# CI/CD Integration and Best Practice

```yaml
name: Playwright Tests
on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Environment to run tests against'
        required: true
        default: 'staging'
        type: choice
        options:
          - staging
          - production
      browser:
        description: 'Browser to run tests on'
        required: false
        default: 'chromium'
        type: choice
        options:
          - chromium
          - firefox
          - webkit

jobs:
  test:
    timeout-minutes: 60
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-node@v3
      with:
        node-version: 20
    - name: Install dependencies
      run: npm ci
    - name: Install Playwright Browsers
      run: npx playwright install --with-deps
    - name: Start backend
      run: ./gradlew bootRun &
    - name: Wait for backend to start
      run: sleep 30
    - name: Start frontend
```

```
   run: npm run dev &
  - name: Wait for frontend to start
   run: sleep 10
  - name: Run Playwright tests
   run: npx playwright test --project=${{ github.event.inputs.browser }}
  - uses: actions/upload-artifact@v3
   if: always()
   with:
    name: playwright-report
    path: playwright-report/
    retention-days: 30

 # Add a job to trigger another workflow in a different repository
 trigger-downstream:
  needs: test  # This ensures this job runs after the test job completes successfully
  runs-on: ubuntu-latest
  steps:
   - name: Trigger downstream workflow
    uses: peter-evans/repository-dispatch@v2
    with:
     token: ${{ secrets.PAT_TOKEN }}  # You need to create this secret in your repository
settings
     repository: your-organization/downstream-repo
     event-type: playwright-tests-completed
     client-payload: '{"environment": "${{ github.event.inputs.environment }}", "status":
"success", "run_id": "${{ github.run_id }}"}'
```

The above code example is best practice to deliver day 1 benefits to the developers and business. This is a GitHub Actions workflow file that sets up and runs Playwright tests, with the addition of a mechanism to trigger a workflow in another repository. Let me explain each part:

## Workflow Triggers and Parameters

- **Name**: "Playwright Tests"
- **Trigger**: Manual trigger (`workflow_dispatch`) with customizable inputs:
  - `environment`: Required selection between "staging" or "production" (defaults to "staging")
  - `browser`: Optional selection of browser - "chromium", "firefox", or "webkit" (defaults to "chromium")

## Main Testing Job

The `test` job does the following in sequence:

1. **Sets up environment**:
    - Runs on latest Ubuntu (`ubuntu-latest`)
    - Times out after 60 minutes
2. **Setup steps**:
    - Checks out the repository code
    - Sets up Node.js version 20
    - Installs npm dependencies with `npm ci`
    - Installs Playwright browsers and their system dependencies
3. **Starts application**:
    - Launches the backend (Spring Boot app via Gradle) in the background
    - Waits 30 seconds for backend initialization
    - Starts the frontend development server with `npm run dev`
    - Waits 10 seconds for frontend initialization
4. **Runs tests**:
    - Executes Playwright tests using the browser selected in the workflow inputs
    - The `--project=${{ github.event.inputs.browser }}` parameter targets specific browser configurations
5. **Preserves test reports**:
    - Uploads the test report as an artifact
    - Keeps reports for 30 days
    - Uploads even if tests fail (`if: always()`)

## Cross-Repository Workflow Trigger

The `trigger-downstream` job:

- Only runs after the `test` job completes successfully (`needs: test`)
- Uses the `peter-evans/repository-dispatch` action to trigger a workflow in another repository
- Requires a Personal Access Token stored as a repository secret (`PAT_TOKEN`)
- Specifies the target repository (`your-organization/downstream-repo`)
- Defines the event type (`playwright-tests-completed`) that the target workflow listens for
- Passes data to the target workflow via `client-payload`:
    - The environment being tested
    - Success status
    - The run ID of the current workflow for reference

This workflow demonstrates a modern CI/CD approach that:

1. Allows manual triggering with configurable parameters
2. Sets up and tests a full-stack application
3. Implements cross-repository workflow orchestration for complex pipelines
4. Preserves test reports for debugging and analysis

The cross-repository feature is particularly useful for scenarios like:

- Triggering deployment after successful tests
- Notifying other teams or systems about test results
- Coordinating workflows across multiple repositories in a microservice architecture

The above code snippet and explanation is extracted as a POC from the following medium article

**https://medium.com/hostspaceng/triggering-workflows-in-another-repository-with-github-actions-4f581f8e0ceb**