TESTIRANJE SOFTVERA - VEŽBE 08

SELENIUM PAGE OBJECT MODEL & SYNCHRONIZATION

PROBLEMI DOSADAŠNJEG PRISTUPA

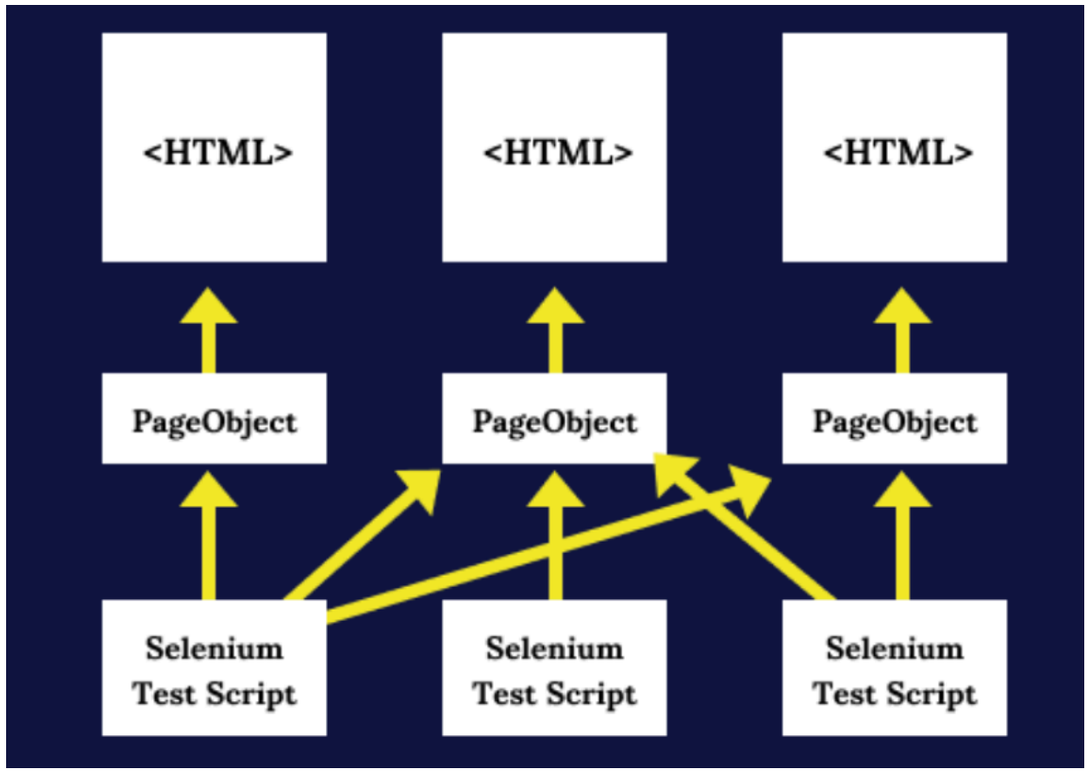
- Selenium omogućava low level API za interakciju sa elementima HTML stranice
- Mane ovog pristupa su:
  - Test slučajevi su teški za čitanje
  - Promene UI-a dovode do promena u testovima
  - Repliciranje lokatora u više različitih testova
- Cilj - modelovati HTML stranicu kao objekat (page objekat)
  - Kretirati klasu koja reprezentuje HTML stranicu koja se testira
  - U atribute izdvojiti sve značajne HTML elemente i omogućiti im pristup pomoću selektora
  - Definisati metode za interakciju sa tim elementima

PAGE OBJECT MODEL

- Page Object Model je dizajn obrazac koji se obično primenjuje prilikom automatskog testiranja veb aplikacija korišćenjem Selenium alata
- Primarni cilj ovog obrasca je da se izbegne dupliranje koda i poboljša njegova ponovna upotreba. Ovo poboljšava opštu lakoću održavanja test slučaja i robusnost testnog koda, čineći ga na taj način programmer-friendly
- Automatski Selenium testovi omogućavaju kretanje do različitih veb stranica i izvršavanje relevantnih radnji nad `WebElement` objektima koji su prethodno identifikovani pomoću odgovarajućih Selenijum lokatora. POM obrazac grupiše sve elemente, radnje i validacije koje se dešavaju na stranici u jednu Page klasu

PAGE OBJECT MODEL

- POM obrazac deklariše kreiranje klase koja predstavlja korisnički interfejs stranice koju želimo da testiramo. Kao što je pomenuto, ova klasa će povezati sve HTML elemente (lokatore) i enkapsulirati interakcije sa korisničkim interfejsom (metode)
- Na primer, svi pozivi WebDriver-a će se izvršavati preko enkapsuliranih metoda. Dakle, ovo je jedino mesto koje treba da se izmeni kada se korisnički interfejs promeni - lokatori elemenata postaju centralizovani
- Česte promene korisničkog interfejsa će uticati samo na datoteke u kojima se nalaze lokatori, uz minimalne (ili nulte) promene u implementaciji testa
- Page objekti su odgovorni za komunikaciju sa veb stranicama koje se testiraju tako što jasno odvajaju stvarni test kod od koda specifičnog za stranicu



POM BEST PRACTICES

- Imajte na umu sledeće najbolje prakse prilikom definisanja Page objekata:
  - Objekat stranice može biti kreiran za celu stranicu/ekran ili samo za neke njegove fragmente
  - Izbegavajte nepotrebnu složenost, što na duge staze dovodi do teško čitljivog koda
  - Objekti stranice bi trebalo da postoje samo ako neko ponašanje ima smisla iz perspektive korisnika
- Kada je u pitanju Page Object šablon neophodno je poštovati sledeća pravila:
  - Page objekti ne sadrže asertacije
  - Page objekti treba da sadrže samo one elemente UI-a koji su potrebni za testove, a ne nužno i sve elemente stranice
  - Testovi ne sadrže lokatore

"IF YOU HAVE WEBDRIVER APIS IN YOUR TEST METHODS, YOU'RE DOING IT WRONG."

Simon Stewart

PAGE FACTORY

- Page Factory je ugrađeni Page Object Model (POM) za Selenium WebDriver i može se smatrati ekstenzijom Page Objekata
- Page Factory stranica je katalizator koji pojednostavljuje korišćenje objekata stranice time što deklariše sledeće:
  - deklaraciju lokatore za sve relativne elemenata stranice na samom početku klase
  - inicijalizaciju svih elemenata u trenutku učitavanja stranice
  - Kreiranjeobjekat same stranice pre korišćenja lociranih elemenata
- Selenium WebDriver sadrži klasu `PageFactory` koja definiše `@FindBy` anotaciju u cilju lociranje veb elemenata. Ovi veb elementi su definisani u klasama veb stranica (ili objektima stranice).
- Klasa `PageFactory` takođe obezbeđuje i metod `initElements` za inicijalizaciju veb elemenata čime garantujemo postojanje elemenata pre izvršavanja bilo kakvih radnji nad njima

INICIJALIZACIJA WEB ELEMENATA

- Postoji nekoliko načina za inicijalizaciju veb elemenata od čega su dva najzastupljenija:
- Opcija 1

```
SampleClass sampleclassPage = PageFactory.initElements(driver, SampleClass.class);
```

  - kreira instancu date klase i postavlja lazy proxy za svaki `WebElements` i `List` atribut koji anotirani pomoću `@FindBy` anotacije
    - Baca izuzetak ukoliko ne uspe instancirati klasa
- Opcija 2
  - Inicijalizacija se može odraditi i na nivou konstruktora

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

LOCIRANJE WEB ELEMENATA

- PageFactory koristi reference na stvarne elemente stranice u cilju inicijalizacije referentnih veb element
- Lokatori se identifikuju upotrebom `@FindBy` anotacije kojom korisnik definiše način lociranja elementa
- Svaki put kada se pozove funkcija za bilo koju interakciju na veb stranici koja koristi ovu promenljivu veb elementa, drajver će prvo ponovo pokušati da pronađe element. Ovo je veoma korisno u slučaju da neki element možda nije vidljiv prilikom samog učitavanja stranice, već nakon izvršavanja neke akcije

```
@FindBy(how= How.XPATH, using="fld_username2")
WebElement usernameTxtBox2;
```

```
@FindBy(xpath = "fld_username2")
private WebElement usernameTxtBox2;
```

- Za elemente koji se inicijalno učitavaju u trenutku učitavanja stranice i naknadno se neće menjati moguće im je pristupiti preko keša. PageFactory definiše i anotaciju `@CacheLookup` za elemente koji ne moraju biti ponovo pronađeni, štedeći vreme i memoriju

```
@FindBy(how= How.XPATH, using="fld_username2")
@CacheLookup
WebElement usernameTxtBox2;
```

- Glavna prednost `@FindBy` anotacije je u tome što omogućava inicijalizaciju elemente stranice bez korišćenja `FindElementa` (ili `FindElements`) metoda

PROBLEMI DOSADAŠNJEG PRISTUPA 2

- Većina veb aplikacija počiva na konceptima asinhronih funkcija, što može dovesti do problema prilikom testiranja (podaci sa servera koji kasne, elementi HTML stranice koji još uvek nisu prikazani)
- Problem može da bude posledica i loše konekcije, pogotovo u slučajevima kada se testira udaljena aplikacija
- Ukoliko pokušamo da pristupimo elementu koji još uvek nije prikazan na stranici dobićemo **ElementNotVisibleException**
- Do sada smo problem rešavali korišćenjem **Thread.sleep()** metode
  - Predstavlja čekanje bez uslova
  - Ne preporučuje se njeno korišćenje u test metodama
- Selenium omogućava dva načina čekanja:
  1. Implicitno čekanje
  2. Eksplicitno čekanje

IMPLICITNO ČEKANJE

- Selenium WebDriver obezbeđuje implicitni metod čekanja za rešavanje problema sinhronizacije
- Implicitno čekanje podrazumeva da će WebDriver čekati određeni vremenski period i u tom periodu neće pokušavati da pronađe element
- Podrazumevano je da implicitno čekanje ima vrednost 0, što dovodi do toga da ukoliko u momentu poziva `findElement` metode, element nije prisutan na stranici, Selenium baca izuzetak
- Izmenom podrazumevanog stanja dobijamo situaciju da će WebDriver nad pozvanom `findElement` (`findElements`) metodom prestati da baca `ElementNotVisibleException` ukoliko element nije pronađen u prvom ciklusu, već će, umesto toga, čekati da istekne određeno vremensko ograničenje
- Jednom postavljeno čekanje se odnosi na sve elemente

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

- Metoda za implicitno čekanje prima dva parametra:
  - Vreme kao Integer vrednost
  - Jedinicu vremena (dan, sat, minuta, sekunda, minuta, milisekunda, mikrosekunda, nanosekunda)

IMPLICITNO ČEKANJE

- Podrazumevano vreme provere prisutnosti elementa je 250 milisekundi (što znači da će WebDriver pretraživati element u DOM-u nakon svakih 250 milisekundi dok se ne pronađe ili dok se navedeno vreme ne završi)
- Implicitno čekanje se može podesiti sa različitim vremenskim okvirima (sati, minute, sekunde itd.)
- Jednom postavljeno, implicitno čekanje se primenjuje za ceo život WebDriver instance
- Implicitno čekanje je deklarisano u jednom redu (obično na nivou setup metoda)
- Ne preporučuje se korišćenje implicitnog čekanja sa velikim brojačem vremena
- Podrazumevana vrednost kašnjenja implicitnog čekanja je 0 sekundi
- Kada se koristi, implicitno čekanje može usporiti izvršenje testa jer će čekati svaki element koji se pojavi u DOM-u

EKSPLICITNO ČEKANJE

- Selenium WebDriver obezbeđuje klase `WebDriverWait` i `ExpectedCondition` za implementaciju eksplicitnog čekanja
- Klasa `ExpectedCondition` obezbeđuje skup unapred definisanih uslova koje treba sačekati pre nego što se nastavi izvršavanje koda
- Za razliku od implicitnog čekanja, eksplicitno čekanje ima nekoliko prednosti:
  - čekanje se svodi na čekanje uspostave određenih uslova. Uslovi mogu biti čekanje na prisustvo veb elementa, čekanje da element postane vidljiv itd.
  - čekanje neće biti izvršeno do isteka timeout-a. Ako je uslov za eksplicitno čekanje zadovoljen, uslov čekanja se napušta i izvršenje se nastavlja
  - čekanje neće biti primenjeno na sve elemente, već samo na određene, na kome je postavljeno

EKSPLICITNO ČEKANJE

- Klasa `WebDriverWait`
  - Pruža mogućnost da se sačeka uslov tokom izvršavanja koda i proveri da li je element prisutan/vidljiv/omogućen/onemogućen itd
- Klasa `ExpectedCondition`
  - Pruža skup uobičajenih uslova (koje možemo da koristimo u komandi WebDriverWait) koji govore WebDriver-u da sačeka pre nego što nastavi sa izvršavanjem koda na osnovu unapred definisanog uslova

EKSPLICITNO ČEKANJE

1	Uslov	Opis
2	alertIsPresent()	Proverava postojanje alert-a
3	elementSelectionStateToBe()	Proverava da li prosleđeni element može biti selektovan
4	elementToBeClickable()	Proverava da li je element vidljiv I enable-ovan kako bi se mogla izvršiti klik komanda
5	elementToBeSelected()	Proverava da li je element selektovan
6	frameToBeAvailableAndSwitchToIt()	Proverava da li je moguće prebaciti se na frame
7	invisibilityOfElementLocated	Proverava da li se element koji odgovara lokatoru ne nalazi ili nije vidljiv u DOM stablu
8	invisibilityOfElementWithText()	Proverava da li se element koji odgovara tekstu ne nalazi ili nije vidljiv u DOM stablu
9	presenceOfAllElementsLocatedBy()	Proverava da li se elementi koji odgovaraju određenom lokatoru nalaze u DOM stablu
10	presenceOfElementLocated()	Proverava da li se element nalazi u DOM stablu
11	textToBePresentInElement()	Proverava da li se prosleđeni tekst nalazi u elementu
12	textToBePresentInElementLocated()	Proverava da li je se prosleđeni tekst nalazi u elementu koji odgovara prosleđenom lokatoru
13	textToBePresentInElementValue()	Proverava da li se prosleđeni tekst nalazi u vrednosti atributa elementa
14	titleIs()	Proverava naslov trenutne stranice
15	titleContains()	Proverava postojanje substringa u naslovu stranice (case sensitive)
16	visibilityOf()	Provereva da li je element DOM stabla prisutan I vidljiv
17	visibilityOfElementLocated()	Proverava da li je element koji pripada lokatoru prisutan i vidljiv
18	visibilityOfAllElements()	Proverava da li su svi prosleđeni elementi prisutni i vidljivi
19	visibilityOfAllElementsLocatedBy()	Proverava da li su svi elementi koji odgovaraju prosleđenom lokatoru prisutni i vidljivi
	Više na:	https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/support/ui/

EKSPLICITNO ČEKANJE

- Klasa `WebDriverWait`
  - Pruža mogućnost da se sačeka uslov tokom izvršavanja koda i proveri da li je element prisutan/vidljiv/omogućen/onemogućen itd
- Klasa `ExpectedCondition`
  - Pruža skup uobičajenih uslova (koje možemo da koristimo u komandi WebDriverWait) koji govore WebDriver-u da sačeka pre nego što nastavi sa izvršavanjem koda na osnovu unapred definisanog uslova