**Hướng dẫn chi tiết Frontend Test Automation với Selenium**

**Bước 1: Khởi động Frontend Application**

**1.1 Chuẩn bị Frontend Project**

**1.2 Cấu hình môi trường test**

// package.json - Thêm script cho test environment

{

"scripts": {

"start": "react-scripts start",

"start:test": "REACT\_APP\_ENV=test react-scripts start",

"build:test": "REACT\_APP\_ENV=test npm run build",

"test:e2e": "npm run start:test & sleep 10 && npm run selenium:test"

}

}

// .env.test - Environment variables cho testing

REACT\_APP\_API\_URL=http://localhost:8080/api

REACT\_APP\_ENV=test

REACT\_APP\_TEST\_TIMEOUT=30000

**1.3 Kiểm tra ứng dụng hoạt động**

* **Truy cập**: http://localhost:3000 (React) hoặc http://localhost:4200 (Angular)
* **Kiểm tra**: Trang load thành công, không có lỗi console
* **Test data**: Đảm bảo có dữ liệu test sẵn sàng

**Bước 2: Setup Selenium Test Environment**

**2.1 Cài đặt Selenium và Dependencies**

**Java Project Setup**

<!-- pom.xml -->

<dependencies>

<!-- Selenium WebDriver -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.15.0</version>

</dependency>

<!-- TestNG hoặc JUnit -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.8.0</version>

</dependency>

<!-- WebDriverManager -->

<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>

<version>5.6.2</version>

</dependency>

<!-- ExtentReports -->

<dependency>

<groupId>com.aventstack</groupId>

<artifactId>extentreports</artifactId>

<version>5.1.1</version>

</dependency>

</dependencies>

**Python Project Setup**

# Tạo virtual environment

python -m venv selenium\_env

source selenium\_env/bin/activate # Linux/Mac

# selenium\_env\Scripts\activate # Windows

# Cài đặt packages

pip install selenium

pip install pytest

pip install pytest-html

pip install webdriver-manager

pip install allure-pytest

# requirements.txt

selenium==4.15.0

pytest==7.4.3

pytest-html==4.1.1

webdriver-manager==4.0.1

allure-pytest==2.13.2

python-dotenv==1.0.0

**2.2 Cấu hình WebDriver**

**Java Configuration**

// src/main/java/config/WebDriverConfig.java

package config;

import io.github.bonigarcia.wdm.WebDriverManager;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.chrome.ChromeOptions;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.firefox.FirefoxOptions;

public class WebDriverConfig {

private static WebDriver driver;

public static WebDriver getDriver(String browserName) {

if (driver == null) {

switch (browserName.toLowerCase()) {

case "chrome":

WebDriverManager.chromedriver().setup();

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.addArguments("--headless"); // Chạy headless nếu cần

chromeOptions.addArguments("--no-sandbox");

chromeOptions.addArguments("--disable-dev-shm-usage");

chromeOptions.addArguments("--window-size=1920,1080");

driver = new ChromeDriver(chromeOptions);

break;

case "firefox":

WebDriverManager.firefoxdriver().setup();

FirefoxOptions firefoxOptions = new FirefoxOptions();

firefoxOptions.addArguments("--headless");

driver = new FirefoxDriver(firefoxOptions);

break;

default:

throw new IllegalArgumentException("Browser not supported: " + browserName);

}

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

}

return driver;

}

public static void closeDriver() {

if (driver != null) {

driver.quit();

driver = null;

}

}

}

**Python Configuration**

# config/webdriver\_config.py

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options as ChromeOptions

from selenium.webdriver.firefox.options import Options as FirefoxOptions

from webdriver\_manager.chrome import ChromeDriverManager

from webdriver\_manager.firefox import GeckoDriverManager

import os

class WebDriverConfig:

driver = None

@classmethod

def get\_driver(cls, browser\_name="chrome", headless=False):

if cls.driver is None:

if browser\_name.lower() == "chrome":

chrome\_options = ChromeOptions()

if headless:

chrome\_options.add\_argument("--headless")

chrome\_options.add\_argument("--no-sandbox")

chrome\_options.add\_argument("--disable-dev-shm-usage")

chrome\_options.add\_argument("--window-size=1920,1080")

service = Service(ChromeDriverManager().install())

cls.driver = webdriver.Chrome(service=service, options=chrome\_options)

elif browser\_name.lower() == "firefox":

firefox\_options = FirefoxOptions()

if headless:

firefox\_options.add\_argument("--headless")

service = Service(GeckoDriverManager().install())

cls.driver = webdriver.Firefox(service=service, options=firefox\_options)

cls.driver.maximize\_window()

cls.driver.implicitly\_wait(10)

return cls.driver

@classmethod

def close\_driver(cls):

if cls.driver:

cls.driver.quit()

cls.driver = None

**2.3 Page Object Model Setup**

**Java Page Object Example**

// src/main/java/pages/LoginPage.java

package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.openqa.selenium.support.ui.ExpectedConditions;

import java.time.Duration;

public class LoginPage {

private WebDriver driver;

private WebDriverWait wait;

// Page Elements

@FindBy(id = "username")

private WebElement usernameField;

@FindBy(id = "password")

private WebElement passwordField;

@FindBy(xpath = "//button[@type='submit']")

private WebElement loginButton;

@FindBy(css = ".error-message")

private WebElement errorMessage;

@FindBy(css = ".success-message")

private WebElement successMessage;

// Constructor

public LoginPage(WebDriver driver) {

this.driver = driver;

this.wait = new WebDriverWait(driver, Duration.ofSeconds(10));

PageFactory.initElements(driver, this);

}

// Page Actions

public void enterUsername(String username) {

wait.until(ExpectedConditions.visibilityOf(usernameField));

usernameField.clear();

usernameField.sendKeys(username);

}

public void enterPassword(String password) {

passwordField.clear();

passwordField.sendKeys(password);

}

public void clickLoginButton() {

loginButton.click();

}

public void login(String username, String password) {

enterUsername(username);

enterPassword(password);

clickLoginButton();

}

// Verification Methods

public boolean isErrorMessageDisplayed() {

try {

return errorMessage.isDisplayed();

} catch (Exception e) {

return false;

}

}

public String getErrorMessage() {

wait.until(ExpectedConditions.visibilityOf(errorMessage));

return errorMessage.getText();

}

public boolean isLoginSuccessful() {

try {

wait.until(ExpectedConditions.visibilityOf(successMessage));

return true;

} catch (Exception e) {

return false;

}

}

}

**Python Page Object Example**

# pages/login\_page.py

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.common.exceptions import TimeoutException

class LoginPage:

def \_\_init\_\_(self, driver):

self.driver = driver

self.wait = WebDriverWait(driver, 10)

# Locators

self.username\_field = (By.ID, "username")

self.password\_field = (By.ID, "password")

self.login\_button = (By.XPATH, "//button[@type='submit']")

self.error\_message = (By.CSS\_SELECTOR, ".error-message")

self.success\_message = (By.CSS\_SELECTOR, ".success-message")

def enter\_username(self, username):

element = self.wait.until(EC.visibility\_of\_element\_located(self.username\_field))

element.clear()

element.send\_keys(username)

def enter\_password(self, password):

element = self.driver.find\_element(\*self.password\_field)

element.clear()

element.send\_keys(password)

def click\_login\_button(self):

self.driver.find\_element(\*self.login\_button).click()

def login(self, username, password):

self.enter\_username(username)

self.enter\_password(password)

self.click\_login\_button()

def is\_error\_message\_displayed(self):

try:

self.wait.until(EC.visibility\_of\_element\_located(self.error\_message))

return True

except TimeoutException:

return False

def get\_error\_message(self):

element = self.wait.until(EC.visibility\_of\_element\_located(self.error\_message))

return element.text

def is\_login\_successful(self):

try:

self.wait.until(EC.visibility\_of\_element\_located(self.success\_message))

return True

except TimeoutException:

return False

**Bước 3: Test thủ công trước khi Automation**

**3.1 Manual Test Cases**

**Test Case 1: Valid Login**

Test Case ID: TC\_LOGIN\_001

Test Case Name: Valid Login Test

Preconditions: User has valid credentials

Steps:

1. Navigate to login page

2. Enter valid username

3. Enter valid password

4. Click login button

Expected Result: User successfully logged in and redirected to dashboard

**Test Case 2: Invalid Login**

Test Case ID: TC\_LOGIN\_002

Test Case Name: Invalid Login Test

Preconditions: Application is accessible

Steps:

1. Navigate to login page

2. Enter invalid username

3. Enter invalid password

4. Click login button

Expected Result: Error message displayed, user remains on login page

**3.2 Thực hiện Manual Testing**

1. **Mở browser** và navigate đến ứng dụng
2. **Test từng chức năng** theo test cases đã định nghĩa
3. **Ghi nhận kết quả**: Pass/Fail cho mỗi test case
4. **Kiểm tra UI elements**: Buttons, links, forms hoạt động đúng
5. **Verify data**: Dữ liệu hiển thị và xử lý chính xác

**3.3 Identify Elements cho Automation**

<!-- Thêm test-id attributes vào HTML elements -->

<input id="username" data-testid="username-input" />

<input id="password" data-testid="password-input" />

<button type="submit" data-testid="login-button">Login</button>

<div class="error-message" data-testid="error-message"></div>

**Bước 4: Test Automation Implementation**

**4.1 Basic Test Cases**

**Java TestNG Example**

// src/test/java/tests/LoginTest.java

package tests;

import config.WebDriverConfig;

import pages.LoginPage;

import org.testng.annotations.\*;

import org.testng.Assert;

import org.openqa.selenium.WebDriver;

public class LoginTest {

private WebDriver driver;

private LoginPage loginPage;

private String baseUrl = "http://localhost:3000";

@BeforeClass

public void setUp() {

driver = WebDriverConfig.getDriver("chrome");

driver.get(baseUrl);

loginPage = new LoginPage(driver);

}

@Test(priority = 1)

public void testValidLogin() {

loginPage.login("admin", "password123");

Assert.assertTrue(loginPage.isLoginSuccessful(),

"Login should be successful with valid credentials");

}

@Test(priority = 2)

public void testInvalidLogin() {

driver.get(baseUrl); // Reset to login page

loginPage.login("invalid", "invalid");

Assert.assertTrue(loginPage.isErrorMessageDisplayed(),

"Error message should be displayed for invalid credentials");

String actualError = loginPage.getErrorMessage();

Assert.assertTrue(actualError.contains("Invalid credentials"),

"Error message should contain 'Invalid credentials'");

}

@Test(priority = 3)

public void testEmptyFields() {

driver.get(baseUrl);

loginPage.login("", "");

Assert.assertTrue(loginPage.isErrorMessageDisplayed(),

"Error message should be displayed for empty fields");

}

@AfterClass

public void tearDown() {

WebDriverConfig.closeDriver();

}

}

**Python Pytest Example**

# tests/test\_login.py

import pytest

from config.webdriver\_config import WebDriverConfig

from pages.login\_page import LoginPage

class TestLogin:

@pytest.fixture(autouse=True)

def setup(self):

self.driver = WebDriverConfig.get\_driver("chrome")

self.base\_url = "http://localhost:3000"

self.driver.get(self.base\_url)

self.login\_page = LoginPage(self.driver)

yield

WebDriverConfig.close\_driver()

def test\_valid\_login(self):

"""Test login with valid credentials"""

self.login\_page.login("admin", "password123")

assert self.login\_page.is\_login\_successful(), \

"Login should be successful with valid credentials"

def test\_invalid\_login(self):

"""Test login with invalid credentials"""

self.driver.get(self.base\_url) # Reset to login page

self.login\_page.login("invalid", "invalid")

assert self.login\_page.is\_error\_message\_displayed(), \

"Error message should be displayed for invalid credentials"

error\_message = self.login\_page.get\_error\_message()

assert "Invalid credentials" in error\_message, \

"Error message should contain 'Invalid credentials'"

def test\_empty\_fields(self):

"""Test login with empty fields"""

self.driver.get(self.base\_url)

self.login\_page.login("", "")

assert self.login\_page.is\_error\_message\_displayed(), \

"Error message should be displayed for empty fields"

**4.2 Advanced Test Scenarios**

**Data-Driven Testing với TestNG**

// src/test/java/tests/DataDrivenLoginTest.java

package tests;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class DataDrivenLoginTest extends BaseTest {

@DataProvider(name = "loginData")

public Object[][] getLoginData() {

return new Object[][] {

{"admin", "password123", true},

{"user1", "pass123", true},

{"invalid", "invalid", false},

{"", "", false},

{"admin", "", false}

};

}

@Test(dataProvider = "loginData")

public void testLoginWithMultipleData(String username, String password, boolean expectedResult) {

driver.get(baseUrl);

loginPage.login(username, password);

if (expectedResult) {

Assert.assertTrue(loginPage.isLoginSuccessful(),

"Login should be successful for: " + username);

} else {

Assert.assertTrue(loginPage.isErrorMessageDisplayed(),

"Error should be displayed for: " + username);

}

}

}

**Parameterized Testing với Pytest**

# tests/test\_login\_parameterized.py

import pytest

class TestLoginParameterized:

@pytest.mark.parametrize("username,password,expected\_success", [

("admin", "password123", True),

("user1", "pass123", True),

("invalid", "invalid", False),

("", "", False),

("admin", "", False)

])

def test\_login\_with\_multiple\_data(self, setup, username, password, expected\_success):

self.login\_page.login(username, password)

if expected\_success:

assert self.login\_page.is\_login\_successful(), \

f"Login should be successful for: {username}"

else:

assert self.login\_page.is\_error\_message\_displayed(), \

f"Error should be displayed for: {username}"

**4.3 Complex UI Interactions**

**Handling Dropdowns, Alerts, và Windows**

// src/main/java/pages/ComplexPage.java

package pages;

import org.openqa.selenium.support.ui.Select;

import org.openqa.selenium.Alert;

public class ComplexPage {

public void selectFromDropdown(String value) {

WebElement dropdown = driver.findElement(By.id("country"));

Select select = new Select(dropdown);

select.selectByVisibleText(value);

}

public void handleAlert(boolean accept) {

Alert alert = driver.switchTo().alert();

if (accept) {

alert.accept();

} else {

alert.dismiss();

}

}

public void switchToNewWindow() {

String mainWindow = driver.getWindowHandle();

Set<String> allWindows = driver.getWindowHandles();

for (String window : allWindows) {

if (!window.equals(mainWindow)) {

driver.switchTo().window(window);

break;

}

}

}

public void uploadFile(String filePath) {

WebElement fileInput = driver.findElement(By.id("file-upload"));

fileInput.sendKeys(filePath);

}

}

**4.4 Cross-Browser Testing**

**TestNG Cross-Browser Configuration**

<!-- testng.xml -->

<?xml version="1.0" encoding="UTF-8"?>

<suite name="CrossBrowserSuite" parallel="tests" thread-count="3">

<test name="ChromeTest">

<parameter name="browser" value="chrome"/>

<classes>

<class name="tests.LoginTest"/>

</classes>

</test>

<test name="FirefoxTest">

<parameter name="browser" value="firefox"/>

<classes>

<class name="tests.LoginTest"/>

</classes>

</test>

<test name="EdgeTest">

<parameter name="browser" value="edge"/>

<classes>

<class name="tests.LoginTest"/>

</classes>

</test>

</suite>

**Pytest Cross-Browser với fixtures**

# conftest.py

import pytest

from config.webdriver\_config import WebDriverConfig

@pytest.fixture(params=["chrome", "firefox"], scope="class")

def browser\_setup(request):

driver = WebDriverConfig.get\_driver(request.param)

request.cls.driver = driver

yield

WebDriverConfig.close\_driver()

**Bước 5: Test Execution và Automation**

**5.1 Running Tests**

**Maven Commands (Java)**

# Chạy all tests

mvn test

# Chạy specific test class

mvn test -Dtest=LoginTest

# Chạy với specific browser

mvn test -Dbrowser=chrome

# Generate reports

mvn test -Dsurefire.reporting.enabled=true

**Pytest Commands (Python)**

# Chạy all tests

pytest

# Chạy specific test file

pytest tests/test\_login.py

# Chạy với HTML report

pytest --html=reports/report.html --self-contained-html

# Chạy parallel tests

pytest -n auto

# Chạy với specific marker

pytest -m smoke

**5.2 Continuous Integration**

**GitHub Actions Workflow**

# .github/workflows/selenium-tests.yml

name: Selenium Test Automation

on:

push:

branches: [ main, develop ]

pull\_request:

branches: [ main ]

schedule:

- cron: '0 2 \* \* \*' # Daily at 2 AM

jobs:

frontend-tests:

runs-on: ubuntu-latest

strategy:

matrix:

browser: [chrome, firefox]

node-version: [18.x]

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: ${{ matrix.node-version }}

cache: 'npm'

- name: Install dependencies

run: npm ci

- name: Start frontend application

run: |

npm run build

npm run start:test &

sleep 30

- name: Setup Python

uses: actions/setup-python@v4

with:

python-version: '3.9'

- name: Install Python dependencies

run: |

pip install -r requirements.txt

- name: Run Selenium tests

run: |

pytest tests/ \

--browser=${{ matrix.browser }} \

--html=reports/report-${{ matrix.browser }}.html \

--self-contained-html \

--junitxml=reports/junit-${{ matrix.browser }}.xml

- name: Upload test reports

uses: actions/upload-artifact@v3

if: always()

with:

name: test-reports-${{ matrix.browser }}

path: reports/

- name: Publish test results

uses: dorny/test-reporter@v1

if: always()

with:

name: Selenium Test Results (${{ matrix.browser }})

path: reports/junit-${{ matrix.browser }}.xml

reporter: java-junit

**5.3 Docker Integration**

**Dockerfile cho Selenium Grid**

# Dockerfile.selenium

FROM selenoid/vnc:chrome\_78.0

USER root

# Install Node.js

RUN curl -sL https://deb.nodesource.com/setup\_18.x | bash -

RUN apt-get install -y nodejs

# Copy application

COPY . /app

WORKDIR /app

# Install dependencies

RUN npm install

# Expose port

EXPOSE 3000

# Start application

CMD ["npm", "start"]

**Docker Compose**

# docker-compose.yml

version: '3.8'

services:

frontend-app:

build: .

ports:

- "3000:3000"

environment:

- REACT\_APP\_ENV=test

selenium-hub:

image: selenium/hub:4.15.0

ports:

- "4444:4444"

chrome-node:

image: selenium/node-chrome:4.15.0

environment:

- HUB\_HOST=selenium-hub

depends\_on:

- selenium-hub

firefox-node:

image: selenium/node-firefox:4.15.0

environment:

- HUB\_HOST=selenium-hub

depends\_on:

- selenium-hub

test-runner:

build:

context: .

dockerfile: Dockerfile.test

depends\_on:

- frontend-app

- selenium-hub

command: pytest tests/ --html=reports/report.html

volumes:

- ./reports:/app/reports

**5.4 Reporting và Monitoring**

**ExtentReports Configuration (Java)**

// src/main/java/utils/ExtentManager.java

package utils;

import com.aventstack.extentreports.ExtentReports;

import com.aventstack.extentreports.reporter.ExtentSparkReporter;

public class ExtentManager {

private static ExtentReports extent;

public static ExtentReports createInstance(String fileName) {

ExtentSparkReporter htmlReporter = new ExtentSparkReporter(fileName);

htmlReporter.config().setTheme(Theme.STANDARD);

htmlReporter.config().setDocumentTitle("Automation Test Report");

htmlReporter.config().setReportName("Frontend Test Results");

extent = new ExtentReports();

extent.attachReporter(htmlReporter);

extent.setSystemInfo("OS", System.getProperty("os.name"));

extent.setSystemInfo("Browser", "Chrome");

return extent;

}

}

**Allure Reports (Python)**

# pytest.ini

[tool:pytest]

addopts = --alluredir=allure-results

markers =

smoke: Smoke tests

regression: Regression tests

login: Login functionality tests

# Generate Allure report

allure generate allure-results --clean

allure serve allure-results

**Best Practices**

**1. Test Organization**

* **Page Object Model**: Tách biệt page logic và test logic
* **Base Classes**: Tạo base classes cho common functionality
* **Test Data Management**: Sử dụng external files (JSON, CSV, Excel)

**2. Element Identification**

* **Stable Locators**: Ưu tiên ID > Name > CSS > XPath
* **Custom Attributes**: Sử dụng data-testid attributes
* **Dynamic Waits**: Sử dụng WebDriverWait thay vì Thread.sleep

**3. Test Maintenance**

* **Regular Updates**: Update tests khi UI thay đổi
* **Modular Design**: Tạo reusable components
* **Error Handling**: Implement proper exception handling

**4. Performance Optimization**

* **Parallel Execution**: Chạy tests song song
* **Headless Mode**: Sử dụng cho CI/CD
* **Resource Management**: Proper driver cleanup

**5. Debugging và Troubleshooting**

* **Screenshots**: Capture screenshots on failure
* **Logging**: Detailed logging for debugging
* **Video Recording**: Record test execution for analysis