Visual Regression Testing with Percy or Playwright

Visual regression testing ensures that the **UI doesn't break unexpectedly** by comparing current screenshots with known-good "baselines". It catches layout shifts, missing elements, font issues, and styling regressions that traditional assertions miss.

Two of the most popular tools in this domain are **Percy** (by BrowserStack) and **Playwright** (with built-in screenshot comparison support). Let's explore how they work and when to use each.

🟢 Option 1: Visual Testing with Percy

✅ Pros:

- Works with multiple test frameworks (Selenium, Cypress, Playwright)
- Handles parallelization and responsive testing
- Offers web dashboard for reviewing diffs
- Integrates well with CI/CD (GitHub Actions, GitLab, Jenkins)

🧪 Example: Percy with Playwright

Install Percy CLI and SDK:

```
npm install --save-dev @percy/cli @percy/playwright
```

Run Percy with Playwright:

```
// tests/visual.spec.js
const { test } = require('@playwright/test');
const percySnapshot = require('@percy/playwright');

test('homepage visual test', async ({ page }) => {
```

```
  await page.goto('http://localhost:3000');
  await percySnapshot(page, 'Homepage Snapshot');
});
```

Run test with Percy:

```
npx percy exec -- npx playwright test
```

Percy will upload screenshots to your Percy dashboard, where you can **review and approve** or **reject** changes.

🧩 CI/CD Integration

Both Percy and Playwright can run in CI:

- **Percy** uses `PERCY_TOKEN` to securely upload screenshots to their dashboard.
- **Playwright** works with any CI tool (e.g., GitHub Actions), and stores screenshots in `test-results` by default.

📌 Summary

Tool	Best For	Notes
Percy	Teams needing review UI, responsive testing	Powerful, polished, requires a cloud token
Playwright (native)	Lightweight local workflows	Fast, minimal setup, best for smaller teams

Visual regression testing is essential for frontend stability and user confidence. Whether you choose Percy for rich dashboards or Playwright for speed and simplicity, integrating these into your test suite pays off in catching UI bugs before they reach users.