Automation for Web

Group 07

Table of contents

01

INTRODUCTION

What is Automation Testing?

- A software testing technique that uses specialized tools and scripts to automatically execute test cases
- Compare actual outcomes with expected results, and generate detailed reports
- Does not require human intervention during execution

Why Automation?

Speed: Test suites can be executed much faster than manual testing.

Accuracy: Eliminates human errors.

Efficiency: Reduces manual effort, saving time for testers to focus on complex test scenarios.

Reusability: Test scripts can be reused across multiple versions or environments.

Coverage: Allows running a large number of test cases.

Which test cases to automate?

- High-risk or business-critical workflows
- Repetitive regression tests
- Data-intensive tests
- Cross-browser or cross-platform scenarios
- Time-consuming manual processes

Process of Automation Testing

Test Tool Selection

Choose suitable tools based on project requirements (e.g., Selenium for web, Appium for mobile).

Define Scope of Automation

Identify test cases that provide maximum ROI.

Three ideas

Planning, Design, and Development

Create automation strategy, test scripts, and environment setup.

Test Execution

Run automated tests and collect reports.

Maintenance

Update scripts as application changes to ensure accuracy.

Automation Testing Frameworks

- **Linear/Record & Playback:** Easiest to use, minimal coding.
- **Modular/Component:** Breaks tests into reusable modules.
- **Data-Driven:** Separates test data from scripts.
- **Keyword-Driven:** Uses keywords to represent actions.
- **Hybrid:** Combines multiple frameworks for flexibility.
- **BDD (Behavior-Driven Development):** Uses natural language syntax to improve collaboration between testers and developers.

Popular Automation Tools

- **Selenium:** Open-source tool for web application testing.
- **Appium:** Mobile automation framework supporting Android and iOS.
- **TestNG, JUnit:** Testing frameworks for managing test execution.
- **Cypress, Playwright:** Modern automation tools with faster execution and better debugging.

02

SELENIUM

A Quick Walkthrough

2004 - Birth of Selenium

- Created by Jason Huggins.
- Original Name: "JavaScript TestRunner".
- Renamed: Selenium Core.

2006 - The Evolution

- Selenium RC was developed.
- Solved a major security issue (cross-domain policy) that limited Selenium Core.

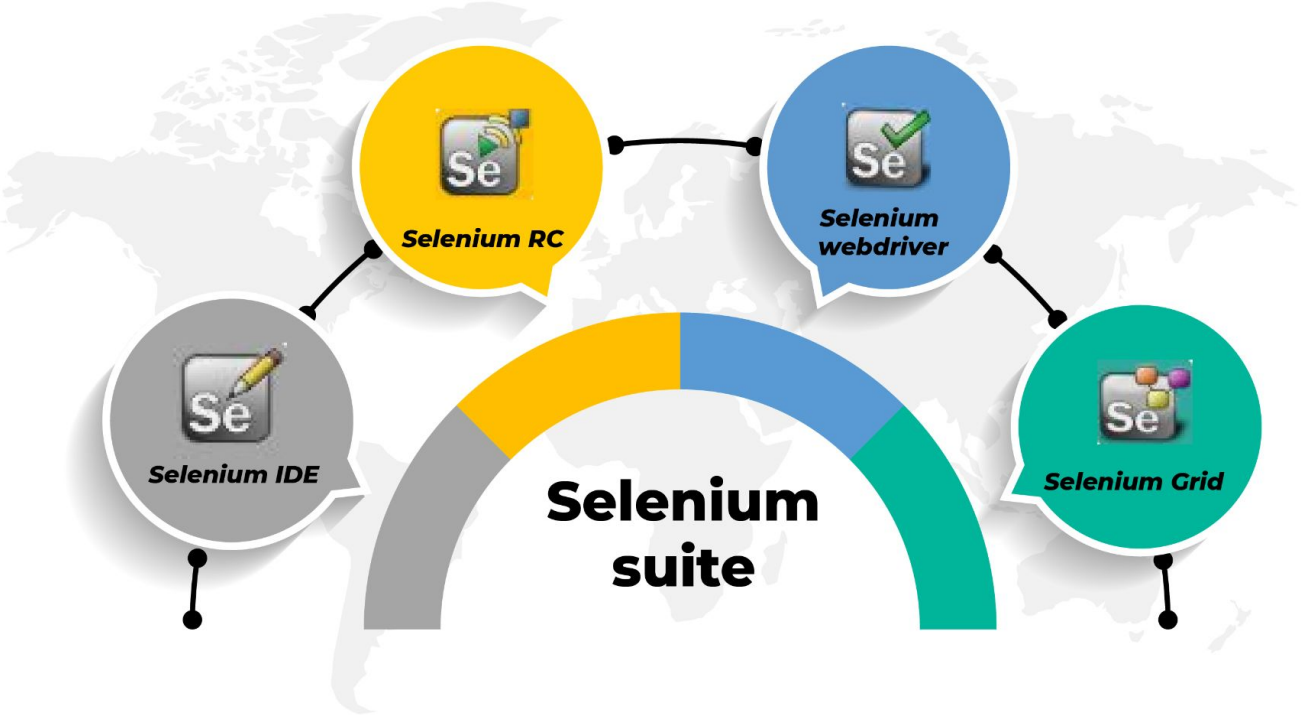
2008 & Beyond: The Modern Era

- Selenium 1 merged with a newer, faster project called "WebDriver".
- This created Selenium 2, which evolved into the Selenium WebDriver (3 & 4) we use today.

What is *Selenium*?

- An open-source framework for automating web browsers.
- Selenium is not just one tool or API, it comprises many tools.
- Simulates a real user interacting with a web application: clicks buttons, fills forms, navigates pages, verifies results, etc.

Selenium Suite



Environment Setup (Python)

1. Install Python
   - Ensure Python is installed on your system (from python.org).
2. Install Required Libraries
   - Need two main packages

Install Libraries

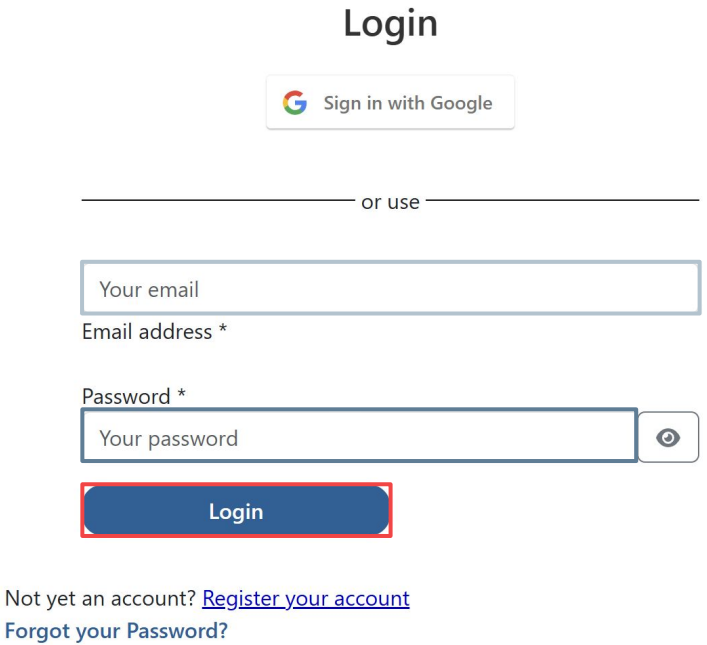
```
# Installs the main Selenium library
pip install selenium

# Installs the manager to auto-download the correct browser driver
pip install webdriver-manager
```

How Selenium Finds Web Elements

Selenium needs to identify and interact with web elements (such as buttons, inputs, links, etc).

→ To do this, Selenium uses **locators**.



What are *Locators?*

A locator is a way to identify elements on a page. It is the argument passed to the finding element methods.

Example of interaents

Locator

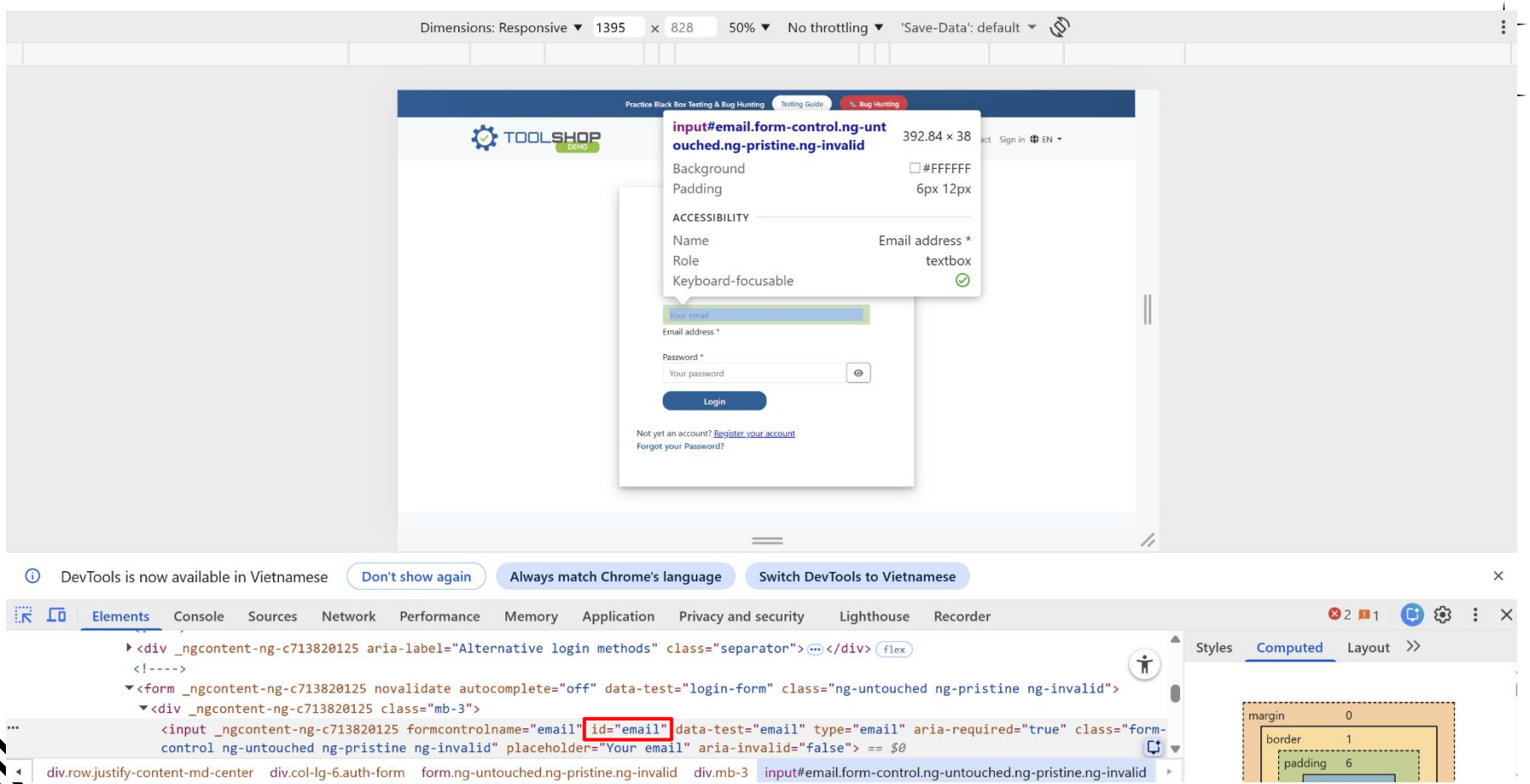
```
email_input = driver.find_element(By.ID, "email")
password_input = driver.find_element(By.NAME, "password")
login_button = driver.find_element(By.CSS_SELECTOR, "button.login-btn")
```

Common Selenium Locators

Locator	Description
class name	Locates elements whose **class name** contains the search value
css selector	Locates elements matching a **CSS selector**
id	Locates elements whose **ID attribute** matches the search value
name	Locates elements whose **NAME attribute** matches the search value

Common Selenium Locators

Locator	Description
link text	Locates **anchor (<a>) elements** whose **visible text** matches the search value
partial link text	Locates **anchor (<a>) elements** whose **visible text contains** the search value (only the **first match** is selected)
tag name	Locates elements whose **HTML tag name** matches the search value
xpath	Locates elements matching an **XPath expression**


Dimensions: Responsive
1395
828
50%
No throttling
'Save-Data': default
Practice Black Box Testing & Bug Hunting
Testing Guide
Bug Hunting
TOOLSHOP
DEMO
input#email.form-control.ng-untouched.ng-pristine.ng-invalid
392.84 × 38
Background
#FFFFFF
Padding
6px 12px
ACCESSIBILITY
Name
Email address *
Role
textbox
Keyboard-focusable
Your email
Email address *
Password *
Your password
Login
Not yet an account? Register your account
Forgot your Password?
DevTools is now available in Vietnamese
Don't show again
Always match Chrome's language
Switch DevTools to Vietnamese
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
<div _ngcontent-ng-c713820125 aria-label="Alternative login methods" class="separator">...
flex
<!---->
<form _ngcontent-ng-c713820125 novalidate autocomplete="off" data-test="login-form" class="ng-untouched ng-pristine ng-invalid">
<div _ngcontent-ng-c713820125 class="mb-3">
<input _ngcontent-ng-c713820125 formcontrolname="email" id="email" data-test="email" type="email" aria-required="true" class="form-control ng-untouched ng-pristine ng-invalid" placeholder="Your email" aria-invalid="false"> == $0
div.row.justify-content-md-center
div.col-lg-6.auth-form
form.ng-untouched.ng-pristine.ng-invalid
div.mb-3
input#email.form-control.ng-untouched.ng-pristine.ng-invalid
Styles
Computed
Layout
margin
0
border
1
padding
6

Interacting
with web elements

Action	Description
Click	Click the element
Send keys	Type text or send keyboard input
Clear	Reset the content of an element
Submit	Submit a form
Select	Select options from dropdowns

Waits – An essential concept

Problem: Your automation script often runs **much faster than** the browser can load or render elements.

→ **NEED TO USE WAITS**

▶▶▶

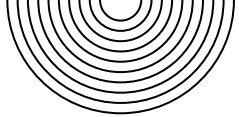
Why *Thread.sleep()* is a bad idea?

- Hard wait: Thread.sleep() in Java or **time.sleep()** in Python
- It pauses execution for a fixed amount of time

Ex: time.sleep(5)

=> Meaning: "Pause the entire script for exactly 5 seconds, no matter what."

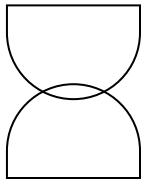
▶▶▶



Why *Thread.sleep()* is a bad idea?

1. **Slow:** If the element appears in 1 second, your script wastes 4 useless seconds. (100 tests = 400+ wasted seconds).

2. **Unreliable** (Flaky): If the network is slow and the element takes 6 seconds to load, your script waits 5s, wakes up, finds nothing, and still FAILS.



▶▶▶

The Good Way: Implicit Wait

Concept: You tell the driver one time: *"Hey, any time you use find_element and can't find something, please keep retrying for up to X seconds before you give up."*

Implicit Wait

```
# Set a global implicit wait for the entire driver session
driver.implicitly_wait(10)

# Selenium will keep retrying this for up to 10 seconds
username_input = driver.find_element(By.ID, "username")
username_input.send_keys("user123")
```

The Good Way: Implicit Wait

Pro: Very easy, solves many basic loading issues.

Con: Only checks one condition: "Is the element present in the HTML?" - It doesn't care if the element is visible or clickable.

The Best Way: Explicit Wait

Concept: "Wait for a specific condition to be true, for a maximum amount of time."

- You create a wait object (e.g., WebDriverWait(driver, 10) for a 10s max wait).

- You tell the wait object to wait .until() your Expected Condition is met.

Explicit Wait

Explicit Wait

```python
# Create a "waiter" object with a 10-second timeout
wait = WebDriverWait(driver, 10)

try:
    # EXAMPLE 1: Wait until a button is CLICKABLE
    # This is the best way to wait for a button!
    print("Waiting for the submit button...")
    submit_button = wait.until(
        EC.element_to_be_clickable( (By.ID, "submit-id") )
    )
    # Code continues ONLY when the button is clickable
    submit_button.click()
    print("Button clicked.")

    # EXAMPLE 2: Wait until an error message is VISIBLE
    print("Waiting for error message to appear...")
    error_message = wait.until(
        EC.visibility_of_element_located( (By.CLASS_NAME, "error-text") )
    )
    print(f"Found error: {error_message.text}")

except Exception as e:
    # This is a TimeoutException
    print(f"Test failed: Element was not ready in 10 seconds. {e}")
```

POM

Introduction

A design pattern that encourages creating **separate classes for each page** of the application.

Each class holds **locators** and **methods/actions** for that page.

Example Structure

```
LoginPage.py        → locators + login actions
HomePage.py         → locators + actions for home page
BasePage.py         → common methods (click, send_keys, wait)
Tests/
    test_login.py   → calls methods from LoginPage
```

Data-Driven Testing Framework

A framework where test data is separated from test scripts, allowing tests to run with multiple sets of input without rewriting the code.

Sources of data: CSV, Excel, JSON, database, or even hardcoded lists.

Benefits:

- Reduces duplicate test scripts
- Makes adding new test scenarios easier
- Improves test coverage

Cross-Browser Testing

- Users access web apps via various browsers (Chrome, Firefox, Safari, Edge) and OS versions.
- Different Rendering Engines (Blink, Gecko, WebKit) interpret CSS/JS differently.

-> *A feature working on Chrome might break on Safari.*

- Need to ensure consistent functionality and UX for all users, regardless of their browser choice.

Handling Multiple Browsers with Selenium

- Write one test script (Java/Python), run it anywhere.
- Selenium interacts with specific browsers via Drivers:
  - *ChromeDriver* -> Google Chrome
  - *GeckoDriver* -> Firefox
  - *SafariDriver* -> macOS Safari

Handling Multiple Browsers with Selenium

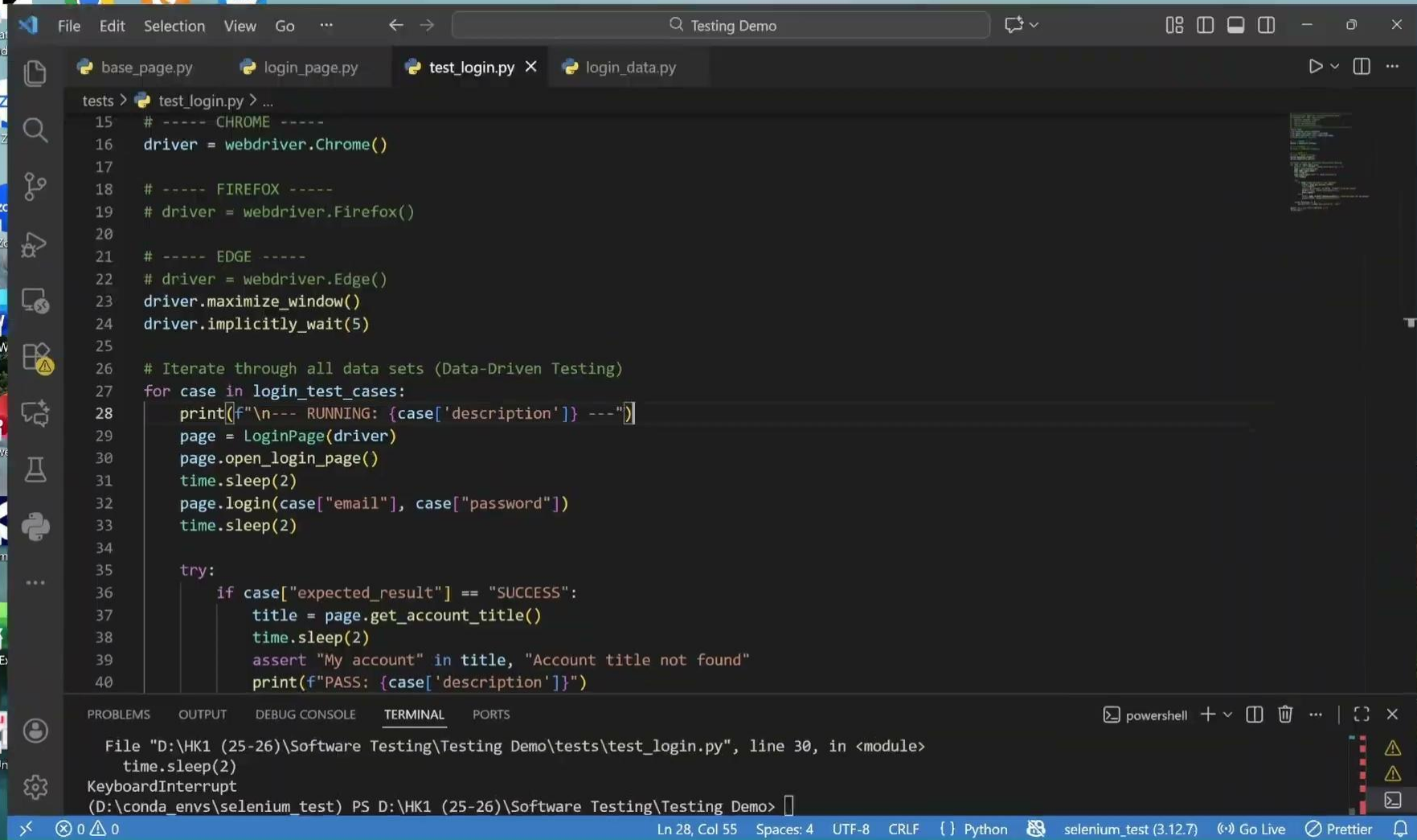
Cross-Browser

```python
from selenium import webdriver

# 1. Configuration
browser_name = "safari"  # Options: "chrome", "firefox", "edge", "safari"

# 2. Driver Factory Logic
if browser_name == "chrome":
    driver = webdriver.Chrome()
elif browser_name == "firefox":
    driver = webdriver.Firefox()
elif browser_name == "edge":
    driver = webdriver.Edge()
elif browser_name == "safari":
    driver = webdriver.Safari() # Built-in on macOS

# 3. Test Script
driver.get("https://www.google.com")
print(driver.title)
driver.quit()
```


File Edit Selection View Go ···
Testing Demo
base_page.py
login_page.py
test_login.py
login_data.py
tests > test_login.py > ...
15 # ----- CHROME -----
16 driver = webdriver.Chrome()
17
18 # ----- FIREFOX -----
19 # driver = webdriver.Firefox()
20
21 # ----- EDGE -----
22 # driver = webdriver.Edge()
23 driver.maximize_window()
24 driver.implicitly_wait(5)
25
26 # Iterate through all data sets (Data-Driven Testing)
27 for case in login_test_cases:
28 print(f"\n--- RUNNING: {case['description']} ---")
29 page = LoginPage(driver)
30 page.open_login_page()
31 time.sleep(2)
32 page.login(case["email"], case["password"])
33 time.sleep(2)
34
35 try:
36 if case["expected_result"] == "SUCCESS":
37 title = page.get_account_title()
38 time.sleep(2)
39 assert "My account" in title, "Account title not found"
40 print(f"PASS: {case['description']}")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
File "D:\HK1 (25-26)\Software Testing\Testing Demo\tests\test_login.py", line 30, in <module>
time.sleep(2)
KeyboardInterrupt
(D:\conda_envs\selenium_test) PS D:\HK1 (25-26)\Software Testing\Testing Demo>
Ln 28, Col 55 Spaces: 4 UTF-8 CRLF { } Python selenium_test (3.12.7) Go Live Prettier

Common Challenges in Selenium

1. Dynamic IDs/ XPath

- IDs or Classes change on every page load or database update (e.g., submit_123 -> submit_234).

- Static locators become invalid immediately.

=> *NoSuchElementExceptions*

Solution:

- **Locate elements relative to stable parents or siblings** (1)**.**
- *The Concept:*
  - Dynamic IDs (e.g., id="ember123") break easily.
  - Visible Text (Labels, Titles) is stable.
  - **Strategy**: Use the Stable Neighbor (Label) as an anchor to find the Dynamic Target (Input/Button).

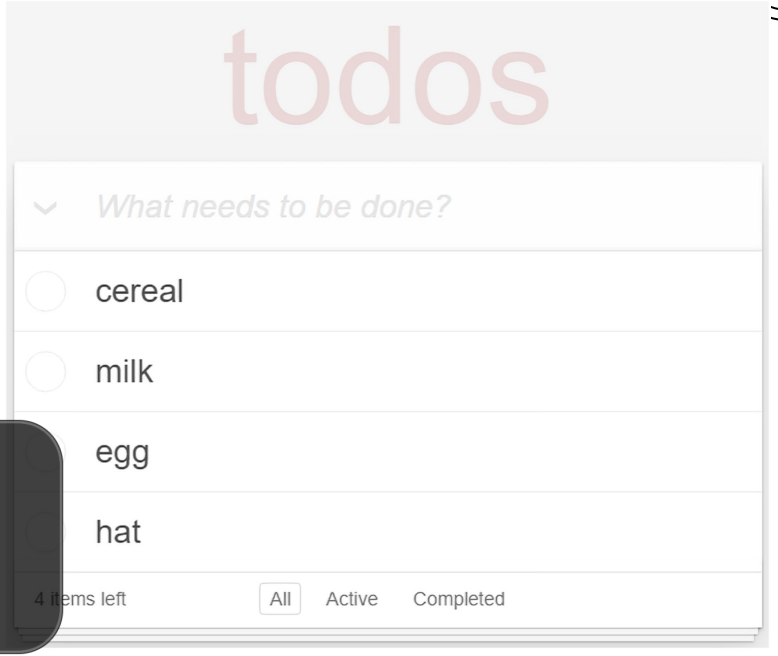
(1) *Nguyen, V., To, T., & Diep, G. H. (2021). Generating and selecting resilient and maintainable locators for Web automated testing. Software Testing, Verification and Reliability, 31(3), e1760.*

Handling Structural Changes (TodoMVC)

Problem: List items change order (e.g., "Cereal" moves down).

Solution:

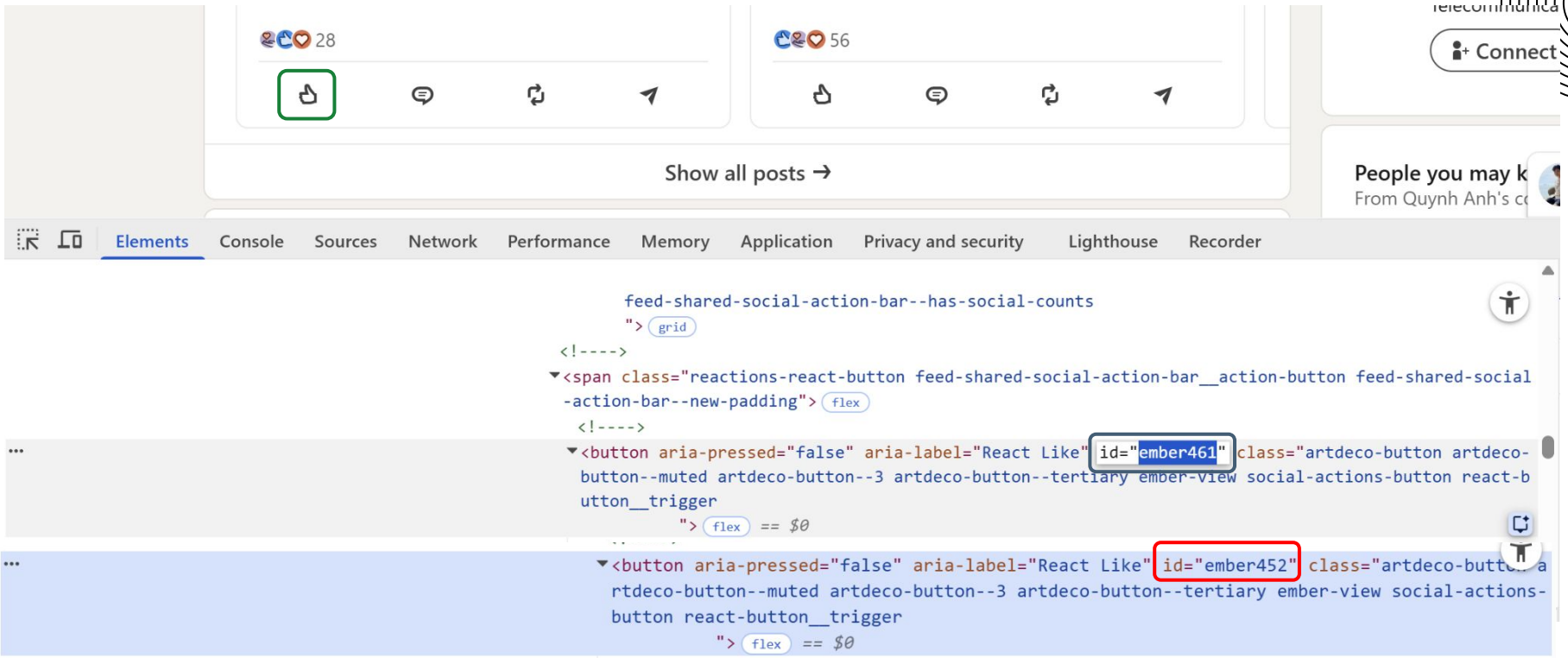
- Instead of index: `//input[1]` (Fragile).



Use Neighbor

```
//label[text()='cereal']/preceding-sibling::input.
```

Handling Dynamic IDs (LinkedIn)



Handling Dynamic IDs (LinkedIn)

Problem: Frameworks like EmberJS generate random IDs on every reload.

Solution:

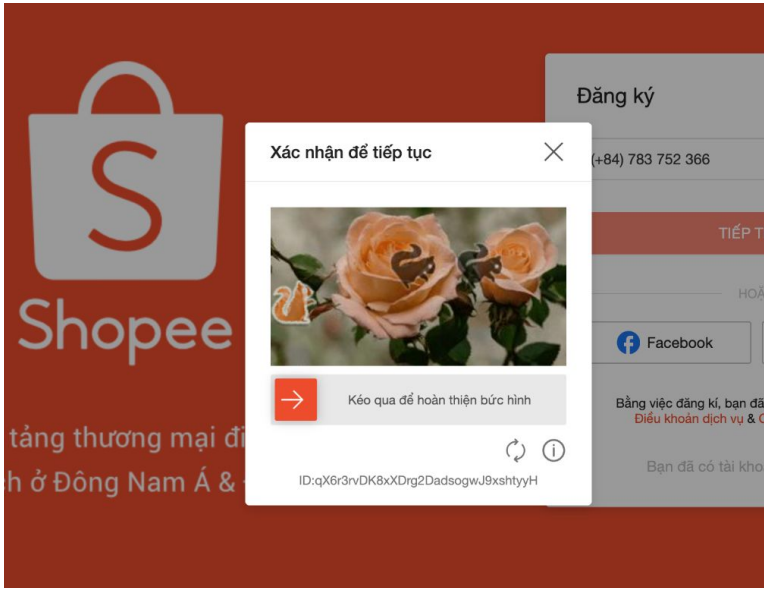
- Instead of ID: `//button[@id='ember4661']` (Fails on reload).

Use Neighbor

```
(.//*[text()='username'])[1]/following::button[@aria-label='React Like'][1]
```

2. Captcha & OTP

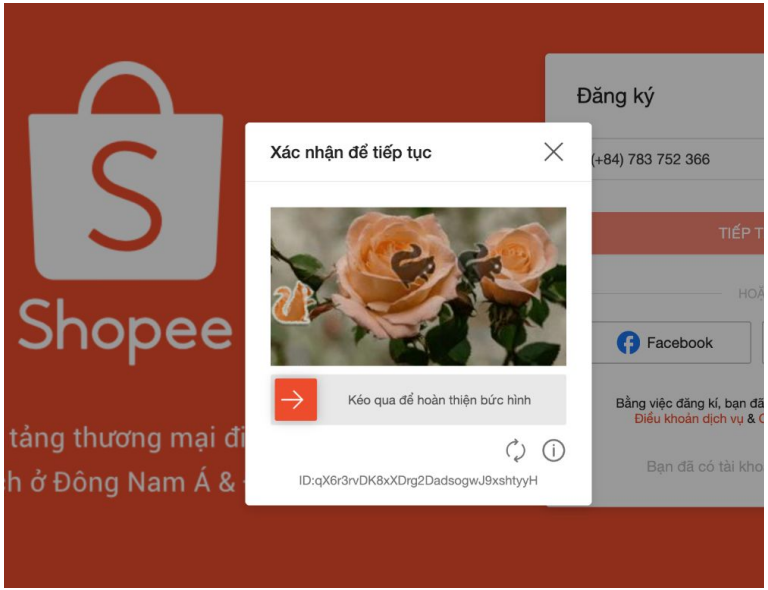
- Designed specifically to block automation tools.
- Captcha requires human cognitive interaction.
- OTPs require external devices



2. Captcha & OTP

Solution:

- Disable Captcha in Dev or Staging environments.
- Static OTP: Configure a hardcoded code (ex: 999999) for test accounts.

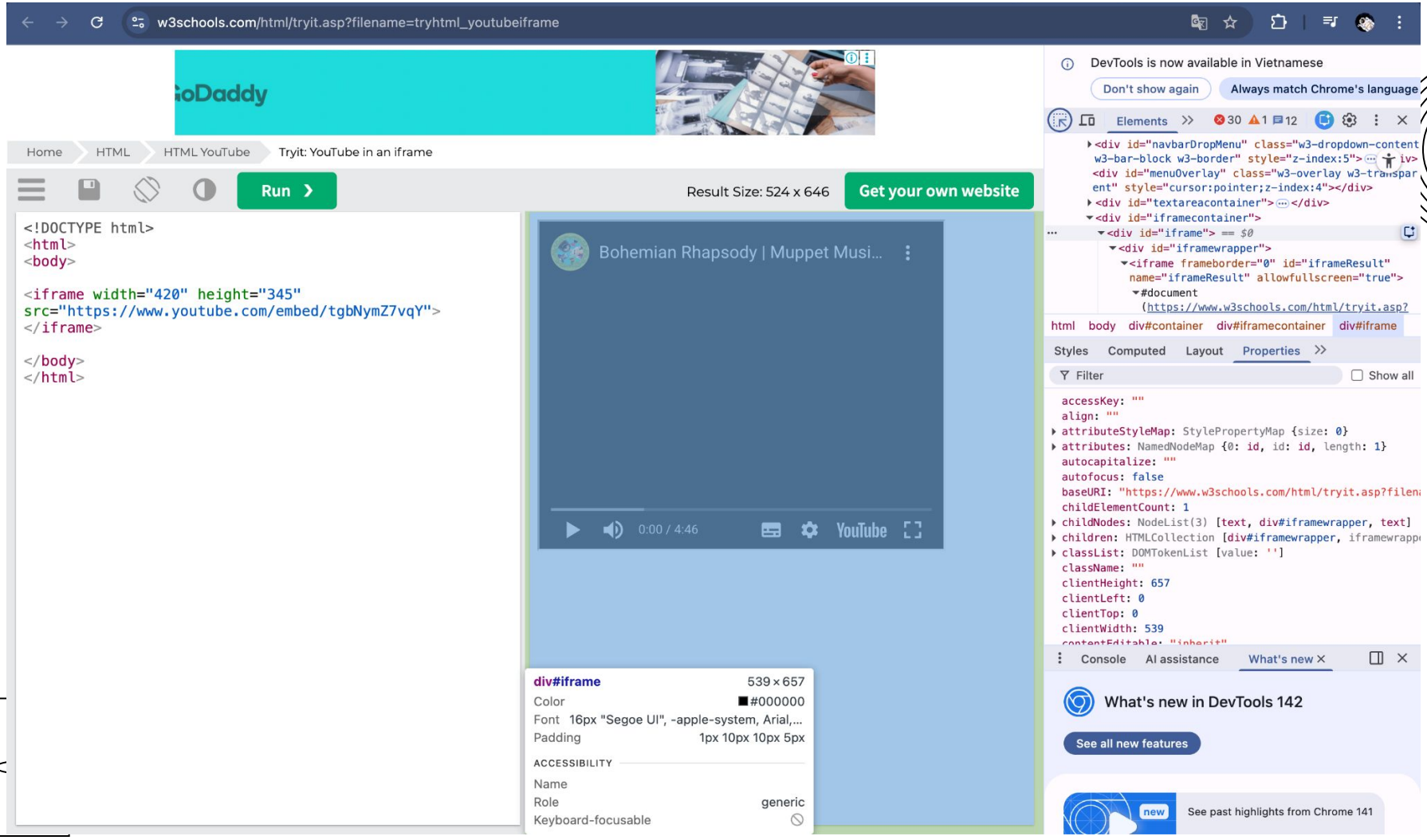


3. iFrames

- A webpage embedded inside another webpage
- Selenium driver focuses only on the main page context.

iFrame

```
<div id="modal">
  <iframe id="buttonframe" name="myframe"  src="https://seleniumhq.github.io">
   <button>Click here</button>
 </iframe>
</div>
```


w3schools.com/html/tryit.asp?filename=tryhtml_youtubeiframe
GoDaddy
Home
HTML
HTML YouTube
Tryit: YouTube in an iframe
Run
Result Size: 524 x 646
Get your own website
<!DOCTYPE html>
<html>
<body>
<iframe width="420" height="345"
src="https://www.youtube.com/embed/tgbNymZ7vqY">
</iframe>
</body>
</html>
Bohemian Rhapsody | Muppet Musi...
0:00 / 4:46
YouTube
DevTools is now available in Vietnamese
Don't show again
Always match Chrome's language
Elements

 == $0

<iframe frameborder="0" id="iframeResult" name="iframeResult" allowfullscreen="true">
#document
(https://www.w3schools.com/html/tryit.asp?
html body div#container div#iframecontainer div#iframe
Styles
Computed
Layout
Properties
Filter
Show all
accessKey: ""
align: ""
attributeStyleMap: StylePropertyMap {size: 0}
attributes: NamedNodeMap {0: id, id: id, length: 1}
autocapitalize: ""
autofocus: false
baseURI: "https://www.w3schools.com/html/tryit.asp?filen
childElementCount: 1
childNodes: NodeList(3) [text, div#iframewrapper, text]
children: HTMLCollection [div#iframewrapper, iframewrapp
classList: DOMTokenList [value: '']
className: ""
clientHeight: 657
clientLeft: 0
clientTop: 0
clientWidth: 539
div#iframe
539 × 657
Color
#000000
Font
16px "Segoe UI", -apple-system, Arial,...
Padding
1px 10px 10px 5px
ACCESSIBILITY
Name
Role
generic
Keyboard-focusable
Console
AI assistance
What's new
What's new in DevTools 142
See all new features
new
See past highlights from Chrome 141

3. iFrames

Solution:

- Explicitly switch the driver's focus.

  ex: driver.switch_to.frame("frame_id")

- Must switch back to the main page after action.

  ex: driver.switch_to.default_content()

04

AI-First

4.1

Integrate AI with Selenium

Generative AI in Test Design

- AI analyzes User Stories/Requirements/Interfaces to generate Test Cases.
- Optimization: AI removes redundant test cases
- Converts Test Steps into boilerplate Selenium scripts (Java/Python).
- Tools: ChatGPT, GitHub Copilot,Gemini.

You are a QA engineer. Analyze the UI in the uploaded image and generate a complete set of functional test cases. Include: - Test Case ID - Test Case Description - Preconditions - Test Steps - Expected Result Do NOT assume features that are not visible in the image. If some behavior is unclear, list your assumptions separately.

Below is a **complete, clean, English functional test-case set** based strictly on what is visible in the UI. No hidden features are assumed. Any unclear behaviors are listed in **Assumptions** at the end.

✅ Functional Test Cases – Customer Registration Form

---

1. Required Field Validation

TC001 – First name is required

- **Description:** Verify the form does not submit when First name is empty.
- **Preconditions:** Registration page is open.
- **Test Steps:**
  1. Leave *First name* empty.
  2. Fill all other fields with valid data.
  3. Click **Register**.
- **Expected Result:**
  - An error message is shown for First name.
  - Form is not submitted.

TC002 – Last name is required

- Same structure as TC001 but for *Last name*.

TC003 – Date of Birth is required

- Empty Date of Birth → Click Register
- **Expected:** Error displayed; form not submitted

Review all generated test cases, remove duplicates, and optimize the list.
Merge similar cases and ensure full coverage of UI and functional behaviors shown in the image.

Below is the **optimized, consolidated, and duplicate-free test case set**, ensuring **complete coverage** of everything visible in the UI.

I merged similar/overlapping cases, removed redundancy, and grouped them logically.
No assumptions beyond what the UI explicitly shows.

✅ Optimized Functional Test Cases – Customer Registration Form

---

1. Required Fields

TC01 – Required field validation (all mandatory fields)

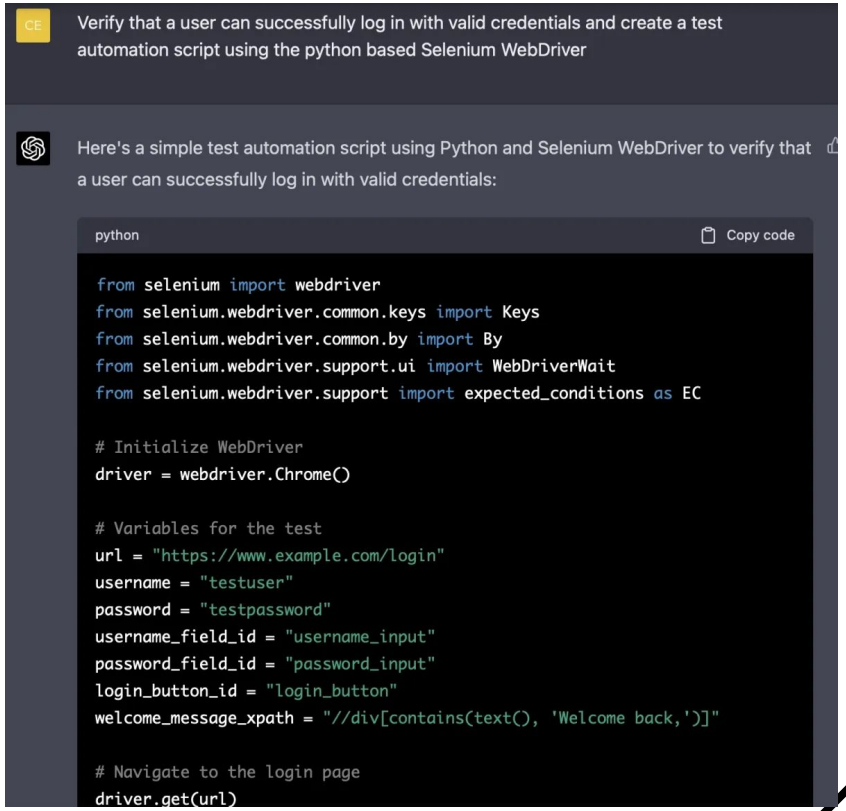
- **Description:** Verify the form does not submit when any required field is empty.
- **Preconditions:** Registration page is open.
- **Test Steps:**
  1. Leave one required field empty (repeat once for each field):
     - First name
     - Last name
     - Date of Birth
     - Street
     - Postal code
     - City
     - State
     - Country
     - Phone
     - Email address
     - Password
  2. Fill all other fields with valid values.
  3. Click **Register**.

Example

- Writing test scripts is **time-consuming**

➔ Generative AI writes Selenium tests from natural language prompts.

➔ AI tools automatically explore web flows and generate tests.

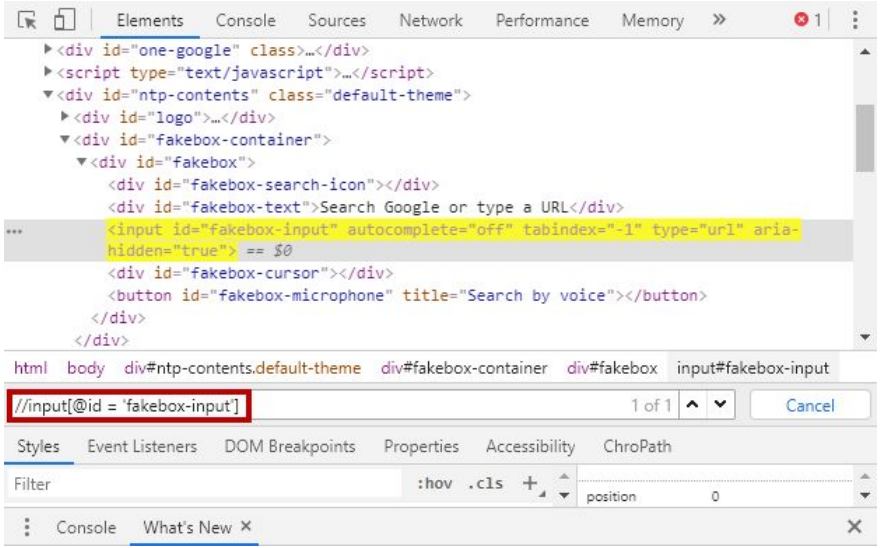


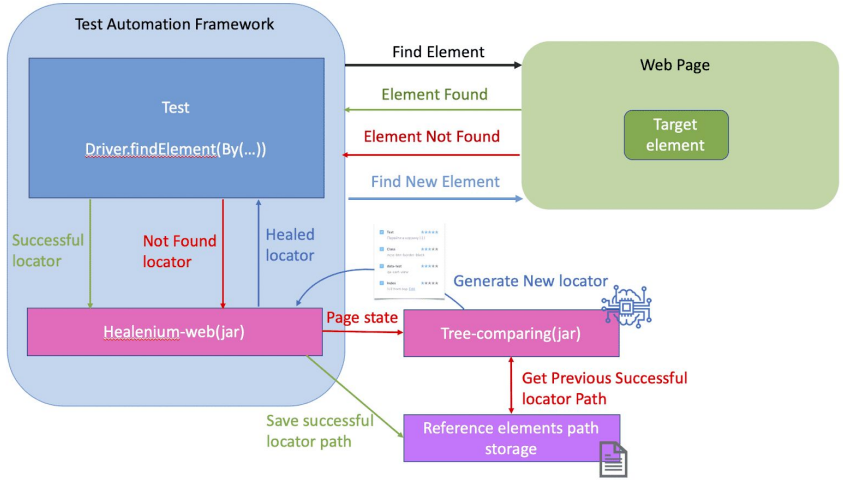
Self-Healing Execution

- Selenium tests rely on **fixed locators** (IDs, XPaths). When **developers update** the UI, **tests break**.

➔ AI **“learns” elements** using multiple attributes (text, position, color, etc.). When the **UI changes**, AI **automatically finds and fixes** the locator so that tests continue running smoothly.

➔ Tool: Healenium



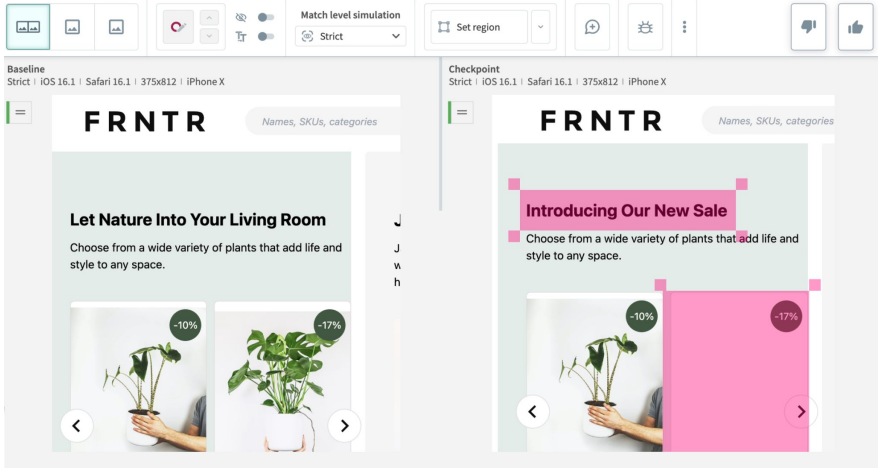


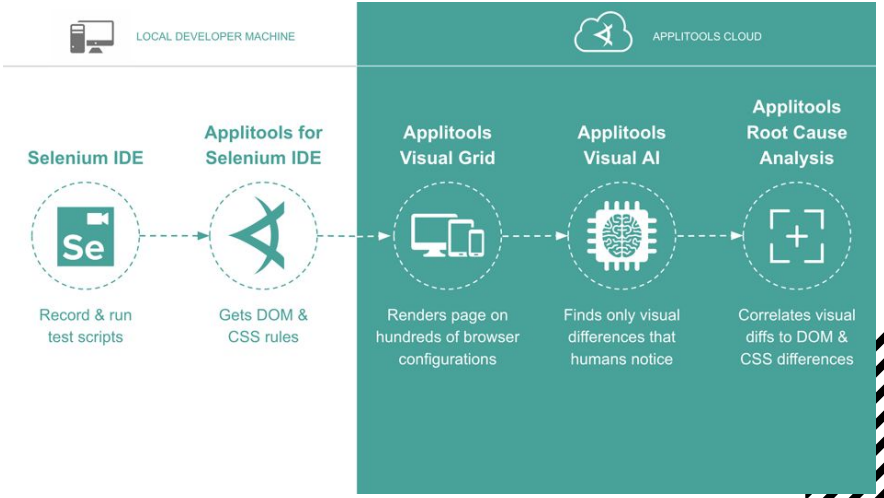
Visual AI Validation

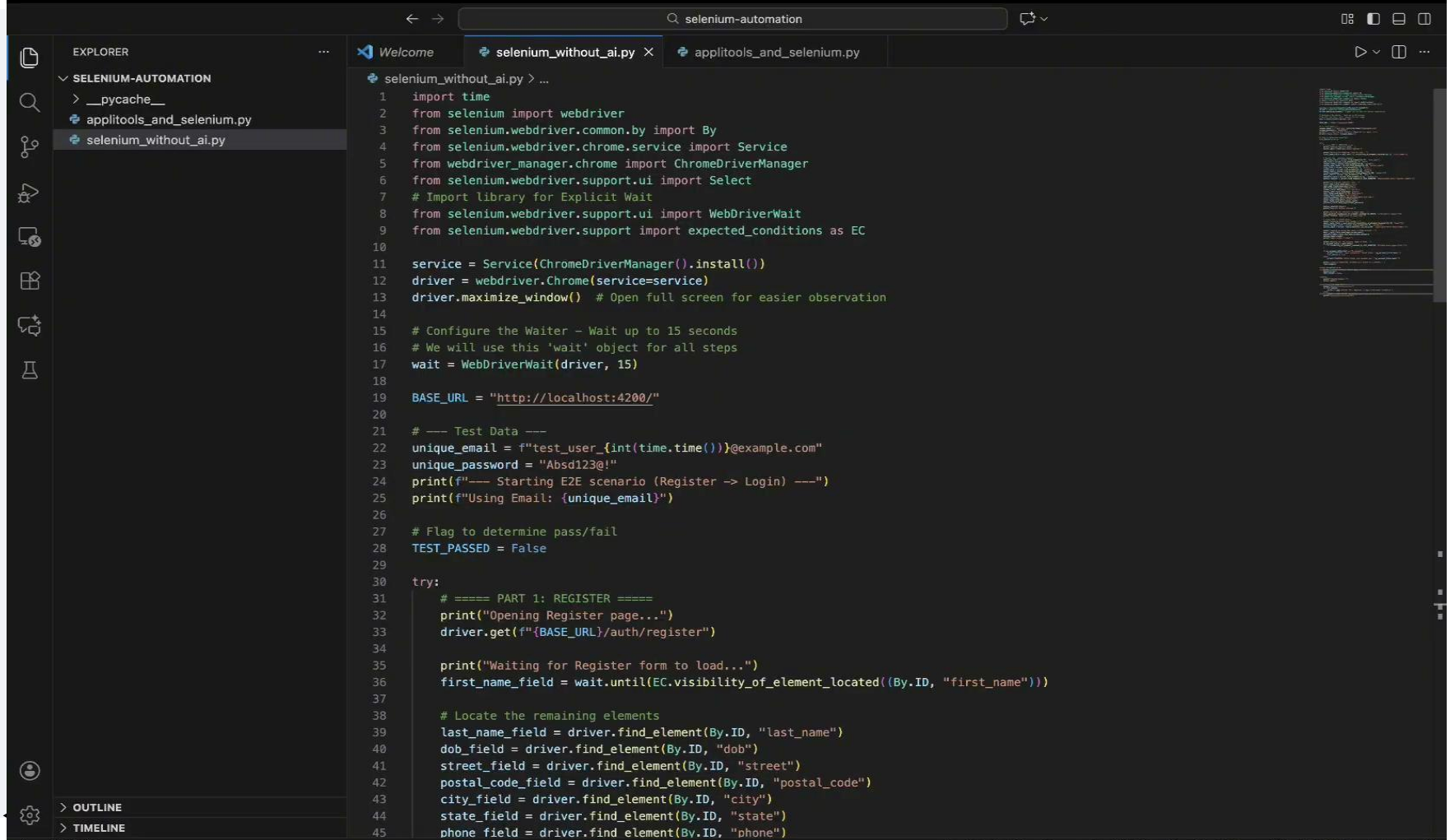
- Selenium only reads code (DOM) - it **cannot see the interface**. It misses layout breaks, overlapping text, or wrong colors.

➔ AI compares the “baseline” image with the current UI screenshot to find visual inconsistencies.

➔ Tool: **Applitools**






selenium-automation
EXPLORER
SELENIUM-AUTOMATION
__pycache__
applitools_and_selenium.py
selenium_without_ai.py
Welcome
selenium_without_ai.py
applitools_and_selenium.py
selenium_without_ai.py > ...
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import Select
# Import library for Explicit Wait
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)
driver.maximize_window() # Open full screen for easier observation
# Configure the Waiter - Wait up to 15 seconds
# We will use this 'wait' object for all steps
wait = WebDriverWait(driver, 15)
BASE_URL = "http://localhost:4200/"
# --- Test Data ---
unique_email = f"test_user_{int(time.time())}@example.com"
unique_password = "Absd123@!"
print(f"--- Starting E2E scenario (Register -> Login) ---")
print(f"Using Email: {unique_email}")
# Flag to determine pass/fail
TEST_PASSED = False
try:
    # ===== PART 1: REGISTER =====
    print("Opening Register page...")
    driver.get(f"{BASE_URL}/auth/register")
    print("Waiting for Register form to load...")
    first_name_field = wait.until(EC.visibility_of_element_located((By.ID, "first_name")))
    # Locate the remaining elements
    last_name_field = driver.find_element(By.ID, "last_name")
    dob_field = driver.find_element(By.ID, "dob")
    street_field = driver.find_element(By.ID, "street")
    postal_code_field = driver.find_element(By.ID, "postal_code")
    city_field = driver.find_element(By.ID, "city")
    state_field = driver.find_element(By.ID, "state")
    phone_field = driver.find_element(By.ID, "phone")
OUTLINE
TIMELINE
Ln 109, Col 19
Spaces: 4
UTF-8
LF
Python
3.10.19 (selenium)


Make Selenium **smarter, more stable, and more efficient**


4.2

AI-Native Low-Code Platforms

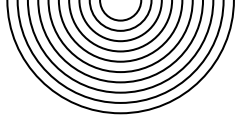
AI-Powered Automation Platforms

- Moving from raw coding to intelligent, low-code platforms.
- Popular Tools: Katalon Platform, TestComplete, Testim, Mabl, Tricentis Tosca.
- AI features are built-in, no need to install separate libraries.
- AI automatically handles page loading (solves Flakiness).
- Learns from past failures to optimize test runs.

Katalon

- A comprehensive Quality Management Platform for Web, Mobile, API, and Desktop testing.
- Integrated GPT to generate code from text
- Native mechanisms to handle dynamic IDs and loading issues automatically.
- AI generates regression tests based on real user interactions in Production environments.

Q&A



Thanks