

Testautomatisering med CI

Skriftlig del



Nina Persson

EC Utbildning, MVT22

2023-03-27

# Testverktyg väljer också testområde

Svårt men spännande, det tog ganska mycket tid och funderande innan jag hittade rätt.

## Funderingar innan valet

API eller inte API, det är frågan.

Då man skulle göra allt själv utan varken hjälp från skolkamrater eller lärare så förstod jag att jag inte kunde göra det för svårt, samtidigt ville jag ju lära mig ordentligt och något som man skulle ha nytta av på en framtida arbetsplats.

Jag tänkte först att testa API:er var det som skulle imponera mest både på mig själv, lärare och framtida chefer men det kändes också som det svåraste och faktiskt också det tråkigaste. Jag visste inte var det fanns roliga API:er som vi kunde få tillgång till heller.

Hade jag fått lära mig det bättre under teorilektionerna så hade jag kanske vågat ge mig på det men som det var nu var det för svårt. Och SoapUI gav heller inte inspiration till att gå djupare in i det.

Mot slutet av funderingsperioden hittade jag Postman som verktyg för att testa API:er och det såg lite roligare ut men då hade jag nog redan bestämt mig för vilken hemsida jag ville GUI-testa så det var redan försent.

Jag hade kunnat testa API:er på Spotify visade det sig men tyckte det var lite för tråkigt det också.

Varför jag verkligen ens övervägde API-testning berodde på att vi hade fått lära oss att det var bättre att automatisera API:er än end-to-end-tester enligt testpyramiden men efter mycket om och men så brydde jag mig inte om det. Som första testautomationsövning ever fick det bli något roligt och lättare men som ändå kunde vara bra att kunna.

# Val av testverktyg

Så, då valet till slut föll på GUI-testning och jag hade min hemsida klar så var det bara verktyg kvar att välja. Vi hade ju fått en genomgång av läraren av både Selenium IDE och Selenium Webdriver så någon av dessa kände jag nog att det skulle bli.

När jag först provade Selenium IDE så kändes det väldigt enkelt och hanterbart och jag tänkte att jag skulle börja där för att bara få ihop 5 snabba tester men sedan gå över till Selenium Webdriver om jag hade mer tid. Problemet var bara att när jag skulle börja med testerna till inlämningsuppgiften så fungerade ingenting, jag hade helt glömt bort hur man skulle göra. Dessutom var jag rädd att det inte skulle gå att köra IDE:n tillsammans med Jenkins. Ett ytterligare problem som talade emot Selenium IDE var att jag kom ihåg att jag tyckte att det inte fanns så många roliga alternativ för asserts, det gick inte att säga ”contain” i texter eller webbadresser t.ex. utan man var tvungen att asserta en hel text. Det kändes för begränsande.

Då tänkte jag att okej då, jag får hoppa på det både coolare och läskigare Selenium Webdriver men då jag just hade kämpat mig igenom just automatiska tester så kände jag att det skulle bli roligt också. Jag hade ju redan en hel del testmallar från den tidigare kursen och det fanns ju internet och chatGPT till min hjälp. Det kändes även bättre att kunna säga att jag hade gjort det till en ev. senare arbetsgivare samt att det var mycket mer flexibelt än Selenium IDE gällande vilka tester man kunde göra.

# Testformat

Testerna sattes upp enligt formatet Arrange - Act – Assert där man först öppnar hemsidan, sedan agerar (söker igenom hemsidan och klickar på saker t.ex.) och sedan ber man programmet testa/verifiera något på sidan. Mer om hur exakt jag har gjort det kommer under nästa rubrik.

Först tänkte jag att jag skulle vara lite avancerad och göra som läraren hade gjort, d.v.s. skapa tre olika filer:

main = test\_tui.py/test\_scarenergy.py

Metodsamlingsfil = helper\_tests.py

Konstantsamlingsfil = page\_objects.py

men jag har tagit bort dem igen eftersom det blev mycket krångligare och svårare att ha överblick, dessutom går det bra att lägga alla förberedelser och konstanter i Setup-metoden så jag kände att det var mer rimligt att ha både Setup och Teardown i samma fil som de övriga testerna. Testerna är heller inte så många och testar just nu ganska samma sak, så det kändes heller inte meningsfullt att dela upp dem. På det här sättet har man mer överblick och testen berättar en historia och blir mer och mer avancerade för varje test.

Det hela börjar med en setup som startar webdrivern och sedan hämtar/öppnar hemsidan den ska testa. Det sista som händer är att fönstret maximeras. Denna setup sker inför varje test i filen.

”Teardown:en” startar efter varje test i filen och tar bort alla cache och cookies samt stänger ner testfönstret. Den återställer state:n till som den var innan testet kördes för att inte skapa några beroenden eller konstigheter.

# Testdesign

Jag hade lite svårt att veta vad som var viktigt att testa och det stoppade upp mig lite i uppgiften men så sökte jag på nätet och hittade en hemsida där jag fick lite tips: https://www.nexcess.net/blog/ecommerce-website-testing-test-cases/

De tester jag har valt att utföra är således bl.a.:

* Att en produktsida visas
* Att köpsidan visas
* Att en kund kan lägga i varor i varukorgen
* Att varukorgen visas
* Att kunden kommer till kassan

Jag tycker att testen och dess uppbyggnad är väldigt väl förklarade i testfilen redan så det känns som att om jag förklarar dem helt igen att det blir väldigt överflödigt. Men säg till om jag även ska göra det i denna inlämning så gör jag det.

Jag tycker det har varit kul och lärorikt att leka lite med olika typer av asserts och inte bara ta samma hela tiden.

Vissa av testerna är tyvärr lite flaky och fungerar bara ibland, speciellt ”test\_add\_shopping\_cart”. Där har jag fått lägga till fler och fler explicit waits tills att nu varje moment har en wait, ändå så failar testet fortfarande ibland och man får ett meddelande om att ”cart” inte hittas i current URL t.ex. Men jag känner inte att det går att göra så mycket mer åt det nu. Jag valde explicit waits för att det var en rekommendation från läraren och testvärlden. Jag har hela tiden också valt full XPATH som element locator då den är den mest specifika locator:n.

Jag har även försökt att hålla mig till principen om att ett test ska undersöka/verifiera en sak men just i ovanstående test så finns det två asserts i samma test.

# CI och Pipeline

Jag valde Jenkins som CI-verktyg då det var det som läraren visade och jag knappt ens efter teorin förstod vad CI var för något. Att lära sig tre nya program och processer under en kurs kändes som sagt ganska övermäktigt så jag fick ta något som kändes ”enkelt”, men ännu mer, något som läraren kunde och kunde hjälpa mig med.

Jag satte upp Jenkins att skanna commits mot Github.

Jenkinsfilen är basen i det hela och det som får testerna att köra och den blev väldigt enkel. En sak som vi råddes att göra var att ladda ner och köra Selenium Webdriver och alla dess plug-ins på en virtual environment men då jag tyckte det lät läskigt och svårt så gjorde jag inte det utan allt finns lokalt på min dator, därför ”agent: built-in” nedan.

pipeline {

agent { label "built-in" }

stages {

stage("Running tests") {

steps {

bat "python -m unittest test\_scarenergy.py"

echo "Testing"

}

}

}

post("Cleanup") {

always {

cleanWs()

echo "Cleaning up"

}

}

}

Stages kör programmet och där lägger man in kommandot för hur man triggar i gång testerna: "python -m unittest test\_scarenergy.py".

När alla tester körts igenom så kör man även en ”clean up” för att återställa state:n som den var innan testerna kördes, precis som tearDown-metoden i testfilen gör.

Den enda andra uppsättning vi gjorde som inte var default var att vi sa till Jenkins att leta efter nya commits med periodic scan. Det var tydligen något problem med att göra helautomatiska triggar.