

Let's turn this into a mini "final hands-on project" with Cypress + a simple Node.js CRUD app.

We'll do 3 things:

1. Build a **very simple Node.js CRUD todo app**
2. Add **real-world end-to-end Cypress tests**
3. Learn **how to debug & speed up those tests + best practices**

I'll assume nothing and go step by step. You can literally copy-paste and run.

1 Create a simple Node.js CRUD app

Step 1: Folder & dependencies

```
mkdir node-crud-cypress  
cd node-crud-cypress
```

```
npm init -y  
npm install express  
npm install --save-dev cypress nodemon
```

Update `package.json` scripts:

```
"scripts": {  
  "start": "node server.js",  
  "dev": "nodemon server.js",  
  "cypress": "cypress open"  
}
```

Step 2: Create `server.js`

This will:

- Serve an **in-memory todo API** (`/api/todos`)

- Serve a **simple HTML UI** from `public/index.html`

```
// server.js
const express = require('express');
const path = require('path');

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

// In-memory "DB"
let todos = [
  { id: 1, text: 'Learn Cypress basics', done: false },
];

app.use(express.json());
app.use(express.static(path.join(__dirname, 'public')));

// Get all todos
app.get('/api/todos', (req, res) => {
  res.json(todos);
});

// Create todo
app.post('/api/todos', (req, res) => {
  const { text } = req.body;
  if (!text) return res.status(400).json({ error: 'Text is required' });

  const newTodo = { id: Date.now(), text, done: false };
  todos.push(newTodo);
  res.status(201).json(newTodo);
});

// Update todo
app.put('/api/todos/:id', (req, res) => {
  const id = Number(req.params.id);
  const { text, done } = req.body;

  const todo = todos.find((t) => t.id === id);
  if (!todo) return res.status(404).json({ error: 'Not found' });

  if (text !== undefined) todo.text = text;
  if (done !== undefined) todo.done = done;
});
```

```

    res.json(todo);
});

// Delete todo
app.delete('/api/todos/:id', (req, res) => {
  const id = Number(req.params.id);
  todos = todos.filter((t) => t.id !== id);
  res.status(204).send();
});

// Utility endpoint for tests to reset data
app.post('/api/test/reset', (req, res) => {
  todos = [{ id: 1, text: 'Learn Cypress basics', done: false }];
  res.status(200).json({ ok: true });
});

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

```

Step 3: Create simple front-end `public/index.html`

`mkdir public`

Create `public/index.html`:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Todo App</title>
    <style>
      body { font-family: sans-serif; max-width: 600px; margin: 20px auto; }
      li { margin: 4px 0; }
      .done { text-decoration: line-through; }
      button { margin-left: 4px; }
    </style>
  </head>
  <body>
    <h1>Todo App</h1>

    <form id="new-todo-form">

```

```
<input  
  id="new-todo-input"  
  name="text"  
  placeholder="New todo..."  
  data-testid="new-todo-input"  
/>  
<button type="submit" data-testid="add-todo-btn">Add</button>  
</form>  
  
<ul id="todo-list" data-testid="todo-list"></ul>  
  
<script>  
  const listEl = document.getElementById('todo-list');  
  const formEl = document.getElementById('new-todo-form');  
  const inputEl = document.getElementById('new-todo-input');  
  
  async function fetchTodos() {  
    const res = await fetch('/api/todos');  
    const todos = await res.json();  
    renderTodos(todos);  
  }  
  
  function renderTodos(todos) {  
    listEl.innerHTML = "";  
    todos.forEach((todo) => {  
      const li = document.createElement('li');  
      li.dataset.id = todo.id;  
      li.dataset.testid = 'todo-item';  
  
      const span = document.createElement('span');  
      span.textContent = todo.text;  
      if (todo.done) span.classList.add('done');  
      span.dataset.testid = 'todo-text';  
  
      const toggleBtn = document.createElement('button');  
      toggleBtn.textContent = todo.done ? 'Undo' : 'Done';  
      toggleBtn.dataset.testid = 'toggle-done-btn';  
  
      const deleteBtn = document.createElement('button');  
      deleteBtn.textContent = 'Delete';  
      deleteBtn.dataset.testid = 'delete-todo-btn';  
  
      toggleBtn.addEventListener('click', () => toggleTodo(todo));  
      deleteBtn.addEventListener('click', () => deleteTodo(todo.id));  
    });  
  }  
</script>
```

```
        li.appendChild(span);
        li.appendChild(toggleBtn);
        li.appendChild(deleteBtn);

        listEl.appendChild(li);
    });
}

async function createTodo(text) {
    await fetch('/api/todos', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ text }),
    });
    await fetchTodos();
}

async function toggleTodo(todo) {
    await fetch('/api/todos/' + todo.id, {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ done: !todo.done }),
    });
    await fetchTodos();
}

async function deleteTodo(id) {
    await fetch('/api/todos/' + id, { method: 'DELETE' });
    await fetchTodos();
}

formEl.addEventListener('submit', (e) => {
    e.preventDefault();
    const text = inputEl.value.trim();
    if (!text) return;
    createTodo(text);
    inputEl.value = "";
});

fetchTodos();
</script>
</body>
</html>
```

Run the app:

```
npm run dev
```

Open <http://localhost:3000> and confirm UI works.

2 Real-world end-to-end Cypress scenario

We'll now:

- Configure Cypress
- Write full **CRUD + toggling** flow
- Use **API reset** for stable tests
- Show how to debug + improve performance

Step 4: Cypress setup

From the project root:

```
npx cypress open
```

- Choose **E2E Testing**
- Framework: “JavaScript” (default)
- Browser: any (Chrome is fine)
- Cypress will create `cypress.config.js` and some example specs.

Update `cypress.config.js`:

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
```

```
e2e: {  
  baseUrl: 'http://localhost:3000',  
},  
});
```

Delete sample spec files if you want, or keep them.

Step 5: Add our real-world E2E test

Create [cypress/e2e/todo.e2e.cy.js](#):

```
describe('Todo App - real world E2E', () => {  
  // Reset backend before each test for consistency  
  beforeEach(() => {  
    cy.request('POST', '/api/test/reset');  
    cy.visit('/');  
  });  
  
  it('loads initial todos', () => {  
    cy.contains('h1', 'Todo App').should('be.visible');  
  
    cy.get('[data-testid="todo-list"]')  
      .children()  
      .should('have.length', 1);  
  
    cy.get('[data-testid="todo-item"]')  
      .first()  
      .within(() => {  
        cy.get('[data-testid="todo-text"]').should(  
          'have.text',  
          'Learn Cypress basics'  
        );  
      });  
  });  
  
  it('allows user to create a new todo (Create)', () => {  
    cy.get('[data-testid="new-todo-input"]').type('Write Cypress E2E test');  
    cy.get('[data-testid="add-todo-btn"]').click();  
  
    cy.get('[data-testid="todo-list"]')  
      .children()  
      .should('have.length', 2);  
  });
```

```

    cy.contains('[data-testid="todo-text"]', 'Write Cypress E2E test')
      .should('exist');
  });

it('allows user to toggle a todo (Update)', () => {
  cy.contains('[data-testid="todo-text"]', 'Learn Cypress basics')
    .parents('[data-testid="todo-item"]')
    .as('firstTodo');

  cy.get('@firstTodo')
    .find('[data-testid="toggle-done-btn"]')
    .click();

  cy.get('@firstTodo')
    .find('[data-testid="todo-text"]')
    .should('have.class', 'done');
});

it('allows user to delete a todo (Delete)', () => {
  cy.get('[data-testid="todo-item"]')
    .should('have.length', 1)
    .as('todos');

  cy.get('@todos')
    .first()
    .find('[data-testid="delete-todo-btn"]')
    .click();

  // After delete, list should be empty
  cy.get('[data-testid="todo-list"]').children().should('have.length', 0);
});

it('end-to-end user flow: create -> toggle -> delete', () => {
  // Create
  cy.get('[data-testid="new-todo-input"]').type('Full E2E flow');
  cy.get('[data-testid="add-todo-btn"]').click();

  cy.contains('[data-testid="todo-text"]', 'Full E2E flow')
    .parents('[data-testid="todo-item"]')
    .as('flowTodo');

  // Toggle
  cy.get('@flowTodo')

```

```

    .find('[data-testid="toggle-done-btn"]')
    .click();

    cy.get('@flowTodo')
      .find('[data-testid="todo-text"]')
      .should('have.class', 'done');

    // Delete
    cy.get('@flowTodo')
      .find('[data-testid="delete-todo-btn"]')
      .click();

    cy.contains('[data-testid="todo-text"]', 'Full E2E flow')
      .should('not.exist');
  });
}
);

```

Run the tests in **Cypress UI**:

```

npm run dev      # terminal 1
npm run cypress  # terminal 2

```

Pick `todo.e2e.cy.js` and watch it run – this is your **real-world end-to-end scenario**:

- Resets DB
 - Visits the app
 - Performs full CRUD via UI
 - Uses **API for setup/cleanup** (best practice)
-

3 Debugging & improving Cypress test performance

Now imagine this is a real project and tests are slow or flaky. Here's how to approach it like a pro.

A. Debugging tests

1. Use `cy.only` while debugging

```
it.only('end-to-end user flow: create -> toggle -> delete', () => {
  // ...
});
```

This runs just that test in the runner, saving time.

2. Use `cy.pause()` to inspect step-by-step

Inside your test:

```
cy.get('[data-testid="new-todo-input"]').type('Full E2E flow');
cy.pause();
cy.get('[data-testid="add-todo-btn"]').click();
```

- Cypress will stop after `cy.pause()`
 - You can interact with the app manually in the browser
 - Then click “Resume” in the UI
-

3. Use `cy.log()` and browser devtools

```
cy.log('Creating todo: Full E2E flow');
```

Plus:

- Open the **DevTools console** in Cypress runner
- Check **network tab** for failed API calls
- Check JavaScript errors

4. Screenshots & videos (in `cypress run`)

Run headless:

```
npx cypress run --spec cypress/e2e/todo.e2e.cy.js
```

- Cypress automatically captures **screenshots on failure**
 - Videos of each test run
 - You can review them to see what went wrong
-

B. Improving test performance

1. Avoid `cy.wait(5000)` with fixed timeouts

Bad:

```
cy.wait(5000);
cy.get('[data-testid="todo-item"]').should('have.length', 2);
```

Good (use Cypress retry-ability):

```
cy.get('[data-testid="todo-item"]').should('have.length', 2);
```

Cypress will **retry get + should** until the assertion passes or times out.

2. Use API for setup instead of UI

We already did this:

```
beforeEach(() => {
  cy.request('POST', '/api/test/reset');
```

```
    cy.visit('/');
});
```

This is **faster and more stable** because:

- No UI steps just to reach a state
- Directly resetting in-memory data

You can extend this pattern:

- `/api/test/createTodo`
 - `/api/test/seed`
-

3. Group tests logically & reuse flows

You did that with:

```
describe('Todo App - real world E2E', () => { ... });
```

Inside, you reuse:

- `beforeEach` for setup
- Common selectors `[data-testid="..."]`

This reduces DOM queries & complexity.

4. Run headless in CI

For performance in CI, use:

```
npx cypress run --browser chrome --headless
```

Combined with:

- Parallelization (later with Cypress Cloud or CI matrix)
 - Smaller spec files (per feature)
-

4 Final Q&A / Best-practice “cheat sheet”

You can treat this as your **review slide** when teaching:

Test structure

- Use **describe** per feature/module (e.g., Todo App, Users, Auth)
 - Use **it** blocks to describe user behavior:
"allows user to create a new todo" instead of "test1".
-

Selectors

- Use **stable selectors** like **data-testid**
(don't rely on CSS classes that may change with styling)
-

State management

- Use **API endpoints** for:
 - resetting DB
 - seeding data
 - cleaning up
- Keep each test **independent** of others

Flakiness prevention

- Never use blind `cy.wait(xxxx)` unless absolutely necessary
 - Prefer retries:
 - `cy.get(...).should(...)`
 - Handle dynamic elements with:
 - `cy.contains(...)`
 - `within(...)`
 - Always understand where the async behavior is (network, animations, etc.)
-

Performance

- Run only focused tests locally with `it.only`, `describe.only`
 - Run **full suite headless** in CI
 - Split specs by feature for parallel CI runs
-

If you want, next step we can:

- Add **one more “real-world” twist** (e.g. login screen + protected todo page)
- Or turn this into a **teaching plan** for your workshop with slides + exercises based on this app.