Datum: 2023-02-23

Skrivet av: Admir Ramic

**Dokumentation om uppsättning och testdesignen av mina automatiserade tester**

För mina automatiserade tester använde jag programmeringsspråket python och unittest som testramverk. I början av filen importerade jag alla nödvändiga moduler from selenium och unittest. Eftersom jag inte hade många tester som jag skulle kunna kategorisera efter olika områden som tester utförs på, utan bara 7 tester som testar olika saker, så skaffade jag bara en fil och en klass för mina tester.

Min WebTest klass börjar med en setup som påverkar hela klassen och det är en classmethod som heter setUpClass som utför vissa moment som ska göras bara en gång innan alla tester. Den importerar driver till webbläsaren (i mitt fall Chrome driver) och skapar ett objekt som heter ”driver” och som ska sedan användas i alla testmetoder. Jag skapade också ett objekt som heter ”mywait” av klassen ”WebDriverWait för att sedan använda det för explicitly wait på vissa ställen som har synkroniseringsproblem. I classmethoden setUpClass skrev jag också kommando ”cls.driver.maximize\_window()” som ska maximera browserfönster i början av exekvering av mina tester.

I slutet av alla tester har jag skapat en teardown som också påverkar hela klassen. Det är en klassmetod som heter ”tearDownClass” och som innehåller kommando ”cls.driver.quit” som stänger av browsern helt efter att alla tester har körts klart.

Efter klassmetoden ”setUpClass” har jag skapat en metod som heter ”setUp” som körs i början av varje testmetod. Den hämtar hemsidan jag vill testa, i mitt fall <https://www.axesslogistics.se> och accepterar cookies genom att klicka på ”acceptera” i pop-upp fönstret. Med andra ord skapar den förutsättningar för en test.

Jag har också skapat en teardown metod som heter ”tearDown” och som körs efter varje test. Den rensar cookies med hjälp av kommando ”self.driver.delete\_all\_cookies()”. Metoden ser till allt ska rensas från föregående test så att det inte kan påverka andra tester.

**Mina tester:**

**def test\_link\_path(self):**

Testet verifierar att länken ”VÅRA TJÄNSTER” leder till förväntad rubrik, med samma namn. Webbelement av länken ”VÅRA TJÄNSTER” är först sparad i variabeln ”link\_våra\_tjänster” och från den är sedan extraherad text som är sedan sparad i variabeln ”expected”. Sedan öppnades artikeln ”VÅRA TJÄNSTER” genom att utföra ”click()” kommando på ”link\_våra\_tjänster”. Efter att artikeln ”VÅRA TJÄNSTER” öppnas, sparas text från rubriken i variabeln ”actual”. Sedan jämförs ”expected” med ”actual” och om de är likadana, testen har passerad. Jag har lagt till metoden ”upper()” för både ”actual” och ”expected” för att vara säker att testen ska passera även om den skulle jämföra stora med små bokstäver.

**def test\_change\_language(self):**

Testet verifierar att byta språkfunktionen verkligen byter språk från svenska till engelska. Den börjar med att hitta webbelement till knappen ”SPRÅK” och utföra ”click()” aktion på den. Sedan hittar man webbelement till knappen ”ENGLISH” och utför aktion ”click()” på den. Efter detta jämför man om texten på språkknappen har bytt text till ”LANGUAGE”. Om ja, testet har passerat och det är grönt.

**def test\_adresss(self):**

Testet verifierar att Malmö anläggning finns med under fliken ”KONTAKT/SUPPORT” och om adressuppgifterna stämmer med förväntade. Webbelement till knappen ”KONTAKT/SUPPORT” hittades och utfördes ”click()” aktion på den. När sidan med kontaktuppgifter öppnas, då jämförs text som extraheras ur webbelementen som visar adressuppgifterna med förväntade. Om de är samma, då har testet passerat.

**def test\_invalid\_login(self):**

Med det här testet verifierar man att felmeddelande ”Felaktigt användarnamn eller lösenord” visas när man försöker logga in med ogiltiga uppgifter. Genom att hitta rätta webbelement, man utför ”click()” actions för att komma till inloggningssidan. Sedan med kommandon ”send\_keys” skriver man påhittade inloggningsuppgifter och sedan utför man ”click()” action på logga in webbelement. Efter detta hittar man webbelement som föreställer pop-upp fönster och extraherar text ur den. Sedan jämför man texten med förväntad text ”Felaktigt användarnamn och lösenord. Om de är likadana, testet är OK.

**def test\_back\_to\_main\_page(self):**

Testet testar om man kommer tillbaka till företagets startsida när man klickar på logon ”Axess Logistic”. Man förbereder testet genom att först spara startsidans titel i en variabel som heter ”main\_title” och sedan hittar man webbelement som föreställer knappen ”nyheter” och utför ”click()” action på den. När man har kommit till sidan ”Nyheter” sparar man sidans titel i en variabel och sedan jämför den med den förväntade för att säkerställa att man har kommit till nyheter. Sedan utför man ”click()” action på webelementet som föreställer logon och sedan jämför om startsidans titel i variabeln ”main\_titel” är likadan som titel i hemsidan vi har kommit till. Om de är likadana och om den första assertion lyckades, testet är OK.

**def test\_right\_news\_artickle(self):**

Testet verifierar att man har fått rätt nyhetsartikeln som länken hänvisar till. Det är mycket likt testet ”def test\_link\_path”.

**def test\_open\_linked\_homepage(self):**

Man gör testet för att verifiera att byta land till Norge verkligen leder till Norges hemsida. Testet verifierar att man har kommit till Norges hemsida genom att jämföra URL från Norges hemsida med förväntad ”https://www.axesslogistics.no”. Om det stämmer, testet är OK.