Automatyczne testy akceptacyjne - selenium

Automatyczne testy akceptacyjne

Idea testów akceptacyjnych

- Głównym celem testów akceptacyjnych nie jest znajdywanie błędów, a upewnienie się, że aplikacja spełnia oczekiwania klienta i użytkowników
- Przyjęło się, że na tym etapie aplikacja musi wstępnie działać, a testy są wykonywane po stronie klienta lub też przez użytkowników końcowych
- Testy dzielimy:
  - alfa - są wykonywane przez twórców oprogramowania na środowisku testowym
  - beta – są wykonywane u klienta w jego własnej lokalizacji
- Należy ustalić sposób komunikacji między twórcami, a klientem w celu sygnalizacji napotkanych problemów
- Przykład testu
  - Załóżmy, że portal społecznościowy wypuszcza dużą funkcjonalność. Część testów akceptacyjnych jest wykonywana po stronie klienta, przez wybraną grupę użytkowników. Otrzymują oni również narzędzie, w którym dokonują oceny funkcjonalności, zanim owa funkcjonalność trafi do wszystkich

Automatyczne testy akceptacyjne

Idea testów automatycznych

- Testy akceptacyjne to przede wszystkim testowanie UI
- Dzielimy je na na manualne i automatyczne
- Testy manualne są żmudne i kosztowne czasowo, dlatego powstały programy automatyzujące scenariusze testowe
- Testy automatyczne uzupełniają testy manualne, ale nie są wstanie ich całkowicie zastąpić
- Zalety:
  - możliwość zbudowania scenariusza testowego oraz odtwarzania kroków w sposób powtarzalny
  - większa kontrola nad potencjalnym defektami w systemie
- Wady:
  - utrzymywanie testów jest kosztowne

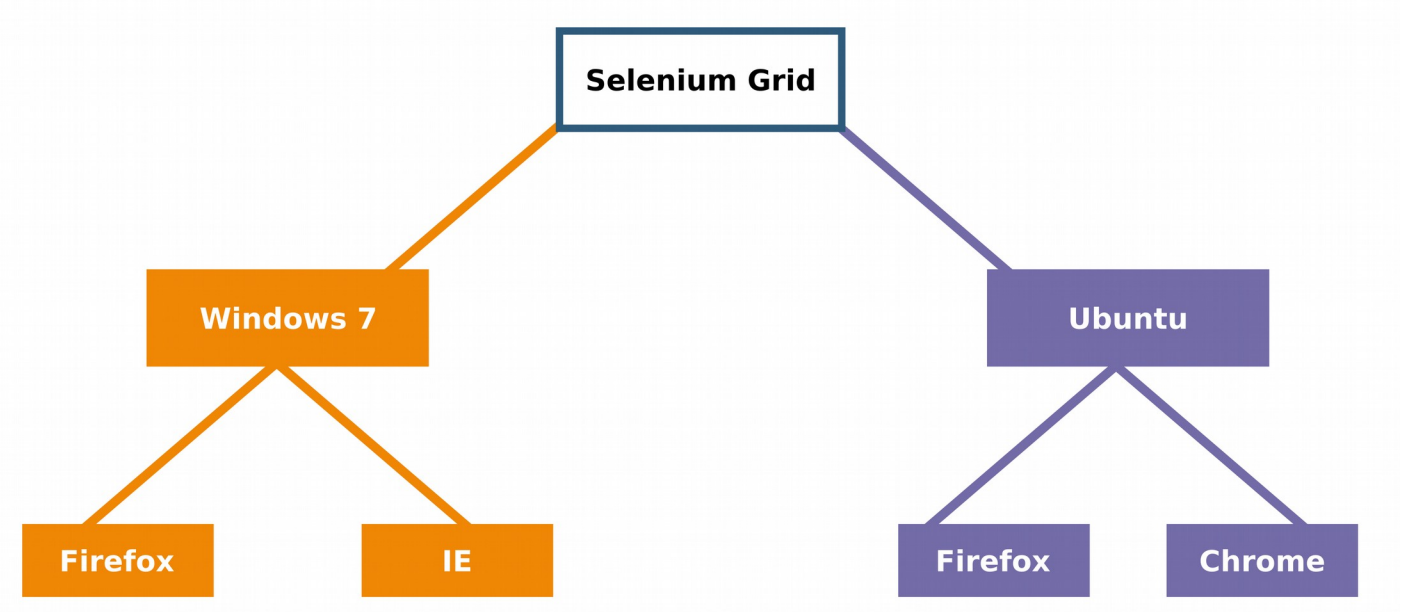
Automatyczne testy akceptacyjne

Selenium

- Selenium to darmowe narzędzie pozwalające przeprowadzić testy automatyczne dla aplikacji webowych
- Wykorzystuje przeglądarkę do sprawdzenia działania kodu i obsługuje wszystkie popularne programy tego rodzaju (m in. Firefox, Safari i Chrome) co umożliwia sprawdzenie oprogramowania dla różnych konfiguracji
- Umożliwia współdziałanie z przeglądarkami bez interfejsu, co pozwala przeprowadzać testy znacznie szybciej i za pomocą mniejszej ilości zasobów

Automatyczne testy akceptacyjne
Selenium Grid

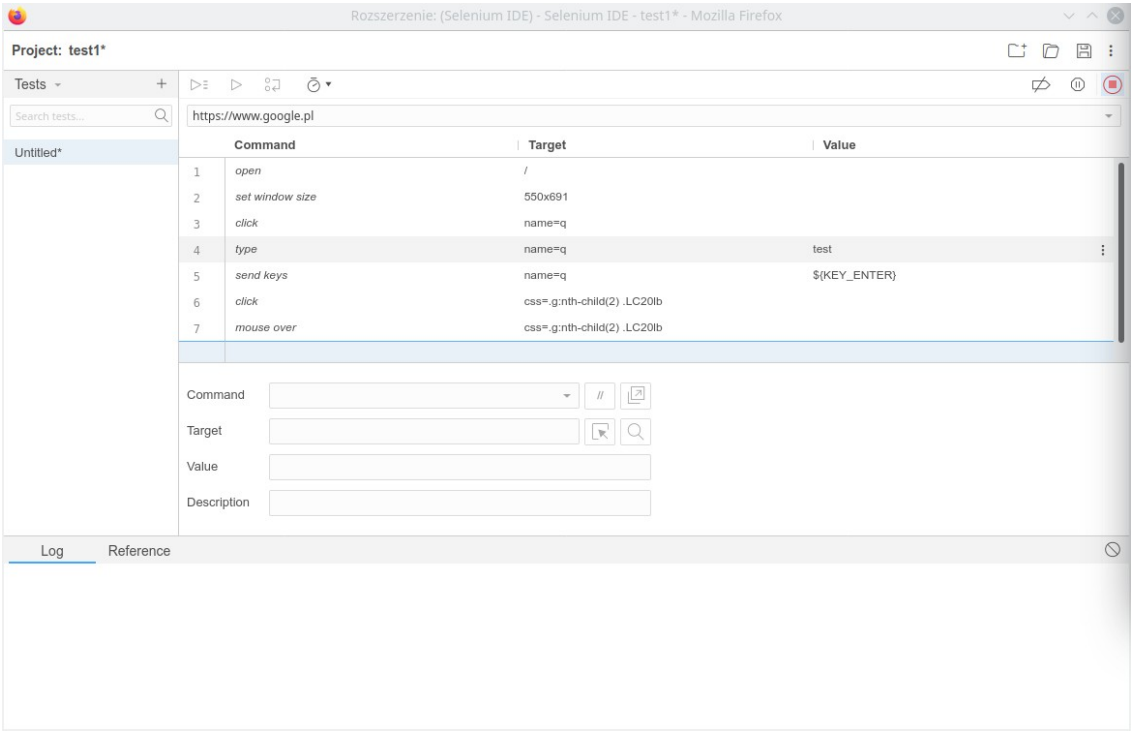
- Selenium Grid – umożliwia wykonywanie testów na wielu serwerach umożliwiając jednoczesne przetestowanie różnych konfiguracji
- Budowa
  - hub - maszyna, która jest centralnym punktem Grida. Otrzymuje żądania i dystrybuuje je pomiędzy node'y
  - node – (minimum jeden) maszyna testowa z własną konfiguracją systemu operacyjnego, przeglądarki i innych parametrów



Automatyczne testy akceptacyjne

Selenium IDE

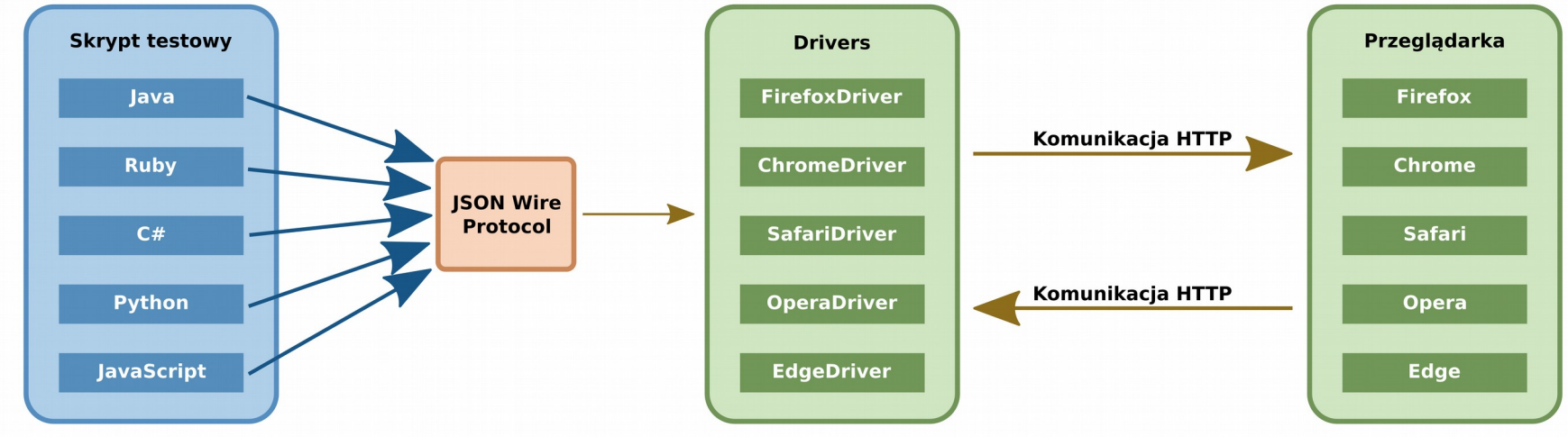
- Selenium IDE - wtyczka do przeglądarki, którą można wykorzystać do tworzenia testów na podstawie zarejestrowanych operacji wykonywanych przez użytkownika (nagrywarka).
- Testy choć generowane w ten sposób pozwalają szybko uzyskać wyniki, nie są polecane, ponieważ taki test nie wspiera reużywalności kodu. W momencie gdy coś się zmieni w naszej aplikacji, łatwiej jest usunąć wszystko i nagrać jeszcze raz zamiast poprawiać gotowe scenariusze



Automatyczne testy akceptacyjne

Selenium WebDriver

- Selenium WebDriver – wykorzystując popularne języki programowania, umożliwia wykonywanie testów automatycznych



- Skrypt testowy posiada oficjalne wsparcie dla takich języków jak Java, C#, Ruby, Python, JavaScript
- Odpala on serwer selenium (driver) zgodny z daną przeglądarką (Google Chrome, Mozilla Firefox, Internet Explorer, Opera, Safari)
- Test wysyła zapytania do serwera selenium poprzez JSON, serwer to przetwarza aby było zrozumiałe dla przeglądarki oraz wysyła do niej oczekując na odpowiedź

Automatyczne testy akceptacyjne

Selenium WebDriver - WebElements

- Testowanie opiera się na WebElementach czyli wszystkich elementach widocznych na stronie. Wyszukujemy je przy pomocy lokatorów:
  - **Atrybutu id**

    <input type=”text” id=”uniqueId”>

    Lokator: By.id("uniqueId")
  - **Atrybutu name**

    <button type=”text” name=”decision”>Submit</button>

    Lokator: By.name("decision")
  - **Class name**

    #superButton – znajdź element o id „superButton”

    .superClass – znajdź wszystkie elementy zawierające klasę „superClass”

    <input type=”text” class=”superClass”>

    Lokator: By.className("superClass")

Automatyczne testy akceptacyjne

Selenium WebDriver - WebElements

- **XPath** - język ścieżek XML. Wskazuje ścieżki do elementów, które dzielimy na relatywne, absolutne lub można wykorzystać atrybuty

  Do wyznaczania ścieżek wykorzystujemy następujące elementy:

  - / – rozpoczyna wyszukiwanie od elementu root
  - // – zaznacza wszystkie elementy w dokumencie
  - nazwa_elementu – wskazuje wszystkie elementy danego typu
  - . – służy do zaznaczenia aktualnego elementu
  - .. – wskazuje rodzica aktualnego elementu
  - @ – służy do określania atrybutów

  Ścieżka absolutna wskazująca na całą stronę

  Lokator: By.xpath("/html")

  Ścieżka relatywna wskazująca wszystkie elementy <button> na stronie

  Lokator: By.xpath("//button")

Automatyczne testy akceptacyjne

Selenium WebDriver - WebElements

- **XPath** c.d.

  Ścieżka relatywna wskazująca wszystkie elementy <button>, które są dziećmi elementu

  Lokator: By.xpath ("//div/button")

  Szukanie elementów <button> zawierających type=’submit’

  Lokator: By.xpath("//button[@type='submit']")

  Metoda niezalecana, ponieważ ścieżki są nietrwałe w momencie zmiany struktury na stronie

- **LinkText** - bazuje na innerText znacznika <a>

  <a>nazwa linku

  Lokator: By.linkText("nazwa linku")

  Metoda niezalecana gdyż tekst może się zmieniać np. w momencie posiadania wielu wersji językowych

- WebElementy możemy odnajdywać na stronie np. wykorzystując w firefoxie inspektora znajdującego się w części „Dla twórców witryn”

Automatyczne testy akceptacyjne

Selenium WebDriver

- Przykładowy scenariusz testowy dla logowania na stronie

  (http://automationpractice.com/index.php)

  - Wejdź na stronę logowania
  - Wpisz dane logowania
  - Wybierz przycisk logowania
  - Sprawdź poprawność logowania

- Problemy skryptu testowego
  - Test jest bardzo słabo czytelny
  - W chwili zmiany frameworka testowego, musimy zmienić również nasze testy. Wszelkie zmiany dotykają naszych testów

```
public class SeleniumTest {

    private WebDriver driver;

    @Before
    public void init() {
        System.setProperty("webdriver.gecko.driver", "driver/geckodriver");
        driver = new FirefoxDriver();
    }

    @Test
    public void selenium() throws Exception {
        String address = "http://automationpractice.com/index.php";
        driver.get(address);

        WebElement loginButton = driver.findElement(By.className("login"));

        loginButton.click();

        WebElement loginField = driver.findElement(By.id("email"));
        loginField.sendKeys("selenium@wp.pl");

        WebElement passwordField = driver.findElement(By.id("passwd"));
        passwordField.sendKeys("abcABC");

        WebElement submitLogin = driver.findElement(By.id("SubmitLogin"));
        submitLogin.click();

        WebElement userLink = driver.findElement(By.linkText("Sele nium"));

        assertTrue(userLink.isDisplayed());
    }

    @After
    public void close() {
        driver.quit();
    }
}
```

Automatyczne testy akceptacyjne

Selenium WebDriver

- Page Object Pattern (POP) - jest to podstawowy wzorzec używany przy implementacji testów automatycznych. Wprowadza ideologię mającą na celu odizolowanie stron, widoków i elementów na stronie od samych testów
- Testy stają się wtedy czytelniejsze, łatwiejsze w implementacji oraz prostsze w utrzymaniu
- Pomaga z problemami związanymi ze zmianami w testowanym oprogramowaniu
- Wzorzec możemy zastosować do wszystkich elementów stron – daje nam to logiczny przepływ w kodzie testu

Automatyczne testy akceptacyjne
Selenium WebDriver

- Modyfikacja testu logowania wykorzystując Page Object Pattern

```java
public class LoginPage {

    private WebDriver driver;

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    @FindBy(id = "email")
    private WebElement loginField;

    @FindBy(id = "passwd")
    private WebElement passwordField;

    @FindBy(id = "SubmitLogin")
    private WebElement submitLogin;

    public void loginOnPage(String login, String password) {
        loginField.sendKeys(login);
        passwordField.sendKeys(password);
        submitLogin.click();
    }
}
```

```java
public class SeleniumTest {

    private WebDriver driver;

    @Before
    public void init() {
        System.setProperty("webdriver.gecko.driver", "driver/geckodriver");
        driver = new FirefoxDriver();
    }

    @Test
    public void selenium() throws Exception {
        String address = "http://automationpractice.com/index.php";

        driver.get(address);

        WebElement loginButton = driver.findElement(By.className("login"));
        loginButton.click();

        LoginPage loginPage = new LoginPage(driver);
        loginPage.loginOnPage("selenium@wp.pl", "abcABC");

        WebElement userLink = driver.findElement(By.linkText("Sele nium"));

        assertTrue(userLink.isDisplayed());

    }

    @After
    public void close() {
        driver.quit();
    }
}
```

Wykorzystujemy PageFactory z inicjalizacją w konstruktorze, tak aby pola z adnotacja findby były wypełniane w momencie utworzenia obiektu

Automatyczne testy akceptacyjne
Selenium WebDriver – podejście standardowe

- Metoda testów omawianych do tej pory
- Kilka standardowych metod:
  - WebDriver.get(String url) – otwórz stronę o podanym adresie
  - WebDriver.close() – zamknij aktywne okno
  - WebDriver.getCurrentUrl() – zwróć url aktualnie otwartej strony
  - WebDriver.Navigation.refresh() – odśwież stronę
  - WebDriver.Navigation.back() – nawiguj „wstecz” w historii przeglądarki
  - WebDriver.Navigation.forward() – nawiguj „dalej” w historii przeglądarki
  - WebElement.click() – kliknij dany element
  - WebElement.sendKeys(<tekst>) – wpisz tekst w element (np. polu tekstowym)
  - WebElement.clear() – jeżeli elementem jest pole tekstowe, wyczyść jego wartość
  - WebElement.getText() – zwróć tekst elementu
- Metody te pozwalają na zasymulowanie tylko podstawowych akcji użytkownika

Automatyczne testy akceptacyjne

Selenium WebDriver – wykorzystanie klasy Actions

- Klasa Actions udostępnia dużą paletę akcji wykonywanych za pomocą klawiatury i myszki. Można je łączyć w ciąg realizowanych po sobie czynności

```
@Test
public void seleniumAction1() throws Exception {
    String address = "http://demo.guru99.com/test/drag_drop.html";

    driver.get(address);

    WebElement From = driver.findElement(By.xpath("//*[@id='credit2']/a"));

    WebElement To = driver.findElement(By.xpath("//*[@id='bank']/li"));

    Actions builder = new Actions(driver);

    builder.dragAndDrop(From, To)
            .perform();

}
```

- Scenariusz testowy:
  - wejdź na stronę
  - przeciągnij upuść element
- Metoda perform() buduje i wykonuje zadaną akcje

Automatyczne testy akceptacyjne

Selenium WebDriver – wykorzystanie klasy Actions

```
@Test
public void seleniumAction2() throws Exception {
    String address = "http://demo.guru99.com/test/drag_drop.html";

    driver.get(address);

    Actions builder = new Actions(driver);

    builder.keyDown(Keys.SHIFT)
            .click(driver.findElement(By.id("item1")))
            .click(driver.findElement(By.id("item10")))
            .keyUp(Keys.SHIFT)
            .dragAndDrop(driver.findElement(By.id("cdk-drop-list-1")), driver.findElement(By.id("cdk-drop-list-2")))
            .perform();

}
```

- Scenariusz testowy:
  - wejdź na stronę
  - wciśnij klawisz SHIFT
  - wybierz elementy od 1 do 10
  - puść klawisz SHIFT
  - przeciągnij upuść element

Automatyczne testy akceptacyjne

Selenium WebDriver – wykorzystanie klasy Actions

- Kilka standardowych metod:
  - Actions.contextClick(WebElement onElement) – kliknij prawym przyciskiem myszy na podanym elemencie
  - Actions.doubleClick() – wykonaj podwójne kliknięcie
  - Actions.moveToElement(WebElement toElement) – ustaw kursor myszy na wskazanym elemencie
  - Actions.keyDown(Keys theKey) – wciśnij przycisk klawiatury
  - Actions.keyUp(Keys theKey) – puść przycisk klawiatury
  - Actions.dragAndDrop(WebElement source, WebElement target) – przeciągnij element „source” i upuść go na elemencie „target”

Automatyczne testy akceptacyjne

Selenium WebDriver – wykorzystanie klasy Robot

- Klasa Robot podobnie jak Action, symuluje klawiaturę i myszkę. Ma mniejsze możliwości niż Actions, ale posiada możliwość kliknięcia czy pisania nie tylko do aplikacji webowej, ale również do okienek systemowych

```
@Test
public void seleniumRobot() throws Exception {

    Robot robot = new Robot();
    robot.mouseMove(500, 500);
    robot.mousePress(InputEvent.BUTTON1_MASK);
    robot.mouseRelease(InputEvent.BUTTON1_MASK);
    robot.keyPress(KeyEvent.VK_7);
    robot.keyRelease(KeyEvent.VK_7);

}
```

- Scenariusz testowy:
  - stwórz instancję klasy Robot
  - przesuń kursor myszy na pozycję x=500px, y=500px
  - kliknij lewym przyciskiem myszy
  - napisz 7

Automatyczne testy akceptacyjne

Selenium WebDriver – wykorzystanie klasy Robot

- Kilka standardowych metod:
  - Robot.mouseMove(int x, int y) – umieść kursor myszki na wskazanych koordynatach ekranu
  - Robot.mousePress(int button) – kliknij przyciskiem myszki
  - Robot.keyPress(int keycode) – wciśnij dany klawisz
  - Robot.keyRelease(int keycode) – puść dany klawisz
  - Robot.mouseWheel(int wheelAmt) – użyj scroll’a myszki
  - Robot.createScreenCapture(Rectangle screenRect) – zrób screenshot zdefiniowanego obszaru ekranu

Automatyczne testy akceptacyjne

Selenium WebDriver – wykorzystanie interfejsu JavascriptExecutor

- Interface JavascriptExecutor umożliwia wykonywanie skryptów javascript z poziomu testu

```
@Test
public void seleniumRobot() throws Exception {

    String address = "http://automationpractice.com/index.php";

    driver.get(address);

    JavascriptExecutor js = (JavascriptExecutor) driver;

    js.executeScript("document.getElementById('search_query_top').value='test';");

}
```

- Zadaniem testu jest wejście na stronę oraz w górnym polu wyszukiwarki wpisać tekst: „test”
- Z wykorzystaniem tego interfejsu można wpisywać wartości, klikać przyciski, zmieniać aktywność przycisków lub przesuwać scrollem