

Let's do this step-by-step around a *very simple* Node.js + Express CRUD app, and then layer Cypress + CI/CD + JIRA + Slack/Teams on top.

I'll assume:

- You already have a **Node.js CRUD app + Cypress** tests running locally.
- Code is in a Git repo (GitHub / GitLab / Bitbucket etc.).

If you don't yet have the Node CRUD + Cypress basic setup, tell me and I'll quickly scaffold that first. For now I'll focus on **Integrated Workflows**:

1. CI/CD: run Cypress in pipeline
 2. JIRA: create issues automatically for failures
 3. Slack/Teams: send failure alerts
-

1. Prepare your project for CI

Step 1.1: Make sure Cypress can run headless

In your project root:

```
npx cypress run
```

- This should run your tests **without opening the GUI**.
- Fix any local issues before touching CI.

Step 1.2: Add basic npm scripts

In `package.json`:

```
{  
  "scripts": {  
    "test": "npm run cy:run",  
    "cy:run": "cypress run",  
  },  
}
```

```
"start": "node server.js",      // or nodemon, etc.
"start:test": "NODE_ENV=test node server.js"
}
}
```

We'll use these scripts inside the CI pipeline.

2. Configure Cypress in CI/CD (example: GitHub Actions)

Let's take **GitHub Actions** as a concrete example. (Concept is same for GitLab CI, Jenkins, etc.)

Step 2.1: Create workflow file

Create directory & file:

```
mkdir -p .github/workflows
touch .github/workflows/cypress.yml
```

Step 2.2: Basic workflow content

Put this into `.github/workflows/cypress.yml`:

```
name: CI - Node CRUD + Cypress
```

```
on:
```

```
  push:
```

```
    branches: [ main ]
```

```
  pull_request:
```

```
    branches: [ main ]
```

```
jobs:
```

```
  test:
```

```
    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 app

run: npm run start:test &

- name: Wait for app to be ready

run: npx wait-on http://localhost:3000

- name: Run Cypress tests

run: npm run cy:run

- name: Archive Cypress videos and screenshots

uses: actions/upload-artifact@v4

if: failure()

with:

name: cypress-artifacts

path: |

cypress/videos

cypress/screenshots

What's happening:

- **Checkout code** – gets your Node app + Cypress tests.
- **Setup Node** – installs a Node version.
- **Install dependencies** – `npm install`.
- **Start app** – your Express CRUD is started on port 3000.
- **Wait-on** – only run tests when `http://localhost:3000` is responding.
- **Run Cypress tests** – `npx cypress run`.
- If Cypress fails, videos/screenshots are uploaded as pipeline artifacts.

For GitLab/Jenkins: same logic:

- install Node & Cypress,
- start app,
- wait for app,
- run `npx cypress run`.

Once you push this file to `main`, GitHub will automatically run Cypress on each push/PR.

3. JIRA Integration – Logging failed tests as issues

Goal: **When Cypress tests fail in CI, create a JIRA ticket automatically.**

Simplest approach:

- Use CI pipeline's condition `if: failure()`
- Call JIRA REST API to create an issue.

We'll do it in GitHub Actions style, but it's the same idea elsewhere.

Step 3.1: Create a JIRA API token

1. Go to your JIRA Cloud account.
2. Go to **Account settings** → **Security** → **API tokens**.
3. Generate a new token and copy it.

You now have:

- JIRA base URL: `https://yourcompany.atlassian.net`
- JIRA email: `you@example.com`
- JIRA API token: `xxxxx`
- Default Project Key: e.g. `QA` or `TEST`

Step 3.2: Add secrets to your repo

In GitHub:

- Go to **Settings** → **Secrets and variables** → **Actions** → **New repository secret**.
- Add:
 - `JIRA_BASE_URL` = `https://yourcompany.atlassian.net`
 - `JIRA_EMAIL` = your email
 - `JIRA_API_TOKEN` = your token
 - `JIRA_PROJECT_KEY` = e.g. QA

Step 3.3: Add a JIRA issue creation step in workflow

Update `.github/workflows/cypress.yml` with a new step **after** Cypress run:

```
- name: Create JIRA issue on failure
  if: failure()
  run: |
    ISSUE_SUMMARY="Cypress tests failed in CI for $GITHUB_REPOSITORY"
    ISSUE_DESCRIPTION="Branch: $GITHUB_REF\nCommit: $GITHUB_SHA\nRun:
$GITHUB_RUN_ID\n\nCheck Cypress artifacts for logs, screenshots and videos."

    curl -X POST \
      -H "Content-Type: application/json" \
      -u "${{ secrets.JIRA_EMAIL }}:${{ secrets.JIRA_API_TOKEN }}" \
      --data "{
        \"fields\": {
          \"project\": { \"key\": \"${{ secrets.JIRA_PROJECT_KEY }}\" },
          \"summary\": \"${ISSUE_SUMMARY}\",
          \"description\": \"${ISSUE_DESCRIPTION}\",
          \"issuetype\": { \"name\": \"Bug\" }
        }
      }" \
      "${{ secrets.JIRA_BASE_URL }}/rest/api/3/issue"
```

Explanation:

- `if: failure()` → run this only when earlier steps (Cypress) fail.
- `curl` hits **JIRA REST API** to create a **Bug**.
- Description includes branch, commit, and GitHub run ID so the dev can quickly open CI and see Cypress artifacts (screenshots, videos).

Beginner Tip: Don't worry about the JSON too much; it's just telling JIRA:

- which project (`project.key`)
- issue summary
- description
- type = Bug

You can later enhance this to:

- Parse the test report (e.g. JUnit XML) and include *which test* failed.
 - Add labels like `cypress`, `automated-test`.
-

4. Failure Notifications in Slack

Goal: When Cypress fails in CI, send a message to a Slack channel.

Step 4.1: Create an Incoming Webhook in Slack

1. In Slack, go to **Apps** → search for **Incoming WebHooks** (or create a Slack app).
2. Configure a new Incoming Webhook:
 - Choose the channel (e.g., `#qa-alerts`).
 - Copy the **Webhook URL** (looks like `https://hooks.slack.com/services/...`).

3. Add this webhook URL as a GitHub secret:

- `SLACK_WEBHOOK_URL` = that URL.

Step 4.2: Add Slack notification step in workflow

In `.github/workflows/cypress.yml`, add:

```
- name: Notify Slack on failure
  if: failure()
  run: |
    PAYLOAD=$(cat <<EOF
    {
      "text": "🔴: Cypress tests failed for *${GITHUB_REPOSITORY}* on branch
*${GITHUB_REF}*.\nCheck run:
${GITHUB_SERVER_URL}/${GITHUB_REPOSITORY}/actions/runs/${GITHUB_RUN_ID}"
    }
    EOF
    )

    curl -X POST \
      -H 'Content-type: application/json' \
      --data "$PAYLOAD" \
      "${{ secrets.SLACK_WEBHOOK_URL }}"
```

What happens:

- Only runs on **failure**.
- Sends a simple message with a link to the failing run.

You can later:

- Include emojis for severity,
 - Include failing test name(s),
 - Use attachments/blocks for prettier messages.
-

5. Failure Notifications in Microsoft Teams

Teams also supports **Incoming Webhooks**.

Step 5.1: Create incoming webhook in Teams

1. In Teams, go to the channel (e.g., **QA Alerts**).
2. Click channel ... → **Connectors** → **Incoming Webhook**.
3. Configure a name + icon, and copy the **Webhook URL**.

Add it in GitHub secrets:

- **TEAMS_WEBHOOK_URL** = that URL.

Step 5.2: Add Teams notification step

Add this step in workflow:

```
- name: Notify Microsoft Teams on failure
  if: failure()
  run: |
    PAYLOAD=$(cat <<EOF
    {
      "@type": "MessageCard",
      "@context": "http://schema.org/extensions",
      "summary": "Cypress Test Failure",
      "themeColor": "FF0000",
      "title": "Cypress tests failed",
      "sections": [
        {
          "activityTitle": "Repo: ${GITHUB_REPOSITORY}",
          "text": "Branch: ${GITHUB_REF}\nRun:
${GITHUB_SERVER_URL}/${GITHUB_REPOSITORY}/actions/runs/${GITHUB_RUN_ID}"
        }
      ]
    }
    EOF
    )

    curl -H "Content-Type: application/json" \
```

```
-d "$PAYLOAD" \  
"$${ secrets.TEAMS_WEBHOOK_URL }}"
```

This will post a card message to Teams when tests fail.

6. How this all fits your simple Node.js CRUD app

Imagine your CRUD app has Cypress tests:

- `cypress/e2e/create_item.cy.js`
- `cypress/e2e/update_item.cy.js`
- `cypress/e2e/delete_item.cy.js`

Each commit / PR will:

1. **CI runs:**

- Install dependencies
- Start Node CRUD app
- Run Cypress headless

2. **If tests pass:**

- Pipeline is green.
- Optional: you can deploy.

3. **If tests fail:**

- Artifacts: videos/screenshots uploaded.
- **JIRA ticket** automatically created with run details.
- **Slack and/or Teams** messages sent to alert your team.

So with one push:

- CI/CD tells you if the new CRUD change broke UI or API.
 - If broken, your team sees it on Slack/Teams and there's a JIRA bug created.
-

7. Minimal checklist for you (as a beginner)

Follow this order:

1. ☒ Get `npx cypress run` working **locally**.
 2. ☒ Add npm scripts (`cy:run`, `start:test`).
 3. ☒ Create `.github/workflows/cypress.yml` with:
 - Start app → wait-on → run Cypress → upload artifacts.
 4. ☒ Get at least one **failing** run and see that CI shows Cypress failure.
 5. ☒ Set up **JIRA API token** + GitHub secrets → confirm a JIRA Bug is created on failure.
 6. ☒ Set up **Slack webhook** → confirm a message is posted on failure.
 7. ☒ Optionally add **Teams webhook** similarly.
-

If you want, next I can:

- Take your existing Node CRUD repo structure and **write the exact CI YAML** tailored to it,
- Or show a **full example repo layout** (`server.js`, `cypress.config`, `e2e specs`) and how all this plugs in.