Let’s walk through this like you’re building it fresh, step by step, using:

- A **simple Node.js + Express CRUD app**
- **Cypress + Cypress Cloud**
- And then how to use **visual debugging features**:
    - Test Replay in CI
    - Error Groups
    - Flaky Test Detection & History

---

0. Big picture: what “Visual Reviews” actually means

Once tests run in CI and are recorded to **Cypress Cloud**, you get:

1. **Test Replay in CI**
    - Watch a **video** of the exact run
    - See **DOM snapshots** for each step
    - See **console logs, network calls, errors**, etc.
2. **Failure Debugging with Error Groups**
    - Cypress Cloud groups failures with the **same error stack / message** together
    - You can spot patterns: “Ah, most failures are due to login timing,” etc.
3. **Flaky Test Detection & History**
    - Same test passes sometimes, fails sometimes = **flaky**
    - Cloud shows:
        - Pass/fail history over time

- Flaky label + count
- Which branches / commits are most flaky

Everything we do below has one goal: **get your CRUD app + Cypress hooked to Cloud**, then **run in CI** and use those visual tools.

---

1. Create a simple Node.js CRUD app

1.1. Setup project

```
mkdir visual-review-crud
cd visual-review-crud
npm init -y

# Install server deps
npm install express cors body-parser
# For dev
npm install --save-dev nodemon
```

1.2. Add a simple Express server

Create `server.js`:

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

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

app.use(cors());
app.use(bodyParser.json());

// In-memory "database"
let items = [];
let idCounter = 1;

// CREATE
```

```
app.post('/api/items', (req, res) => {
  const { name } = req.body;
  if (!name) return res.status(400).json({ error: 'Name is required' });

  const newItem = { id: idCounter++, name };
  items.push(newItem);
  res.status(201).json(newItem);
});

// READ ALL
app.get('/api/items', (req, res) => {
  res.json(items);
});

// UPDATE
app.put('/api/items/:id', (req, res) => {
  const id = Number(req.params.id);
  const { name } = req.body;

  const item = items.find(i => i.id === id);
  if (!item) return res.status(404).json({ error: 'Item not found' });

  item.name = name || item.name;
  res.json(item);
});

// DELETE
app.delete('/api/items/:id', (req, res) => {
  const id = Number(req.params.id);
  items = items.filter(i => i.id !== id);
  res.status(204).send();
});

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

Update `package.json` scripts:

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

Run:

npm run dev

Check in browser: `http://localhost:3000/api/items` → should return `[]`.

---

2. Add a tiny front-end (simple HTML) for CRUD

Create `public/index.html`:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>CRUD App</title>
</head>
<body>
  <h1>Items</h1>

  <form id="item-form">
    <input type="text" id="item-name" placeholder="Item name" />
    <button type="submit">Add Item</button>
  </form>

  <ul id="items-list"></ul>

  <script>
    const API_URL = 'http://localhost:3000/api/items';

    async function fetchItems() {
      const res = await fetch(API_URL);
      const items = await res.json();
      const list = document.getElementById('items-list');
      list.innerHTML = '';
      items.forEach(item => {
        const li = document.createElement('li');
        li.textContent = item.name;
        li.dataset.id = item.id;
```

```
      const delBtn = document.createElement('button');
      delBtn.textContent = 'Delete';
      delBtn.onclick = () => deleteItem(item.id);

      li.appendChild(delBtn);
      list.appendChild(li);
    });
  }

  async function addItem(name) {
    await fetch(API_URL, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ name }),
    });
    fetchItems();
  }

  async function deleteItem(id) {
    await fetch(`${API_URL}/${id}`, { method: 'DELETE' });
    fetchItems();
  }

  document.getElementById('item-form').addEventListener('submit', (e) => {
    e.preventDefault();
    const input = document.getElementById('item-name');
    const name = input.value.trim();
    if (!name) return alert('Name is required');
    addItem(name);
    input.value = '';
  });

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

Serve this static file from Express by updating `server.js`:

```
const path = require('path');
// ...
app.use(express.static(path.join(__dirname, 'public')));
```

Now `http://localhost:3000/` shows your simple CRUD UI.

---

3. Install and set up Cypress + Cypress Cloud

3.1. Install Cypress

npm install --save-dev cypress

Add script:

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

Run once to scaffold:

npx cypress open

Cypress will create `cypress/` folder and `cypress.config.js` (or `cypress.config.mjs`).

3.2. Add a basic CRUD test

Create `cypress/e2e/crud.cy.js`:

```
describe('CRUD App', () => {
  const baseUrl = 'http://localhost:3000';

  it('can create and delete an item', () => {
    cy.visit(baseUrl);

    cy.get('#item-name').type('Test Item');
    cy.contains('button', 'Add Item').click();

    cy.contains('li', 'Test Item')
      .should('exist')
      .within(() => {
        cy.contains('button', 'Delete').click();
```

```
    });

  cy.contains('li', 'Test Item').should('not.exist');
 });
});
```

Run (make sure `npm run dev` is running in another terminal):

npx cypress run --spec cypress/e2e/crud.cy.js

---

4. Connect project to Cypress Cloud

> You'll need a free Cypress Cloud account.

4.1. Install Cypress Cloud dependency

npm install --save-dev @cypress/webpack-preprocessor
(Not strictly required just for Cloud, but common. If default setup works, you can skip.)

More important: **login & set up project**.

From terminal:

npx cypress cloud login

This will open a browser, ask you to login. Once logged in, Cypress will generate a **record key** for your project (`CYPRESS_RECORD_KEY`).

You'll then update your `cypress.config.js` to include projectId. For example:

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

module.exports = defineConfig({
 e2e: {
  baseUrl: 'http://localhost:3000',
  setupNodeEvents(on, config) {
   // node event listeners here
  },
 },
 projectId: 'abcd1234', // from Cypress Cloud
```

});

4.2. Record tests to Cypress Cloud (locally first)

Run tests with `--record` (and optionally `--key`):

npx cypress run --record --key YOUR_RECORD_KEY --spec cypress/e2e/crud.cy.js

Output will include a **Cloud URL** like:

> Recorded run: https://cloud.cypress.io/projects/abcd1234/runs/5

Visit that link: this is where **Visual Reviews** happen.

---

5. Test Replay in CI (Visual Debugging)

Now let's wire this into a real CI pipeline. I'll use **GitHub Actions** as example (same idea for GitLab/Jenkins).

5.1. Add GitHub Actions workflow

Create `.github/workflows/cypress.yml`:

```
name: Cypress Tests

on:
  push:
    branches: [ main ]
  pull_request:

jobs:
  cypress-run:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node
        uses: actions/setup-node@v4
```

```
      with:
        node-version: '20'

    - name: Install dependencies
      run: npm install

    - name: Start server
      run: npm run start &
      env:
        NODE_ENV: test

    - name: Wait for server
      run: npx wait-on http://localhost:3000

    - name: Run Cypress tests in CI and record
      run: npx cypress run --record --key ${{ secrets.CYPRESS_RECORD_KEY }} --spec
cypress/e2e/crud.cy.js
      env:
        CYPRESS_PROJECT_ID: abcd1234
```

In GitHub repo settings → **Secrets and variables** → **Actions** → add:

- `CYPRESS_RECORD_KEY` = your record key

5.2. How Test Replay works (what you'll *see* as beginner)

After a CI run:

1. Go to Cypress Cloud project.
2. Open the **latest run** (from CI).
3. Click a test (e.g., `CRUD App can create and delete an item`).
4. You'll see:
    - **Video** of the whole run.
    - **Command log** (each Cypress command).
    - **DOM snapshot** at each step.

- **Console logs, network tab, screenshots** for failures.

This is the **visual replay** of your CI run.
You can pause, jump to specific steps, and inspect the UI like a time machine.

---

6. Failure Debugging with Error Groups

Let's intentionally introduce a small bug so we can see error groups.

6.1. Introduce a bug

Change your test to expect wrong text:

```
// cypress/e2e/crud.cy.js
it('can create and delete an item', () => {
  cy.visit('/');

  cy.get('#item-name').type('Test Item');
  cy.contains('button', 'Add Item').click();

  // WRONG EXPECTATION:
  cy.contains('li', 'Some Other Text') // will fail
    .should('exist');
});
```

Commit and push → CI runs → test fails → recorded to Cloud.

6.2. Error groups in Cypress Cloud (what to click)

In Cypress Cloud:

1. Open your project.
2. Go to **Runs**.
3. Find the **failed run**.
4. You'll see:
    - A **Failed** label on the run.

- If you click into the run, there will be a section for **Error groups** or a sidebar showing **Top failing tests / Top errors**.

5. Click the error group:
    - It shows:
        - The **error message** (e.g., “Expected to find content: ‘Some Other Text’ but never did”).
        - Stack trace.
        - All **test runs** that failed with the *same* error.

As you add more failures with similar patterns (e.g., multiple tests failing due to timeout or missing element), Cypress Cloud groups them.

How this helps you visually:

- Instead of checking each run individually, you see:
  “20 failures all due to `cy.contains('Submit').click()` not finding the button”.
- You can then replay **one** of those failures, fix the root cause, and many tests will be fixed.

---

7. Flaky Test Detection & History

Now we’ll simulate a **flaky** test: passes sometimes, fails sometimes.

7.1. Make test flaky on purpose (for learning)

Example: randomly fail in the test:

```
it('sometimes fails (flaky demo)', () => {
  cy.visit('/');

  const random = Math.random();
  cy.log(`Random value: ${random}`);

  if (random < 0.5) {
```

```
    // Simulate a failure
    cy.wrap(null).should('equal', 1);
  } else {
    // Normal passing path
    cy.get('#item-name').type('Flaky Item');
    cy.contains('button', 'Add Item').click();
    cy.contains('li', 'Flaky Item').should('exist');
  }
});
```

Run this multiple times in CI (push a few commits or re-run pipeline).
Result: same test **sometimes passes, sometimes fails**.

7.2. See flakiness in Cypress Cloud

In Cloud UI:

1. Open the **Specs / Tests** tab.
2. Find the flaky test (e.g., `sometimes fails (flaky demo)`).
3. Cypress shows:
    - Test status trends (green/red over time).
    - A **Flaky** label if it meets their internal flakiness criteria (e.g., at least one pass and one fail in recent runs).
4. You can:
    - Click the test → see **runs history**.
    - Click into **failed runs** → watch **video replay** for failing cases.
    - Compare with **passing runs** to see what changed (e.g., timing, network, environment).

In real life, flakiness usually comes from:

- Not waiting for API response (element not ready).
- Animations / transitions not finished.

- Using `cy.wait(1000)` instead of deterministic waits like `cy.intercept` + `cy.wait('@alias')`.

Cloud's flake history + videos help you visually spot:

- "In failing runs, spinner never disappeared."
- "In passing runs, spinner disappears quickly."

---

8. How all three concepts tie back to your CRUD app

With this simple CRUD app + Cypress + Cloud, you can practice:

8.1. Test Replay in CI

- Go to a failed run.
- Replay video.
- Inspect DOM and network.
- Fix the issue.

8.2. Error Groups

- Introduce multiple similar failures (e.g., wrong selectors in several tests).
- See how Cypress groups them.
- Use group info to prioritize what to fix first.

8.3. Flaky Test Detection & History

- Add one or two intentionally flaky tests.
- Let CI run them multiple times.

- Observe:
    - Flaky label.
    - Pass/fail history.
    - Compare failing vs passing video replays.

---

9. Suggested learning order for you (as a beginner)

1. **Get the CRUD app working locally.**
2. **Add Cypress and write 1 simple passing test.**
3. **Connect Cypress to Cloud & record a local run.**
4. **Set up CI (GitHub Actions) to run tests and record to Cloud.**
5. **Intentionally break a test → observe failure replay + error group.**
6. **Add a flaky test → run CI multiple times → study flakiness view.**
7. Slowly remove intentional flakiness and bugs, and use the same tools on real issues.