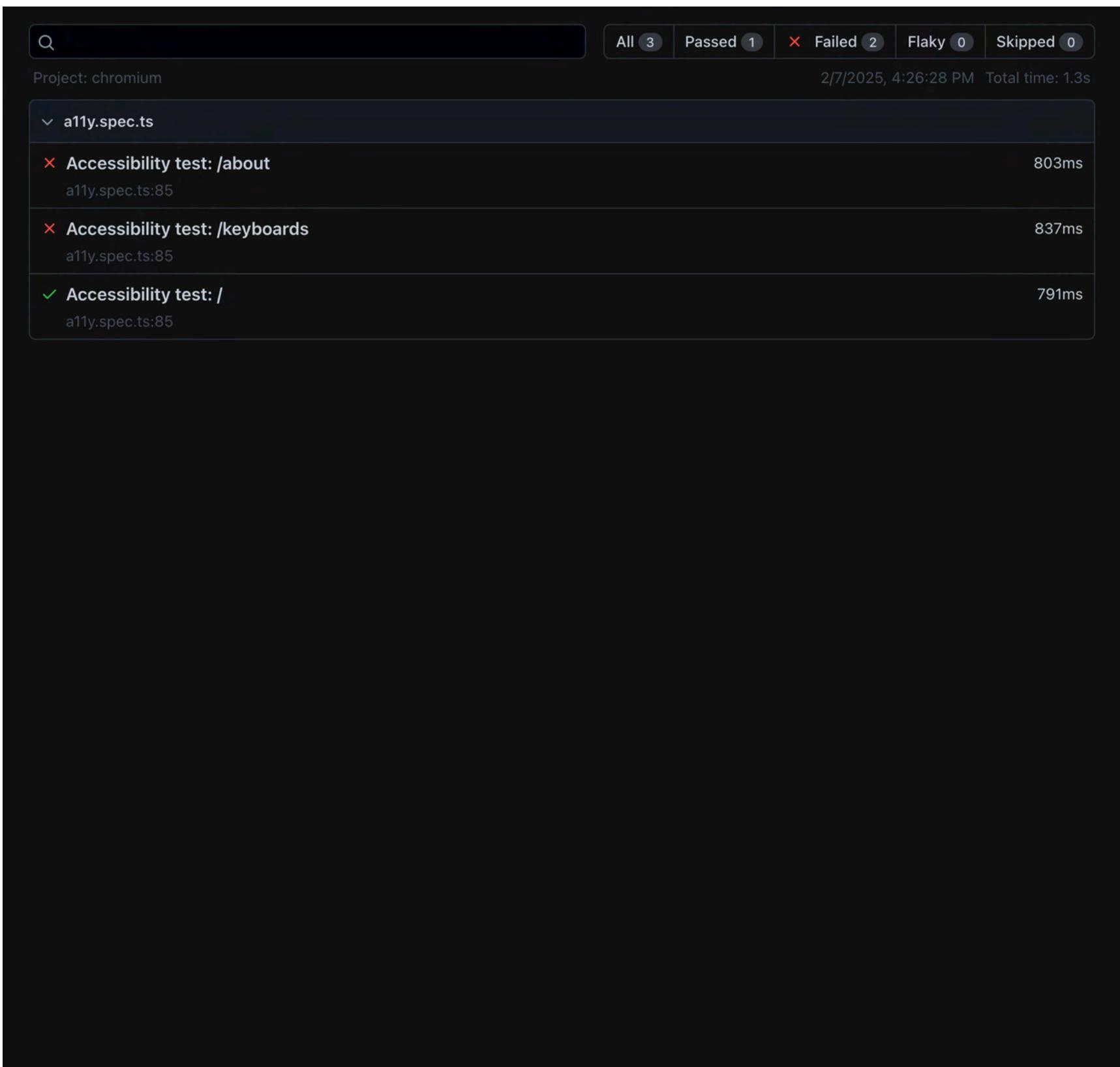
```
REACT-MIAMI-2025
├── .github
├── .next
├── node_modules
├── playwright-report
└── public
  └── src
    ├── app
    │   └── about
    │       ├── # page.module.css
    │       └── page.tsx
    └── keyboards
        ├── # page.module.css
        └── page.tsx
    └── favicon.ico
    └── # globals.css
    └── layout.tsx
    └── # page.module.css
    └── page.tsx
    └── components
    └── test-results
    └── tests
        └── __screenshots__
            └── a11y.spec.ts
            └── generated-urls.json
            └── utils.spec.ts
            └── vrt.spec.ts
    └── .gitignore
    └── dockerfile
    └── eslint.config.mjs
    └── next-env.d.ts
    └── next.config.ts
    └── package-lock.json
    └── package.json
    └── playwright.config.ts
    └── ! pnpm-lock.yaml
    └── README.md
    └── tsconfig.json
```

```
{
  "name": "react-miami-2025",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "playwright:generate-urls": "playwright test utils",
    "playwright:a11y:local": "pnpm playwright:generate-urls && playwright test a11y && pnpm open:playwright-report",
    "open:playwright-report": "open ./playwright-report/index.html",
  },
  ...
}
```

Generate test URLs

Run generation command,
run a11y tests, open report

Automated accessibility testing



A screenshot of a dark-themed user interface for automated accessibility testing. At the top, there is a search bar with a magnifying glass icon and a status bar showing "All 3", "Passed 1", "Failed 2", "Flaky 0", and "Skipped 0". Below this, the project name "chromium" is displayed along with the date and time "2/7/2025, 4:26:28 PM" and total execution time "Total time: 1.3s". The main area shows a list of test cases under the category "a11y.spec.ts". The tests are listed as follows:

- Accessibility test: /about** (Failed) - a11y.spec.ts:85, 803ms
- Accessibility test: /keyboards** (Failed) - a11y.spec.ts:85, 837ms
- Accessibility test: /** (Passed) - a11y.spec.ts:85, 791ms

Automated accessibility testing

Q All 3 Passed 1 Failed 2 Flaky 0 Skipped 0 « previous next »

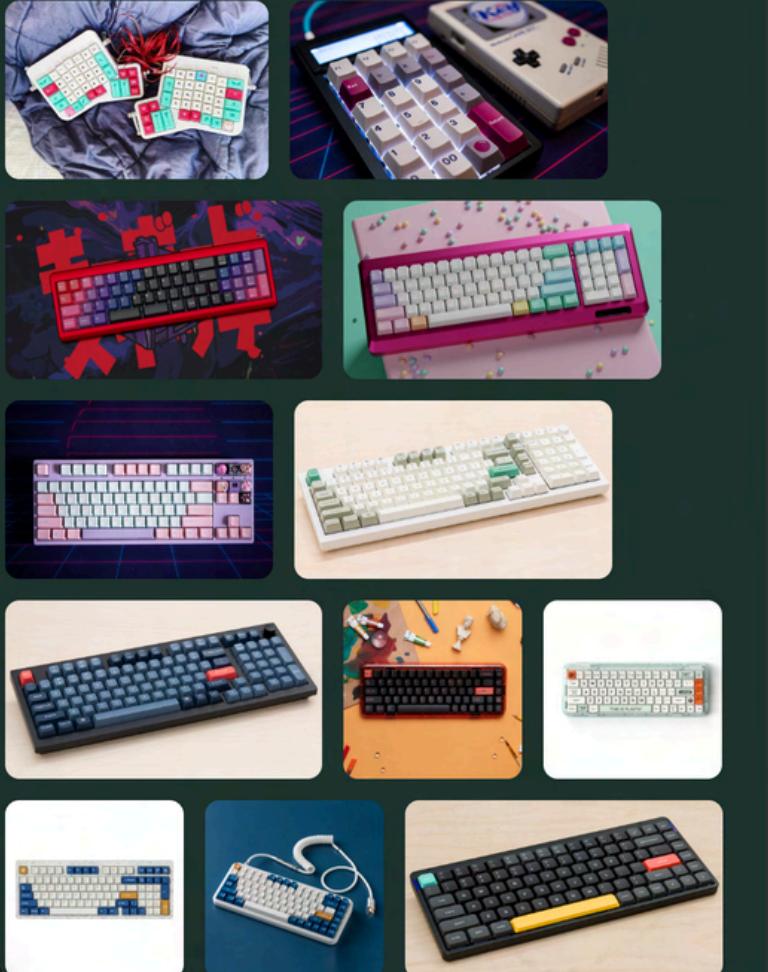
Accessibility test: /keyboards

a11y.spec.ts:85 837ms

```
: expect(received).toEqual(expected) // deep equality
expected = 1
eived + 1104
ay []
ay [
hiect [
  "description": "Ensure <img> elements have alternative text or a role of none or presentation",
  "help": "Images must have alternative text",
  "helpUrl": "https://dequeuniversity.com/rules/axe/4.10/image-alt?application=playwright",
  "id": "image-alt",
  "impact": "critical",
  "nodes": Array [
    Object {
      "all": Array [],
      "any": Array [
        Object {
          "data": null,
          "id": "has-alt",
          "impact": "critical",
          "message": "Element does not have an alt attribute",
          "relatedNodes": Array [],
        },
        Object {
          "data": null,
          "id": "aria-label",
          "impact": "critical",
          "message": "aria-label attribute does not exist or is empty",
          "relatedNodes": Array [],
        },
        Object {
          "data": null,
          "id": "aria-labelledby",
          "impact": "critical",
          "message": "aria-labelledby attribute does not exist, references elements that do not exist or r
        },
      ],
    },
  ],
]
```

Home About Keyboards

Inside peek at my favorite keyboards



```
...<!DOCTYPE html> == $0
<html lang="en"> (scroll)
  > <head>@@</head>
  > <body class="__variable_4d318d __variable_ea5f4b">
    > <nav class="navigation_nav_WNYUT">@@</nav>
    > <main>
      > <h1>Inside peek at my favorite keyboards @@</h1>
      > <div class="page_keyboardsContainer_0DIGO"> (flex)
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
      > </div>
    > </main>
    > <script src="/_next/static/chunks/webpack.js?v=1741728444416" async></script>
    > <script>(self._next_f=self._next_f||[]).push([0])</script>
    > <script>@@</script>
    > <script>@@</script>
    > <script>@@</script>
    > <script>@@</script>
  </body>
```

Automated accessibility testing

Home About Keyboards

Inside peek at my favorite keyboards

The screenshot shows a web browser window with the title "Inside peek at my favorite keyboards". Below the title is a grid of 12 images, each showing a different custom or artisanal keyboard. The keyboards are arranged in three rows of four. The first row includes a white keyboard with red and green keys, a white keyboard with pink and purple keys, a red keyboard with blue and purple keys, and a pink keyboard with white and yellow keys. The second row includes a white keyboard with purple and pink keys, a white keyboard with grey and white keys, a black keyboard with red and blue keys, and a black keyboard with red and orange keys. The third row includes a white keyboard with blue and yellow keys, a white keyboard with blue and white keys, a black keyboard with red and yellow keys, and a black keyboard with red and yellow keys.

Elements Console Sources Network Performance Memory

```
...<!DOCTYPE html> == $0
<html lang="en"> (scroll)
  > <head>(...)</head>
  > <body class="__variable_4d318d __variable_ea5f4b">
    > <nav class="navigation_nav_WNYUt">(...)</nav>
    > <main>
      > <h1>Inside peek at my favorite keyboards </h1>
      > <div class="page_keyboardsContainer_0DIGO"> (flex)
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
        > 
      > </div>
    > </main>
  > </body>
<!doctype>
```

Accessibility test: /keyboards

All 3 Passed 3 Failed 0 Flaky 0 Skipped 0 « previous

chromium

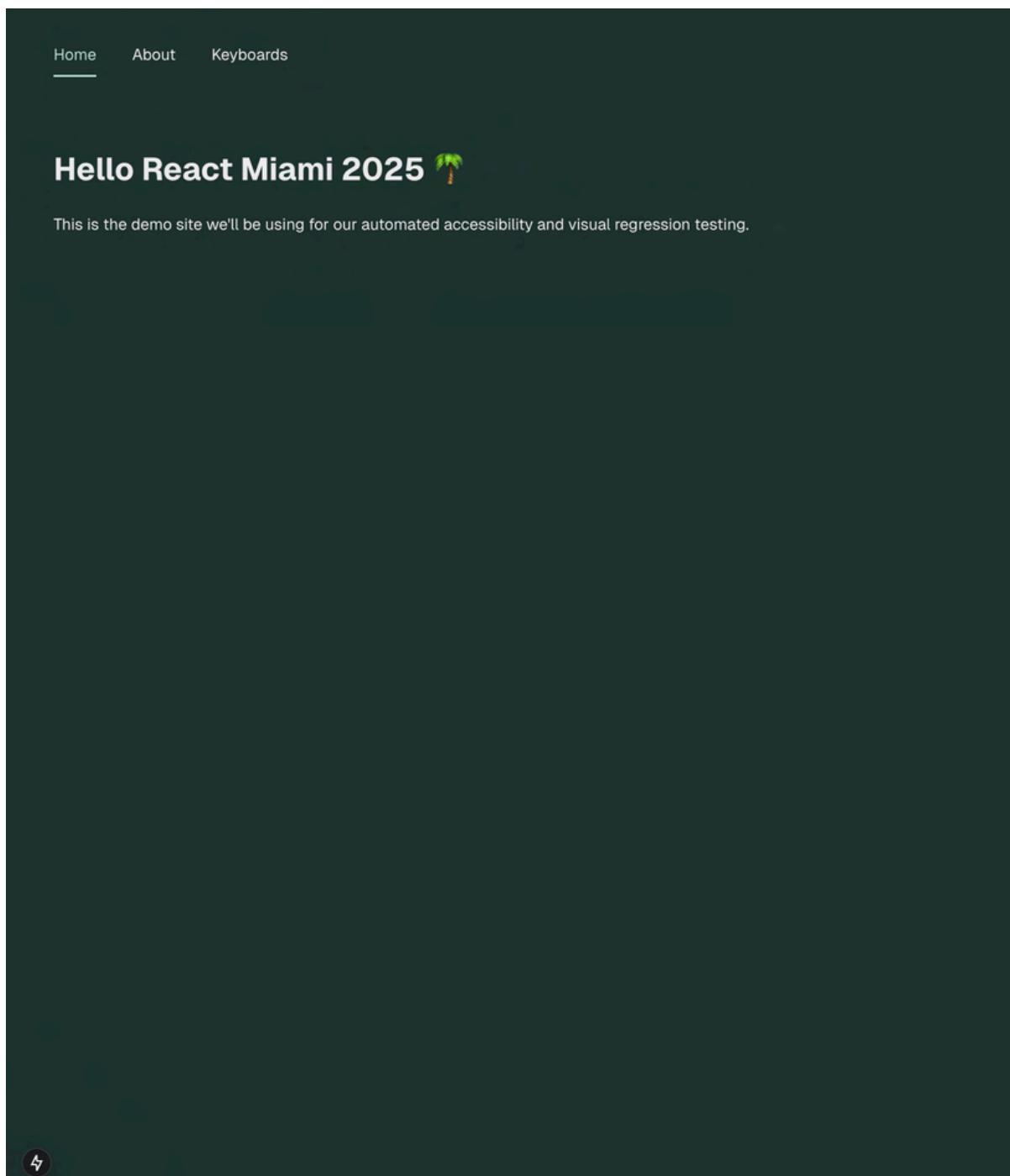
✓ Run

Test Steps

Step	Status	Time
> ✓ Before Hooks	Passed	316ms
> ✓ page.goto(/keyboards) — a11y.spec.ts:86	Passed	463ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:236	Passed	81ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:237	Passed	2ms
> ✓ frame.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:315	Passed	3ms
> ✓ frame.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:318	Passed	51ms
> ✓ browserContext.newPage — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:39	Passed	39ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:353	Passed	73ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:372	Passed	2ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:354	Passed	1ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:359	Passed	7ms
> ✓ page.evaluate — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:365	Passed	26ms
> ✓ page.close — ../node_modules/pnpm/@axe-core+playwright@4.10.1/node_modules/@axe-core/playwright/dist/index.js:368	Passed	3ms
> ✓ expect.toEqual — a11y.spec.ts:89	Passed	1ms
> ✓ After Hooks	Passed	8ms

Automated visual regression testing

Our demo NextJS app



Home About Keyboards

Hi, I'm Isabela 🙌

Lead software engineer @ Disney Seattle, WA

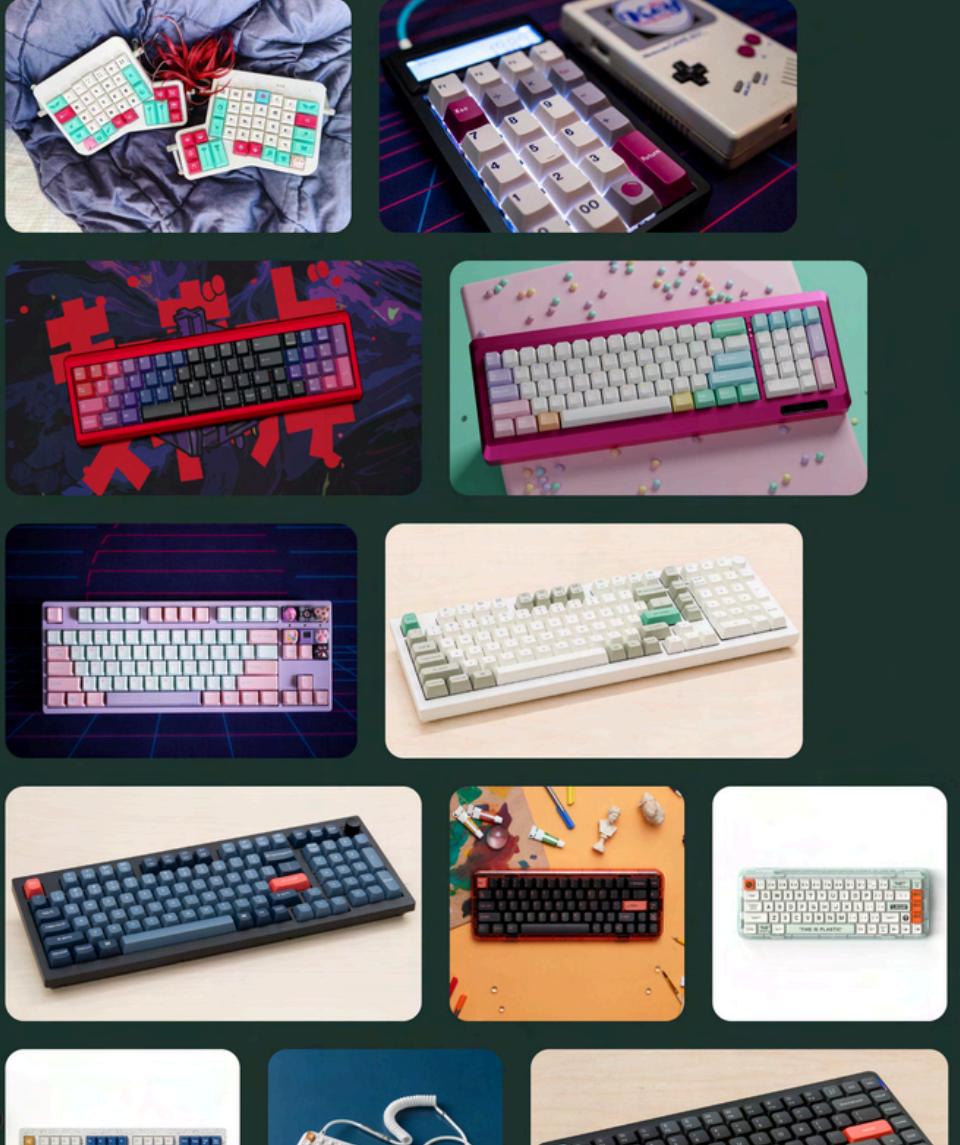


Isabela is a lead software engineer at Disney, helping build the future of Disney streaming services. Previously at Microsoft, Google, and various startups, she's focused her career in building delightful and accessible user-focused products.

Isabela loves the intersection of code, design, and tooling. Her goal is to empower the community to create user-focused experiences by connecting the dots across these areas. You can find her on Twitter, posting about mechanical keyboards, plants, and the web.

Home About Keyboards

Inside peek at my favorite keyboards



Automated visual regression testing

```
REACT-MIAMI-2025
> .github
> .next
> node_modules
> playwright-report
> public
< src
  < app
    < about
      # page.module.css
      ☀ page.tsx
    < keyboards
      # page.module.css
      ☀ page.tsx
    ★ favicon.ico
    # globals.css
    ☀ layout.tsx
    # page.module.css
    ☀ page.tsx
  > components
  > test-results
< tests
  > __screenshots__
    TS a1y.spec.ts
    {} generated-urls.json
    TS utils.spec.ts
    TS vrt.spec.ts
    .gitignore
    Dockerfile
    JS eslint.config.mjs
    TS next-env.d.ts
    TS next.config.ts
    {} package-lock.json
    {} package.json
    TS playwright.config.ts
    ! pnpm-lock.yaml
    README.md
    TS tsconfig.json
```

```
import { expect, test } from '@playwright/test';
import fs from 'fs';
import path from 'path';

/**
 * Use the same approach as we do for accessibility testing.
 */

const filePath = path.join(__dirname, 'generated-urls.json');
const json = fs.readFileSync(filePath, {encoding: "utf-8"} );
const urls = JSON.parse(json);

for (const url of urls) {
  test(`Visual regression test: ${url}`, async ({ page }) => {
    await page.goto(url);

    const screenshotName = url.replace(/[^a-z0-9]/gi, '-').toLowerCase();

    await expect(page).toHaveScreenshot(`${screenshotName}.png`);
  });
}
```

Navigate to the route

Clean up the screenshot file name

Assert that the new screenshot
matches the baseline

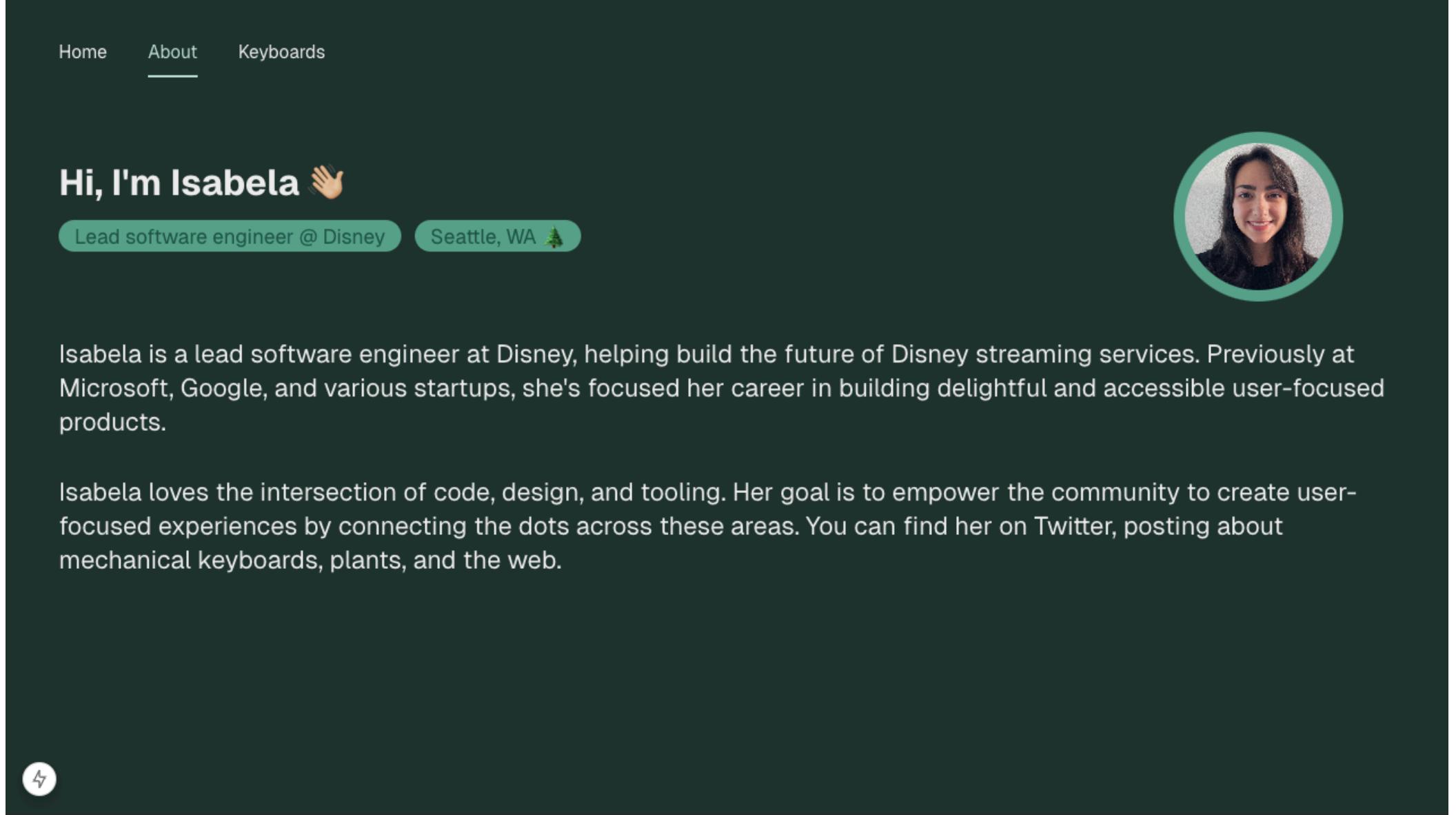
Automated visual regression testing

```
REACT-MIAMI-2025
├── .github
├── .next
├── node_modules
├── playwright-report
└── public
  └── src
    ├── app
    │   └── about
    │       ├── # page.module.css
    │       └── ☀️ page.tsx
    ├── keyboards
    │   ├── # page.module.css
    │   └── ☀️ page.tsx
    └── ★ favicon.ico
    └── # globals.css
    └── ☀️ layout.tsx
    └── # page.module.css
    └── ☀️ page.tsx
    └── components
    └── test-results
    └── tests
        └── __screenshots__
            └── TS a11y.spec.ts
        └── {} generated-urls.json
        └── TS utils.spec.ts
        └── TS vrt.spec.ts
    └── .gitignore
    └── 🛠 dockerfile
    └── JS eslint.config.mjs
    └── TS next-env.d.ts
    └── TS next.config.ts
    └── {} package-lock.json
    └── {} package.json
    └── TS playwright.config.ts
    └── ! pnpm-lock.yaml
    └── ⓘ README.md
    └── TS tsconfig.json
```

```
{
  "name": "react-miami-2025",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "playwright:generate-urls": "playwright test utils",
    "playwright:a11y:local": "pnpm playwright:generate-urls && playwright test a11y && pnpm open:playwright-report",
    "playwright:vrt:local": "pnpm playwright:generate-urls && playwright test vrt && pnpm open:playwright-report",
  },
  ...
}
```

Run generation command,
run visual regression tests,
open report

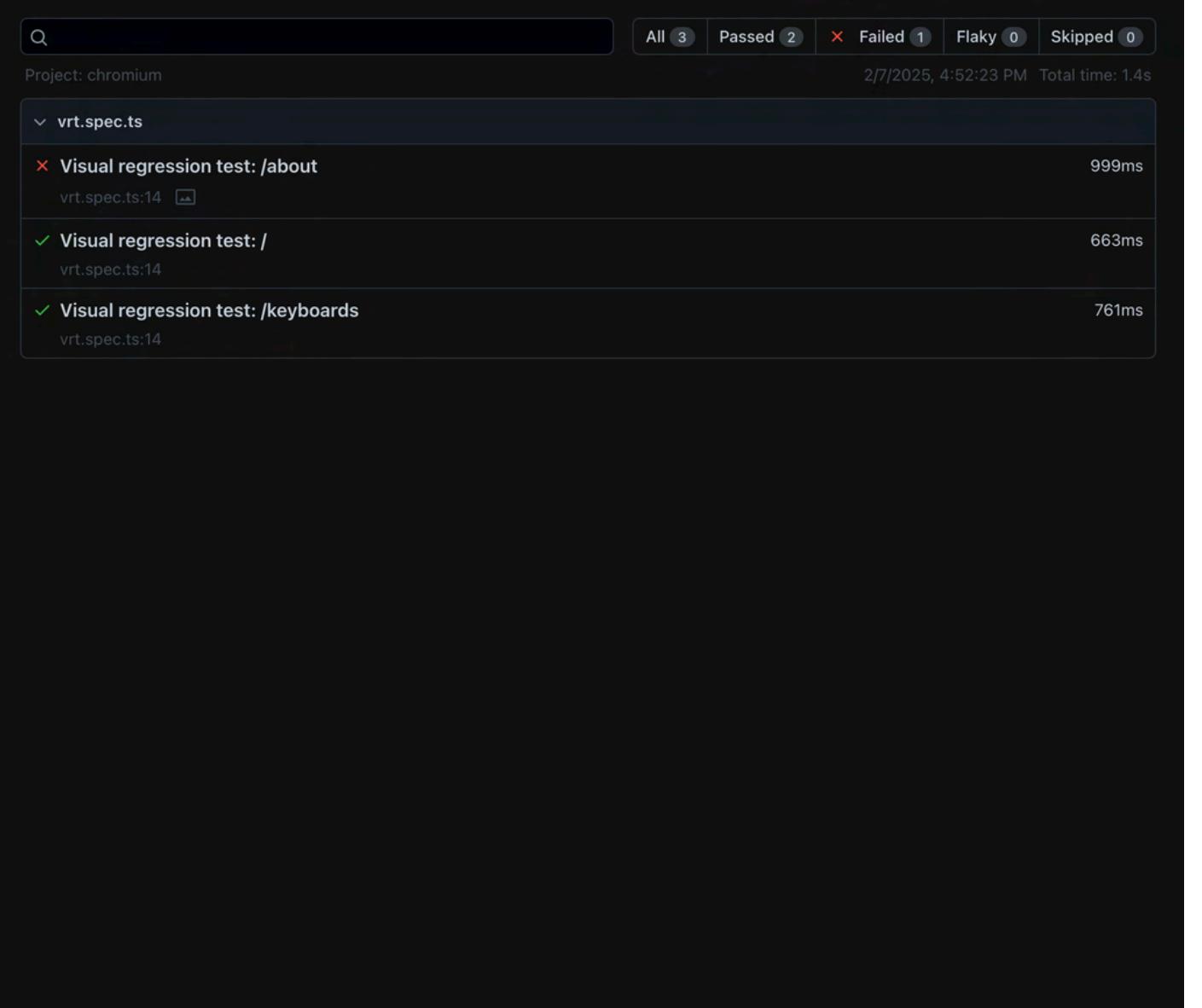
Automated visual regression testing



The screenshot shows a dark-themed website for Isabelo. At the top, there's a navigation bar with links for "Home", "About", and "Keyboards". The "About" link is underlined, indicating it's the current page. Below the navigation, a large heading says "Hi, I'm Isabelo" followed by a waving hand emoji. Underneath this, there are two teal-colored buttons: one for "Lead software engineer @ Disney" and another for "Seattle, WA" which includes a small Christmas tree icon. To the right of the text area is a circular profile picture of a woman with long dark hair, smiling. At the bottom left of the page is a small circular icon containing a lightning bolt symbol.

Isabelo is a lead software engineer at Disney, helping build the future of Disney streaming services. Previously at Microsoft, Google, and various startups, she's focused her career in building delightful and accessible user-focused products.

Isabelo loves the intersection of code, design, and tooling. Her goal is to empower the community to create user-focused experiences by connecting the dots across these areas. You can find her on Twitter, posting about mechanical keyboards, plants, and the web.



The screenshot shows a visual regression test results interface. At the top, there's a search bar and a status summary: "All 3", "Passed 2", "Failed 1", "Flaky 0", and "Skipped 0". Below this, the date and time are listed as "2/7/2025, 4:52:23 PM Total time: 1.4s". The main area displays a list of test results under the "vrt.spec.ts" project. The list includes:

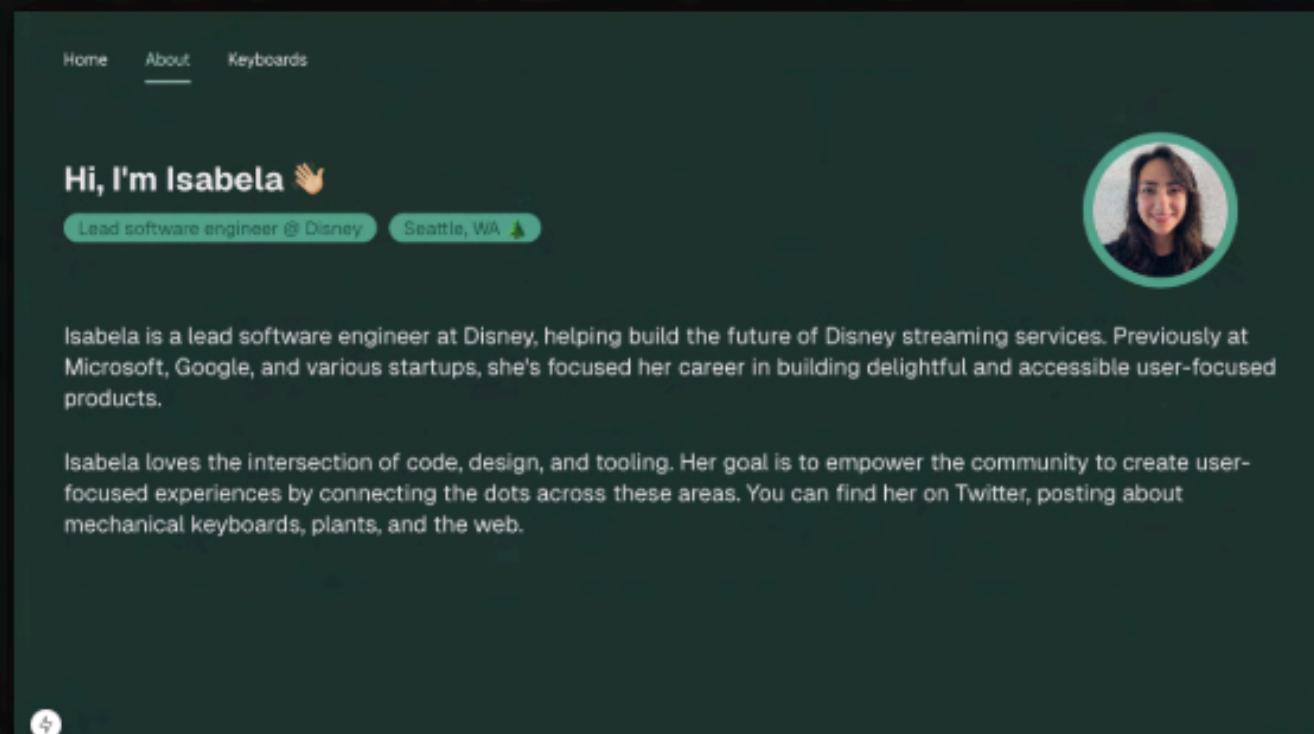
- Visual regression test: /about (Failed, vrt.spec.ts:14) - 999ms
- Visual regression test: / (Passed, vrt.spec.ts:14) - 663ms
- Visual regression test: /keyboards (Passed, vrt.spec.ts:14) - 761ms

Automated visual regression testing

▼ Image mismatch: -about.png

Diff Actual Expected **Side by side** Slider

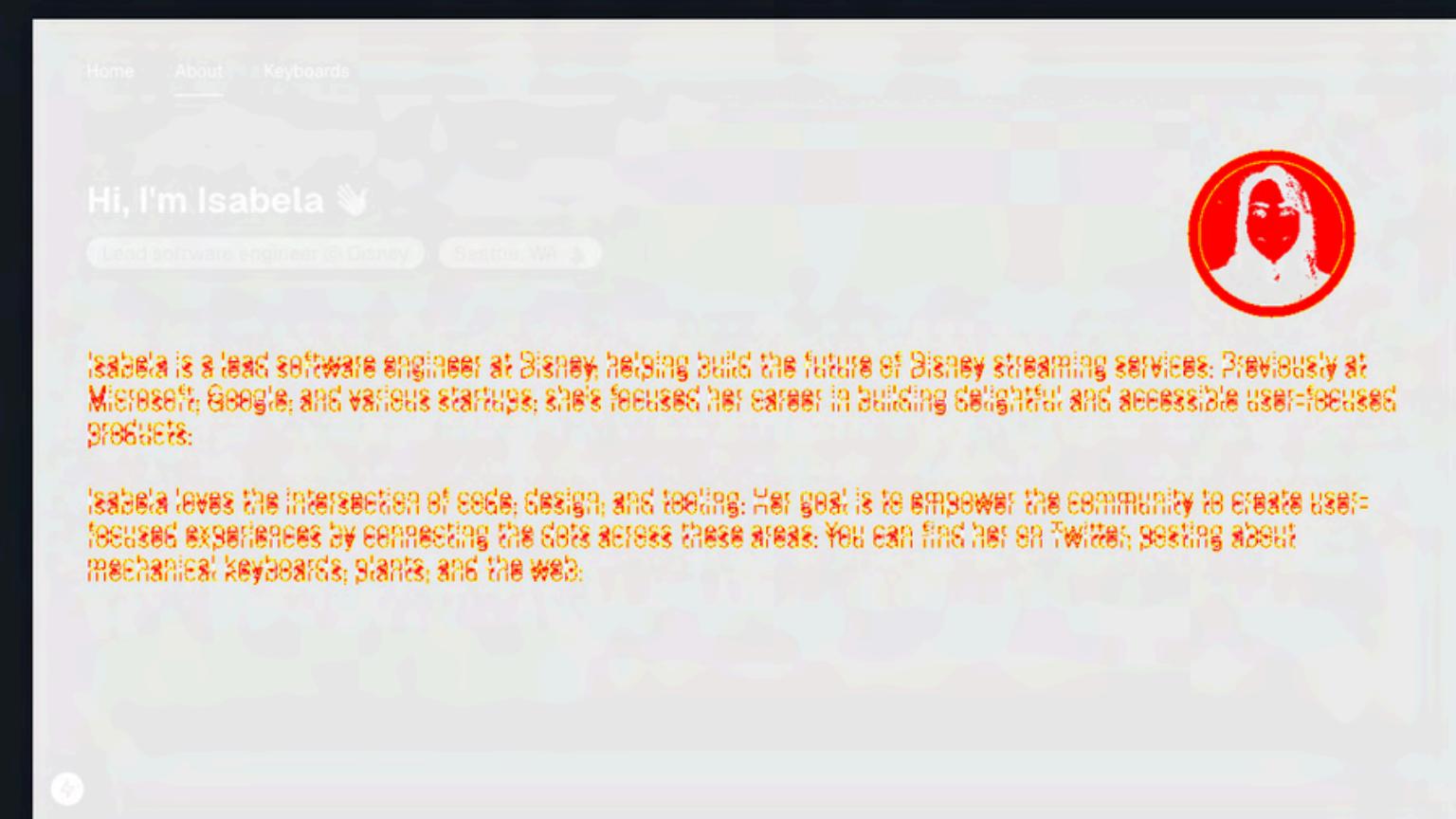
Expected 1280 x 720



Errors

```
Error: expect(page).toHaveScreenshot(expected)
30696 pixels (ratio 0.04 of all image pixels) are different.
```

Diff Actual Expected Side by side Slider



Let's talk about noise and flakiness

Causes

- Different environments, viewports, resolutions
- Animations
- Async loading / dynamic content



WARNING

Browser rendering can vary based on the host OS, version, settings, hardware, power source (battery vs. power adapter), headless mode, and other factors. For consistent screenshots, run tests in the same environment where the baseline screenshots were generated.

Let's talk about noise and flakiness

Causes

- **Different environments, viewports, resolutions**
- Animations
- Async loading / dynamic content

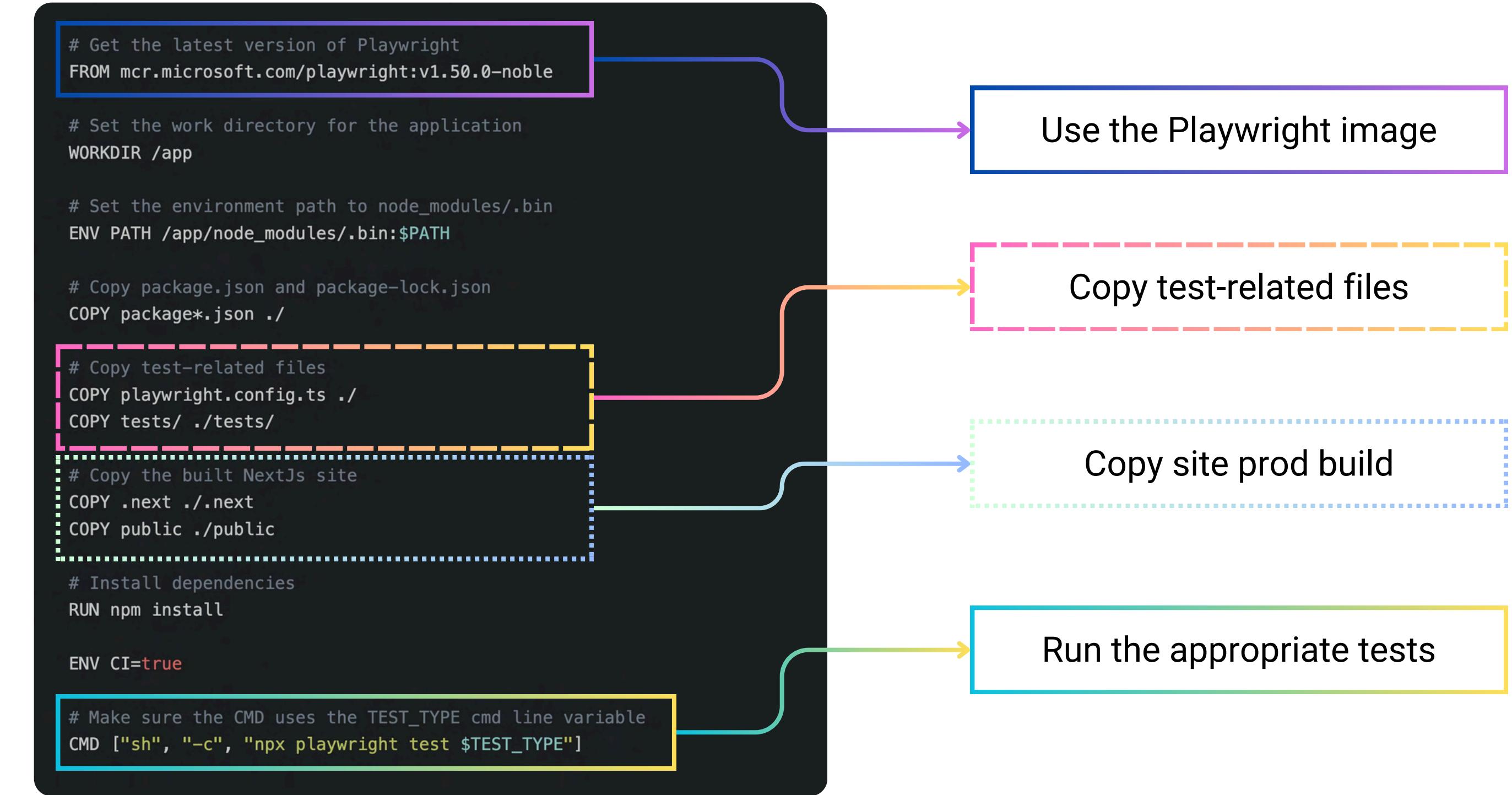
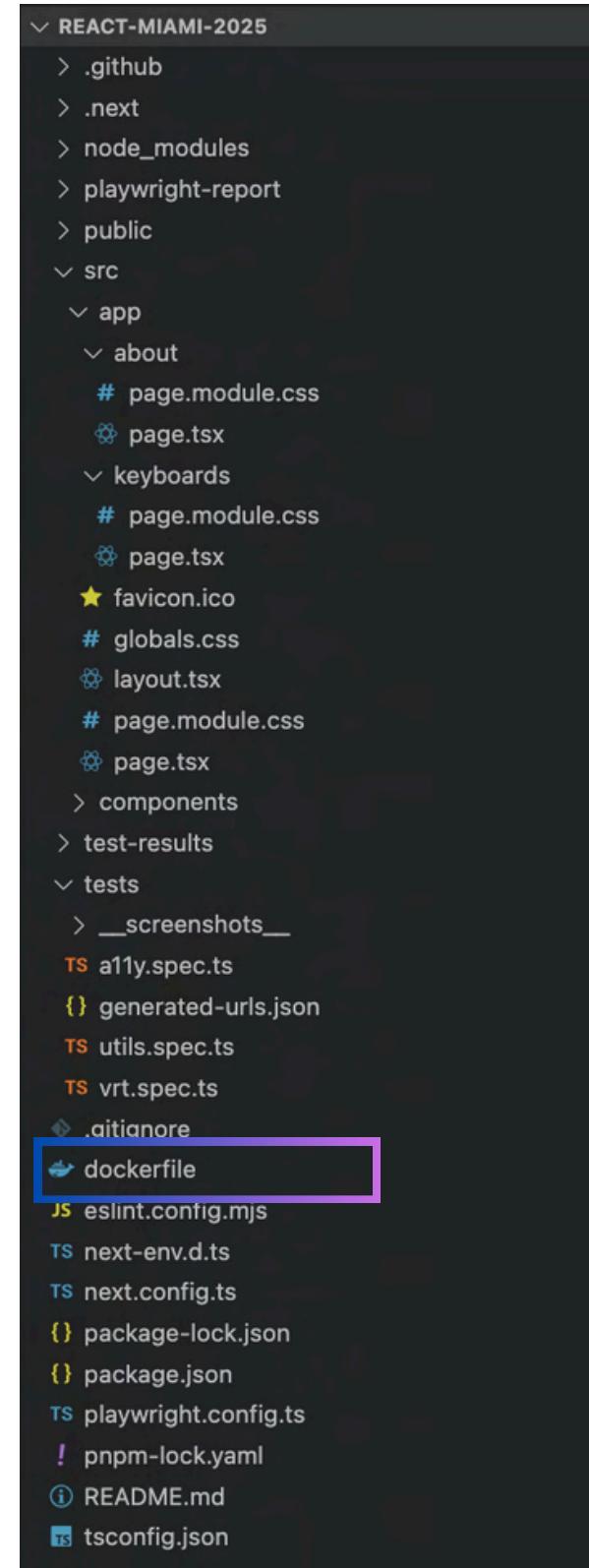


WARNING

Browser rendering can vary based on the host OS, version, settings, hardware, power source (battery vs. power adapter), headless mode, and other factors. For consistent screenshots, run tests in the same environment where the baseline screenshots were generated.

Adding Docker to the workflow

Adding Docker to the workflow



Adding Docker to the workflow

```
REACT-MIAMI-2025
> .github
> .next
> node_modules
> playwright-report
> public
< src
  < app
    < about
      # page.module.css
      page.tsx
    < keyboards
      # page.module.css
      page.tsx
    < favicon.ico
    # globals.css
    layout.tsx
    # page.module.css
    page.tsx
  > components
  > test-results
  < tests
    < __screenshots__
      a1y.spec.ts
      generated-urls.json
      utils.spec.ts
      vrt.spec.ts
    < .gitignore
    dockerfile
    eslint.config.mjs
    next-env.d.ts
    next.config.ts
    package-lock.json
    package.json
    playwright.config.ts
    ! pnpm-lock.yaml
    README.md
    tsconfig.json
```

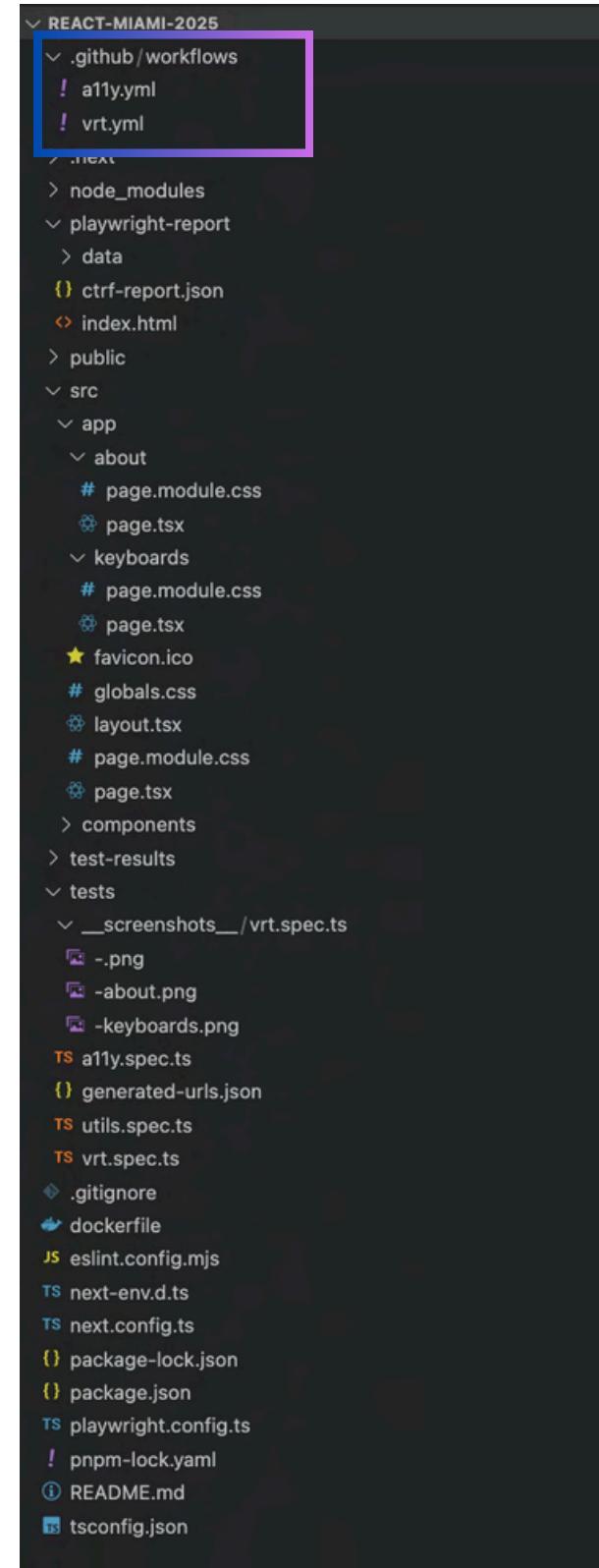
```
{
  "name": "react-miami-2025",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "playwright:generate-urls": "playwright test utils",
    "playwright:a11y:local": "pnpm playwright:generate-urls && playwright test a11y && pnpm open:playwright-report",
    "playwright:vrt:local": "pnpm playwright:generate-urls && playwright test vrt && pnpm open:playwright-report",
    "playwright:a11y:container": "pnpm build && pnpm build:docker && TEST_TYPE=a11y pnpm run:docker",
    "playwright:vrt:container": "pnpm build && pnpm build:docker && TEST_TYPE=vrt pnpm run:docker",
    "build:docker": "docker build -t react-miami-2025-playwright .",
    "run:docker": "docker run -e TEST_TYPE react-miami-2025-playwright",
    "copy:docker-report": "docker cp $(docker ps -alq):/app/playwright-report ./playwright-report",
    "cleanup:docker": "docker stop $(docker ps -alq) && docker rm $(docker ps -alq)"
    "open:playwright-report": "open ./playwright-report/index.html",
  },
  ...
}
```

Build the site, build docker image, run appropriate test

Helper Docker commands

Putting it all together in GitHub actions

Putting it all together in GitHub actions



```
name: Run visual regression tests

on:
  pull_request:
    branches:
      - '**' # Trigger action on all PR branches.

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      # Step 1: Checkout the code from the PR
      - name: Checkout code
        uses: actions/checkout@v2

      # Step 2: Set up Node.js (for Next.js build)
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '20'

      # Step 3: Install dependencies
      - name: Install dependencies
        run:
          npm install -g pnpm # Install pnpm globally
          pnpm install # Install dependencies using pnpm

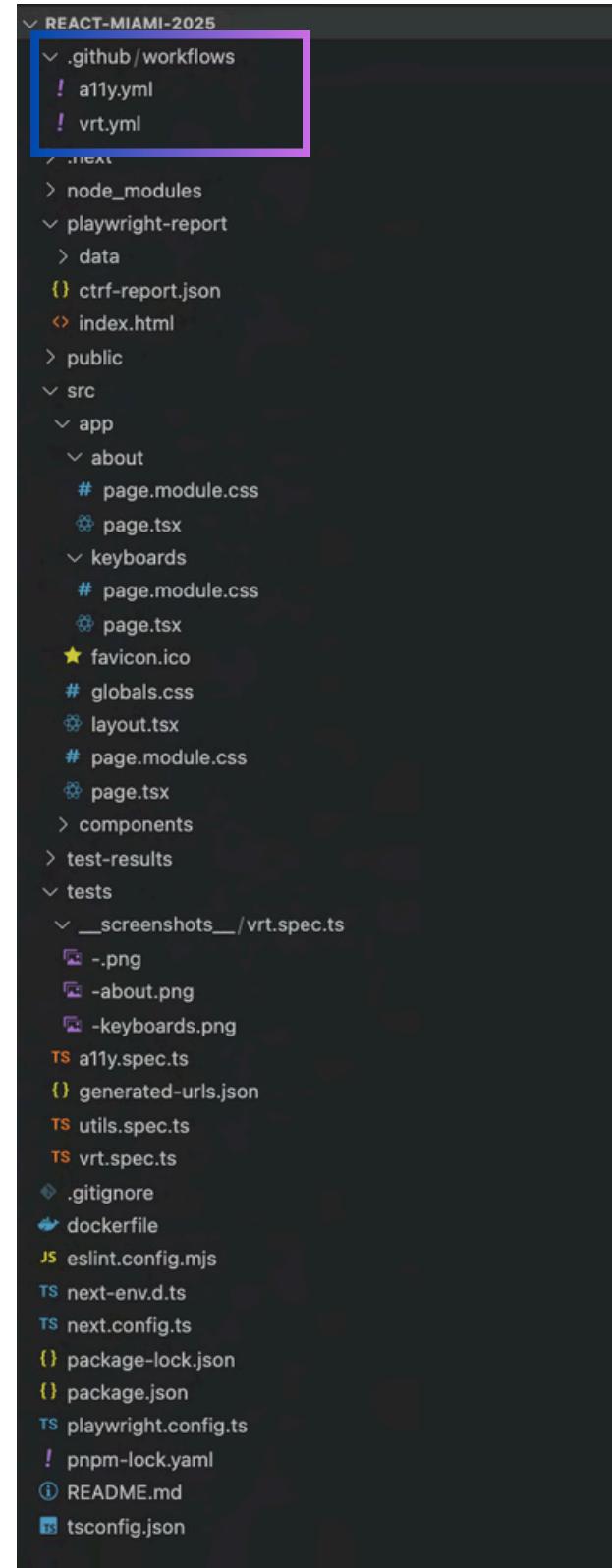
      # Step 4: Run the test command - builds the NextJS app, the Docker container, and runs tests
      - name: Run tests
        run: pnpm playwright:vrt:container
```

Trigger when we push changes to any PR branch

Generic setup work for any action in our codebase

Actually run our tests

Putting it all together in GitHub actions



```
# Step 5: Custom condition to fail job if the tests step failed
- name: Check test result and fail job if needed
  if: failure() # This will run only if the previous step fails
  run: |
    echo "Tests failed, marking job as failed"
    exit 1 # Explicitly fail the job

- name: Copy test results from Docker container to host
  if: always()
  run: pnpm copy:docker-report

# Step 6: Upload Playwright reports as artifacts
- name: Upload Playwright JSON and HTML report artifacts
  if: always()
  uses: actions/upload-artifact@v4
  with:
    name: playwright-report # Artifact name
    path: playwright-report/

# Step 7: Render the JSON report in the job for easier debugging
- name: Publish Test Report
  uses: ctrf-io/github-test-reporter@v1
  with:
    report-path: 'playwright-report/*.json'
    if: always()

# Step 8: Clean up Docker images (to save space on the runner)
- name: Clean up Docker images
  if: always()
  run: |
    docker container prune -f # Remove the container
    docker image prune -f      # Remove unused images
```

Explicitly fail the job if the tests fail, but still copy the test report from Docker

Upload the test report as an artifact in our job run

Always publish the test report artifact

Docker cleanup work

Putting it all together in GitHub actions

The screenshot shows a GitHub pull request page for the repository `isabelacmor / playwright-docker`. The pull request is titled "Sample PR to trigger GitHub actions #1". The status bar indicates "Open" and "isabelacmor wants to merge 1 commit into `main` from `update-readme`".

Below the title, there are tabs for Conversation (0), Commits (1), Checks (0), and Files changed (1). A comment from `isabelacmor` states:

This PR updates the readme and demos triggering the GitHub actions for both accessibility and visual regression tests. On the first commit, both tests will intentionally fail. The accessibility test will fail because of missing alt text on images. The visual regression test will fail because of a mismatched icon on the home page.

The commit message is "Update read me" and the commit hash is "30079ac".

At the bottom, a summary box shows:

- Some checks haven't completed yet** (2 in progress checks)
 - Run accessibility tests / build (pull_request)** In progress — This check has started... [Details](#)
 - Run visual regression tests / build (pull_request)** In progress — This check has started... [Details](#)
- This branch has no conflicts with the base branch**
Merging can be performed automatically.

At the bottom right, there are buttons for "Merge pull request" and "You can also open this in GitHub Desktop or view command line instructions."

Putting it all together in GitHub actions

The screenshot shows a GitHub pull request page for the repository `isabelacmor / playwright-docker`. The pull request is titled "Sample PR to trigger GitHub actions #1". The status bar indicates "isabelacmor wants to merge 1 commit into `main` from `update-readme`".

Below the title, there are links for "Conversation 0", "Commits 1", "Checks 0", and "Files changed 1". A comment from `isabelacmor` states:

This PR updates the readme and demos triggering the GitHub actions for both accessibility and visual regression tests. On the first commit, both tests will intentionally fail. The accessibility test will fail because of missing alt text on images. The visual regression test will fail because of a mismatched icon on the home page.

The commit message is "Update read me" with a commit hash of `30079ac`.

A prominent yellow callout box displays the GitHub Actions results:

- All checks have failed** (2 failing checks)
 - Run accessibility tests / build (pull_request)** Failing after 1m
 - Run visual regression tests / build (pull_request)** Failing after 1m
- This branch has no conflicts with the base branch**
Merging can be performed automatically.

At the bottom, there is a "Merge pull request" button and a note: "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)".

Putting it all together in GitHub actions

← Back to pull request #1

Sample PR to trigger GitHub actions #1

Summary

Triggered via pull request 2 minutes ago
isabelacmor opened #1 update-readme

Status: Failure | Total duration: 1m 37s | Artifacts: 1

Jobs

build

a11y.yml
on: pull_request

build 1m 29s

Run details

Usage

Workflow file

build summary

Summary

Tests 📈	Passed ✅	Failed ❌	Skipped ⏱	Pending 🚧	Other ?	Flaky 💀	Duration 🕒
3	2	1	0	0	0	0	6.2s

Failed Tests

Failed Tests ❌	Fail Message
	Error: expect(received).toEqual(expected) // deep equality - Expected - 1 + Received + 1104 - Array [] + Array [+ Object { + "description": "Ensure elements have alternative text or a role of none or presentation", + "help": "Images must have alternative text", + "helpUrl": "https://dequeuniversity.com/rules/axe/4.10/image-alt?application=playwright", + "id": "image-alt", + "impact": "critical", + "nodes": Array [+ Object { + "all": Array [], + "any": Array [+ Object {

← Back to pull request #1

Sample PR to trigger GitHub actions #1

Summary

Triggered via pull request 3 minutes ago
isabelacmor opened #1 update-readme

Status: Failure | Total duration: 1m 35s | Artifacts: 1

Jobs

build

vrt.yml
on: pull_request

build 1m 27s

Run details

Usage

Workflow file

build summary

Summary

Tests 📈	Passed ✅	Failed ❌	Skipped ⏱	Pending 🚧	Other ?	Flaky 💀	Duration 🕒
3	2	1	0	0	0	0	5.2s

Failed Tests

Failed Tests ❌	Fail Message
	Visual regression test: / Error: expect(page).toHaveScreenshot(expected) 2347 pixels (ratio 0.01 of all image pixels) are different. Expected: /app/tests/__screenshots__/vrt.spec.ts/-_.png Received: /app/test-results/vrt-Visual-regression-test--chromium-retry2--actual.png Diff: /app/test-results/vrt-Visual-regression-test--chromium-retry2--diff.png Call log: - expect.toHaveScreenshot(-.png) with timeout 5000ms - verifying given screenshot expectation - taking page screenshot - disabled all CSS animations - waiting for fonts to load... - fonts loaded - 2347 pixels (ratio 0.01 of all image pixels) are different. - waiting 100ms before taking screenshot - taking page screenshot - disabled all CSS animations

Putting it all together in GitHub actions

The screenshot shows a GitHub pull request page for the repository `isabelacmor / playwright-docker`. The pull request is titled "Sample PR to trigger GitHub actions #1" and is marked as "Open". The user `isabelacmor` has commented: "This PR updates the readme and demos triggering the GitHub actions for both accessibility and visual regression tests. On the first commit, both tests will intentionally fail. The accessibility test will fail because of missing alt text on images. The visual regression test will fail because of a mismatched icon on the home page." Below the comment, there are two commits: "Update read me" (status: `✗ 30079ac`) and "Add alt text to images; Update screenshot for home page" (status: `✓ 43d32af`). A sidebar on the right provides status for GitHub Actions checks: "All checks have passed" (2 successful checks), "Run accessibility tests / build (pull_request)" (Successful in 1m), "Run visual regression tests / build (pull_request)" (Successful in 1m), and "This branch has no conflicts with the base branch" (Merging can be performed automatically). At the bottom, there is a "Merge pull request" button and a note: "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)". A "Try the new merge experience" link is at the bottom right.

Putting it all together in GitHub actions

← Back to pull request #1
✓ Sample PR to trigger GitHub actions #2

Summary

Triggered via pull request 2 minutes ago
isabelacmor synchronize #1 update-readme Status Success Total duration 1m 33s Artifacts 1

Jobs
build

a11y.yml
on: pull_request

build 1m 25s

Run details Usage Workflow file

build summary

Summary

Tests 📈	Passed ✅	Failed ✗	Skipped 🔍	Pending 🚧	Other ?	Flaky 💀	Duration ⏱
3	3	0	0	0	0	0	2.6s

Failed Tests
No failed tests ✨

Flaky Tests
No flaky tests ✨

Skipped
No skipped tests ✨

Tests

Test Name	Status	Flaky	Duration
Accessibility test: /	✓		581ms
Accessibility test: /about	✓		571ms
Accessibility test: /keyboards	✓		734ms

[Github Test Reporter by CTRF](#) ❤️
Job summary generated at run-time

← Back to pull request #1
✓ Sample PR to trigger GitHub actions #2

Summary

Triggered via pull request 3 minutes ago
isabelacmor synchronize #1 update-readme Status Success Total duration 1m 36s Artifacts 1

Jobs
build

vrt.yml
on: pull_request

build 1m 26s

Run details Usage Workflow file

build summary

Summary

Tests 📈	Passed ✅	Failed ✗	Skipped 🔍	Pending 🚧	Other ?	Flaky 💀	Duration ⏱
3	3	0	0	0	0	0	1.8s

Failed Tests
No failed tests ✨

Flaky Tests
No flaky tests ✨

Skipped
No skipped tests ✨

Tests

Test Name	Status	Flaky	Duration
Visual regression test: /	✓		303ms
Visual regression test: /about	✓		322ms
Visual regression test: /keyboards	✓		536ms

[Github Test Reporter by CTRF](#) ❤️
Job summary generated at run-time

More optimizations

Consider...

- One action responsible for building production site
- More advanced tooling for accepting new baselines
- Adopting other tools like Jenkins for CI/CD as you scale

Final thoughts

Automated testing lets you...

- Ship high-quality code with confidence
- Focus on adding user value

Quick wins with...

- Accessibility testing
- Visual regression testing
- CI/CD integration

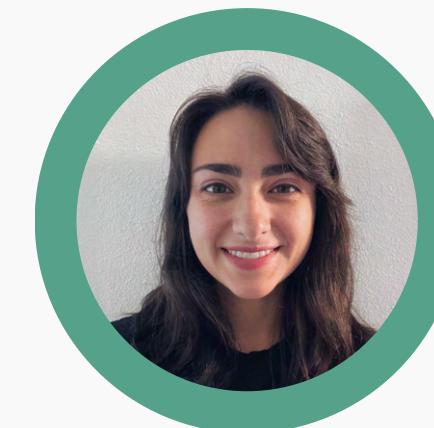
Resources

- Demo repo

<https://github.com/isabelacmor/playwright-docker>

- Slides

<https://github.com/isabelacmor/talks/blob/master/automated-playwright.md>



Want to chat more?

Find me at the conference or online



@isabelacmor