Let’s turn your **simple Node.js CRUD app** into an **accessibility-aware app** and wire it up with **Cypress + axe-core** so you can:

1. Run accessibility audits in Cypress
2. Automate WCAG compliance checks
3. Debug and fix accessibility violations

I’ll go **step by step**, assuming:

- You know basic JavaScript
- You’re okay running commands in terminal
- You’re on macOS and already have Node.js installed

Step 0 – Create a simple Node.js “CRUD-ish” app

We’ll build a tiny **Todo app** (list + create = enough for a11y demo).

0.1 – Create project & install dependencies

```
mkdir node-a11y-todo
cd node-a11y-todo

npm init -y
npm install express body-parser
```

0.2 – Create `server.js`

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const PORT = 3000;

// parse form body
```

```
app.use(bodyParser.urlencoded({ extended: true }));

// in-memory "DB"
let todos = [
  { id: 1, title: 'Learn Cypress', done: false },
  { id: 2, title: 'Add accessibility tests', done: false },
];

app.get('/', (req, res) => {
  res.redirect('/todos');
});

// ❗ First version: INTENTIONALLY with some accessibility issues
app.get('/todos', (req, res) => {
  const listItems = todos
    .map((t) => `<li>${t.title}</li>`)
    .join('');

  res.send(`
    <!doctype html>
    <html lang="en">
    <head>
      <meta charset="utf-8" />
      <title>Todo App</title>
    </head>
    <body>
      <h1>Todo List</h1>
      <ul>
        ${listItems}
      </ul>

      <!-- Bad: input without label, only placeholder -->
      <form method="POST" action="/todos">
        <input name="title" placeholder="New todo" />
        <!-- Bad: button with no type and vague text -->
        <button>Add</button>
      </form>

      <!-- Bad: image without alt -->
      <img src="https://via.placeholder.com/150" />

    </body>
    </html>
  `);
});

app.post('/todos', (req, res) => {
  const title = (req.body.title || '').trim();
  if (title) {
    todos.push({ id: Date.now(), title, done: false });
  }
  res.redirect('/todos');
});

app.listen(PORT, () => {
  console.log(`App running at http://localhost:${PORT}/todos`);
});
```

Run it:

```
node server.js
```

Open `http://localhost:3000/todos` in the browser to confirm it works.

We **intentionally** added some a11y problems so axe can catch them.

Step 1 – Add Cypress + axe-core (cypress-axe)

We'll use **cypress-axe**, which wraps **axe-core** (an industry-standard accessibility engine used by many tools)(Deque).

1.1 – Install test dependencies

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

1.2 – Open Cypress once (to scaffold folders)

```
npx cypress open
```

- Choose **E2E Testing**

- Cypress will create `cypress/` folder and a config file (e.g. `cypress.config.js` or `cypress.config.mjs`).

Close the Cypress window after it finishes scaffolding.

Step 2 – Configure Cypress to use `cypress-axe`

2.1 – Add axe commands to support file

Open `cypress/support/e2e.js` (or `cypress/support/e2e.ts`) and add:

```
// cypress/support/e2e.js
import 'cypress-axe';
```

This gives you handy commands like:

- `cy.injectAxe()` – injects axe into the page
- `cy.checkA11y()` – runs accessibility checks and fails the test on violations(GitHub)

No extra config needed.

Step 3 – First accessibility audit with Cypress

Now we'll write a simple test that:

1. Visits the app
2. Injects axe
3. Runs an accessibility scan
4. Fails if violations are found

3.1 – Create the test file

Create `cypress/e2e/a11y-basic.cy.js`:

```
// cypress/e2e/a11y-basic.cy.js

describe('Todo app basic accessibility', () => {
  it('has no obvious a11y violations on load', () => {
    cy.visit('http://localhost:3000/todos');

    // Inject axe-core into the page
    cy.injectAxe();

    // Run accessibility checks on the whole page
    cy.checkA11y();
  });
});
```

3.2 – Run the test

With your app still running (`node server.js`), run:

```
npx cypress run --spec cypress/e2e/a11y-basic.cy.js
```

or in the interactive UI:

```
npx cypress open
# select E2E, pick a browser, then select a11y-basic.cy.js
```

You should see **failing tests** because:

- The form field has no associated label
- The image has no `alt`
- Maybe missing button type / accessible name details

`cypress-axe` will log **“A11Y ERROR!”** entries in the Cypress Command Log with details(GitHub).

✅ You’ve now **run an automated accessibility audit** with Cypress.

Step 4 – Automating WCAG compliance checks

axe-core rules map to WCAG success criteria. You can restrict checks to specific **WCAG levels** using tags like:

- `wcag2a`, `wcag2aa`, `wcag21a`, `wcag21aa` etc.(W3C)

4.1 – Add a WCAG-focused test

Create `cypress/e2e/a11y-wcag.cy.js`:

```
// cypress/e2e/a11y-wcag.cy.js

describe('Todo app WCAG A/AA compliance', () => {
  it('meets WCAG 2.x A/AA rules (axe core subset)', () => {
    cy.visit('http://localhost:3000/todos');
    cy.injectAxe();

    cy.checkA11y(null, {
      runOnly: {
        type: 'tag',
        values: ['wcag2a', 'wcag2aa', 'wcag21a', 'wcag21aa'],
      },
    });
  });
});
```

Run it:

```
npx cypress run --spec cypress/e2e/a11y-wcag.cy.js
```

This is now:

- **Automated WCAG checks** (for the subset of WCAG rules that axe can determine automatically).
- Perfect to plug into CI later (GitHub Actions / GitLab / Jenkins).

Step 5 – Debugging accessibility violations

Now let’s **debug** and **fix** the issues Cypress/axe is finding.

5.1 – See detailed violations in the test

You can add a callback to `cy.checkA11y` to log extra info:

```
// cypress/e2e/a11y-debug.cy.js

describe('Todo app accessibility – debug', () => {
  it('logs detailed violations', () => {
    cy.visit('http://localhost:3000/todos');
    cy.injectAxe();

    cy.checkA11y(null, null, (violations) => {
      // Log each violation in an easier-to-read format
      violations.forEach((v) => {
        // Shows rule id, impact, and description
        cy.log(`${v.id} [${v.impact}] - ${v.help}`);
        v.nodes.forEach((node) => {
          cy.log('Offending HTML: ' + node.html);
        });
      });
    });
  });
});
```

Run this spec and watch Cypress logs to see:

- **Rule ID** (e.g. `label`, `image-alt`, etc.)
- **Impact**: minor / moderate / serious / critical
- **Help text**: what’s wrong
- **HTML snippet**: where the problem is

You can also open DevTools in Cypress Runner and see axe output in the console(GitHub).

Step 6 – Fix the accessibility violations in the app

Let’s fix the common issues one by one.

6.1 – Add a proper label for the input

Change this **bad** form:

```
<form method="POST" action="/todos">
  <input name="title" placeholder="New todo" />
  <button>Add</button>
</form>
```

to this **better** markup:

```
<form method="POST" action="/todos">
  <label for="title">New todo</label>
  <input id="title" name="title" />
  <button type="submit">Add todo</button>
</form>
```

Key improvements:

- `label` with `for="title"` + input with `id="title"`
- `button type="submit"`
- Button text now more descriptive: “Add todo”

6.2 – Add alt text to images

Change:

```
<img src="https://via.placeholder.com/150" />
```

to:

```
<img src="https://via.placeholder.com/150" alt="Placeholder illustration" />
```

If image is purely decorative, you can use `alt=""` so screen readers ignore it.

6.3 – Slightly update heading structure (optional)

We already have:

```
<h1>Todo List</h1>
```

That's okay, but in bigger pages, make sure headings are hierarchical (`h1` → `h2` → `h3`, no skipping).

6.4 – Updated `server.js` (fixed version of `/todos` handler)

Replace the `app.get('/todos', ...)` function with:

```
app.get('/todos', (req, res) => {
  const listItems = todos
    .map((t) => `<li>${t.title}</li>`)
    .join('');

  res.send(`
    <!doctype html>
    <html lang="en">
    <head>
      <meta charset="utf-8" />
      <title>Todo App - Accessible</title>
    </head>
    <body>
      <header>
        <h1>Todo List</h1>
      </header>

      <main>
        <section aria-labelledby="todo-list-heading">
          <h2 id="todo-list-heading">Your todos</h2>
          <ul>
            ${listItems}
          </ul>
        </section>

        <section aria-labelledby="add-todo-heading">
          <h2 id="add-todo-heading">Add a new todo</h2>
          <form method="POST" action="/todos">
            <label for="title">New todo</label>
            <input id="title" name="title" />
            <button type="submit">Add todo</button>
          </form>
        </section>

        <img
          src="https://via.placeholder.com/150"
          alt="Placeholder illustration"
        />
      </main>
    </body>
    </html>
  `);
});
```

Restart the app:

```
Ctrl + C
node server.js
```

Step 7 – Re-run the Cypress + axe tests

Now run your tests again:

```
npx cypress run --spec cypress/e2e/a11y-basic.cy.js
npx cypress run --spec cypress/e2e/a11y-wcag.cy.js
```

You should see:

- ✅ Fewer or no violations
- If any remain, check the logs and refine the HTML/CSS accordingly.

At this point you are:

- **Running accessibility audits with Cypress** (axe-core via `cypress-axe`)
- **Automating WCAG A/AA checks** using `runOnly` tags
- **Debugging & fixing violations** by reading the axe output and updating your UI

Step 8 – What next? (Optional but useful)

When you're comfortable:

1. **Run a11y checks on multiple states**
   - After creating a todo
   - After deleting / updating a todo
2. **Add to CI**
   - In CI, run: `npx cypress run --spec "cypress/e2e/a11y-*.cy.js"`
   - Make builds fail if there are serious accessibility violations(CircleCI)
3. **Combine automated + manual testing**
   - Keyboard-only navigation checks
   - Quick screen reader pass (NVDA/VoiceOver)(BrowserStack)

If you like, next I can:

- Add **edit/delete routes** to make it a full CRUD app and
- Show how to write Cypress tests that both **check functionality + accessibility together** for each operation.