Szakdolgozat

Tóth László

2009

Debrecen

Debreceni Egyetem

Informatikai Kar

Automatizált tesztelés Seleniummal

Témavezető:

Dr. Bujdosó Gyöngyi

Egyetemi tanársegéd

Szerző:

Tóth László

Programozó Matematikus

Tartalomjegyzék

1. Bevezetés

Egy szoftver fejlesztése során - különös tekintettel a szoftverek egyre növekvő komplexitására - egyre nagyobb szerepet kap a szoftverek minőségének biztosítása, amelynek kulcsfontosságú eleme a tesztelés. A fejlesztés során az alkalmazott korszerű fejlesztési eszközök és módszerek ellenére elkerülhetetlenül hibák kerülnek a szoftverekbe, átlagosan 30-40 hiba található az elkészült program minden 1000 sorában. Ezeknek a hibáknak - vagy legalábbis döntő többségüknek - a felderítése és kijavítása elengedhetetlen a szoftver biztonságos használatához. Ehhez nyújt segítséget a tesztelés, ami tulajdonképpen a szoftver használata és működésének megfigyelése, dokumentálása szimulált környezetben. A tesztelés során olyan környezetet kell kialakítani, amely lehetőleg a szoftver használatának minden területére, a tesztelt program működésének minden szélsőséges esetére kiterjed. A tesztelés során dokumentálni kell a szoftver összes, az elvárttól eltérő működését. További fontos tényező, hogy a szoftvertesztelés nem csak a szoftver elkészültéig tart, hanem végig kíséri a szoftver teljes életciklusát, a későbbi újabb verziók kibocsátását. Szakdolgozatomban bemutatom a teszteléshez kapcsolódó fogalmakat, folyamatokat, majd bemutatok egy eszközt, ami a tesztelői munkát segítve, úgymond „automatizálja” azt. Ennek az eszköznek a neve: Selenium. Végül a működését egy rövid példán keresztül is bemutatom.

2. Tesztelés során előforduló fogalmak, definíciók

A rendszertesztelés során elég sok teszteléssel kapcsolatos szakkifejezés megjelenik. Ezek a fogalmak a következők:

- Összehasonlító teszt (benchmark): olyan teszt mely egy új, vagy ismeretlen rendszer alkalmas teljesítmény paramétereit hasonlítja egy már ismert rendszer jellemzőihez, melyeket referenciának tekint.

- Hiba, hiány (defect): azon ellentmondások, eltérések, anomáliák, különböző súlyosságú hibák, felvetődött, s nem tisztázott kérdések vagy változtatási kérések összefoglaló neve, melyek a tesztelési fázis folyamán merülnek fel. Ez a tesztelés legfontosabb eredménye.

- Egységteszt: a rendszer egységeinek, különálló részeinek funkcionális tesztelésére szolgál. Az egységek egymástól független működési képességét vizsgálja, ahol az egység lehet egy szoftver (függvény, vagy gomb), illetve az üzleti folyamat szempontból különálló rész is.

- Felhasználói elfogadási teszt (UAT): a UAT a projekt egyik befejező fázisa, s jellemzően azelőtt fordul elő, hogy a kliens, vagy megrendelő elfogadná az új rendszert. A teszt eredménye azt mutatja meg a megrendelő számára, hogy a rendszere hogyan fog működni éles körülmények között.

- Funkcionális teszt: tesztelési stratégia, mely arra fókuszál, hogy a rendszer funkciói, szervizei, metódusai és felhasználói igényei a szoftver specifikációban foglaltaknak megfelelnek e. Az adat-, és adatbázis integrációs tesztek szintén a funkcionális tesztelés feladatkörébe tartoznak.

    - Általános funkcionális teszt: A rendszer működésének vizsgálata normál működés esetén. A teszt során ellenőrizzük, hogy a rendszer funkciói az elvártnak megfelelően működnek.

    - Szélsőérték funkcionális teszt: A rendszer működésének vizsgálata szélső bemeneti/kimeneti értékek esetén. A teszt során ellenőrizzük, hogy a rendszer funkciói az elvártnak megfelelően működnek.

- Installációs teszt: megvizsgálja, hogy az alkalmazás korrekt módon végrehajtható-e különböző hardver és szoftver környezetben és extrém körülmények között, mint pl.: nincs elég tároló kapacitás, vagy hirtelen megszűnik a hálózati feszültség ellátása. A rendszer telepítésének tesztelése különböző eshetőségek (különböző hardver-szoftver konfigurációk, feltételek) esetén

- Konfigurációs teszt: teszteli, hogy a rendszer különböző hardver- és szoftverbeállítások mellett is teljesíti-e a specifikációs követelményeket.

- Mennyiségi teszt (Volume teszt): a teszt célja megvizsgálni, hogy az alkalmazás tudja-e kezelni azt az adatmennyiséget és minőséget, melyet a specifikáció megkövetel.

- Biztonsági vagy Hozzáférési teszt: az ilyen típusú tesztek célja, hogy ellenőrizze a szoftver hozzáférési jogainak rendszere az előre definiált szabályoknak megfelelően működik. (pl. egy adott függvény csak a megfelelő jogokkal rendelkező felhasználó által érhető el)

- Lefedettség teszt: tesztelési stratégia, mely megmutatja, hogy a szoftver meglévő forráskódjából mekkora rész (hány %) aktivizálódik a teljes használat során. Nagy segítséget jelent a redundáns részek felderítésében, s ezzel hatékony kódolási gyakorlatra sarkall.

- Használhatósági teszt: egy olyan vizsgálati fajta deríti fel, hogy a felhasználók mennyire jól tudják használni az ember által előállított objektumokat (úgy, mint web oldal, számítógép interfész, egy dokumentum, vagy egy eszköz) jövőbeli céljaik elérésének érdekében. A használhatósági teszt egy adott objektumot, vagy objektumok egy kisebb csoportját vizsgálja, miközben az általánosan elfogadott ember-számítógép kapcsolat interakcióit használja.

- Hiba-tolerancia teszt: ebben az esetben a rendszert direkt módon hibás/nem megfelelő adatokkal, kapcsolatokkal teszteljük azt vizsgálva, hogy az alkalmazás funkcionalitása hogyan kezeli a különböző hibákat. (szükségességi ellenőrzés, validáció, hibaüzenetek, stb.)

- Integrációs teszt: az integrált rendszer széleskörű vizsgálata, ide értve a kapcsolódó rendszereket és interfészeket. A teszt célja az egész üzleti folyamat vizsgálata, a kapcsolódó rendszerek és modulok együttműködésének ellenőrzése.

- Rendszerteszt: a rendszer (alkalmazás) funkcionalitásának önmagában való tesztelése, egyéb kapcsolódó rendszerek figyelembe nem vételével.

- Regressziós teszt: felderíti, hogy a verziók közötti változtatások okoznak e problémákat a rendszer azon részeiben, melyen direkt módon nem lettek megváltoztatva.

- RUP: A Rational Unified Process egy iteratív és növekvő szoftver mérnöki módszertan, mely az UML (Unified Modelling Language) szabványos jelölésrendszerét használja. A fejlesztés esetében ezen módszertan (mely teljesíti a követelményeket) segítségével a végső rendszer egymást követő iterációk alapján épül fel. A rendszer új, kibővített változata az egymást követő, újabb iterációkon alapul, melyek az előző iteráció eredményeit veszik figyelembe.

- Strukturált teszt: a tesztelendő rendszer (modul) belső struktúráján (forráskód) alapul, kiegészítve forráskód lefedettségi vizsgálattal. Ugyanaz a vizsgálat, mint a fehér-doboz teszt.

- Load teszt: vizsgálja, hogy a rendszer működési korlátai elfogadhatóak-e különböző mértékű (növekvő) szimulált terhelések alatt. Fontos, hogy egy adott mérési folyamat során a tesztelt rendszer konfigurációján nem változtatunk. A mérések a terhelési paraméterek és a válaszidők kapcsolatát vizsgálják, mely paraméterek mérésenként különböznek. Ezek a terhelési paraméterek lehetnek pl. a terhelő felhasználók száma, a végrehajtott tranzakciók száma, stb.

- Stabilitás teszt: átlagos, vagy annál alacsonyabb terhelésnek veti alá a vizsgált rendszert, de a terhelés hatásait hosszabb távon vizsgálja (2-3 órától akár 24-36 óráig). Ezen mérés során felderíthető, hogy a vizsgált rendszer nem fogyaszt-e el pl. memória), vagy nem telít-e be (pl. tároló kapacitás) valamilyen erőforrást, amitől a rendszer összeomolhat.

- Teszteset: a tesztelés alapegysége. Kialakítása egy-egy adott rendszerfunkció működését tükrözi. Definiálja a felhasználók feladatait, a funkció bemenő és elvárt kimenő adatpárosait és a vizsgálat alatt elvárt működést.

- Tesztkörnyezet: a vizsgált rendszer egy specifikus „másolata” egy adott hardver-szoftver környezetben. Azért állítjuk elő, hogy a vizsgált rendszer működését mérni és monitorozni tudjuk.

- Teszt szkript: egymást követő lépések instrukciói, melyek egy adott folyamat pontos végrehajtásához szükségesek. A teszteket ezen instrukciók alapján kell végre hajtani. A formális megjelenése ezeknek az instrukcióknak a következőek lehetnek:

    - szöveges formában dokumentált instrukciók, melyek a sikeres manuális végrehajtáshoz szükségesek

    - parancsok gyűjteménye, mely egy tesztelői program segítségével automatikusan végrehajtható

- Versenyteszt, Kompetitív teszt: megvizsgálja, hogy a rendszer képes-e helyesen kezelni a többszörös, egyidejű hozzáférést adott erőforrásokhoz (pl. adatbázisok, memória blokkok, stb.).

- Teljesítménytesztelés

    - Általános teljesítményteszt: A rendszer működésének sebességét összevetjük az elvárt értékekkel. A teszt segítségével fényt deríthetünk a rendszersebesség szempontjából kritikus pontjaira, szűk keresztmetszeteire.

    - Referencia teljesítményteszt: A rendszer működésének sebességét egy másik, a tesztelt rendszerhez hasonló funkcionalitású rendszer sebességével vetjük össze.

    - Terheléses teljesítményteszt: A rendszer sebességének vizsgálata különböző munkaterhelések esetén. A teszt során rögzítjük a terhelés mértékét (pl. felhasználók, tranzakciók száma), valamint a rendszer működési sebességét az adott terhelés alatt.

- Konkurencia teljesítményteszt A rendszer teljesítményének vizsgálata abban az esetben, ha a rendszernek többszörösen kell ugyanahhoz az erőforráshoz (pl. ugyanahhoz az adatrekordhoz) hozzáférnie.
- Sokk teljesítményteszt: A rendszer működésének vizsgálata extrém körülmények között. Ezek az extrém körülmények lehetnek a tervezettnél nagyobb terhelések, kevés memória/erőforrások, hardver problémák, áramszünet, stb.

3. Tesztelési technikák

Ahhoz, hogy egy szoftvert tesztelni tudjunk, szükségünk van egy, a szoftver elvárt működésére vonatkozó modellre. A specifikáció alapján előállított szoftvertermék működését össze kell hasonlítani a specifikációnak tökéletesen megfelelő referencia egység működésével. Teszteléskor a szoftver specifikációja és egyéb követelmények alapján meghatározott teszteseteket használunk a teszt elvégzésére.

3.1 Általános tesztkörnyezetek

Tesztkörnyezet alatt azt a speciális szoftver és hardver környezetet kell érteni melyben a tesztelést elvégezzük. A tesztkörnyezetek különböző szolgáltatásokat nyújtanak a tesztek módszeres elvégzéséhez. A tesztelt szoftvert futtató tesztelést dinamikus tesztnek nevezzük. A tesztkörnyezetek több komponensből épülnek fel, ezek a következők: a teszteket végrehajtó komponens(test driver, test harness, test bench), teszt automatizáló és teszt konfiguráció manager komponens (test CM), a tesztelendő egység "beműszerezésére" szolgáló komponens (instrumentation) és a tesztelt egység külső szoftver környezetét szimuláló csonkok (stubs).

3.2 Tesztelő alkalmazások

A tesztelő alkalmazások valamilyen formában biztosítják az előbb leírt tesztkörnyezetet egy adott hardver (rendszer architektúra) ill. szoftver környezetben (operációs rendszer, programozási nyelv). A szolgáltatások köre általában a következő: teszt konfiguráció management, automatizált tesztelés, debugger, teljesítmény tesztek és profilok készítése, statikus analízis, az eredmények dokumentálása és megjelenítése.

3.3 Teszt konfiguráció management

A teszt konfiguráció management célja, hogy szabályozottan biztosítsa a megfelelő szoftver egységek tesztelését a megfelelő tesztesetekkel és ezt megfelelően dokumentálja. Ez a rendszer teremti meg a teszt végrehajtó komponens működésének feltételeit és működteti azt. Az alábbi folyamatok adminisztrációját kell elvégeznie: cél egység "beműszerezése", a tesztelő által megadott tesztesetek és teszt paraméterek alapján a teszt beállítása, majd pedig a teszt elvégzése és az eredmények kezelése.

3.4 Beműszerezés

A teszteléshez szoftveregységet felszereljük a működéséről tájékoztató, a teszt megkívánta helyzetek azonosításához, lefedettségi adatok szolgáltatásához és a teszt közbeni beavatkozáshoz szükséges kapcsolódási pontokkal. Ez a "műszerezettség" teremti meg a kapcsolódási pontokat a teszt végrehajtó komponens és a tesztelt egység között. Két féle beműszerezésről beszélhetünk, a forráskód ill. a végrehajtható kód beműszerezéséről. A forráskód felszerelése esetén debuggolást segítő, adatokat naplózó, kontextusokat azonosító program kódot írunk a szoftver forráskódjába. Ide tartozik még a külső csonkok, csomagoló osztályok megírása is. A forráskód műszerezés széles körben alkalmazható és nagy szabadságot biztosít a teszteléskor de nehéz karbantartani, megnöveli a fordítás/tesztelés ciklus idejét és nem utolsó sorban módosítja az eredeti kódot, aminek mellékhatásai lehetnek. A végrehajtható kód műszerezése fordításkor történik, a forráskód változtatása nélkül. Ilyen jellegű műszerezés pl. a végrehajtható kódot a forráskóddal összekapcsoló debug információk elhelyezése. Léteznek ún. patch szintű eszközök melyek a forráskód nélkül, a végrehajtható kód futásakor lépnek működésbe (dynamic action linking). A végrehajtható kód műszerezés csak korlátozottan használható, de ugyanakkor működés közben csak minimálisan zavaró.

3.5 Modulteszt és JUnit

A komponenseket összeépítés előtt is tesztelni kell. Ha a termék nagyon bonyolult, akár több lépcsőre is szükség lehet (egy összeépített rendszer is esetleg csak egy komponens egy nagyobban). A modultesztek célja a tesztelendő komponens minél alaposabb "megmozgatása" az összeépítés előtt. A megmozgatás a programmodul nyilvános és kevésbé nyilvános metódusainak alapos végighívogatását jelenti. Előfordulhat, hogy az alapos végighívogatáshoz nem elegendőek a nyilvános metódusok, ez esetben a modulteszt céljára külön tesztmetódusokat szokás nyitni, amelyek a modul nehezen elérhető részeihez engednek hozzáférést. A modulteszt sikerességének mérőszáma a lefedettség (coverage). Ez azt jelenti, a tesztelendő modul hány százalékát "mozgatta meg" a tesztprogram, vannak-e olyan részek, amelyek nem hajtódtak végre (tehát a hibáik nem derülhettek ki). Kétféle lefedettséget szokás használni modulteszt során: sor vagy függvénylefedettséget. Értelemszerűen a sorlefedettség azt mondja meg, a modul összes sorának hány százaléka hajtódott végre a teszt során, míg a függvénylefedettségnél a végrehajtott függvények/összes függvények arány számít. A JUnit nem képes önmagában lefedettséget mérni, erre más eszközök vannak (pl. Rational

PureCoverage vagy a Sitraka JProbe Coverage) viszont a keretrendszer kis mérete miatt kellemesen használható lefedettségmérő eszközökkel.

A JUnit eredetileg az IBM által fejlesztett, de jelenleg szabad forrású szoftverként hozzáférhető modulteszt-keretrendszer. Fő funkciói a következők:

- Tesztek automatikus futtatása egyben vagy részenként
- Teszteredmény-áttekintés és kijelzés
- Hierarchikus tesztstruktúra-támogatás
- Tesztvégrehajtás többféle felületről

Néhány dolog, amire a JUnit nem képes:

- Tesztek tervezése
- Automatikus tesztprogramgenerálás
- Lefedettség és teljesítménymérés

A JUnit ennek ellenére sokat segít a modultesztek írásában, tehát érdemes megismerkedni vele.

3.6 Automatizált tesztelés

Az automatizált tesztelés két dolgot foglal magában. Az egyik az automatizált teszteset generálás, másik pedig az automatizált tesztvégrehajtás. Az automatizált teszteset generálás általában strukturális (white-box) tesztek készítésénél működik, az alkalmazott technika általában szimbolikus végrehajtáson alapul. Amennyiben a szoftver specifikálására valamilyen formális keretben került sor, úgy ebből szintén lehetséges automatizált teszteset származtatás (black-boksz tesztelés). Tesztesetek létrehozását segítő eljárások lehetnek pl. teszt templatek használata ill. teszt "factory" osztályok használata objektum orientált rendszerek tesztelésénél. Az automatizált tesztvégrehajtás lehetővé teszi, hogy a teszt konfiguráció manager vezérelje a tesztek végrehajtását, a tesztek megismételhetők legyenek (regressziós tesztelés), ill. nagy mennyiségű tesztet lehessen végrehajtani. Az alkalmazott megoldások miatt szét kell választani a parancssor interfészű és a grafikus felhasználói

felületű szoftverek automatizált tesztelését. Az első esetben a tesztadatok és eredmények egyszerű be- kivitel átirányításával adat alapú tesztelést végezhetünk (egyszerű test driver). Grafikus interfészek teszteléséhez léteznek makró rögzítő rendszerek, de ezek csak korlátozottan használhatók. Hatékonyabb automatizálást tesz lehetővé az adat alapú architektúra vagyis a tesztesetek szemantikus adatainak függetlenítése a konkrét felhasználói felületre vonatkozó adatoktól. Használható megoldás még keretrendszer alapú architektúra alkalmazása. Ilyenkor egy magas szintű tesztdefiniáló környezetet hoznak létre, melynek primitívjeit a grafikus felületen végrehajtható műveletek jelentik. Mindkét módszer a tesztek karbantartását, újrahasznosítását könnyíti meg.

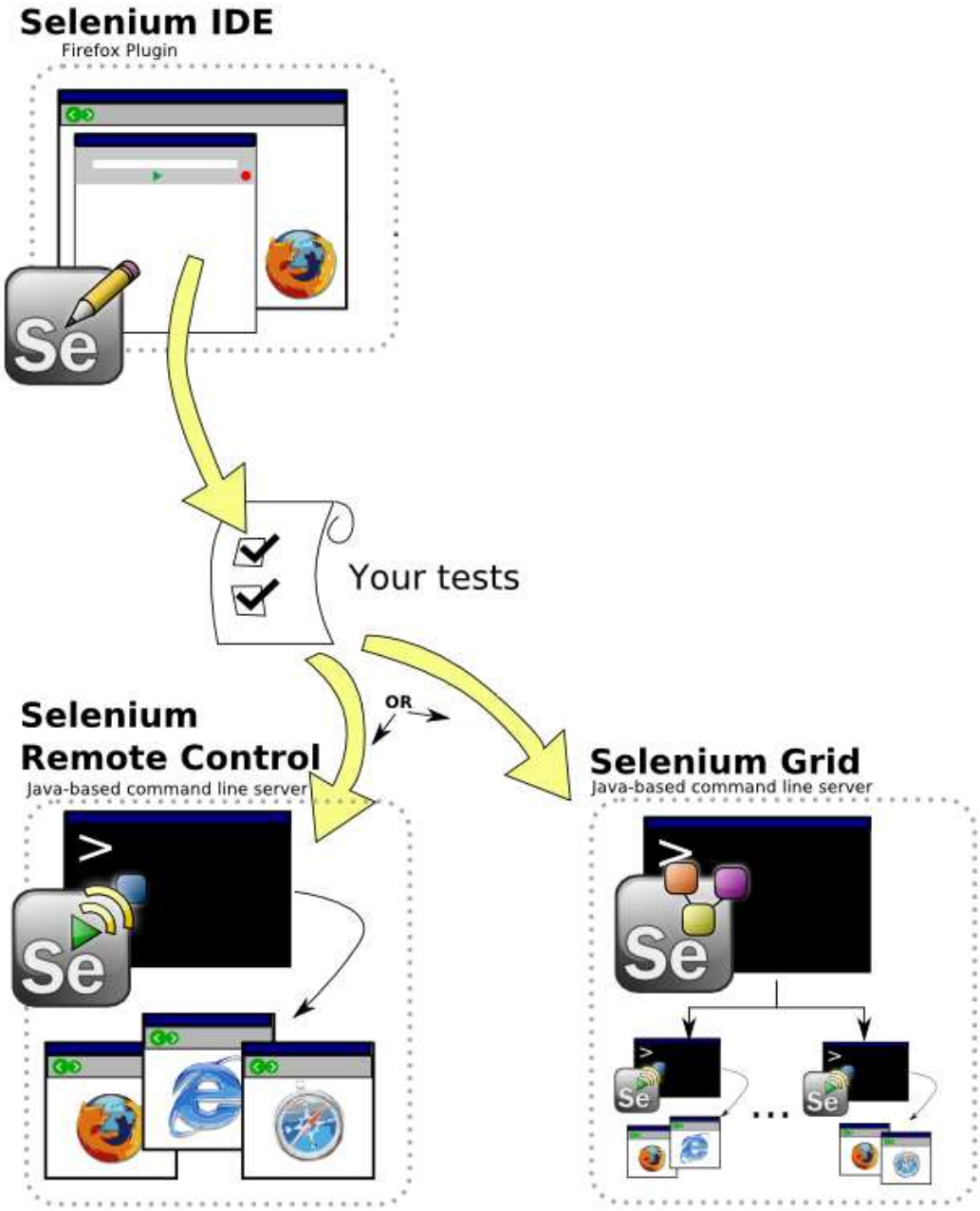
4. Selenium

4.1 Történeti áttekintés

A Selenium egy hordozható szoftvertesztelő keretrendszer webes alkalmazásokhoz. A teszteket el lehet készíteni HTML-ben és még számos népszerű programozási nyelven. A tesztek futtatására számos böngésző alkalmas. A szoftver alkalmazható Windowson, Linuxon és Machintoson egyaránt. A Seleniumot 2004-ben Chicago-ban a ThoughtWorks-nél fejlesztette ki egy programozókból és tesztelőkből álló csapat. A szoftver nyílt forráskódú, szabadon letölthető és ingyenesen használható.

4.2 Selenium működése

A Selenium működése az alábbi ábrán nagyon jól lekövethető.



1. Selenium működése

Az ábra rövid leírása:

- Selenium IDE rész

A Selenium IDE egy Firefox plugin, aminek segítségével rögzíthetjük és lejátszhatjuk a teszteket. A Selenium IDE többféle programozási nyelven tartalmazza a rögzített teszteket, pl.: Ruby, Perl, PHP, Java, C#, Python, amiket kilehet exportálni és be lehet helyezni egy már létező tesztkörnyezetbe.

- Your tests rész

A Selenium parancsokkal elkészített tesztet elküldi a Selenium Remote Control vagy a Selenium Grid szervernek.

- A Selenium Remote Contol rész

A Selenium Remote Contol elindít egy böngészőt és lefuttatja a rögzített tesztet.

- A Selenium Grid rész

A Selenium Grid több Selenium Remote Control szervert koordinál, így több platformon is tudjuk futtatni a rögzített tesztet.

4.3 A Selenium által támogatott platformok

- Böngészők

Böngészők	Selenium IDE	Selenium Remote Control	Selenium Core
Firefox 3	Tesztek rögzítése és lejátszása	Böngésző indítása, teszt futtatása	Tesztek futtatása
Firefox 2	Tesztek rögzítése és lejátszása	Böngésző indítása, teszt futtatása	Tesztek futtatása
Internet Explorer	Nem támogatott	Böngésző indítása, teszt futtatása	Tesztek futtatása
Safari	Nem támogatott	Böngésző indítása, teszt futtatása	Tesztek futtatása
Opera	Nem támogatott	Böngésző indítása, teszt futtatása	Tesztek futtatása

- Operációs Rendszerek

Operációs rendszer	Selenium IDE	Selenium Remote Control	Selenium Core
Windows	Működik a Firefox 2-vel vagy magasabb verzióval	Böngésző indítása, teszt futtatása	Tesztek futtatása
OS X	Működik a Firefox 2-vel vagy magasabb verzióval	Böngésző indítása, teszt futtatása	Tesztek futtatása
Linux	Működik a Firefox 2-vel vagy magasabb verzióval	Böngésző indítása, teszt futtatása	Tesztek futtatása
Solaris	Működik a Firefox 2-vel vagy magasabb verzióval	Böngésző indítása, teszt futtatása	Tesztek futtatása

- Programozási nyelvek

Programozási nyelv	Selenium IDE	Selenium Remote Control	Selenium Core
C #	Generálja a kódot	Könyvtár (driver) támogatás	n / a
Java	Generálja a kódot	Könyvtár (driver) támogatás	n / a
Perl	Generálja a kódot	Könyvtár (driver) támogatás	n / a
PHP	Generálja a kódot	Könyvtár (driver) támogatás	n / a
Python	Generálja a kódot	Könyvtár (driver) támogatás	n / a
Ruby	Generálja a kódot	Könyvtár (driver) támogatás	n / a

4.4 Selenium IDE

A Selenium IDE egy integrált fejlesztési környezet Selenium tesztek készítéséhez. Firefox kiegészítőjeként hozták létre, lehetővé téve, hogy teszteket rögzítsünk, lejátszunk és debuggoljunk vele. A Selenium IDE tartalmazza a teljes Selenium Core-t, amely lehetővé teszi, hogy könnyen és gyorsan rögzítsünk és lejátszunk teszteket olyan környezetben, ahol futni fog.

A Selenium IDE nem csak a tesztek felvételére szolgáló eszköz, hanem egy komplett IDE. Ki lehet választani, hogy a felvételi módot használjuk, vagy egyszerűen kézzel szerkeszthetjük a saját scriptjeinket, amit aztán „le is játszhatunk” ezzel futtatva azokat. Az automatikus

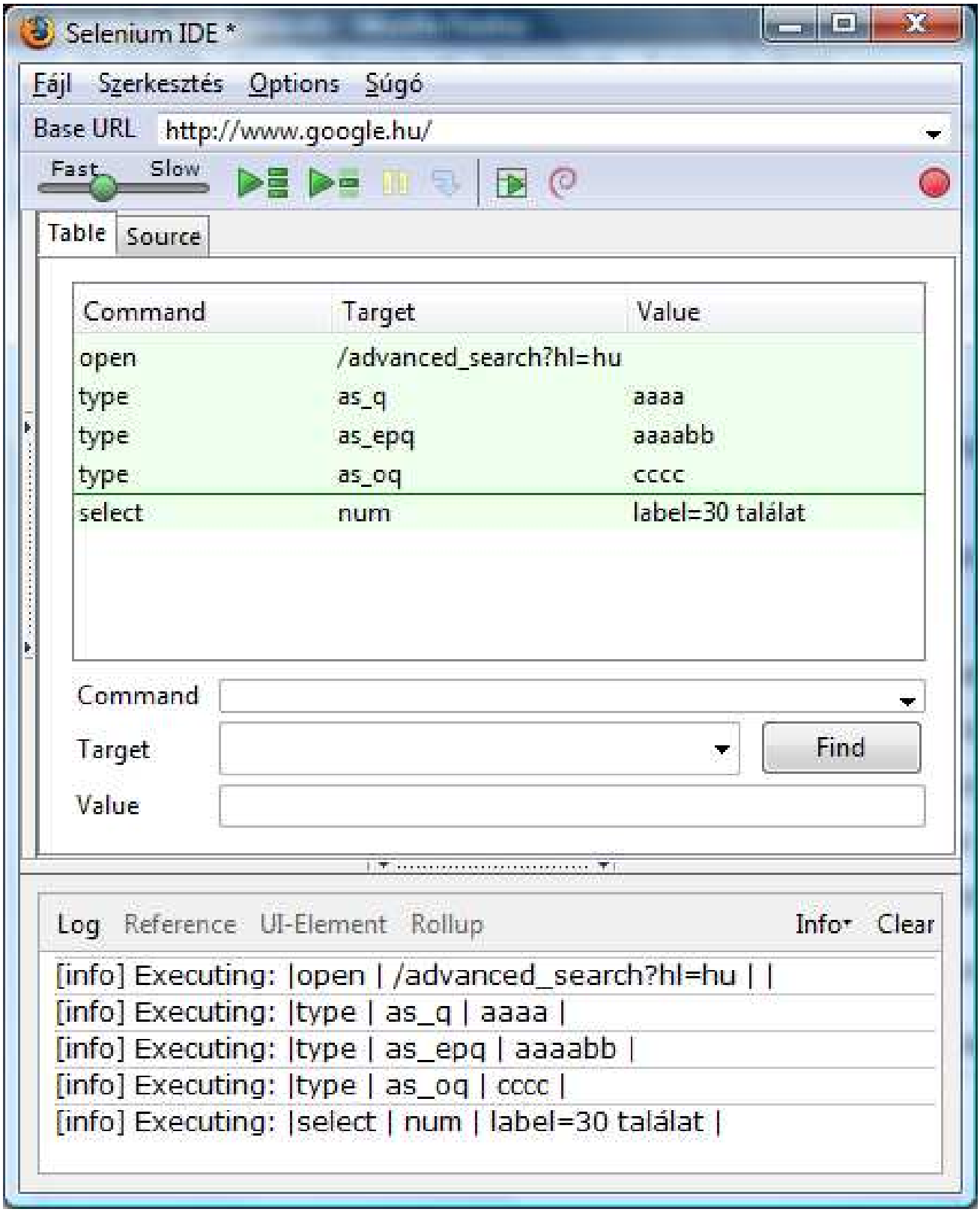
parancsfelismerőjével és gyorsaságával a Selenium IDE egy ideális környezetet teremt Seleniumos tesztek létrehozására és futtatására függetlenül attól, hogy milyen nyelvet választottunk.

4.4.1 Tulajdonságai

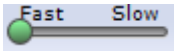
- Tesztek könnyű rögzítése és lejátszása
- Intelligens mezőfelismerés, használva az ID-t, nevet vagy esetlegesen az XPath-t, ha arra lenne szükség
- Automata kódkiegészítés az összes Seleniumos parancsra
- Lépesenkénti tesztek használata
- Debuggolás esetlegesen breakpointok segítségével
- Lehet menteni a teszteket HTML, Java és egyéb már említett nyelveken.
- Külön opció van arra, hogy felismerje és rögzítse a tesztelni kívánt oldal címét.

4.4.2 Röviden a használatról

A kiegészítő letöltése és telepítése Firefox böngészőben a következő webcímről lehetséges: http://seleniumhq.org/download/. A letöltés után az IDE automatikusan feltelepül a Firefox böngészőbe. Elindítása az „Eszközök”menüből lehetséges, indítás után megjelenik a következő ablak.



1. Selenium IDE

A böngészőben elindítjuk a tesztelni kívánt oldalt. Az IDE ablakban a felvétel gombot () bekapcsoltra állítjuk. (Indítás után alapértelmezettként be van kapcsolva) Elkezdjük a felület tesztelését, adatok felvitele, legördülő listákból kiválasztás, gombok megnyomása, chekboxok, rádiógombok és egyéb a felületen előforduló elemek használata. Minden, amit csinálunk a Table ablakba rögzítődik command, target és value formában. A HTML formátum a Source fül kiválasztásával nézhető meg. A rögzített parancsokat kézileg is kiegészíthetjük. Egyszerre több tesztet is rögzíthetünk. A teszt vagy tesztek futtatása a lejátszás gomb lenyomására történik, attól függően, hogy mind () vagy csak az aktuális () tesztet futtattuk. A futás sebességét a 

csúszka megfelelő irányba mozgatásával tudjuk kezelni az indítás előtt. Az eredmény a Log ablakban látható. Esetlegesen ha más formátumban szeretnénk látni a forrást, akkor az Options/Formátum menüben tudjuk a megfelelőt kiválasztani.

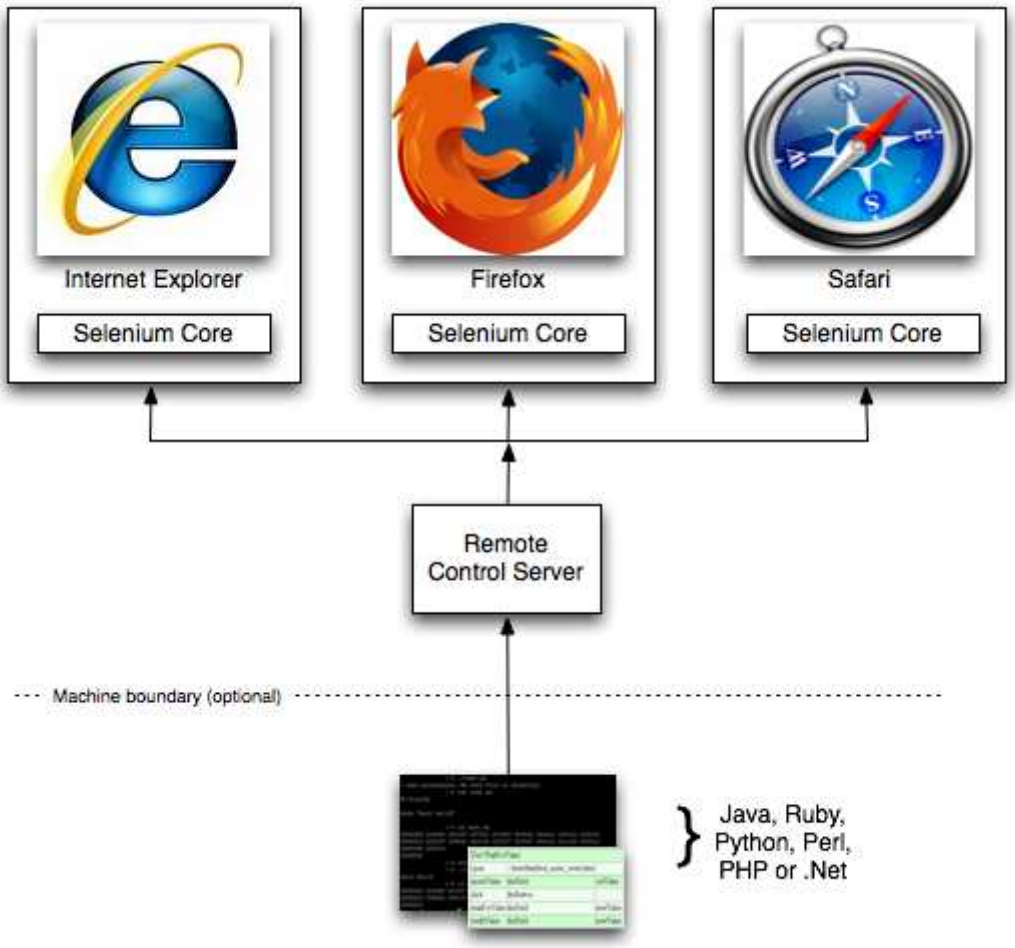
4.5 Selenium Remote Control

A Selenium Remote Control egy tesztelési eszköz, amely lehetővé teszi, hogy bármilyen webes alkalmazásra automatizált UI tesztet készítsünk bármilyen programozási nyelven és alkalmazhatunk bármilyen JavaScript futására alkalmas böngészőben.

A Selenium Remote Control két részből áll:

1. A szerver, ami automatikusan elindítja és leállítja a böngészőt.
2. Felhasználói könyvtár, a használt programozási nyelvvel.

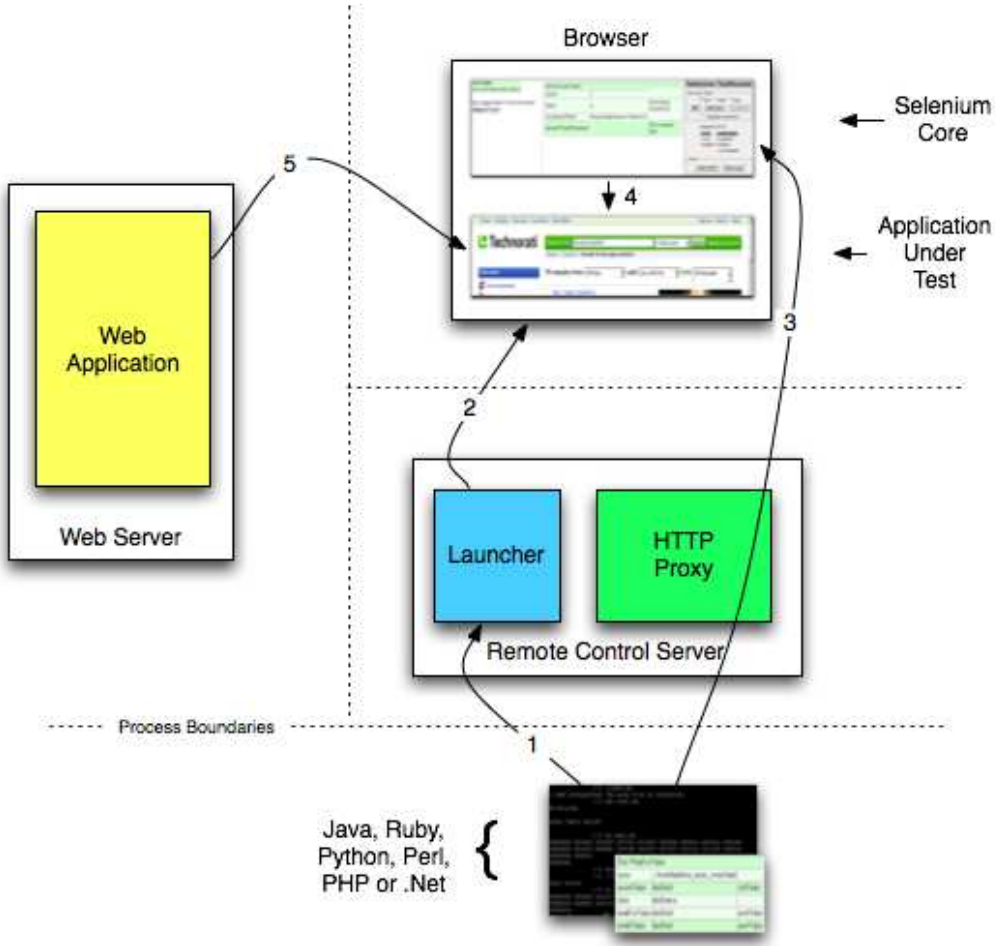
A Remote Control szerver a Selenium Core csomagokból áll, és automatikusan betölti azokat a megadott böngészőbe. Működése a következő ábrán látható.



1. A Selenium RC architektúrális felépítése

4.5.1 Selenium Remote Control működése

A Selenium szerver közvetlenül kommunikál az AJAX-ot használó böngészővel. (XMLHttpRequest). Közvetlenül parancsokat tudunk küldeni a szervernek egyszerű HTTP GET / POST kéréseket használva, ami azt jelenti, hogy bármilyen programozási nyelvet használhatunk, ami a HTTP kéréseket Selenium teszttté konvertálja a böngészőben. A Selenium Server egyfajta kliens konfigurált HTTP proxy, ami a böngésző és a tesztelt weboldal között áll. Ez lehetővé teszi, hogy a Selenium-kompatibilis böngésző futtassa a Javascripteket tetszőleges weboldalon.



1. Selenium RC architektúrális működése

A teszt a kiválasztott nyelven indul a következőképpen:

1. A kliens/driver eléri a Selenium-szervert.
2. A Selenium-szerver elindít a böngészőben egy URL-t, ez fog betöltődni a Selenium Core weboldalán.
3. A Selenium Core megkapja az első utasítást a kliens/driver-től (HTTP Proxy-n keresztül a Selenium RC Serverig)
4. A Selenim-Core első utasítása a tipikusan a tesztelendő felület kezdőoldalának a megnyitása.
5. A webserver megkeresi ezt az oldalt és megjeleníti egy ablakban.

4.6 Selenium Core

A Selenium Core webes alkalmazások tesztelésére használatos eszköz. A tesztek futtatása közvetlenül a böngészőben történik, mintha egy felhasználó csinálná a lépéseket. Az egyetlen olyan eszköz, ami mondhatni platformfüggetlen, mivel használható Internet Explorer, Firefox böngészőkben, Windows, Linux és Macintosh operációs rendszer alatt. A Selenium Core egy egyedi mechanizmust használ, ami lehetővé teszi, hogy sok platformon futhat. Teljes mértékben JavaScript/DHTML alapú nyílt forráskódú szoftver, ami ingyenesen letölthető és használható. Futtatásához telepíteni kell egy webszervert a gépre, ami bármi lehet (én weblogic alatt használom).

4.6.1 Selenium Core működése

A Selenium a tesztbe beágyazott JavaScriptet és Iframe-ket használ a böngésző automata motorjával. Ez a technika bármely JavaScript alapú böngésző alatt működik. Mivel a böngészők JavaScript kezelése különböző, ezért a népszerűbb böngészőkben nagyobb a támogatás.

4.7 Selenium Referencia

A Seleniumos parancs az, ami megmondja, hogy a mit tegyen a Selenium. A parancsoknak három fajtája létezik:

1. Actions: Azok a parancsok tartoznak ide, amelyek az alkalmazás működését manipulálják. Ilyenek például a „klikkelj ide”, „válaszd ki ezt”, „írd be ezt a szöveget ide”. Sok parancs úgynevezett „andWait” utótaggal rendelkezik, aminek hatására a megfelelő parancs után vár a megadott időt, amíg az oldal újra betöltődik, a server oldali kérés visszaérvezik stb.

2. Accessors: Megvizsgálja az alkalmazás egy állását és tárolja az eredményt egy változóban. Automatikusan generált vizsgálatot használ hozzá.

3. Assertions: Hasonló az Accessorhoz, de azt ellenőrzi, hogy az alkalmazás egy állása megfelelő-e és, hogy mi várható. A Seleniumos Asserteknek három fajtája létezik:

    a. assert: ha az assert hamissal tér vissza, akkor a teszt megáll

b. verify: ha a verify hamissal tér vissza, akkor a teszt folytatódik és logolja a hibát.

c. waitFor: addig vár, amíg bizonyos feltételek nem teljesülnek. Ezek hasznosak Ajaxos alkalmazások tesztelésekor. Folytatja a futást, ha a feltétel teljesül vagy megáll, ha nem teljesül.

A Seleniumos parancs felépítése a következő:

1. Command: A parancs neve
2. Target: A parancs első paramétere (kötelező megadni)
3. Value: A parancs értéke, parancstól függően opcionális

4.7.1 Element Locator

Element locatornak hívják a HTML-nek azon részét, amire a parancs vonatkozik. Ez gyakorlatilag a parancs target része. A locator formátuma a következő:

```
locatortype = argument
```

Íme néhány mód a locatorok kiválasztására:

- identifer = *id*: Válasszuk ki azt az elemet, amelynek az @id attribútuma *id*. Ha nem találja, akkor az első olyat, amelynek a @name attribútuma *id*.
- id = *id*: Válasszuk ki azt az elemet, amelynek az @id attribútuma *id*.
- name = *name*: Válasszuk ki azt az elemet, amelynek az @name attribútuma:
  - username
  - name=username.

  Ez kiegészülhet még egy paraméterrel szóközzel elválasztva, ami a value.

  pl: name = *flavour* value = *chocolate*
- dom = *javascriptExpression*: Bármilyen elem megkeresésére, hivatkozására használható JavaScript kifejezés.

    - `dom=document.forms['myForm'].myDropdown`
    - `dom=document.images[56]`
    - `dom=function foo() { return document.links[1]; }; foo();`
- xpath = *xpathExpression*: Válasszuk ki az xpath kifejezés által megadott elemet.
    - `xpath=//img[@alt='The image alt text']`
    - `xpath=//table[@id='table1']//tr[4]/td[2]`
    - `xpath=//a[contains(@href,'#id1')]`
    - `xpath=//a[contains(@href,'#id1')]/@class`
    - `xpath=(//table[@class='stylee'])//th[text()='theHeader Text']/../td`
    - `xpath=//input[@name='name2' and @value='yes']`
    - `xpath=//*[text()="right"]`
- link = *textPattern*: Válasszuk az adott szöveget tartalmazó linket.
    - `link = link szövege`
- css = *cssSelectorSyntax*: Válasszuk az adott css selectort használó elemet.
    - `css=a[href="#id3"]`
    - `css=span#firstChild + span`

4.7.2 Element Filter

Az Element Filtereket használjuk a locatorral megadott elemek szűrésére. Támogatott szűrőelemek:

- value = *valuePattern*: Válasszuk ki a megadott értékű elemet
- index = *index*: Válasszuk ki a megadott indexű elemet.

4.7.3 Mintaillesztések

A következő mintaillesztési konvenciók használatosak Selenium parancs value értékére:

- glob: *pattern*: Válasszuk a mintának megfelelő karaktersorozatot. Két jokerkaraktert használhatunk, „*”, ami bármilyen karaktersorozatot és a „?”, ami bármilyen egyéni karaktert jelent.
- regexp: *regexp*: Válasszuk ki a reguláris kifejezésnek megfelelő karaktersorozatot.
- regexpi: *regexpi*: Válasszuk ki az eszközfüggetlen reguláris kifejezésnek megfelelő karaktersorozatot.
- exact: *string*: Válasszuk ki azt a karaktersorozatot, ami pontosan ugyanaz, mint a megadott minta.

4.7.4 Seleniumos Action parancsok

- `addLocationStrategy (strategyName, functionDefinition)`

  Definiál egy új funkciót a Seleniumnak, ami meghatározza az elem helyét az oldalon. Argumentumai:

  - `strategyName`: Definiálja a stratégia nevét, csak betű karakterek használhatóak, whitespace-ek nélkül.
  - `functionDefinition`: A string definiálja a JavaScriptet.

- `addScript (scriptContent, scriptTagId)`

  Betölti a script tartalmát az új megadott scriptbe. Argumentumai:

  - scriptContent: A Javascript tartalmazza a hozzáadott scriptet.
  - scriptTagId: Az új script tag Id-ja. Megadása opcionális.

- `addSelection (locator, optionLocator)`

  Argumentumai:

  - locator: Element locator, ami egy multi-select box azonosítója
  - optionlocator: a hely meghatározására van lehetőség.

- `allowNativeXpath (allow)`

  Megadja, hogy a Selenium a böngésző által alapértelmezett Xpath-t vagy a JavaScriptes Xpath-t használja. Argumentumai:

  - allow: logikai érték, true esetén a böngésző által alapértelmezett Xpath-t, false esetén a JavaScriptes Xpath-t használja.

- `altKeyDown ()`

  Lenyomja az Alt gombot, és lenyomva tartja a doAltUp ( ) parancsig, vagy egy új oldal betöltéséig.

- `altKeyUp ()`

  Engedje el az ALT billentyűt.

- `answerOnNextPrompt (answer)`

  Argumentumok:

  - answer

- `assignId (locator, identifier)`

  Ideiglenesen egy azonosítót ad a megadott elemnek, így ha később hivatkoznánk rá, nem kell XPath-t használni, ami lassabb és bonyolultabb. Ez a név eltűnik az oldal újratöltésekor. Argumentumai:

  - locator: Element locator
  - identifer: Karaktersorozatot kell használni azonosítónak.

- `break ()`

  Megállítja a teszt futását, amíg meg nem nyomjuk a tovább gombot. Hibakeresésnél hasznos, viszont kézileg kell beavatkozni, hogy tovább fusson a teszt.

- `captureEntirePageScreenshot (filename, kwargs)`

Screenshotot csinál az aktuális ablakról és elmenti egy png kiterjesztésű fájlba. Argumentumai:

- filename: meg kell adni a file nevét teljes elérési úttal. Könyvtárakat nem tud létrehozni, ha a megadott útvonalban nem létezik valamelyik könyvtár, akkor hibát kapunk.
- kwargs: módosíthatjuk vele a screenshot tulajdonságait. pl.: „background = # CCFFDD”

- `check (locator)`

  Kiválasztja a választó gombokat (checkbox / rádiógomb). Bepipálja a checkboxot, kiválasztja a rádiógombot. Argumentuma:

  - locator: Element locator

- `chooseCancelOnNextConfirmation ()`

  Alapértelmezetten Selenium window.confirm () függvény igazzal tér vissza mintha a felhasználó az OK gombra kattintott volna. A parancs hatására a következő megerősítő üzenet visszatérési értéke hamis lesz, ezáltal a Mégse gomb lenyomását szimulálja. A Selenium automatikusan visszatér az igaz visszatérési értékhez.

- `chooseOkOnNextConfirmation ()`

  Előző parancs visszavonása.

- `click (locator)`

  Rákattint a megadott elemre. pl. link, gomb, checkbox, rádiógomb. Ha a kattintás után az oldal újratöltődik, akkor meg kell hívni a waitForPageToLoad utasítást. Argumentuma:

  - locator: Element locator

- `clickAt (locator, coordString)`

Rákattint a megadott elemre a megadott koordinátán. pl. link, gomb, checkbox, rádiógomb. Ha a kattintás után az oldal újratöltődik, akkor meg kell hívni a waitForPageToLoad utasítást. Argumentumai:

  - locator: Element locator
  - coordString: meghatározza az elem x,y pozícióját (pl.: 10,20)

- `close ()`

  A címsorban lévő bezár gombra kattintást szimulálja, és bezárja az ablakot.

- `contextMenu (locator)`

  Az adott elem helyi menüjének a megnyitását szimulálja, mintha egy „jobb-klikket” nyomnánk az egérrel az elemen. Argumentuma:

  - locator: Element locator

- `contextMenuAt (locator, coordString)`

  Az adott koordinátán lévő elem helyi menüjének a megnyitását szimulálja, mintha egy „jobb-klikket” nyomnánk az egérrel az elemen. Argumentuma:

  - locator: Element locator
  - coordString: meghatározza az elem x,y pozícióját (pl.: 10,20)

- `controlKeyDown ()`

  Lenyomja az Control gombot, és lenyomva tartja a doControlUp ( ) parancsig, vagy egy új oldal betöltéséig.

- `controlKeyUp ()`

  Engedje el a Control gombot.

- `createCookie (nameValuePair, optionsString)`

Létrehoz egy új sütit, az elérési útja megegyezik a tesztelés alatt álló oldaléval. Argumentumai:

- nameValuePair: A süti neve és értéke pl.: „név=érték”
- optionsString: sütik tulajdonságai. Megadható tulajdonságok a következők: „path”, „max-age” és „domain”. Formátuma pl.: „path=útvonal”, „max-age=60”, „domain=.foo.com”.

- `deleteAllVisibleCookies ()`

  Az összes sütit törli az oldalon.

- `deleteCookie (name, optionsString)`

  Törli a meghatározott útvonalon az adott nevű sütit. Argumentumai:

  - name: A süti neve
  - optionsString: sütik tulajdonságai. Megadható tulajdonságok a következők: „path”, „domain” és „recurse”. Formátuma pl.: „path=útvonal”, „domain=.foo.com”, „recurse=true”.

- `doubleClick (locator)`

  Kattints a megadott elemre duplán. pl. link, gomb, checkbox, rádiógomb. Ha a kattintás után az oldal újratöltődik, akkor meg kell hívni a waitForPageToLoad utasítást. Argumentuma:

  - locator: Element locator

- `doubleClickAt (locator, coordString)`

  Kattints a megadott elemre duplán a megadott koordinátán. pl. link, gomb, checkbox, rádiógomb. Ha a kattintás után az oldal újratöltődik, akkor meg kell hívni a waitForPageToLoad utasítást. Argumentumai:

  - locator: Element locator

- coordString: meghatározza az elem x,y pozícióját (pl.: 10,20)

- `dragAndDrop (locator, movementsString)`

  Megfog egy elemet és egy bizonyos távolságra helyezi. Argumentumai:

  - locator: Element locator
  - movementsString: a jelenlegi helyéről a megadott koordinátára helyezi az elemet.

- `echo (message)`

  Adott üzenetet ír a Selenium ablak harmadik táblázatára. Hasznos hibakereséskor. Argumentuma:

  - message: a kiírt üzenet.

- `fireEvent (locator, eventName)`

  Explicit szimulál egy adott eseményt. Argumentumai:

  - locator: Element locator
  - eventName: a lehetséges események: „focus” vagy „blur”.

- `focus (locator)`

  Vigye a fókuszt a megadott elemhez, például ha az elem egy beviteli mező, akkor a kurzor az adott területre kerül. Argumentuma:

  - locator: Element locator

- `goBack ()`

  Szimulálja, hogy a felhasználó rákattint a "vissza" gombra a böngészőben.

- `highlight (locator)`

  Rövid időre megváltoztatja az elem háttérszínét sárgára. Hasznos hibakereséskor. Argumentuma:

- locator: Element locator

- `keyDown (locator, keySequence)`

  Szimulálja, hogy a felhasználó lenyomja az adott gombot. Argumentumai:

  - locator: Element locator
  - keysequence: vagy egy karaktersorozat, ami egy ASCII kód, vagy egy karakter. pl.: „\119”, „w”

- `keyPress (locator, keySequence)`

  Szimulálja, hogy a felhasználó elengedi az adott gombot. Argumentumai:

  - locator: Element locator
  - keysequence: vagy egy karaktersorozat, ami egy ASCII kód, vagy egy karakter. pl.: „\119”, „w”

- `metaKeyDown ()`

  Lenyomja az meta gombot, és lenyomva tartja a doMetaUp ( ) parancsig, vagy egy új oldal betöltéséig.

- `metaKeyUp ()`

  Engedje el a meta gombot.

- `mouseDown (locator)`

  Szimulálja, hogy a felhasználó megnyomja a bal egérgombot a megadott elemen. Argumentuma:

  - locator: Element locator

- `mouseDownAt (locator, coordString)`

Szimulálja, hogy a felhasználó megnyomja a bal egérgombot a megadott helyen. Argumentumai:

- locator: Element locator
- coordString: meghatározza az elem x,y pozícióját (pl.: 10,20)

- `mouseDownRight (locator)`

  Szimulálja, hogy a felhasználó megnyomja a jobb egérgombot a megadott elemen. Argumentuma:

  - locator: Element locator

- `mouseDownRightAt (locator, coordString)`

  Szimulálja, hogy a felhasználó megnyomja a jobb egérgombot a megadott helyen. Argumentumai:

  - locator: Element locator
  - coordString: meghatározza az elem x,y pozícióját (pl.: 10,20)

- `mouseMove (locator)`

  Szimulálja, hogy a felhasználó megnyomja az egérgombot megadott elemen. Argumentuma:

  - locator: Element locator

- `mouseMoveAt (locator, coordString)`

  Szimulálja, hogy a felhasználó megnyomja az egérgombot a megadott helyen.

  - locator: Element locator
  - coordString: meghatározza az elem x,y pozícióját (pl.: 10,20)

- `mouseOut (locator)`

  Szimulálja, hogy a felhasználó mozgatja az egérmutatót. Argumentuma:

    - locator: Element locator.

- `open (url)`

  Megnyitja a paraméterként megkapott URL címet a tesztkörnyezetben. Argumentuma:

    - url: A megnyitni kívánt URL cím.

- `openWindow (url, windowID)`

  Megnyit egy felugró ablakot (ha még nem nyitott). Megnyitása után az ablakot ki kell választani, erre a selectWindow parancs alkalmas. Argumentumai:

    - url: A megnyitni kívánt URL cím, amely lehet üres is.
    - windowID: A kiválasztott ablak JavaScript ablak ID-ja

- `pause (waitTime)`

  A teszt a megadott ideig várakozik. Argumentuma:

    - waitTime: a várakozási idő ezredmásodpercben.

- `refresh ()`

  Szimulálja, hogy a felhasználó rákattint a "Frissítés" gombra a böngészőben.

- `removeAllSelections (locator)`

  Az összes kiválasztott listán megszünteti a kiválasztást. Argumentuma:

    - locator: Element locator

- `removeSelection (locator, optionLocator)`

  Eltávolítja a kiválasztást a megadott legördülő listáról a megadott option locator segítségével. Argumentumai:

    - locator: Element locator

- optionLocator: helymeghatározó (label az alapértelmezett) Bővebben a select utasításnál.

- `runScript (script)`

  Létrehoz egy új, "script" tag-et a tesztablak body részében, és hozzáfűz speciális szöveget a parancsrészhez. Parancsfájlok futtatására a "getEval" parancsot kell használni. Argumentuma:

  - script - a futtatandó JavaScript kódrészlet

- `select (selectLocator, optionLocator)`

  A legördülő listából kiválaszt egy elemet. Argumentumai:

  - selecLocator: Element locator
  - optionLocator: A optionLocator megadására alkalmazhatóak a mintaillesztésnél megadott opciók a következőképpen:
    - label=labelpattern – Ez az alapértelmezett
    - label=regexp:^[Oo]ther
    - value = valuePattern
    - id = id
    - index = index

- `selectFrame (locator)`

  Kiválasztja a keretet az aktuális ablakban. A keret azonosítására használható a következő kifejezés is: dom=frames["main"].frames["subframe"] Argumentuma:

  - locator: Element locator

- `selectWindow (windowID)`

Kijelöl egy felugró ablakot, ha egy felugró ablakban ki van jelölve, minden parancs arra az ablakra vonatkozil. Ha a főablakot szeretnék újra alapértelmezetté tenni, akkor használjuk ezt a parancsot még egyszer null paraméterrel. Argumentuma:

  - windowID: A kiválasztott ablak JavaScript ablak ID-ja

- `setBrowserLogLevel (logLevel)`

  Beállítja a böngésző logolási szintjét. Argumentuma:

  - logLevel: Lehetséges értékei: „debug”, „info”, „warn”, „error” vagy „off”.

- `setCursorPosition (locator, position)`

  A megadott pozícióba viszi a kurzort. Ha a megadott elem nem szöveg, vagy neviteli mező, akkor hibát fog jelezni ez a parancs és a teszt leáll. Argumentuma:

  - locator: Element locator

  - position: Numerikus számmal kell megadni az értékét. 0-nál nem mozdul. Lehet negatív számot is használni, ha pl. a kurzor a szöveg végén áll.

- `setMouseSpeed (pixels)`

  Beállítja a pixelek számát a „mousemove” parancsokhoz. Az alapértelmezett érték 10 pixel. Argumentuma:

  - pixels: pixelek száma

- `setSpeed (value)`

  Beállítja a Seleniumos parancsok futásának a sebességét. Alapértelmezettként ez az érték 0, tehát nincs semmiféle késleltetés a parancsok között. Argumentuma:

  - value: értéke milliszekundumban

- `setTimeout (timeout)`

Megadja azt az időt ameddig a parancsok után vár a Selenium, hogy jó-e a parancs. Az open és waitFor parancsokban ez benne van, alapértelmezett értéke 30 másoperc. Ezt az értéket változtatja meg. Argumentuma:

- timeout: ezredmásodpercben megadott érték, amely után hibával tér vissza

- `shiftKeyDown ()`

  Lenyomja az shift gombot, és lenyomva tartja a doshiftUp ( ) parancsig, vagy egy új oldal betöltéséig.

- `shiftKeyUp ()`

  Engedje fel a Shift billentyűt.

- `type (locator, value)`

  Begépeli a paraméterként megadott értéket a beviteli mezőbe.

  Argumentuma:

  - locator: Element locator
  - value: Bármilyen karakter.

- `typeKeys (locator, value)`

  Úgy írja be az adott szöveget, mintha betűnként gépelnénk, akkor hasznos, ha a karakterekhez valamilyen esemény van rendelve pl. automatikus kitöltés. Argumentuma:

  - locator: Element locator
  - value: Bármilyen karakter.

- `uncheck (locator)`

  Törölje a kijelölést a bejelölt rádiógombról vagy checkboxról. Argumentuma:

  - locator: Element locator

- `waitForCondition (script, timeout)`

  Addig futtat egy JavaScript részt, amíg igazzal tér vissza. Argumantumai:

  - script: A futtatandó JavaScript.
  - timeout: ezredmásodpercben megadott érték, amely után hibával tér vissza

- `waitForFrameToLoad (frameAddress, timeout)`

  A megadott ideig vár, amíg a frame betöltődik. Argumentumok:

  - frameAddress - a szerver oldali FrameAddress
  - timeout: ezredmásodpercben megadott érték, amely után hibával tér vissza

- `waitForPageToLoad (timeout)`

  A megadott ideig vár, amíg az oldal betöltődik. Argumentuma:

  - timeout: ezredmásodpercben megadott érték, amely után hibával tér vissza

- `waitForPopUp (windowID, timeout)`

  A megadott ideig vár, amíg a felugró ablak megjelenik és betöltődik. Argumentumai:

  - windowID: A kiválasztott ablak JavaScript ablak ID-ja
  - timeout: ezredmásodpercben megadott érték, amely után hibával tér vissza

- `windowFocus ()`

  Fókuszt ad kiválasztott ablaknak.

- `windowMaximize ()`

  Átméretezi a kiválasztott ablakot, akár teljes képernyőssé is teheti.

A teljesség igény nélkül ezek voltak a Seleniumos Action parancsok. Ki kell emelnem, hogy csak azokat említettem meg, amelyek Java alatt is előfordulnak, mivel én azzal dolgozom és azokat, használom. A következőekben még felsorolás szintjén leírom az

accessoros parancsokat. Az ok, amiért csak felsorolás szinten írok ezekről, az, hogy valamiért - számomra érthetetlen módon - nem szerepelnek a Java api-ban, így nem használtam még egyiket sem.

4.7.5 Seleniumos Accessor parancsok

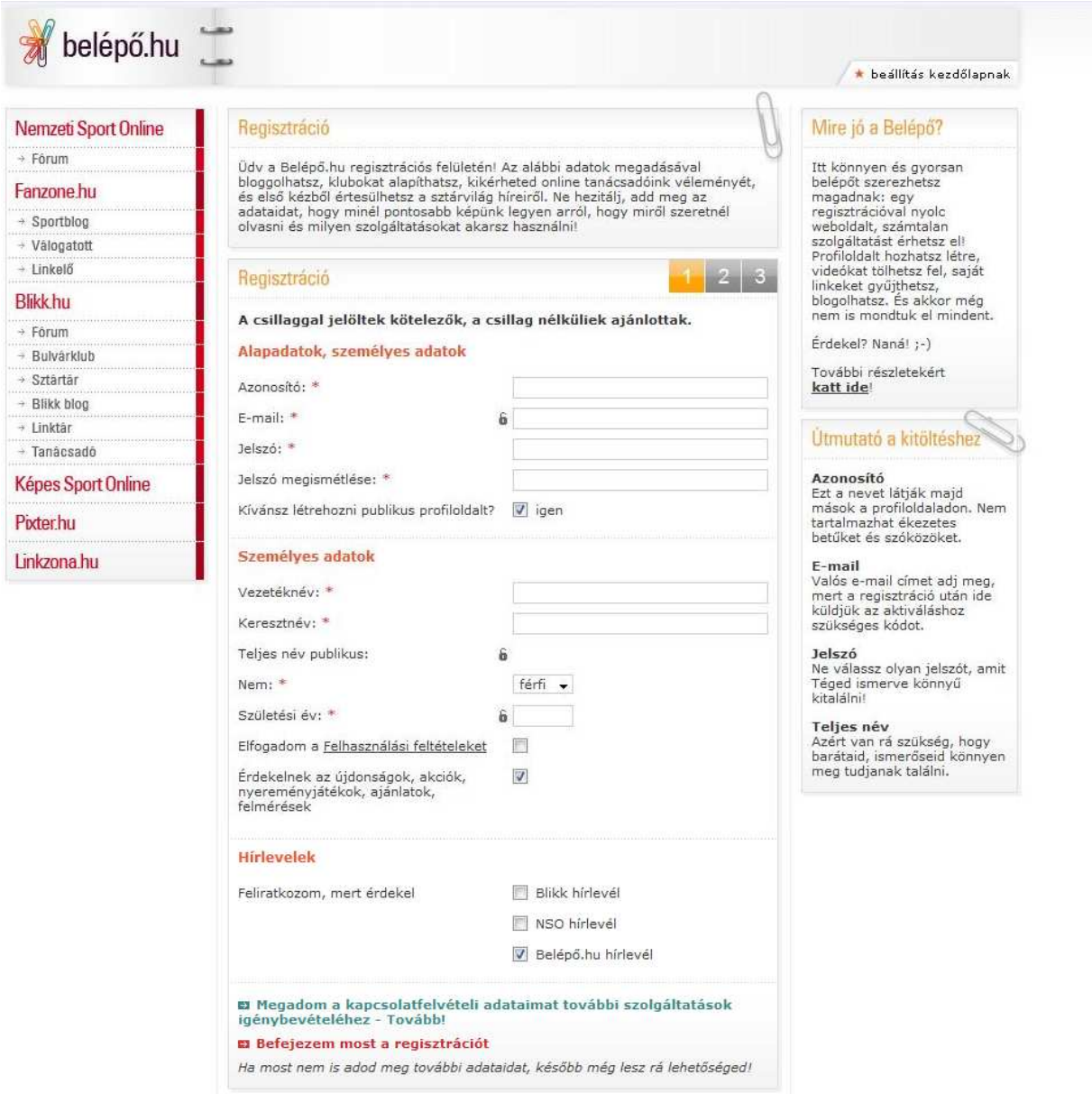
- assertErrorOnNext (message)
- assertFailureOnNext (message)
- assertSelected (selectLocator,optionLocator)
- storeAlert (variableName)
- storeAllButtons (variableName)
- storeAllFields (variableName)
- storeAllLinks (variableName)
- storeAllWindowIds (variableName)
- storeAllWindowNames (variableName)
- storeAllWindowTitles (variableName)
- storeAttribute (attributeLocator, variableName)
- storeAttributeFromAllWindows (attributeName, variableName)
- storeBodyText (variableName)
- storeConfirmation (variableName)
- storeCookie (variableName)
- storeCookieByName (name, variableName)
- storeCursorPosition (locator, variableName)
- storeElementHeight (locator, variableName)

- storeElementIndex (locator, variableName)
- storeElementPositionLeft (locator, variableName)
- storeElementPositionTop (locator, variableName)
- storeElementWidth (locator, variableName)
- storeEval (script, variableName)
- storeExpression (expression, variableName)
- storeHtmlSource (variableName)
- storeLocation (variableName)
- storeMouseSpeed (variableName)
- storePrompt (variableName)
- storeSelectedId (selectLocator, variableName)
- storeSelectedIds (selectLocator, variableName)
- storeSelectedIndex (selectLocator, variableName)
- storeSelectedIndexes (selectLocator, variableName)
- storeSelectedLabel (selectLocator, variableName)
- storeSelectedLabels (selectLocator, variableName)
- storeSelectedValue (selectLocator, variableName)
- storeSelectedValues (selectLocator, variableName)
- storeSelectOptions (selectLocator, variableName)
- storeSpeed (variableName)
- storeTable (tableCellAddress, variableName)

- storeText (locator, variableName)
- storeTitle (variableName)
- storeValue (locator, variableName)
- storeWhetherThisFrameMatchFrameExpression (currentFrameString, target, variableName)
- storeWhetherThisWindowMatchWindowExpression (currentWindowString, target, variableName)
- storeXpathCount (xpath, variableName)
- storeAlertPresent (variableName)
- storeChecked (locator, variableName)
- storeConfirmationPresent (variableName)
- storeCookiePresent (name, variableName)
- storeEditable (locator, variableName)
- storeElementPresent (locator, variableName)
- storeOrdered (locator1, locator2, variableName)
- storePromptPresent (variableName)
- storeSomethingSelected (selectLocator, variableName)
- storeTextPresent (pattern, variableName)
- storeVisible (locator, variableName)

5. Példa a Selenium működésére

A következő rövid kis példával szeretném bemutatni egy Seleniumos tesztfolyamatot a gyakorlatban (ahogy én csinálom). A tesztelendő felület a Nemzeti Sport Online oldalának a regisztrációs része.



Az oldal elérése a http://www.belepo.hu/regisztracio/?registration_brand=nso címen van. A tesztelés lépései a következők:

1. Megnyitjuk Firefox böngészővel a tesztelni kívánt oldalt (ebben az esetben NSO regisztrációs részét)

2. Elindítjuk a Selenium IDE-t. Alapértelmezettként a formátum HTML-re van állítva, érdemes ezen hagyni, amíg rögzítjük a tesztet, hogy ki tudjuk aztán próbálni, hogy megfelelően működik-e.

3. Az összes beviteli mezőbe tetszőleges szöveget írunk, minden lehetséges checkboxot kipróbálunk, hogy az IDE rögzítse a tesztet.

4. Miután rögzítettük a tesztet átváltjuk a kívánt programozási nyelvre, ebben az esetben Java-ra, és kész is van az első teszt, amit „csak” általánosítani kell, valamint teszteseteket kell hozzá csinálni.

5. A teszteseteket érdemes excelben csinálni, mivel létexik Java-hoz egy jxl.jar, amivel fel tudjuk olvasni a kívánt tesztesetet.

A Java kód „általánosítása” a következő:

1. Először létrehozunk egy VO osztályt, amiben a felvehető paramétereket határozzuk meg. Példánk esetében:

```
package selenium;

public class NsoRegistrationVO {

private java.lang.String username;
private java.lang.String is_email_public;
private java.lang.String email;
private java.lang.String passwd;
private java.lang.String passagain;
private java.lang.String request_profile;
private java.lang.String lastname;
private java.lang.String firstname;
private java.lang.String is_realname_public;
private java.lang.String gender;
private java.lang.String is_birth_year_public;
private java.lang.String birth_year;
private java.lang.String accept;
private java.lang.String is_dm_allowed;
private java.lang.String nl_blikk;
private java.lang.String nl_nso;
private java.lang.String nl_belepo;

public java.lang.String getAccept() {
            return accept;
      }
public void setAccept(java.lang.String accept) {
            this.accept = accept;
      }
public java.lang.String getBirth_year() {
```

```
            return birth_year;
      }
public void setBirth_year(java.lang.String birth_year) {
            this.birth_year = birth_year;
      }
public java.lang.String getEmail() {
            return email;
      }
public void setEmail(java.lang.String email) {
            this.email = email;
      }
public java.lang.String getFirstname() {
            return firstname;
      }
public void setFirstname(java.lang.String firstname) {
            this.firstname = firstname;
      }
public java.lang.String getGender() {
            return gender;
      }
public void setGender(java.lang.String gender) {
            this.gender = gender;
      }
public java.lang.String getIs_birth_year_public() {
            return is_birth_year_public;
      }
public void setIs_birth_year_public(java.lang.String
is_birth_year_public) {
            this.is_birth_year_public = is_birth_year_public;
      }
public java.lang.String getIs_dm_allowed() {
            return is_dm_allowed;
      }
public void setIs_dm_allowed(java.lang.String is_dm_allowed) {
            this.is_dm_allowed = is_dm_allowed;
      }
public java.lang.String getIs_email_public() {
            return is_email_public;
      }
public void setIs_email_public(java.lang.String is_email_public) {
            this.is_email_public = is_email_public;
      }
public java.lang.String getIs_realname_public() {
            return is_realname_public;
      }
public void setIs_realname_public(java.lang.String is_realname_public) {
            this.is_realname_public = is_realname_public;
      }
public java.lang.String getLastname() {
            return lastname;
      }
public void setLastname(java.lang.String lastname) {
            this.lastname = lastname;
      }
public java.lang.String getNl_belepo() {
            return nl_belepo;
      }
public void setNl_belepo(java.lang.String nl_belepo) {
```

```
            this.nl_belepo = nl_belepo;
      }
public java.lang.String getNl_blikk() {
            return nl_blikk;
      }
public void setNl_blikk(java.lang.String nl_blikk) {
            this.nl_blikk = nl_blikk;
      }
public java.lang.String getNl_nso() {
            return nl_nso;
      }
public void setNl_nso(java.lang.String nl_nso) {
            this.nl_nso = nl_nso;
      }
public java.lang.String getPassagain() {
            return passagain;
      }
public void setPassagain(java.lang.String passagain) {
            this.passagain = passagain;
      }
public java.lang.String getPasswd() {
            return passwd;
      }
public void setPasswd(java.lang.String passwd) {
            this.passwd = passwd;
      }
public java.lang.String getRequest_profile() {
            return request_profile;
      }
public void setRequest_profile(java.lang.String request_profile) {
            this.request_profile = request_profile;
      }
public java.lang.String getUsername() {
            return username;
      }
public void setUsername(java.lang.String username) {
            this.username = username;
      }

}
```

2. Ezek után létrehozunk egy Util osztályt, amelyben az IDE-vel rögzített tesztnek a paramétereit általánosítjuk.

```
package selenium;

import com.thoughtworks.selenium.DefaultSelenium;

public class NsoRegistrationUtil {

      public static final String EMPTY_STRING = "";


      public static boolean isEmptyString(String string)
      {
```

```
        return ((string == null || string.trim().equals(EMPTY_STRING))
        ? true : false);
}

public static boolean isNotEmptyString(String string)
{
        return !isEmptyString(string);
}
public static void pageOne(DefaultSelenium selenium,
NsoRegistrationVO nsoRegistrationVO)
{
        if(isNotEmptyString(nsoRegistrationVO.getUsername()))
                selenium.type("username",
                nsoRegistrationVO.getUsername());

        if(isNotEmptyString(nsoRegistrationVO.getIs_email_public()) &&
                nsoRegistrationVO.getIs_email_public().equals("Nem"))
                selenium.click("is_email_public");

        if(isNotEmptyString(nsoRegistrationVO.getEmail()))
                selenium.type("email", nsoRegistrationVO.getEmail());

        if(isNotEmptyString(nsoRegistrationVO.getPasswd()))
                selenium.type("passwd", nsoRegistrationVO.getPasswd());

        if(isNotEmptyString(nsoRegistrationVO.getPassagain()))
                selenium.type("passagain",
                nsoRegistrationVO.getPassagain());

        if(isNotEmptyString(nsoRegistrationVO.getRequest_profile()) &&
                nsoRegistrationVO.getRequest_profile().equals("Nem"))
                selenium.click("request_profile");


        if(isNotEmptyString(nsoRegistrationVO.getLastname()))
                selenium.type("lastname",
                nsoRegistrationVO.getLastname());

        if(isNotEmptyString(nsoRegistrationVO.getFirstname()))
                selenium.type("firstname",
                nsoRegistrationVO.getFirstname());

        if(isNotEmptyString(nsoRegistrationVO.getIs_realname_public())
        && nsoRegistrationVO.getIs_realname_public().equals("Nem"))
                selenium.click("is_realname_public");

        if(isNotEmptyString(nsoRegistrationVO.getGender()))
                selenium.select("gender", nsoRegistrationVO.getGender());

        if(isNotEmptyString(nsoRegistrationVO.getIs_birth_year_public()
) && nsoRegistrationVO.getIs_birth_year_public().equals("Nem"))
                selenium.click("is_birth_year_public");

        if(isNotEmptyString(nsoRegistrationVO.getBirth_year()))
                selenium.type("birth_year",
                nsoRegistrationVO.getBirth_year());
```

```
            if(isNotEmptyString(nsoRegistrationVO.getAccept()) &&
            nsoRegistrationVO.getAccept().equals("Igen"))
                  selenium.click("accept");

            if(isNotEmptyString(nsoRegistrationVO.getIs_dm_allowed()) &&
            nsoRegistrationVO.getIs_dm_allowed().equals("Nem"))
                  selenium.click("is_dm_allowed");

            if(isNotEmptyString(nsoRegistrationVO.getNl_blikk()) &&
            nsoRegistrationVO.getNl_blikk().equals("Igen"))
                  selenium.click("nl_blikk");

            if(isNotEmptyString(nsoRegistrationVO.getNl_nso()) &&
            nsoRegistrationVO.getNl_nso().equals("Igen"))
                  selenium.click("nl_nso");

            if(isNotEmptyString(nsoRegistrationVO.getNl_belepo()) &&
            nsoRegistrationVO.getNl_nso().equals("Nem"))
                  selenium.click("nl_belepo");

      }
}
```

3. Majd létrehozunk egy osztályt a teszt futtatására.

```
package selenium;

import java.io.File;
import java.io.IOException;
import java.util.LinkedHashMap;
import java.util.Map;

import junit.framework.TestCase;
import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.SeleniumException;

public class NsoRegistrationTest extends TestCase{

      private DefaultSelenium selenium;
      private Map<String, Long> columns =
            new LinkedHashMap<String, Long>();


      public void setUp() throws Exception
      {
            super.setUp();
        selenium =
createSeleniumClient("http://www.belepo.hu/regisztracio/?registration_brand
=nso");
        selenium.start();
    }
```

```
protected DefaultSelenium createSeleniumClient(String url)
throws Exception
{
  return new DefaultSelenium("localhost", 4444, "*iexplore", url);
}


public void tearDown()  throws Exception
{
           super.tearDown();
}

private int getColumnId(String columnName)
{
            return columns.get(columnName).intValue();

}

public void test() throws BiffException, IOException
{

      File inputWorkbook = new File("d:/nsoteszt.xls");

      Workbook w;

try {

      w = Workbook.getWorkbook(inputWorkbook);

      Sheet sheet = w.getSheet("nsoteszt");

      for(int j=1; j<sheet.getColumns(); j++)
      {
            columns.put(sheet.getCell(j, 0).getContents(), new
            Long(j));
      }

      NsoRegistrationVO nsoRegistrationVO = new NsoRegistrationVO();

      for (int i=1; i<sheet.getRows(); i++)
      {

selenium.open("http://www.belepo.hu/regisztracio/?registration_brand=
nso");
      selenium.waitForPageToLoad("30000");

nsoRegistrationVO.setUsername(sheet.getCell(getColumnId("username"),
i).getContents());

nsoRegistrationVO.setIs_email_public(sheet.getCell(getColumnId("is_em
ail_public"), i).getContents());

nsoRegistrationVO.setEmail(sheet.getCell(getColumnId("email"),
i).getContents());

nsoRegistrationVO.setPasswd(sheet.getCell(getColumnId("passwd"),
i).getContents());
```

```
        nsoRegistrationVO.setPassagain(sheet.getCell(getColumnId("passagain")
        , i).getContents());

        nsoRegistrationVO.setLastname(sheet.getCell(getColumnId("lastname"),
        i).getContents());

        nsoRegistrationVO.setFirstname(sheet.getCell(getColumnId("firstname")
        , i).getContents());

        nsoRegistrationVO.setIs_realname_public(sheet.getCell(getColumnId("is
        _realname_public"), i).getContents());

        nsoRegistrationVO.setGender(sheet.getCell(getColumnId("gender"),
        i).getContents());

        nsoRegistrationVO.setIs_birth_year_public(sheet.getCell(getColumnId("
        is_birth_year_public"), i).getContents());

        nsoRegistrationVO.setBirth_year(sheet.getCell(getColumnId("birth_year
        "), i).getContents());

        nsoRegistrationVO.setAccept(sheet.getCell(getColumnId("accept"),
        i).getContents());

        nsoRegistrationVO.setIs_dm_allowed(sheet.getCell(getColumnId("is_dm_a
        llowed"), i).getContents());

        nsoRegistrationVO.setNl_blikk(sheet.getCell(getColumnId("nl_blikk"),
        i).getContents());

        nsoRegistrationVO.setNl_nso(sheet.getCell(getColumnId("nl_nso"),
        i).getContents());

        nsoRegistrationVO.setNl_nso(sheet.getCell(getColumnId("nl_belepo"),
        i).getContents());


              NsoRegistrationUtil.pageOne(selenium, nsoRegistrationVO);

              selenium.click("link=Befejezem most a regisztrációt");
              selenium.waitForPageToLoad("30000");

        if (selenium.isTextPresent
        (sheet.getCell(getColumnId("request"),i).getContents()))
                         System.out.println("Jó teszteset: "+i);

        else selenium.captureScreenshot("d:/Hiba/"+i+"");


              }
        } catch (SeleniumException ex) {
                    fail(ex.getMessage());
                    throw ex;
              }
    }

}
```

4. Végezetül létrehozzuk az Excel táblában a teszteseteket. Az táblának az első sora a felvehető paraméterek nevei, az utolsó oszlopot nevezzük el „request”-nek, amibe az elvárt eredményt írjuk. Az elvárt eredmény a link lenyomása után megjelenő szövegrész lesz. A Selenium megvizsgálja, hogy az elvárt szöveg megjelenik-e a felületen. Ha igen, akkor kiírja a konzolra, hogy: „Jó teszteset”, ha nem, akkor készít egy screenshoot-ot és elmenti a megadott útvonalon lévő könyvtárba. Az excel xls formátumba készítsük el, mivel a jxl.jar csak ezt tudja kezelni.

5. Futtatni JUnit teszttel lehet a selenium server elindítása után.

6. Összefoglalás

Témaválasztásom célja az volt, hogy szerettem volna egy kis betekintést nyújtani a tesztelés rejtelmeibe, mivel úgy gondolom, hogy egyre inkább előtérbe kerül ez a szakma is. Külföldi cégek már régebben felismerték, hogy igen is szükség van külön tesztelői csapatra egy projekt fejlesztése során, de Magyarországon ez csak most kezd igazán elterjedni. Kisebb projekteknél még így is előfordul, hogy a tesztelésre szánt időt alulbecsülik, hogy ezzel próbálják a költségeket csökkenteni. Ennek a következménye az lesz, hogy az alkalmazás vagy tesztelés nélkül, vagy csak félig-meddig letesztelve kerül a megrendelőhöz. Ha a hiba kiderül a költség sokkal több lesz, mintha bele lett volna kalkulálva a projekt folyamatába a tesztelés is, még olyan áron is, hogy esetleg később lesz átadva a megrendelőnek. Ezért van szükség a Seleniumra, vagy az ahhoz hasonló rendszerekre, hogy ezek segítségével lerövidítsük a tesztelésre szánt időt. Azt fontos megjegyezni, hogy egy rendszer sem képes helyettesíteni az embert, önmagában egy szoftver kevés arra, hogy minden összefüggést lefedjen, és persze a teszteseteket is meg kell csinálni, hogy a robot jól működjön. Úgy gondolom, hogy a Selenium egyre szélesebb körben kezd elterjedni, ez nem csak annak, köszönhető, hogy ingyenesen használható, hanem, hogy használata egyszerű és könnyen elsajátítható.

7. Köszönetnyílvánítás

Szeretném megköszönni a türelmét és segítségét a dolgozat elkészítésében Dr. Bujdosó Gyöngyi tanárnőnek, valamint a Neuron Kft. dolgozóinak.

Külön szeretném megköszönni a feleségemnek és a családomnak a segítséget, és hogy tanulmányaim során végig mellettem álltak.

8. Irodalomjegyzék

Selenium:

- http://seleniumhq.org/
- http://en.wikipedia.org/wiki/Selenium_(software)

Tesztelés:

- http://www.alvicom.hu/hun/index.php
- http://hu.wikipedia.org/wiki/Szoftvertesztel%C3%A9s
- http://www.dcs.vein.hu/~simon/oktatas/ST/23.pdf