Accessible Web Testing with Cypress and Axe Core

Vitaly Skadorva
February 26, 2025

Agenda

- **Speaker:** Vitaly Skadorva
- **Role:** QA Automation Specialist at Intact Financial Corporation
- **Linkedin**: https://www.linkedin.com/in/vitalyskadorva/
- **My Book**: https://a.co/d/gGls5fn

Why Accessibility Matters

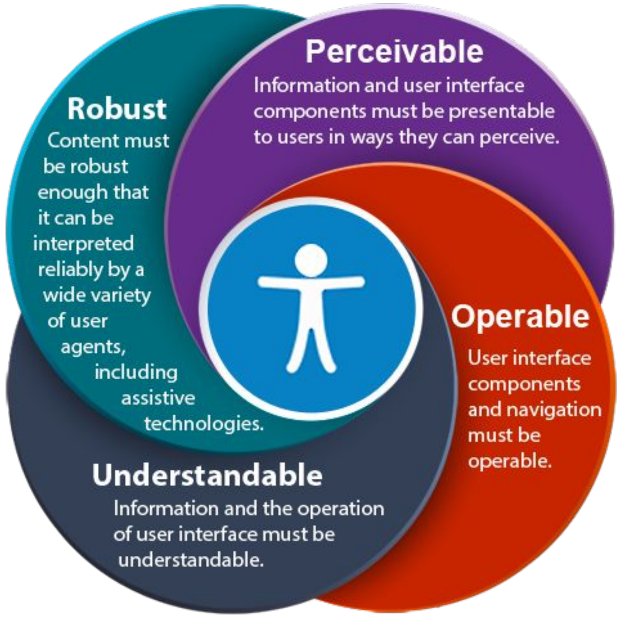
Statistics on Disability

- About 15% of people live with disabilities worldwide.
- The European Accessibility Act starts June 28, 2025.
- Websites in the EU must meet WCAG 2.2 Level AA.
- The ADA ensures equal access for all in the U.S.
- Accessibility fosters inclusive experiences for every user.
- Ignoring accessibility limits access to crucial services.

Benefits of Accessibility

- Compliance with WCAG and Section 508 prevents legal issues.
- Accessibility enhances user experience, increasing customer loyalty.
- Accessible design improves product quality for all users.

Accessibility Standards & Guidelines


Perceivable
Information and user interface components must be presentable to users in ways they can perceive.
Robust
Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.
Operable
User interface components and navigation must be operable.
Understandable
Information and the operation of user interface must be understandable.

WCAG's POUR Principles

The Web Content Accessibility Guidelines (WCAG) are structured around four principles: Perceivable, Operable, Understandable, and Robust. These principles ensure that content is accessible to all users.

Common Accessibility Issues

Frequent issues include missing alt text for images, low color contrast making text hard to read, and mis-labeled forms that confuse users. Identifying these issues is crucial for compliance.

Testing with Cypress & Axe

Cypress and Axe can be used to test for these guidelines effectively. Tests can identify violations related to alt text, contrast ratios, and form labeling, enabling early detection and remediation.

Web Accessibility Challenges

High Inaccessibility Rates

Over 96% of top million sites have serious accessibility issues. (Source: WebAIM)

Non-Compliance with WCAG 2.1

A staggering 98% of sites fail Web Content Accessibility Guidelines (WCAG) 2.1. (Source: AccessiBe)

Prevalent Accessibility Errors

Average of 56.8 accessibility errors occurs per web page. (Source: WebAIM)

Lack of Progress for Users

50% of individuals with disabilities see no web accessibility improvements. (Source: WebAIM)

Missing Alt Text in Images

22.1% of homepage images lack alt text for screen readers. (Source: WebAIM)

Fortune 100 Accessibility Audit

WCAG 2.1 audit found 815,600 issues on Fortune 100 websites. (Source: Ovum)

Overview

Introducing the Tools

Cypress Overview

Cypress is a JavaScript-based end-to-end testing framework designed for modern web applications. It provides fast, reliable testing with an intuitive interface.

Axe Core Introduction

Axe Core is a powerful accessibility testing engine developed by Deque Systems. It identifies accessibility violations based on WCAG guidelines.

cypress-axe Plugin

The cypress-axe plugin integrates Axe Core with Cypress, allowing developers to perform automated accessibility checks within their end-to-end tests.

Why Use Them Together?

Combining Cypress with Axe Core enables automated accessibility checks within your test suite, ensuring compliance with WCAG and other accessibility standards.

Getting Started

Installation & Setup for Cypress Accessibility Testing

Install Necessary Packages

Use npm to install axe-core, cypress, and cypress-axe:

`***npm install --save-dev axe-core cypress cypress-axe***`.

Import Commands

Include the commands in your `*cypress/support/e2e.js*` file:

`***import 'cypress-axe'***`.

Inject axe-core Runtime

Use `***cy.injectAxe()***` after `***cy.visit()***` to inject axe-core before running `***checkA11y***`.

Write Accessibility Tests

Start testing with:

`***it('Has no detectable a11y violations on load', () => {***

    cy.checkA11y()

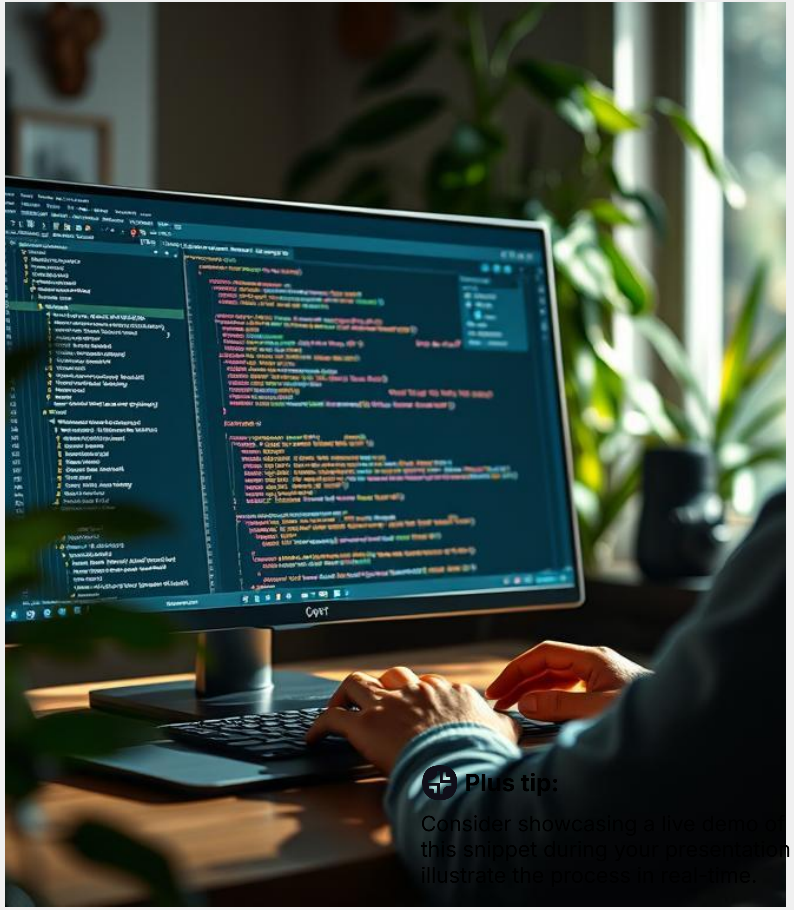
})`.

Usage

Basic Cypress + Axe Usage

Testing for Accessibility

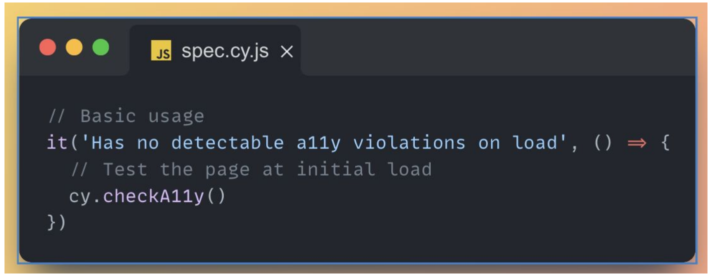
- Use `***cy.injectAxe()***` to load the Axe accessibility testing engine into the DOM.
- The method `***cy.checkA11y()***` scans the page for accessibility violations using default rules.
- This approach allows developers to ensure that the initial page load is free from detectable accessibility issues.



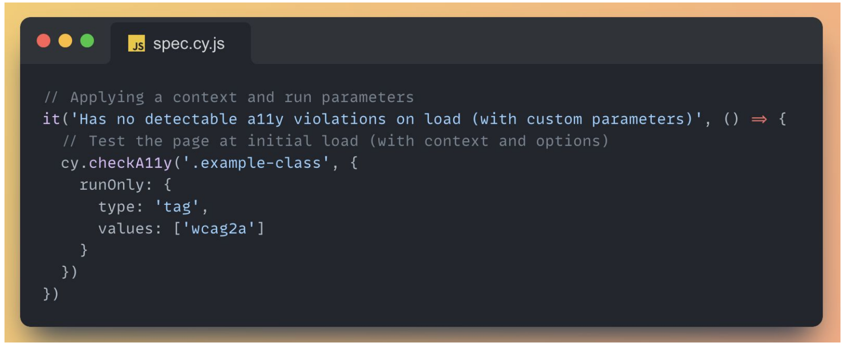
Plus tip:

Consider showcasing a live demo of this snippet during your presentation to illustrate the process in real-time.

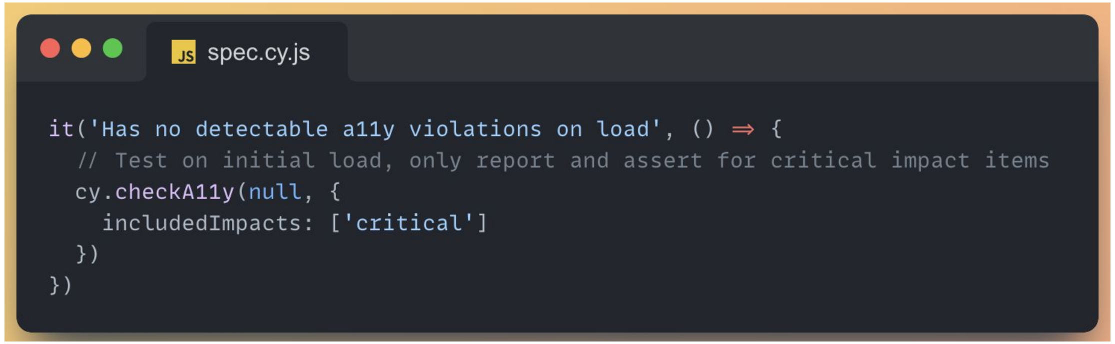
Basic Usage


spec.cy.js

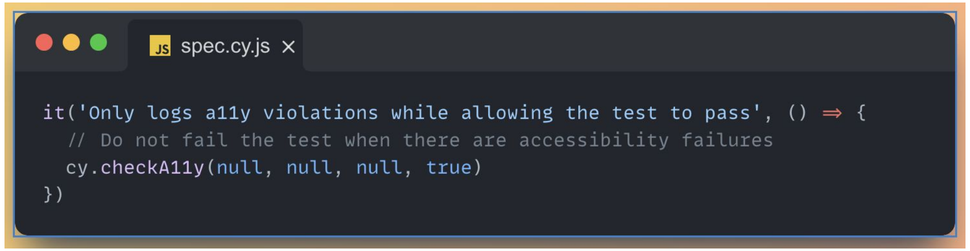
```js
// Basic usage
it('Has no detectable a11y violations on load', () => {
  // Test the page at initial load
  cy.checkA11y()
})
```


spec.cy.js

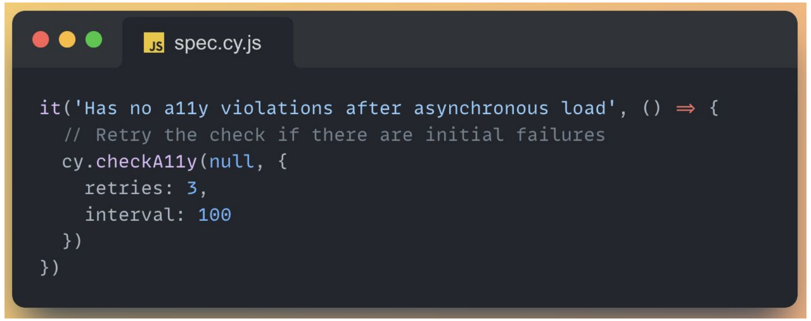
```js
// Applying a context and run parameters
it('Has no detectable a11y violations on load (with custom parameters)', () => {
  // Test the page at initial load (with context and options)
  cy.checkA11y('.example-class', {
    runOnly: {
      type: 'tag',
      values: ['wcag2a']
    }
  })
})
```


spec.cy.js

```js
it('Has no detectable a11y violations on load', () => {
  // Test on initial load, only report and assert for critical impact items
  cy.checkA11y(null, {
    includedImpacts: ['critical']
  })
})
```

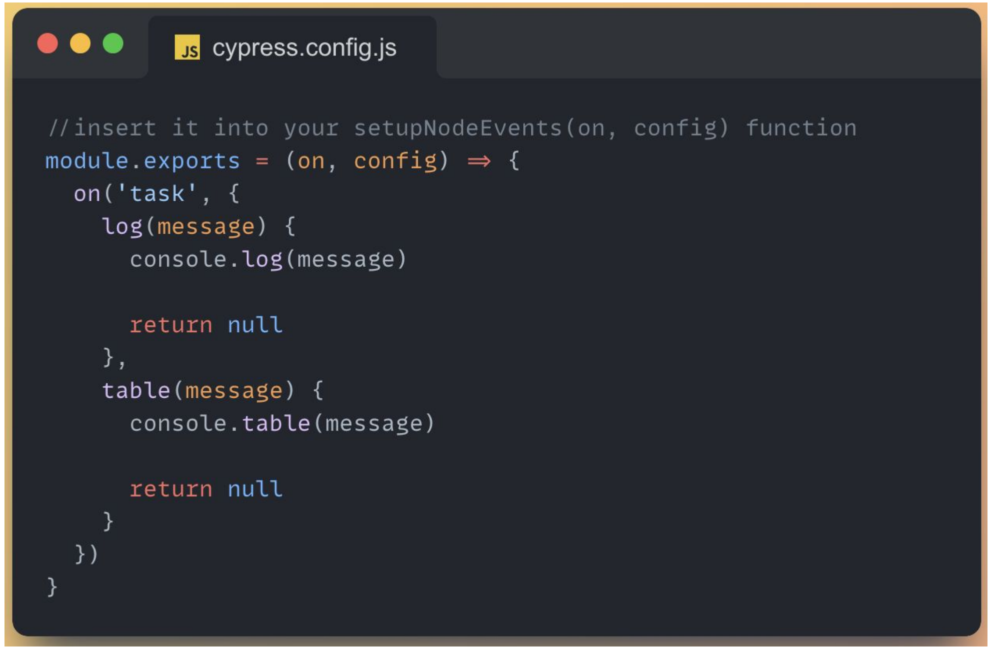

spec.cy.js

```js
it('Only logs a11y violations while allowing the test to pass', () => {
  // Do not fail the test when there are accessibility failures
  cy.checkA11y(null, null, null, true)
})
```

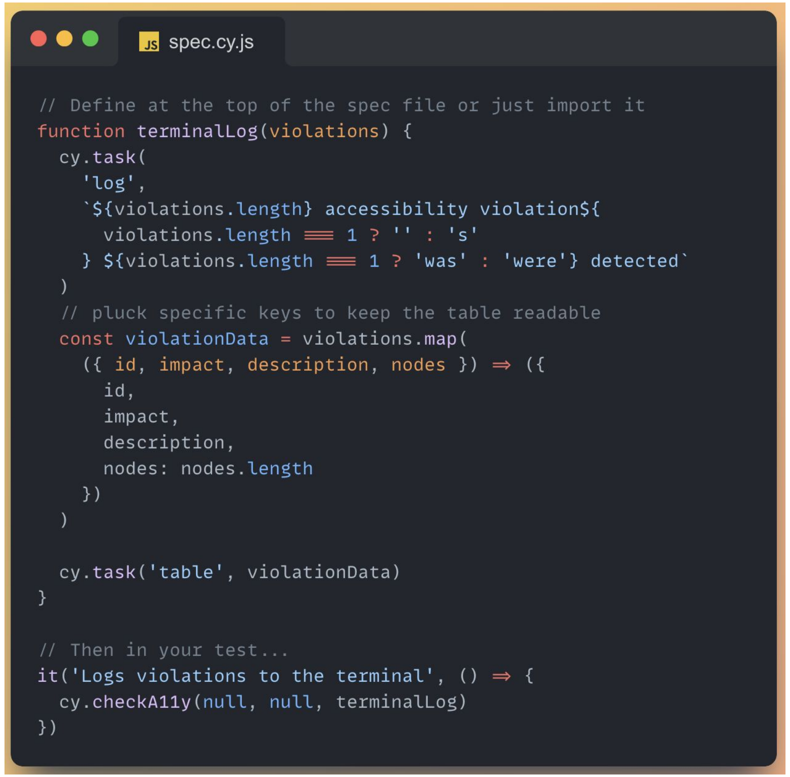

spec.cy.js

```js
it('Has no a11y violations after asynchronous load', () => {
  // Retry the check if there are initial failures
  cy.checkA11y(null, {
    retries: 3,
    interval: 100
  })
})
```

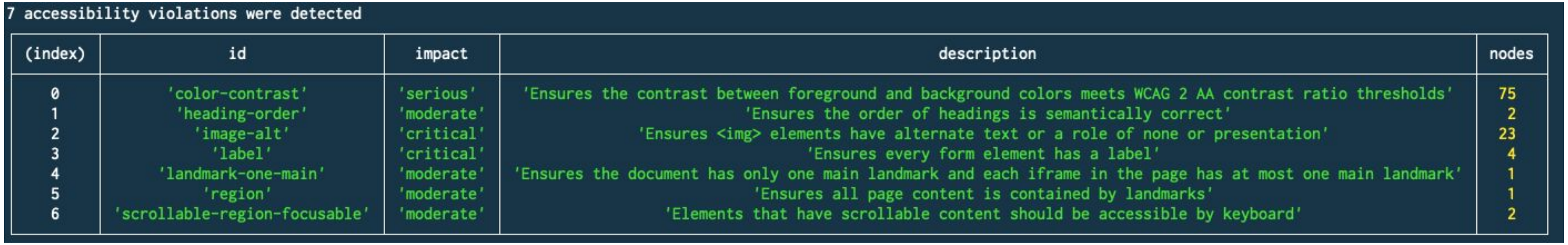
Custom Logging



```js
//insert it into your setupNodeEvents(on, config) function
module.exports = (on, config) => {
  on('task', {
    log(message) {
      console.log(message)

      return null
    },
    table(message) {
      console.table(message)

      return null
    }
  })
}
```



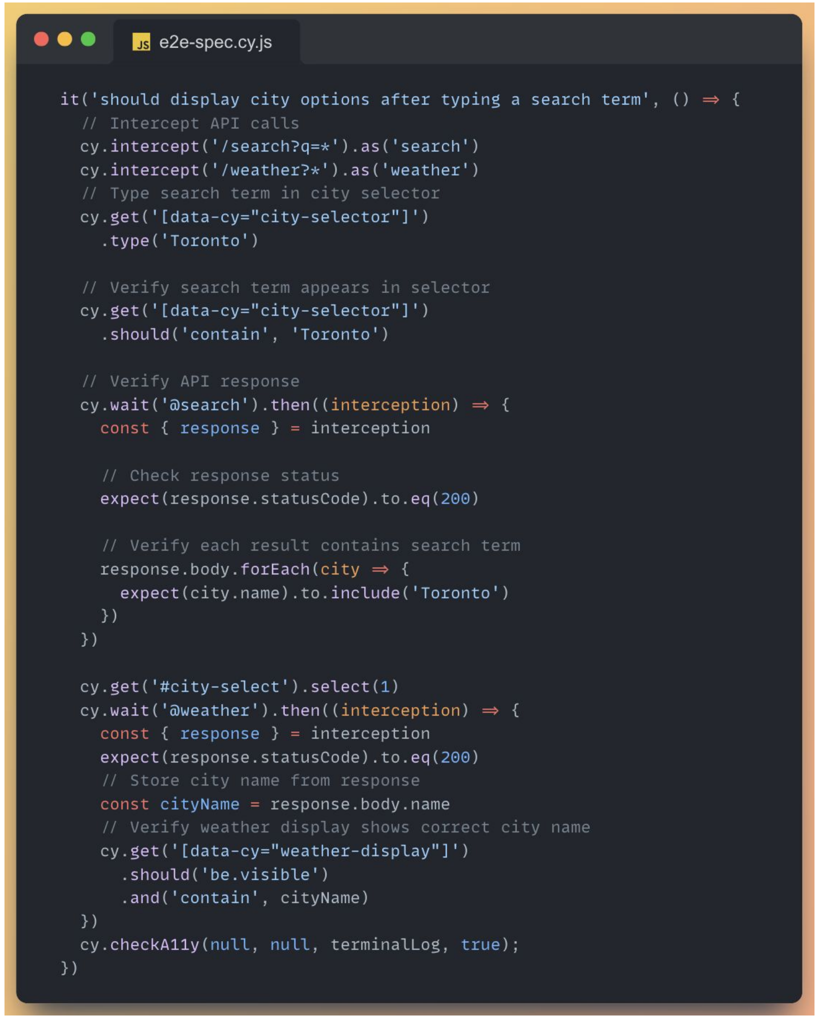
```js
// Define at the top of the spec file or just import it
function terminalLog(violations) {
  cy.task(
    'log',
    `${violations.length} accessibility violation${
      violations.length === 1 ? '' : 's'
    } ${violations.length === 1 ? 'was' : 'were'} detected`
  )
  // pluck specific keys to keep the table readable
  const violationData = violations.map(
    ({ id, impact, description, nodes }) => ({
      id,
      impact,
      description,
      nodes: nodes.length
    })
  )

  cy.task('table', violationData)
}

// Then in your test...
it('Logs violations to the terminal', () => {
  cy.checkA11y(null, null, terminalLog)
})
```



```
7 accessibility violations were detected
┌─────────┬────────────────────────────┬────────────┬──────────────────────────────────────────────────────────────────────────────────────────┬───────┐
│ (index) │             id             │   impact   │                                       description                                        │ nodes │
├─────────┼────────────────────────────┼────────────┼──────────────────────────────────────────────────────────────────────────────────────────┼───────┤
│    0    │      'color-contrast'      │ 'serious'  │ 'Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds' │  75   │
│    1    │      'heading-order'       │ 'moderate' │                  'Ensures the order of headings is semantically correct'                 │   2   │
│    2    │        'image-alt'         │ 'critical' │        'Ensures <img> elements have alternate text or a role of none or presentation'     │  23   │
│    3    │          'label'           │ 'critical' │                         'Ensures every form element has a label'                         │   4   │
│    4    │     'landmark-one-main'    │ 'moderate' │ 'Ensures the document has only one main landmark and each iframe in the page has at most one main landmark' │   1   │
│    5    │          'region'          │ 'moderate' │                   'Ensures all page content is contained by landmarks'                   │   1   │
│    6    │ 'scrollable-region-focusable' │ 'moderate' │          'Elements that have scrollable content should be accessible by keyboard'        │   2   │
└─────────┴────────────────────────────┴────────────┴──────────────────────────────────────────────────────────────────────────────────────────┴───────┘
```

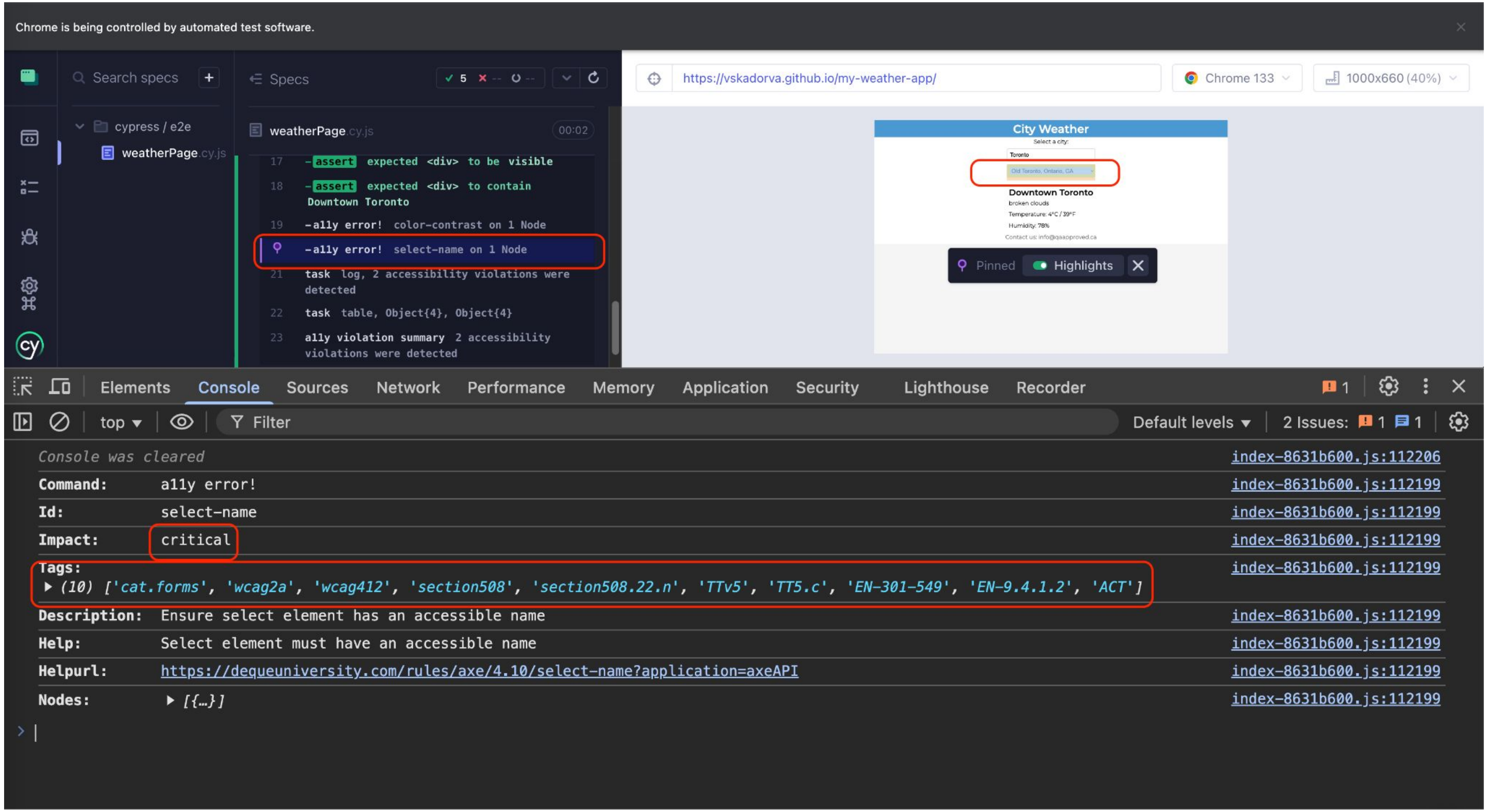
Example #1 – E2E Test

- This **test suite** verifies **core functionality** of the **App**.
- It starts by **injecting Axe** to find accessibility issues.
- The suite checks **critical UI components** for presence and labels.
- It **intercepts API calls** to validate accurate **search** results.
- It ensures the UI updates correctly based on API response (no mocks left).
- Finally, an **accessibility check** ensures usability for all users.


e2e-spec.cy.js

```
it('should display city options after typing a search term', () ⇒ {
  // Intercept API calls
  cy.intercept('/search?q=*').as('search')
  cy.intercept('/weather?*').as('weather')
  // Type search term in city selector
  cy.get('[data-cy="city-selector"]')
    .type('Toronto')

  // Verify search term appears in selector
  cy.get('[data-cy="city-selector"]')
    .should('contain', 'Toronto')

  // Verify API response
  cy.wait('@search').then((interception) ⇒ {
    const { response } = interception

    // Check response status
    expect(response.statusCode).to.eq(200)

    // Verify each result contains search term
    response.body.forEach(city ⇒ {
      expect(city.name).to.include('Toronto')
    })
  })

  cy.get('#city-select').select(1)
  cy.wait('@weather').then((interception) ⇒ {
    const { response } = interception
    expect(response.statusCode).to.eq(200)
    // Store city name from response
    const cityName = response.body.name
    // Verify weather display shows correct city name
    cy.get('[data-cy="weather-display"]')
      .should('be.visible')
      .and('contain', cityName)
  })
  cy.checkA11y(null, null, terminalLog, true);
})
```

Example #1 - E2E Test


Chrome is being controlled by automated test software.

https://vskadorva.github.io/my-weather-app/

City Weather
Select a city:
Toronto
Old Toronto, Ontario, CA
Downtown Toronto
broken clouds
Temperature: 4°C / 39°F
Humidity: 78%
Contact us: info@qaapproved.ca

weatherPage.cy.js

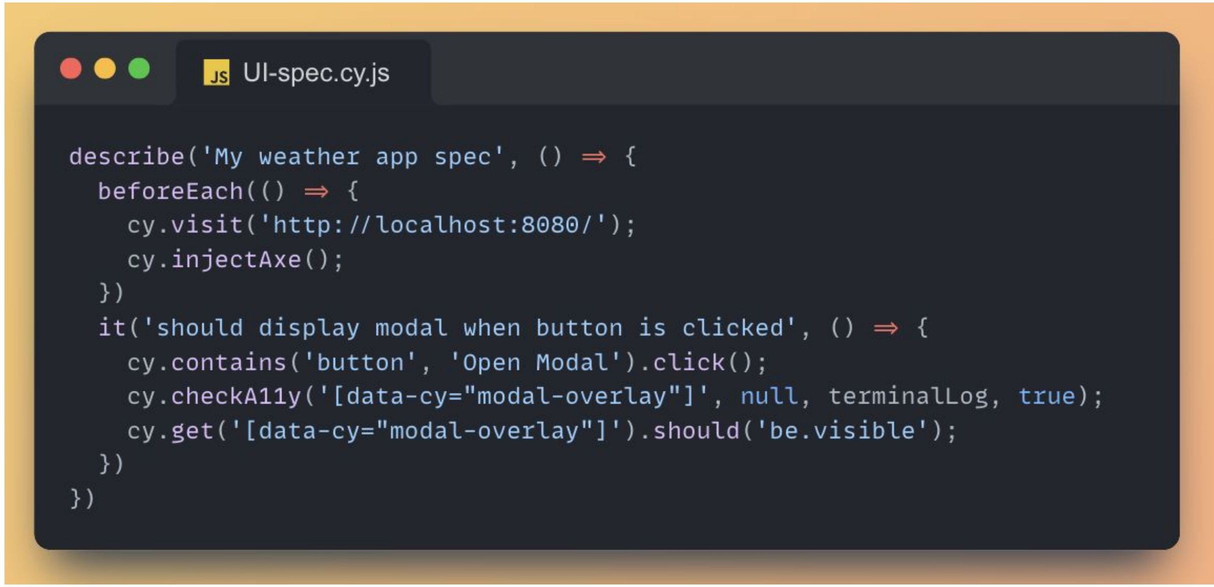
```
-assert  expected <div> to be visible
-assert  expected <div> to contain
 Downtown Toronto
-a11y error!  color-contrast on 1 Node
-a11y error!  select-name on 1 Node
task  log, 2 accessibility violations were
detected
task  table, Object{4}, Object{4}
a11y violation summary  2 accessibility
violations were detected
```

```
Console was cleared                                  index-8631b600.js:112206
Command:     a11y error!                             index-8631b600.js:112199
Id:          select-name                             index-8631b600.js:112199
Impact:      critical                                index-8631b600.js:112199
Tags:                                                index-8631b600.js:112199
▶ (10) ['cat.forms', 'wcag2a', 'wcag412', 'section508', 'section508.22.n', 'TTv5', 'TT5.c', 'EN-301-549', 'EN-9.4.1.2', 'ACT']
Description: Ensure select element has an accessible name    index-8631b600.js:112199
Help:        Select element must have an accessible name     index-8631b600.js:112199
Helpurl:     https://dequeuniversity.com/rules/axe/4.10/select-name?application=axeAPI    index-8631b600.js:112199
Nodes:       ▶ [{…}]                                 index-8631b600.js:112199
>
```

DEMO

Example #2 – UI Test (Visual page elements)

- Accessibility testing is essential for forms and pop-ups.
- Modals can complicate user interactions, requiring thorough testing.
- Dynamic content may cause accessibility issues when opened.
- Test accessibility before and after opening dynamic elements.
- Cypress combined with Axe enables detailed accessibility testing.
- This applies to both static and dynamic content elements.

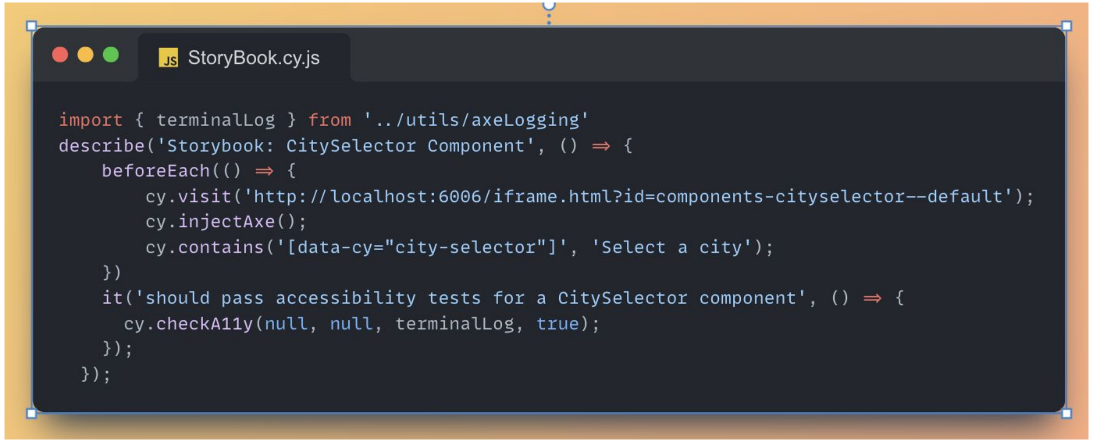


```
describe('My weather app spec', () => {
  beforeEach(() => {
    cy.visit('http://localhost:8080/');
    cy.injectAxe();
  })
  it('should display modal when button is clicked', () => {
    cy.contains('button', 'Open Modal').click();
    cy.checkA11y('[data-cy="modal-overlay"]', null, terminalLog, true);
    cy.get('[data-cy="modal-overlay"]').should('be.visible');
  })
})
```

DEMO

Example #3 – Storybook Integration

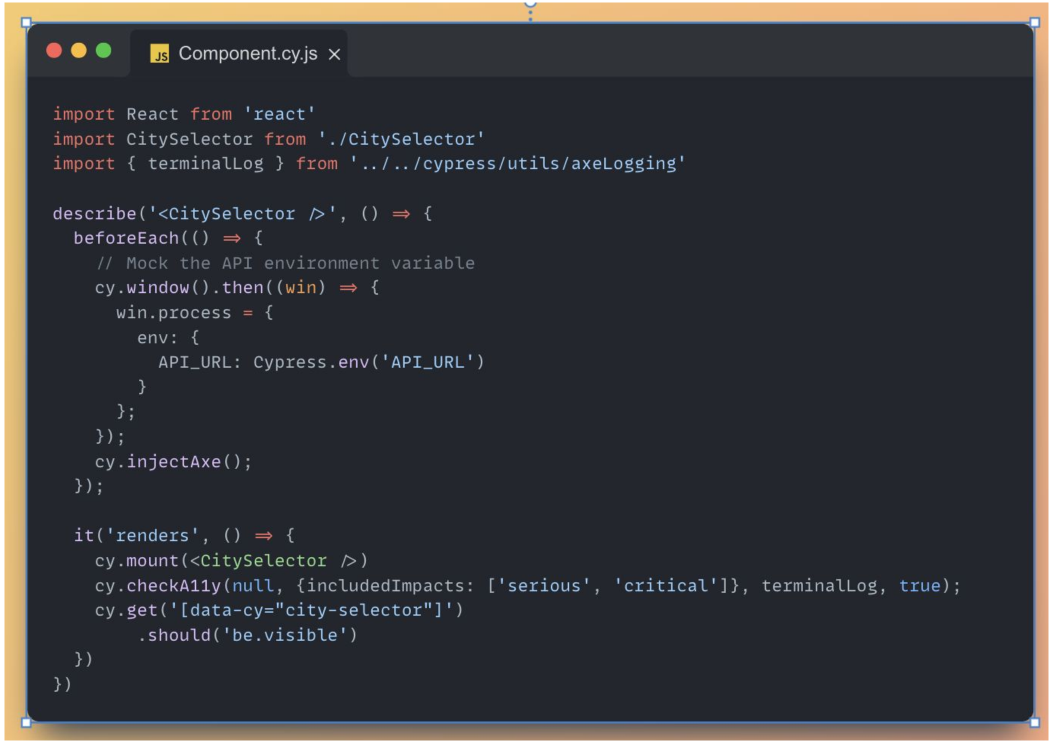
- Integrate accessibility checks in Storybook to ensure components are usable by everyone.
- Run Axe checks for each component story to catch issues early in the development process.
- This approach helps identify and fix accessibility problems at the component level before integration into larger applications.



```
import { terminalLog } from '../utils/axeLogging'
describe('Storybook: CitySelector Component', () => {
    beforeEach(() => {
        cy.visit('http://localhost:6006/iframe.html?id=components-cityselector--default');
        cy.injectAxe();
        cy.contains('[data-cy="city-selector"]', 'Select a city');
    })
    it('should pass accessibility tests for a CitySelector component', () => {
      cy.checkA11y(null, null, terminalLog, true);
    });
  });
```

DEMO

Example #4 – Component Tests



Component.cy.js

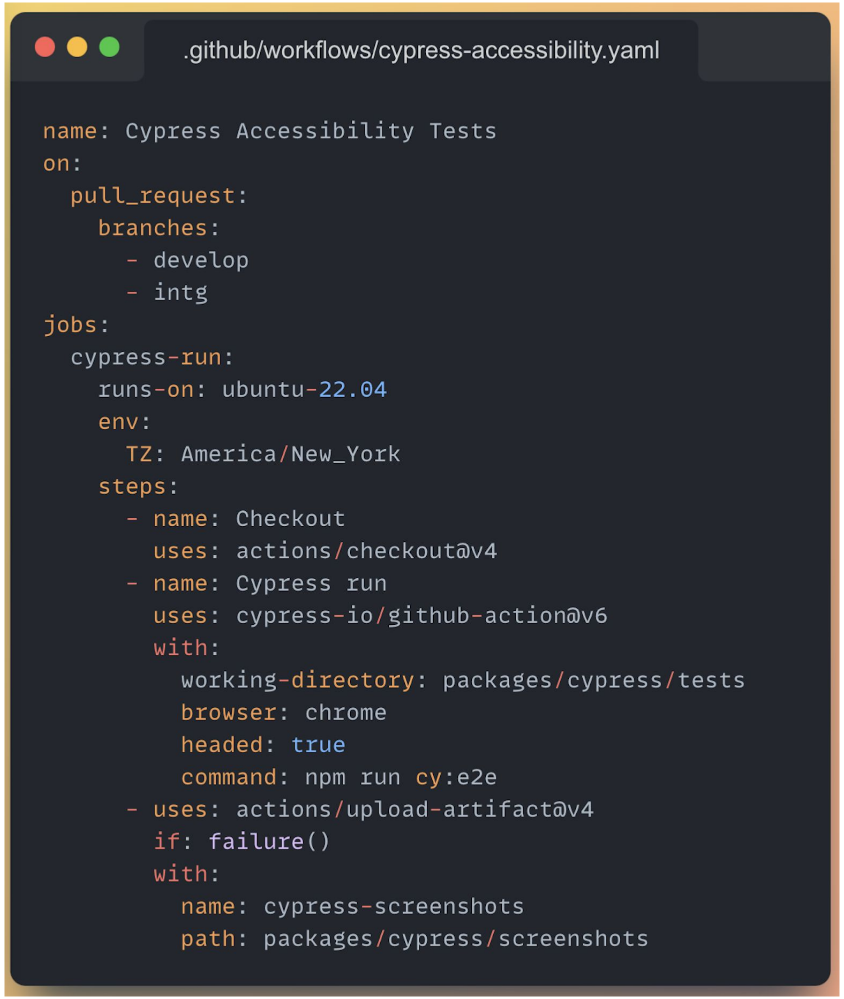
```
import React from 'react'
import CitySelector from './CitySelector'
import { terminalLog } from '../../cypress/utils/axeLogging'

describe('<CitySelector />', () => {
  beforeEach(() => {
    // Mock the API environment variable
    cy.window().then((win) => {
      win.process = {
        env: {
          API_URL: Cypress.env('API_URL')
        }
      };
    });
    cy.injectAxe();
  });

  it('renders', () => {
    cy.mount(<CitySelector />)
    cy.checkA11y(null, {includedImpacts: ['serious', 'critical']}, terminalLog, true);
    cy.get('[data-cy="city-selector"]')
        .should('be.visible')
  })
})
```

- Cypress Component Testing directly tests React, Vue, Angular components.
- Offers fast feedback loops for testing accessibility during development.
- Perfect for 'shift-left' testing to catch early issues.
- Combine accessibility tests with visual regression and snapshot tests.

DEMO

CI/CD Integration

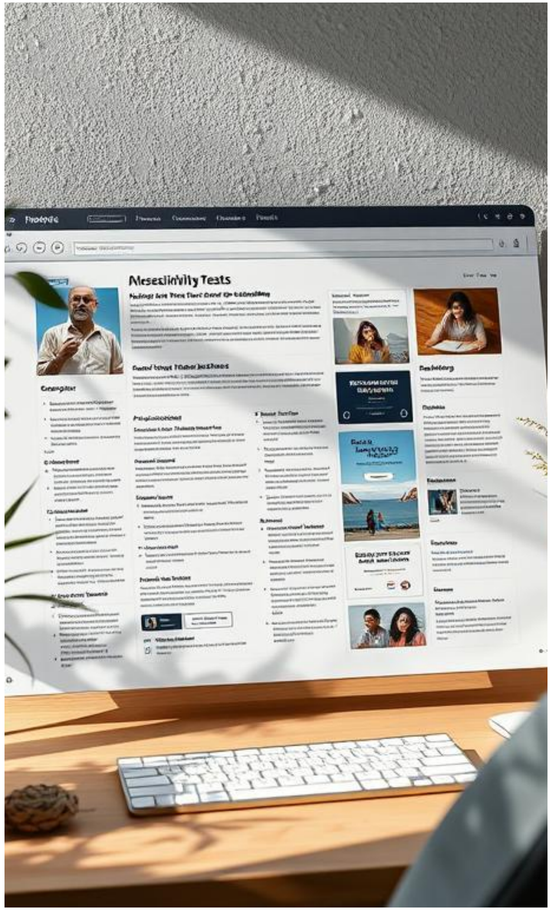
- **Single Source of Truth:** Because Axe is injected directly in your Cypress tests, you don't need separate scripts or tools for accessibility checks.
- **Multiple Test Types:** You can run them on E2E, Component, Storybook, UI-integration.
- **Fail the Pipeline (If Desired):** You can configure your pipeline to fail if certain severity levels of accessibility issues are found, or simply log them as warnings until you're ready to enforce.
- **Scalable Approach:** Start small (critical checks) and expand to more thorough scans as your team grows comfortable fixing issues.


.github/workflows/cypress-accessibility.yaml

```yaml
name: Cypress Accessibility Tests
on:
  pull_request:
    branches:
      - develop
      - intg
jobs:
  cypress-run:
    runs-on: ubuntu-22.04
    env:
      TZ: America/New_York
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Cypress run
        uses: cypress-io/github-action@v6
        with:
          working-directory: packages/cypress/tests
          browser: chrome
          headed: true
          command: npm run cy:e2e
      - uses: actions/upload-artifact@v4
        if: failure()
        with:
          name: cypress-screenshots
          path: packages/cypress/screenshots
```

Handling Accessibility Violations & Prioritization

- Implement a **triage** process to classify accessibility violations by severity, frequency, and user impact.
- Utilize tools like **ARIA** roles, semantic HTML, and design adjustments to address identified issues effectively.
- Prioritize fixing **critical** issues that impact the largest number of users first, followed by less severe violations.
- Encourage **collaboration** between **designers**, **developers**, and **QA** teams to ensure all perspectives are considered in the remediation process.



Challenges

Common Pitfalls & Challenges

- Over-reliance on automated scanning can lead to missed issues; automated tools detect about **30-50%** of accessibility errors.
- Dynamic content in single-page applications (SPAs) may **not fully load** when tests run, causing false negatives in accessibility scans.
- Visual complexities, such as **carousels** and **infinite scrolls**, can obscure underlying accessibility issues, requiring additional checks.
- **Manual checks** are essential for a comprehensive accessibility strategy, including color **contrast verification** and **screen reader** testing.

Best Practices Recap

- Automate accessibility tests early in the development process to catch issues sooner.
- Integrate accessibility testing into the CI/CD pipeline to ensure continuous monitoring and compliance.
- Pair automated testing with manual audits to cover edge cases and provide a comprehensive assessment.
- Educate and train teams on accessibility best practices to foster a culture of inclusivity and awareness.

Resources

- Axe Core documentation: https://www.deque.com/axe/core-documentation/
- Cypress accessibility guide: https://docs.cypress.io/accessibility/guides/introduction
- WCAG 2.1 Checklist: https://www.w3.org/WAI/WCAG21/quickref/
- Cypress-axe plugin: https://www.npmjs.com/package/cypress-axe
- Open source accessibility plugins: https://www.cypress.io/blog/open-source-accessibility-plugins-in-cypress

Q&A