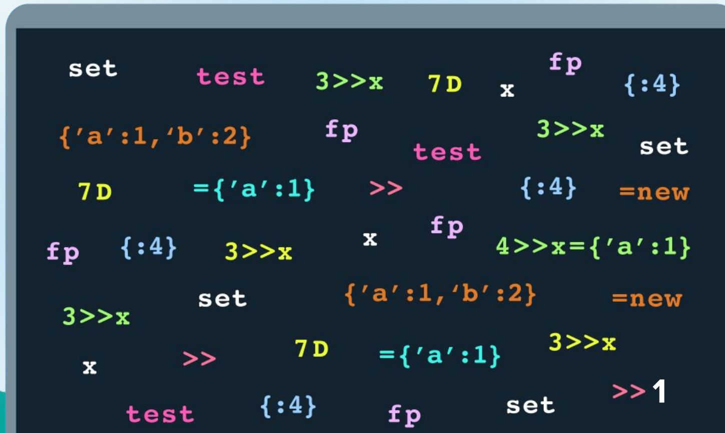
The Magic Of Self-Healing Tests

Execution Cloud



What we'll be looking at today

.01 The Make Up Of A Test

.02 Challenges With Locators

.03 Self-Healing Locators

.04 Live Demos

Each Test Case Has Three Main Parts

Test Data

The properties and objects that make up the data that will be entered into tests

Test Interaction

Mimicking a user by interacting with an application - clicking, typing, navigating

Test Assertion

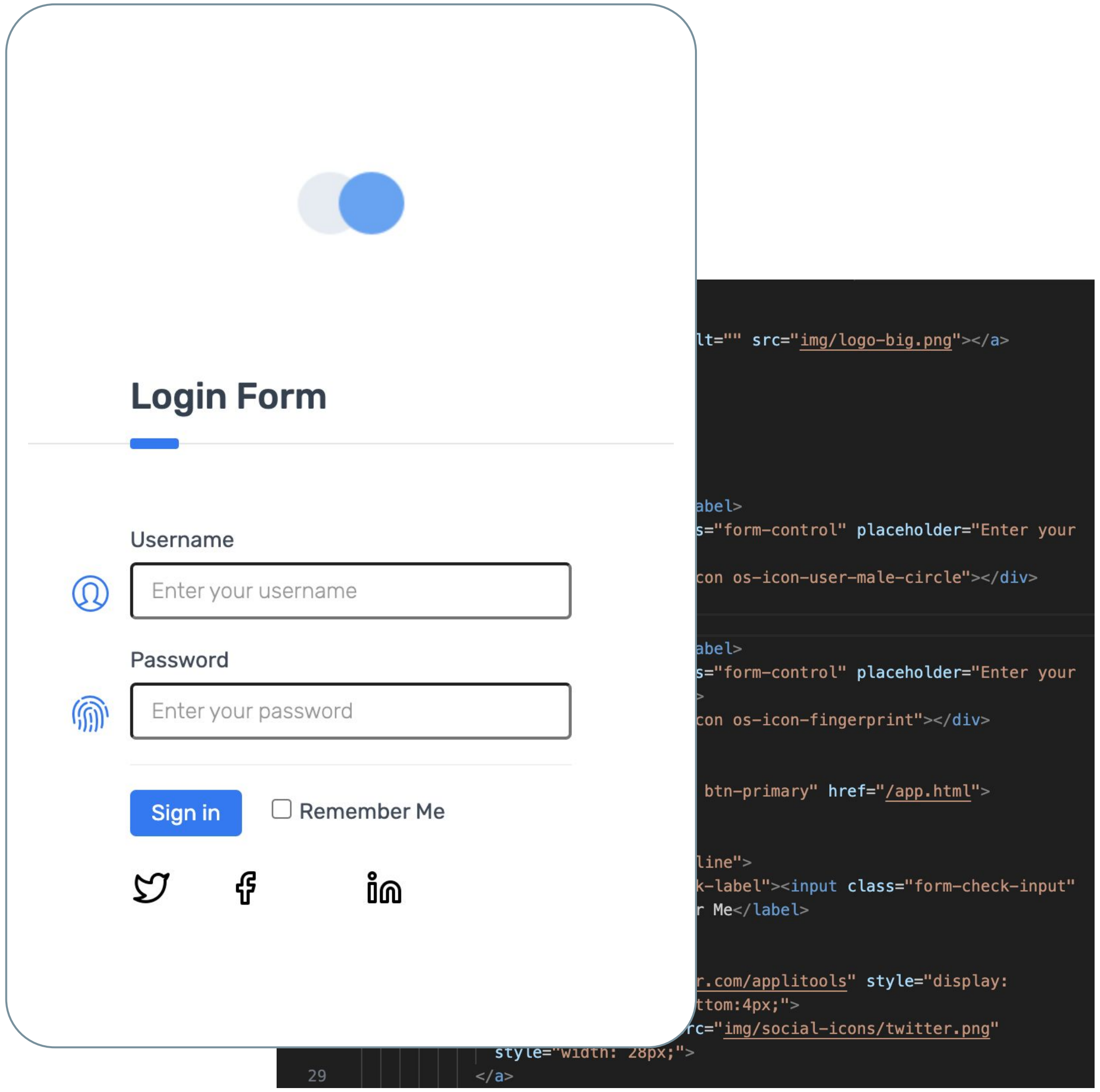
Validating that the interaction produced an outcome that was expected

Our Test Case [In English]

1. Visit Login Page
2. Click the “User Name” form entry
3. Enter in our user name
4. Click the “Password” form entry
5. Enter in our password
6. Click the blue submit button
7. Make sure dashboard URL loaded

$\left.\begin{matrix}1\\2\\3\\4\\5\\6\end{matrix}\right\}$ 85% of our test case is simple interaction with the app

But in order to “click” this blue Sign In button, we must rely on the DOM.

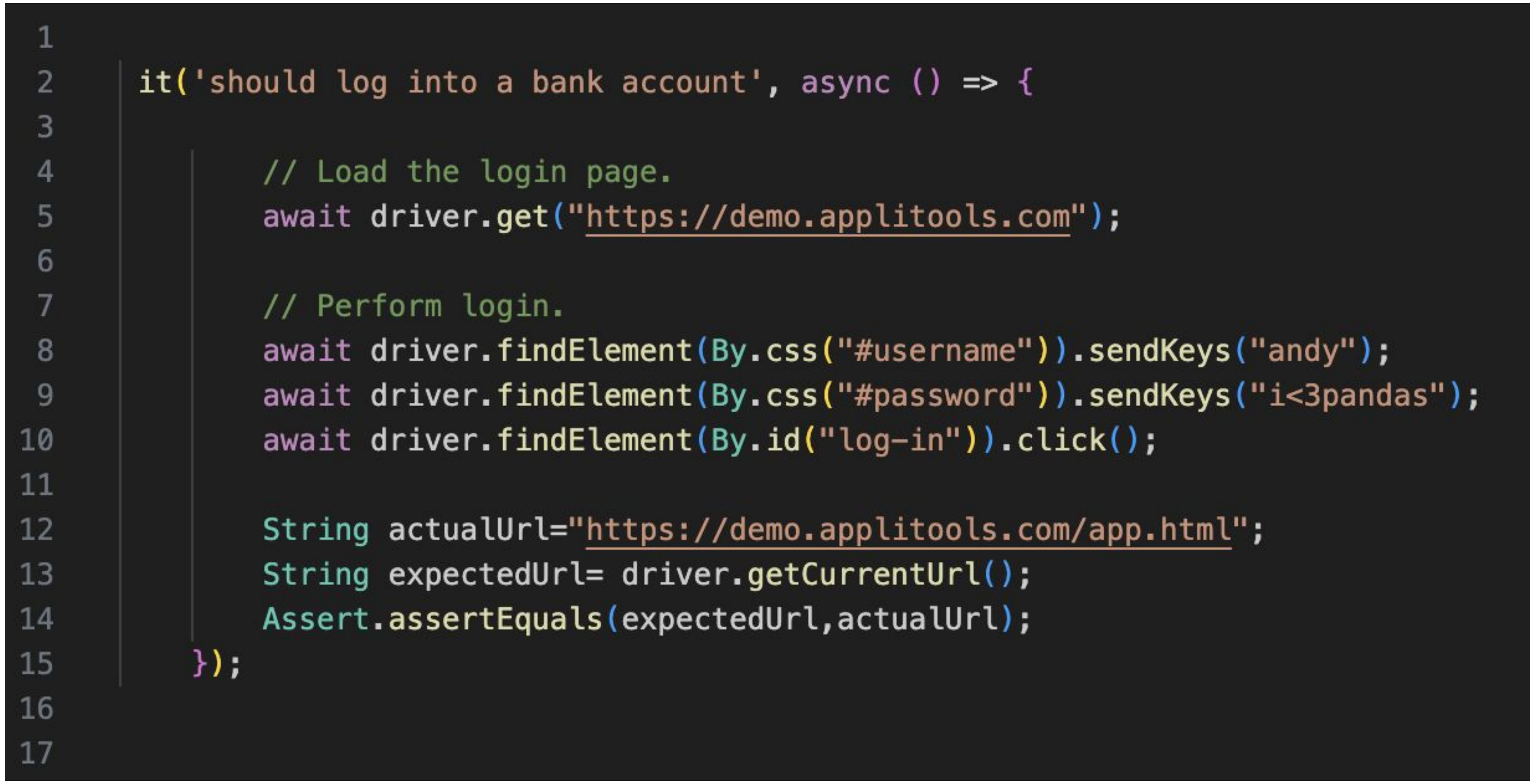


```
lt="" src="img/logo-big.png"></a>

abel>
s="form-control" placeholder="Enter your

con os-icon-user-male-circle"></div>

abel>
s="form-control" placeholder="Enter your
>
con os-icon-fingerprint"></div>

 btn-primary" href="/app.html">

line">
k-label"><input class="form-check-input"
r Me</label>

r.com/applitools" style="display:
ttom:4px;">
rc="img/social-icons/twitter.png"
  style="width: 28px;">
</a>
```

Our Test Case [Code]

Riddled with Locators from the DOM

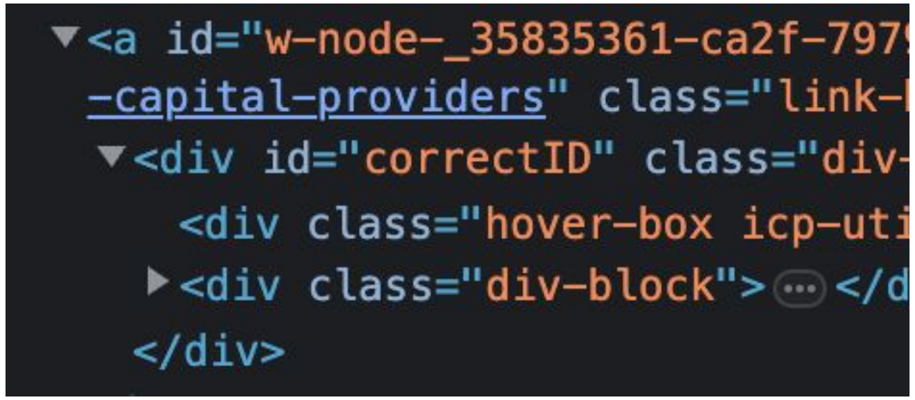
- Primary cause for flaky tests and delayed feedback
- Extremely high test maintenance overhead
- Archaic way to select an element



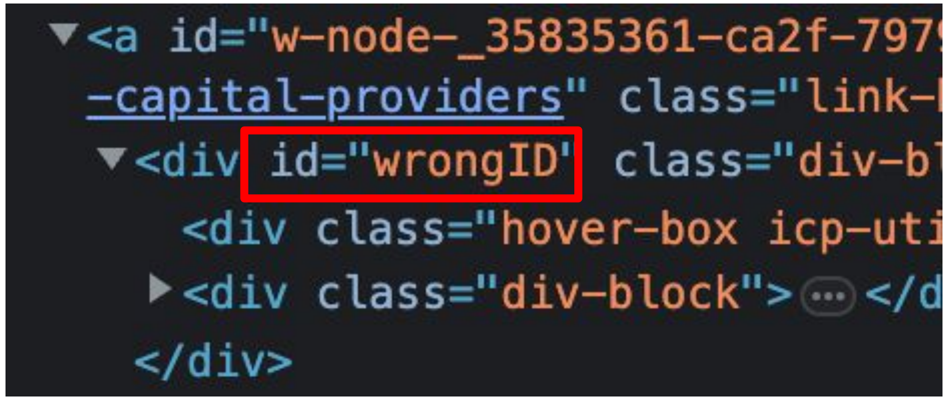
```

it('should log into a bank account', async () => {

      // Load the login page.
      await driver.get("https://demo.applitools.com");

      // Perform login.
      await driver.findElement(By.css("#username")).sendKeys("andy");
      await driver.findElement(By.css("#password")).sendKeys("i<3pandas");
      await driver.findElement(By.id("log-in")).click();

      String actualUrl="https://demo.applitools.com/app.html";
      String expectedUrl= driver.getCurrentUrl();
      Assert.assertEquals(expectedUrl,actualUrl);
  });


```

applitools

Test flakiness comes from problems with Interaction, **not** Verification.



```
▼<a id="w-node-_35835361-ca2f-797
  -capital-providers" class="link-
  ▼<div id="correctID" class="div-
     <div class="hover-box icp-uti
    ▶<div class="div-block">⋯</d
   </div>
```



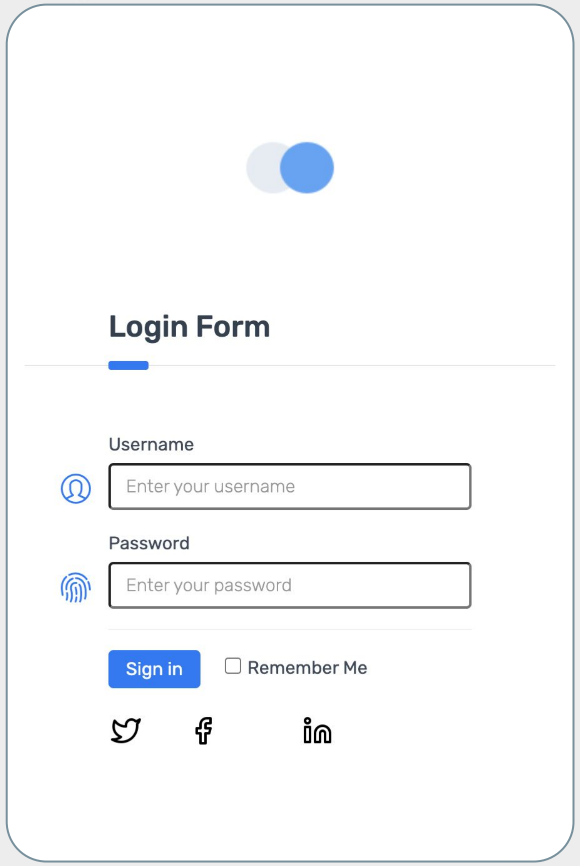
```
▼<a id="w-node-_35835361-ca2f-797
  -capital-providers" class="link-
  ▼<div id="wrongID" class="div-bl
     <div class="hover-box icp-uti
    ▶<div class="div-block">⋯</d
   </div>
```

NoSuchElementException

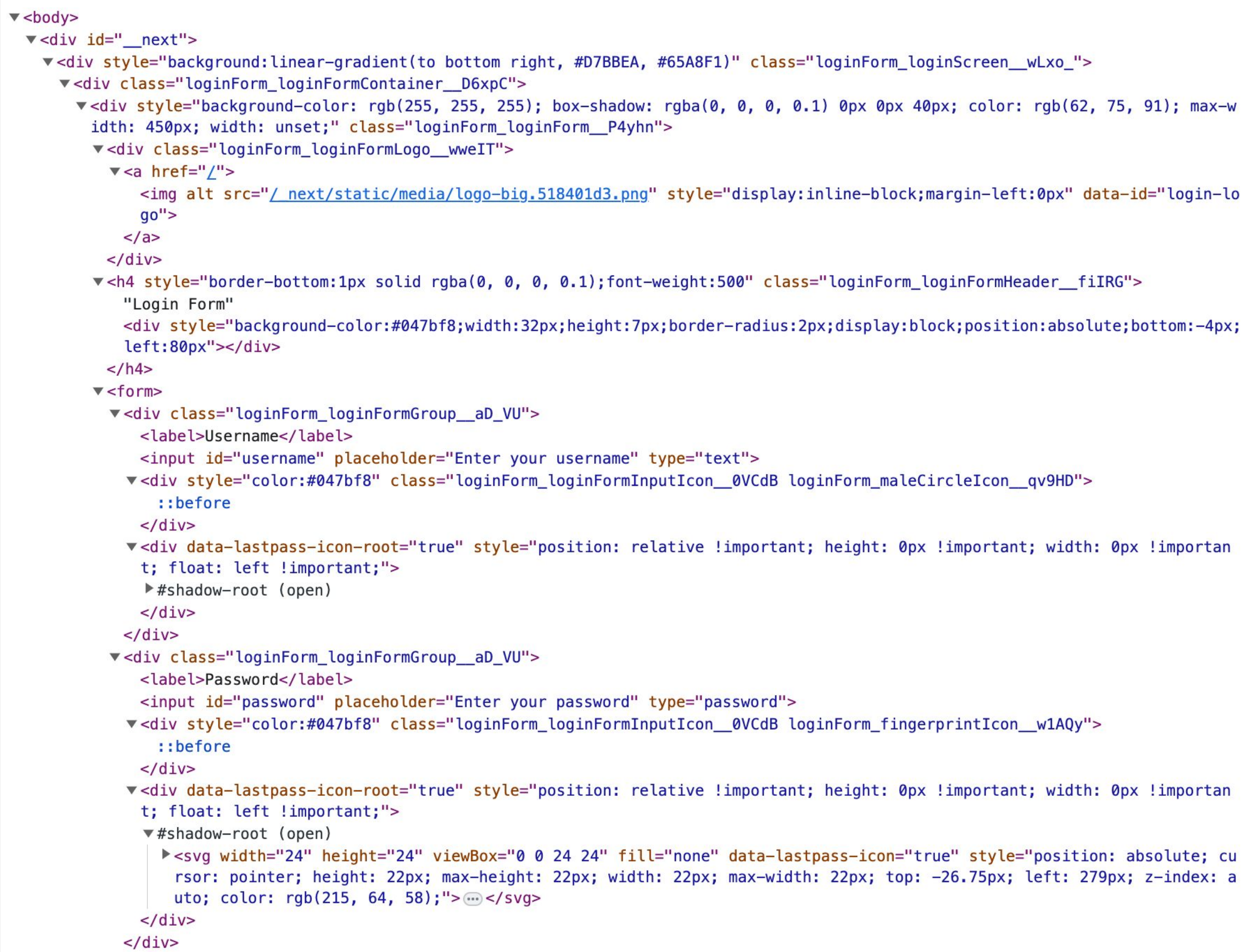
This small change from the dev team will fail the entire test case - even if nothing **actually changed** in the UI.

applitools

Because testing tools “see” like a…

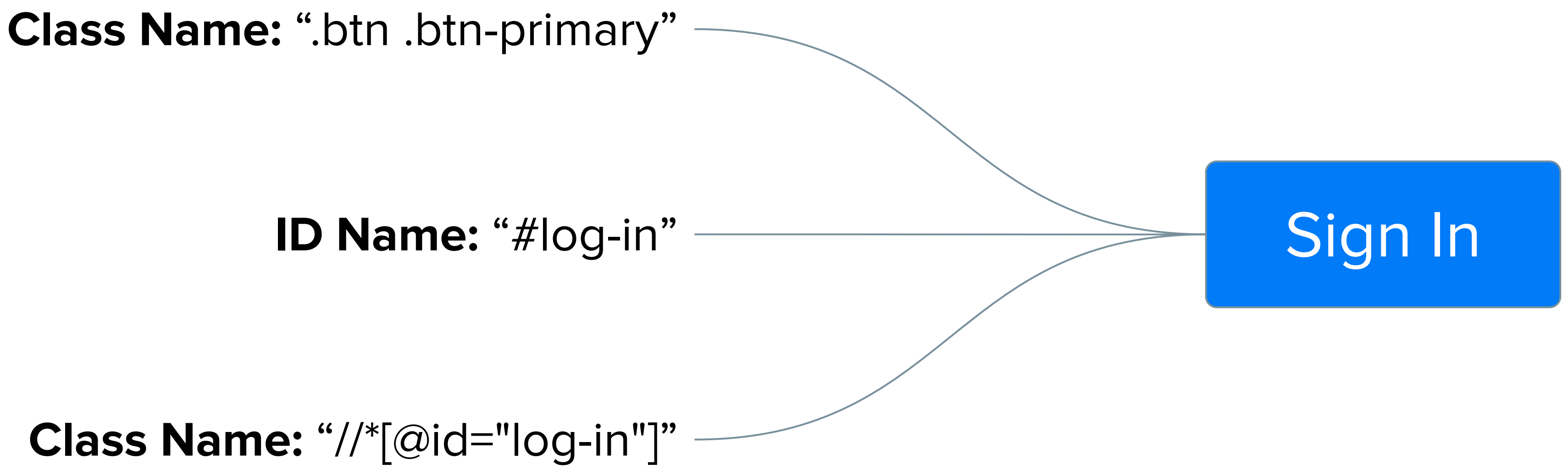


computer.

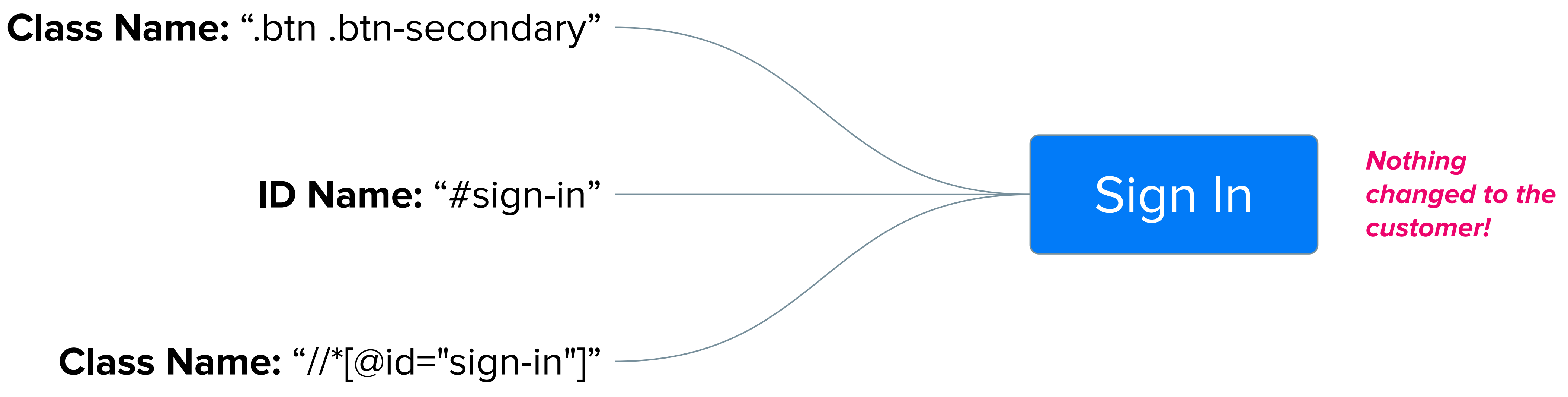


```
▼<body>
 ▼<div id="__next">
  ▼<div style="background:linear-gradient(to bottom right, #D7BBEA, #65A8F1)" class="loginForm_loginScreen__wLxo_">
   ▼<div class="loginForm_loginFormContainer__D6xpC">
    ▼<div style="background-color: rgb(255, 255, 255); box-shadow: rgba(0, 0, 0, 0.1) 0px 0px 40px; color: rgb(62, 75, 91); max-w
     idth: 450px; width: unset;" class="loginForm_loginForm__P4yhn">
     ▼<div class="loginForm_loginFormLogo__wweIT">
      ▼<a href="/">
        <img alt src="/_next/static/media/logo-big.518401d3.png" style="display:inline-block;margin-left:0px" data-id="login-lo
        go">
       </a>
      </div>
     ▼<h4 style="border-bottom:1px solid rgba(0, 0, 0, 0.1);font-weight:500" class="loginForm_loginFormHeader__fiIRG">
       "Login Form"
       <div style="background-color:#047bf8;width:32px;height:7px;border-radius:2px;display:block;position:absolute;bottom:-4px;
       left:80px"></div>
      </h4>
     ▼<form>
      ▼<div class="loginForm_loginFormGroup__aD_VU">
        <label>Username</label>
        <input id="username" placeholder="Enter your username" type="text">
       ▼<div style="color:#047bf8" class="loginForm_loginFormInputIcon__0VCdB loginForm_maleCircleIcon__qv9HD">
         ::before
        </div>
       ▼<div data-lastpass-icon-root="true" style="position: relative !important; height: 0px !important; width: 0px !importan
        t; float: left !important;">
        ▶#shadow-root (open)
        </div>
       </div>
      ▼<div class="loginForm_loginFormGroup__aD_VU">
        <label>Password</label>
        <input id="password" placeholder="Enter your password" type="password">
       ▼<div style="color:#047bf8" class="loginForm_loginFormInputIcon__0VCdB loginForm_fingerprintIcon__w1AQy">
         ::before
        </div>
       ▼<div data-lastpass-icon-root="true" style="position: relative !important; height: 0px !important; width: 0px !importan
        t; float: left !important;">
        ▼#shadow-root (open)
         ▶<svg width="24" height="24" viewBox="0 0 24 24" fill="none" data-lastpass-icon="true" style="position: absolute; cu
          rsor: pointer; height: 22px; max-height: 22px; width: 22px; max-width: 22px; top: -26.75px; left: 279px; z-index: a
          uto; color: rgb(215, 64, 58);">⋯</svg>
        </div>
       </div>
```

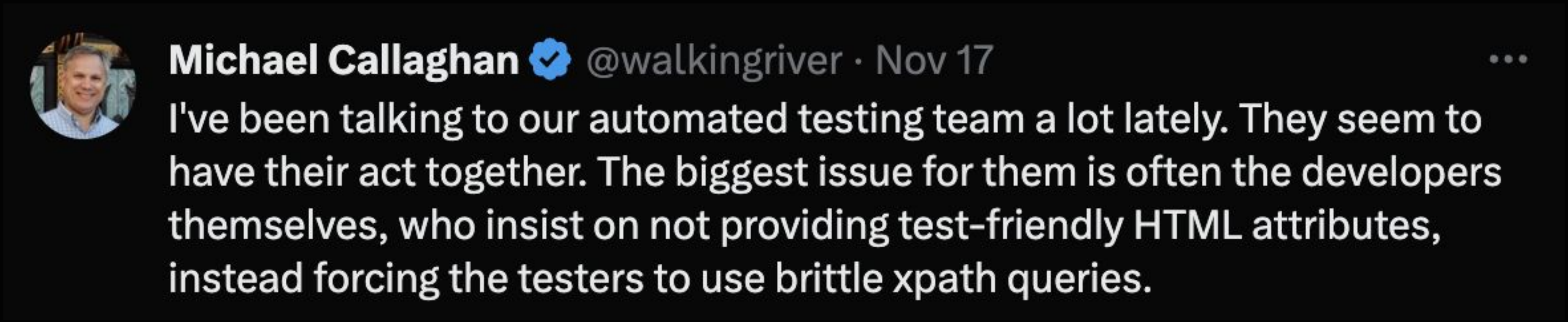
Version A


Class Name: “.btn .btn-primary”
ID Name: “#log-in”
Class Name: “//*[@id="log-in"]”
Sign In

Version B


Class Name: “.btn .btn-secondary”
ID Name: “#sign-in”
Class Name: “//*[@id="sign-in"]”
Sign In
Nothing changed to the customer!

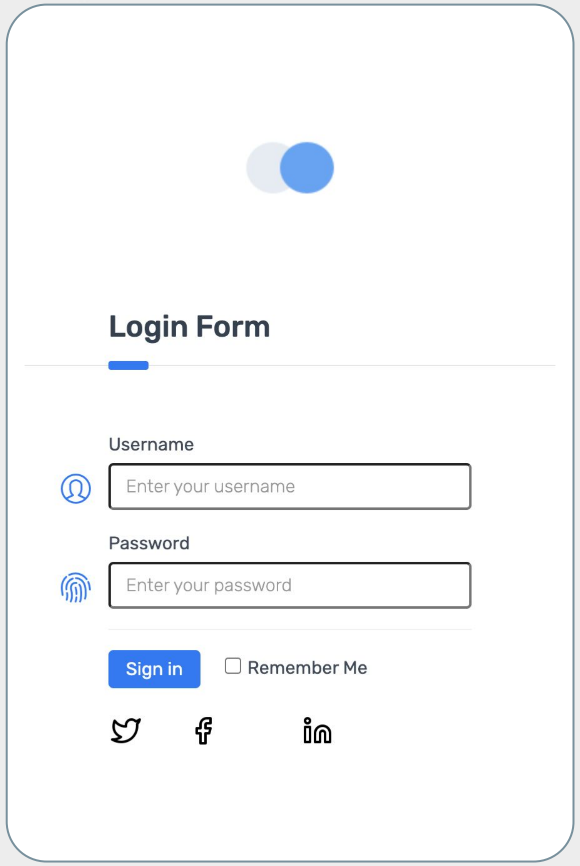
Our test fails now!



Michael Callaghan @walkingriver · Nov 17

I've been talking to our automated testing team a lot lately. They seem to have their act together. The biggest issue for them is often the developers themselves, who insist on not providing test-friendly HTML attributes, instead forcing the testers to use brittle xpath queries.

Applitools sees your application like...


Login Form

Username

Enter your username

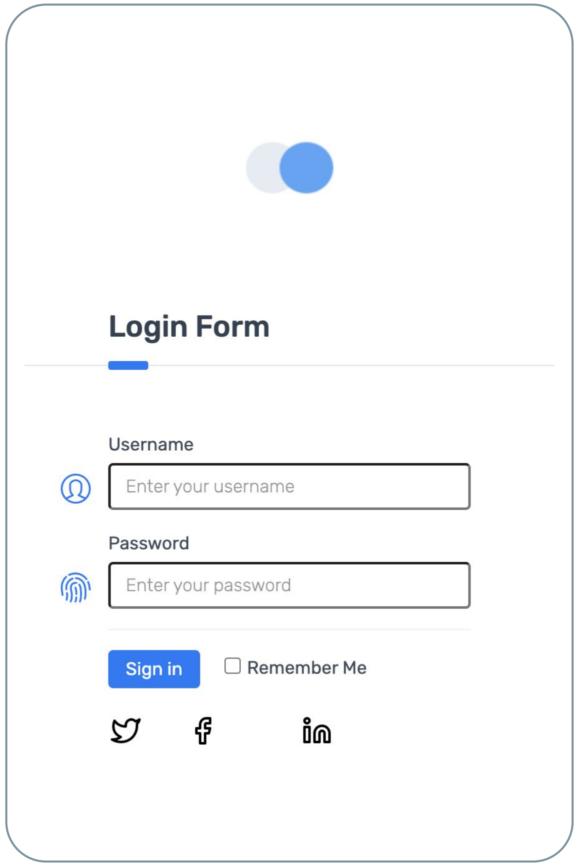
Password

Enter your password

Sign in

Remember Me

Your **users**.


Login Form

Username

Enter your username

Password

Enter your password

Sign in

Remember Me

Applitools Execution Cloud

The first self-healing test infrastructure for open-source frameworks

Applitools Execution Cloud is a cloud-based testing platform that enables teams to run their tests with open source frameworks, **with or without Eyes Checkpoints added,** on AI-powered testing infrastructure. Our intelligent infrastructure can run tests in parallel and self-heal tests that break due to flaky locators.

ACCELERATE TESTING

Launching and running your tests in the cloud at infinite scale reduces testing time and infrastructure maintenance.

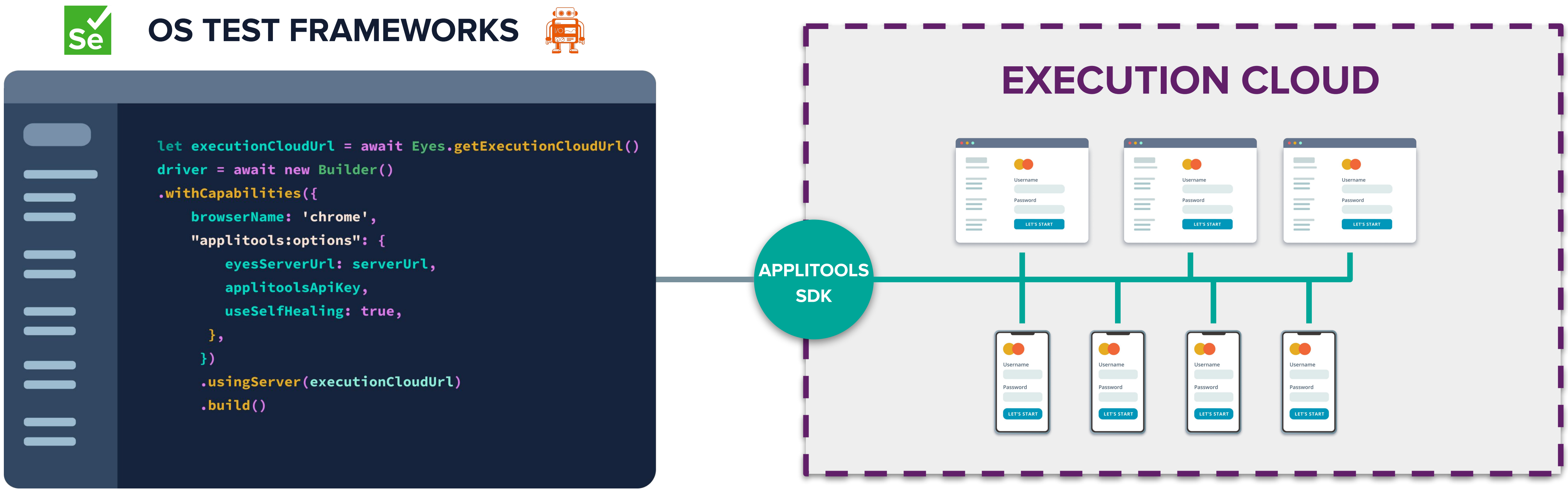
SELF-HEAL TESTS

Self-heals tests from open source frameworks that fail from changing locators used during navigation so less tests fail.

IMPROVE COVERAGE

Run tests with or without Eyes, in parallel, on the Execution Cloud to improve your overall test coverage.

Quickly integrate your framework and run tests in Parallel



```
let executionCloudUrl = await Eyes.getExecutionCloudUrl()
driver = await new Builder()
.withCapabilities({
    browserName: 'chrome',
    "applitools:options": {
        eyesServerUrl: serverUrl,
        applitoolsApiKey,
        useSelfHealing: true,
      },
     })
     .usingServer(executionCloudUrl)
     .build()
```

"The ability to run tests at scale in the cloud and leverage AI capabilities represents a major shift in the way that testing is done"

Mike Millgate - Director of QE @ EverFi

How does self-healing work?

- Every time we find and element
  - **Capture hundreds of data points about the element**
    - All attributes, location in hierarchy, details of ancestor and neighbor elements
  - **Store data in a DB using the locator as key**
- When we can’t locate an element using a given locator
  - **Retrieve information from the DB using the failed locator**
  - **Use proprietary algorithms to find the element based on that information**
  - **If successful, update the DB and suggest a new locator in our dashboard**

What can we heal?

- We can find an element even if simultaneously
  - **Element properties change (e.g., ID, class, tag name, custom, etc)**
  - **Text changes (clickable text, input value, label, placeholder)**
  - **DOM position changes (hierarchy, position in list)**
  - **Size and location changes**
- Adaptive
- Implicitly wait for elements

When is self-healing useful

- Avoid test failures and test maintenance following UI changes
- UI pages that frequently change
- Poor locator authoring skills
- Apps with weak locators
- Apps with dynamically generated UI (dynamic ids)

Let’s see it live!