TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

UI testide implementeerimine MS Dynamics CRMi näitel

Bakalaureusetöö

Üliõpilane: Sergei Malõšev
Üliõpilaskood: 073703IABB
Juhendaja: Teodor Luczkowski

Tallinn
2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

_____ _____

(kuupäev) *(allkiri)*

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on kirjeldada kasutajaliidese automaattestide implementeerimist MS Dynamics CRM projekti näitel.

Töö käigus antakse ülevaadet projektist ning testitavast rakendusest, seejärel tutvustatakse millised on sobivad tööriistad ja mustrid kasutajaliidese automaattestimiseks ning kuidas neid kohandada arvestades MS Dynamics CRM'i eripära. Lisaks kirjeldatakse konfiguratsiooni testide automaatseks käivitamiseks pideva integratsiooni serveris.

Töö tulemusena valmib automaattestide terviklahendus, mis peaks tagama kiiret ja sagedast tagasisidet, ülevaadet vigade olemasolust süsteemi ärikriitilistes komponentides ning seeläbi kiirendama arendusprotsessi ja vähendama manuaalset testimist.

Antud töö võib olla kasulik abimaterjal kõikidele teistele kes puutuvad kokku MS Dynamics CRM'i automatiseerimisega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 54 leheküljel, 6 peatükki, 16 joonist, ühte tabelit.

Abstract

The goal of this bachelors thesis is to implement user interface automated tests based on MS Dynamics CRM project.

The work gives overview of project and testable application, introduces used technologies and patterns for user interface automation and how adjust them considering MS Dynamics CRM singularity. Additionally, describes creating configuration to start automated tests within existing continuous integration system.

This work results providing automated tests complete solution which should provide quick feedback about system business-critical components functionality, improve development process and significantly decrease amount of manual testing.

The thesis is in Estonian language and contains 54 pages of text, 6 chapters, 16 figures, and one table.

Lühendite ja mõistete sõnastik

CRM	***Customer Relationship Management*** Kliendihaldus tarkvara
UI	***User Interface*** Rakenduse kasutajaliides
API	***Application Programming Interface*** Reeglistik, mille alusel rakendusprogramm kasutab operatsioonisüsteemi või teise rakendusprogrammi teenuseid.
Git	***Git*** Versioonihaldus tarkvara
Selenium	***Selenium*** Tarkvara testimis raamistik
WebDriver	***WebDriver*** Draiver veebilehitseja juhtimiseks Seleniumi abil
IDE	***Integrated Development Environment*** Integreeritud programmeerimiskeskkond
NuGet	***NuGet*** Rakendus mis võimaldab tarkvarapakettide otsimist installeerimist, uuendamist
XRM	***Extended Relationship Management*** Laiendatud kliendisuhete haldus. Strateegiline ühtse süsteemi arendamine kontseptsioon, mille autor on Doug Laney [1]
URL	***Uniform Resource Locator*** Unikaalne internetiaadress

NUnit	***NUnit***
	Testimise raamistik .Net programmeerimiskeelte jaoks
TeamCity	***TeamCity***
	Koosteprotsessi ja pideva integratsiooni server
Repository	***Repository***
	Hoidla, rakendustarkvara juurde kuuluva info andmebaas
CIP	***Customer Identification Program***
	Firmasisene infosüsteem klientide aadressite hoidmiseks.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Tarkvaraarendus projektid, eriti suuremad ja keerukamad vajavad täiendavaid investeeringuid testimisse. Agiilse tarkvaraarenduse üks põhiprintsiipe on tarnida töötav osa funktsionaalsust kindlate intervallide järel, nii tihti kui võimalik[2]. Sellises lähenemises automaattestimise roll tarkvaraarendusprotsessis muutub järjest tähtsamaks. Väiksemate projektide puhul investeerimine automaattestidesse võib tunduda ebamõistlikult kulukaks kuid suuremate projektide puhul see lisainvesteering lõppkokkuvõttes tasub ennast ära[3]. Automatiseerimine aitab vähendada korduvat, aeganõudvat manuaalset testimist.

1.1 Taust ja probleem

Autor töötab ettevõttes, mille põhitegevus on logistikateenuste pakkumine. Firma peakontor asub Hamburgis ning Tallinnas asub firma üks IT osakondadest. Konkreetse projekti tiimis on kokku 8 liiget, tarkvara testimise eest vastutab Quality Engineer. Tiimis lähtutakse agiilse tarkvaraarenduse printsiipidest.

Klientide ja pakkumiste halduseks kasutatakse MS Dynamics CRM (Microsoft Dynamics Customer Relationship Management) tarkvara. Antud kommertstarkvara on karbitoode, mis sobib paljudele väikeettevõttetele, kes peavad liiga kulukaks tarkvaraarendusse investeerida. Suuremad firmad kasutuavad Microsofti poolt ette antud SDK (Software Development Kit) millega saab süsteemi funktsionaalsust vastavalt firma vajadustele edasi arendada, sealhulgas integreerida teiste infosüsteemidega. Funktsionaalsuse keerukuse kasvades tõuseb ka vajadus tagada tarkvara kõrge kvaliteet.

Tööle asumise ajal projekti alles alustatud ning autorile anti eesmärk implementeerida kasutajaliidese automaattestid. Enne antud projekti kallal tööle asumist autor puutus kokku enamasti manuaalse testimisega, testide automatiseerimisega vaid vähesel määral. Eelnevalt autor uuris millised on kokkupuuted teistel firmadel MS Dynamicsi CRMi automaattestidega kohtudes nii ühe teise MS Dynamicsi CRM'i arenduse peal spetsialiseeruva firma esindajaga kui ka internetis infot otsides. Paraku asjakohast informatsiooni internetis on üsna vähe. Tuginedes Dynamics CRM'i arendamisega kokku puutunud inimeste kogemustele põhjuseks

võib olla see, et enamus firmasid kasutavad rakenduse vaikimisi funktsionaalsust, vähesed firmad tellivad funktsionaalsuse täiendavat arendamist Dynamics CRM'il spetsialiseeruvatelt kolmandatelt firmadelt ning vaid väikseim osa loob arenduseks eraldi tiimi.

1.2 Ülesande püstitus

Eesmärgid mida töös saavutatakse:

1. Välja selgitada miks on antud projekti jaoks vajalikud automaattestid.
2. Uurida millised kitsaskohad on antud rakenduses millega tuleb arvestada testide loomisel.
3. Valida sobiv tehnoloogia, tööriistad ja mustrid testide kirjutamiseks arvestades süsteemi eripära.
4. Integreerida testid olemasolevasse koosteprotsessi süsteemi automaatseks käivitamiseks.

1.3 Metoodika

Autor tutvub testitava rakendusega et välja selgitada selle testimiskõlbulikkust ning kitsaskohti millega tuleb testide loomisel arvestada. Samuti tutvub rakenduse arendamiseks kasutuses olevate tööriistade ja tehnoloogiatega et leida ühised puutepunktid. Valib testimis raamistikku ja selle ühildumisvõimalusi olemasolevate tehnoloogiatega. Valitakse mõne funktsionaalsuse kasutusjuhud ning kirjutatakse selle alusel automaattest. Alustatakse testi lihtsamast implementatsioonist ja seejärel rakendatakse erinevaid disainimustreid. Samuti tutvub olemasoleva koosteprotsessi süsteemiga, et leida võimalusi testide integreerimiseks.

1.4 Ülevaade tööst

Antud bakalaureusetöö on jaotatud viieks osaks.

Esimeses peatükis tutvustati töö vajalikkust, eesmärke ja metoodikat.

Teises peatükis kirjeldatakse MS Dynamics CRM'i ning selle ülesehitust. Konkreetse projekti põhist funktsionaalsust.

Kolmandas peatükis uuritakse millised on selle rakenduse kitsaskohad millega tuleb testide loomisel arvestada.

Neljandas peatükis tutvustatakse milliste tööriistade ja tehnoloogiatega tuleb töö käigus kokku puutuda.

Viiendas peatükis kirjeldatakse testide loomist, nende ülesehitust, test andmete ettevalmistust, ning lõpetuseks testide käivitamisest pideva integratsiooni serveris.

2. Testitava rakenduse tutvustus

2.1 Rakenduse tutvustus

Dynamics CRM on kliendisuhete haldus tarkvara pakett mida pakub Microsoft. Sarnaselt teiste CRM tarkvaradele Dynamics võimaldab ettevõtetel hallata potentsiaalsete ja olemasolevate klientide andmed tsentraliseeritud viisil ühest kohast. Selle abil saab mitmeid tegevusi automatiseerida, infot süstematiseerida ja hallata ning erinevate inimeste ja osakondade tööd sünkroniseerida. Antud karbitoode on suunatud põhiliselt müügi, turunduse või teeninduse sektorile kuid Microsoft reklaamib oma toodet ka kui XRM platvormi ning julgustab oma partnereid süsteemi kohandada vastavalt firma vajadustele.

MS Dynamics CRM on server-klient põhine veebirakendus.

Rakenduse kohandamine saavutatav kolme erineva viisiga:

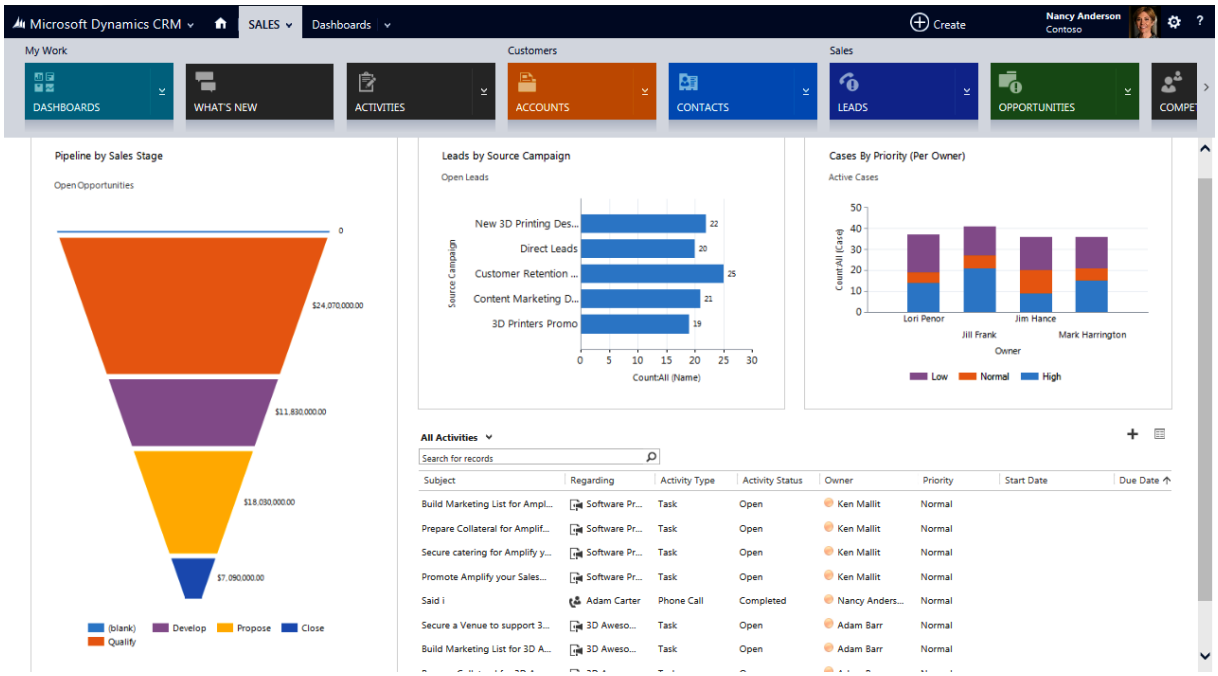
1. Koodi-vaba kohandamine. Dynamics CRM pakub väga paindlikku platvormi süsteemi kohandamiseks (customization). Veebirakenduse administreerimis sektsioonis on ette antud tööriistad millega saab luua uusi üksuseid (entity), luua vorme, lisada välju, tekitada erinevaid vaateid ja lisada lihtsamaid äriprotsesse (business process flows) otse veebilehitsejast.

2. Kliendi poolne kohandamine: Kui koodivabad tööriistad ei ole piisavad siis süsteemi on võimalik täiendada JavaScript koodiga. Selle abil on võimalik peita/kuvada välju sõltuvalt teatud kriteeriumitest, määrata väljad kohustuslikuks/mittekohustuslikuks, täita välju mingite väärtustega sõltuvalt mingist teisest kriteeriumist, kuvada modaalseid dialoogi kasutaja mingi tegevuse peale.

3. Serveri poolne kohandamine: Kui ülalmainitutest vahenditest ei piisa, on süsteemi funktsionaalsust on võimalik täiendada pluginnide abil mida arendajad kirjutatavad C# keeles kasutades .Net Frameworki. Pluginne käivitatakse tavaliselt kui mingi sündmus või tegevus on toimunud, näiteks kirje uuendamine või kustutamine. Plugin kasutab spetsiaalset SDK’d mida pakub Microsoft. Lisaks pluginne on võimalik käivitada koos erinevate äriprotsessidega, mida eelnevalt luuakse kasutajaliidesest. Peale selle kõike

süsteemi on võimalik integreerida kõik muud eraldiseisvad veebirakendused, mis teeb Dynamics CRMi väga paindlikus platvormiks ja mis suudab rahuldada kõiki ettevõte vajadusi.

2.2 Rakenduse funktsionaalsus



Joonis 1. MS Dynamics CRM kasutajaliides.

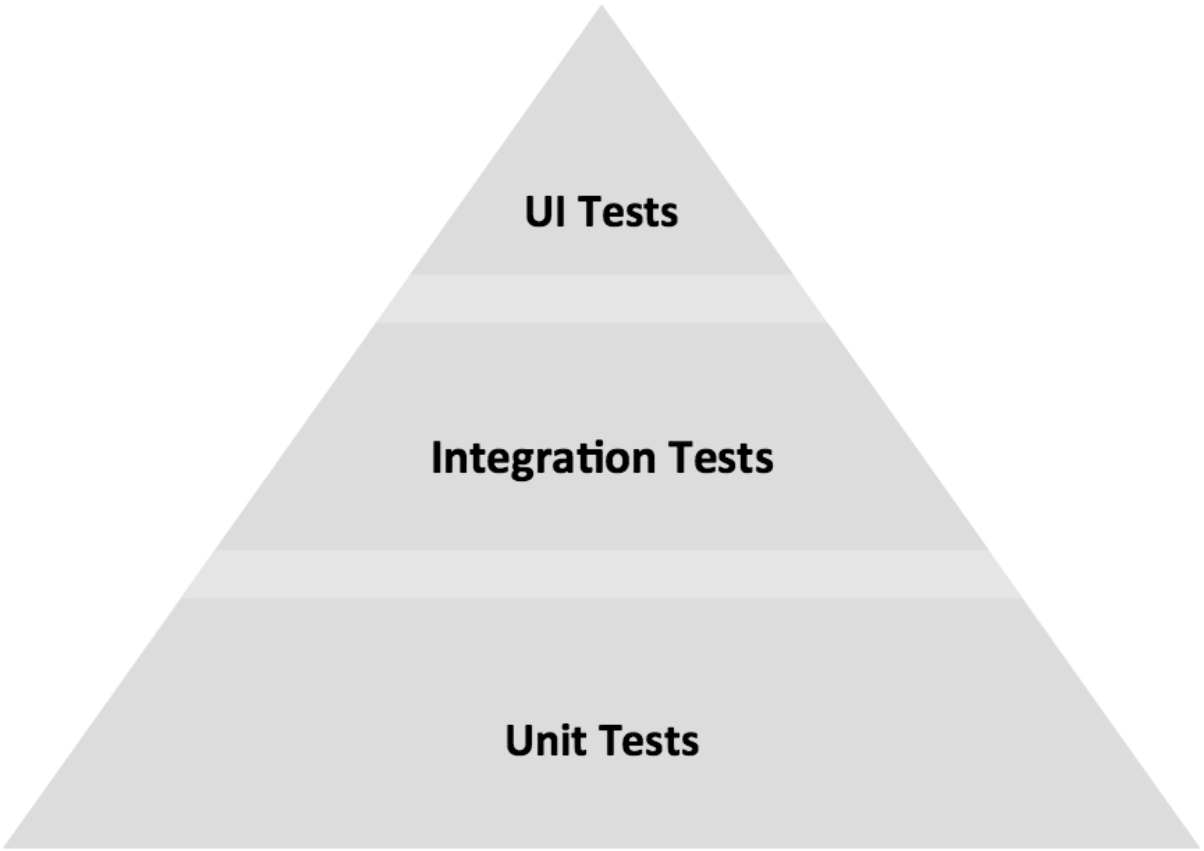
Põhifunktsionaalsus mida kasutajad igapäevatöös kasutavad:

- Klientide haldus: iga kirje sisaldab informatsiooni klientide kohta, nende aadressid, kontaktid, pakkumiste ülevaade, dokumentide ülevaade.
- Kontaktide haldus: kirjed sisaldavad informatsiooni isikute kohta kellega töö käigus kontakteerutakse. Tavaliselt ühel kliendil võib olla mitu kontakte erinevate vastutusaladega.
- Pakkumiste haldus: potentsiaalse müügi kirjed. Sisaldavad informatsiooni marsruudi ning kauba liigi, kaalu, suuruse ja muude parameetrite kohta
- Tenderite haldus: nende alla kuuluvad pakkumised mis hõlmavad suuri hankeid ning vajavad eraldi kinnitust eri osakondade poolt.

2.3 Kasutajaliidese testide vajalikkus

Automaatset testimist peetakse otsustavaks suuremate veebirakenduste edukuses, see säästab palju aega ja aitab tarnida kvaliteetsemat tarkvara.

Testitava rakenduse funktsionaalsus pidevalt kasvab, seda ära testida manuaalselt lühikese aja jooksul on üpris keeruline. Samuti ei ole mõistlik katta kogu funktsionaalsust kasutajaliidese automaattestidega. Mike Cohni testimispüramiidi kontseptsiooni järgi peaks võimalikult paju funktsionaalsuse testimist tegema madalamates kihtides nagu unit-testides või integratsiooni testides, mis on palju kiiremad. Kasutajaliidese testid peaksid katma vaid seda funktsionaalsust millega alumised kihid ei saa katta[4].



Joonis 2. Testpüramiidi kontseptsioon

Kasutajaliidese testimine on protsess millega tehakse kindlaks kas rakendus töötab õigesti ja selles ei esine vigu. Kasutajaliidese testid võivad olla teostatud testija poolt manuaalselt kui

automaatselt kasutades spetsiaalset tarkvara. Automaatsed kasutajaliidese testid on põhimõtteliselt manuaalsete tegevuste automatiseerimine.

Automaatsete testide eelised võrreldes manuaalse testimisega:

1. Automaattestid aitavad vähendada korduvat käsitsi testimist

2. Annavad kiiret tagasisidet

3. Kindlustavad, et testimine on alati teostatud õigete andmetega, eel- ja järeltingimustega.

4. Puudub inimfaktor – testid on täidetud alati kindla stsenaariumi järgi

5. Automaattestid ei vaja inimese sekkumist – teste võib käivitada ka öösiti

Alampeatükis 2.1 tutvustati testitava rakenduse erinevaid funktsionaalsuse laiendamise võimalusi. Eesmärk on katta kasutajaliidese regressiooni ja sutsutestidega Dynamics CRMi firmasiseselt arendatud funktsionaalsust.

Põhiline fookus:

- Ärikriitiline funktsionaalsus, mida kasutatakse kõige rohkem – kui arendajad parandasid viga, lisasid uut koodi või tegid refaktoorimist peab olema kindel et olemasolev funktsionaalsus ei läinud katki.

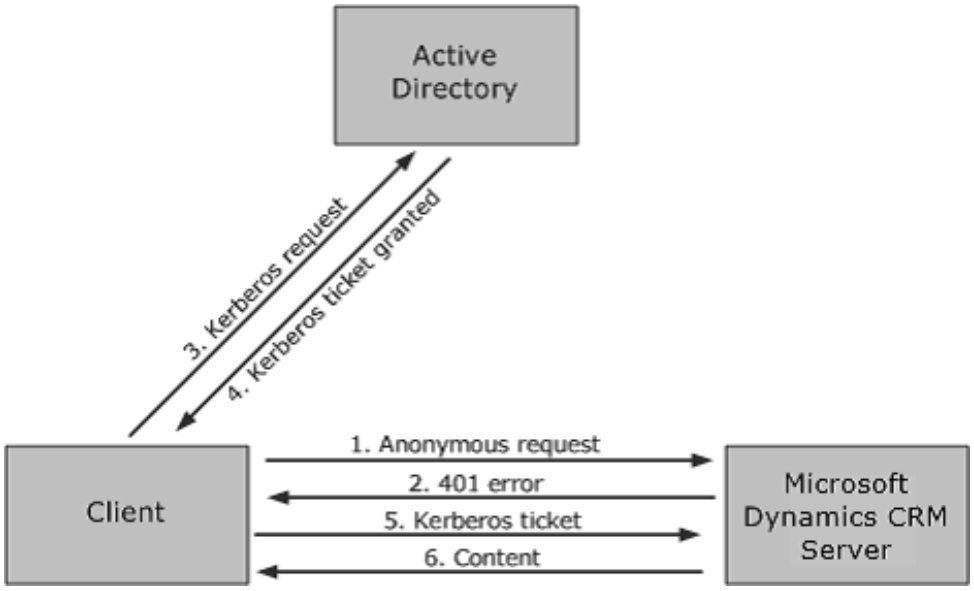
- Aeganõudvad stsenaariumid – on olemas sellised kasutusjuhud kus kogu testi käigus tuleb täita üle saja erinevat välja ja logida sisse mitme erineva kasutajaga selleks et äriprotsessi lõpuni viia.

- Keerulise ettevalmistusega stsenaariumid – sellised kasutusjuhud, kus kõigepealt tuleb luua testiks vajalikud andmed või või viia rakendus kindlasse seisu, alles seejärel alustada testimisega.

3. Dynamics CRM veebirakenduse kitsaskohad automatiseerimise seisukohast

3.1 Autentimine

MS Dynamics CRM kasutab Windows Active Directory autentimist. Kui kasutaja üritab sisse logida anonüümselt siis vastuseks tuleb 401 viga. Nagu eelpool mainitud soovitab Microsoft kasutada eelkõige nende enda veebilehitsejat - Internet Explorerit. Selleks et autentimine töötaks korrektselt peab Internet Exploreri seadetes olema lubatud „Automaatne sisselogimine praeguse kasutajanime ja parooliga“. Sellisel juhul CRMi navigeerides läheb päring Active Directory’sse kus kontrollitakse kas praegusel Windowsi kasutajal on konto olemas, seejärel väljastatakse Kerberos’i *ticket* ja autenditakse CRM’i[5] Automaatne testimine eeldab ka erinevate kasutajate ja rollidega stsenaariumite kivitamist seega sellele tuleb leida sobiv lahendus.

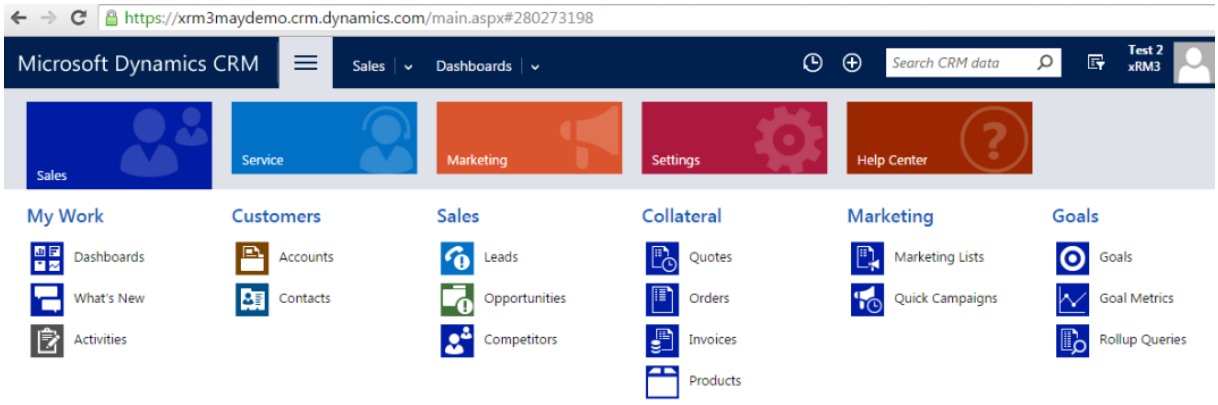


Joonis 3. Active Directory autentimine

3.2 Navigeerimine

Rakenduses navigeerimine toimub hüpikmenüüde abil. Hüpikmenüüd avanevad ka siis kui hiire kursor on menüü kohal. Navigeerimis menüü on alati nähtav sõltumata sellest millisel lehel kasutaja parasjagu viibib. Kui kõik avanenud alammenüüd ei mahu ekraanile siis neid saab paremale ja vasakule kerida.

Tavaliste veebirakenduste puhul sageli navigeerimistee juba sisaldub URL'i sees (näiteks *example.com/homepage/accountpage*). Dynamics CRM'is navigeerides veebilehitseja otsinguribal ei ole unikaalseid linke, on vaid automaatselt genereeritud number (*main.aspx#280273198*) mis vastutab selle eest et oleks avatud õiged kirjed veebilehitsejas „edasi" – „tagasi" nuppudele vajutades.



Joonis 4. Navigeerimine

3.3 Vormide täitmine

Üks erilisemaid MS Dynamics CRM'i kitsaskohti automaattestide jaoks on vormide täitmine. Vormide välju saab liigitada kolmeks: tekstiväli, rippmenüü, otsinguväli. Vaatleme neid lähemalt:

1. Tekstiväljad on kujundatud nii et teksti sisestamist võimaldatakse alles peale seda kui kasutaja klikkib välja peale ja muudab seda aktiivseks:

Joonis 5. Tekstiväli kui kasutaja kursor on selle kohal

Joonis 6. Tekstiväli pärast seda kui kasutaja on selle peale klikkinud

Vaatleme HTML koodi mis on selle välja taga kui väli on mitteaktiivne:

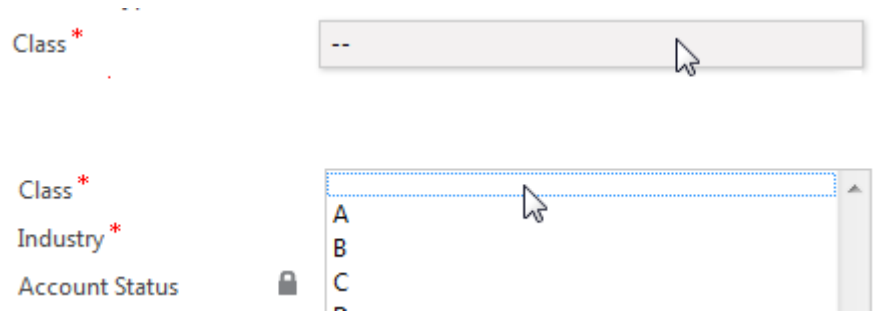
```
<div id="name" class="ms-crm-Inline-Chrome nvarchar" tabindex="1030" data-
layout="0" data-fdeid="PrimaryEntity" data-formid="8448b78f-8f42-454e-8e2a-
f8196b0419af" data-attributename="name" data-initialized="true" aria-
describedby="name_c" title="Select to enter data">
```

HTML kood välja taga kui väli on aktiivne:

```
<input id="name_i" class="ms-crm-InlineInput" type="text" attrname="name"
attrpriv="7" maxlength="400" title="" aria-labelledby="name_c name_w"
controlmode="normal" style="ime-mode: active;">
```

Automaattestide aspektist sellest infost järeldan, et tekstivälja täitmiseks tuleks teha kaks operatsiooni: kõigepealt klikkida elemendile millel on üks lokaator, seejärel sisestada tekst elemendile millel on teine lokaator.

2. Rippmenüü sarnaselt tekstiväljale saab olla nii mitteaktiivne kui ka aktiivne.

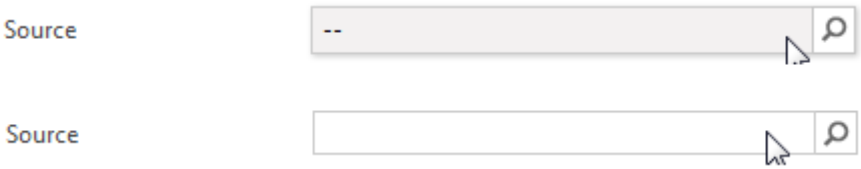


Joonis 7. Rippmenüü aktiivne ja mitteaktiivne olek

Sarnaselt tekstiväljale näeme HTMLi koodis kahte erinevust:

`<div id="accountratingcode"` … ja `<select id="accountratingcode_i"`

3. Otsingu väljad näevad välja järgmiselt:



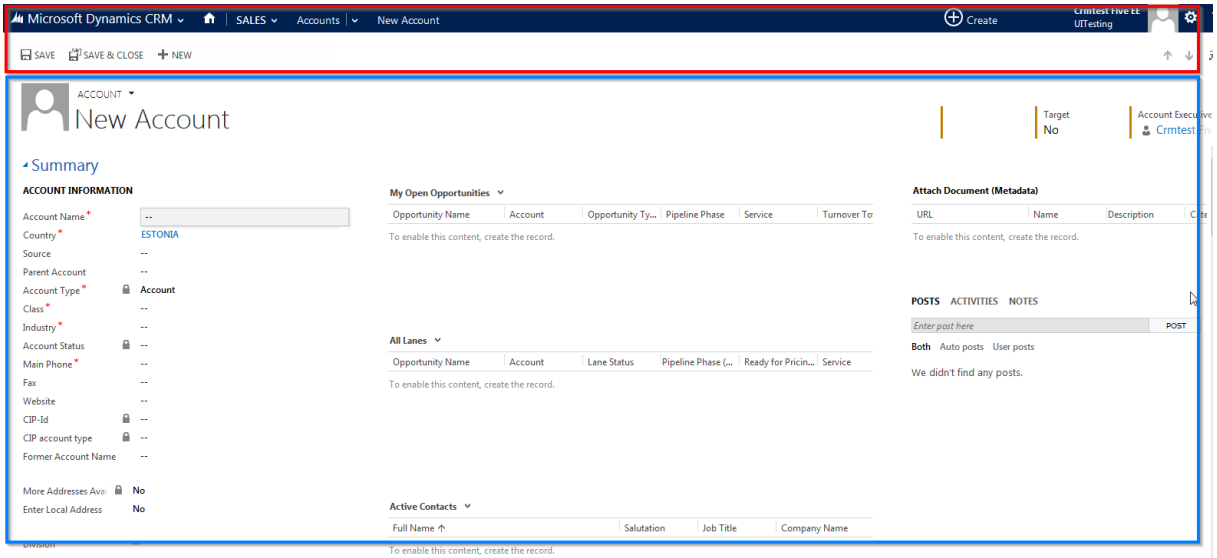
Joonis 8. Otsinguvälja aktiivne ja mitteaktiivne olek

Väljade HTML koodi erinevused:

```
<div id="kn_sourceid" … ja <input id="kn_sourceid_ledit"
```

Eelneva vaatluse põhjal võib järeldada, et väljade täitmiseks kõigepealt tuleb klikkida elemendile lokaatoriga *„id“* ning seejärel sõltuvalt kas tegemist on ühe või teise tüübi väljaga töötada samuti lokaatoriga *„id“* kuid mille sufiks on kas *„_i“* või *„_ledit“*.

3.4 Rakenduse „raamid“ ehk iFrame



Joonis 4. Rakenduse raamid

Vaatleme MS Dynamics CRM'i raamide ehk *iFrame*'ide ülesehitust. Rakendust võib tingimisi jaotada kaheks osaks: vaikimisi raam ja sisu raam. Joonise 4 ülemises osas on alati vaikimisi raam kuid alumine võib muutuda vaatamata sellele et avatakse see sama vorm. See on tingutud sellest et Dynamics CRM salvestab eelnevat raami vahemällu. Näiteks kui

kasutaja avab veebilehitsejat ja navigeerib konto vormile siis sisu näidatakse raamis `<iframe id="contentIFrame1"..` . Kui kasutaja kõigepealt avab kontakti vormi siis sisu näidatakse samuti raamis `<iframe id="contentIFrame1"` aga kui seejärel avab konto vormi siis sisu näidatakse juba `<iframe id="contentIFrame0"` raamis, kuigi vorm on täpselt sama. Automaattestide aspektist tuleb arvestada et sama sisu võib olla erinevates raamides.

4. Tööriistad ja tehnoloogiad

4.1 Visual Studio

Rakenduse arendamiseks tiimis kasutatakse Visual Studio 2015 tarkvara. Automaattestide kirjutamiseks on Visual Studiol kõik eeldused olemas seega otsus langes seda kasutusele võtta. Lisaks tulevikus ka arendajad saavad vajadusel testide kirjutamisel osa võtta.

4.2 Selenium

Selenium on populaarne teek automaattestide kirjutamiseks. Selenium API-ga saab juhtida veebilehitsejat – klikkida nuppudel, linkidel, piltidel, sisestada teksti teisisõnu emuleerida kasutajate tegevust. Seleniumi tagavaks plussiks on tasuta litsents, enamlevinute programmeerimiskeelte tugi, mitme veebilehitsejate tugi ja kõige tähtsam - tugev kogukond internetis.

4.3 NUnit

NUnit on tasuta litsentsiga unit testing teek .Net platvormile. Sellega saab käivitada Seleniumi teste nii IDE keskkonnast kui ka pideva integratsiooni keskkonnast. Alternatiiviks võib kasutada Visual Studioga kaasas olev MSTest, kuid NUnit'i tugevaks plussiks on see et testtsükli jooksul ebaõnnestunud testid saab automaatselt taaskäivitada[6]. Lisaks NUnit toetab testide kategooriate kaupa jaotamist. See võimaldab käivitada teste gruppide või kategooriate kaupa pideva integratsiooni serverist[7].

4.4 NUGet

NUget on tarkvara pakettide haldur millega saab pakette installida tsentraalsest repository'st. NUget automaatselt kopeerib teekide failid projekti ning uuendab konfiguratsiooni faile. Paketi kustutamisel Nuget kustutab vajalikud failid ja taastab konfiguratsiooni.

NUget'i Visual Studio laiendusega on võimalik installida kõik vajaminevad paketid korraga:

```
Install-Package Selenium.WebDriver
Install-Package Selenium.Support
Install-Package NUnit
Install-Package WebDriver.IEDriverServer.win32
```

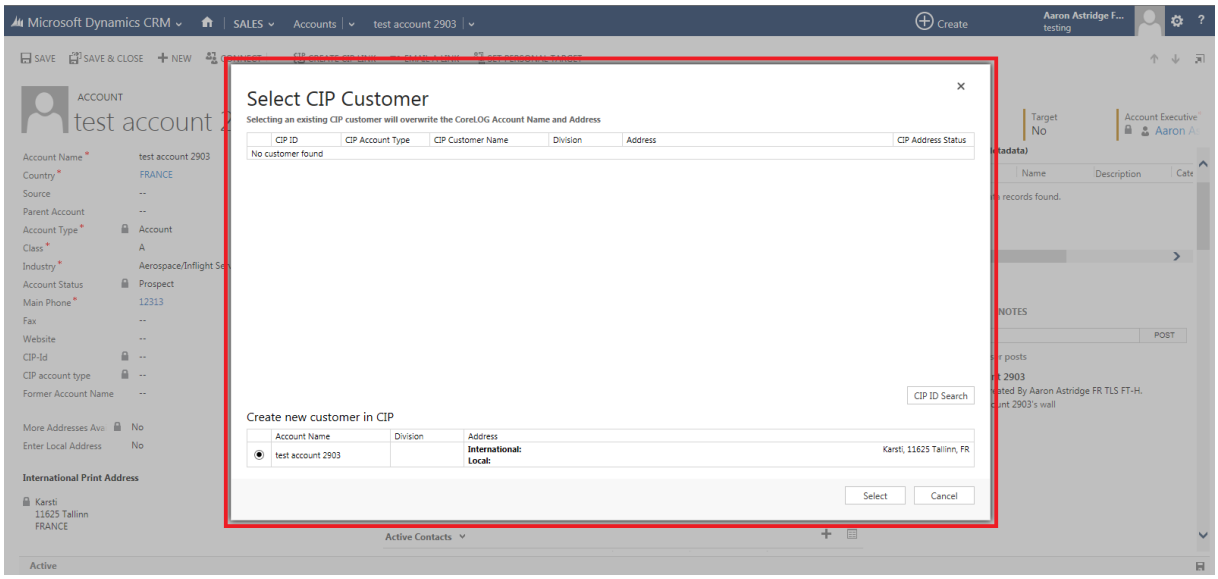
4.5 TeamCity

TeamCity on *build* halduse ja pideva integratsiooni tarkvara. See koosneb kesk serverist mis on ühenduses *agent*'idega, mis omakorda on installitud erinevatesse virtuaalmasinatesse. Server vastutab versioonikontrolli monitoorimise ja ülesannete jagamise eest *agent*'idele (näiteks kompileerimine, testide käivitamine). TeamCity's on *out-of-box* NUniti tugi mis täiendavalt julgustas seda testimis framework kasutusele võtta.

5. Automaattestide implementeerimine

5.1 Kasutujuhud

Selles peatükis vaatleme üks väljavalitud kasutusjuht mille näitel loome töötav UI test ning abistavad klassid ja meetodid. Antud kasutusjuhus toimub konto loomine ning selle sidumine ühe teise infosüsüsteemiga. See funktsionaalsus on loodud kasutades kõiki kolme Dynamics CRM'i laiendamise vise. Konto vorm on tehtud koodi-vaba *customization*'iga, kasutades administraatori menüüs pakutavaid tööriistu. Avanenud *iFrame*'is on arendajate poolt tehtud HTML vorm koos JavaScripti funktsioonidega. "Select" nupu vajutamisel toimub andmete salvestamine serveri poole peal kasutades samuti majasiseselt arendatud .Net pluginnide funktsioone.



Joonis 10. Dialoog konto sidumiseks CIP infosüsteemiga. Osa firmasisest arendust.

Kirjeldus	Kasutusjuht `NewAccountCreateNewCipTest.cs` : Konto loomine ning sidumine CIP süsteemiga.
Sammud	1. Logi sisse tavakasutajana 2. Navigeeri Konto lehele 3. Ava uue konto vorm 4. Täida konto andmed 5. Salvesta konto 6. Klikki “Create CIP Link” 7. Vali “Uus konto”
Oodatav tulemus	Kontole on lisandunud CIP-id number

Tabel 1. Kasutusjuht

5.2 Veebi elementide lokaatorite määramine

Alampeatükis 3.3 olid välja toodud MS Dynamics CRM vormide täitmise eripärad koos vormielementide lokaatorite näidetega. Selles alampeatükis analüüsitakse veebielementide lokaatorite leidmist MS Dynamics CRM’is.

Selenium WebDriver kasutab niinimetatud *lokaatoreid* nuppude, väljade, linkide ja muude elementide leidmiseks ja manipuleerimiseks veebilehel. Selenium toetab kokku 8 erinevat lokaatori tüüpi: *ID, Name, Link Text, Partial Link Text, Tag Name, CSS Class, CSS Selector, XPath.* Järgnevates testide koodinäites eelistatakse enamasti ID lokaatoreid ja seda mitmel põhjusel. ID on sarnaselt Name lokaatoriga elemendi leidmise kiiruse aspektist kõige efektiivsemad [8]. W3C standardi järgi ID peab olema unikaalne igal veebielemendil [9], see tähendab et ei tohiks ette tulla probleemi kui ühe IDga on leitud mitu elementi. ID ei sõltu elemendi tüübist ega hierarhilisest paigutusest seega kui arendaja muudab elemendi tüübi või paigutust WebDriver jätkuvalt suudab seda üles leida. ID-d sageli kasutatakse JavaScripti poolt seega arendaja peaks vältima nende muutmist. Peale kõike selle Microsoft julgustab

kasutada ID-sid elementide leidmises JavaScriptis[10], selle tõttu on ka kõikides elementides ID-d automaatselt määratud MS Dynamics CRMi poolt. Firma sisestes arendustes arendajad jälgivad head tava ja määravad igale veebielementidele käsitsi ID lokaatoreid. Eranditeks võib pidada tabeleid kus elementide leidmiseks kasutatakse testides XPath lokaatoreid.

5.3 Väljavalitud kasutusjuhu automaattest

Selles alapeatükis loome eelnevalt väljavalitud kasutusjuhu testi kasutades Selenium WebDriver'i.

```
public class NewAccountCreateNewCipTest
    {

        [Test]
        public void NewAccountCreateNewCip()
        {
            //draiveri initsialiseerimine
            IWebDriver driver = new InternetExplorerDriver();

            //sisselogimine
            string urllogin = "https://GER%5C" + username + ":" + User.Password + "@"
                                + Settings.Default.ServerUrl;
            driver.Navigate().GoToUrl(urllogin);
            driver.Navigate().GoToUrl(Settings.Default.ServerUrl);

            //navigeerimine konto lehele
            driver.FindElement(By.Id("TabSFA")).Click();
            driver.FindElement(By.Id("nav_accts")).Click();

            driver.SwitchTo().Frame("contentIframe1");

            //uue vormi avamine
            driver.FindElement(By.Id("account|NoRelationship|HomePageGrid|
              Mscrm.HomepageGrid.account.NewRecord")).Click();

            //nime sisestamine
            driver.FindElement(By.Id("name")).Click();
            driver.FindElement(By.Id("name_i")).SendKeys("Tere AS");

            //konto valdkonna valimine
            driver.FindElement(By.Id("accountratingcode")).Click();
            new SelectElement(driver.FindElement(By.Id("accountratingcode_i")))
              .SelectByText("Aerospace");

            //telefoni sisestamine
            driver.FindElement(By.Id("telephone ")).Click();
            driver.FindElement(By.Id("telephone_i")).SendKeys("55545555");

            //linna sisestamine
            driver.FindElement(By.Id("city")).Click();
            driver.FindElement(By.Id("city_i")).SendKeys("Tallinn");
```

```
            //salvestamine
            driver.FindElement(By.Id("save")).Click();

            Assert.AreEqual("Tere AS", driver.FindElement(By.Id("name")).Text);

            //CIP vormi avamine
            driver.FindElement(By.Id("newcip")).Click();

            //CIP kinnitamine
            driver.FindElement(By.Id("approve")).Click();

            Assert.AreEqual("Operational", driver.FindElement(By.Id("type")).Text);


        });
```

Kuigi see test täidab oma funktsiooni on selles on järgmised probleemid:

1. Selle loetavus on väga madal. Koodis on palju WebDriver'i meetodeid mis pimendavad kogu testi eesmärki.

2. Madal hooldatavus. Selles testis on kirjeldatud rakenduse lokaatorid. Kui kasutajaliides peaks muutuma aga äriloogika jääb samaks siis peab lokaatorid muutma kõikides testides.

3. Madal korduvkasutatavus. Kui piirduda üksikute testidega siis kodruvkasutatavus ei ole tähtis argument kuid kui kirjutada regressiooniteste on korduvkasutatavus oluline aspekt.

4. Programmeerimise aspektist on oluline probleem duplikaadid koodis. Üks võimalus seda vältida on paigutada WebDriveri väljakutseid *wrappida* eraldi meetoditesse [11]. Lisaks siin näeme punktis 3.3 mainitud MS Dynamics CRM vormide täitmise kitsaskohti ehk topelt tegevused väljade täitmisel.

5.4 Page Object disainimuster

Eelmises alampeatükis kirjeldatud probleemidele lahenduseks on PageObject disainimustri rakendamine. Selle mudeli järgi on igal rakenduse veebilehel vastav repositooriumi klass, kus on kapseldatud kõik võimalikud manipulatsioonid elementidega ja nende lokaatorid ühe konkreetse veebilehe jaoks. Sellisel juhul, pärast muudatust rakenduse kasutajaliideses tuleb muuta vaid konkreetse veebilehe Page Object'i, testid aga, mis kasutavad seda lehte jäävad muutmata[12][13].

Page Object'ide mudel võimaldab erinevat abstraktsuse tasemet testi- ja Selenium Webdriver'i koodi vahel. Selle arendamisel esmalt peab kindlaks tegema kas veebilehte tuleb jaotada struktuurselt või funktsionaalselt[14].

Page Objecti funktsionaale implementatsiooni näide: veebilehte jaotatakse funktsionaalsuse järgi

```
public class AccountPage
    {
        public AccountPage(IWebDriver driver) : base(driver)
        {
        }

        internal void CreateAccount(string name, string city)
        {
              driver.FindElement(By.Id("name")).Click();
              driver.FindElement(By.Id("name_i")).SendKeys("name");

              driver.FindElement(By.Id("city")).Click();
              driver.FindElement(By.Id("city_i")).SendKeys("city");

              driver.FindElement(By.Id("save_btn")).Click();


        }
    }
```

Sellise lähenemise eeliseks on see et veebilehe struktuur on täielikult abstraheeritud testide kihist. Näiteks kui kasutajaliidesesse lisatakse märkeruut „nõus tingimustega" ja see peab olema testide jaoks valitud siis seda tuleb muuta vaid ühes meetodis ja see ei avalda mingit mõju testide kihile sest nemad jätkivalt kasutavad sama meetodit konto loomiseks.

Page Objecti struktuurse implementatsiooni näide: veebilehte jaotatakse elementide järgi, millega tuleb testide koostamisel manipuleerida.

```
public class AccountPage
    {
        public AccountPage(IWebDriver driver) : base(driver)
        {
        }

        internal void TypeName(string name)
        {
              driver.FindElement(By.Id("name")).Click();
              driver.FindElement(By.Id("name_i")).SendKeys("name");
        }
        internal void Select(string city)
              driver.FindElement(By.Id("city")).Click();
              driver.FindElement(By.Id("city_i")).SendKeys("city");
        }
        internal void ClickSave()
              driver.FindElement(By.Id("save_btn")).Click();


        }
    }
```

Selline lähenemine annab rohkem paindlikkust kuna see avaldab kõiki elemente veebilehel. Vastavalt vajadusele neid saab testide kihis kombineerida või muuta järjestust. Selle puuduseks on see et kui kasutajaliidesesse lisatakse märkeruut nagu ülaltoodud näites siis koodi tuleb lisada nii Page Object klassi kui ka testi klassi.

MS Dynamics CRM'i puhul on mõistlik kasutada mõlemat lähenemist. Struktuurset lähenemist juhul kui on näiteks konto loomise vormile arendajate poolt lisatud täiendavat JavaScripti loogikat milleta tuleb testide käigus arvestada – märkeruut tühjaks jätmise korral tuleb ette modaalne dialoog mingi teavitusega. Funktsionaalset lähenemist aga siis, kui mingi testi käigus on vaja kombineerida mitu järjestikku tegevist millel on vaid karbitoode funktsionaalsus ning seda ei ole tarvis eraldi testida.

PageObject'i rakendamise eelised:

1. Testide kood ja veebilehe elementide kood on üks teisest eraldatud. Kui tehakse muudatus kasutajaliideses aga äriloogika jääb samaks siis ka testid jäävad muutmata.

2. Mitu erinevat testi saavad taaskasutada ühte PageObjecti klassi. Tänu sellele saab teha erinevaid testi kombinatsioone kasutades samu PageObject'e.

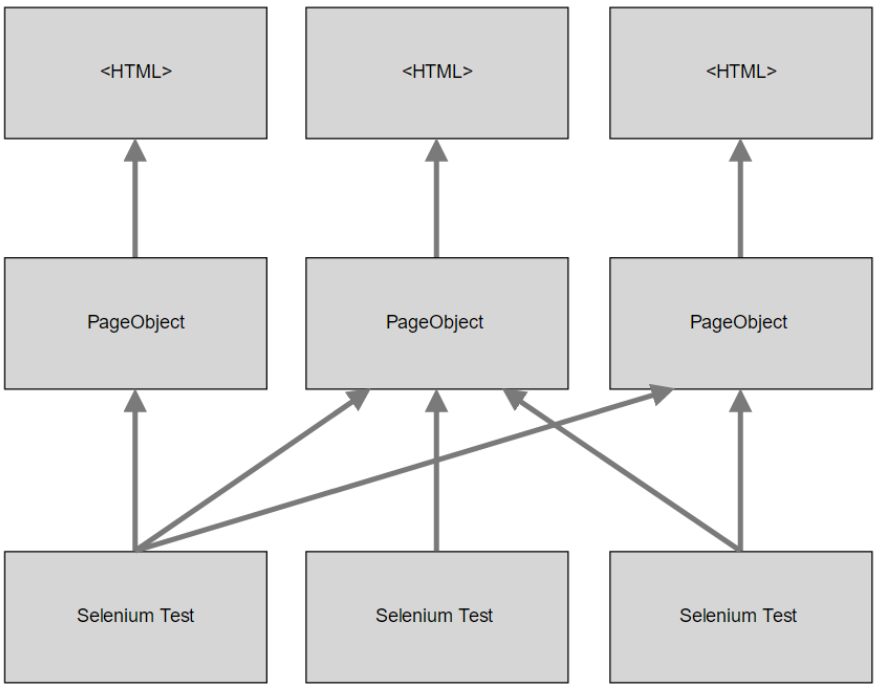
3. Meetodite nimetused on realistlikumad ja arusaadavamad mis teeb testi lugemist ja kirjutamist kergemaks. Näiteks nime sisestamiseks on loodud meetod “TypeName”, valdkonna valikuks “SelectIndistry” ja salvestamiseks nuppu vajutamisel “ClickSave”.

PageObject’i rakendamise puudused:

1. Testide keerukus suureneb. Nagu nimigi ütleb ei saa enam teste kirjutada protseduurses stiilis - peab investeerima aega ja ressurssi raamistikku loomisesse.

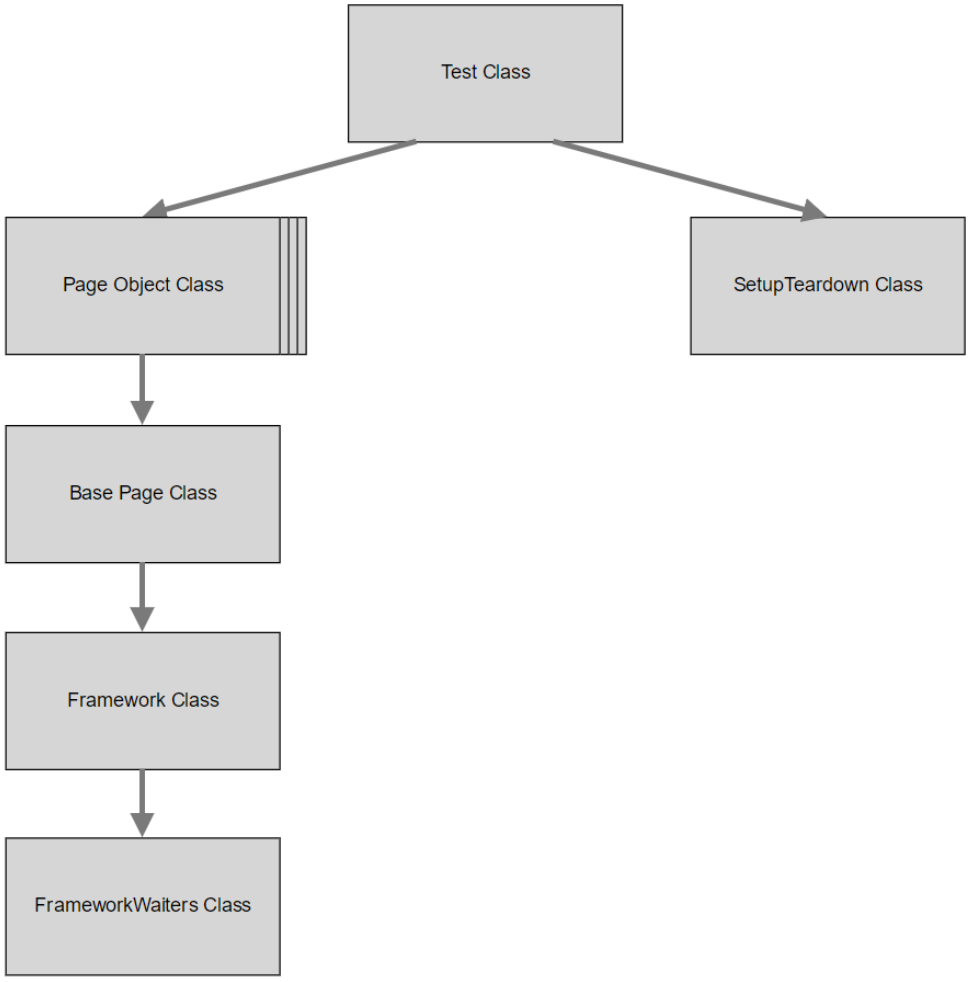
2. Tuleb järgida programmeerimise hea tava printsiipe, et kood oleks järjepidev ja lihtsasti arusaadav, muidu see võib kiiresti muutuda segaseks ja raskesti hooldatavaks – see nõuab testijalt täiendavaid oskusi.

Võrreldes eeliseid ja puuduseid tuleb lähtuda sellest, kui palju kasutusjuhte on plaanis automatiseerida. Kui vajadus on vaid üksikute suitsutestide järele, siis piisab ka protseduurselt kirjutatud ja hästi kommenteeritud koodist. Kui vajadus on aga nii suitsutestide kui ka regressioonitestide järele, ning testitava rakenduse funktsionaalsus kasvab tulevikus veelgi, on mõistlik rakendada PageObject disanimustrit, sest hiljem testide kirjutamine ja haldamine muutub palju kiiremaks ja lihtsamaks.



Joonis 11. PageObject disainimuster

Antud projekt on koostatud lähtudes Page Object disainimustrist kus igal rakenduse lehel on oma klass kus kirjeldatud kõik konkreetse veebilehe lokaatorid, näiteks nii nupud, tekstiväljad, rippmenüüd, kui ka raamid koos muude elementidega mis on omased ainult sellele lehele. Page Object klass laiendab BasePage klassi, kus on kirjeldatud lokaatorid mis võivad olla osa ükskõik millisest muust lehest, näiteks menüü. BasePage laiendab omakorda Framework classi kus on kirjeldatud meetodid kuidas veebilehe elementidega simuleerida kasutaja tegevust, näiteks klikkimine, teksti sisestamine, rippmenüüde valimine. FrameworkWaiters on kõige madalamal tasemel olev klass kus kirjeldatakse elementide sündmusi ja ooteaegu. Base klassis, mis pärineb Test klassist, kirjeldatakse testi käivitamise eel ja järeltingimusi nagu brauseri avamine ja sulgemine.



Joonis 12. Projekti struktuur.

Võttes aluseks teadmisi punktis 5.1 ja 5.2 vaatleme antud kasutusjuhu testi realiseerimist kasutades struktuurset Page Object mustrit detailsemalt järgmistes alampeatükkides.

5.4.1 Testi klass

Selles alampeatükis on näidatud sisuliselt sama test mis oli punktis 5.3 kuid juba kasutades Page Object disainimustrit. Sellest koodist on kerge välja lugeda mida test täpsemalt. Seda koodi saab korduvkasutada ja kombineerida teiste test stsenaariumite loomise jaoks.

```
public class NewAccountCreateNewCipTest : SetupTeardown
{
    string name = "TestNewCipAccount " + DateTime.Now;

    [Test]
    [Retry(User.RetryCount)]
    [Category("Nightly")]
    public void NewAccountCreateNewCip()
    {
        Assert.DoesNotThrow(() =>
        {
            Login(User.Sales5);
            var accountPage = new AccountPage(driver);
            accountPage.NavigateSalesAccounts();
            accountPage.ClickNew();
            accountPage.TypeName(name);
            accountPage.SelectClass("A");
            accountPage.SelectIndustry("Aerospace/Inflight Services");
            accountPage.TypeTelephone("+37255123123");
            accountPage.TypeCity("Tallinn");
            accountPage.TypePostalCode("11625");
            accountPage.ClickSave();

            Assert.AreEqual(name, accountPage.GetHeader());

            var cipFrame = new AccountCIPFrame(driver);
            accountPage.ClickCreateCIPLink();
            cipFrame.ClickSelectBtn();

            Assert.AreEqual("10", accountPage.GetCIPId().Length.ToString());
            Assert.AreEqual("Operational", accountPage.GetCIPAccountType());

        });
```

Testklass laiendab SetupTeardown klassi mida tutvustatakse järgmistes peatükkides.

Test on NUniti atribuut mis määratleb antud meetodit testi meetodiks.

Retry on NUniti atribuut mis testi ebaõnnestumise korral laseb testi uuesti käima esimese õnnestumiseni või kuni „n“ korda. Praktilistes katsetustes tuli välja et UI testid oma loomust ei ole väga stabiilsed veebilehitseja, serveri või muude faktorite tõttu, eriti siis, kui rakendus sõltub kolmanda osapoole teenustest või rakendustest. Näiteks antud näite testi osa on CIP numbri määramine mida küsitakse kolmanda osapoole serverist. Selleks et testid ei katkeks

rakendusest mittesõltuval põhjusel laseme ebaõnnestumise korral testi korrata kuni 3 korda. Kordamise võimlas on peamine põhjus miks sai valitud NUnit raamistik mitte Microsofti oma MSTest.

Category on samuti NUniti atribuut mis lubab teste jaotada kategooriate järgi selleks et neid erineval ajal või erinevas järjekorras käivitada. Sellest räägitakse lähemalt punktis 5.5.

Testi alguses logitakse kasutaja sisse. Pärast sisselogimist luuakse AccountPage klass milles on määratud kõik selle veebilehe elemendid. Testi käigus täidetakse vormi ning salvestatakse. Seejärel kontrollitakse kas salvestamine õnnestus. Pärast seda toimub konto sidumine välise CIP infosüsteemiga ning kontroll kas konto sai CIP-numbrit ja õiget staatust.

5.5 DRY printsiip

Eelnevalt tutvustati Page Object disainimustri, mis on antud projekti testide aluseks, kuid sellel on veel arenguruumi. Kõige ilmsem probleem on selles et veebirakenduses võivad olla selliseid elemente mis võivad korduda paljudel, kuid mitte kõigil teistel lehtedel. Selleks et mitte kirjeldada korduvaid elemente igas Page Object'i klassis on mõistlik DRY (Don't Repeat Yourself) printsiipi järgides eemaldada mittevajalikud duplikaadid ja kasutada pärimist[15]. Base Object klassi kirjeldatakse ühiseid abstraktseid elemente näiteks veebilehe menüü, päis, jalus, hüpikaknad. Iga Page Object klass pärineb Base Page klassist.

DRY printsiibi alla võib liigitada ka testi eel- ja postoperatsioone nagu SetUp() ja TearDown() mis on ühised meetodid iga testi jaoks. Nendes meetodites ette valmistatakse testimiskeskkonda ja käivitatakse veebilehitesejat, testi lõpus veebilehitsejat suletakse, samad tegevused iga testi alguses ja lõpus. Seega iga testi klass pärineb SetupTeardown klassist.

Liikudes projekti hierarhias veel madalamale kihile saab DRY printsiipi rakendada mitte ainult konkreetsete veebielementide kirjeldamiseks vaid ka ühiseid tegevusi nende elementidega näiteks klikkimine ja teksti sisestamine. Nagu eelnevalt oli mainitud Dynamics CRMis teksti sisestamiseks teksti väljale on vaja teha kaks järjestikkust operatsiooni:

```
driver.FindElement(By.Id("name")).Click();
driver.FindElement(By.Id("name_i")).SendKeys("Tere AS");
```

Neid operatsioone saab *wrappida* ehk sisse pakkida ühe geneerilise meetodi „Type" alla:

```
Type(By.Id("name"), "Tere AS");
```

Sarnast lähenemist saab rakendada ka rippmenüü, otsinguvälja ja teistele geneerilistele tegevustele. MS Dynamics CRMi puhul see on eriti oluline selle pärast, et uuema versiooni välja tulekul võib muutuda vormide kujundus või väljade käitumine. Seega DRY printsiibi rakendamisel koodis on vähem duplikaate ja seda on kergem hallata. Veebilehtede kattuvad elemendid on ühes klassis ja iga elemendi tüübi jaoks on loodud geneerilised meetodid. BasePage, SetupTeardown ja geneerilise meetoditega klasside implementatsiooni tutvustatakse järgmistes alampeatükkides.

5.5.1 Konkreetse veebilehe klass ja baas klass

Kasutades vaid Page Object mustrit, mis on toodud välja koodi näides Lisa 1, on kirjeldatud osa konto veebilehe unikaalseid elemente ja tegevused nende elementidega. Kuna paljud elemendid võib liigutada mingi tüübi, näiteks andmete sisestamine otsinguväljale, tavalise tekstiväljale või kuupäeva väljale, saab nende jaks luua eraldi meetodid mis täidavad kindlat funktsiooni. Lisa 2 koodinäites on näidatud sama veebilehe nii Page Object muster kui ka DRY muster: kõik loogika, millega erinevate tüübiga elemente manipuleeritakse on abstraheeritud ja viidud madalamasse kihti – Framework klassi.

Alampeatükis 3.2 tehti kindlaks et ka Dynamics CRM'is on selliseid elemente mida saab näha mitmel erineval veebilehel – näiteks navigatsiooni paneel. Ükskõik millisele lehele kasutaja satub navigatsioonipaneel on alati nähtav. Selleks on loodud Base page klass kus saab kirjeldada kõikide veebilehtede ühiselemente. Kõik konkreetsete veebilehtede PageObject klassid pärinevad baas-PageObject klassist.

5.5.2 Framework klass

Baas Page klass kasutab Framework klassi meetodeid selleks, et teha mingit tegevust konkreetsete veebielementidega. Punktis 3.3 tutvustati MS Dynamics CRM'i eripära erinevate elemendi tüüpide ja vormide täitmise osas.

Type on teksti sisestamise meetod tekstiväljale. Alampeatükis 5.2 tehti kindlaks et eelistatum lokaator elementide leidmiseks on Id. Type meetodis kõigepealt parsitakse By objekti et kindlaks teha mis lokaatoriga on tegemist: kas Id või mingi muuga. Esimesel juhul toimub tekstivälja peale klikkimine etteantud lokaatori järgi. Seejärel tekstiväli muutub aktiivseks ning otsitakse juba tekstivälja teine element „_i" sufiksiga. Pärast seda toimub teksti sisestamine. Kui Type meetodile anti ette element mingi teise lokaatoriga kui „Id", siis toimub teksti sisestamine nagu tavalise tekstivälja jaoks, ilma esmase klikkimiseta (vt koodinäide Lisa 4).

Sarnaselt Type meetodile on realiseeritud TypeLookup ja Select meetodid mis vastutavad otsinguvälja ja rippenüü eest. Kõigepealt tehakse kindlaks mis lokaatoriga on tegu, kui „Id" siis toimub 2 järjestikkust tegevust: kõigepealt klikkitakse elemendile seejärel sisestatakse tekst või valitakse väärtus rippmenüüst.

5.5.3 FrameworkWaiters

Kõige madalama taseme klass mis kutsub otse Selenium Webdriver'i meetodeid. Framework klass pärineb FramworkWaiters klassist.

WaitForVisible meetodis otsitakse elementi teatud aega kuni see muutub nähtavaks.

Punktis 3.4 toodi välja MS Dynamics CRM'i eripära raamide kuvamise suhtes. Nimelt üks ja sama veebileht võib olla erinevas raamis sõltuvalt sellest kas seda avati kohe või pärast teatud tegevusi. WaitForFrameAndSwitch meetod kõigepealt läheb vaikimisi raami peale seejärel selle raami peale mis on nähtav. Praktiliste katsetuste käigus selgus et see meetod alati suudab leida õiget raami ja test saab jätkuda.

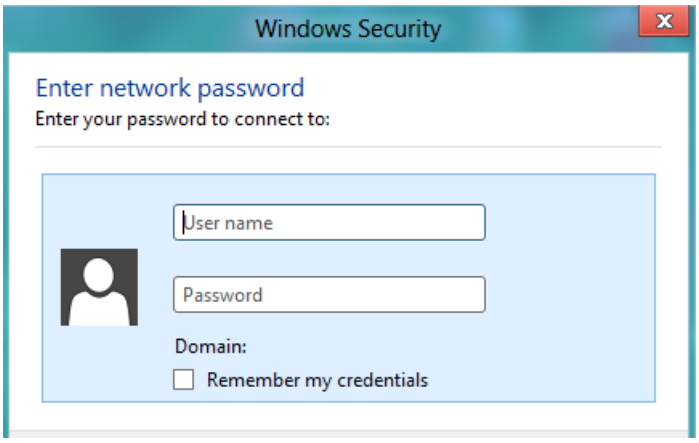
5.5.4 Setup Teardown klass

Lisa 5 toodud klassi koodinäites toimub veebilehitseja käivitamine aja sulgemine vastavalt *SetUp()* ja *TearDown()* meetodites.

Serveri URL on tõstetud eraldi projekti Settings faili, selleks, et vajadusel testi käivitada erinevates keskkondades ilma koodi muutmata.

Testide praktilistes katsetustes on välja tulnud, et vahel harva Internet Exploreri või WebDriveri protsess testi lõpus jääb Windowsi Task Manageris rippuma. Seega sai lisatud *SetUp* klassi täiendav meetod mis kontrollib kas parasjagu ei ole ühtegi protsessi rippuma jäänud vanast testist, positiivsel juhul protsessi lõpetatakse, alles seejärel käivitatakse test.

Klassis on realiseeritud ka veebirakendusse autentimise meetod. Vaatleme Login meetodit mida kutsutakse välja testklassidest. Rakenduse tutvustuses oli mainitud, et Dynamics CRMi üks eripäradest on see, et sellel puudub autentimise võimalus veebi kaudu. Veebirakendus kasutab Windowsi kasutaja Active Directory andmeid ja logib nendega automaatselt sisse. Kasutajaliidese testimiseks peab olema võimalus logida sisse erinevate kasutajatega. Internet Exploreri seadetes saab automaatset sisselogimist küll välja lülitada, kuid kuna Seleniumi on mõeldud veebilehtede kontrollimiseks, ei ole võimalik sellega kontrollida Windowsi-põhist autentimise dialoogi akent.



Joonis 2. Windowsi-põhine autentimine

Erinevalt teistest levinud veebilehitsejatest nagu Chrome ja Firefox, Internet Explorer ei luba ka vaikimisi kasutada URLi põhist autentimist sellisel kujul nagu *http://username:password@example.com/*. Microsoft pakub siiski võimalust sellest piirangust mööda minna[16]. Windowsi registris tuleb muuta DWORD väärtust 0-lt 1 peale järgmises

kohas:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Internet
Explorer\Main\FeatureControl\FEATURE_HTTP_USERNAME_PASSWORD_DISABLE
```

5.6 Testide integreerimine MS Dynamics CRM API-ga

Selleks et test tervikuna oleks tõhus ja töökindel ei piisa testide käivitamisest ainult kasutajaliidesest. Korralikult disainitud testi põhilised omadused on:

1. Täpsus: test peaks tegema kindla asja – kui rakenduses on mingi viga mis ei ole otseselt seotud testiga siis see ei peaks testi ebaõnnestuma.

2. Sõltumatus: ühe testi tulemus ei pea mõjutama mingi teise testi tulemust.

3. Kiirus: mida kiiremini test jookseb seda kiiremini saab tagasisided ja seda kasulikum on test.

Need omadused kindlasti ei ole ainukesed mis teevad testi heaks, küll aga ainult Selenium WebDriver'iga raskesti saavutavad. Põhjus on selles et paljud kasutajaliidese testid võivad sisaldada näiteks nii konto loomist, autentimist, navigeerimist ja alles pärast neid tegevusi algab konkreetse funktsionaalsuse testimine.

Selleks et andmete ettevalmistumist või veebirakenduse viimist mingisse kindlasse seisu mitte kasutada kasutajaliidest lahenduseks võib kasutusele võtta veebiteenuseid (web services) näiteks REST, SOAP või SQL skriptid andmete sisestamiseks otse andmebaasi.

Tõenäoliselt SQL skriptid oleks kiireim lahendus kuid Dynamics CRMi puhul tabelite, vaadete, andmete sisestamine, muutmine või kustutamine otse andmebaasist Microsoft liigitab kui mittetoetatavat modifikatsiooni[17], mis võib viia ettearvamatu tagajärgedeni, kuigi tehniliselt see on võimalik. Samuti ei garanteeri Microsoft selliste muudatuste korrektset töötamist pärast Dynamics CRMi uuendust. Toetatud andmebaasi tegevuste alla võib liigitada vaid indeksite lisamine või uuendamine.

Microsoft pakub erinevaid veebiteenuseid rakenduse laiendamiseks või andmete lugemiseks/muutmiseks kasutajaliidese väliselt. Üks kõige levinum arendajate seas on „Organizaton service“, samuti tuntud kui „SOAP endpoint“. Oganization veebiteenus on optimeeritud .Net raamistiku jaoks ning kõik serveri-poolsed pluginnid ja töövood samuti kasutavad seda veebiteenust.

Testide praktiliste katsetuste käigus tuli välja et praegustes testjuhtudes on olemas mitmed testid millistes on ühine osa. Punktis 5.1 on kirjeldatud kasutusjuht kus luuakse konto ja seejärel seotakse välise infosüsteemiga. Vaatleme näiteks sellist kasutusjuhtu kus luuakse konto, seotakse välise infosüsteemiga juba mingite teiste andmetega. Nendel kahel testil on 2 ühist osa – konto loomine. Seda võib liigitada testi eel-operatsiooniks.

Selle projekt kirjutamise ajal arendajad juba kasutasid Organization veebiteenust integratsiooni testide jaoks. Kuna osadel integratsiooni testidel on kasutajaliidese testidega kattuvad ühisosad otsustati taaskasutada seda API-d ka kasutajaliidese testide jaoks.

```
public class NewAccountCreateNewCipTestV2Test : SetupTeardown
    {

        string name = "TestNewCipAccount " + DateTime.Now;

        [Test]
        [Retry(User.RetryCount)]
        [Category("Nightly")]
        public void NewAccountCreateNewCipV2()
        {
            Assert.DoesNotThrow(() =>
            {
                var accountService = new AccountService(User.Sales5);
                var accountGuid = accountService.CreateAccount(name);

                Login(User.Sales5);
                var accountPage = new AccountPage(driver);
                accountPage.Navigate(accountGuid.ToString());

                var cipFrame = new AccountCIPFrame(driver);
                accountPage.ClickCreateCIPLink();
                cipFrame.ClickSelectBtn();

                Assert.AreEqual("10", accountPage.GetCIPId().Length.ToString());
                Assert.AreEqual("Operational", accountPage.GetCIPAccountType());

            });

        }
    }
```

Selles näites on sama test mis punktis 5.3.2 kuid nüüd on konto loomiseks kasutusel Organization veebiteenuse API. Kõigepealt initsialiseeritakse AccountService klass ja määratakse kasutaja millega hakatakse toiminguid tegema. Seejärel kutsutakse CreateAccount meetod ja edastatakse tulevase konto nime. Meetod tagastab selle konto unikaalset GUID. Seejärel toimub veebilehitseja avamine ja navigeerimine juba eelnevalt genereeritud kontole. Pärast seda toimuvad samad tegevused mis kasutusjuhus 5.3.2.

```
internal class AccountService : BaseService
    {
        public AccountService(string username) : base(username)
        {
        }

        internal Guid CreateAccount(string name)
        {
            using (var xrmService = CreateXrmService())
            {
                var CountryId =
                xrmService.Repository<Country>()
                .GetFirstOrDefault(x=>x.Name == "ESTONIA", x => x.CountryId);

                Account = new Account
                {
                    Name = name,
                    AccountTypeGlobal = Account.AccountTypeGlobalEnum.Account,
                    Address1CountryId = new EntityReference(Country
                            .EntityLogicalName, CountryId.Value),
                    AccountRatingCode = Account.AccountRatingCodeEnum.A,
                    Industry = Account.IndustryEnum.Automotive,
                    Telephone1 = "55555555",
                    Address1_City = "Tallinn",
                    Address1_PostalCode = "11655"


                };
                xrmService.Repository<Account>().Insert(account);
                xrmService.SaveChanges();
                return account.Id;
            }
        }

    }
```

AccountService klassis kirjeldatud tegevused mida tehakse kasutades Organization veebiteenust. Antud näites on kirjeldatud CreateAccount meetod, millele edastatakse soovitud konto nime. Meetodi sees toimub ühenduse loomine veebiteenusega, konto objekti loomine, ning selle salvestamine. Meetod tagastab konto unikaalset GUID.

5.7 Testide integreerimine TeamCity build serveriga

Testide käivitamine toimub olemasolevas TeamCity pideva integratsiooni serveris, mida kasutatakse ka arendajate kirjutatud koodi kompileerimiseks, pakkimiseks ja Dynamics CRMi uuendamiseks.

UI testide jaoks on eraldatud virtuaalmasin kus on installeeritud TeamCity agent ja Internet Explorer 11. Kuna Microsoft soovitab Dynamics CRMi jaoks kasutada Internet Explorerit ja see on ka firma sisene kokkulepe ei olnud vajadust teiste veebilehitsejate installeerimiseks. Testide koodi hoitaks Git repositooriumis.

Testid on üles ehitatud nii, et igas test klassis on üks test. Igal testil on NUniti atribuut Category selleks, et teste saaks jaotada ja käivitada kategooriate kaupa. Retry atribuut testi ebaõnnestumise korral laseb seda uuesti jooksutada.

Testid on jaotatud kahte kategooriasse: suitsutestid ja regressioonitestid. Suitsutestid annavad kiiret tagasisidet, veendumiseks, et süsteemi põhifunktsionaalsus töötab korrektsekt. Ajakulukad regressioonitestid, mis katavad süsteemi laiemat funktsionaalsust, on paigutatud eraldi kategooria alla. Selleks on TeamCity’s loodud vastavalt kaks konfiguratsiooni. Esimese kategooria *trigger*’iks on määratud ükskõik milline muudatus testkoodi repositooriumis või ükskõik milline muudatus Dynamics CRMi firmasiseses versioonijärgus. Seega suitsuteste käivitatakse kohe kui on lisatud või muudetud testide koodi või kui toimunud Dynamics

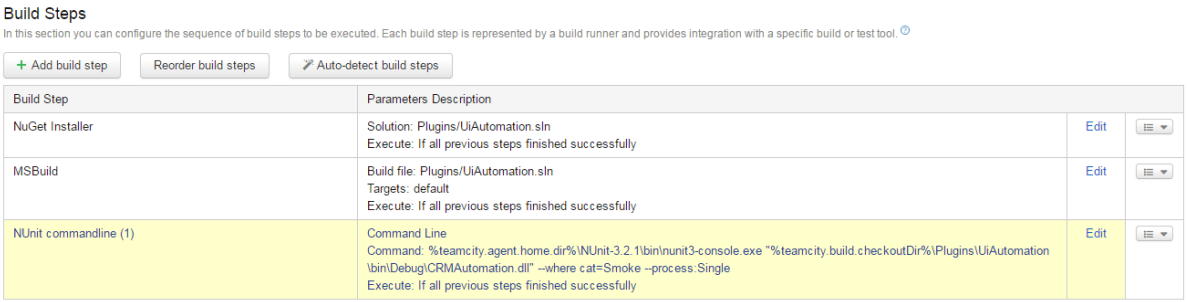
CRMi kohandamine ühest kolmest alampunktis 2.1 kirjeldatud viisist: kas muudatused *customization*'is, JavaScriptis või *plugin*'ides. Regressioonitestid käivitatakse samuti kui on toimunud üks eelpool nimetatud muudatustest, kuid vaid öösiti ja kindlatel kellaaegadel.

TeamCity konfiguratsioon koosneb kolmest saammust.

1. NuGet Installer – kontrollib et kõik vajalikud paketid nagu Selenium, NUnit, Internet Explorer draiver ja nende õiged versioonid oleksid virtuaalmasinasse installinud.

2. MSBuild – kompileerib repositooriumi viimast koodi *.dll* failiks

3. NUnit – hoolitseb testide käivitamise ja *xml* raporti genereerimise eest.



Joonis 13. TeamCity konfiguratsioon.

6. Kokkuvõte

Käesoleva bakalaureuse töö eesmärgiks oli UI testide implementeerimine MS Dynamics CRMi keskkonda. Antud töös analüüsiti rakenduse spetsiifilisi kitsaskohti automaattestide koostamise aspektist. Tutvustati tarkvara ja tööriistu millega on võimalik automaatteste kirjutada antud rakenduse jaoks, tutvustati projekti ülesehitust ja testkoodi ning selle projekti integreerimist olemasolevasse pideva integratsiooni süsteemi.

Lahenduse väljatöötamiseks kulus ligikaudu 3-4 nädalat, selle aja jooksul käis pidevalt raamistikku koodi täiendamine ja paralleelselt testide ümberkirjutamine. Praeguseks on valminud kokku ligikaudu 70 testi ja on avastatud mitmed regressioonivead.

Tänu Page Objecti disainimustrile testide kirjutamine muutus palju kiiremaks kuna saab taaskasutada juba eelnevalt kirjeldatud veebielemente. Lisaks, selle aja jooksul seoses uuendusega uuele Dynamics CRM versioonile muutus ühe kindla veebielemendi tüüpi loogika ning testide parandamiseks piisas muudatuse tegemisest vaid ühes kohas - raamistikku klassis.

Kasutajaliidese testide integreerimine Dynamics CRM'i API'ga oli teine suurem muudatus testide kirjutamise lähenemisel. Tänu sellele, et aega andmete ettevalmistamisel või rakenduse kindlasse seisundi viimisel kulub märkimisväärselt vähem, vähenes kogu aeg mis kulub testide jooksutamiseks ligikaudu poole võrra. Lisaks vähenes ebaõnnestunud testide arv, mis ei ole otseselt seotud veaga rakenduse funktsionaalsuses.

Töö tulemusena tehti kindlaks, et MS Dynamics CRM'i edasiarendust on võimalik testida kasutades enamlevinud automaattestimise raamistikku. Väljavalitud testjuhtudele tutvustati reaalset testide implementatsiooni, nende integreerimist koosteprotsessi süsteemi.

Töö käigus valminud lahenduse edaspidiseks täiendamiseks on plaanis uurida Page Object'ide automaatse genereerimise võimalust. Selleks, et kätte saada kõik elementide lokaatorid ja nende tüübid igal veebilehel saab kasutada Dynamics CRMi API'id ning seejärel teha skript Page Objektide genereerimiseks. Selleks, et testide tulemus oleks arusaadavam ja seda saaks ka äri poolele edastada on plaanis uurida täiendavaid testraportite genereerimise võimalusi.

Antud töö saab olla kasulikuks materjaliks teistele testijatele kes puutuvad kokku MS Dynamics CRM'iga ja soovivad süsteemi automatiseerida.

Summary

The goal of this bachelors thesis was to implement UI tests on top of MS Dynamics CRM software. In this work was done analysis of application-specific bottleneck in aspect of testing automation. Author introduced software and tools which are suitable for created automated tests for particular application, introduced project structure and testing code, project setup in continuous integration system.

For developing solution it was spent around 3-4 weeks, during this time there was continuous improvement of framework code and in parallel tests were rewritten. At the moment there is around 70 tests and multiple regression issues were found.

Due to Page Object pattern writing the tests became more faster because now it is possible to reuse previously described web elements. Additionally, during this time there was an update to the new version of Dynamics CRM and logic of one particular element type representation was changed. It was enough to change testing framework only in one place.

Integrating UI tests with Dynamics CRM API was second major change in tests writing approach. Due to the fact that data preparation or moving application to certain state takes less time, the whole time spent on running tests reduced in a half. Additionally, reduced number of failed tests which are not related to bug or functionality of application.

As a result it was determined that it is possible to test MS Dynamics CRM further development using a more common software testing framework. The selected test cases presented in real tests implementation, their integration into the assembly system.

To improve developed solution in future it is planned to investigate the possibility of Page Object's automatic generation. It should be possible by retrieving elements and their locators using Dynamics CRM API. Also it is planned to improve test reporting solution and automatic reports generation.

This work can be useful for the other testers who are exposed to MS Dynamics CRM and want to contribute in UI automation.

Kasutatud kirjandus

[1] D. Laney, „The Great Enterprise Balancing Act: Extended Relationship Management“ [Võrgumaterjal] http://blogs.gartner.com/doug-laney/files/2012/02/ad1074-The-Great-Enterprise-Balancing-Act-Extended-Relationship-Management-XRM.pdf [Kasutatud Mai 2017]

[2] „What is Agile Software Development and Agile Manifesto?“ [Võrgumaterjal] http://istqbexamcertification.com/what-is-agile-manifesto-and-agile-software-development/ [Kasutatud Mai 2017]

[3] „Is Automated Functional Testing Really Worth It?“ [Võrgumaterjal] http://blog.iseinc.biz/is-automated-functional-testing-really-worth-it [Kasutatud Mai 2017]

[4] M. Fowler „Test Pyramid concept“ [Võrgumaterjal] https://martinfowler.com/bliki/TestPyramid.html [Kasutatud Mai 2017]

[5] „Microsoft Dynamics CRM Authentication (On-premises)“ [Võrgumaterjal] https://community.dynamics.com/crm/b/webfortisblogunleashingyourcrm/archive/2015/02/24/microsoft-dynamics-crm-authentication-on-premises [Kasutatud Mai 2017]

[6] „NUnit documentation. Retry attribute“ [Võrgumaterjal] https://github.com/nunit/docs/wiki/Retry-Attribute [Kasutatud Mai 2017]

[7] „NUnit documentation. Category attribute“ [Võrgumaterjal] https://www.nunit.org/index.php?p=category&r=2.4.8 [Kasutatud Mai 2017]

[8] „Selenium documentation. Location strategies“ [Võrgumaterjal] http://www.seleniumhq.org/docs/06_test_design_considerations.jsp#location-strategies [Kasutatud Mai 2017]

[9] „Writing reliable locators for Selenium and WebDriver tests“ [Võrgumaterjal] https://blog.mozilla.org/webqa/2013/09/26/writing-reliable-locators-for-selenium-and-webdriver-tests/ [Kasutatud Mai 2017]

[10] „Microsoft documentation. Use the Xrm.Page object model“ [Võrgumaterjal] https://msdn.microsoft.com/en-us/library/gg328474.aspx [Kasutatud Mai 2017]

[11] „Selenium documentation. Test design considerations“ [Võrgumaterjal] http://www.seleniumhq.org/docs/06_test_design_considerations.jsp [Kasutatud Mai 2017]

[12] „Analysis and Design of Selenium WebDriver Automation Testing Framework“ [Võrgumaterjal] http://www.sciencedirect.com/science/article/pii/S1877050915005396 [Kasutatud Mai 2017]

[13] „Improving Test Suites Maintainability with the Page Object Pattern“ [Võrgumaterjal] http://softeng.disi.unige.it/publications/2013-leotta-ICSTW.pdf [Kasutatud Mai 2017]

[14] „Abstracting Selenium tests using Page Object model“ https://www.3pillarglobal.com/insights/abstracting-selenium-tests-using-page-object-model [Kasutatud Mai 2017]

[15] “Selenium Design Patterns and Best Practices“ D.Kovalenko, Packt Publishing, 2014 [Kasutatud Mai 2017]

[16] „Internet Explorer does not support user names and passwords in Web site addresses „ [Võrgumaterjal] https://support.microsoft.com/en-us/help/834489/internet-explorer-does-not-support-user-names-and-passwords-in-web-site-addresses-http-or-https-urls [Kasutatud Mai 2017]

[17] „Microsoft documentation. Supported extensions for Microsoft Dynamics 365“ [Võrgumaterjal] https://msdn.microsoft.com/en-us/library/gg328350.aspx [Kasutatud Mai 2017]

Lisa 1 – Konto lehe Page Object disainimuster

Siin on välja toodud vaid osa konto lehe PageObject klassist põhiliste meetoditega. Igat veebilehe elementi saab sellisel moel kirjeldada, täpsemalt mis tegevust saab selle elemendiga teha.

```
public class AccountPage : BasePage

    {
        public AccountPage(IWebDriver driver) : base(driver)
        {
        }

        internal void ClickNew()
        {
              driver.FindElement
              (By.Id("account|NoRelationship|HomePageGrid|Mscrm.HomepageGrid.accou
              nt.NewRecord")).Click();

        }

        internal void TypeName(string value)
        {
              driver.FindElement(By.Id("name")).Click();
              driver.FindElement(By.Id("name_i")).SendKeys(value);
        }

        internal void SelectType(string value)
        {
              driver.FindElement(By.Id("kn_accounttypeglobal ")).Click();
              new SelectElement(driver.FindElement(By.Id("kn_accounttypeglobal_i")))
              .SelectByText(value);

        }

        internal void TypeActivityPlan(string value)
        {
              driver.FindElement(By.Id("kn_activityplanid")).Click();
              driver.FindElement(By.Id("kn_activityplanid_i")).SendKeys(value);

        }
    }
```

Lisa 2 – Konto lehe Page Object ja DRY disainimuster

```
public class AccountPage : BasePage
    {
        public AccountPage(IWebDriver driver) : base(driver)
        {

        }

        internal void ClickNew()
        {
            SwitchToDefaultFrame();
            Click(By.Id("account|NoRelationship|HomePageGrid|Mscrm.HomepageGrid.accou
              nt.NewRecord"));
            WaitForFrameAndSwitch();
        }

        internal void TypeName(string value)
        {
            Type(By.Id("name"), value);
        }

        internal void SelectType(string value)
        {
            Select(By.Id("kn_accounttypeglobal"), value);
        }

        internal void TypeActivityPlan(string value)
        {
            TypeLookup(By.Id("kn_activityplanid"), value);
        }

    }
```

Lisa 3 – Baas Page klass

```
public class BasePage : Framework
{

    public BasePage(IWebDriver driver) : base(driver)
    {
    }

    //Sitemap Navigation
    public void NavigateSalesContacts()
    {
        SwitchToDefaultFrame();
        Click(By.Id("TabSFA"));
        Click(By.Id("nav_conts"));
    }

    public void NavigateSalesAccounts()
    {
        SwitchToDefaultFrame();
        Click(By.Id("TabSFA"));
        Click(By.Id("nav_accts"));
    }

    public void NavigateSalesActivities()
    {
        SwitchToDefaultFrame();
        Click(By.Id("TabSFA"));
        Click(By.Id("nav_activities"));
    }

    public void NavigateSalesOpportunities()
    {
        SwitchToDefaultFrame();
        Click(By.Id("TabSFA"));
        Click(By.Id("nav_oppts"));
    }

    //Direct navigation
    public void Navigate(String entity, String guid)
    {
        string urllogin = Settings.Default.ServerUrl + "main.aspx?etn=" + entity +
        "&pagetype=entityrecord&id=%7B" + guid + "%7D";
        driver.Navigate().GoToUrl(urllogin);
    }
```

Lisa 4 – Framework ja FrameworkWaiters klassid

Iga Baas Page Object klass kasutab Framework klassi meetodeid selleks et teha mingit tegevust konkreetsete veebielementidega.

```
public class Framework : FrameworkWaiters
{

    public Framework(IWebDriver driver) : base(driver)
    {
    }

    public IWebElement Type(By locator, string text)
    {
        string loc = locator.ToString();
        string bystr = loc.Substring(7) + "_i";

        if (loc.Substring(0, 5).Equals("By.Id"))
        {
            WaitForVisible(locator).Click();
            WaitForVisible(By.Id(bystr)).SendKeys(text);
            return WaitForVisible(By.Id(bystr));
        }
        else
        {
            WaitForVisible(locator).SendKeys(text);
            return WaitForVisible(locator);
        }

    }

    public IWebElement TypeLookup(By locator, string text)
    {
        string loc = locator.ToString();
        string bystr = loc.Substring(7) + "_ledit";

        if (loc.Substring(0, 5).Equals("By.Id"))
        {
            WaitForVisible(By.Id(locator)).Click();
            WaitForVisible(By.Id(locType)).SendKeys(text);
        }
        else
        {
            WaitForVisible(locator).SendKeys(text);
        }
        return WaitForVisible(By.Id(locType));
    }
```

```
    public void Select(By locator, string text)
    {
        string loc = locator.ToString();
        string bystr = loc.Substring(7) + "_i";

        if (loc.Substring(0, 5).Equals("By.Id"))
        {
            WaitForVisible(By.Id(locator)).Click();
            new SelectElement(WaitForVisible(By.Id(bystr))).SelectByText(text);
        }
        else
        {
            new SelectElement(driver.FindElement(locator)).SelectByText(text);
        }

    }
}
```

FrameworkWaiters on kõige madalama taseme klass mis kutsub otse Selenium Webdriver'i teeki meetodeid.

```
public class FrameworkWaiters
{
    protected IWebDriver driver;

    public FrameworkWaiters(IWebDriver driver)
    {
        this.driver = driver;
    }

    public IWebElement WaitForVisible(By locator)
    {
        var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(20));
        return wait.Until(ExpectedConditions.ElementIsVisible(locator));

    }

    public void WaitForFrameAndSwitch()
    {
        driver.SwitchTo().DefaultContent();
        var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(20));
        IWebElement frame =
        wait.Until(ExpectedConditions
          .ElementIsVisible(By.XPath("//iframe[contains(@style,' visible')]")));
        driver.SwitchTo().Frame(frame);

    }
```

```
    public void WaitForFrameAndSwitch(string framename)
    {
        string temp = framename;
        try
        {
            driver.SwitchTo().DefaultContent();
            var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(20));
            IWebElement frame =
            wait.Until(ExpectedConditions
               .ElementIsVisible(By.XPath("//iframe[contains(@style,' visible')]")));
            driver.SwitchTo().Frame(frame);



            var wait2 = new WebDriverWait(driver, TimeSpan.FromSeconds(20));
            var frames = driver.FindElements(By.XPath("//iframe[contains(@id,'" + temp
                + "')]"));
            for (int i = 0; i < frames.Count; i++)
            {
                if (frames[i].Displayed)
                {
                    driver.SwitchTo().Frame(frames[i]);
                    break;
                }

            }

        }
        catch (Exception e)
        {

        }

    }
}
```

Lisa 5 – SetupTeardown klass

Klass pärineb igast testi klassist. Enne testi alati käivitatakse eeloperatsiooni SetUp() meetod ja iga testi lõpus postoperatsiooni TearDown() meetod.

```
public class SetupTeardown
{
    protected IWebDriver driver;

    [SetUp]
    public void SetUp()
    {
        Process[] processlist = Process.GetProcesses();
        foreach (Process theprocess in processlist)
        {
            if (theprocess.ProcessName.Contains("iexplore") ||
                theprocess.ProcessName.Contains("iedriverserver"))
            {
                try
                {
                    theprocess.Kill();
                }
                catch (Exception e)
                {
                    Console.WriteLine("Exception: " + e);
                }

            }

        }
        DesiredCapabilities dc = DesiredCapabilities.InternetExplorer();
        dc.SetCapability("enablePersistentHover", false);
        dc.SetCapability("RequireWindowFocus", false);
        driver = new InternetExplorerDriver("C:/Selenium/");
        driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(5));
    }

    [TearDown]
    public void TearDown()
    {
        driver.Quit();

    }

    public void Login(string username)
    {
        string urllogin = "https://GER%5C" + username + ":" + User.Password + "@"
                                + Settings.Default.ServerUrl;
        driver.Navigate().GoToUrl(urllogin);
        driver.Manage().Window.Maximize();
        AcceptEmailNotification();

    }
}
```