SECTIONS 7-9: CI/CD Pipeline, Employee Usage & Multi-Team Management

SECTION 7 — CI/CD Pipeline for Playwright

Framework

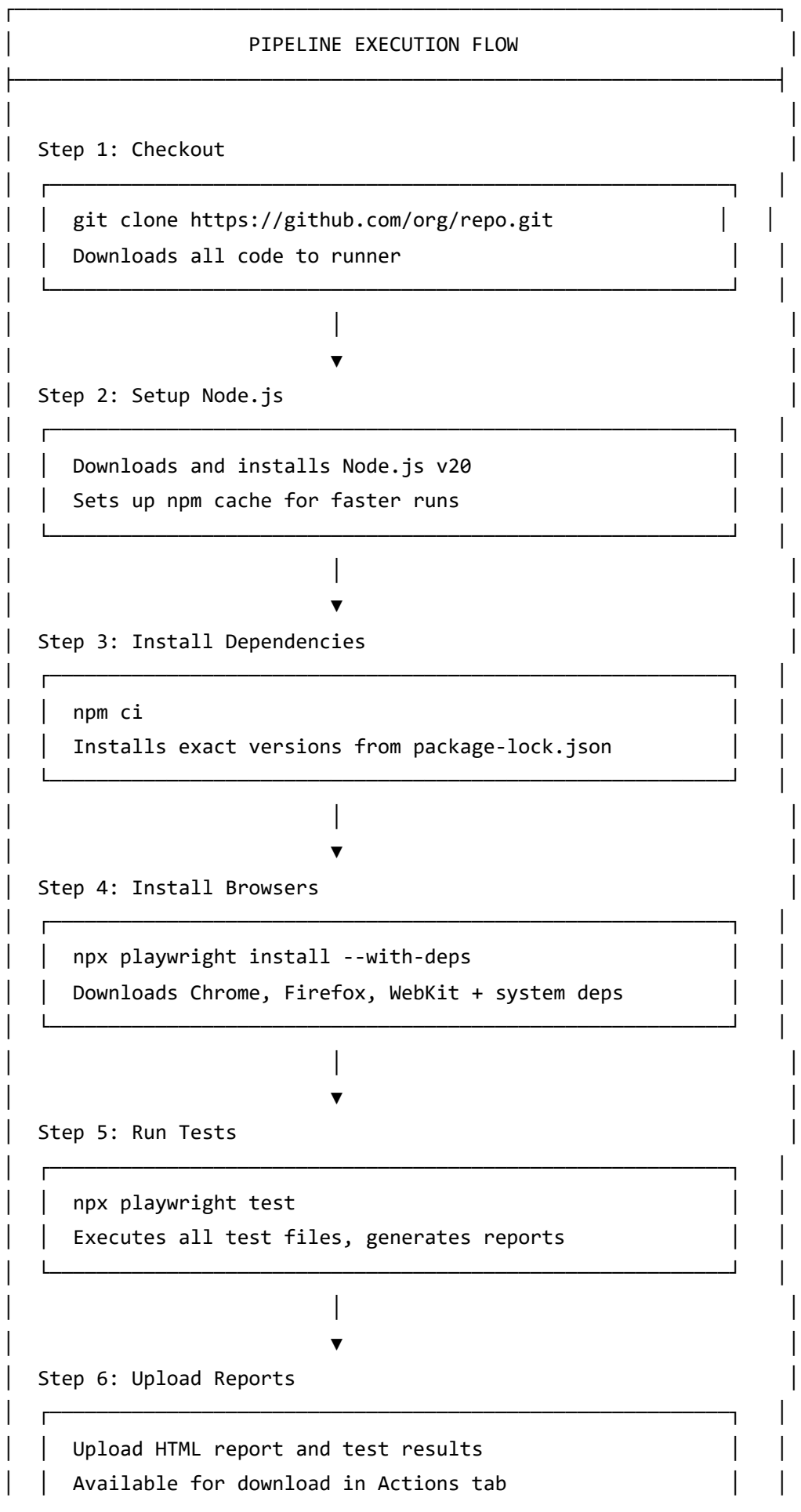
7.1 Complete Pipeline Configuration

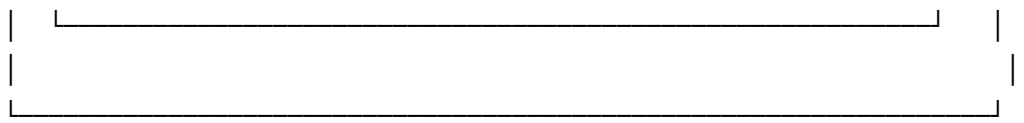
```yaml
# .github/workflows/playwright.yml
name: Playwright Tests

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main, develop]
  workflow_dispatch:
    inputs:
      environment:
        description: 'Test environment'
        required: true
        default: 'staging'
        type: choice
        options:
          - development
          - staging
          - production

jobs:
  test:
    name: Run Playwright Tests
    runs-on: ubuntu-latest
    timeout-minutes: 60

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

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'

      - name: Install dependencies
        run: npm ci
```

```yaml
- name: Install Playwright browsers
  run: npx playwright install --with-deps

- name: Run Playwright tests
  run: npx playwright test
  env:
    BASE_URL: ${{ secrets.BASE_URL }}
    TEST_USERNAME: ${{ secrets.TEST_USERNAME }}
    TEST_PASSWORD: ${{ secrets.TEST_PASSWORD }}

- name: Upload HTML report
  uses: actions/upload-artifact@v4
  if: always()
  with:
    name: playwright-report-${{ github.run_id }}
    path: playwright-report/
    retention-days: 30

- name: Upload test results
  uses: actions/upload-artifact@v4
  if: failure()
  with:
    name: test-results-${{ github.run_id }}
    path: test-results/
    retention-days: 7
```

7.2 Running Tests in Pipeline

Step-by-Step Execution:


PIPELINE EXECUTION FLOW
Step 1: Checkout
git clone https://github.com/org/repo.git
Downloads all code to runner
Step 2: Setup Node.js
Downloads and installs Node.js v20
Sets up npm cache for faster runs
Step 3: Install Dependencies
npm ci
Installs exact versions from package-lock.json
Step 4: Install Browsers
npx playwright install --with-deps
Downloads Chrome, Firefox, WebKit + system deps
Step 5: Run Tests
npx playwright test
Executes all test files, generates reports
Step 6: Upload Reports
Upload HTML report and test results
Available for download in Actions tab



7.3 Generating HTML Reports

Playwright Config for Reports:

```
// playwright.config.ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
    reporter: [
        ['html', {
            open: 'never',
            outputFolder: 'playwright-report'
        }],
        ['list'],                    // Console output
        ['json', {
            outputFile: 'test-results.json'
        }],
        ['junit', {
            outputFile: 'junit-results.xml'
        }]
    ],
});
```

Report Types:

Reporter	Output	Use Case
`html`	Interactive HTML	Human review
`list`	Console text	Quick CI feedback
`json`	JSON file	Integrations
`junit`	XML file	CI tools (Jenkins)

7.4 Uploading Reports as Artifacts

Basic Upload:

```yaml
- name: Upload HTML report
  uses: actions/upload-artifact@v4
  if: always()                    # Upload even if tests fail
  with:
    name: playwright-report
    path: playwright-report/
    retention-days: 30
```

Multiple Artifacts:

```yaml
- name: Upload all reports
  uses: actions/upload-artifact@v4
  if: always()
  with:
    name: test-artifacts
    path: |
      playwright-report/
      test-results/
      videos/
      screenshots/
```

Downloading Artifacts:

1. Go to repository → Actions tab
2. Click on completed workflow run
3. Scroll to "Artifacts" section
4. Click artifact name to download ZIP

7.5 Handling Failures

Conditional Steps Based on Failure:

```yaml
steps:
  - name: Run tests
    id: test
    run: npx playwright test
    continue-on-error: true      # Don't stop pipeline

  - name: Upload failure screenshots
    if: failure()                # Only on failure
    uses: actions/upload-artifact@v4
    with:
      name: failure-screenshots
      path: test-results/**/*.png

  - name: Notify team on failure
    if: failure()
    run: |
      echo "Tests failed! Notifying team..."
      # Add Slack/Teams notification here

  - name: Check test results
    if: always()
    run: |
      if [ "${{ steps.test.outcome }}" == "failure" ]; then
        echo "Some tests failed - check artifacts"
        exit 1
      fi
```

Failure Notification Example (Slack):

```yaml
- name: Notify Slack on failure
  if: failure()
  uses: 8398a7/action-slack@v3
  with:
    status: failure
    text: 'Playwright tests failed on ${{ github.ref }}'
    webhook_url: ${{ secrets.SLACK_WEBHOOK }}
```

7.6 Retrying Flaky Tests

Method 1: Playwright Built-in Retries

```
// playwright.config.ts
export default defineConfig({
    retries: process.env.CI ? 2 : 0,    // 2 retries in CI, 0 locally
});
```

Method 2: Test-Level Retries

```
test('flaky test', async ({ page }) => {
    test.info().config.retries = 3;     // This test gets 3 retries
    // test code
});
```

Method 3: Retry Failed Tests Only

```
- name: Run tests (first attempt)
  id: first-run
  run: npx playwright test
  continue-on-error: true

- name: Retry failed tests
  if: steps.first-run.outcome == 'failure'
  run: npx playwright test --last-failed    # Playwright 1.40+
```

7.7 Pipeline Best Practices

Practice	Description
Use `npm ci`	Faster, reliable dependency install
Cache dependencies	Reduce install time
Set timeouts	Prevent stuck jobs
Upload on `always()`	Get reports even on failure
Use matrix for browsers	Test all browsers
Shard for speed	Split tests across runners

Caching Example:

```yaml
- name: Setup Node.js
  uses: actions/setup-node@v4
  with:
    node-version: 20
    cache: 'npm'          # Caches npm dependencies

- name: Cache Playwright browsers
  uses: actions/cache@v3
  with:
    path: ~/.cache/ms-playwright
    key: playwright-${{ runner.os }}-${{ hashFiles('package-lock.json') }}
```

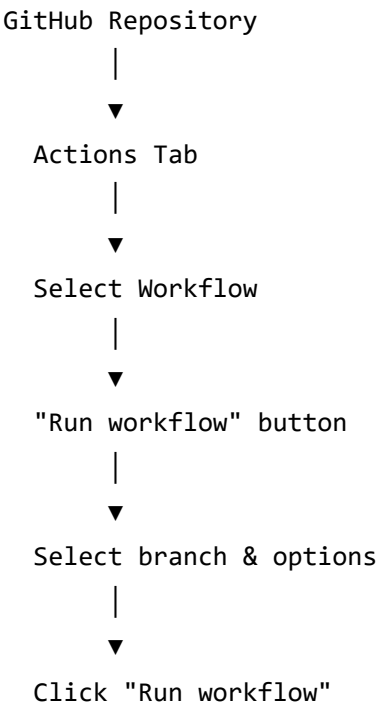
SECTION 8 — How Employees Use CI/CD Pipeline

8.1 How Each Employee Triggers Pipeline

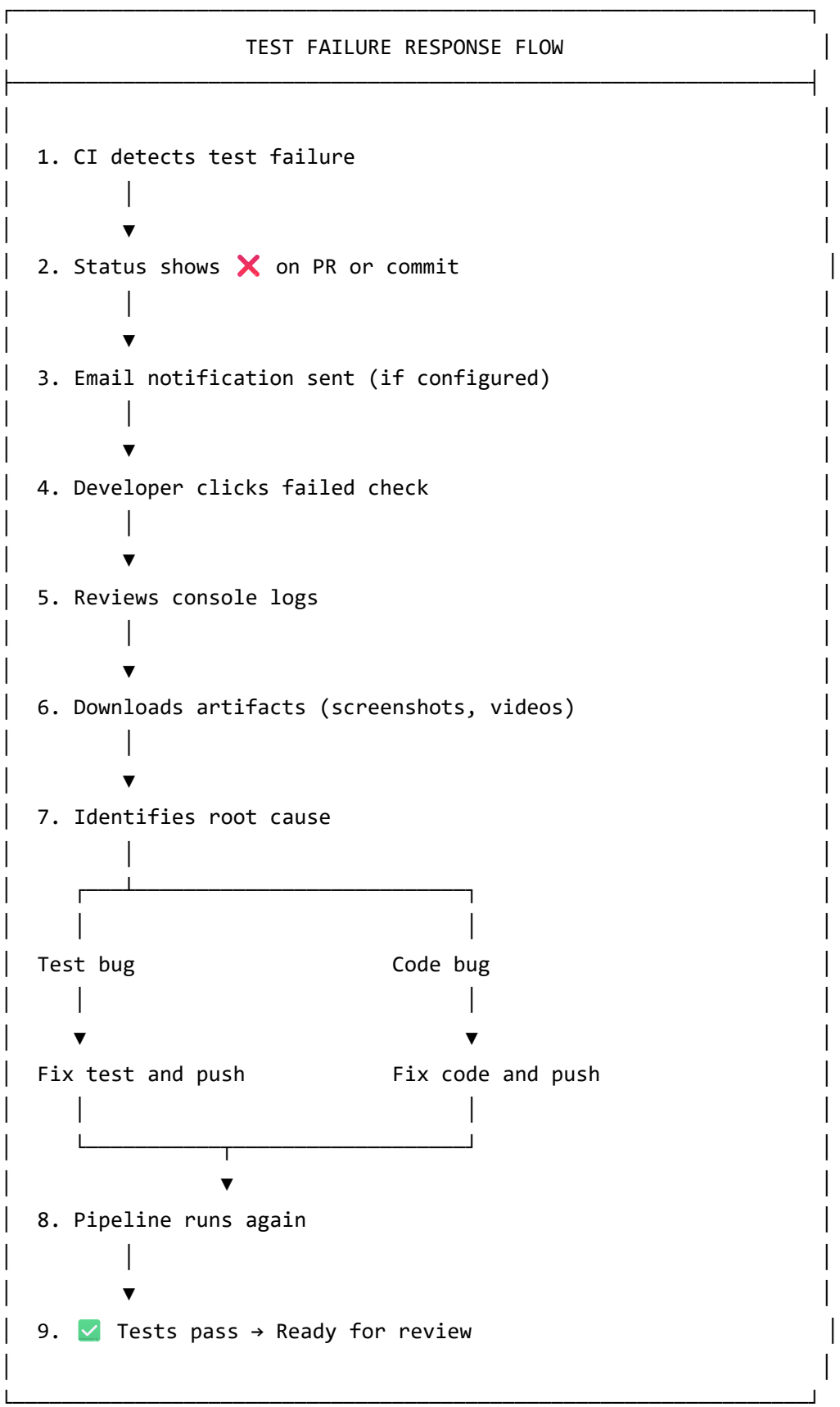
Automatic Triggers:

Action	Trigger	Pipeline Runs
Push to feature branch	`push`	✅ If configured
Create Pull Request	`pull_request`	✅ Yes
Push to develop	`push`	✅ Yes
Merge to main	`push`	✅ Yes

Manual Triggers:


GitHub Repository
Actions Tab
Select Workflow
"Run workflow" button
Select branch & options
Click "Run workflow"

8.2 What Happens When Tests Fail

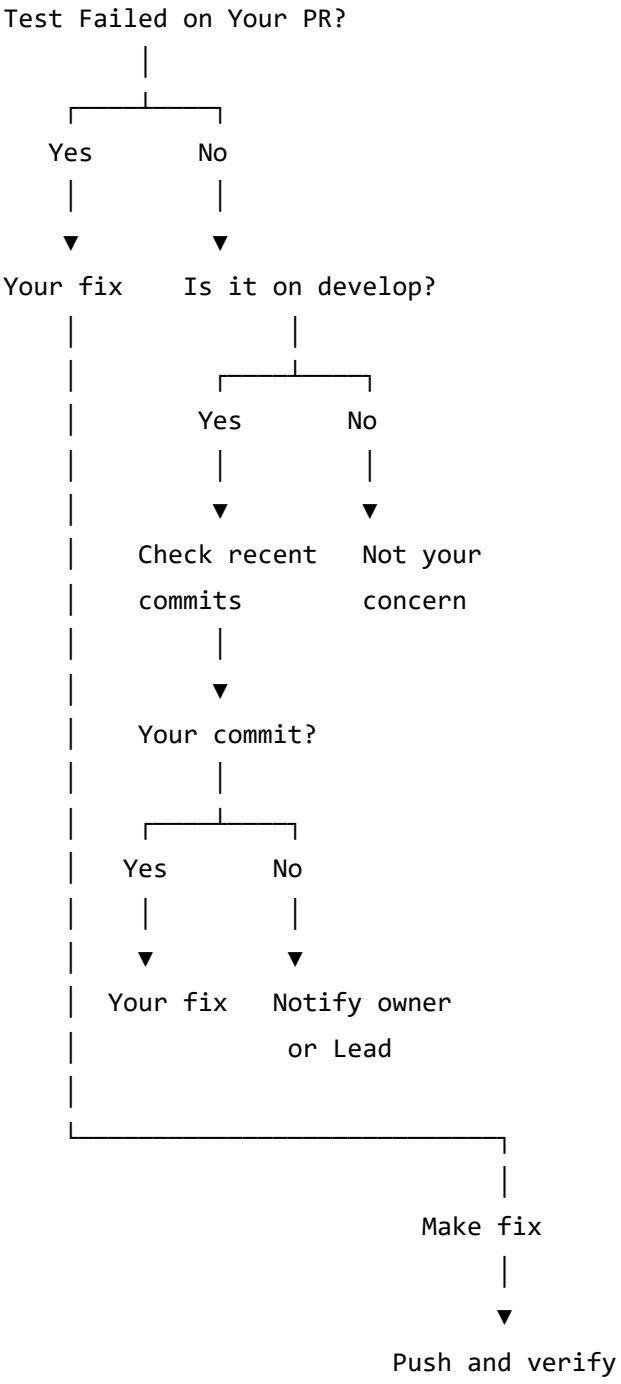


8.3 Who Fixes Failures - Ownership Model

Ownership Rules:

Failure Scenario	Owner	Responsibility
Your PR failed	You	Fix immediately
Develop broken after your merge	You	High priority fix
Develop broken by someone else	That person	You wait or help
Main broken	Automation Lead + Team	Emergency fix
Flaky test failing randomly	Senior Engineer	Investigation

Decision Tree:



8.4 How to Avoid Blocking Others

Best Practices:

Practice	Description
Run tests locally first	Catch failures before push
Fix failures immediately	Don't leave develop broken

Practice	Description
Small PRs	Easier to identify issues
Communicate	Let team know if you broke something
Monitor after merge	Stay online after merging

Anti-Patterns to Avoid:

❌ Don't	✅ Do Instead
Push and leave for the day	Stay to verify pipeline passes
Ignore red pipeline	Fix it or revert
Merge despite failing tests	Wait for green build
Blame others without checking	Investigate your changes first
Skip local testing	Always run `npx playwright test`

8.5 Responding to Pipeline Failures

Step-by-Step Response:

```
# 1. Check if it's your code
git log --oneline -5              # See recent commits

# 2. Run failing tests locally
npx playwright test tests/login.spec.ts --headed

# 3. If you broke it, fix it
# Make your changes...

# 4. Verify fix locally
npx playwright test

# 5. Push the fix
git add .
git commit -m "fix: resolve failing login test"
git push

# 6. Monitor pipeline
# Go to GitHub Actions and watch the run
```

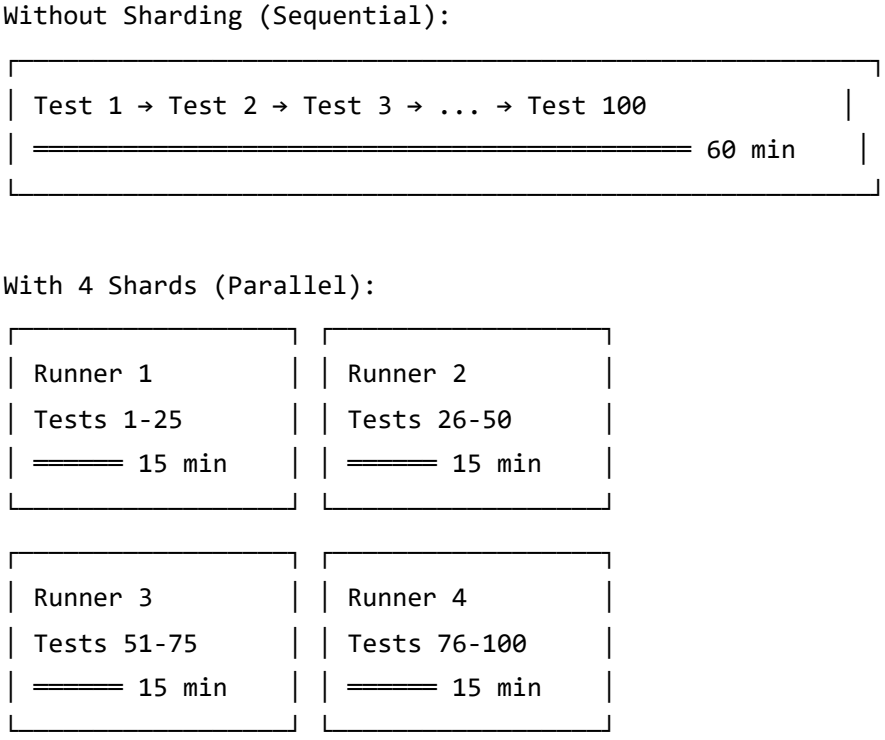
SECTION 9 — Managing CI/CD for Multiple Teams

9.1 Parallel Execution

Running Tests in Parallel:

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false          # Don't cancel other jobs on failure
      matrix:
        shard: [1/4, 2/4, 3/4, 4/4]

    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
      - run: npm ci
      - run: npx playwright install --with-deps
      - run: npx playwright test --shard=${{ matrix.shard }}
```

Visual Explanation:



9.2 Module-Wise Test Execution

Folder Structure by Module:

```
tests/
├── login/              ← Team A
│   ├── login.spec.ts
│   └── login-data.spec.ts
├── registration/       ← Team B
│   ├── register.spec.ts
│   └── register-validation.spec.ts
├── payments/           ← Team C
│   └── checkout.spec.ts
└── reports/            ← Team D
    └── dashboard.spec.ts
```

Running Specific Modules:

```yaml
jobs:
  login-tests:
    runs-on: ubuntu-latest
    steps:
      - run: npx playwright test tests/login/

  payment-tests:
    runs-on: ubuntu-latest
    steps:
      - run: npx playwright test tests/payments/
```

Or Using Matrix:

```yaml
strategy:
  matrix:
    module: [login, registration, payments, reports]

steps:
  - run: npx playwright test tests/${{ matrix.module }}/
```

9.3 Tag-Based Execution

Adding Tags to Tests:

```
// Use annotations in test file
test('login with valid credentials @smoke @login', async ({ page }) => {
    // test code
});

test('complex payment flow @regression @payment', async ({ page }) => {
    // test code
});

test('critical checkout @critical @payment', async ({ page }) => {
    // test code
});
```

Running by Tags:

```
# Run only smoke tests
npx playwright test --grep @smoke

# Run login tests only
npx playwright test --grep @login

# Run everything except slow tests
npx playwright test --grep-invert @slow
```

Pipeline with Tag Selection:

```yaml
on:
  workflow_dispatch:
    inputs:
      test-tag:
        description: 'Test tag to run'
        required: true
        default: '@smoke'
        type: choice
        options:
          - '@smoke'
          - '@regression'
          - '@critical'

jobs:
  test:
    steps:
      - run: npx playwright test --grep ${{ inputs.test-tag }}
```

9.4 Environment Separation

Different Environments:

Environment	Purpose	URL
Development	Latest unstable code	dev.example.com
QA	QA testing	qa.example.com
Staging	Pre-production	staging.example.com
Production	Live site	www.example.com

Configuring Environments in Pipeline:

```yaml
on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Target environment'
        required: true
        type: choice
        options:
          - development
          - qa
          - staging

jobs:
  test:
    environment: ${{ inputs.environment }}
    env:
      BASE_URL: ${{ vars.BASE_URL }}          # Environment-specific
      TEST_USER: ${{ secrets.TEST_USER }}     # Environment-specific

    steps:
      - run: npx playwright test
```

Setting Up GitHub Environments:

1. Go to repository Settings
2. Click "Environments"
3. Create environment (e.g., "staging")
4. Add environment-specific secrets and variables
5. Optionally add protection rules (approvals)

9.5 Secrets Management

Types of Secrets:

Level	Scope	Example Use
Repository	This repo only	API keys for this project
Organization	All org repos	Shared service accounts
Environment	Specific env	Prod vs staging credentials

Setting Repository Secrets:

```
Repository → Settings → Secrets and variables → Actions → New repository secret
```

Secret Best Practices:

✅ Do	❌ Don't
Use secrets for passwords	Hardcode credentials in code
Use environment variables	Log secret values
Rotate secrets regularly	Share secrets in chat
Use least privilege	Give full access
Audit secret access	Ignore who has access

Accessing Secrets:

```yaml
env:
  DB_PASSWORD: ${{ secrets.DB_PASSWORD }}
  API_KEY: ${{ secrets.API_KEY }}
  TEST_USER: ${{ secrets.TEST_USER }}
```

9.6 Access Control

Team-Based Permissions:

Role	Repository Access	Actions Access
Read	View code	View workflow runs
Triage	Issues/PRs	View workflow runs
Write	Push to branches	Trigger workflows
Maintain	Manage settings	Manage secrets
Admin	Full control	Full control

Protected Branches:

```
Settings → Branches → Add branch protection rule

□ Require a pull request before merging
  □ Require approvals (1 or more)
□ Require status checks to pass
  □ Require branches to be up to date
  □ Status checks: "test" (your workflow job)
□ Do not allow bypassing above settings
```

This ensures:

- No direct pushes to main/develop
- Tests must pass before merge
- At least one reviewer approves

Continue to Section 10-13 and Bonus...