Automated Visual Regression & Accessibility Testing

(made easy ✨)

React Miami 2025

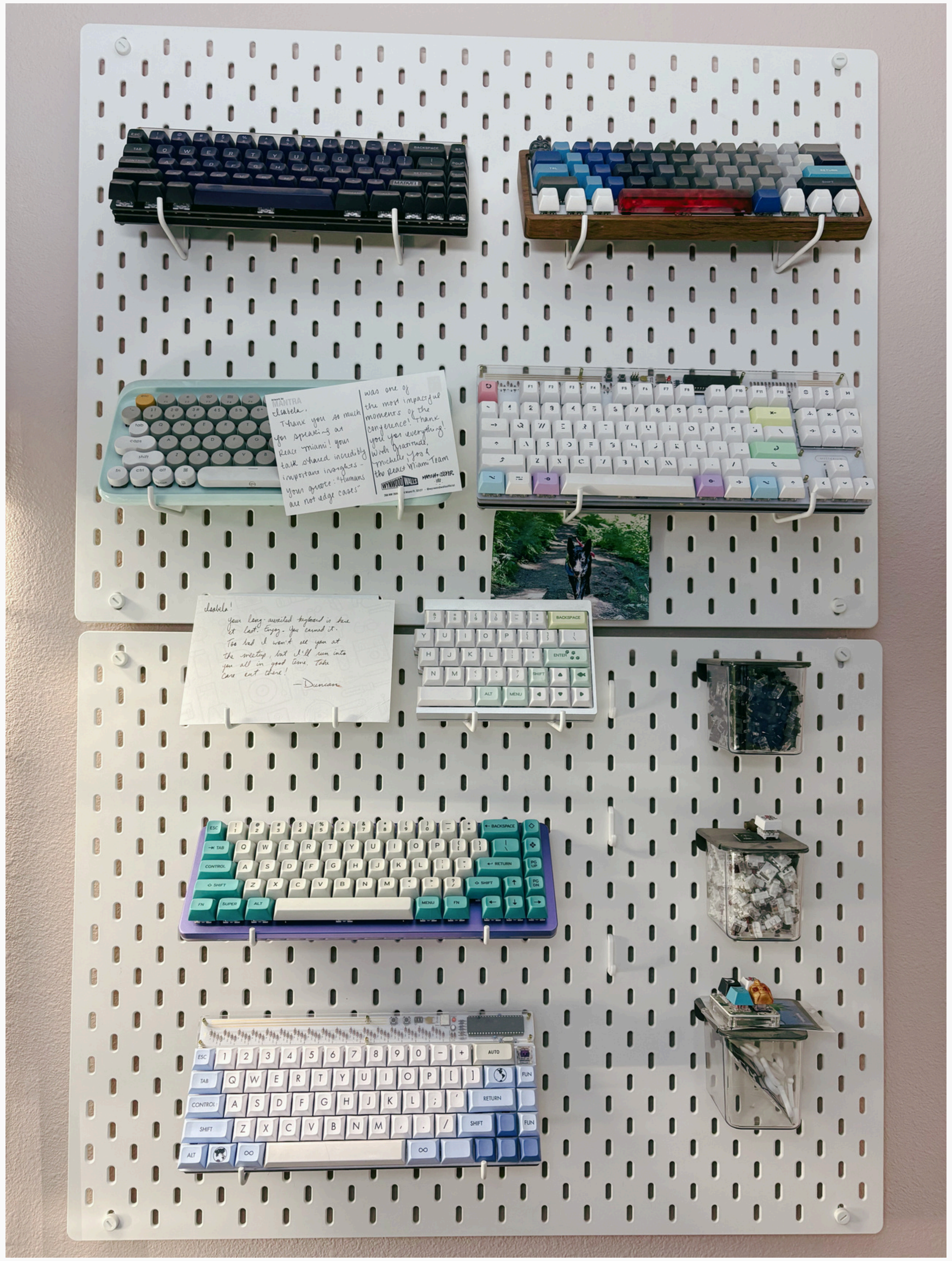
Hi, I'm Isabela

Lead software engineer @ Disney

Seattle

@isabelacmor


MIAMI BEACH
FL




Noogle
Microsoft
Esc
DALMATIAN COAST
CROATIA
ISABELA
Beginnings

Automated visual regression & accessibility testing

- What's in our toolbox?
- Automated accessibility testing
- Automated visual regression testing
- Putting it all together in GitHub actions

@isabelacmor

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

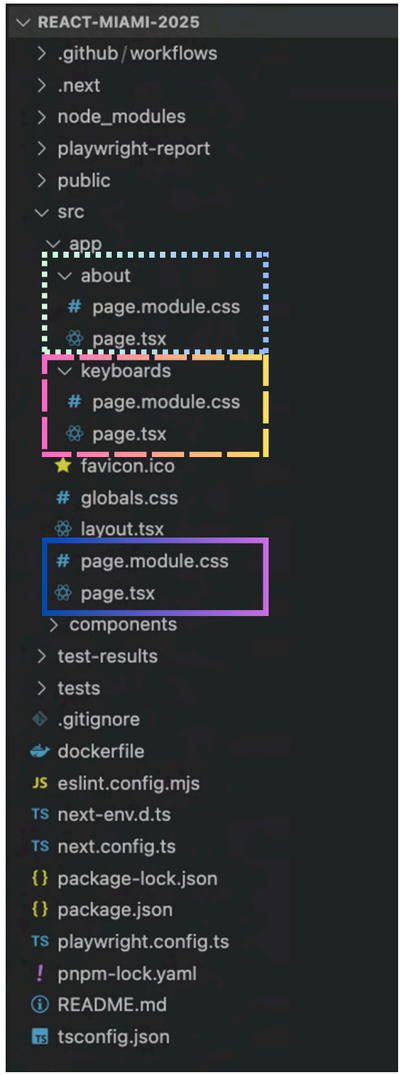
GitHub Actions

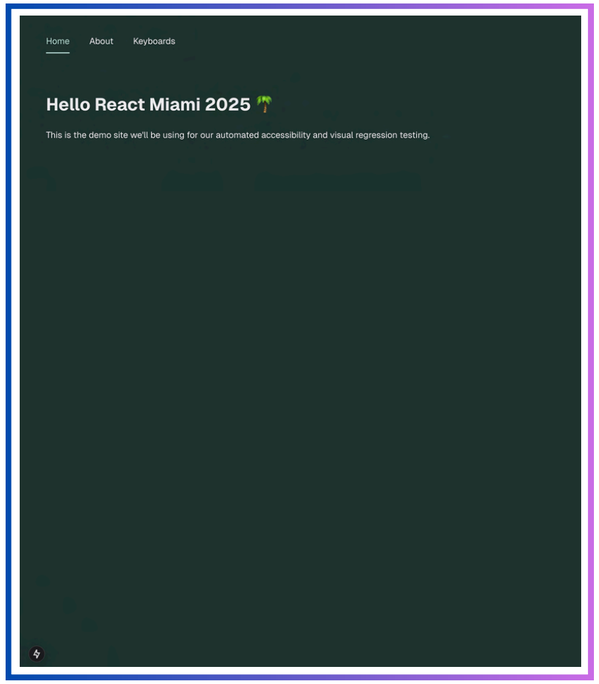
Docker

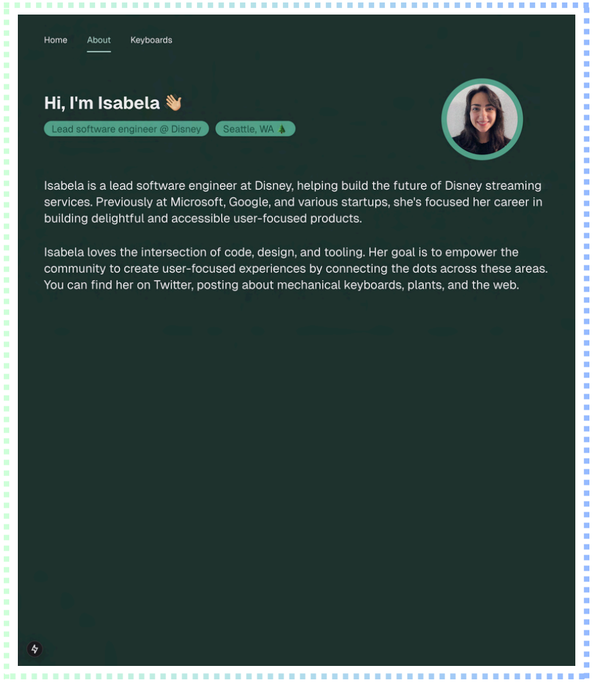
- Platform for building and running containers
- Ensures consistency across environments

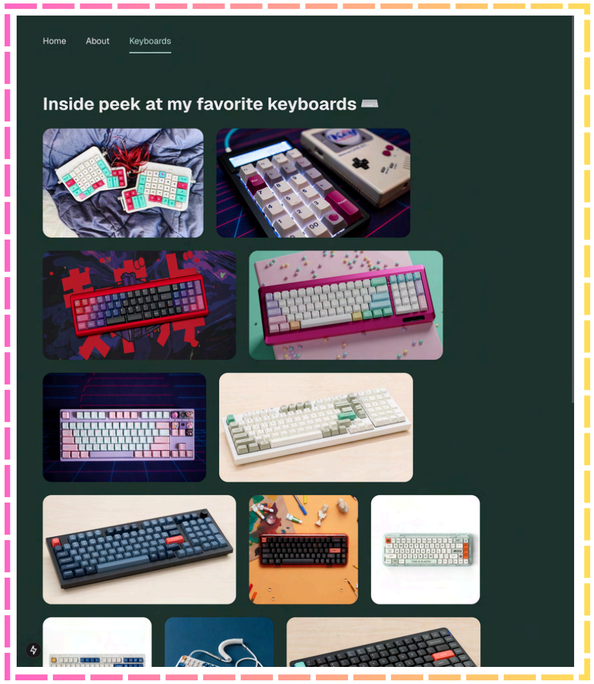
Automated accessibility testing

Our demo NextJS app







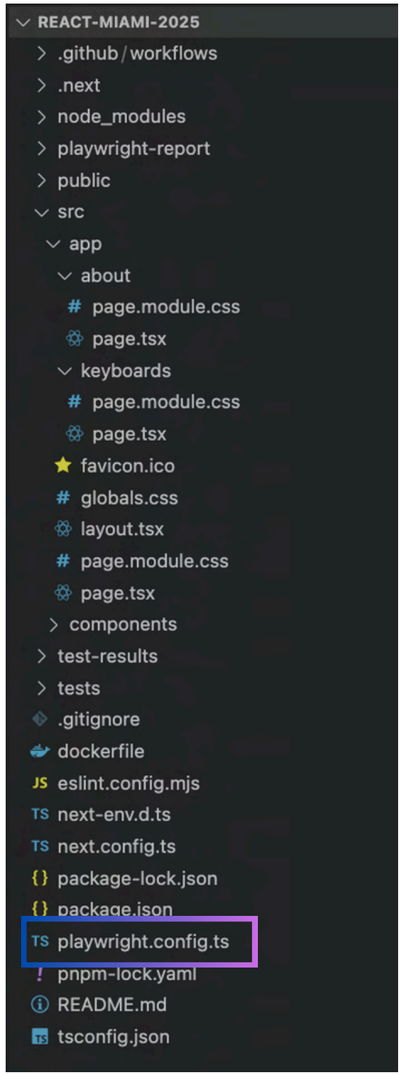


Automated accessibility testing

```
pnpm create playwright
```

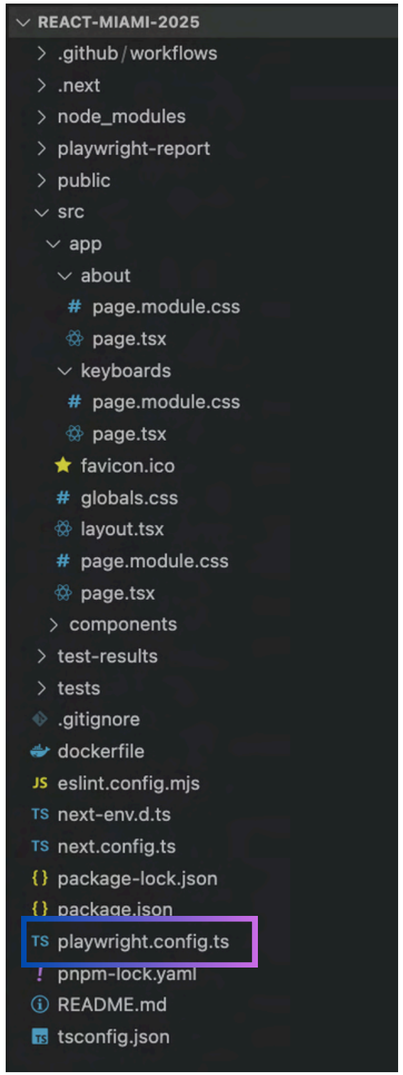
- Installs Playwright and takes you through setup steps

Automated accessibility testing



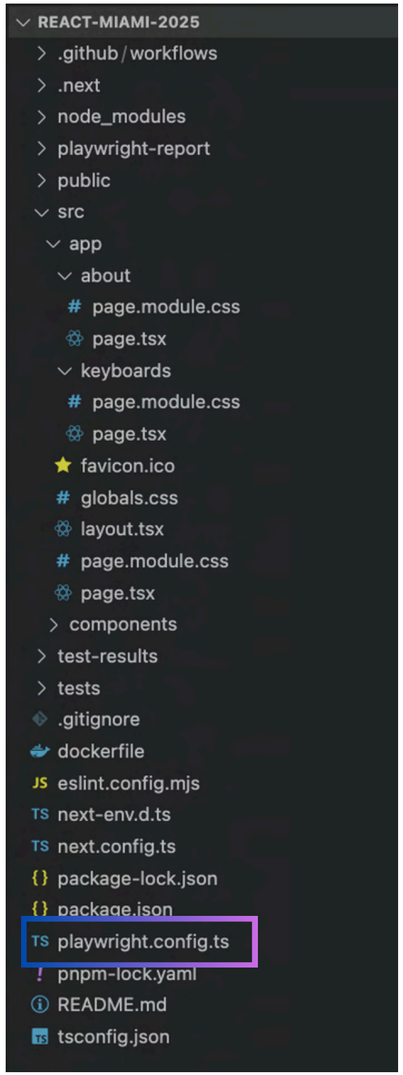

```ts
import { defineConfig, devices } from '@playwright/test';

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



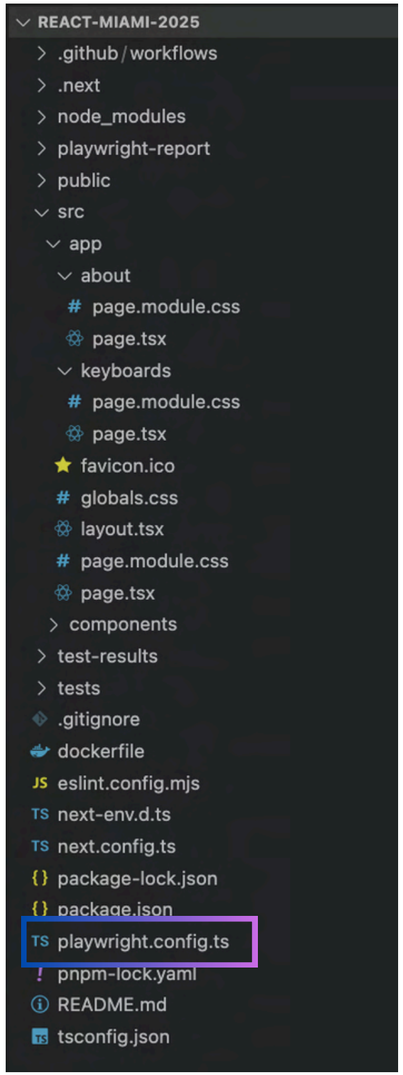

```ts
import { defineConfig, devices } from '@playwright/test';

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



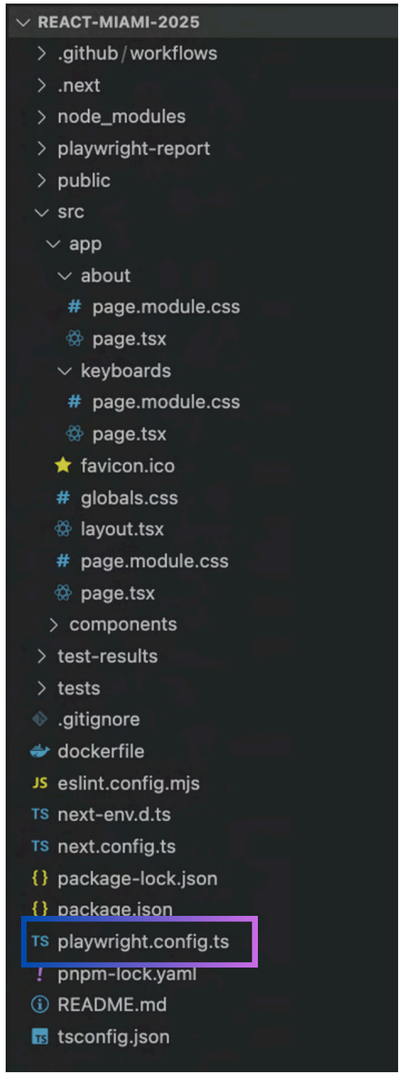

```ts
import { defineConfig, devices } from '@playwright/test';

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



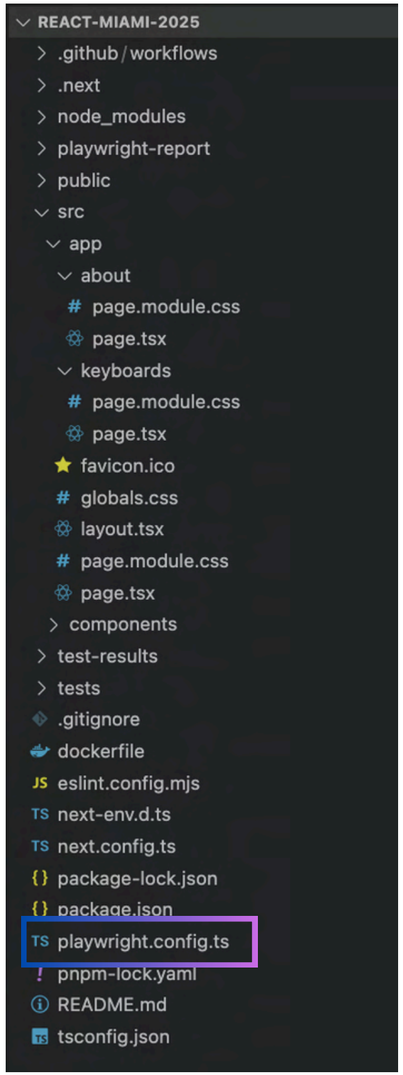

```ts
import { defineConfig, devices } from '@playwright/test';

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



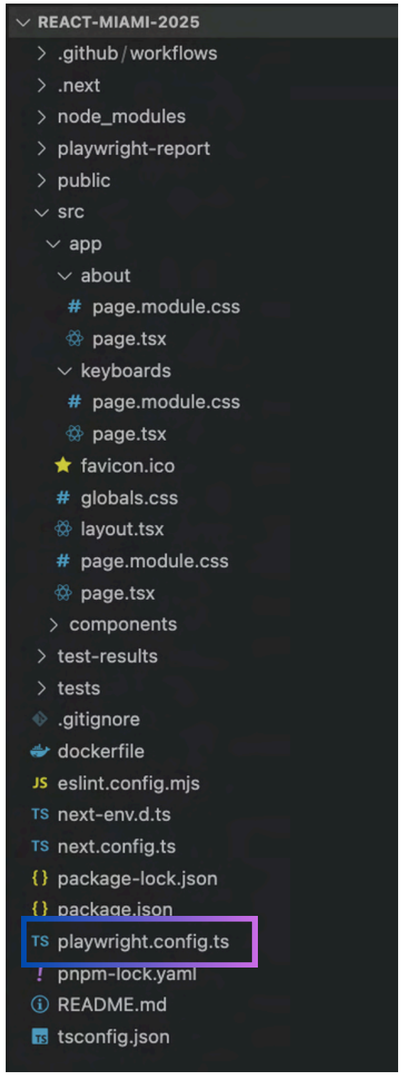

```ts
import { defineConfig, devices } from '@playwright/test';

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



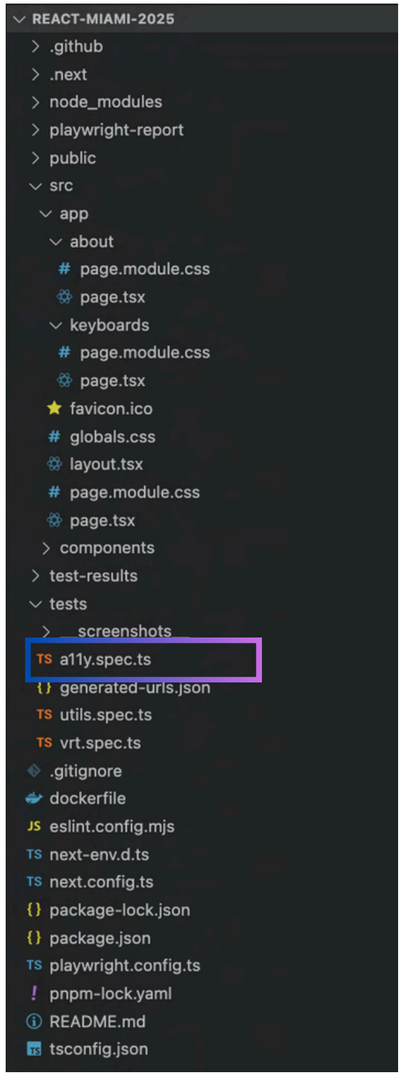

```ts
import { defineConfig, devices } from '@playwright/test';

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




```ts
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



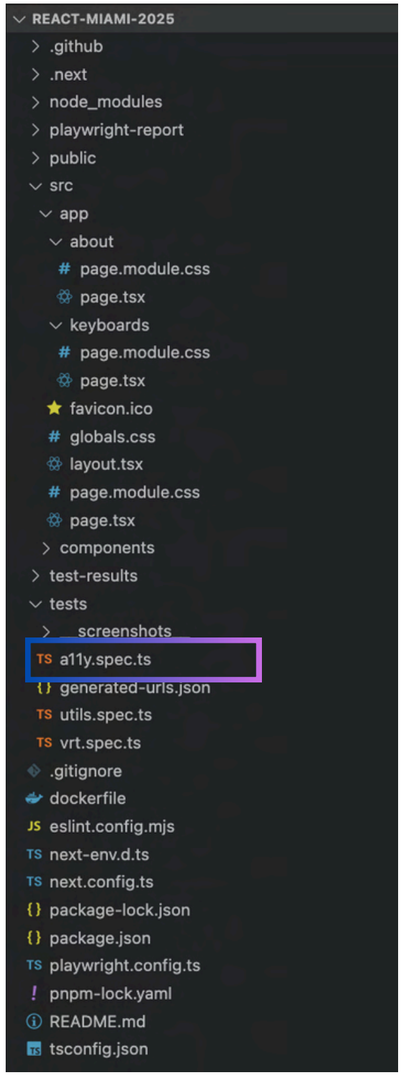

```ts
import AxeBuilder from '@axe-core/playwright';
import { expect, test } from '@playwright/test';

/**
 * 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

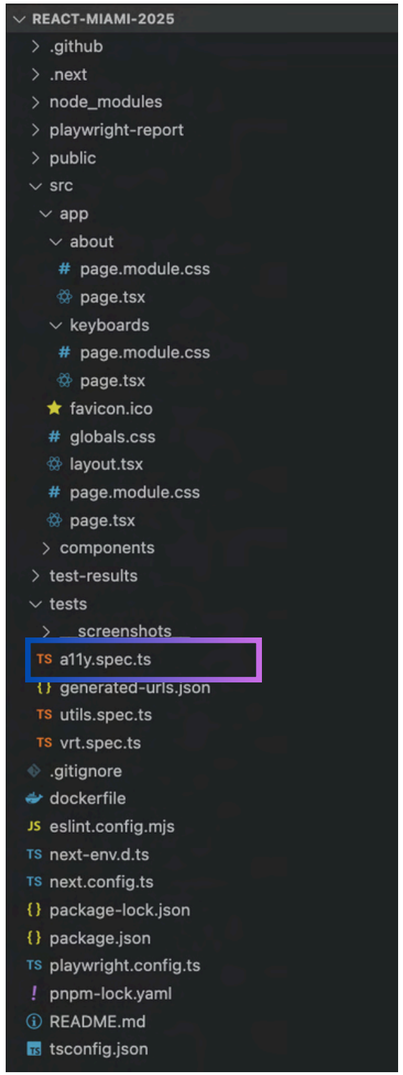
Automated accessibility testing



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

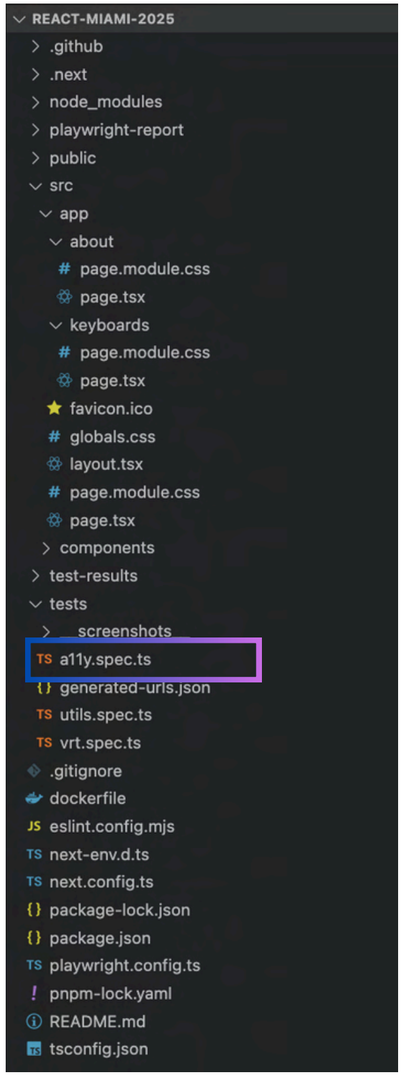


How do we scale this?

```
/**
 * 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 eac
 *
 * Error: Playwright Test did not expect test() to be call
 */
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.

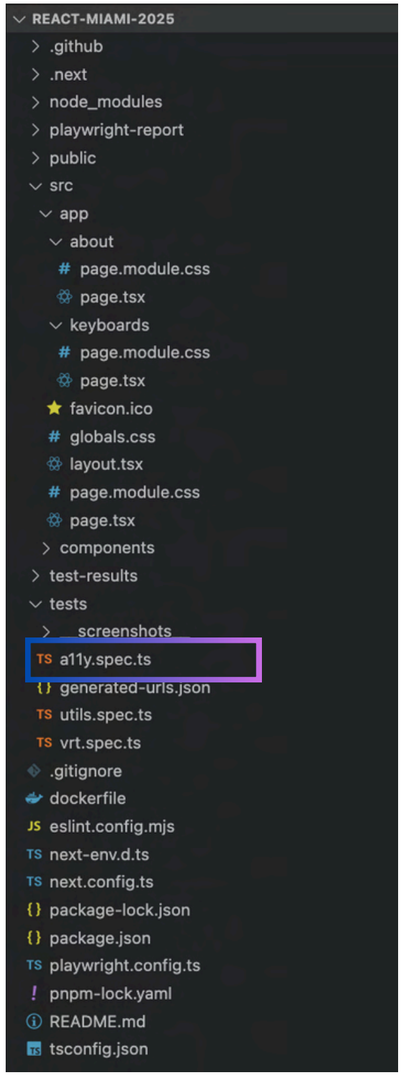
Automated accessibility testing



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…

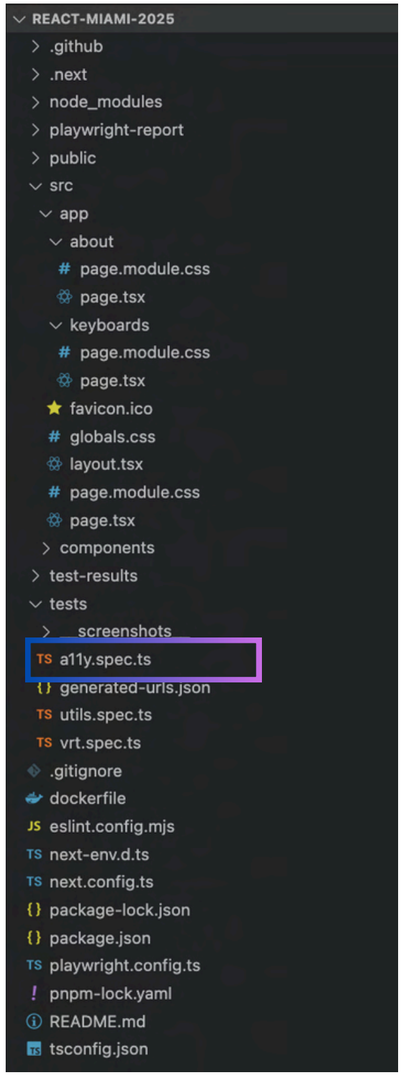
Pros:

- 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.

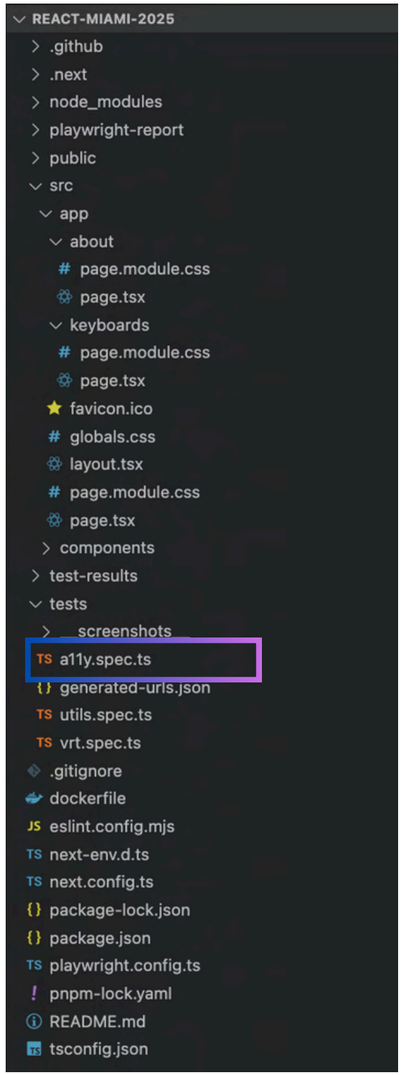
Automated accessibility testing



How do we scale this?

```
/**
 * Approach #2: Multiple tests with pre-generation.
 * Before we run the a11y 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 a11y 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

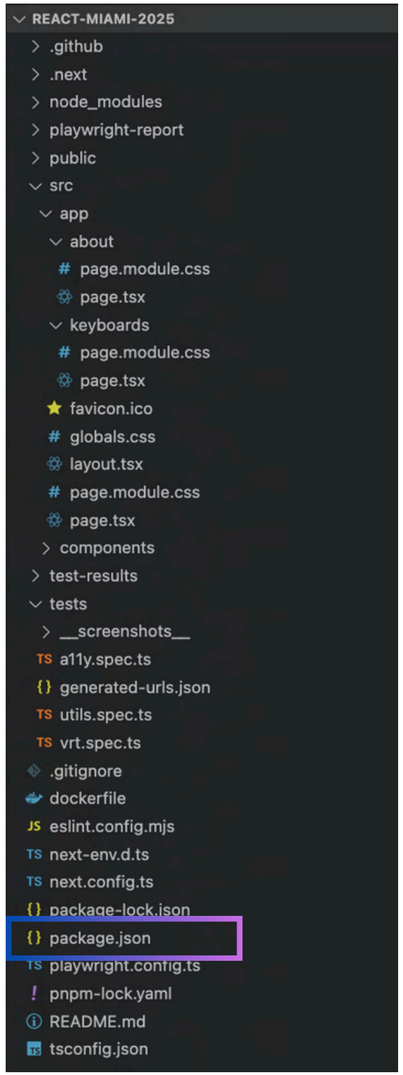
Pros:

- 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

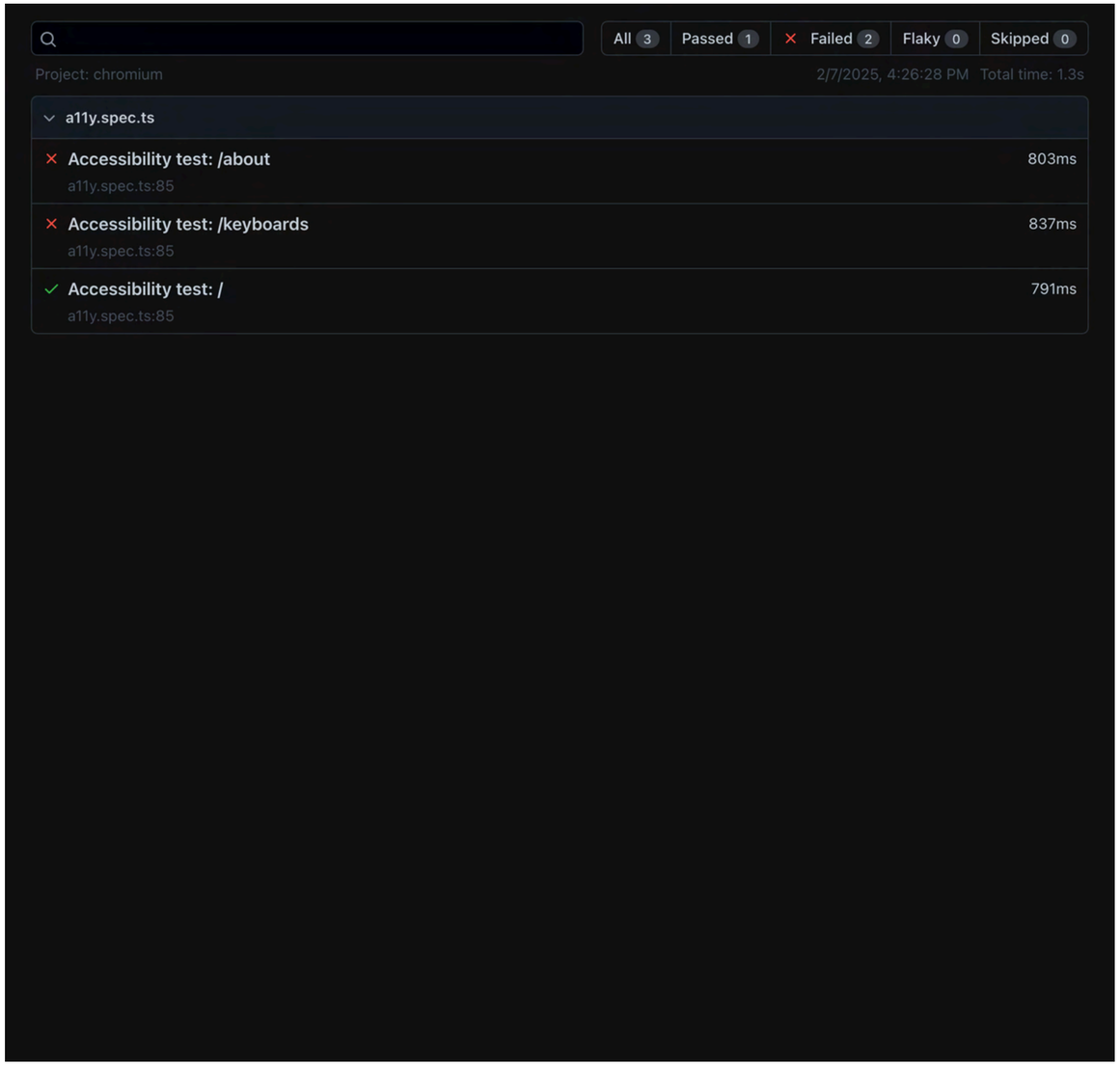



```
{
  "name": "react-miami-2025",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "playwright:generate-urls": "playwright test utils",
    "playwright:a11y:local": "pnpm playwright:generate-urls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

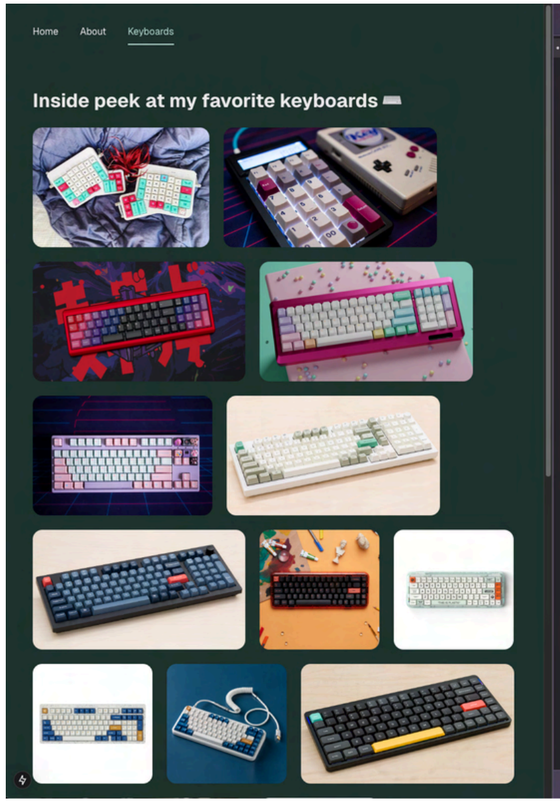
Generate test URLs

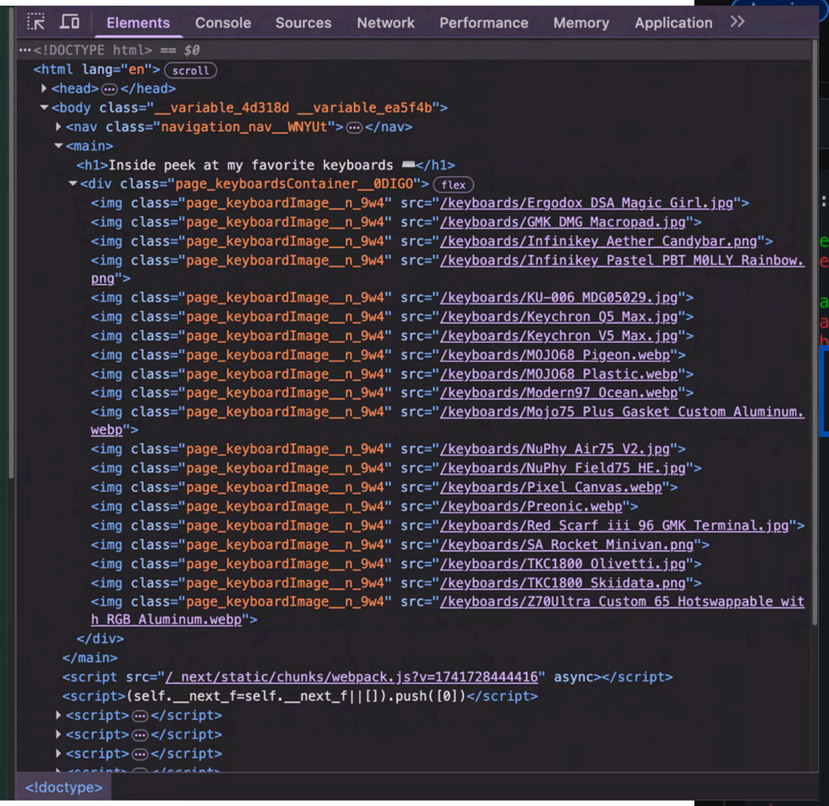
Run generation command, run a11y tests, open report

Automated accessibility testing



Automated accessibility testing





All 3 | Passed 1 | Failed 2 | Flaky 0 | Skipped 0

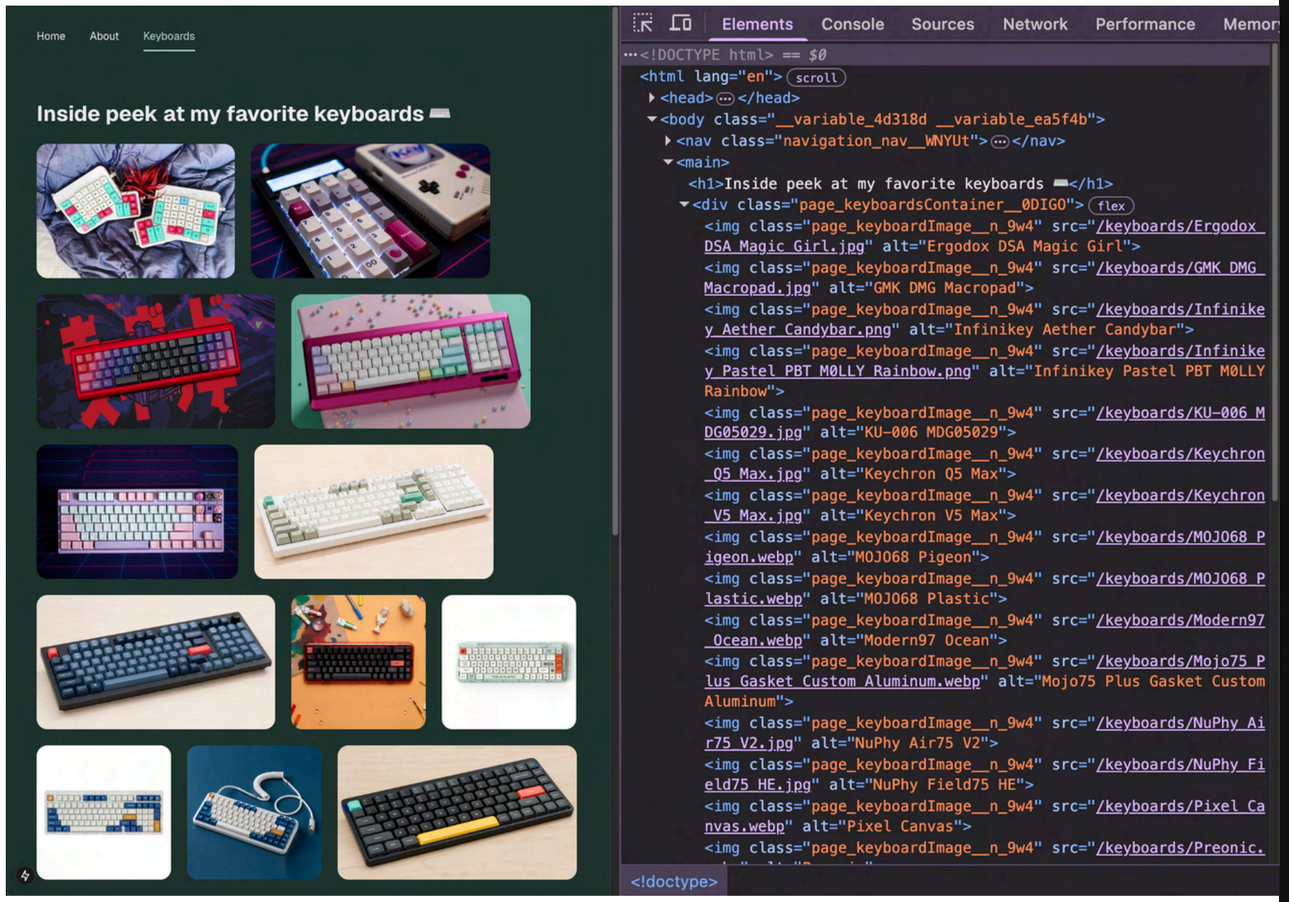
« previous next »

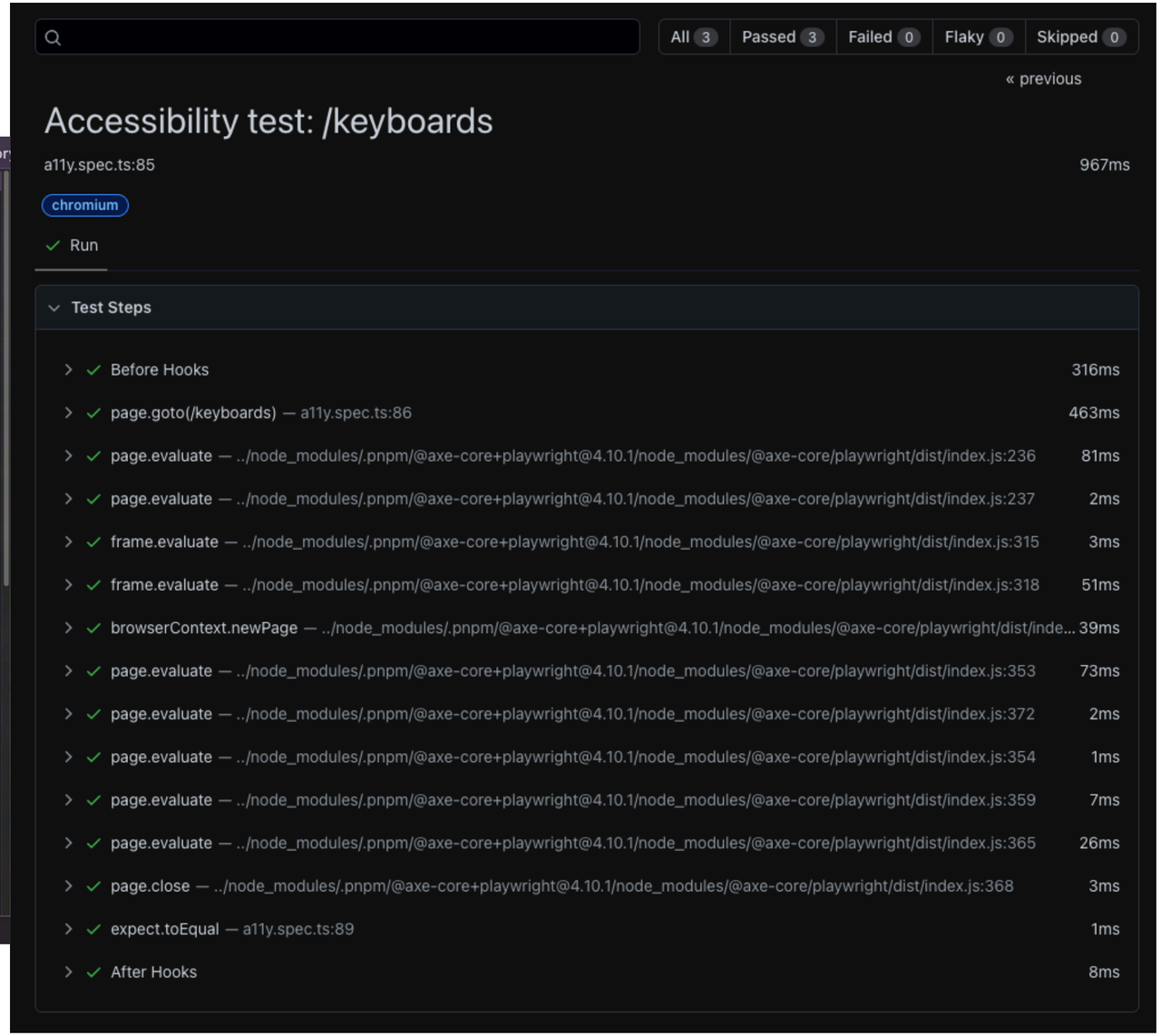
Accessibility test: /keyboards

a11y.spec.ts:85 837ms

```
: expect(received).toEqual(expected) // deep equality

ected  -    1
eived  + 1104

ay []
ay [
bject {
 "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
         "relatedNodes": Array [],
       },
```

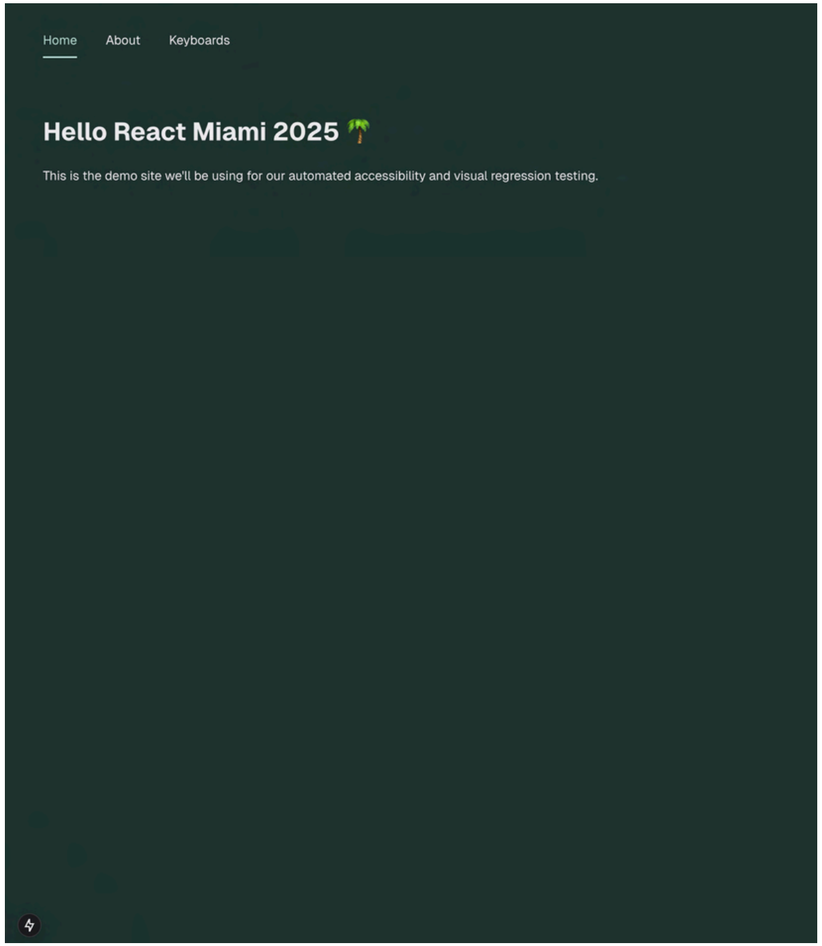
Automated accessibility testing

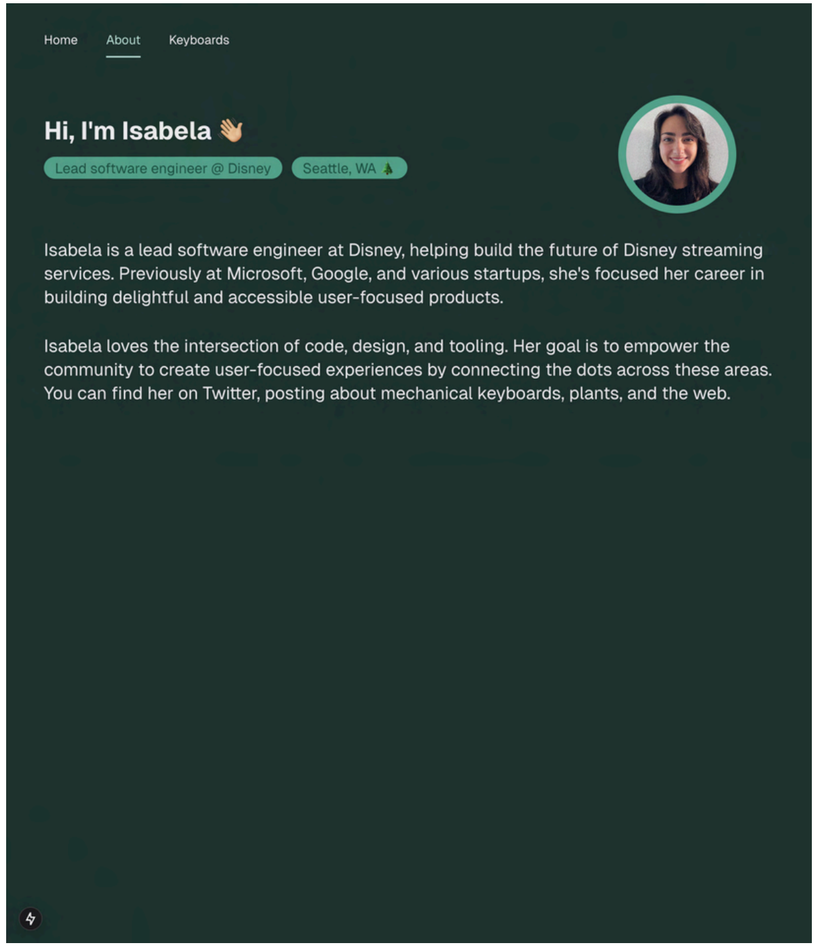


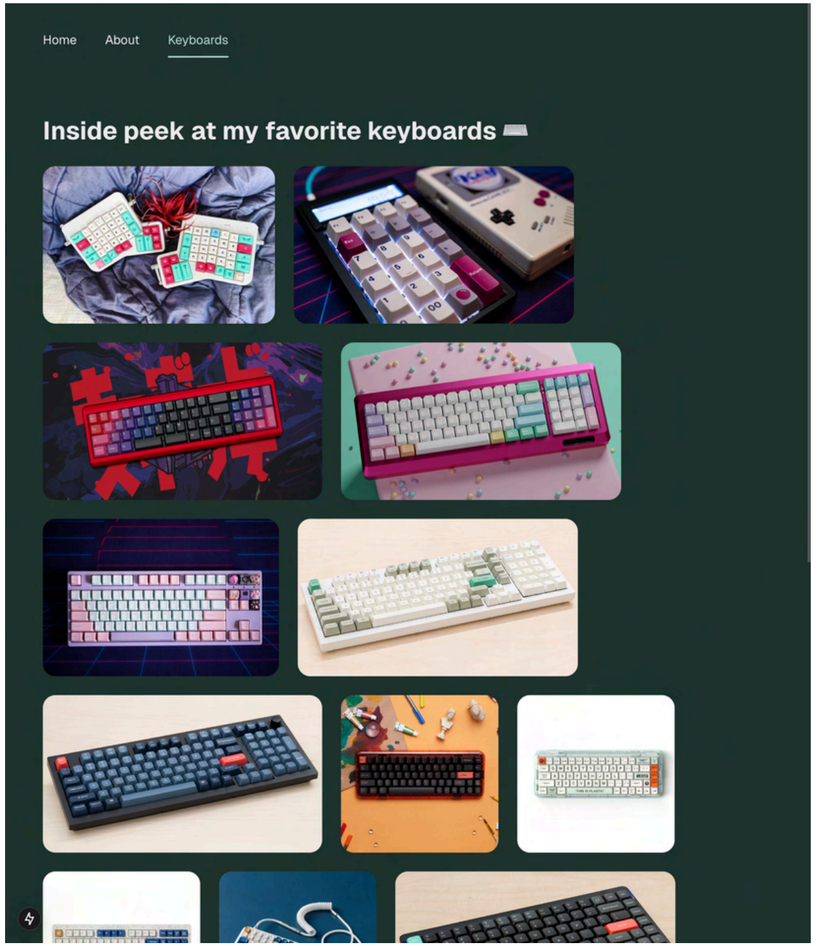


Automated visual regression testing

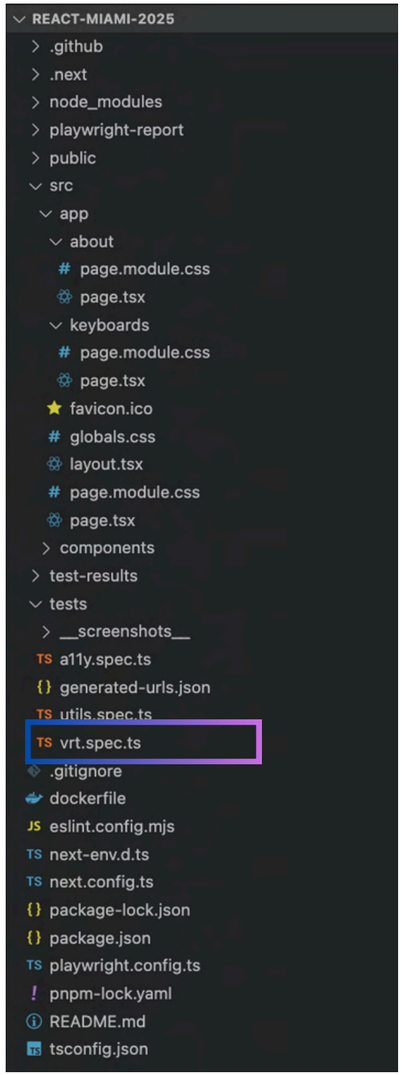
Our demo NextJS app







Automated visual regression testing



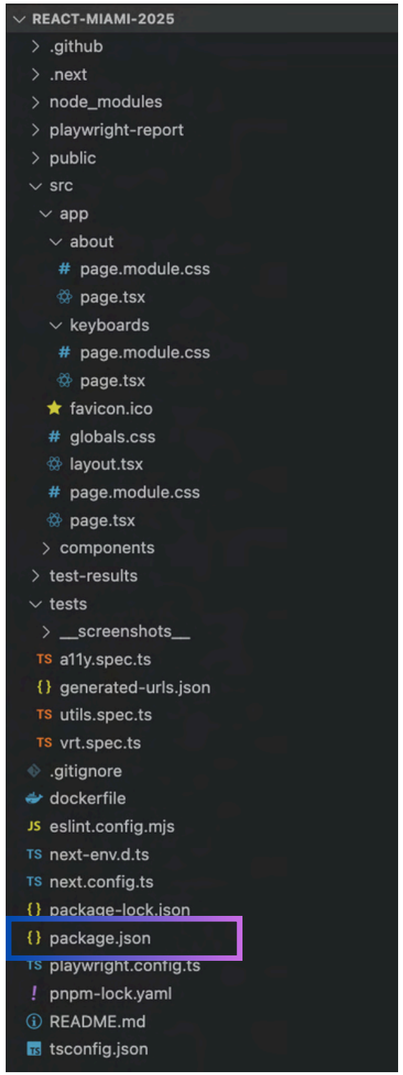

```ts
import { expect, test } from '@playwright/test';
import fs from 'fs';
import path from 'path';

/**
 * 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

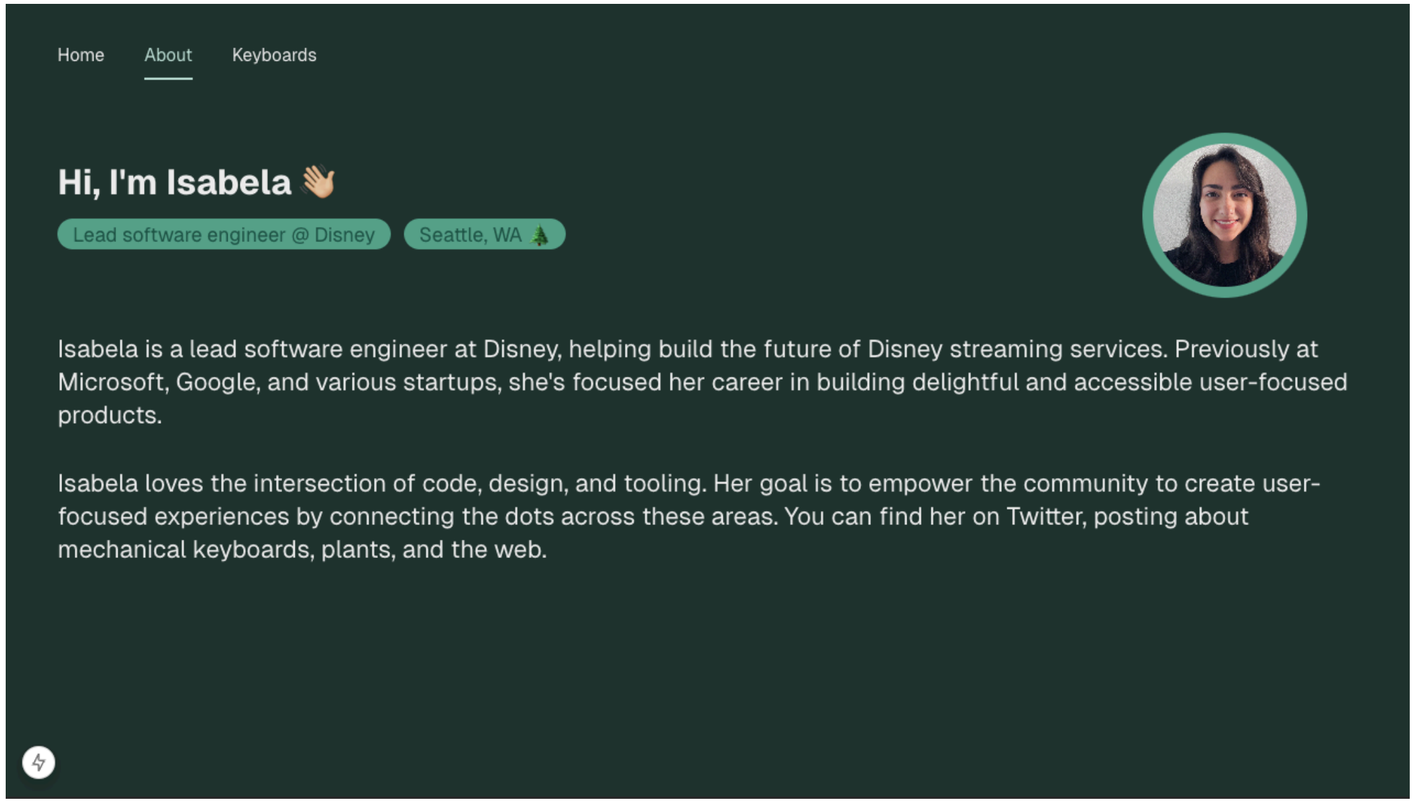
Automated visual regression testing

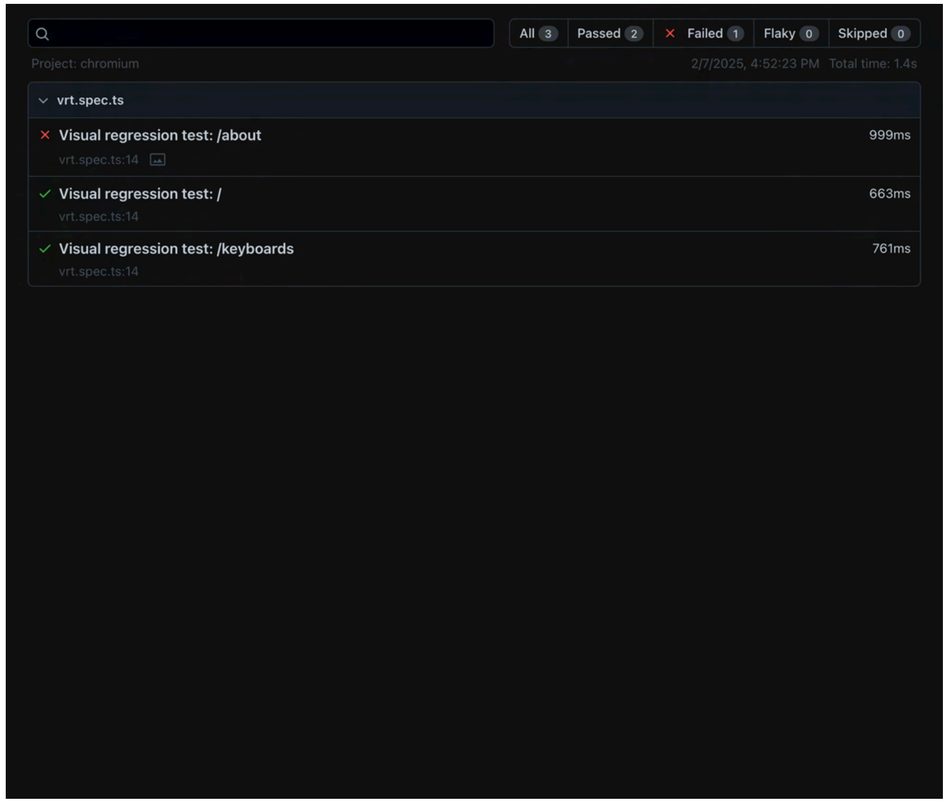



```
{
  "name": "react-miami-2025",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "playwright:generate-urls": "playwright test utils",
    "playwright:a11y:local": "pnpm playwright:generate-urls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

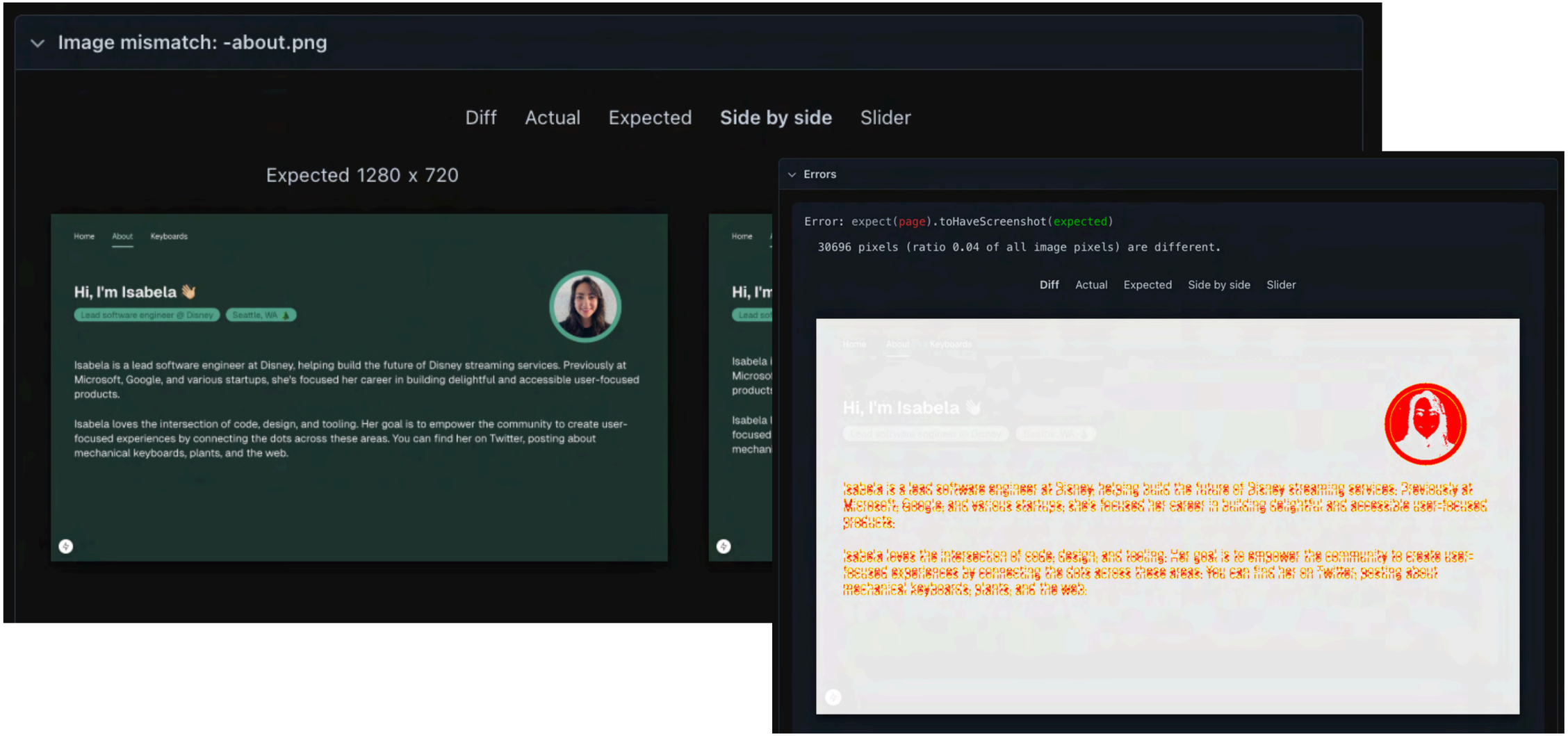
Run generation command, run visual regression tests, open report

Automated visual regression testing





Automated visual regression testing



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

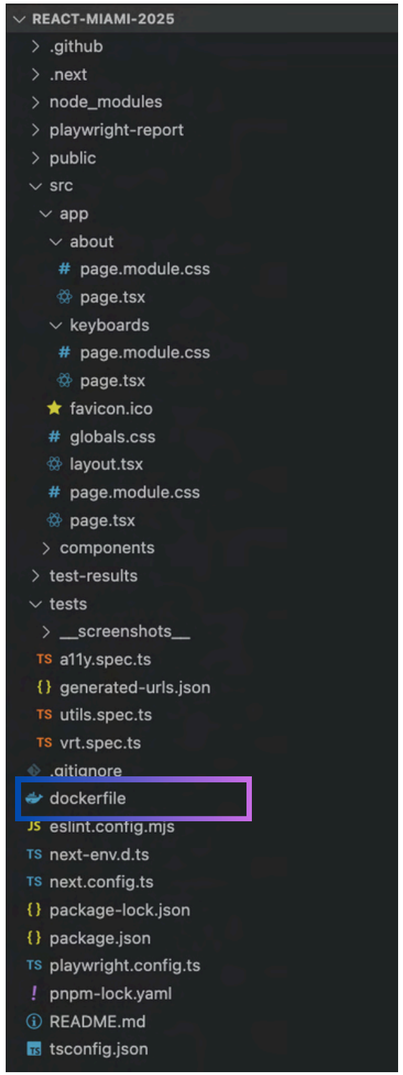
Causes

- **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




```dockerfile
# Get the latest version of Playwright
FROM mcr.microsoft.com/playwright:v1.50.0-noble

# Set the work directory for the application
WORKDIR /app

# Set the environment path to node_modules/.bin
ENV PATH /app/node_modules/.bin:$PATH

# Copy package.json and package-lock.json
COPY package*.json ./

# Copy test-related files
COPY playwright.config.ts ./
COPY tests/ ./tests/

# Copy the built NextJs site
COPY .next ./.next
COPY public ./public

# Install dependencies
RUN npm install

ENV CI=true

# Make sure the CMD uses the TEST_TYPE cmd line variable
CMD ["sh", "-c", "npx playwright test $TEST_TYPE"]
```

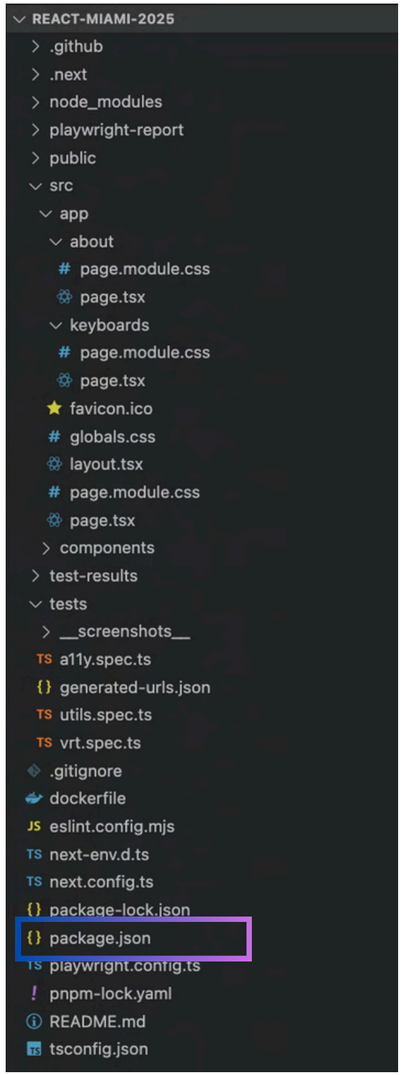
Use the Playwright image

Copy test-related files

Copy site prod build

Run the appropriate tests

Adding Docker to the workflow



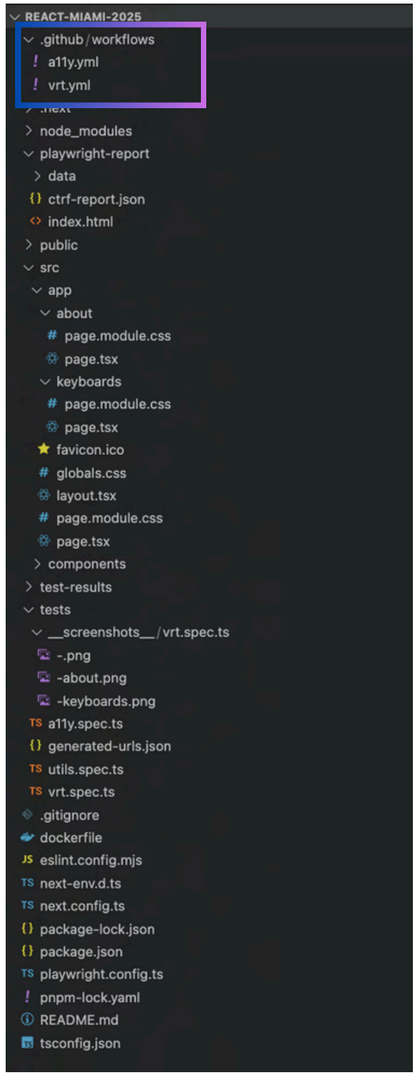

```
{
  "name": "react-miami-2025",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "playwright:generate-urls": "playwright test utils",
    "playwright:a11y:local": "pnpm playwright:generate-urls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



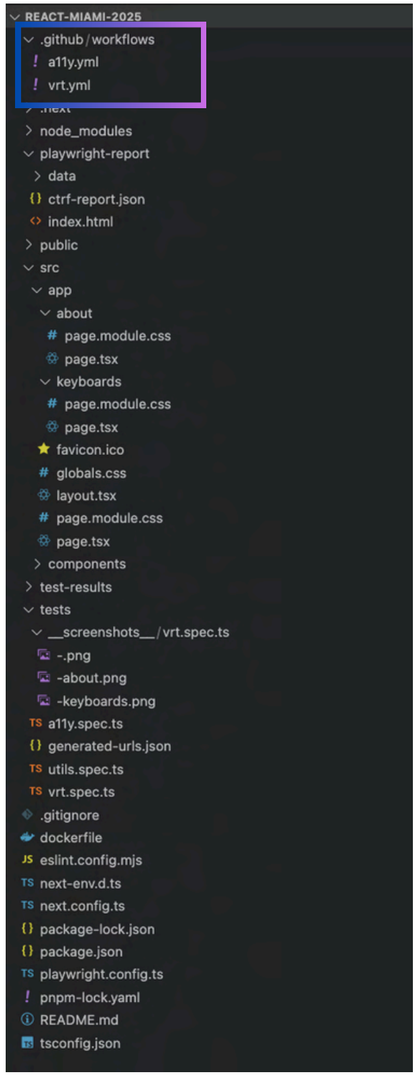

```yaml
name: Run visual regression tests

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




```yaml
# 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
```

```yaml
# 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/
```

```yaml
# 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()
```

```yaml
# 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
```

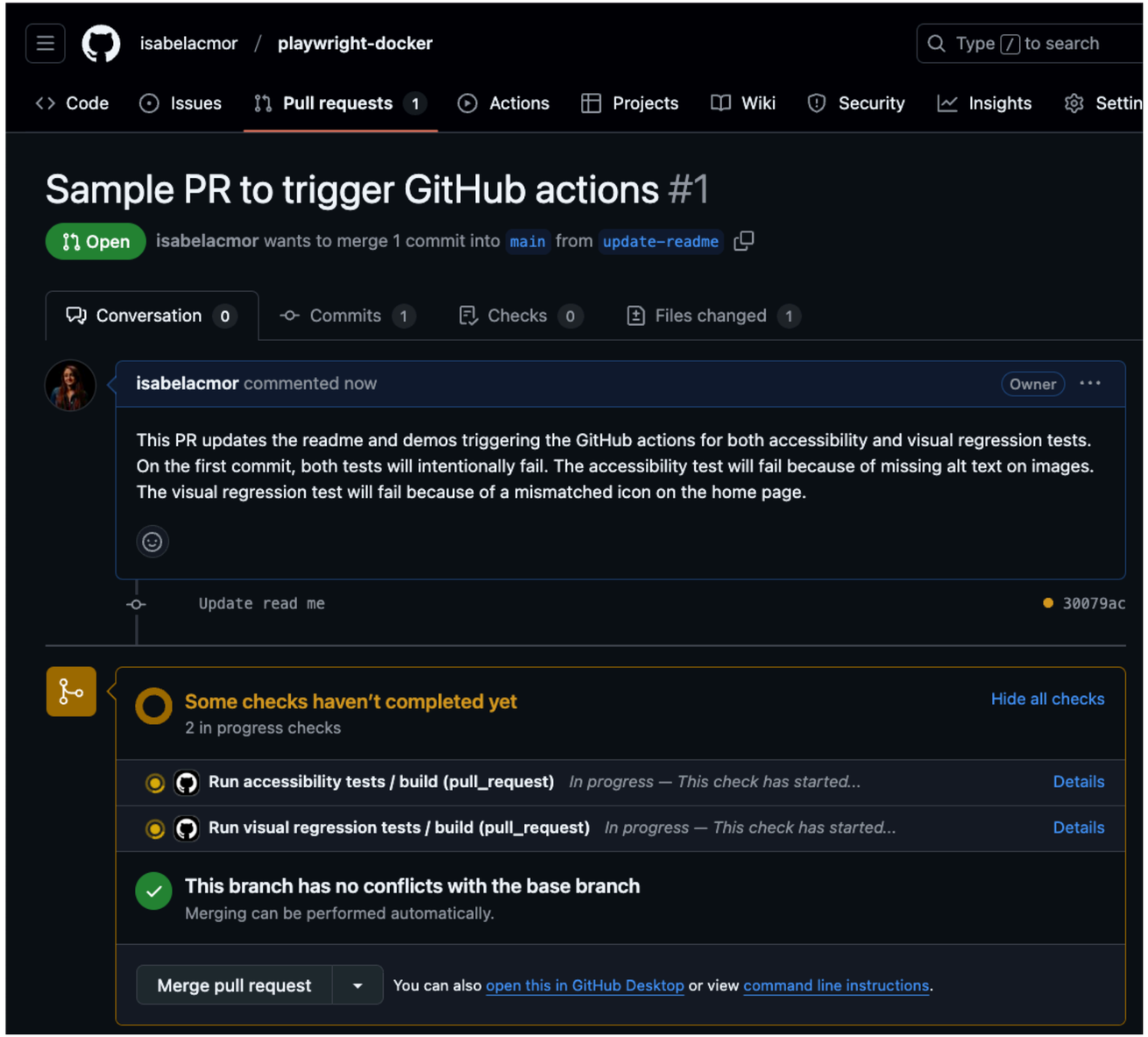
Explicitly fail the job if the tests fail, but still copy the test report from Docker

Upload the test report as an artifact in our job run

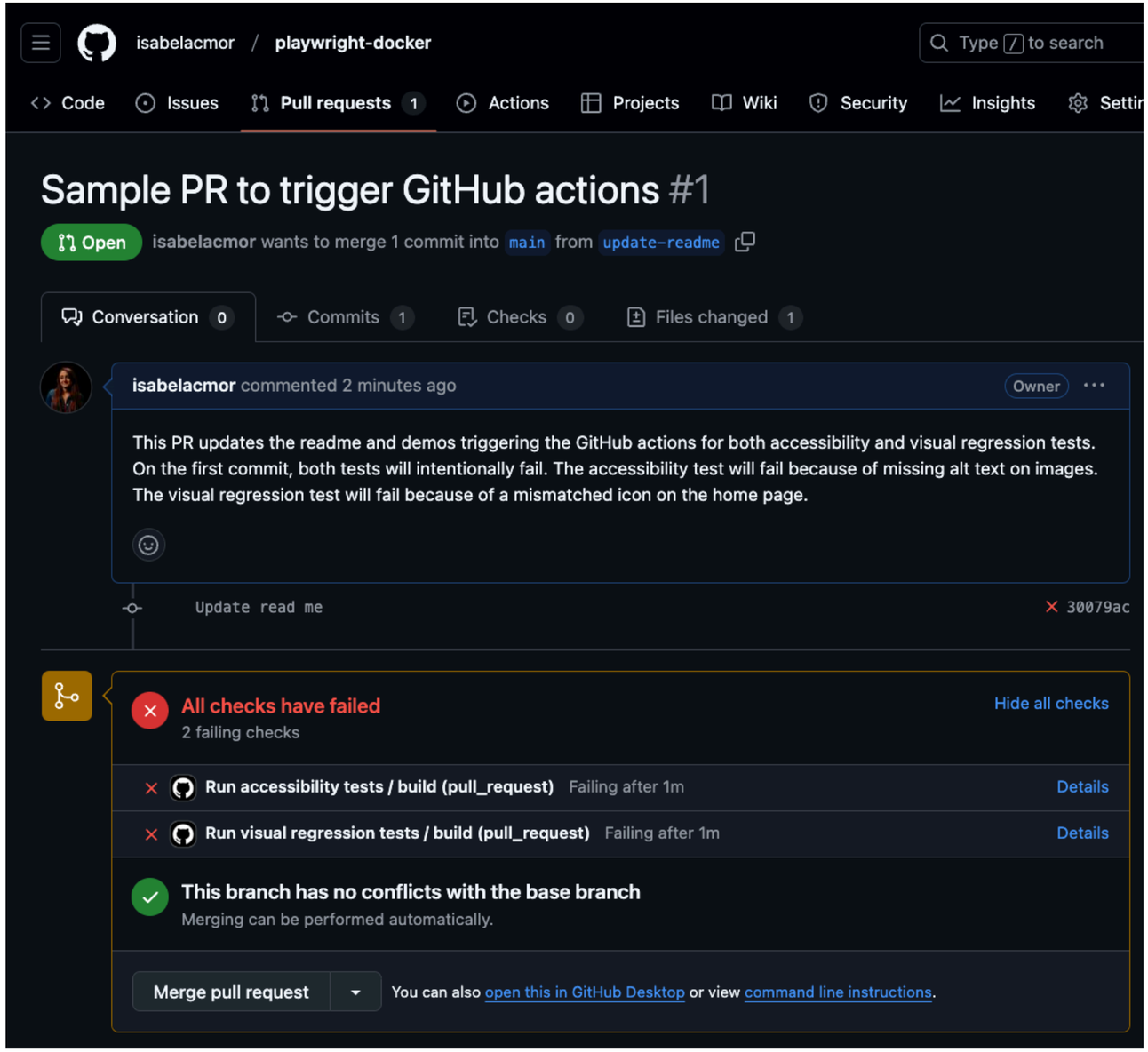
Always publish the test report artifact

Docker cleanup work

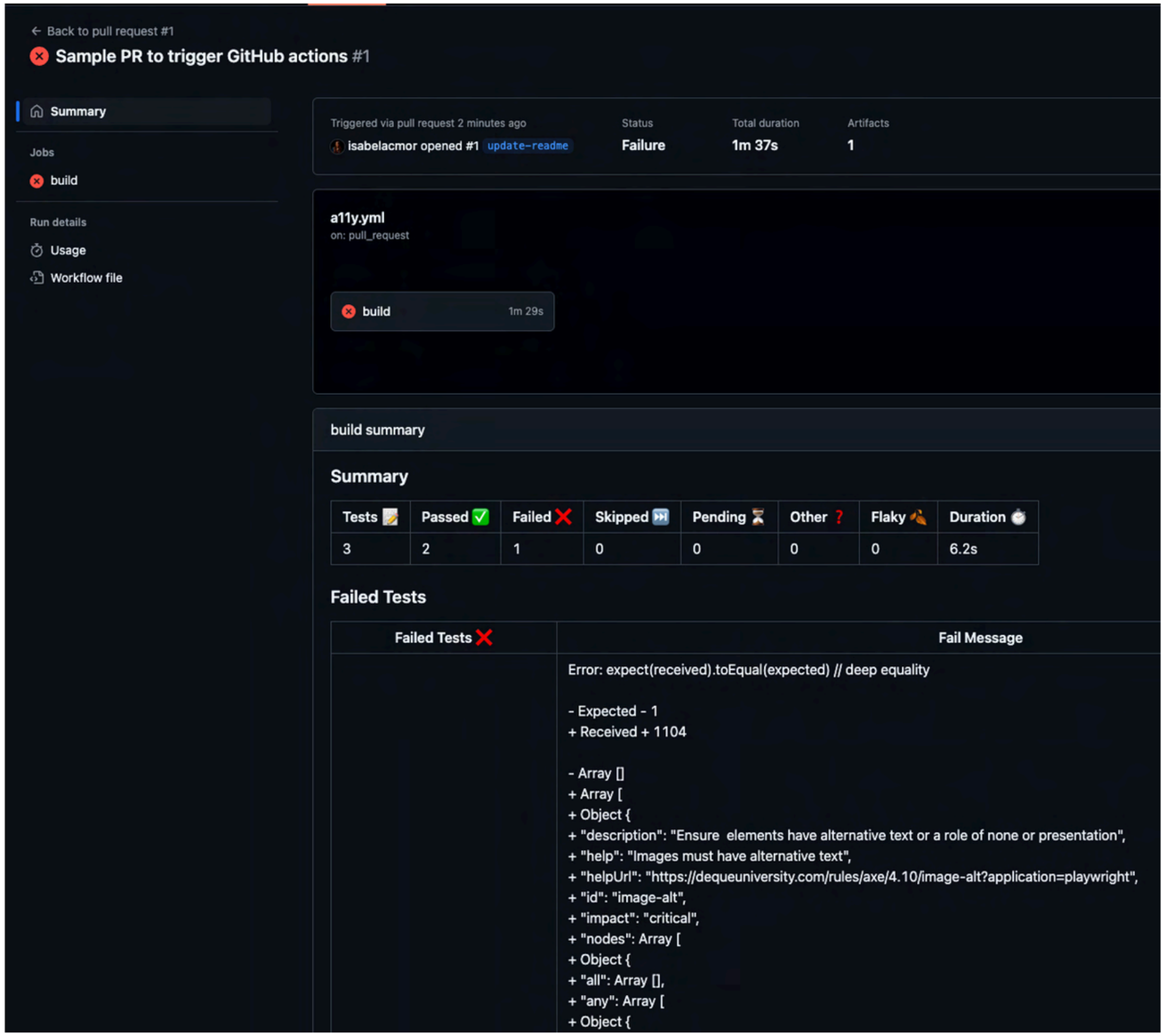
Putting it all together in GitHub actions

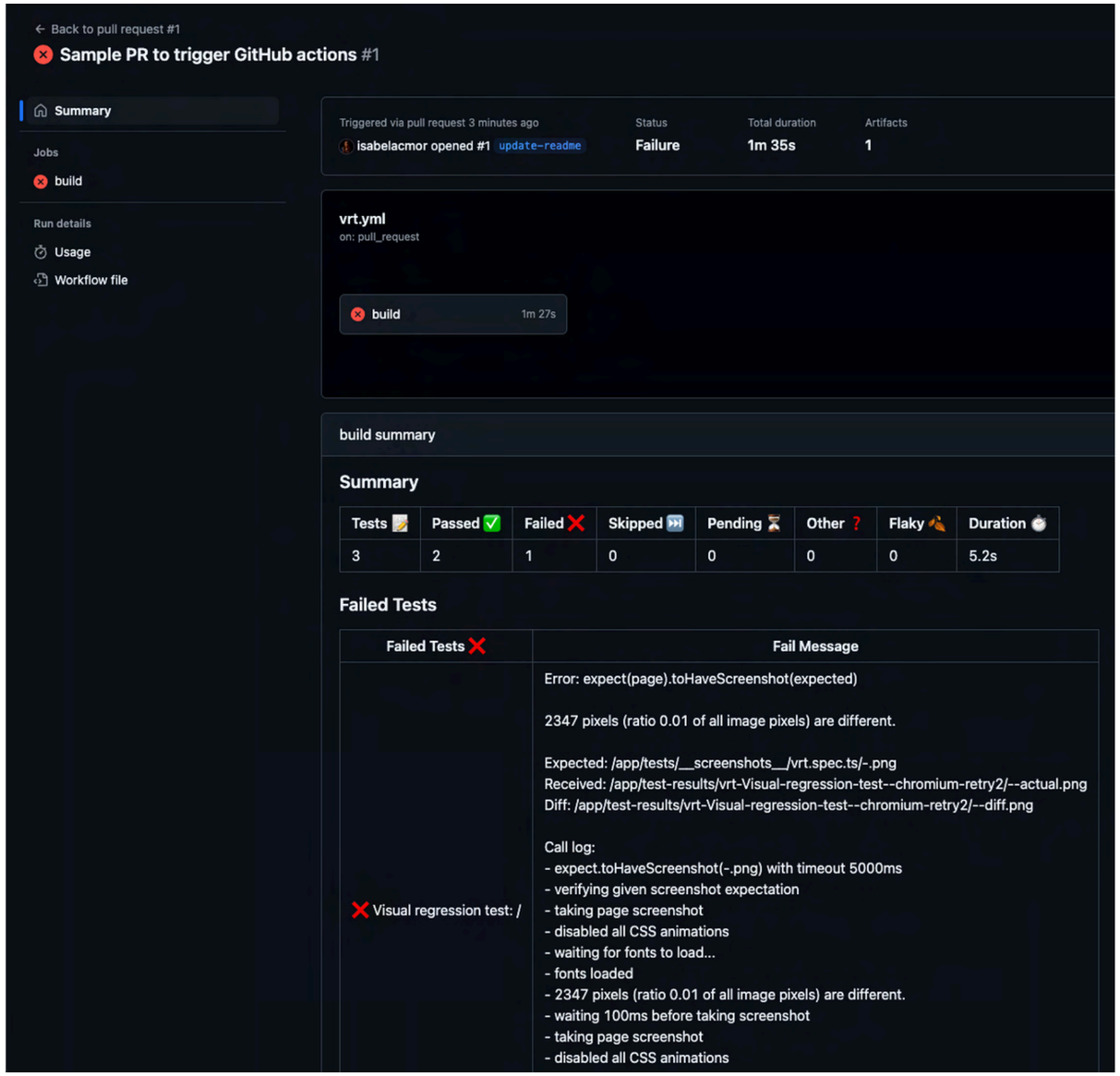


Putting it all together in GitHub actions

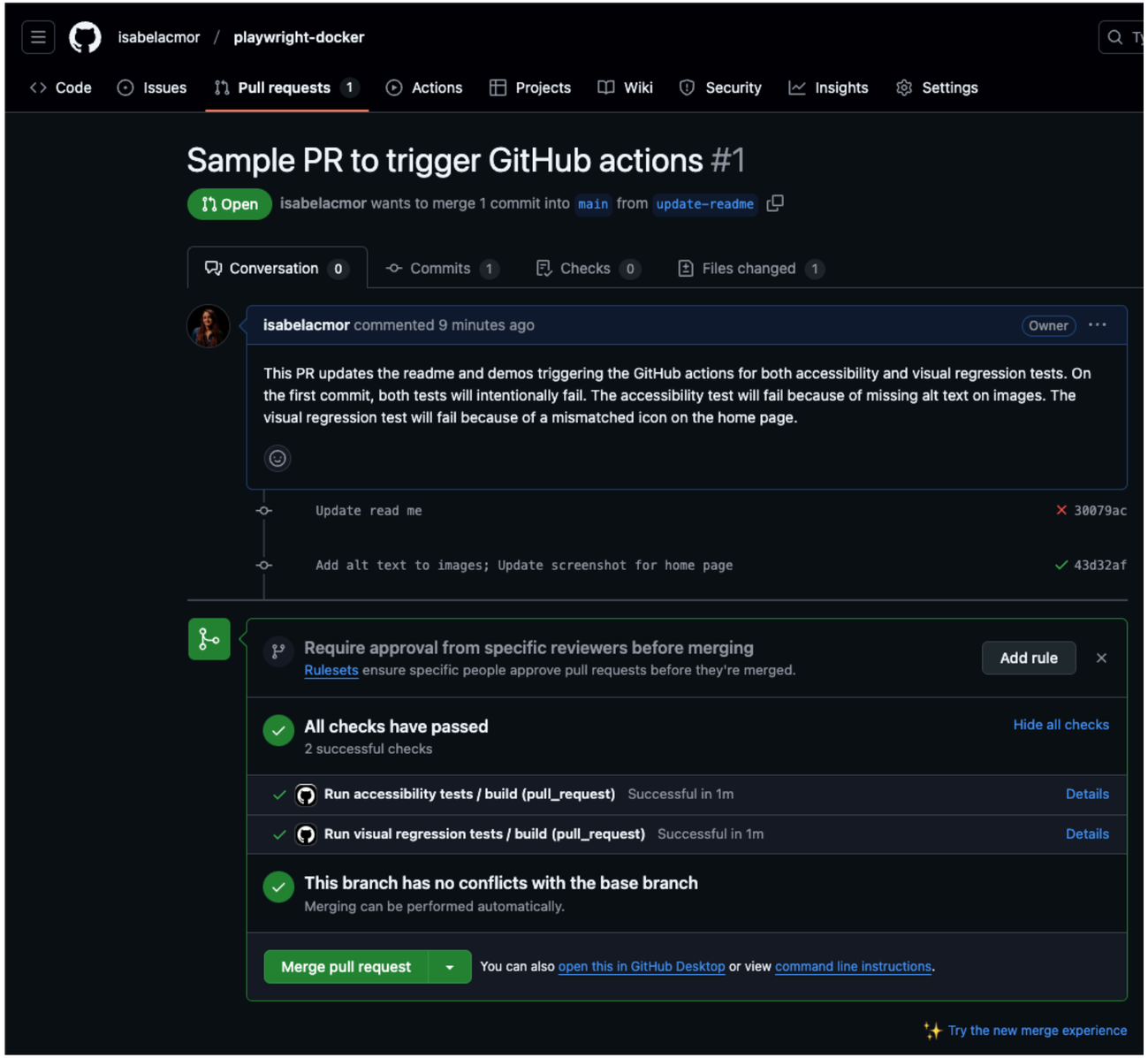


Putting it all together in GitHub actions

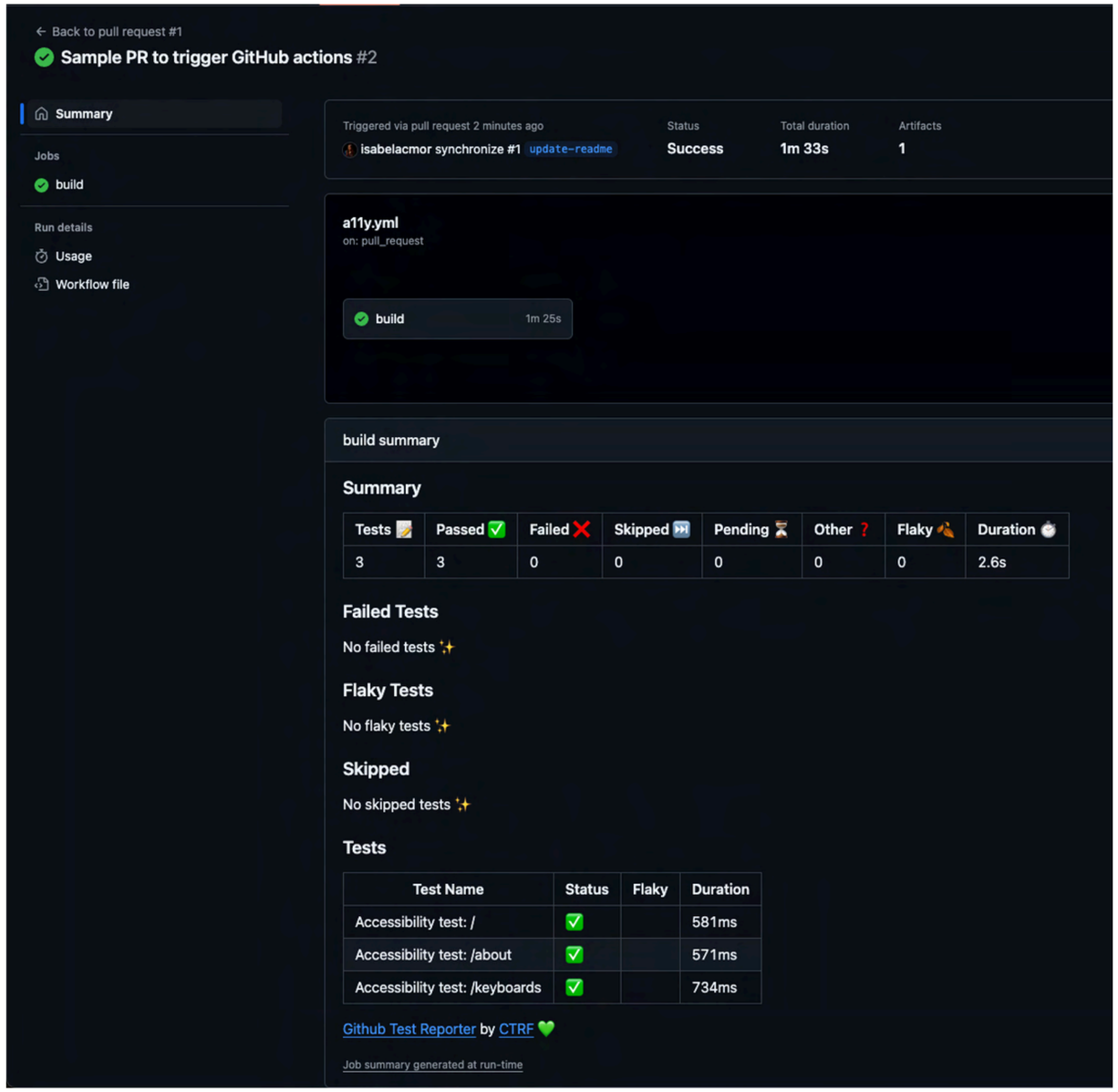


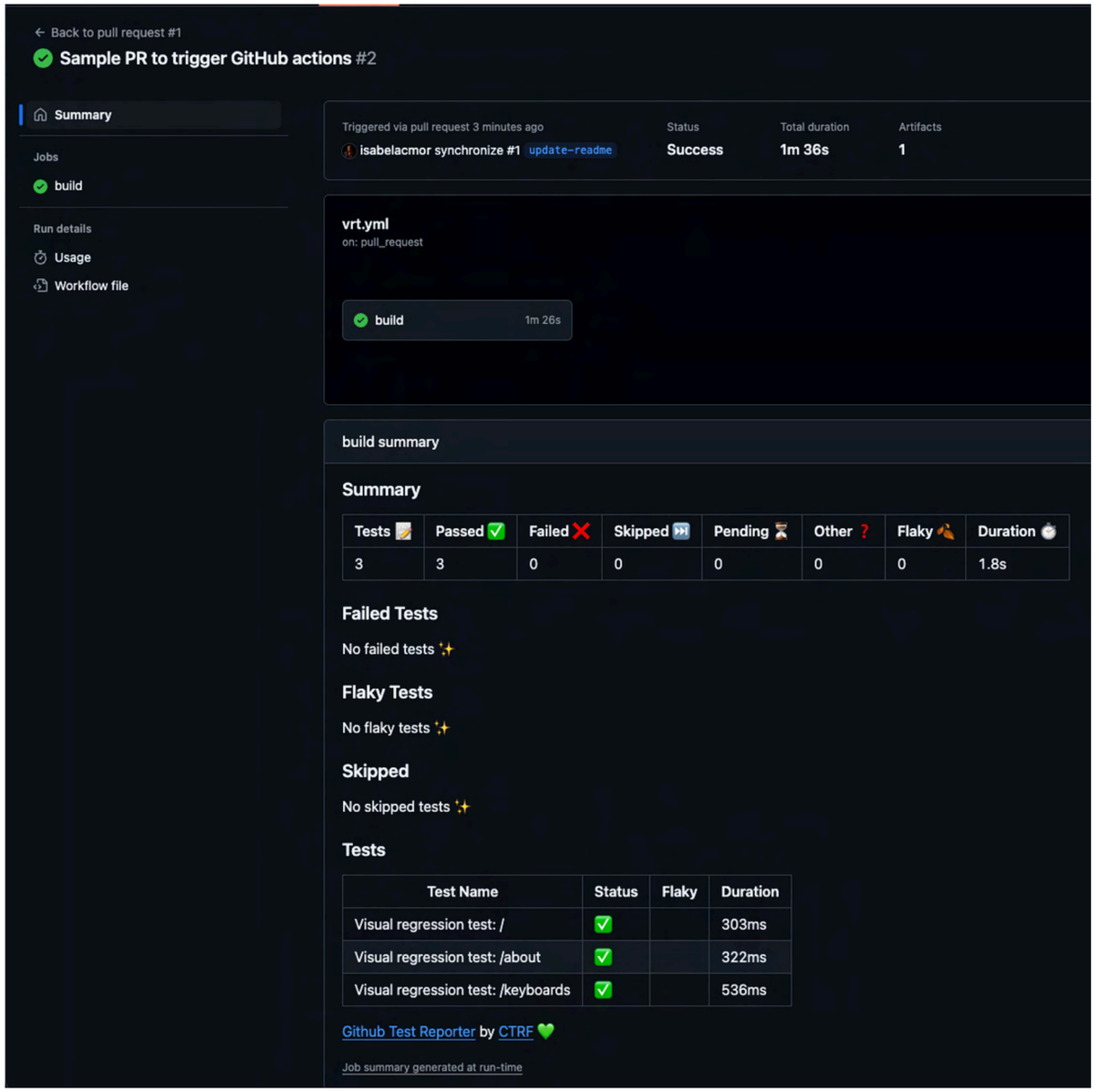


Putting it all together in GitHub actions



Putting it all together in GitHub actions





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