გია სურგულაძე,
მიხეილ გულიტაშვილი,
ნინო კივილაძე

Web-აპლიკაციების ტესტირება, ვალიდაცია და ვერიფიკაცია

„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, მიხეილ გულიტაშვილი,
ნინო კივილაძე

Web-აპლიკაციების ტესტირება, ვალიდაცია და ვერიფიკაცია

დამტკიცებულია:
სტუ-ს „IT-კონსალტინგის”
სამეცნიერო ცენტრის
სარედაქციო-საგამომცემლო
კოლეგიის მიერ

თბილისი
2015

უაკ 004.5

განიხილება მართვის საინფორმაციო სისტემების პროგრამული უზრუნველყოფის შექმნის ბიზნესპროცესების ობიექტორიენტირებული ანალიზი, დაპროექტება, დეველოპმენტი, ტესტირება და დანერგვა. პროგრამული პროდუქტის ხარისხის სრულყოფის მიზნით განსაკუთრებით გამახვილებულია ყურადღება Web-სისტემების სასიცოცხლო ციკლის მენეჯმენტის საკითხებზე მათი ტესტირების, ვერიფიკაციის და ვალიდაციის საფუძველზე. წარმოდგენილია Web-აპლიკაციების ტრადიციული და ავტომატური ტესტირების არსებული მეთოდები და ინსტრუმენტული საშუალებები. შემოთავაზებულია მოდელები, მათი კლასიფიკაცია და რეალიზაციის გზები. შესაბამისი თვალსაზრისით რეალიზებულია პრაქტიკული ექსპერიმენტების შედეგები MsVisual Studio 2012/Selenium RC/WebDriver/AutoIT გარემოში.

მონოგრაფია ორიენტირებულია მართვის საინფორმაციო სისტემების სპეციალობის მაღალი კურსის ბაკალავრებზე, მაგისტრანტ-დოქტორანტებსა და პროგრამული უზრუნველყოფის ტესტირების საკითხებით დაინტერესებულ მკითხველზე.

რეცენზენტები:

- პროფ, გიორგი გოგიჩაიშვილი (საქ. მეცნიერებათა ეროვნული აკადემიის წევრ-კორესპოდენტი)
- პროფ. ეკატერინე თურქია
- პროფ. გელა ღვინეფაძე

პროფ. გ. სურგულაძის რედაქციით

ISBN 978-9941-0-7682-4

Georgian Technical University

Gia Surguladze, Mikheil Gulitashvili,
Nino Kiviladze

WEB-APPLICATIONS TESTING, VALIDATION AND VERIFICATION

Supported by DAAD (Germany)

The present book discusses Software testing, verification and validation problems of Management Information Systems (MIS) based on object-oriented methodology. Existing methods and tools of traditional and automatic testing of web applications are presented. Results of practical experiments are realized in the environment of Ms Visual Studio 2012/Selenium RC/WebDriver/AutoIT environment.

ISBN 978-9941-0-7682-4

ავტორთა შესახებ:

გია სურგულაძე - სტუ-ს პროფესორი, ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული „ინფორმატიზაციის საერთაშორისო აკადემიის (IIA)” ნამდვილი წევრი, სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრის“ ხელმძღვანელი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ბერლინის ჰუმბოლდტის, ნიურნბერგ-ერლანგენის და სხვა უნივერსიტეტების მიწვეული პროფესორი 1991-2014 წწ. 300-ზე მეტი სამეცნიერო ნაშრომის ავტორი, მათ შორის 60 წიგნის და 42 ელ-სახელმძღვანელოსი მართვის საინფორმაციო სისტემების ინჟინერინგის სფეროში.

მიხეილ გულიტაშვილი - სტუ-ს ასოც.პროფესორი, აკადემიური დოქტორი ინფორმაციული ტექნოლოგიების სპეციალობით. დაცული დისერტაციის თემა: „ორგანიზაციული მენეჯმენტის IT-კონსალტინგის მოდელების და მეთოდების დამუშავება და კვლევა სისტემების ინტეგრაციის მიზნით (პროგრამების ტესტირება)“. არის 15 სამეცნიერო ნაშრომის ავტორი და სამი საერთაშორისო კონფერენციის მონაწილე-მომხსენებელი.

ნინო კივილაძე - სტუ-ს დოქტორანტი ინფორმაციული ტექნოლოგიების სპეციალობით. მუშაობს დისერტაციაზე: „კორპორაციული Web-აპლიკაციების აგების ტექნოლოგიები სერვის--ორიენტირებული არქიტექტურით“. არის რამდენიმე რეალური პროექტის, ხუთი სამეცნიერო სტატიის ავტორი და ორი საერთაშორისო კონფერენციის მონაწილე- მომხსენებელი.

სარჩევი

გამოყენებული აბრევიატურები

API -	Application Programming Interface
AJAX -	Asynchronous JavaScript and XML
AU3 -	AutoIT V3
BDT -	Build Deploy Test
BPM -	Business Process Management
CSS -	Cascading Style Sheets
COM -	Component Object Model
CASE -	Computer Aided System Engineering
DOM -	Document Object Model
DLL -	Dynamic Link Library
EJB3 -	Enterprise JavaBeans
XAML -	Extensible Application Markup Language
XML -	Extensible Markup Language
GUI -	Graphical User Interface
HR -	Human Resources
IDE -	Integrated development environment
IE -	Internet Explorer
JAR -	Java Archive
JDBC -	Java Datadabse Connectivity
JDK -	Java Development Kit
JRE -	Java Runtime Environment
JVM -	Java virtual machine
JS -	JavaScript
RC -	Remote Control
SQL -	Structured Query Language
TFS -	Team Foundation Server
URL -	Uniform Resource Locator

შესავალი

ორგანიზაციული მენეჯმენტის ბიზნეს-პროცესების მხარდაჭერა ინფორმაციული ტექნოლოგიებით IT-კონსალტინგის ერთერთი ძირითადი ამოცანაა, რომელიც სისტემის ინტეგრაციის მიზნით იხილავს IT-სერვისების დაგეგმვის, დაპროექტების, დეველოპმენტის, ტესტირების და დანერგვის შესაძლებლობებს, შესაბამისი ხარჯების ოპტიმიზაციით, ბიზნეს-პროცესების ეფექტურობის, მართვადობის და გამჭვირვალობის ამაღლებით, აგრეთვე ერთიანი IT-ინფრასტრუქტურის შექმნით. თანამედროვე საინფორმაციო ტექნოლოგიები გვთავაზობს ბიზნეს–პროცესების მოდელირების, დაპროექტების, პროგრამული რეალიზაციის და ტესტირების უნიფიცირებულ (UML) და მოქნილ (Agile) მეთოდებს და ინსტრუმენტულ (CASE) საშუალებებს [1-4].

ობიექტორიენტირებული მოდელირების მიდგომამ და დაპროგრამების ახალმა პარადიგმებმა საფუძველი ჩაუყარა პროგრამული სისტემების სასიცოცხლო ციკლის პროცესების სრულყოფას. სისტემების შექმნის და/ან გაფართოების მიზნით მნიშვნელოვანი საკითხია ინტეგრაციის პროცესების მართვა [1].

IT-კონსალტინგი განსაკუთრებულ ყურადღებას უთმობს ტესტირების ახალ მეთოდოლოგიებს, ვინაიდან IT-სერვისების (ან ბიზნესპროცესების პროგრამული მოდულების) დამუშავების პროცესში ტესტირება და შეფასების განსაზღვრა მეტად მნიშვნელოვანია. აქ წყდება საკითხი დამუშავებული სერვისების საბოლოო გამოყენების ან მათი შემდეგი ვერსიის შექმნის შესახებ, რაც კარგად აისახება პროგრამის სასიცოცხლო ციკლის იტერაციულ-ინკრემენტალურ მოდელზე.

მაგალითად, ავტომატური Regress და Unit ტესტირების მეთოდოლოგია პროგრამული სისტემების შექმნის ავტომატურ პრინციპებზეა აგებული [5]. ავტომატური ტესტირება გამოიყენება დიდი და საშუალო პროგრამული პროექტების ვალიდაცია–ვერიფიკაციისთვის, იგი ეფუძნება პროგრამული კოდით მატესტირებელი სცენარების შექმნას და დამხმარე ინსტრუმენტებით არსებული აპლიკაციის დამოწმებას. Agile მეთოდოლოგიის მიმდევრები (მაგალითად, ექსტრემალური პროგრამირება), Scrum, Rational Unified Process (RUP), Dynamic Systems Development Method (DSDM) და სხვ.) აქცენტს მაღალორგანიზებულ გუნდური პროგრამირების პრინციპებზე აკეთებენ, ცდილობენ გამორიცხონ პროგრამული ხარვეზების დაშვება განპირობებული ადამიანური ფაქტორებით [5-21].

თანამედროვე საინფორმაციო ტექნოლოგიებში ავტომატური ტესტირება ახალი დარგია და იგი დღითი–დღე ხდება პოპულარული. ტესტირება არის პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის შემადგენელი აუცილებელი ეტაპი, რომელიც პირდაპირ განსაზღვრავს აპლიკაციის ხარისხს.

ამერიკისა და ევროპის უნივერსიტეტებსა და Software-ის საპროექტო ინსტიტუტებში, დიდი სერიოზული პროექტები მოწმდება ავტომატური მატესტირებელი ტექნოლოგიის პრინციპებით. საქართველოში ეს მიმართულება დამკვიდრების სტადიაშია, თუმცა ბევრი მოწინავე საჯარო და კერძო სტრუქტურის ორგანიზაციები უკვე აქტიურად იყენებს ტესტირების ავტომატიზაციის საშუალებებს [5,28].

პროგრამულ ინჟინერიაში საინფორმაციო სისტემების დაპროექტების, რეალიზაციასა და შემოწმების თვალსაზრისით, ავტომატური ტესტირება იქცა უნივერსალურ მეთოდოლოგიად, რომელიც განკუთვნილია როგორც პროგრამული სისტემების ტესტერებისთვის, ისე ბიზნეს-ანალიტიკოსებისთვის და პროგრამისტებისთვის. ამ ტექნოლოგიის აქტიურმა გამოყენებამ აუცილებელი გახადა ავტომატური ტესტირების ინსტრუმენტების განვითარება და თანამედროვე ტექნოლოგიებთან თავსებადი ვერსიების შექმნა.

წიგნში განიხილება ხელით ტესტირების სკითხიც, მისი შედარება ავტომატურ ტესტირებასთან, გამოყენების სფეროები, ნაჩვენებია მათი მოხმარების ხელსაყრელი არეები. ყოველივე ზემოთქმული მეტყველებს თემის აქტუალურობასა და მის მნიშვნელობაზე.

ნაშრომის მიზანია ორგანიზაციული მართვის სისტემების გამოყენებითი პროგრამული აპლიკაციების ავტომატური ტესტირების საკითხების კვლევა, მისი კავშირი სისტემების ინტეგრაციის პრობლემებთან. ობიექტ–ორიენტირებული დაპროგრამების ახალი ტექნოლოგიებით მანქანური ვალიდაციების, ფასიანი და უფასო მატესტირებელი სისტემების გაცნობა, Selenium ტექნოლოგიის და მისი შემადგენელი ინსტრუმენტების მიმოხილვა (6-22).

დასმული მიზნის მისაღწევად განიხილება შემდეგი ძირითადი ამოცანები:

- ორგანიზაციული მართვის სრულყოფის თვალსაზრისით საინფორმაციო სისტემების ტესტირების პროცესის დაპროექტების და მოდელირების თანამედროვე მეთოდებისა და ინსტრუმენტული

საშუალებების ანალიზი, მათი კლასიფიკაცია ბიზნესპროცესების ასახვის შესაძლებლობათა ევოლუციის მიხედვით;

- ბიზნეს-პროცესების IT-სერვისების სადემონსტრაციო მატესტირებელი ტესტ-სკრიპტების ჩაწერა ავტომატურ რეჟიმში თანამედროვე ტექნოლოგიების ბაზაზე;

- ჩაწერილი ტესტ-სკრიპტების იმპორტ-ექსპორტი სხვადასხვა პროგრამულ ენებზე და შესრულებაზე გაშვებისას წარმოქმნილი პრობლემები, მატესტირებელი სისტემის ნაკლოვანებების გამოააშკარავება ტექნოლოგიური შეუთავსებლობის პრინციპებით;

- მატესტირებელი სერვერის ტექნოლოგიური განხილვა, მისი სტრუქტურის მოდელირება, ტესტირების რეპორტები და ბიზნეს აპლიკაციის ავტომატური ტესტირების ანალიზი;

- პარალელური ტესტირების საჭიროება, მისი ტექნიკური რეალიზება სადემონსტრაციო ამოცანით. მიმდევრობითი ტესტირების შედარება პარალელურ ტესტირებასთან შესრულების დროის ოპტიმიზების მიზნით;

- ტესტ-ქეისების, ტესტ-სკრიპტების, სცენარების და ტესტირების ძირითადი ტერმინების განმარტება, მათი რეალიზება სხვადასხვა დაპროგრამების ენებით და სკრიპტებით;

- SilverLight ტექნოლოგიის მატესტირებელი plugin-ი .NET პლატფორმაზე, ბიზნესაპლიკაციის ტესტირების შედეგების ანალიზი ვიდეო ჩანაწერით, Screenshot-ით და პროგრამული კოდით.

კვლევის ობიექტის სახით ნაშრომში განიხილება ინფორმაციული და პროგრამული უზრუნველყოფის ინფრასტრუქტურა და პროგრამული რეალიზაცია. კვლევის მეთოდებად კი განიხილებოდა ავტომატური ტესტირების მეთოდი, ობიექტ-

ორიენტირებული დაპროგრამების, სკრიპტინგ ენების ტესტირების მეთოდები, მიმდევრობითი და პარალელური ტესტირების პროცესები და მათი რეალიზაციის ინსტრუმენტები. საინფორმაციო სისტემების ტესტირების კლასიფიკაცია და მათი გამოყენების თეორიული არეები, ბიზნესაპლიკაციათა ტესტირების ახალი ტექნოლოგიები, სცენარების ჩაწერის პროგრამული ინსტრუმენტები.

ნაშრომის ორიგინალობა მდგომარეობს ორგანიზაციული მენეჯმენტის სისტემების ინტეგრაციის მიზნით IT-კონსალტინგის მოდელების და მეთოდების შემუშავებასა და ბიზნესპროცესების (IT-სერვისების) პროგრამული სისტემების სასიცოცხლო ციკლის მართვის საკითხების კვლევის ჩატარებაში. კერძოდ განტერის იტერაციულ-ინკრემენტალური „ფაზა-ფუნქციების“ მოდელის საფუძველზე შესწავლილ იქნა IT-სერვისების ტესტირების შედეგების დამოკიდებულება ინტეგრაციის პროცესებზე.

– განხორციელდა მსგავსი საპრობლემო სფეროების ტესტირების პრობლემების და ამოცანების ჩამოყალიბება, საინფორმაციო სისტემების კლასიფიკაცია და დაპროექტების და მოდელირების შესაბამისი ტექნოლოგიების განსაზღვრა მათ გადასაწყვეტად;

ნაშრომის შედეგებს აქვს პრაქტიკული ღირებულება, ვინაიდან ის შეიძლება გამოყენებულ იქნას სხვადასხვა დარგებში ბიზნესაპლიკაციების ტესტირების, ვერიფიკაციის, ვალიდაციის და პროგრამული რეალიზაციის ამოცანების გადასაწყვეტად ახალი ინფორმაციული ტექნოლოგიებით.

წიგნის პირველი თავი ეხება პროგრამული უზრუნველყოფის ტესტირების თანამედროვე ტექნოლოგიების მიმოხილვას. გადმოცემულია ტესტირების, ვალიდაციის და ვერიფიკაციის პროცესების ძირითადი ცნებები, Web აპლიკაციების ავტომატური ტესტირების არსი. ავტომატური ტესტირების წარმოშობის ისტორია და განვითარება. აღწერილია ხელით ტესტირების შედარება ავტომატურ ტესტირებასთან, მათი უარყოფითი და დადებითი მხარეები. განხილულია ტესტირების ძირითადი პრინციპები და კონცეფციები. შემოთავაზებულია პროგრამული ტესტირების კლასიფიკაცია და მოკლედ მიმოხილულია ძირითადი ტესტირების ტიპები. განხილულია ბაგის წარმოშობის ისტორია და აღწერილია მისი არსებობის გამო წარმოქმნილი წინააღმდეგობები.

ბოლოს, დასმულია ამოცანა ორგანიზაციული მართვის საინფორმაციო სისტემებში ავტომატური ტესტირების დანერგვის და გამოყენების პროცესების შემდგომი სრულყოფის შესახებ.

მეორე თავში განიხილება ორგანიზაციული მართვის სისტემების პროგრამული უზრუნველყოფის აგების თანამედროვე ინსტრუმენტული საშუალებები. კერძოდ „მაიკროსოფთის“ ფირმის Web-აპლიკაციების (სისტემების) ASP.NET, ASP.Silverlight და ASP MVC ტექნოლოგიები VisualStudio.NET, C# და SQL Server პაკეტების ბაზაზე. განიხილება მათი მახასიათებლები, შესაძლებლობები და განვითარების ტენდენციები. შემოთავაზებულია კონკრეტული Web-სისტემების რეალიზაციის მაგალითები.

მესამე თავში ფართოდაა მიმოხილული Web-აპლიკაციების ავტომატური მატესტირებელი სისტემა Selenium, მისი შემადგენელი კომპონონტები და მატესტირებელი ინსტრუმენტები. განხილულია

მატესტირებელი სცენარის ჩაწერა, ტესტ-სკრიპტების და ტესტ-სცენარების შექმნა მოცემული ტექნოლოგიით. აღწერილია ტესტირების ძირითადი ქმედებების, ვალიდაციის და ვერიფიკაციის ბრძანებები. გამოკვლეულია AJAX ტექნოლოგიით შექმნილი აპლიკაციების ტესტირებისას წარმოქმნილი პრობლემები ავტომატური ტესტის შესრულების თვალსაზრისით. მიმოხილულია აპლიკაციის ელემენტებზე წვდომის გზები – ლოკატორები, მათი კლასიფიკაცია და ერთმანეთთან შედარება ტესტის სწრაფად შესრულება – სტაბილურობის თვალსაზრისით. განხილულია ავტომატურ ტესტებში სკრიპტინგ ენებისა და ობიექტორიენტირებული ენების გამოყენება და მათი უპირატესობები და ა.შ.

მეოთხე თავში გადმოცემულია Selenium Grid ტექნოლოგია სერვისების პარალელური ავტომატური ტესტირებისთვის [18]. ტესტირების ავტომატიზაციის საკითხები Microsoft Test Manager, Visual Studio 2012, Selenium თანამედროვე პროგრამული ინსტრუმენტით. მოცემულია Java-ს მატესტირებელი „ფრეიმ-ვორკები“ – JUnit და TestNG. წარმოდგენილია ავტომატური ტესტირების შედეგების აღწერა, რეპორტ-ფაილების გენერირება. წარმოდგენილია Desktop აპლიკაციების მატესტირებელი ინსტრუმენტი AutoIT და მისი სკრიპტინგ ენა, დემონსტრირებულია ავტომატური სცენარები პრაქტიკული მაგალითებით.

მეხუთე თავი ეხება ორგანიზაციული მართვის სისტემების დაპროექტების და აგების ობიექტ-ორიენტირებული მეთოდების გამოყენების აღწერას. კერძოდ, უნივერსიტეტის მართვის საინფორმაციო სისტემის მონაცემთა ბაზის და მომხმარებელთა ინტერფეისების პროგრამული უზრუნველყოფის რეალიზაციის და

ტესტირების ამოცანების განხილვას „მაიკროსოფთის“ ახალი WPF, WF და WCF ტექნოლოგიების საფუძველზე.

ავტორები გამოთქვამენ იმედს, რომ წიგნის მკითხველი ობიექტურად შეაფასებს მასში განხილულ საკითხებს და მიღებულ შედეგებს. აღნიშნული სამეცნიერო-ტექნიკური მიმართულება, რომელიც პროგრამული უზრუნველყოფის ხარისხის მართვის საკითხებთან იკვეთება, საქართველოში შედარებით ახალია და მოითხოვს როგორც თეორიულ, ასევე პრაქტიკულ გამოცდილებას.

წიგნი არაა დაზღვეული ტერმინოლოგიური თუ შინაარსობრივი უზუსტობებისგან, ამიტომ წინასწარ გიხდით ბოდიშს და მადლობელი ვიქნებით ჩვენი გულისხმიერი მკითხველი კოლეგების, თუ მოგვაწვდიან საქმიან შენიშვნებს და რჩევებს.

- საკონტაქტო ელ-მისამართები:
  - g.surguladze@gtu.ge
  - mikheil.gulitashvili@gmail.com
  - nino.kiviladze@gmail.com

I თავი
გამოყენებითი პროგრამული სისტემების ტესტირების თანამედროვე ინფორმაციული ტექნოლოგიები

1.1 ბიზნესპროცესების ინფორმაციული მხარდაჭერა და IT-სერვისების ტესტირების ძირითადი ცნებები

თანამედროვე ინფორმაციული ტექნოლოგიების ბაზარი გამოირჩევა პროგრამული პროდუქტების დიდი სიუხვით. პროგრამული ინდუსტრია, მისი დეველოპერული კომპანიები ქმნის სხადასხვა სფეროსთვის საჭირო საინფორმაციო სისტემების პროგრამულ უზრუნველყოფებს (IT-სერვისებს), რომლებიც ეფექ-ტურად გამოიყენება მათ პრაქტიკულ საქმიანობაში.

რა არის პროგრამული ინსტრუმენტი და რატომ არის ის ასეთი საჭირო ?

ამ საკითხის გადაწყვეტა IT-კონსალტინგის ფუნქციაა. სახელმწუფო ორგანიზაციები და ბიზნესის კერძო სტრუქტურები თითქმის მთლიანად კომპიუტერიზებულია. ინფორმაციული ტექნოლოგიების ბაზარზე ძალზე მნიშვნელოვანია საუკეთესო პროგრამული პროდუქტის მოპოვება, მაგრამ ეს არც ისე მარტივია. არსებობს მრავალი სხვადასხვა ხელმისაწვდომი პროგრამა, რომელთაგანაც საუკეთესო აპლიკაციებს აფასებენ ხარისხით [15].

ორგანიზაციაში IT-კონსალტინგმა უნდა განსაზღვროს ასეთი საკითხები ბიზნეს-პროცესების ეფექტური მართვის მხარდაჭერის თვალსაზრისით.

პროგრამული პაკეტების ტესტირება კი არის ის პროცესი, რომელიც იკვლევს აპლიკაციის ფუნქციებს, რათა მივიღოთ ინფორმაცია პროდუქტის ხარისხზე.

პროგრამის ტესტირება გვიჩვენებს სოფტის ობიექტურ სახეს. ტესტირება მოიცავს პროგრამული აპლიკაციის შესრულებაზე

გაშვების პროცესს, რომელიც გამიზნულია იპოვნოს პროგრამული შეცდომა ან დეფექტი.

შევნიშნოთ, რომ კარგი ტესტი არის ის, რომელსაც აქვს მაღალი შესაძლებლობა, რომ იპოვნოს აღმოუჩენელი შეცდომა [16].

პროგრამული პაკეტების ტესტირება შეიძლება კიდევ განვსაზღვროთ როგორც პროცესი, რომლის დროსაც მტკიცდება (ვალიდაცია) და მოწმდება (ვერიფიკაცია), რომ კომპიუტერული პროგრამა (აპლიკაცია):

- არის მოცემული მოთხოვნების შესაბამისი;
- მუშაობს აღწერის შესაბამისად;
- შესაძლებელია შევიდეს ხმარებაში;
- აკმაყოფილებს დაინტერესებული მხარის მოთხოვნებს

რას ნიშნავს პროგრამის **ვერიფიკაცია** და **ვალიდაცია** ?

ვალიდაცია არის პროცესი, რომლის დროსაც მოწმდება რომ პროდუქტის დიზაინი აკმაყოფილებს, ეთანადება დასმულ მოთხოვნებს, ანუ პროგრამული უზრუნველყოფა ეთანადება მომხმარებლის მოთხოვნებს.

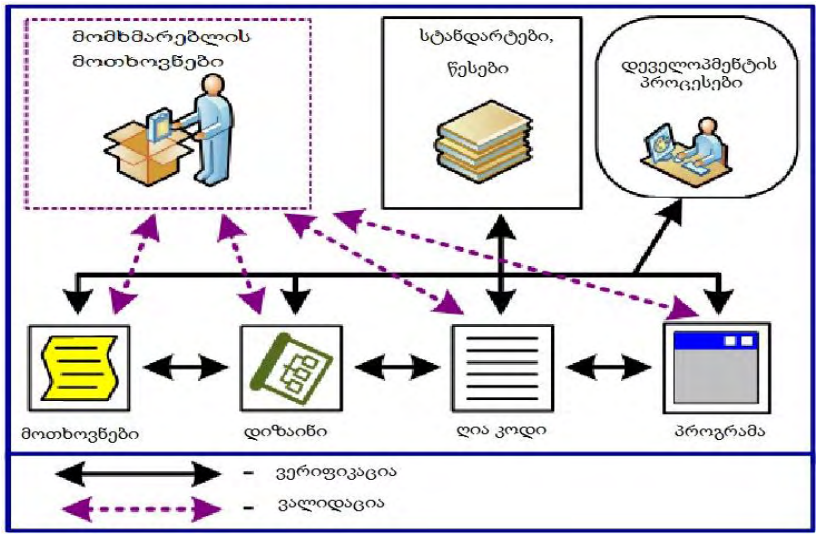
ვერიფიკაცია არის პროცესი, რომლის დროსაც პროგრამული უზრუნველყოფა მოწმდება შექმნის პროცესში, ანუ აკმაყოფილებს თუ არა წინასწარ განსაზღვრულ სტანდარტულ მოთხოვნებს.

ვერიფიკაცია და ვალიდაცია არ არის ერთიდაიგივე, ისინი ხშირად ერევათ ერთმანეთში. მათ შორის განსხვავებაა [11]:

- ვალიდაცია: ვქმნით ჩვენ სწორ პროდუქტს?
- ვერიფიკაცია: ვქმნით ჩვენ პროდუქტს სწორად?

სხვა სიტყვებით რომ ვთქვათ ვალიდაცია უზრუნველყოფს, რომ პროდუქტი ეთანადებოდეს მომხმარებლის მოთხოვნებს. ვერიფიკაცია კი უზრონველყოფს, რომ პროდუქტი შეიქმნას მოთხოვნების შესაბამისად. ვალიდაცია უზრუნველყოფს, რომ

„თქვენ შექმენით სწორი აპლიკაცია", ვერიფიკაცია კი უზრუნველყოფს რომ, „თქვენ შექმენით აპლიკაცია სწორად". საილუსტრაციოდ იხ ნახ.1.1 [9-16].

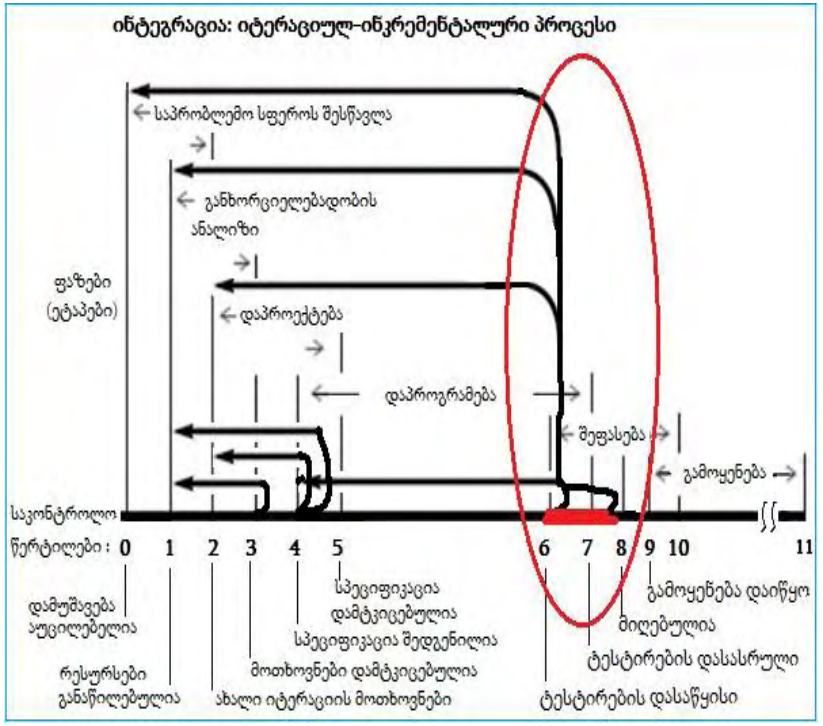


ნახ.1.1. ვერიფიკაცია და ვალიდაცია

პროგრამული უზრუნველყოფის შექმნის პროცესის მართვა მისი სასიცოცხლო ციკლის საფუძველზე, მათ შორის ტესტირების ეტაპიც (საკონტროლო წერტილი იხ. ნახ.1.2) უნდა გამოვიყენოთ ყოველთვის, საიმედო პროგრამის მისაღებად.

ტესტირებას არ შეუძლია გამოავლინოს პროგრამის ყველა დეფექტი. მისი ძირითადი მიზანია, რომ გამოავლინოს პროგრა-მული შეცდომები, რათა მოხდეს აღმოჩენილი დეფექტების გასწორება. პროგრამული კოდის ტესტირებისას, რომლის დროსაც მისი გაშვება ხდება სხვადასხვა გარემოში, მოწმდება პროგრამული კოდის პირობები: აკეთებს ის იმას, რაც არის წინასწარ განსაზღვრული რომ გააკეთოს ? რა რესურსები სჭირდება მას ამის

გასაკეთებლად და ა. შ. თანამედროვე ეპოქაში პროგრამული უზრუნველყოფის შექმნის პროცესში, ტესტირების ორგანიზება შეიძლება განცალკევებული იყოს პროგრამირების ჯგუფისგან, ე.წ. ტესტირების ჯგუფი. განარჩევენ რამდენიმე როლს ტესტირების ჯგუფში: პროექტის მენეჯერი, ჯგუფის ლიდერი, ტესტერი და სხვ.



ნახ.1.2. პროგრამული სისტემის სასიცოცხლო ციკლი ტესტირებით

ყველა პროგრამულ პროდუქტს აქვს გამოყენების შესაბამისი სფერო. მაგალითად, ვიდეო თამაშების პროგრამული უზრუნველ-ყოფის სფერო არის მთლიანად განსხვავებული საფინანსო ბანკის

პროგრამის გამოყენების სფეროსგან. მაშასადამე, როდესაც კომპანია ქმნის ან ინვესტიციას დებს პროგრამულ პროდუქტში, იგი შეფასდება, თუ როგორ მოერგება მომხმარებლების მოთხოვნებს, თუ იქნება მისაღები პროგრამის გამოყენების სფეროსთვის, მისი მყიდველებისთვის და დაინტერესებული პირებისთვის.

პროგრამების ტესტირება არის სწორედ ამ კრიტერიუმებით შეფასების პროცესი [16]. ზემოაღნიშნულის გარდა არსებობს ტესტირების განსხვავებული განსაზღვრებები. მაგალითად, ტესტირება არის პროცესი, როდესაც მოწმდება კომპიუტერული პროგრამის სისწორე, სისრულე და ხარისხი.

ტესტირება არის „კომპიუტერული პროდუქტის დაკითხვის პროცესი იმისთვის, რომ შეფასდეს იგი". ტესტერი კითხვებს უსვამს აპლიკაციას და პროგრამა პასუხობს რაღაც რეაქციით [5].

კომპიუტერული პროგრამის ტესტირებისას ხარისხის განმსაზღვრელი ატრიბუტებია:

- საიმედოობა;
- სტაბილურობა;
- შენარჩუნებადობა.

კომპიუტერული აპლიკაცია შეიძლება იყოს მცდარი, რადგან იგი შექმნილია ადამიანის მიერ. ამიტომ, პროგრამის შექმნის თანმხლები პროცესი უნდა იყოს ხარისხის კონტროლი.

პროგრამისტი, საშუალოდ ხარჯავს თავისი დროის 40% ტესტირებაზე და დანარჩენ დროს ახმარს პროგრამის შექმნას. თუმცა არსებობს ისეთი სპეციალური პროგრამები, რომელთა ტესტირებაც 3-5 ჯერ ძვირი ჯდება ვიდრე მათი შექმნა. მაგალითად სიცოცხლისთვის საშიში პროგრამები (ფრენის კონტროლი, რეაქტორის მონიტორინგი, სამედიცინო აპარატურა), რომელთა ტესტირებაც გაცილებით მეტ რესურსს მოითხოვს ვიდრე სხვა მოქმედებები.

1.2. ტესტირების პრინციპები

ქვემოთ მოცემულია ტესტირების ძირითადი პრინციპები:

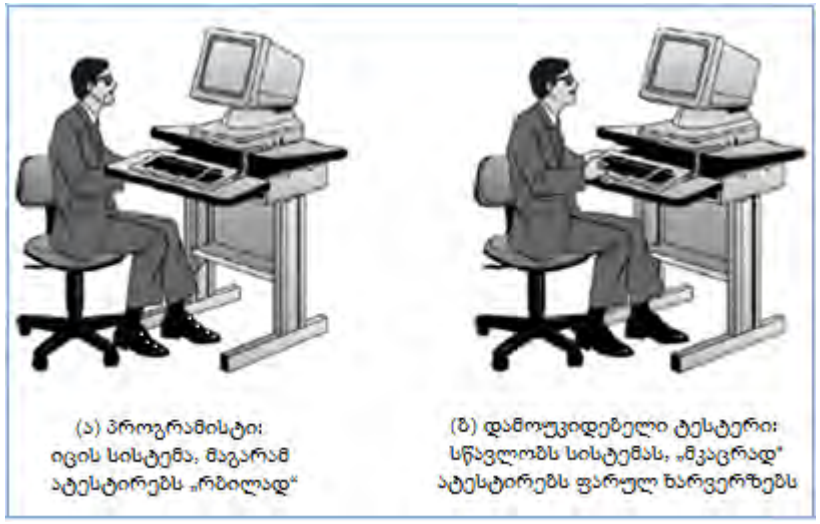
1. *ყველა ტესტი უნდა იყოს მომხმარებლის მოთხოვნების შესაბამისი.* ეს ნიშნავს, რომ გამოააშკარაოს ნებისმიერი დეფექტი, რომელიც შეიძლება წარმოქმნილ იქნას მომხმარებლის მოთხოვნების შესრულების დროს.

2. *ტესტი უნდა დაიგეგმოს ტესტირების დაწყებამდე დიდი ხნით ადრე.* მოთხოვნების მოდელის დასრულების შემდეგ შეიძლება დაიწყოს ტესტირების დაგეგმვა. დეტალური ტესტქეისები შეიძლება დაიწეროს როგორც კი დიზაინის მოდელი დასრულდება.

3. *ტესტირება სასურველია დაიწყოს მარტივი მოდულით და პროგრესირდეს კომპლექსური მოდულისკენ.* პირველი დაგეგმილი და გაშვებული ტესტი ზოგადად ფოკუსირებულია ინდივიდუალურ კომპონენტებზე. როგორც კი ტესტი პროგრესირდება, შეცდომების პოვნის ფოკუსირება გადადის მრავალ კომპლექსურ კომპონენტებზე და მათ შორის კავშირებზე [5-18].

4. *ამომწურავი ტესტირება შეუძლებელია.* პროგრამის ცვლილების ალბათობა საშუალო ზომის პროგრამებში განსაკუთრებით დიდია. ამ მიზეზის გამო შეუძლებელია, რომ გავუშვათ ყველა კომბინაციის ტესტი. მიუხედავად ამისა შესაძლებელია, რომ გამოვიკვლიოთ პროგრამული ლოგიკა და დავრწმუნდეთ რომ ყველა პირობა სრულდება პროგრამულ დონეზე.

5. *ეფექტუროზისთვის ტესტი უნდა იყოს დამოუკიდებელი.* პროგრამული ინჟინერი, რომელმაც შექმნა სისიტემა, არ არის საუკეთესო შემმოწმებელი (ნახ.1.3).



ნახ.1.3. ტესტის დამოუკიდებლობა

1.3. WEB აპლიკაციების ტესტირება

Web-ზე ბაზირებული პროგრამული აპლიკაციების გამოყენება დღეისათვის გახდა კომპიუტერის მომხმარებლების ცხოვრების ნაწილი. ყოველი ჩვენგანი იყენებს ინტერნეტ ბრაუზერს, რათა შევამოწმოთ ელექტრონული ფოსტა, წავიკითხოთ ახალი ამბები, დავრეგისტრირდეთ სოციალურ ქსელში, მოვიძიოთ ჩვენთვის სასურველი ინფორმაცია და ა. შ. ჩვენ ისე თავისუფლად ვიყენებთ სიტყვას „გოოგლე" როგორც ნებისმიერ ზმნას. გამოვხატავთ ჩვენ დამოკიდებულებას ბრაუზერების მიმართ და ვმსჯელობთ მათ უპირატესობებზე [6].

დღესდღეობით არსებული პროგრამული უზრუნველყოფის უდიდესი ნაწილი არის Web-ზე ბაზირებული პროგრამული პროდუქტი, რომელთაც მოვიხმართ ინტერნეტ ბრაუზერების

საშუალებით. ეპოქაში, სადაც არის მაღალი საპასუხისმგებლო პროგრამული უზრუნველყოფის პროცესები, სადაც მრავალი ორგანიზაციები იყენებს UML და Agile მეთოდოლოგიებს, ტესტირების ავტომატიზაცია ხდება დღითიდღე პოპულარული [3,4,13,14,21,27,28,30].

პროგრამული პროდუქტის ეფექტურ ტესტირებას დიდ მნიშვნელობას ანიჭებენ კომპანიები და ორგანიზაციები. პროგრამული უზრუნველყოფის ტესტირების ავტომატიზაცია ნიშნავს გამოიყენო სხვა პროგრამული ინსტრუმენტი, რომლის საშუალებითაც მოახდენ სატესტირებელი პროგრამის მრავალჯერად ტესტირებას. ტესტირების ავტომატიზაციას გააჩნია სხვადასხვა უპირატესობები, რომელთაგანაც აღსანიშნავია მისი **განმეორებადობა და სიჩქარე**, რომელიც საჭიროა ტესტის გასაშვებად [8].

ტესტირების ავტომატიზაცია არის პროგრამული პროდუქტის შექმნის შემადგენელი ეტაპი, რომელიც მოიცავს:

- განმეორებად ტესტირებას;
- სწრაფ უკუკავშირს პროგრამისტ-დეველოპერებთან;
- ტესტ-ქეისების შესრულებაზე გაშვებას შეუზღუდავად;
- Agile და XP მეთოდოლოგიების მხარდაჭერას;
- კარგად ორგანიზებულ ტესტ-ქეისების დოკუმენტაციას;
- შეცდომების აღწერას;
- ხელით ტესტირებისას გამორჩენილი შეცდომების პოვნას.

ცხადია, რომ თუ მოვახდენთ ჩვენი პროგრამული პროდუქტის ავტომატურ ტესტირებას, ეს იქნება გარანტი შეცდომების თავიდან აცილების, პროგრამის საიმედოობის, ტესტირების დროის დაზოგვის, პროცესის დაჩქარების. მაგრამ პროგრამების ავტომატური ტესტირება ყოველთვის არ არის მიზანშეწონილი.

მაგალითად, წარმოვიდგინოთ პროგრამული პროდუქტი, რომლის სამომხმარებლო ინტერფეისი ხშირად იცვლება, ამ

შემთხვევაში მატესტირებელი სკრიპტების ცვლილება გარდაუვალია. ასევე, როცა დასატესტირებელი პროგრამა პატარა და მარტივია, ამ შემთხვევებში ხელით ტესტირება უფრო მისაღებია, ვიდრე მისი ავტომატიზაცია.

თუ აპლიკაცია სრულდება მოკლე ვადებში, მაშინ ხელით ტესტირება საუკეთესო გადაწყვეტილებაა, რადგან ტესტირების ავტომატიზაციას სჭირდება მეტი დრო [18].

თანამედროვე ინფორმაციულ ტექნოლოგიებში გავრცელებულია მრავალი კომერციული და უფასო პროგრამული ინსტრუმენტი, რომელთა საშუალებითაც შესაძლებელია ავტომატური ტესტირების განხორციელება [22-26].

პაკეტი Selenium შესაძლოა განვიხილოთ როგორც ყველაზე უფრო ფართოდ გავრცელებული და გამოყენებადი უფასო ტესტირების ინსტრუმენტი. იგი შედგება სხვადასხვა პროგრამული პროდუქტისგან, თვითოეული მათგანი თავისი ფუნქციებით უზრუნველყოფს ტესტირების ავტომატიზაციას. მისი პროგრამული ინსტრუმენტების სიმრავლე საშუალებას გვაძლევს, რომ დავატესტიროთ Web-აპლიკაციები ჩვენი მოთხოვნების შესაბამისად.

ტესტირების ავტომატიზაციისას მრავალი სირთულეა, რომელიც დაკავშირებულია განმეორებადი ტესტების შექმნასთან, ტესტის გაშვების სიჩქარის განსაზღვრასთან, სატესტო მონაცემთა ბაზის შექმნასთან და სხვ.

განმეორებადი ტესტის შექმნა გულისხმობს ისეთი მატესტირებელი სცენარის დაწერას, რომლის გაშვებაც მრავალჯერადად არის შესაძლებელი. დღეისთვის გავრცელებულია მრავალი კერძო და ღია პროგრამული პროდუქტი, რომლებიც უზრუნველყოფს Web აპლიკაციების ავტომატურ ტესტირებას.

ფასიანი პროდუქტებიდან შეიძლება გამოვყოთ QTP (HP QuickTest Professional). უფასო პროდუქტებიდან კი: Selenium, Watir, Canoo WebTest, Cucumber, CubicTest და სხვა [2].

Selenium არის უფასო პროგრამული უზრუნველყოფა, განკუთვნილი Web აპლიკაციების ავტომატური ტესტირებისთვის. Selenium ტესტები შესაძლებელია შეიქმნას ისეთ თანამედროვე პროგრამირების ენებზე როგორიცაა: C#, Java, Groovy, Perl, PHP, Python და Ruby. იგი თავსებადია Windows, Linux, Macintosh ოპერაციულ სისტემებთან [2-5].

Selenium შეიქმნა 2004 წელს Jason Huggins-ის მიერ Tought-Works კომპანიაში. მან გაანალიზა, რომ უკეთსი იქნებოდა დაეზოგა ხელით ტესტირებისთვის საჭირო დრო და განეხორციელებინა ერთიდაიგივე ტესტი ავტომატურად, ყოველი პროგრამული ცვლილებისას. მან შექმნა JavaScript ბიბლიოთეკა, რომლითაც შეეძლო ემართა Web - გვერდი. ამ ბიბლიოთეკას ეწოდა Selenium Core, რომელიც საფუძვლად დაედო Selenium Remote Control (RC) და Selenium IDE - ს.

Selenium RC იყო ინოვაცია, რადგან არცერთ სხვა პროდუქტს არ შეეძლო ეკონტროლებინა Web ბრაუზერი პროგრამული ენის საშუალებით.

სიტყვა Selenium შეიქმნა Huggins-ის ხუმრობიდან, რომელიც მიწერა თავის კონკურენტ ვინმე MMercury-ს და უთხრა, რომ „შენ შეგიძლია განიკურნო Selenium დანამატებითო“.

2006 წელს Google-ს ინჟინერმა, Simon Stewart დაიწყო მუშაობა პროექტზე, რომელსაც მან უწოდა WebDriver. Google-ს ყავდა Selenium-ის გამოცდილი მომხმარებლები, მაგრამ ტესტერებს უწევდათ შეზღუდული ფუნქციებით მუშაობა, რაც განპირობებული იყო JavaScript-ით.

Simon Stewart-მა შექმნა მატესტირებელი ინსტრუმენტი, რომლითაც პირდაპირ მიმართავდა ბრაუზერს და იყენებდა ბრაუზერის მეთოდებს. ამით მან თავი აარიდა JavaScript-ს.

WebDriver პროექტის განხორციელების მიზანი იყო ის, რომ გადაეჭრა Selenium-ის „მტკივნეული” საკითხები [7].

1.4. პროგრამის ბაგი

პროგრამის **ბაგი** არის შეცდომა, ნაკლოვანება კომპიუტერულ პროგრამაში, ან სისტემაში, რომელიც იძლევა არასწორ შედეგს.

ბაგების უმრავლესობა წარმოიქმნება შეცდომებისგან, რომლებიც დაშვებულია ადამიანის მიერ შექმნილ პროგრამულ კოდში ან დიზაინში. აპლიკაცია, რომელიც შეიცავს ბევრ ბაგს, რომლებიც ხელს უშლის მის ფუნქციონალობას, ეწოდება „ბაგებიანი“ პროგრამა.

პროგრამის ფუნქციონირებისას ზოგიერთ ბაგს აქვს „უჩინარი" ეფექტი და შეიძლება იარსებოს დიდი ხნის მანძილზე.

ბაგმა შეიძლება გამოიწვიოს პროგრამის გათიშვა, გაჩერება, „დაკიდება“ და სხვა.

ბაგის მოქმედების შედეგი შეიძლება ძალზედ სერიოზული იყოს. მაგალითად, რაკეტა Ariane-5 განადგურდა გაშვებიდან 1 წუთზე ნაკლებ დროში, რომლის მიზეზი იყო მისი კომპიუტერული უზრუნველყოფის „ბაგი“. 1994 წელს, სამხედრო საჰაერო ვერტმფრენი Chinook დაეჯახა მწვერვალს, რომელიც გამოწვეული იყო პროგრამული უზრუნველყოფის ბაგიდან, რითაც იმართებოდა ვერტმფრენის ძრავა [8].

კომპიუტერულ შეცდომას „ბაგი“ უწოდა ვინმე Grace Hopper–მა, რომელიც იყო კომპიუტერის დამწყები სპეციალისტი. 1946 წელს Grace Hopper–ი მუშაობდა გამოთვლით ლაბორატორიაში Mark-II კომპიუტერზე. მანქანაზე მომუშავე ოპერატორებმა მიაკვლიეს შეცდომის მიზეზს. როგორც გაირკვა ეს შეცდომა გამოწვეული იყო

სქემაზე „დასახლებული" ხოჭოსგან. ეს ხოჭო ფრთხილად აიყვანეს და დასვეს ჩანაწერების ჟურნალზე (ნახ.1.4).

ასე წარმოიქმნა სიტყვა „ბაგი“, რომელიც ინგლისურად ხოჭოს ნიშნავს. Hopper არ იყო ის, ვინც ხოჭო იპოვნა. ერთ–ერთი ოპერატორი რომელმაც იგი იპოვნა იყო William Bruke, რომელმაც გასართობად შეინახა მწერი წარწერით: "პირველი ნამდვილი შემთხვევა არსებული ბაგის პოვნისა".

ნახ.1.3. ბაგის წარმოშობის ისტორია

Hopper–ს უყვარდა ამ ამბის მოყოლა და სწორედ მისი წყალობით შემორჩა ეს ამბავი ისტორიას. აღნიშნული ჩანაწერების ჟურნალი თავის მწერით ინახება ამერიკის Smithsonian-ის ნაციონალურ მუზეუმში [8].

1.5. პროგრამული უზრუნველყოფის ტესტირების ტიპები და მათი კლასიფიკაცია

თანამედროვე ინფორმაციულ ტექნოლოგიებში გავრცელებულია მრავალი ტიპის აპლიკაცია, რომლებიც უზრუნველყოფს მომხმარებელთა სხვადასხვა მოთხოვნილებების დაკმაყოფილებას.

გავრცელებულია ისეთი გლობალური აპლიკაციები, როგორიცაა ERP (Enterprise Resource Planning), E-Commerce სისტემები, BPM (Business Process Management), HR (Human Resources) და ა. შ. [31-35,].

მსგავსი ტიპის აპლიკაციებს იყენებს დიდი კორპორაციები, სახელმწიფო სტრუქტურები, რათა მოახდინოს თავისი საქმიანობის ადვილად მართვა.

Desktop აპლიკაციები (სამაგიდო ანუ Windows აპლიკაციები), რომლებიც ადამიანის მიერ გამოიყენება ისეთ მოთხოვნილებათა დასაკმაყოფილებლად, როგორიცაა: გამოთვლები, თამაშები, ტრეინინგები, გართობა და სხვა.

არსებობს სხვადასხვა პლატფორმები (ე.წ. ოპერაციული სისტემები), რომლებიც კომპიუტერული მოწყობილობების რესურსების გამოყენებით უზრუნველყოფს პროგრამული აპლიკაციების მართვას, სისტემების ინტეგრაციას [1,36].

სხვადასხვა ტიპის კომპიუტერულ აპლიკაციებს აქვს განსხვავებული არქიტექტურები, შესაბამისად განსხვავდება მათი შექმნის პროცესიც. მაშასადამე, იმისთვის, რომ ვიზრუნოთ სხვადასხვა ტიპის პროგრამული უზრუნველყოფის შექმნაზე,

ამისათვის არსებობს მრავალი ტიპის პროგრამული ტესტირება [12,37].

განვიხილოთ პროგრამული უზრუნველყოფის ტესტირების ძირითადი ტიპები:

- **Black box testing** (შავი ყუთის ტესტირება) – ტესტირების მეთოდი, რომლის დროსაც ხდება აპლიკაციის ფუნქციონალის ტესტირება შიგა სტრუქტურის გამოკვლევის გარეშე. ამ დროს პროგრამული კოდის ან არქიტექტურის ცოდნა არ არის საჭირო.

„შავი ყუთის" ტიპის ტესტირებისას ტესტერი ამოწმებს თუ რა გააკეთა პროგრამამ და არ აინტერესებს თუ როგორ მიიღო შედეგი.

მაგალითად, კომპიუტერულ პროგრამას შესასვლელზე ვაძლევთ რაღაც მონაცემებს, ანუ ვაწვდით Input-ს, ხდება ამ მონაცემების დამუშავება და შედეგად ვიღებთ Output-ს (ნახ.1.5). აპლიკაციის შიგა პროცესები უგულებელყოფილია.

ნახ.1.4. Black box testing

საბოლოოდ, შეგვიძლია დავასკვნათ, რომ „შავი ყუთის" ტესტირება არის ტესტირების ტიპი, რომლის დროსაც ხდება პროგრამის ფუნქციონალის შემოწმება შიგა სტრუქტურის გამოკვლევის გარეშე.

- **White box testing** (თეთრი ყუთის ტესტირება) – კომპიუტერული პროგრამის ტესტირების მეთოდი, რომლის დროსაც ტესტირების პროცესი მიმდინარეობს პროგრამული უზრუნველყოფის შიგა სტრუქტურის დონეზე.

„თეთრი ყუთის“ ტესტირება მოითხოვს პროგრამის შიგა სტრუქტურის, პროგრამული კოდის ცოდნას. ძირითადად, „თეთრი ყუთის" ტესტირებას ახორციელებენ პროგრამისტები, რადგან, როგორც აღვნიშნეთ, მის დროს აპლიკაციის შემოწმება ხდება პროგრამული კოდის დონეზე. მსგავსი ტიპის ტესტირება დაფუძნებულია პროგრმული კოდის ცოდნაზე, კომპონენტებსა და მათ შორის კავშირებზე, ციკლებზე და ა.შ. [17].

- **Unit testing** (ერთეულოვანი ტესტირება) – როგორც დასახელებიდან ჩანს, „Unit" ტესტირება არის მცირე ზომის ტესტირების მეთოდი, რომლის დროსაც მიმდინარეობს მარტივი მოდულების, ფუნქციების ტესტირება. აღნიშნული ტიპის ტესტირებას აკეთებს პროგრამისტი და არა ტესტერი, რადგან იგი მოითხოვს პროგრამის შიგა დიზაინის, კოდის დეტალურ ცოდნას. ზოგადად, მსგავსი ტიპის ტესტირებას მიმართავენ მაშინ, როდესაც აპლიკაციის არქიტექტურის დიზაინი დასრულებულია.
- **Incremental testing** (ზრდადი ტესტირება) – ტესტირების მეთოდი, რომლის დროსაც პირველ ეტაპზე მიმდინარეობს ქვესისტემების დამოუკიდებლელი ტესტირება, შემდგომ ეტაპზე ტესტირების პროცესი გადადის გაერთიანებულ ქვესისტემებზე და გრძელდება მანამ, სანამ არ მივიღებთ მთლიან სისტემას.
- **Functional testing** (ფუნქციური ტესტირება) – „შავი ყუთის" ტიპის ტესტირება, რომლის დროსაც ტესტირდება პროგრამის ფუნქცი შესასვლელზე მიცემული მონაცემებით და მიღებული შედეგებით. მის დროს განიხილება პროგრამის შიგა სტრუქტურაც. ფუნქციური ტესტირება მოიცავს პროგრამის სამომხმარებლო ინტერფეისის, მონაცემთა ბაზის და პროგრამული უსაფრთხოების ტესტირებას. ფუნქციური ტესტირება შეიძლება განხორციელდეს ხელოვნურად ან ავტომატურად [5].
- **System testing** (სისტემური ტესტირება) – დასრულებული, მთლიანი კომპიუტერული პროდუქტის ტესტირების მეთოდი.

ზოგადად სისტემური ტესტირება მოიცავს სხვადასხვა ტესტების სერიას, რომელთაც აქვს ერთადერთი მიზანი: დაატესტიროს კომპიუტერზე ბაზირებული მთლიანი სისტემა. აპლიკაცია არის კომპიუტერზე ბაზირებული სისტემის შემადგენელი ერთი–ერთი მცირე ელემენტი.

- **Regression testing** (რეგრესიული ტესტირება) – მისი მიზანია დაადასტუროს, რომ ახალმა პროგრამულმა ცვლილებამ არ გამოიწვია უკვე არსებული და შემოწმებული ფუნქციურობის „გაფუჭება". „Regression" ტესტირების დროს მიმდინარეობს უკვე გაშვებული ტესტების ხელახალი გაშვება. აღნიშნული ტესტირების ტიპი უზრუნველყოფს, რომ ახალ პროგრამულ ცვლილებას არ აქვს გვერდითი მოვლენები არსებულ ფუნქციაზე. „Regression" ტესტების გაშვების აუცილებლობა დგება მაშინ, როდესაც ხდება ქვემოთ მითითებული ერთ-ერთი მოვლენა:
  - ✓ იცვლება მომხმარებლის მოთხოვნები, მაშასადამე იცვლება კოდი მოთხოვნების შესაბამისად;
  - ✓ პროგრამას ემატება ახალი მახასიათებელი;
  - ✓ ფიქსირდება პროგრამული დეფექტი.

მსგავსი ტიპის ტესტირებისას გამოყენებადია ავტომატური მატესტირებელი ინსტრუმენტები, რადგან ყოველი პროგრამული ცვლილებისას მიმდინარეობს ერთიდაიმავე ტესტების მრავალჯერადი გაშვება [17].

- **Acceptance testing** (თანხმობის ტესტირება) – კლიენტის მიერ შესრულებული ტესტირების პროცესი, რომლის დროსაც ხდება დასრულებული სისტემის შემოწმება, ეთანადება თუ არა სისტემა მყიდველის მოთხოვნებს. იგი არის ტესტირების ერთ-ერთი ბოლო ფაზა, მანამ, სანამ მოხდება აპლიკაციის კომერციულ წარმოებაში გაშვება [5].

- **Load testing** (დატვირთვით ტესტირება) – აპლიკაციის ტესტირება რთულად დასამუშავებელი მონაცემებით, მაგალითად Web-საიტის დატვირთვით ტესტირება, მისი მიზანია განისაზღვროს თუ რა წერტილში ქვეითდება, ნელდება ან „ეკიდება“ სისტემის მუშაობა.
- **Stress testing** (სტრესული ტესტირება) – ტესტირების მეთოდი, რომლის დროსაც ხორციელდება სისტემის „სტრესული" რეჟიმით მუშაობა, რათა შემოწმდეს როდის ან როგორ „ჩავარდება“ იგი. პროგრამის საწყის მონაცემებად განიხილება დიდი რიცხვები, ან მონაცემთა ბაზის რთული მოთხოვნების გაშვება და ა.შ.
- **Usability testing** (გამოყენებადი ტესტირება) – საბოლოო მომხმარებლის მიერ ყველაზე ხშირად გამოყენებადი ფუნქციების ტესტირება. მისი მიზანია გამოყენებად ფუნქციებში გამოააშკარაოს პროგრამული დეფექტები [5].
- **Security testing** (უსაფრთხოების ტესტირება) – ტესტირების მეთოდი, რომლის დროსაც მოწმდება თუ როგორ არის დაცული სისტემა არასანქცირებული გარე თუ შიგა წვდომისგან, წინასწარგანზრახული დაზიანებისგან და ა.შ. მის დროს ხორციელდება სისტემის ყველა შესაძლო სუსტი წერტილის ტესტირება, რათა არასანქცირებული შეტევისას არ მოხდეს ინფორმაციის წაშლა, ინფორმაციის მოპარვა, პროგრამის მწყობრიდან გამოყვანა და ა.შ. პროგრამის უსაფრთხოების უზრუნველყოფა არ გულისხმობს შემდეგი ტიპის დაცვას (ნახ.1.6):

ნახ.1.5. ნახაზი განწყობისთვის

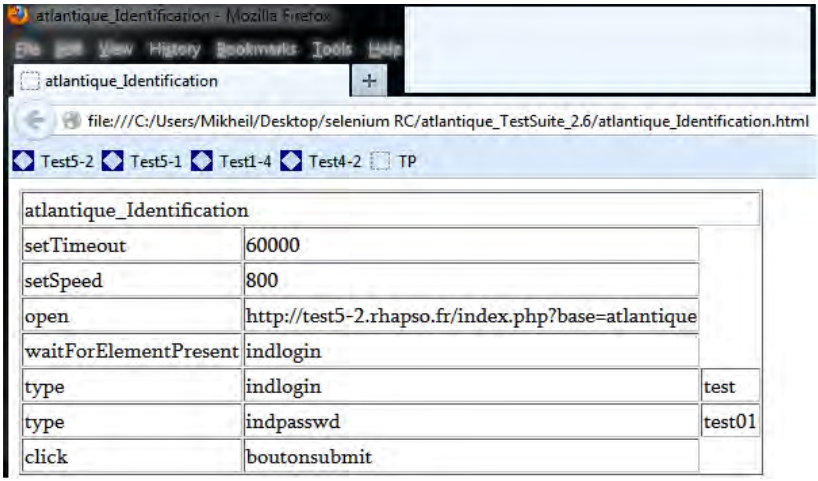
1.6. ტესტ-ქეისები და ტესტ-სუიტები

ტესტ-ქეისი არის იმ ცვლადების, პირობების, მოქმედებების ერთობლიობა, რომელთა საშუალებით მოწმდება პროგრამული უზრუნველყოფის მუშაობის სისწორე. ტესტ-ქეისების საშუალებით ადგენენ სისტემის მუშაობის ხარისხს.

დაწერილ, შექმნილ ტესტ-ქეისს ეწოდება ტესტ-სკრიპტი. იგი ხელოვნურად ან ავტომატურად ჩაწერილი სკრიპტია, რომელიც ახდენს შესაბამის ტესტ-ქეისში მოცემული ინსტრუქციების შესრულებას.

პროგრამული უზრუნველყოფის ტესტირებისას შესაძლებელია დაიწეროს ხელოვნური ტესტ-სკრიპტი, რომელიც შესრულდება ადამიანის მიერ, ან შესაძლებელია გაკეთდეს ტესტ-ქეისში მოცემული ინსტრუქციების ავტომატიზაცია.

1.7 ნახაზზე წარმოდგენილია Selenium ტესტი, რომელიც ავტომატური ტესტ–სკრიპტია.



atlantique_Identification		
setTimeout	60000	
setSpeed	800	
open	http://test5-2.rhapso.fr/index.php?base=atlantique	
waitForElementPresent	indlogin	
type	indlogin	test
type	indpasswd	test01
click	boutonsubmit	

ნახ.1.7. ტესტ–სკრიპტი

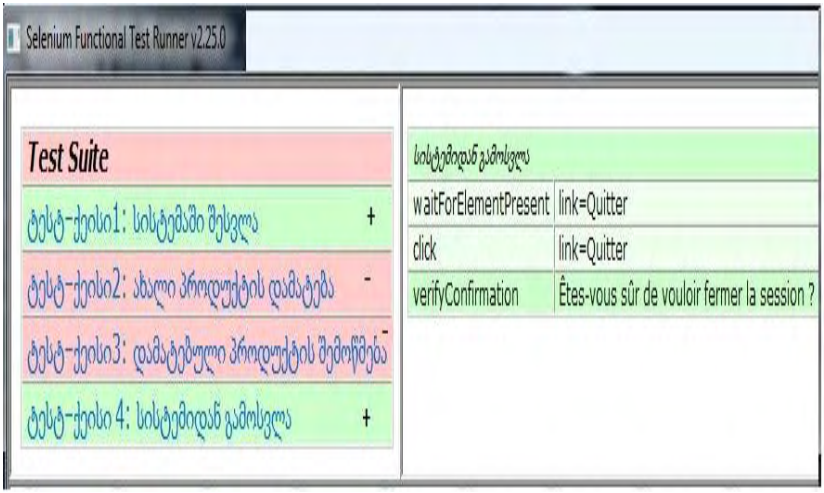
ტესტ-სუიტი არის ტესტ-ქეისების ერთობლიობა. სისტემის ერთი მთლიანი ფუნქციის ტესტირების მიზნით საჭიროა ტესტ-ქეისების გაერთიანება სხვადასხვა ჯგუფებად, ტესტ-სუიტებად.

განვიხილოთ ტესტ-სუიტის მაგალითი, რომელიც შედგება 4 ტესტ-სკრიპტისგან:

- ტესტ-ქეისი 1: სისტემაში შესვლა;
- ტესტ-ქეისი 2: ახალი პროდუქტის დამატება;
- ტესტ-ქეისი 3: დამატებული პროდუქტის შემოწმება;
- ტესტ-ქეისი 4: სისტემიდან გამოსვლა.

განხილულ ტესტ-სუიტში მნიშვნელოვანია ტესტ-ქეისების მიმდევრობა, თითოეული ტესტის წარმატებით დასრულება. მაგალითად, წარმოვიდგინოთ, რომ ტესტ-სუიტის შესრულებისას არ შესრულდა ახალი პროდუქტის დამატება. ამ შემთხვევაში მომდევნო ტესტი – „დამატებული პროდუქტის შემოწმება" „ჩავარდება“, რაც ლოგიკურია – თუ ვერ შეიქმნა პროდუქტი, შესაბამისად ვერ მოხდება მისი შემოწმებაც.

ამ შემთხვევაში ამბობენ, რომ სუიტში შემავალი ტესტ-ქეისები დამოკიდებულია ერთმანეთზე, ანუ შემდგომი ტესტ-სკრიპტის შესრულება დამოკიდებულია წინა ტესტ-სკრიპტის შესრულების შედეგზე (ნახ.1.8).



ნახ.1.8. ტესტ-სუიტი

მწვანე ფერით (მონიშნულია „+“) მოცემულია წარმატებით შესრულებული ტესტ–სკრიპტები, ხოლო წითელ ფერში წარმოდგენილია „ჩავარდნილი“ ტესტები (მონიშნულია „-“).

მოცემული სურათიდან შეგვიძლია დავასკვნათ, რომ ჩავარდა „ახალი პროდუქტის დამატების“ ტესტი, შესაბამისად ჩავარდა მასზე დამოკიდებული მომდევნო ტესტი – „დამატებული პროდუქტის შემოწმება“, ხოლო წარმატებით შესრულებულა სისტემაში „შესვლა – გამოსვლის“ ტესტები [12].

1.7. პირველი თავის დასკვნა

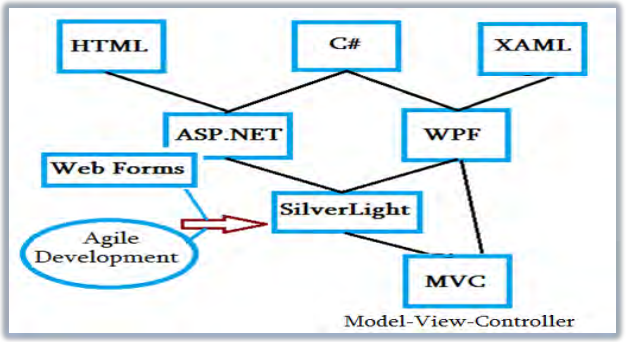
განხილულ იქნა ორგანიზაციული მენეჯმენტის ბიზნეს-პროცესების ავტომატიზაციის ინფორმაციული მხარდაჭერის პრობლემეატიკა, IT-სერვისების მართვის ამოცანების მნიშვნელობა, დაგეგმვის, დაპროექტების, დაპროგრამების და დანერგვის სახით. განსაკუთრებული ყურადღება უნდა მიექცეს პროგრამული სისტემების ინტეგრაციის დროს ტესტირების, ვერიფიკაციის და ვალიდაციის ამოცანებს.

ჩატარებულია პროგრამული ტესტირების სისტემების კლასიფიკაცია და განსაზღვრულია ამოცანა ორგანიზაციული მართვის სისტემებში ბიზნესპროცესების ეფექტურობის ამაღლებისათვის IT-სერვისების (პროგრამების) ტესტირების ჩატარების და შემდეგ დროული მიწოდების შესახებ.

II თავი

Web აპლიკაციების აგების თანამედროვე ინსტრუმენტული საშუალებების ანალიზი „მაიკროსოფთის“ მაგალითზე

განვიხილავთ კომპანია Microsoft-ის თანამედროვე ვებ-ტექნოლოგიებს და მათ გამოყენებას ბიზნეს-აპლიკაციების დამუშავების მიზნით Visual Studio.NET სამუშაო გარემოში (ნახ.2.1).



ნახ.2.1. Web-აპლიკაციების ტექნოლოგიების ევოლუცია

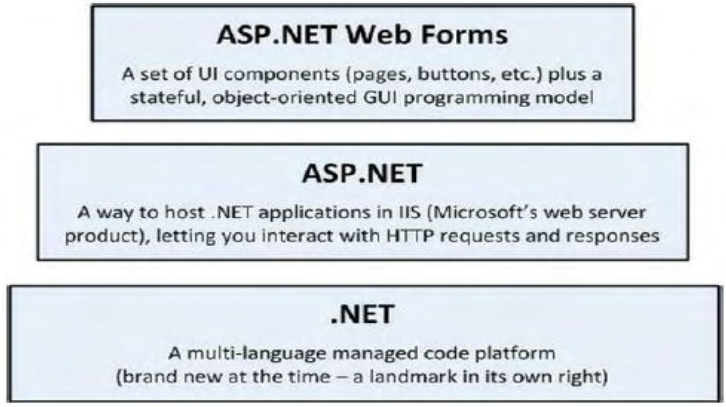
გავაანალიზებთ ASP.NET Web Forms, Silverlight და MVC ტექნოლოგიებს, შევადარებთ მათ მახასიათებლებს, დადებითი და უარყოფითი მხარეების დახასიათებით.

2.1. ASP.NET ტექნოლოგია

Web-ტექნოლოგიის განვითარების ისტორია გასული საუკუნის 60-ანი წლებიდან დაიწყო. ინტერნეტი ექსპერიმენტის დონეზე მუშავდებოდა და ხელმისაწვდომი იყო მომხმარებელთა შეზღუდული რაოდენობისთვის (საგანმანათლებლო სისტემები, ინსტიტუტები, თავდაცვის სამსახურები და ა.შ.). 1990 წელს მოდემის გამოგონებამ მკვეთრად დააჩქარა ინტერნეტზე მოთხოვნილების ზრდა. 1993 წელს შეიქმნა პირველი HTML ბრაუზერი.

ადრეული ვებ-საიტები ბროშურებს წააგავდა: შედგებოდა სტატიკური HTML გვერდებისგან. HTML2-ის შემუშავების შემდეგ გამოჩნდა HTML ფორმები, რომლებიც გარდა ფორმატირებული ტექსტის ასახვისა, კონტროლების (როგორიცაა ჩამოსაშლელი სია, ტექსტური ველი, ღილაკი) გამოყენების საშუალებასაც იძლეოდა.

ადრეულ ვებ-დეველოპმენტს ჰქონდა ბევრი შეზღუდვა: მაგალითად, მრავალი მომხმარებლის ერთდროულად შესვლისას შენელებული მუშაობა/წყვეტა და შედარებით მაღალი პროგრამული ლოგიკის გამოყენების (მომხმარებლის ავტორიზაცია, ინფორმაციის შენახვა, მონაცემთა ბაზიდან ჩანაწერების წამოღება/ასახვა) მწირი შესაძლებლობა. ამ პრობლემების გადასაჭრელად Microsoft-ის კომპანიამ შეიმუშავა უფრო მაღალი დონის დეველოპმენტ-პლატფორმები: ASP და ASP.NET (ნახ.2.2).



ნახ2.2. ASP.NET Web Forms ტექნოლოგიის განვითარების სკალა

2002 წელს Microsoft-მა შეიმუშავა .NET Framework-ის 1.0 ვერსია, რომელსაც მოჰყვა Active Server Pages (ASP) ტექნოლოგიის შემუშავება [38]. ASP.NET აგებულია Common Language Runtime (CLR) გარემოზე, რაც საშუალებას აძლევს დეველოპერს ASP.NET კოდი შეიმუშაოს ნებისმიერ .NET ენაზე (C++, C#, VB .Net).

ეს ტექნოლოგიები იძლეოდა დინამიკური, ინტერაქტიული, სერვერული მხარის ვებ-საიტების შემუშავების საშუალებას. ASP.Net მუშაობს HTTP პროტოკოლზე და HTTP ბრძანებებით ახდენს ორმაგ კომუნიკაციას ბრაუზერსა და სერვერს შორის. ASP.Net აპლიკაცია კომპილირებული კოდია, რომელიც იყენებს .NET Framework-ში არსებულ კლასებს და იერარქიულ სტრუქტურებს. ASP.Net Web Forms მოდელმა კიდევ უფრო განავრცო ვებ-აპლიკაციის მოვლენებით (Events) მართული მოდელი - ბრაუზერი გადასცემს ვებ-ფორმის ინფორმაციას სერვერს და სერვერი საპასუხოდ აბრუნებს დამუშავებულ HTML გვერდს,

Web Forms ტექნოლოგიით „მაიკროსოფთი“ შეეცადა დაემალა HTTP (stateless თვისებით) და HTML (რომელიც მაშინ დეველოპერებისთვის უცნობი იყო). ამისათვის შემუშავდა სამომხმარებლო ინტერფეისის სპეცკომპონენტები – server-ული მხარის კონტროლები, რომლებიც:

- Request-ებს შორის თავიანთი მდგომარეობის (state) შესახებ ინფორმაციას ინახავდა ViewState ობიექტში;
- რენდერდებოდა HTML ფორმატში და ავტომატურად აკავშირებდა კლიენტის მხარის event-ებს (მაგალითად, Button Click) შესაბამის server-ულ event handler-ის კოდთან.

საერთო ჯამში, შეიძლება ითქვას, რომ Web Form-ები გიგანტური აბსტრაქციის შრეა, რომელიც უზრუნველყოფს კლასიკურ, event-ებით მართულ გრაფიკულ სამომხმარებლო ინტერფეისს (GUI) Web-ზე. ამ იდეოლოგიის ძირითადი დანიშნულება იყო Web Development პროცესი დამსგავსებოდა Windows Forms Development პროცესს.

დეველოპერები აღარ საჭიროებდნენ ურთიერთდამოუკიდებელ HTTP Request-ებთან და Response-ბთან მუშაობას, მათ უკვე შეეძლოთ ეფიქრათ როგორც stateful UI -ზე და ამგვარად, Microsoft-მა მოახერხა Desktop -აპლიკაციების დეველოპერების მთელი არმია გადმოეყვანა ვებ-აპლიკაციების სამყაროში.

2.2. ASP.NET Web Forms ტექნოლოგიის ნაკლოვანებები

ტრადიციული ASP.NET Web Form-ების ტექნოლოგიას, რომელსაც მრავალი დადებითი მხარე გააჩნია, აქვს გარკვეული ნაკლოვანებებიც [52]. აღნიშნული ტექნოლოგიის გამოყენება, აღმოჩნდა, რომ უფრო გართულდა შემდეგი მიზეზების გამო:

- ViewState ობიექტის წონა: მექანიზმი, რომელიც უზრუნველყოფდა request-ებს შორის მდგომარეობის (State) შენარჩუნებას, (View State) იწვევდა კლიენტსა და სერვერს შორის დიდი ზომის მონაცემთა ბლოკების ტრანსფერს (100-ობით კილობაიტს აღწევდა ყველაზე მოკრძალებულ ვებ-აპლიკაციაშიც კი). ეს მონაცემთა ბლოკები წინ და უკან „მოგზაურობდა“ ყოველი request-ის დროს, რაც იწვევდა Response Time დროის დაყოვნებას და სერვერის გამტარუნარიანობაზე მოთხოვნის ზრდას;
- გვერდის სასიცოცხლო ციკლი (Page Life Cycle): მექანიზმი, რომელიც აკავშირებს Client-Side Event-ებსა და Server-Side Event Handler-ის კოდს, წარმოადგენს გვერდის სასიცოცხლო ციკლის ნაწილს, რის გამოც მეტად რთული და დელიკატური საკითხი იყო ამ პროცესში გარკვევა და შეცდომების თავიდან არიდება. ხშირი იყო ViewState-ის შეცდომები, ზოგიერთი Event Handler-ი დეველოპერისთვის გაურკვეველი მიზეზების გამო ვერ გაიშვებოდა;
- მცდარი ილუზია განცალკევებული კონცეფციების შესახებ: ASP.NET Web-Forms ტექნოლოგიის code behind მოდელით აპლიკაციის კოდი HTML markup-იდან გატანილი იყო ცალკეულ code-behind კლასებში. ეს მეთოდი მოიწონეს, ვითომ და ლოგიკისა და პრეზენტაციის დონეების განცალკევების გამო, მაგრამ სინამდვილეში ასე არ იყო: დეველოპერი, პირიქით, პრეზენტაციის შრეში წერდა აპლიკაციის ლოგიკის კოდსაც (მაგალითად, code behind-ში შეიძლება შეგვხვედროდა სერვერული ხის იერარქიის სამართავად

დაწერილი კოდიც და აპლიკაციის ლოგიკაც. მაგალითად, მონაცემთა ბაზის ინფორმაციის დამუშავება). საბოლოო შედეგი ხშირად არასტაბილური და არასაიმედო იყო;

- HTML-ზე შეზღუდული კონტროლი: სერვერული კონტროლები რენდერდებოდა HTML-ის სახით. ASP.NET-ის ადრეულ ვერსიებში HTML output-ი არ ემთხვეოდა ვებ-სტანდარტებს, ვერ ხერხდებოდა CSS სტილების კარგი გამოყენება, სერვერული კონტროლები აგენერირებდა ID ატრიბუტის გაურკვეველ მნიშვნელობებს, რომლებიც ართულებდა Javascript-ით მათზე წვდომას. მართალია, ეს პრობლემატური საკითხები თანამედროვე Web Forms ტექნოლოგიაში ბევრად გაუმჯობესებულია, მაგრამ, მიუხედავად ამისა, მაინც საკმაოდ რთულია მოსალოდნელი HTML-ის მიღება;
- ტესტირების დაბალი მხარდაჭერა: Web Forms ტექნოლოგიის შემუშავებისას მის არქიტექტორებს არ შეეძლოო მხედველობაში მიეღო ის ფაქტი, რომ სამომავლოდ ავტომატური ტესტირება პროგრამული უზრუნველყოფის შექმნის არსებით ნაწილი გახდებოდა. არაა გასაკვირი, რომ მჭიდროდ ურთიერთდაკავშირებული არქიტექტურა მოუხერხებელია Unit Testing-ისა და Integration Testing ტესტირების უზრუნველსაყოფად.

2.3. ASP.NET MVC ტექნოლოგია

ASP.NET Web Forms ტექნოლოგიის კრიტიკისა და არა-Microsoft ტექნოლოგიების (განსაკუთრებით - Ruby on Rails ტექნოლოგია, თავისი Agile Development მიდგომებით MVC არქიტექტურით, და HTTP პროტოკოლზე მუშაობის პრინციპით) მზარდი პოპულარობის საპასუხოდ, 2007 წლის ოქტომბერში მაიკროსოფთმა გააკეთა განცხადება ASP.NET პლატფორმაზე

დაშენებული ახალი პროგრამული პლატფორმის გამოშვების შესახებ [52].

ამ ახალ ტექნოლოგიაში კომბინირებულია Model-View-Controller (MVC) არქიტექტურის ეფექტურობა, Agile Development მიდგომის უახლესი იდეოლოგია და ASP.NET პლატფორმის საუკეთესო ნაწილები [53]. იგი ტრადიციული ASP.NET Web Forms ტექნოლოგიის სრულყოფილი ალტერნატივაა, უპირატესია თითქმის ყველა შემთხვევაში, გარდა ტრივიალური ვებ საიტების პროექტირებისა.

ქვემოთ განვიხილავთ თუ როგორ დაძლია ASP.NET MVC პროგრამულმა პლატფორმამ Web Forms პლატფორმის შეზღუდვები და დაუბრუნა ASP.NET ტექნოლოგიას ძველი დიდება.

2.3.1. ASP.NET MVC ტექნოლოგიის უპირატესობანი

- **MVC არქიტექტურა**

მნიშვნელოვანია განვასხვავოთ MVC არქიტექტურული სტანდარტი (pattern) და ASP.NET MVC Framework პროგრამული პლატფორმა [52]. MVC არქიტექტურა ახალი არ არის – იგი სათავეს იღებს 1978 წლიდან (Smaltalk პროექტი, Xerox PARC კვლევითი ჯგუფის მიერ შემუშავებული) და დღესდღეობით დიდი პოპულაროზით სარგებლობს როგორც Web-აპლიკაციების ასაგებად გამოსაყენებელი არქიტექტურული მიდგომა. ამ ფართო აღიარების მიზეზები შემდეგია:

- მომხმარებლის ურთიერთქმედება MVC არქიტექტურით აგებულ აპლიკაციასთან ბუნებრივ ციკლს მიჰყვება – მომხმარებელი ასრულებს მოქმედებას, საპასუხოდ აპლიკაცია ცვლის Data Model ობიექტს და მომხმარებელს უბრუნებს შეცვლილ view ობიექტს, შემდეგ ეს ციკლი ისევ მეორდება. ეს მიდგომა კარგად

არის მორგებული Web აპლიკაციების HTTP პროტოკოლზე HTTP Request და HTTP Response სერიებით მუშაობის პრინციპთან;

- ვებ-აპლიკაციები საჭიროებს რამდენიმე ტექნოლოგიის კომბინირებას (მონაცემთა ბაზა, HTML, მზა ბიბლიოთეკები და სხვა), რომლებიც გადანაწილებულია ლოგიკურ შრეებში (Layers). ამ დაყოფის მიდგომა სრულ თანხმობაშია MVC არქიტექტურის კონცეფციასთან;

ASP.NET MVC Framework პროგრამულმა პლატფორმამ იმპლემენტაცია გაუკეთა MVC არქიტექტურას და ამით დიდი კონკურენცია გაუწია Ruby on Rails და მის მსგავს პროგრამულ პლატფორმებს, ხოლო MVC არქიტექტურა კი .NET სამყაროში წინა პლანზე წამოწია. სხვა, არა-Microsoft პლატფორმაზე მომუშავე დეველოპერების მიერ წლობით დაგროვილი გამოცდილებისა და საუკეთესო პრაქტიკების გადაღებით გამდიდრებულმა ASP.NET MVC პლატფორმამ ბევრად წინ გაუსწრო კონკურენტუნარიან პლატფორმებს (მათ შორის Rails ტექნოლოგიასაც).

➢ განვრცობადობა (Extensibility)

MVC პაკეტი აგებულია ურთიერთდამოუკიდებელი კომპონენტებისგან, რომლებიც აკმაყოფილებს .NET ინტერფეისს ან დაშენებულია აბსტრაქტულ მშობელ კლასზე (abstract base class).

პროგრამისტს იოლად შეუძლია ჩაანაცვლოს არსებული კომპონენტები (როგორიცაა მარშრუტიზაცის სისტემა - routing system, view engine მექანიზმი, controller factory), საკუთარი იმპლემენტაციით. MVC პროგრამული პლატფორმა დეველოპერს სთავაზობს თითოეული კომპონენტის გამოყენების 3 შესაძლო ვარიანტს:

- გამოიყენოს კომპონენტის default-იმპლემენტაცია, რაც აპლიკაციების უმრავლესობისთვის სავსებით საკმარისია;

- მემკვიდრეობის პრინციპით შექმნას ახალი კლასი(subclass) და საკუთარი იმპლემენტაცია დააშენოს default-ობიექტზე;
- ახალი იმპლემენტაციით მთლიანად ჩაანაცვლოს არსებული კომპონენტი.

➢ HTML-ზე და HTTP-ზე მაღალი ხარისხის კონტროლის მექანიზმები

ASP.NET MVC ტექნოლოგია სუფთა, სტანდარტებთან თავსებადი HTML მარკირების საშუალებას იძლევა. MVC ტექნოლოგიაში ჩაშენებული HTML helper მეთოდები იძლევა სტანდარტებთან თავსებად HTML კოდს.

მნიშვნელოვანი ცვლილება მოხდა ფილოსოფიურ მიდგომაში: Web Forms ტექნოლოგიის პრინციპებისგან განსხვავებით, აღარ არის მოწონებული კომპლექსური HTML-კონტროლების გამოყენებისა და დიდი HTML კოდის გენერაციის ტენდენცია. ამის ნაცვლად, MVC ტექნოლოგია მხარს უჭერს ელეგანტური, სადა მარკირების გამოყენებას CSS სტილებით.

რა თქმა უნდა, თუ პროგრამისტს სურს მზა-კონტროლებისა და გაჯეტების გამოყენება სამომხმარებლო ინტერფეისის გასამდიდრებლად (კასკადური მენიუ, კალენდარის კონტროლი და სხვა), ASP.NET MVC ტექნოლოგია თავისი ლოიალური მიდგომით მარკირების მიმართ, ადვილად იძლევა სხვადასხვა მზა ბიბლიოთეკების (jQuery UI, Bootstrap CSS library, Knokout) გამოყენების საშუალებას.

ASP.NET MVC ტექნოლოგიით გენერირებული გვერდები არ შეიცავს View State მდგომარეობის ამსახველ მონაცემებს, ამგვარად მიღებული გვერდები უფრო მსუბუქია, ვიდრე ტიპური ASP.NET ვებ-ფორმები. მიუხედავად თანამედროვე სწრაფი ინტერნეტისა, ეს ეკონომია კვლავ დიდ უპირატესობას იძლევა და ამცირებს ვებ-აპლიკაციის გაშვების დროსა და საფასურს.

ASP.NET MVC მუშაობს HTTP პროტოკოლზე. პროგრამისტს სრული კონტროლი აქვს ვებ-ბრაუზერსა და სერვერს შორის გადაწოდებულ მოთხოვნებზე. AJAX ტექნოლოგია საკმაოდ იოლად არის შემუშავებული და სამომხმარებლო ინტერფეისზე არ ასრულებს ავტომატურ postback მოქმედებებს.

- **ტესტირება**

MVC არქიტექტურა მაღალი ხარისხის ტესტირების საშუალებას იძლევა. ეს შედეგი არქიტექტურული მიდგომის პრინციპებიდან გამომდინარეობს – ლოგიკური ნაწილები ერთმანეთისგან გამიჯნულია და სხვადასხვა შრეზეა განლაგებული [54].

გარდა ამისა, MVC პროგრამული პლატფორმის შექმნისას, მისმა არქიტექტორებმა MVC Framework-ის კომპონენტზე ორიენტირებული დიზაინის თითოეული ლოგიკური კომპონენტი საფუძვლიანად დააპროექტეს Unit Testing და Mocking Tools ხელსაწყოებთან სრული შესაბამისობის მისაღწევად.

Visual Studio პროგრამაში Unit Test პროექტების შესაქმნელად დამატებულია ე.წ. ოსტატები (Wizards), რაც შემდგომში ინტეგრირებადია ტესტირების სხვა ინსტრუმენტებთანაც, როგორიცაა NUnit, xUnit. MVC-აპლიკაციაში იოლად რეალიზებადია ტესტები სხვადასხვა კომპონენტთან მიმართებაში (როგორიცაა Controller, Action), სიმულაციები, სცენარები.

პროგრამისტს შეუძლია მომხმარებლის მოქმედების სიმულაცია გააკეთოს ტესტირების სკრიპტებით და არ ევალება იმის ცოდნა, თუ როგორ არის HTML ელემენტების და CSS კლასების სტრუქტურა, ან დაგენერირებული ID მნიშვნელობები.

➢ მძლავრი მარშრუტიზაციის სისტემა (Routing System)

Web-ტექნოლოგიების განვითარებასთან ერთად, URL მისამართების სტრუქტურაც გაუმჯობესდა. ძველი URL მისამართი თუ ასე გამოიყურებოდა:

```
/App_v2/User/Page.aspx?action=show%20prop&prop_id=82742
```

ახალი URL მისამართი უფრო სუფთა და გასაგები ფორმატითაა წარმოდგენილი:

```
/to-rent/tbilisi/71-chavchavadze-street
```

URL მისამართის სტრუქტურაზე ყურადღების გამახვილებას შემდეგი საფუძვლიანი მიზეზები აქვს:

- საძიებო მანქანები მნიშვნელობას ანიჭებს URL-ში ნაპოვნ საკვანძო სიტყვებს. მაგალითად, ძიება კომბინაციით „rent in Tbilisi“ უფრო მეტი ალბათობით იპოვის სადა URL მისამართს;
- მომხმარებელი ვებ-ბრაუზერის URL სექციაში ხელით კრეფს მარტივ URL-მისამართს;
- მარტივ URL მისამართს მომხმარებელი ადვილად იმეორებს და იმახსოვრებს;
- არ ამჟღავნებს ტექნიკური იმპლემენტაციის დეტალებს (საქაღალდის ან ფაილის დასახელება, აპლიკაციის სტრუქტურა), ამიტომ ვებ-აპლიკაციის სტრუქტურის იმპლემენტაციის შეცვლა არ გამოიწვევს გარე ლინკების დარღვევას.

ადრეულ პროგრამულ პლატფორმებზე სადა URL მისამართების იმპლემენტაცია რთული გზებით მიიღწეოდა, მაგრამ ASP.NET MVC პლატფორმა იყენებს URL Routing ფუნქციას და იძლევა ე.წ. „სუფთა“, აზრიან URL-გზავნილებს. ეს თვისება ამარტივებს თანამედროვე REST-სტილის URL-სქემების შემუშავების პროცედურას.

➢ ASP.NET პლატფორმის საუკეთესო პრაქტიკებზე აგებული სისტემა

Microsoft-ის არსებული ASP.NET პლატფორმა ვებ-აპლიკაციების დეველოპმენტისთვის საჭირო მომწიფებულ და კარგად გამოცდილ კომპონენტთა სიმრავლეს იძლევა.

ASP.NET MVC დაშენებულია .NET პლატფირმაზე – პროგრამული კოდის წერა ნებისმიერ .NET ენაზე არის შესაძლებელი, ხელმისაწვდომია .NET ბიბლიოთეკები და ASP.NET პლატფორმის მზა ფუნქციები, როგორიცაა: Authentication, Membership, Roles, Profiles, Internationalization.

ეს ფუნქციები ისეთივე წარმატებით მუშაობს MVC-აპლიკაციებში, როგორითაც Web Forms აპლიკაციებში. MVC პროგრამული პლატფორმის საფუძველად აღებული ASP.NET პლატფორმა ინსტრუმენტების ფართო არჩევანს იძლევა MVC framework-ზე სამუშაოდ.

MVC პლატფორმამ მშობელი ASP.NET პლატფორმიდან მემკვიდრეობით მიიღო პროგრამირების ბოლო ტენდენციები და ინოვაციები, როგორიცაა: await keyword, lambda expressions, extension methods, anonymous types, dynamic types, LINQ queries.

➢ ღია კოდი (Open Source)

MVC პროგრამული პლატფორმა გახსნილი, ანუ „ღია კოდია“ – მისი პროგრამული კოდის გადმოწერა უფასოა და ჩამოტვირთულ პროექტში შესაძლებელია საკუთარი ექსპერიმენტების ჩაატარებაც კი. ეს განსაკუთრებით მოსახერხებელია საკუთარი რთული კომპონენტების შემუშავებისას ან როდესაც Debug-პროცესში შეცდომის კვალს მივყავართ სისტემურ კომპონენტთან და საჭიროა მის კოდში შესვლა. MVC კოდის გადმოწერა შესაძლებელია შემდეგი URL მისამართიდან:

http://aspnetwebstack.codeplex.com

2.3.2. MVC არქიტექტურა

➢ MVC მოდელის ისტორია

ტერმინი Model-View-Controller პირველად გაჩნდა 1970-იან წლებში Smalltalk ენაზე დაწერილ პროექტში (Xerox PARC ჯგუფის მიერ) - როგორც ადრეული გრაფილული სამომხმარებლო ინტერფეისის მქონე (Graphical User Interface - GUI) აპლიკაციების ორგანიზების მიდგომა [63].

MVC აპლიკაციებში ურთიერთქმედება გულისხმობს ნატურალურ ციკლს მომხმარებლის მოქმედებებსა (use actions) და view-პრეზენტაციის დონის განახლებებს შორის. Viw ლოგიკური შრეზე მდგომარეობის შენახვა არ ხდება (stateless), რაც კარგად შეესაბამება ვებ-აპლიკაციების HTTP პროტოკოლზე Request და Response გზავნილებით მუშაობის პრინციპს.

MVC არქიტექტურა სავალდებულოს ხდის აპლიკაციის ლოგიკურ დანაწილებას (Separation of Concerns) – დომეინ მოდელი და კონტროლერის მოდელი გამიჯნულია პრეზენტაციის დონიდან. ეს ნიშნავს, რომ HTML კოდი განცალკევებულია დანარჩენი პროგრამული კოდისგან. შესაბამისად, - ტესტირება და აპლიკაციის მხარდაჭერა გაიოლებულია.

ფუნქციონალის ლოგიკური დაყოფის მიდგომით MVC ტექნოლოგია განსხვავდება სტანდარტული ვებ-ფორმების ტექნოლოგიისგან (Web Forms), სადაც მომხმარებლის მიერ შეტანილი მონაცემები (user input) გადაეწოდება ვებ-გვერდს (View) და უშუალოდ ვებ-ფორმა არის პასუხისმგებელი შეტანილი ინფორმაციის გაადამუშავებასა და პასუხის დაბრუნებაზე.

MVC არქიტექტუის მქონე აპლიკაციის ფუნქციონალი გადანაწილებულია 3 ძირითად კომპონენტში, რომლებიც აპლიკაციას შემდეგ ლოგიკურ დონეებად ყოფენ: მოდელი (Model),

პრეზენტაციის დონე (View) და კონტროლერი (Controller). განვიხილოთ ეს დონეები ცალ-ცალკე.

- **Model (მოდელი)**

მოდელი არის აპლიკაციის მონაცემთა დომეინი, ანუ მონაცემთა სიმრავლე, რომელთანაც მუშაობს მომხმარებელი.

დომეინ მოდელი შედგება ბიზნეს დომეინის მონაცემების სიმრავლის, ოპერაციების, მონაცემთა ტრანსფორმირებისა და მართვის წესებისგან. მოდელის ფუნქციებში არ შედის გრაფიკული ელემენტების გამოტანა ან request-მოთხოვნების დამუშავება (view და controller დონეების ფუნქცია). მოდელი წარმოადგენს რეალური სამყაროს ობიექტების, წესებისა და ოპერაციების ასახვას პროგრამულ უზრუნველყოფაში. ASP.NET Framework-ის ენაზე მოდელი გულისხმობს:

- C# ტიპების (Classes, Structs, Enums და სხვა) ერთობლიობას, რაც ცნობილია როგორც დომეინის ტიპი (Domain Type);
- დომეინში არსებულ ობიექტებზე შესაძლო ოპერაციები აღწერილია დომეინის ტიპის მეთოდებიში (Domain Method);
- დომეინის წესები იმპლემენტირებულია დომეინის მეთოდების შიგნით.

მოდელი არის აპლიკაციის შიგნით არსებული Entity-ერთეულებისა და ბიზნეს წესების ერთობლიობა. დომეინის შემუშავების მიდგომა ბევრ პრობლემურ საკითხს ჭრის. თუ საჭირო გახდა მონაცემის რაიმე ბიზნეს ერთეულის, პროცესის ან წესის შეცვლა – დომეინ ლოგიკური დონე წარმოადგენს იმ ერთადერთ ადგილს, სადაც აპლიკაციის კოდში ცვლილება მოხდება.

ზოგადი პრაქტიკით მიღებულია დომეინ მოდელის ცალკე ბიბლიოთეკად (C# Assembly) გატანა და აპლიკაციის სხვა პროექტებში მისი ჩამატება Reference-სახით (ლოგიკური შრეების ურთიერთ-გამიჯვნის პრონციპი).

- **View**

პრეზენტაციის შრე არის მომხმარებლის ინტერფეისის იმპლემენტაცია (User Interface – UI). ამ კომპონენტის ფუნქციაა მონაცემთა ასახვა, გამოტანა მომხმარებლისთვის, ზოგადად view წარმოადგენს შაბლონს, რომელიც უზრუნველყოფს Runtime რეჟიმში საბოლოო HTML-ის გენერაციას [52].

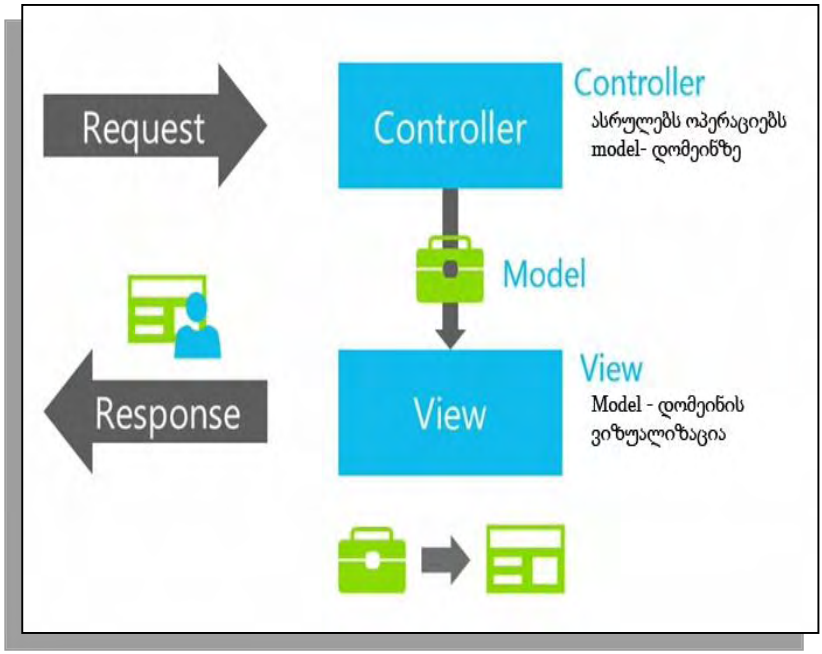
- **Controller (კონტროლერი)**

კონტროლერის დანიშნულება არის მომხმარებლის მიერ შეტანილი ინფორმაციის დამუშავება (Input Control), მომხმარებლის ინტერაქტიულობით უზრუნველყოფა და დაკავშირება სისტემასთან (View - კომპონენტის დაკავშირება Model-კომპონენტთან).

კონტროლერი კითხულობს მონაცემებს ფორმიდან, ამოწმებს შეტანილი მონაცემების ვალიდურობას და უგზავნის მათ model-კომპონენტს. Controler-ობიექტი წყვეტს რომელი view-კომპონენტი ასახოს მომხმარებლის მიერ გადაცემული ინფორმაციის საპასუხოდ [53].

MVC ტექნოლოგიაში controller-კლასები მემკვიდრეობით მოდის System.Web.Mvc.Controller კლასიდან. Controller-კლასის თითოეული ღია მეთოდი (public method) წარმოადგენს ე.წ. Action-მეთოდს, რომელიც მარშრუტიზაციის სისტემით (ASP.NET Routing System) ასოცირებულია კონფიგურირებად URL-მისამართთან.

როდესაც Request-მოთხოვნა URL-მისამართით მიაკითხავს მასთან ასოციერებულ Action-მეთოდს, Controller-კლასში დაიწყება ოპერაციების ჩატარება დომეინ მოდელზე (Domain Model), რის შემდეგაც კონტროლერის Action-მეთოდი ჩატვირთავს HTML View-ობექტს და დაუბრუნებს Response-პასუხს. 2.3 ნახაზზე ასახულია ეს პროცესები.



ნახ.2.3.

ამგავარად, MVC მოდელით აპლიკაციის ლოგიკის ურთიერთ-გამიჯვნა ამარტივებს დასმული ამოცანის სირთულის მართვას, რადგან შესაძლებლობას იძლევა ერთ კონკრეტულ ასპექტზე გაკეთდეს ფოკუსირება. მაგალითად, მუშაობა შეიძლება view-კომპონენტზე ბიზნეს ლოგიკისგან დამოუკიდებლად. ეს ამარტივებს აპლიკაციის ტესტირების პროცესს.

გამარტივებულია პროგრამისტთა გუნდის მუშაობა: სხვადასხვა წევრს აქვს პარალელურად მუშაობის შესაძლებლობა: View-ლოგიკა, ბიზნეს ლოგიკა და კონტროლერის ლოგიკა შეიძლება იწერებოდეს დამოუკიდებლად, სხვადასხვა დეველოპერის მიერ.

2.4. ASP.NET MVC და Web Forms ტექნოლოგიების შედარება

➢ MVC ტექნოლოგიის უპირატესობები

- HTML Output-ის სრული კონტროლი – სტანდარტული ვებ-ფორმებისგან (Web Forms) განსხვავებით. MVC-აპლიკაცია არ იყენებს server control-ებს, როგორიცაა მაგალითად, Grid, Date Picker და სხვა. ეს სერვერული კონტროლები ინარჩუნებს თავიანთ მდგომარეობას დამალულ ველში, ე.წ. View State-ში, რომელზეც დეველოპერს კონტროლი ნაკლებად აქვს. სამომხმარებლო ინტერფეისი იქმნება სტანდარტული HTML-კონტროლებისგან, რაც სრული კონტროლის საშუალებას იძლევა [52];
- კოდის ურთიერთგამიჯვნა: Model-View-Controller მოდელი უზრუნველყოფს ფუნქციის გადანაწილებას სხვადასხვა კომპონენტში. Controller-ი ამუშავებს შემომავალ HTML-მოთხოვნებს, აქედან ხდება გარე ვებ-სერვისების გამოძახება (external web services), რათა შეაგროვოს ინფორმაცია Model-ობიექტში, განსაზღვრავს რომელი view უნდა ჩაიტვირთოს საპასუხოდ. ასეთი გამიჯვნა ამარტივებს ტესტირების პროცესს და იძლევა სუფთა კოდის წერის განხორციელების საშუალებას;
- პარალელური დეველოპმენტის საშუალება – Web Forms აპლიკაციისგან განსხვავებით, სადაც ვებ-ფორმის ლოგიკა code behind-ფაილში არის გატანილი და ართულებს ფუნქციაზე რამდენიმე პროგრამისტის ერთდროულად მუშაობას.

➢ MVC ტექნოლოგიის შეზღუდვები

- ფუნქციის ურთიერთგამიჯვნის გამო აპლიკაციის ლოგიკა გადანაწილებულია სხვადასხვა ფაილში და არ იყრის თავს მხოლოდ ერთ რომელიმე ფიზიკურ ფაილში (როგორც ეს ხდება Web-Forms ტექნოლოგიის შემთხვევაში);

- არ არის სერვერული მხარის მდიდარი სამომხარებლო ინტერფეისის კონტროლები (Rich Server Side User Interface Controls). ამის გამო რთული ფუნქციის და ინტერაქტიულობის მისაღწევად HTML სტანდარტულ კონტროლებზე პროგრმისტს თითქმის თავიდან უწევს ლოგიკის წერა [53];
- სხვადასხვა კონტროლის მდგომარეობის შენახვა-შენარჩუნება, Web-Forms აპლიკაციებთან შედარებით, რთული მისაღწევია, რადგან იქ server side-კონტროლების გამოყენება იძლეოდა ამის საშუალებას, ინარჩუნებდა რა თავის მდგომარეობას კონტროლი postback-ებს შორის (ViewState ობიექტში).

2.5. MVC აპლიკაციის აგება საპრობლემო სფეროსთვის „უნივერსიტეტი“

განვიხილოთ სასწავლო ვებ-აპლიკაცია, სახელწოდებით TechnicalUniversity, რომელიც დემონსტრირებას უკეთებს ASP.NET MVC-5 პლატფორმისა და Entity Framework-6 გამოყენებას Visual Studio 2013 სამუშაო გარემოში [64].

პროგრამული უზრუნველყოფა შემუშავებულია „Code First” მიდგომით თავდაპირველად იწერება პროგრამული კოდი, შემდეგ იქმნება მონაცემთა ბაზა).

სასწავლო აპლიკაცია წარმოადგენს „უნივერსიტეტის“ ვებ-საიტის იმიტაციურ მოდელს. საიტზე გათვალისწინებულია სტუდენტების, ლექტორების, სასწავლო საგნების, ფაკულტეტებისა და სასწავლო ჯგუფების აღრიცხვა, აგრეთვე სტატისტიკური ინფორმაციის დაანგარიშება სტუდენტების განაწილების შესახებ.

აპლიკაციის საწყისი გვერდი ინფორმაციული შინაარსისაა და აღწერს პროგრამული უზრუნველყოფის არქიტექტურასა და ტექნიკურ მახასიათებლებს (ნახ.2.4). რადგან ვებ-საიტი სასწავლო

დანიშნულებას ატარებს, იგი მომხმარებელს სთავაზობს პროგრამული ლოგიკის კოდს და URL მისამართს, საიდანაც პროექტის ჩამოტვირთვა არის შესაძლებელი.

ნახ.2.4. აპლიკაციის საწყისი გვერდი

აპლიკაციის დანარჩენ სექციებში თემატურად გადანაწილებულია ინფორმაცია უნივერსტეტის წევრებისა და სასწავლო პროცესების შესახებ.

მაგალითისთვის, განვიხილოთ ლექტორების ნუსხა (ნახ.2.5). ზედა ცხრილი წარმოადგენს უნივერსიტეტში დაქირავებულ ლექტორთა ინფორმაციას. ჩანაწერის მონიშვნის შემდეგ ცხრილის ქვემოთ გააქტიურდება მეორე ცხრილი, სადაც ნაჩვენებია მონიშნული ლექტორის სასწავლო საგნების სია.

თითოეულ საგანს აქვს მოსანიშნი, რომლის არჩევის შემთხვევაში გამოჩნდება მესამე ცხრილი, სადაც გამოდის მონიშნულ სასწავლო საგანზე დარეგისტრირებული სტუდენტების და მათი აკადემიური შეფასებების ჩამონათვალი.

ნახ.2.5. ლექტორების რეესტრი
(Views/Instructor/Index.cshtml)

- გამოყენებული პროგრამული პაკეტები:
  - Visual Studio 2013
  - .NET 3.5
  - Entity Framework 6 (EntityFramework 5.1.0 NuGet package)
  - Windows Azure SDK 2.2 (არ არის სავალდებულო)

➢ აპლიკაციის დომეინი (Data Model)

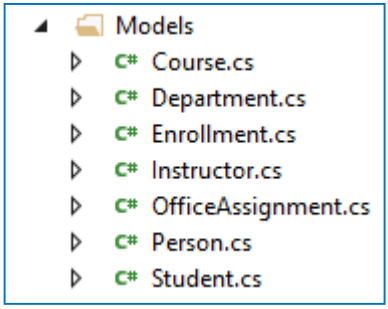
2.6 ნახაზზე ნაჩვენებია მონაცემთა მოდელის სტრუქტურა, რომელიც შედგება შემდეგი ცხრილებისგან:

- Instructor - ლექტორების სიმრავლე;
- Student - სტუდენტების სიმრავლე;
- Course - სასწავლო კურსები;
- Enrollment - სასწავლო ჯგუფები;
- Grade - შეფასებები (enum ტიპის მნიშვნელობები A,B,C,D,F);
- Department - სასწავლო ფაკულტეტები;
- OfficeAssignment - მისამართები და ადგილმდებარეობები.

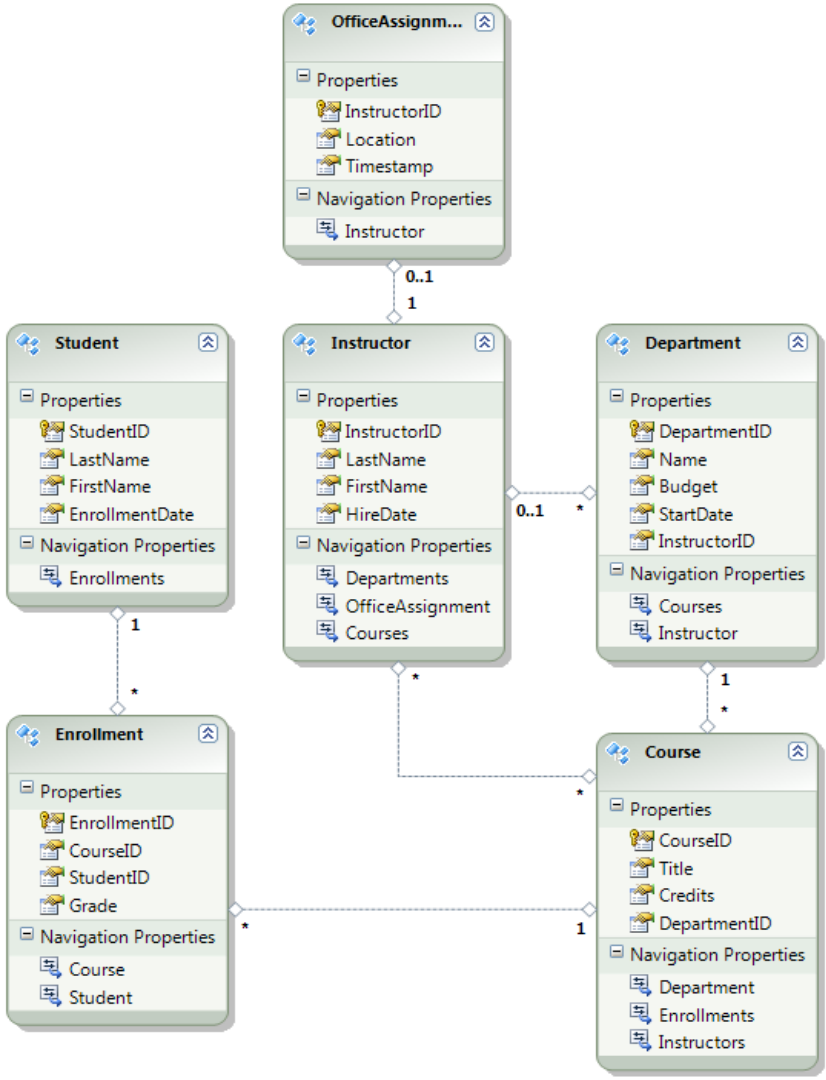
➢ MVC აპლიკაციის ლოგიკური ორგანიზება

TechnicalUniversity ვებ-აპლიკაციის არქიტექტურა MVC სტანდარტების მიხედვით არის აგებული. პროექტი დაყოფილია ლოგიკურ ნაწილებად და პროგრამული კოდის ფაილები ორანიეზებულია საქაღალდეებში: Models, Controllers და Views.

- **Models საქაღალდე**



ნახ.2.7. Models საქაღალდეში გაერთიანებულია აპლიკაციის Model-კლასები



ნახ.2.6. აპლიკაციის მონაცემთა მოდელის სტრუქტურა
(Entity Framework 6)

Model კლასები ვიზუალურად გაერთიანებულია Model საქაღალდეში, ხოლო პროგრამულად - TechnicalUniversity.Models ბიბლიოთეკაში.

მოდელის კლასებში ხდება ვალიდაციების და წესების განსაზღვრა, მონაცემთა ფორმატირების გაწერა (FormatString), სავალდებულოობის (Required) მითითება. მაგალითისთვის, ასეთი შეზღუდვა: „პიროვნების სახელის მითითება სავალდებულოა და არ შეიძლება 50 სიმბოლოზე მეტი იყოს“ (ნახ.2.8).

ეს შეზღუდვა იმპლემენტირებულია [StringLength] ატრიბუტით, ხოლო თარიღის გამოსატანი ფორმატი კი - ატრიბუტით [DisplayDateFormat] (ნახ.2.9).

```
[Required]
[StringLength(50, ErrorMessage = "სახელი არ შეიძლება 50 სიმბოლოზე მეტი იყოს.")]
[Column("FirstName")]
[Display(Name = "სახელი")]
23 references
public string FirstMidName { get; set; }
```

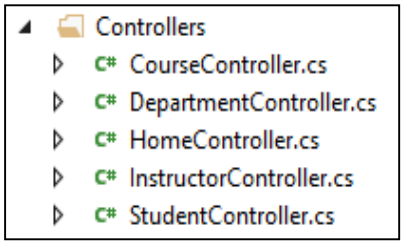
ნახ.2.8. ვალიდაცია

```
26 references
public class Instructor : Person
{
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
                   ApplyFormatInEditMode = true)]
    [Display(Name = "დაქირავების თარიღი")]
    5 references
    public DateTime HireDate { get; set; }
```

ნახ.2.9. თარიღის ფორმატირება

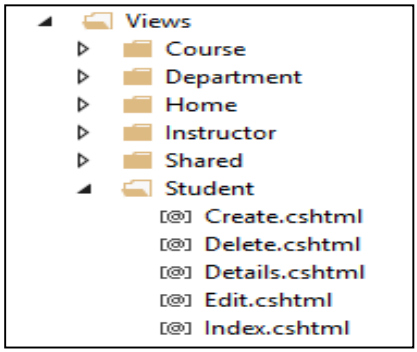
- **Controllers საქაღალდე**

TechnicalUniversity პროგრამის ძირითადი ლოგიკური ერთეულებია: კურსი (CourseController.cs), ფაკულტეტი (DepartmentController.cs), ლექტორი (InstructorController.cs), სტუდენტი (StudentController.cs). ამ ობიექტებზე დაშვებული ფუნქცია განსაზღვრულია Controllers საქაღალდეში გაერთიანებულ Controller-კლასებში. მაგალითად, ახალი სტუდენტის დამატება, რედაქტირება, წაშლა და დათვალიერება - ყველა ეს ფუნქცია რეალიზებულია StudentController.cs კონტროლერ ფაილში (ნახ.2.9).



ნახ.2.10. Controllers საქაღალდეში თავმოყრილია აპლიკაციის ყველა Controller-კლასი

- **Views საქაღალდე**



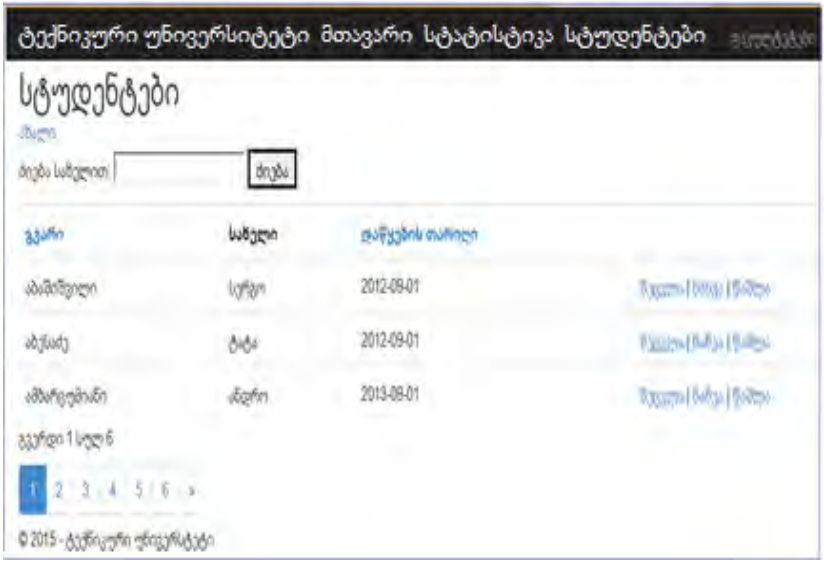
ნახ.2.11. View საქაღალდეში თავმოყრილია აპლიკაციის ყველა View ფაილი (cshtml)

პრეზენტაციის დონის წარმოდგენა (View) კიდევ ერთი მნიშვნელოვანი განმასხვავებელი მომენტია ASP.NET Web Forms ტექნოლოგიასა და ASP ტექნოლოგიას შორის. Web Forms აპლიკაციებში პრეზენტაციის დონეზე გვხვდება .aspx გაფართოების ფაილები, ხოლო პროგრამული ლოგიკა გატანილია .aspx-ფაილების უკანა მხარეს - .aspx.cs ფაილებში (code behind).

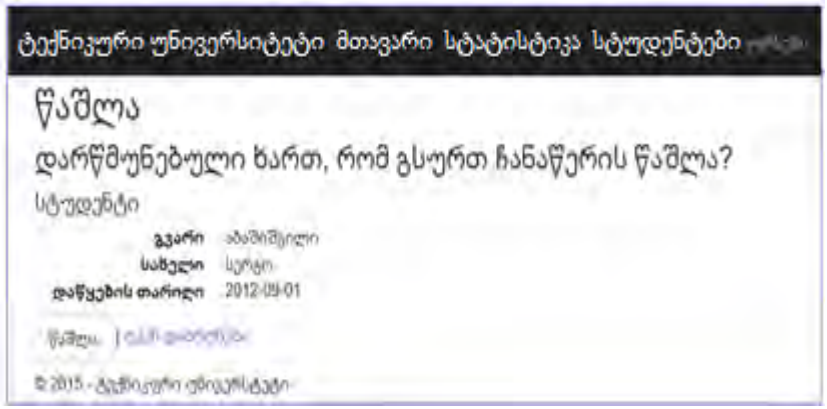
MVC-აპლიკაციებში პრეზენტაციის დონე რეალიზებულია .cshtml გაფართოების ფაილებით, ხოლო პროგრამული ლოგიკა გატანილია Controllers დონეზე. ამასთანავე .cshtml ფაილების რაოდენობა დამოკიდებულია შესასრულებელი ლოგიკური ოპერაციების რაოდენობაზე. მაგალითად სტუდენტების რეესტრი იმპლემენტირებულია Students საქაღალდეში გაერთიანებული .cshtml View-ების ერთობლიობით (ნახ.2.12).

თითო ოპერაციისთვის განკუთვნილია თითო ფაილი (ნახ.2.13, 2.14):

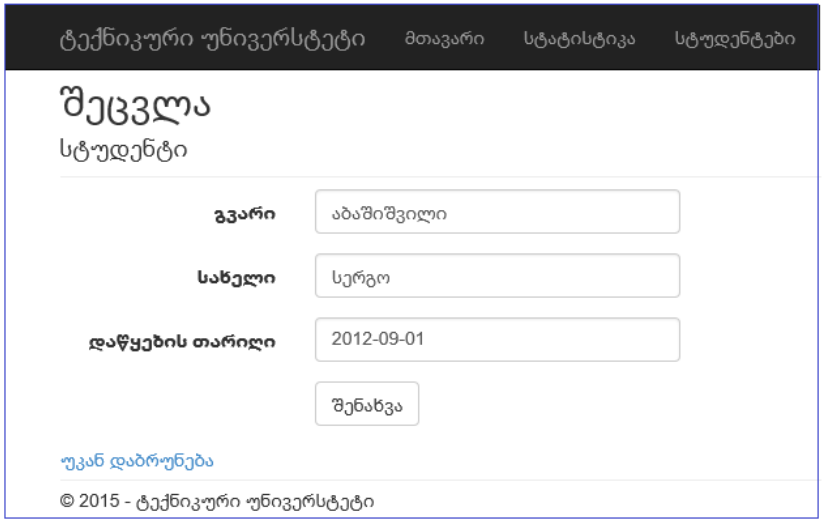
- Create.cshtml - ახალი სტუდენტის დამატების ფორმა;
- Delete.cshtml - სტუდენტის წაშლის ფორმა;
- Details.cshtml - სტუდენტის დეტალური ინფორმაციის ამსახველი ფორმა;
- Edit.cshtml - სტუდენტის ინფორმაციის რედაქტირების ფორმა;
- Index.cshtml - მთავარი View - სტუდენტების სია. თითოეული სტუდენტის ჩანაწერის გასწვრივ დატანილია ღილაკები, რომელზე დაჭერისას მომხმარებელი მოხვდება დანარჩენ .cshtml View-ებზე.



ნახ.2.12. სტუდენტების სია
(Index.cshtml)



ნახ.2.13. სტუდენტების წაშლის ფორმა
(Delete.cshtml)

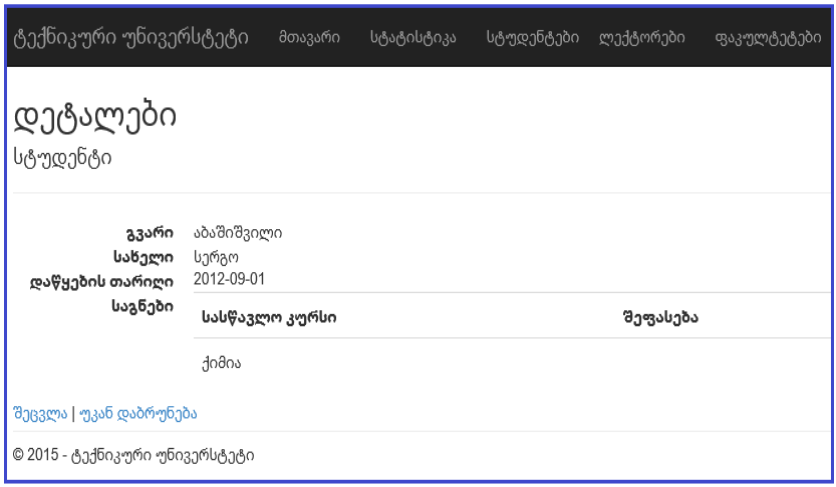


ნახ.2.14. სტუდენტის რედაქტირების გვერდი (Edit.cshtml)

```
@model TechnicalUniversity.Models.Student
@{ ViewBag.Title = "Details";}
<h2>დეტალები</h2>
<div>
    <h4>სტუდენტი</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt> @Html.DisplayNameFor(model => model.LastName) </dt>
        <dd> @Html.DisplayFor(model => model.LastName) </dd>
        <dt> @Html.DisplayNameFor(model => model.FirstMidName) </dt>
        <dd> @Html.DisplayFor(model => model.FirstMidName) </dd>
        <dt> @Html.DisplayNameFor(model => model.EnrollmentDate) </dt>
        <dd> @Html.DisplayFor(model => model.EnrollmentDate) </dd>
        <dt> საგნები </dt>
        <dd>
            <table class="table">
                <tr>
                    <th>სასწავლო კურსი</th>
                    <th>შეფასება</th>
                </tr>
                @foreach (var item in Model.Enrollments)
                {
                    <tr>
                        <td>
                            @Html.DisplayFor(modelItem => item.Course.Title)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.Grade)
                        </td>
                    </tr>
                }
            </table>
        </dd>
    </dl>
</div>
<p>
    @Html.ActionLink("შეცვლა", "Edit", new { id = Model.ID }) |
    @Html.ActionLink("უკან დაბრუნება", "Index")
</p>
```

ნახ.2.15. დეტალური ინფორმაციის
გვერდის კოდი

როგორც ნახაზიდან ჩანს, .cshtml ფაილში HTML კოდი საკმაოდ სუფთად და მარტივად არის წარმოდგენილი. ფაილის წონა მსუბუქია და იოლად მართვადი, რაც MVC აპლიკაციების ერთ-ერთი მთავარი ღირსებაა.



ნახ.2.2.16. სტუდენტის დეტალური ინფორმაციის გვერდი (Details.cshtml)

2.6. MVVM არქიტექტურული მოდელი

განვიხილოთ პროგრამული უზრუნველყოფის შექმნის კიდევ ერთი არქიტექტურული სტანდარტი Model-View-View- Model (MVVM). იგი გამოიყენება პროგრამირების თანამედროვე პლატფორმებში: Windows Presentation Foundation (WPF) და მოგვიანებით ათვისებულ იქნა Silverlight-პლატფორმაზეც [50,51].

- Model – დომეინ ობიექტი. MVVM არქიტექტურული სტანდარტის Model-ის განსაზღვრება ანალოგიურია MVC არქიტექ-ტურული სტანდარტის Model-ის განსაზღვრებისა: მოდელი

წარმოადგენს მონაცემთა სიმრავლეს, რომელზეც მუშაობს პროგრამა (მაგალითად, კლიენტი, საკონტაქტო ინფორმაცია, საწყობი და ა.შ.). იგი არის რეალური სამყაროს ასახვა ვებ-აპლიკაციაში. მოდელი შედგება მხოლოდ ამ მონაცემებისგან და არა სხვა ლოგიკისგან, როგორიცაა სერვივისები, მონაცემთა მართვის ლოგიკა (ბიზნეს ლოგიკის ფუნქცია). Model-ის შრეს არ გააჩნია ინფორმაცია View და ViewModel-შრეების შესახებ [46];

- View – სამომხმარებლო ინტერფეისი, სამუშაო გარემო. ამ დონეზე მონაცემთა პრეზენტაცია ხდება XAML markup-ის საშუალებით. იგი ასევე შეიცავს სამომხმარებლო ინტერფეისისთვის საჭირო რესურსებს (resources), სტილებს (style), მონაცემთა შაბლონებს (data template). ამ დონეზე ხდება მონაცემთა უკეთ აღქმადობის მიზნით მათი ფორმატირება, შეტანილი ინფორმაციის აღქმა (ღილაკზე დაჭერა, მაუსის მოძრაობის აღქმა და რეარგირება). შედგება ე.წ. behaviors, data binding და event. არ საჭიროებს მდგომარეობის შენახვა/შენარჩუნებას (maintain state), რადგან მუდმივად სინქრონიზდება viewModel-თან;
- ViewModel – მომხმარებელთან ურთიერთკავშირის ლოგიკაა, რაც უზრუნველყოფილია მისი შემადგენელი ობიექტებით: Properties, Commands, Observable Commands, Methods, Validations, რომლებიც View-ნაწილთან მჭიდრო კავშირშია. ViewModel-შრე კავშირშია Model დონესთან და მის ინფორმაციას ასახავს View დონისთვის ხელმისაწვდომი სახით. ViewModel იმპლემენტაცია უნდა გაკეთდეს ისე, რომ იგი პირდაპირ კავშირში არ იყოს view დონესთან.

მთავარი ნაწილი, უზრუნველყოფს view და model-შრეების ურთიერთგამოჯვნას, რის შედეგადაც Model ნაწილმა არ იცის View-ს შესახებ და პასუხს აგებს მხოლოდ მონაცემთა შენახვაზე, ხოლო View ნაწილის მოვალეობა არის ფორმატირებული მონაცემების ასახვა.

2.7. ASP.NET Silverlight ტექნოლოგია

Silverlight არის Microsoft-ის ტექნოლოგია. იგი web-პლატფორმაა, რომელიც გამოიყენება ე.წ. „Rich Internet Applications" (RIA) შესაქმნელად. Silverlight-ს აქვს კარგი მხარდაჭერა სხვა ტექნოლოგიების მიერ, Microsoft-ი განაგრძობს მის გამოყენებას მთელ რიგ სხვა პროდუქტებთან ერთობლიობაში (როგორიცაა Windows Azure, Microsoft Lync, Windows Phone 7 აპლიკაციების დეველოპმენტი) [46].

მიუხედავად იმისა, რომ Silverlight-აპლიკაციები ხშირად ვებ-ბრაუზერში არის გაშვებული, ვერსია 3-იდან მოყოლებული, აპლიკაცია შეიძლება ბრაუზერის გარეთაც იქნეს გატანილი Out-of-Browser (OOB) – დესკტოპზე/ან Start Menu-ში იკონის მოთავსებით.

აპლიკაცია ავტომატურად დააფდეითდება, როდესაც სერვერზე ხელმისაწვდომი იქნება ახალი ვერსია.

➢ Silverlight ტექნოლოიის უპირატესობები

ბიზნეს აპლიკაციების შექმნა სხვადასხვა პლატფორმაზე შეიძლება, მაგრამ გარკვეული თვისებების და ფუნქციების გამო ბიზნეს აპლიკაციების დეველოპმენტისას უპირატესობა შეიძლება მიენიჭოს Silverlight ტექნოლოგიას:

- დეველოპმენტის მოდელი მარტივია – კლიენტს აპლიკაციაზე წვდომა აქვს ვებ-ბრაუზერის საშუალებით;
- არ არის აუცილებელი .NET Framework-ის სრული პაკეტის დაყენება, მხოლოდ მცირე ზომის runtime-პაკეტის ინსტალაცია არის საჭირო [46];
- Silverlight-აპლიკაციის წერა შეიძლება ნებისმიერ .Net ენაზე (C#, Visual Basic და ა.შ.);
- Silverlight-აპლიკაცია თავსებადია სხვადასხვა პლატფორმასთან (Windows, Mac);

- HTML-აპლიკაციებთან შედარებით უფრო გაიოლებულია დეველოპმენტის პროცესი.

➢ **როდის აჯობებს Silverlight ტექნოლოგიის გამოყენება ?**

ტექნოლოგიის არჩევისას უნდა ვიხელმძღვანელოთ ობიექტურად და არა მხოლოდ იმ მოტივით, რომ ესა თუ ის ტექნოლოგია „მოდურია" ან „მაგარია".

უნდა გავითვალისწინოთ – არჩეული ტექნოლოგია რამდენად სრულყოფილად დააკმაყოფილებს დასმულ ამოცანას, ბიზნეს-მოთხოვნებს. სხვა ტექნოლოგიების მსგავსად, Silverlight ტექნოლოგიასაც აქვს თავისი ძლიერი და სუსტი მხარეები, რომელიც დეველოპმენტის პროცესის დაწყებამდე წინასწარ უნდა იქნეს განხილული და შეფასებული [46].

➢ **ASP.NET-ტექნოლოგიასთან მიმართებაში შედარება**

- საერთო ფუნქცია:
  - ორივე: ASP.NET და Silverlight ტექნოლოგია ეშვება ვებ-ბრაუზერში.

ASP.NET ტექნოლოგიის **უპრატესობანი** Silverlight-ტექნოლოგიასთან მიმართებაში:

- ASP.NET არ საჭიროებს კლიენტის მანქანაზე plugin-ის ინსტალაციას;
- სრული .NET Framework პაკეტი (ოღონდ server-ის მხარეს მხოლოდ);
- უფრო ხანგრძლივი გამოყენების პრაქტიკა (მოყოლებული .NET Framework-ის შემოსვლიდან);
- ეშვება ინტერნეტთან დაკავშირებულ თითქმის ყველა მოწყობილობაზე;

ASP.NET ტექნოლოგიის **უარყოფით მხარეები** Silverlight-ტექნოლოგიასთან მიმართებაში;

- HTML-ტექნოლოგიით შექმნილი აპლიკაციები სხვადასხვა ვებ-ბრაუზერში და ოპერაციულ სისტემაზე სხვადასხვაგვარად რენდერდება. Silverlight-აპლიკაცია კი ყოველთვის ერთნაირად აისახება. ეს დიდ დროს ზოგავს, რაც გამოწვეულია cross-browser ტესტირების აუცილებლობით [46];
- რადგან Silverlight კლიენტის მხარეს სრულდება, მომხმარებელს არ უწევს ხშირად ლოდინი Server-ის postback-ებზე. ამის გამო Silverlight-აპლიკაცია უფრო ინტერაქტიულია;
- CLR და .NET Framework-ის ქვესიმრავლე გაშვებულია კლიენტის კომპიუტერზე, ამიტომ Silverlight-აპლიკაცია აღარ საჭიროებს დამატებით JavaScript კოდის წერას;
- სამომხმარებლო ინტერფეისის დეველოპმენტისთვის არსებობს საკმაოდ ეფექტური ხელსაწყოები (Visual Studio, Expression Blend - ერთერთი ყველაზე საუკეთესო და გავრცელებული, თავსებადი ყველა პლატფორმასთან);
- აპლიკაციების გაშვება შესაძლებელია offline-რეჟიმში.

➢ **Windows Forms ტექნოლოგიასთან მიმართებაში შედარება**

საერთო ფუნქცია:

- ორივე გადმოწერადია ვებ-ბრაუზერიდან (თუმცა Windows Forms-აპლიკაციები Silverlight-აპლიკაციებისგან განსხვავებით ვებ-ბრაუზერში ვერ გაეშვება) და თვითგანახლებადია ავტომატურ რეჟიმში, როგორც კი სერვერზე ახალი ვერსია გახდება ხელმისაწვდომი;

Windows Forms ტექნოლოგიის **უპრატესობანი** Silverlight-ტექნოლოგიასთან მიმართებაში:

- მინიმალური შეზღუდვები (არ ხდება Sendboxing);

- სრული .NET Framework პაკეტი;
- სამომხმარებლო ინტერფეისის გამარტივებული დეველოპმენტი – არ საჭიროებს XAML-ის საფუძვლიან ცოდნას;
- უფრო ხანგრძლივი გამოყენების პრაქტიკა (მოყოლებული .NET Framework-ის შემოსვლიდან).

Windows Forms ტექნოლოგიის **უარყოფით მხარეები** Silverlight-ტექნოლოგიასთან მიმართებაში:

- Windows Forms ტექნოლოგია საჭიროებს სრული .NET Framework პაკეტის ინსტალაციას;
- შეზღუდულია პლატფორმის მხრივ (Windows Forms-აპლიკაციები ეშვება მხოლოდ Windows ოპერაციულ სისტემაზე);
- Silverlight ტექნოლოგია უფრო მდიდარია data binding ფუნქციებით, ვექტორული გრაფიკით, ანიმაციებით;
- Silverlight ტექნოლოგია Windows Forms ტექნოლოგიასთან შედარებით უზრუნველყოფს უფრო მოქნილ სამომხმარებლო ინტერფეისს და styling-ფუნქციას [46].

2.8. Web-აპლიკაციაში რეპორტების ინტეგრაციის მეთოდები

თანამედროვე, მაღალკონკურენტულ ბიზნეს გარემოში ინფორმაციის ფლობას გადამწყვეტი მნიშვნელობა ენიჭება. ტექნოლოგიის განვითარებასთან ერთად ინფორმაციის შეგროვება გაიოლდა, თუმცა მისი გაანალიზება კვლავ რჩება რთულ და ფაქიზ თემად.

ბიზნეს ანალიზის და ანგარიშგებათა (რეპორტების) შემუშავებისთვის გადამწყვეტი ფაქტორია კარგი, მოქნილი სამუშაო ინსტრუმენტების (Business Intelligence Tools) არსებობა და მათი ეფექტური გამოყენება [1,31,55,56].

- **SQL Server Reporting Services**

SQL Server Reporting Services (SSRS) არის კომპანია Microsoft-ის ინსტრუმენტი, რომელიც შემუშავებულია მონაცემთა ანალიზისა და ანგარიშგებების გენერაციის მიზნით [57,58]. იგი ბიზნეს ანალიზის პლატფორმის (BI) ერთ-ერთი კომპონენტია და მთლიანობაში იძლევა მონაცემთა ანალიზის მოქნილ საშუალებებს. ეს კომპონენტებია:

- SQL Server: ტრადიციული მონაცემთა ბაზის მანქანა, რომელზეც ასევე ინახება SSRS რეპორტების კატალოგი;
- SQL Server Analysis Services (SSAS): კომპონენტი ასრულებს ისეთ ანალიზურ პროცესებს, როგორიცაა მონაცემთა აგრეგაცია და წარმოდგენა სხვადასხვა ჭრილში (მაგალითად, გეოგრაფიული მდებარეობა, დრო);
- SQL Server Integration Services (SSIS): კომპონენტი გამოიყენება მონაცემთა ამოსაღებად, ტრანსფორმირების და ჩატვირთვის მიზნით (Extract, Transform, Load - ETL);
- SQL Server Reporting Services (SSRS): სერვერზე დაფუძნებული, განვრცობადი პლატფორმაა, რომელშიც ხდება ინფორმაციის დამუშავება, ფორმატირება და მომხმარებლისთვის ტრადიციული თუ ინტერაქტიული ფორმატით მიწოდება. SSRS კონფიგურირებადია და ხასიათდება მრავალი ფუნქციით, როგორიცაა მაგალითად, მონაცემთა ვიზუალიზაცია დიაგრამებისა და გრაფიკების სახით, ანგარიშგებათა სხვადასხვა ფორმატით ექსპორტირება (HTML, Excel, PDF), მომხმარებლის ელ-ფოსტაზე გადაგზავნა, დაკონფიგურირება და SharePoint კორპორატიულ პორტალებში ჩაშენება.

- **რეპორტების ინტეგრაცია Web-აპლიკაციებში**

Microsoft-ის კომპანიამ რეპორტინგის სერვისის (Reporting Services) დაპროექტებისას თავიდანვე გაითვალისწინა მისი

განვრცობადობის აუცილებლობა და სერვისის ღია ინტერფეისით წვდომადი გახადა ვებ-აპლიკაციებისა და დაპროგრამების სამუშაო გარემოთა ფართო სპექტრი [59].

განვიხილოთ რეპორტინგის ფუნქციით აპლიკაციის გამდიდრების 3 გავრცელებული ვარიანტი:

- Reporting Server Web Service (ასევე ცნობილია როგორც Reporting Services SOAP API);
- ReportViewer კონტროლი;
- წვდომა URL გზავნილის საშუალებით.

○ *Reporting Server Web Service*

ესაა რეპორტინგის ფუნქციასთან სამუშაო ბაზისური ინტერფეისი. სერვისი უზრუნველყოფს პროგრამისტს აპლიკაციაში რეპორტინგის ფუნქციის ჩაშენების ყველა საჭირო მეთოდითა და კონტროლით. მაგალითად, Report Manager არის აპლიკაცია, რომელიც მოყვება რეპორტინგ სერვისებს და იყენებს Web-სერვისს რეპორტ სერვერის მონაცემთა ბაზის სამართავად. რადგან SOAP API წარმოადგენს ვებ-სერვისს, იგი .NET Framework-ში წვდომადია, სხვა SOAP სერვისების მსგავსად [59]. Proxy-კლასების გენერაციით შესაძლებელია Report Server ვებ-სერვისის მეთოდებზე წვდომა და მათი გამოყენება.

○ *ReportViewer კონტროლი (Visual Studio)*

ReportViewer კონტროლი მოყვება Visual Studio ინსტრუმენტს და აქვს აპლიკაციიდან რეპორტის გაშვების დანიშნულება (ნახ.2.17). არსებობს კონტროლის ორი რეალიზაცია: ერთი - რომელიც მუშაობს Windows Forms აპლიკაციებთან, ხოლო მეორე - Web Forms აპლიკაციებთან. თითოეული კონტროლი უზრუნველყოფილია Report Server-ზე ატვირთული რეპორტების გაშვების და სხვადასხვა

ფორმატში ექსპორტირების ფუნქციონალით (მოსახერხებელია მომხმარებლისთვის, რომლის კომპიუტერზე არაა დაინსტალირებული რეპორტ სერვერი).

ნახ.1.1.17. ReportViewer კონტროლი ვებ-აპლიკაციაში

ReportViewer კონტროლს ახასიათებს ორი რეჟიმი:

- Remote Processing Mode – Report Server სერვერზე ატვირთული რეპორტების ჩვენების რეჟიმი. რეპორტის დამუშავება ხდება სერვერზე. მუშაობის პროცესში იგი იძლევა რამდენიმე სერვერის დატვირთვის ან შესასრულებელი გამოთვლითი სამუშაოების რამდენიმე პროცესორზე გადანაწილების შესაძლებლობას;
- Local Processing Mode – რეპორტების გაშვების ალტერნატიული მეთოდი, როდესაც Reporting Servises სერვისი არ არის დაყენებული სამუშაო მანქანაზე. Remote Processing Mode რეჟიმისგან განსხვავებით, კონტროლში მხოლოდ რეპორტ სერვერის ფუნქციის გარკვეული ქვესიმრავლეა ხელმისაწვდომი. ამ რეჟიმში რეპორტის მონაცემთა სიმრავლის დამუშავება (Data Processing) ReportViewer კონტოლით აღარ ხდება, არამედ თვითონ ვებ-აპლიკაციაშია რეალიზებული, თუმცა კი რეპორტის დამუშავებას (Report Processing) კვლავ კონტროლი ასრულებს.

- *წვდომა URL მისამართით*

URL მისამართით რეპორტებზე წვდომა წარმოადგენს ვებ-აპლიკაციებიდან რეპორტების დათვალიერების კიდევ ერთ მეთოდს და გამოიყენება როდესაც ReportViewer კონტროლი არ არის ხელმისაწვდომი. URL მისამართით წვდომა მოსახერხებელია მომხმარებლისთვის ელ-ფოსტაზე რეპორტების გზავნილების გადასაცემად (ნახ.2.18).

```
<a href="http://server/reportserver?/EmployeeReports/EmployeeList
Drilldown&rs:Command=Render&rc:LinkTarget=main" target="main">
    თანამშრომლების შესახებ ინფორმაციის ნახვა
</a>
```

ნახ.2.18. Web -საიტიდან რეპორტზე URL გზავნილით მიმართვა

- **სერვერული მხარის და ლოკალურად ჩაშენებული რეპორტების შედარება**

რეპორტის ვიზუალური მხარის აწყობა შესაძლებელია Report Server პროექტში, რომელიც მოყვება SQL Server თანამედროვე ვერსიებს და ასევე ინტეგრირებულია Visual Studio პროგრამაში [58]. დიზაინერი (Report Designer) რეპორტის ასაწყობი გრაფიკული გარემოა. დიზაინის აწყობის შემდეგ მომხმარებლისთვის ხელ-მისაწვდომი ხდება რეპორტის ატვირთვა Report Server სერვერზე ან ვებ-აპლიკაციაში მისი ლოკალურად ჩაშენების შესაძლებელობა.

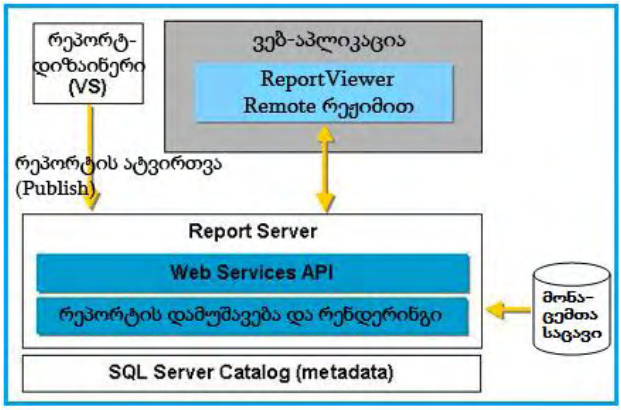
- *სერვერული მხარის რეპორტები*

Report Server სერვერზე ატვირთული რეპორტი არის სტანდარტული (.rdl) გაფართოების ფაილი, რომელიც შეიცავს ინფორმაციას მონაცემთა ბაზასთან დაკავშირებისა და მონაცემთა ამოღების შესახებ.

სერვერულ რეჟიმში ინტეგრირებული რეპორტების შემთხვევაში ვებ-აპლიკაცია უკავშირდება Report Server-ზე ატვირთულ ანგარიშგებას, ხოლო რეპორტის მონაცემებით შევსებას, დამუშავებას და დარენდერებას ასრულებს სერვერი.

Report-სერვერი უზრუნველყოფილია სხვადასხვა სერვისით, როგორიცაა უსაფრთხოება, ისტორიის შენახვა, ანგარიშგებების ელ-ფოსტაზე გამოწერა და ა.შ. (ნახ.2.19).

როდესაც ვებ-აპლიკაციაში რეპორტი ინტეგრირებულია სერვერულ რეჟიმში, მაშინ SQL Server Report Server ლიცენზია სავალდებულოა.



ნახ.2.19. სერვერული მხარის რეპორტის დამუშავების პროცესი

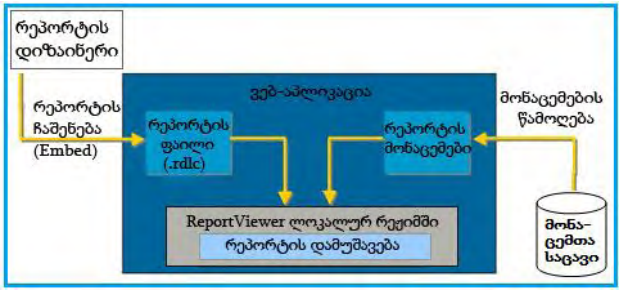
- *ლოკალურად ჩაშენებული რეპორტები*

ვებ-აპლიკაციაში ლოკალურად ჩაშენებული რეპორტი .rdlc გაფართოების ფაილია, რომელიც მოიცავს DataSource ობიექტების შესახებ მეტა-მონაცემებს, მაგრამ არ შეიცავს SQL-სერვერთან კავშირის ან Query პროგრამული კოდის შესახებ ინფორმაციას.

ჩაშენებული რეპორტების ფუნქციის სარეალიზაციოდ Visual Studio პროგრამას მოყვება ReportViewer კონტროლი. მისთვის

სავალდებულოა სამუშაო მანქანაზე დაინსტალირებული იყოს .NET Framework 2.0 ან შემდგომი ვერსიები.

ლოკალურ რეჟიმში ვებ-აპლიკაციის მხარეს ხდება რეპორტის მონაცემებით შევსება. სერვერული რეპორტებისგან განსხვავებით SQL-ლიცენზია არ არის სავალდებულო. რეპორტის დასამუშავებლად საჭირო ფუნქცია უზრუნველყოფილია Report Viewer კონტროლში (ნახ.2.20).



ნახ.2.20. ლოკალური რეპორტის დამუშავების პროცესი

ლოკალურ რეჟიმში რეპორტის სამომხმარებლო ინტერფეისის უზრუნველყოფა და პარამეტრების გადაწოდება აპლიკაციის მხარეს ევალება. რეპორტის მონაცემთა წყაროს როლში შესაძლებლია ADO.NET DataTable ობიექტების გამოყენება.

ლოკალურად ჩაშენებული რეპორტების შემთხვევაში უსაფრთხოების ფუნქციის უზრუნველყოფა ვებ-აპლიკაციის მხარეს გადაინაცვლებს. რეპორტში ჩაშენებული კოდი არ არის უფლებამოსილი ფაილური სისტემის წვდომასა თუ ქსელურ გარემოსთან მუშაობაზე (თუ ცხადად არ აქვს მინიჭებული უფლებები - explicit permissions).

- ***დასკვნა: რეჟიმის გამოყენების შესახებ***

ლოკალური რეჟიმი სერვერულ რეჟიმთან შედარებით ნაკლებად მოქნილია და განკუთვნილია მცირე ან საშუალო ზომის

რეპორტების გენერაციისთვის (რადგან გამოთვლით პროცესები და რეპორტის გენერაცია კლიენტის მანქანაზე სრულდება), stand-alone ტიპის აპლიკაციებში, რომლებიც არ საჭიროებს Report Server-ს.

სერვეული რეპორტების გამოყენება რეკომენდებულია მრავალი მომხმარებლის ერთდროული მუშაობის რეჟიმის შემთხვევაში, რთული და კომპლექსური Query სკრიპტებისგან შემდგარი დიდი მოცულობის მონაცემების ანგარიშგებათა გენერაციის დროს.

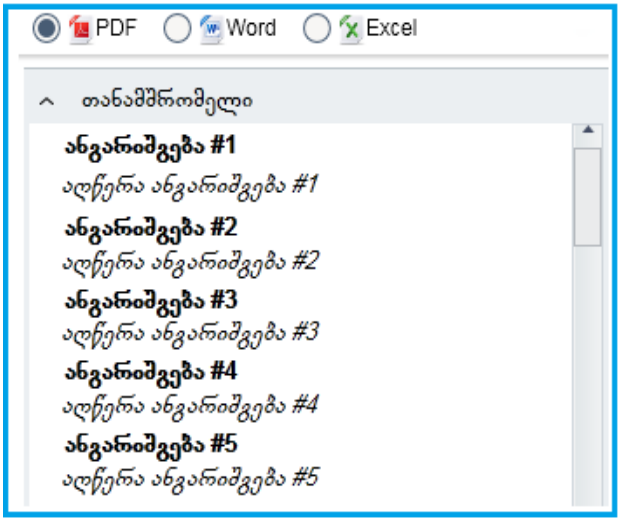
❖ რეპორტების გენერაცია მომხმარებლისთვის ადვილად გამოყენებადი ინტერფეისიდან

განვიხილოთ მაგალითი, სადაც ServerReport კლასის და ReportViewer კონტროლის გამოყენებით ხდება კავშირის დამყარება Reporting Services სერვერთან და Render() მეთოდის გამოყენებით ექსპორტირება სხვადასხვა ფორმატში.

ეს მეთოდი მოსახერხებელია, როდესაც არ გვსურს მომხმარებლის ინტერფეისში ReportViewer კონტროლის გამოტანა, მაგრამ გვჭირდება რეპორტის გენერაცია და ექსპორტირება სხვადასხვა ფორმატით.

მომხმარებლის ინტერფეისზე გამოტანილია სისტემაში არსებული რეპორტების სია (ნახ.2.21).

რეპორტის დასახელება წარმოადგენს გზავნილს, რომლის არჩევითაც ხდება რეპორტის გენერაციის მეთოდის გამოძახება, ამორჩეულ ფორმატში (მაგალითად, PDF) გადაყვანა და დაბრუნებული ბაიტების მასივის ჩაწერა დროებით ფაილში.



ნახ.2.21. ანგარიშგებები: სამომხმარებლო ინტერფეისის მხარე

შემდეგ, მომხმარებელს შეუძლია ფაილი ჩამოტვირთოს საკუთარ კომპიუტერზე ან გახსნას ეკრანზე.

პრერეკვიზიტები:

1. Microsoft Report Viewer 2012 Runtime პაკეტის გადმოწერა და ინსტალაცია. პაკეტის საინსტალაციო ფაილის დასახელება არის ReportViewer.msi და .NET Framework-ზე დაწერილი აპლიკაციების-თვის Microsoft Reporting ტექნოლოგიით შემუშავებულ ანგარიშ-გებათა გაშვების ფუნქციას შეიცავს;
2. ვებ-აპლიკაციის პროექტში გზავნილების (References) საქაღალდეში Microsoft.ReportViewer.WinForms.dll-ის დამატება.

შედეგ ბიჯზე ვქმნით SaveReportToTempFile() მეთოდს, რომელიც აგენერირებს რეპორტს და აბრუნებს მას ბაიტების მასივის (byte[]) სახით.

მეთოდის პარამეტრები:

- ანგარიშგების მისართი Report Server-ზე (reportPath);
- ფორმატის დასახელება, რომელშიც უნდა მოხდეს ანგარიშგების ექსპორტირება (format);
- ანგარიშგების პარამეტრები (param1, param2, param3 და ა.შ.).

ანგარიშგების გენერაციისთვის სავალდებულოა Reporting Services Credentials მანდატის განსაზღვრა. მომხმარებლის დასახელება, პაროლი და დომენის დასახელება კონფიგურირებულია web.config ფაილში, ისევე როგორც Reporting Server სერვერზე ატვირთული რეპორტების საქაღალდის მისამართი (ნახ.2.22).

```
<add key="ReportingServerAddress"
     value="http://localhost/ReportServer"/>
<add key="ReportServiceUserName" value="ReportUser"/>
<add key="ReportServicePassword" value="123"/>
<add key="ReportServiceDomain" value="domain"/>
```

ნახ.2.22. Web.config ფაილში Reporting Services Credentials კონფიგურაცია

2.23 ნახაზზე ასახულია SaveReportToTempFile() მეთოდში ServerReport ობიექტის ინიციალიზაციის მაგალითი Web.config კონფიგურაციის პარამეტრების გამოყენებით.

შემდეგ ბიჯზე ServerReport ობიექტს გადაეცემა ანგარიშგების პარამეტრები და Render() მეთოდის გამოძახებით მოხდება ექსპორტირების ფორმატში გადაყვანა (ნახ.2.24). დაბრუნებული bytes[] მასივი File.WriteAllBytes() მეთოდით იწერება დროებით ფაილში და შემდეგ, მომხმარებლის სურვილისამებრ, შესაძლებელია ამ ფაილის გადმოწერა ან ეკრანზე გახსნა.

```
var TmpServerReport = new ServerReport
{
    ReportServerCredentials = new ReportCredentials
        (
            ConfigurationManager.AppSettings["ReportServiceUserName"],
            ConfigurationManager.AppSettings["ReportServicePassword"],
            ConfigurationManager.AppSettings["ReportServiceDomain"]),
    ReportServerUrl = new Uri(
        ConfigurationManager.AppSettings["ReportingServerAddress"]),
    ReportPath =
        ConfigurationManager.AppSettings["ReportingServerFolder"]
        + ReportingListLine_ReportPath
};
```

ნახ.2.23. SaveReportToTempFile() მეთოდში პარამეტრების გამოყენება

```
TmpServerReport.SetParameters(
    new ReportParameter("StringParameter1", param1));
TmpServerReport.SetParameters(
    new ReportParameter(
        "DateParameter2", param2.ToString("yyyyMMdd")));
TmpServerReport.SetParameters(
    new ReportParameter(
        "DateParameter3", param3.ToString("yyyyMMdd")));
File.WriteAllBytes(
    TempPath, TmpServerReport.Render(ExportFormat));
DocumentStorageBO_SLData DS = new DocumentStorageBO_SLData()
    {
        DocumentStorage_FileName = TmpFileName,
        DocumentStorage_Extention = TmpExtension
    };
return DS;
```

ნახ.2.24. რეპორტისთვის პარამეტრების გადაწოდება და მითითებული ფორმატით ექსპორტირება bytes[] მასივში

2.9. მეორე თავის დასკვნა

თანამედროვე ვებ-ტექნოლოგიების საფუძველზე, კერძოდ Microsoft-ის კომპანიის ახალი პროგრამული პაკეტების გამოყენებით შესაძლებელია მომხმარებელზე ორიენტირებული, მოქნილი და ეფექტური ჰიბრიდული ბიზნეს-აპლიკაციების დამუშავება. ASP.NET Web Forms, Silverlight და MVC ტექნოლოგიების მიზანმიმართული გამოყენების მიზნით განხორციელებულია მათი მახასიათებლების ანალიზი, შედარება და სამუშაო რეჟიმების შესაბამისი რეკომენდაციების შემუშავება.

SQL Server Reporting სერვისები ვებ-აპლიკაციაში ანგარიშგებების ინტეგრაციის მრავალფეროვანი არჩევანის გაკეთების საშუალებას იძლევა. მეთოდოლოგიის არჩევისას უნდა ვიხელმძღვანელოთ დასმული ამოცანიდან გამომდინარე: ლოკალური რეპორტები კარგად მუშაობს მცირე ან საშუალო სიდიდის ინფორმაციის გადამუშავების შემთხვევაში, ხოლო რთული გამოთვლითი სამუშაოების და დიდი ზომის ინფორმაციის გამოტანის დროს რეკომენდებულია სერვერული რეპორტების გამოყენება.

III თავი
Web-სერვისების ავტომატური ტესტირების თანამედროვე საინფორმაციო ტექნოლოგიები და ინსტრუმენტები

3.1. Selenium IDE სამუშაო გარემოს გაცნობა

Selenium-IDE (Integrated Development Environment) არის პროგრამული პროდუქტი, რომლის საშუალებითაც შესაძლებელია ტესტ-ქეისების შექმნა ავტომატურ რეჟიმში. იგი ადვილად გამოყენებადი Firefox ბრაუზერის პლაგინია.

ზოგადად ითვლება, რომ Selenium IDE ყველაზე ეფექტური გზაა ტესტ-ქეისების შესაქმნელად. იგი შეიცავს კონტექსტურ მენიუს, რომლის საშუალებით შეგვიძლია მოვნიშნოთ ვებ–გვერდის UI (User Interface) ელემენტი და შემდგომ ავირჩიოთ შესაბამისი Selenium ბრძანება. Selenium IDE არის არამხოლოდ ტესტ-სკრიპტების წერისას დახარჯული დროის მოგების საშუალება, არამედ იგი საუკეთესო არჩევანია Selenium სკრიპტ-სინტაქსის შესასწავლად.

Selenium IDE თავსებადია მხოლოდ Firefox ბრაუზერზე, არ არსებობს მისი ალტერნატივა, რომელიც სცენარებს ჩაიწერს IE-ზე (Internet Explorer) ან სხვა ბრაუზერზე. მიუხედავად ამისა Selenium IDE-ის და Firefox-ის საშუალებით შექმნილი ტესტ-სკრიპტები შესაძლებელია შესრულებაზე გავუშვათ ნებისმიერ ინტერნეტ ბრაუზერზე (IE, FF,Chrome,Opera,..) Selenium RC-ის საშუალებით [6].

როგორც აღვნიშნეთ, Selenium IDE არის Firefox ბრაუზერის პლაგინი, რომლის ინსტალაციისთვის საჭიროა:

- გავხსნათ ლინკი: http://seleniumhq.org/download/;
- გადმოვწეროთ და დავაინსტალიროთ Selenium IDE როგორც Firefox პლაგინი;

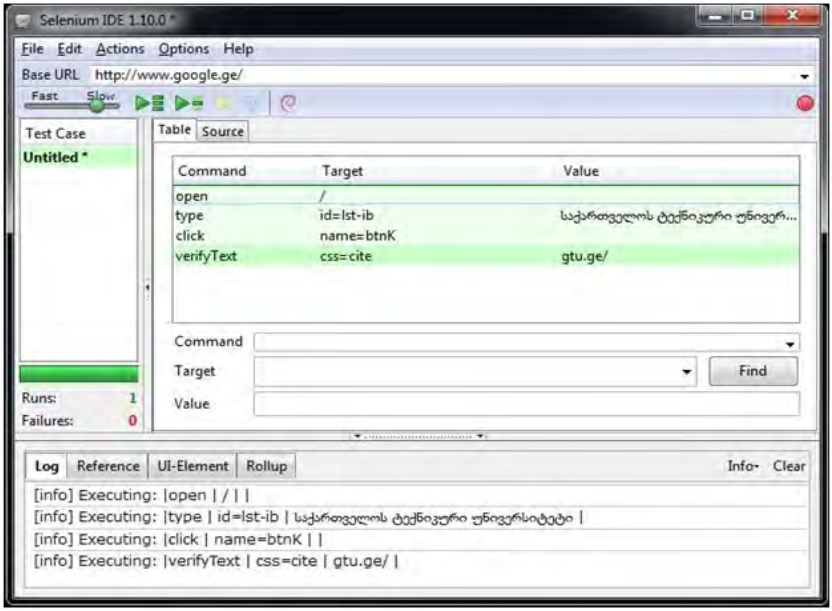
- გადავტვირთოთ Firefox ბრაუზერი;
- Tools მენიუდან გავუშვათ Selenium IDE.

3.1 ნახაზზე ნაჩვენებია Selenium IDE-ს სამუშაო გარემო.

❖ **მენიუ:**

➢ **File** მენიუს აქვს ტესტ-ქეისების და ტესტ-სუიტების ოფციები. მათი გამოყენებით შესაძლებელია ახალი ტესტ-ქეისის დამატება, გახსნა, შენახვა და ექსპორტი სხვადასხვა ენებზე. ასევე შესაძლებელია მიმდინარე ტესტ-ქეისის გახსნა. ყველა ჩამოთვლილი ფუნქცია ვრცელდება ტესტ-სუიტებისთვისაც;

➢ **Edit** მენიუს საშუალებით შესაძლებელია კოპირება, გადაწერა, წაშლა, ყველაფრის მონიშვნა, რათა დავარედაქტიროთ Selenium ბრძანებები მოცემულ ტესტ-ქეისში;

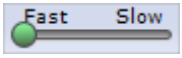


ნახ.3.1. Selenium IDE სამუშაო გარემო

- **Options** მენიუ განკუთვნილია Setting–ის ცვლილებების-თვის. აქედან შესაძლებელია განისაზღვროს ბრძანების ლოდინის დრო, ტესტ-ქეისების ენა და სხვა;
- **Help** მენიუ არის სტანდარტული Firefox help მენიუ, რომელიც შეიცავს Selenium IDE – ის ოფიციალურ დოკუმენტაციას.

❖ **პროგრამული პანელი:**

პროგრამულ პანელზე განლაგებულია ღილაკები, რომლებიც უზრუნველყოფს ტესტ-ქეისების ჩაწერას, გაშვებას, კონტროლს, დებაგს და ა.შ. Selenium IDE-ს პროგრამული პანელი:



Speed Control (სიჩქარის კონტროლი): აკონტროლებს ტესტ–ქეისის შესრულებაზე გაშვების სიჩქარეს;

Run All (ყველას გაშვება): შესრულებაზე უშვებს მოცემულ ტესტ-სუიტს, თავისი შემადგენელი ტესტ–ქეისებით;

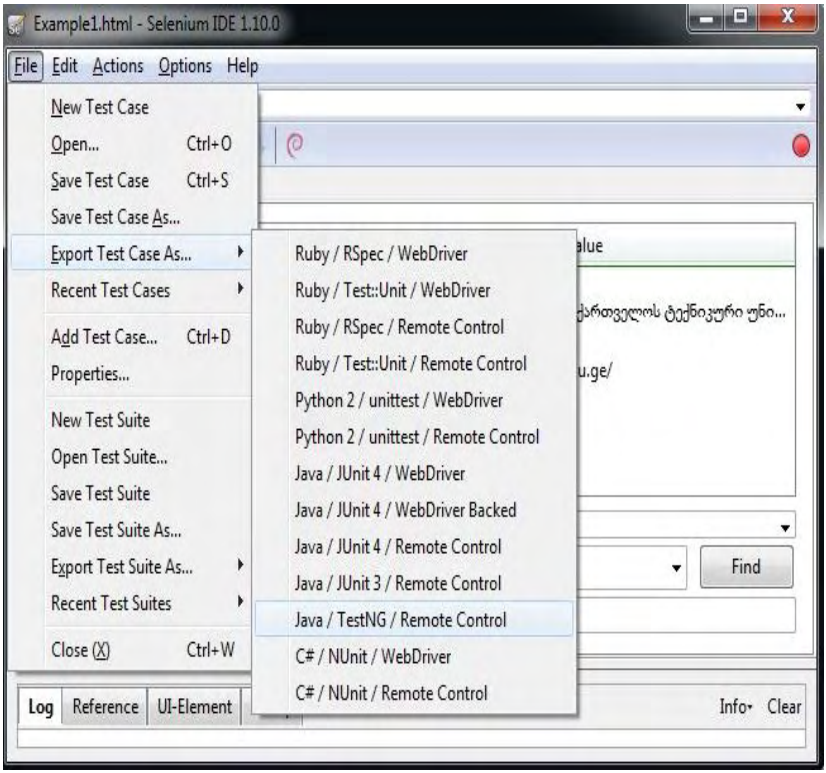
Run (გაშვება): შესრულებაზე უშვებს მონიშნულ ტესტს. როცა მოცემულია მხოლოდ ერთი მარტივი ტესტი, ამ შემთხვევაში ამ ღილაკს და "Run All" ღილაკს ერთიდაიგივე ფუნქცია აქვს;

Pause / Resume (გაჩერება, განახლება): უზრუნველყოფს გაშვებული ტესტ-ქეისის გაჩერება-განახლებას;

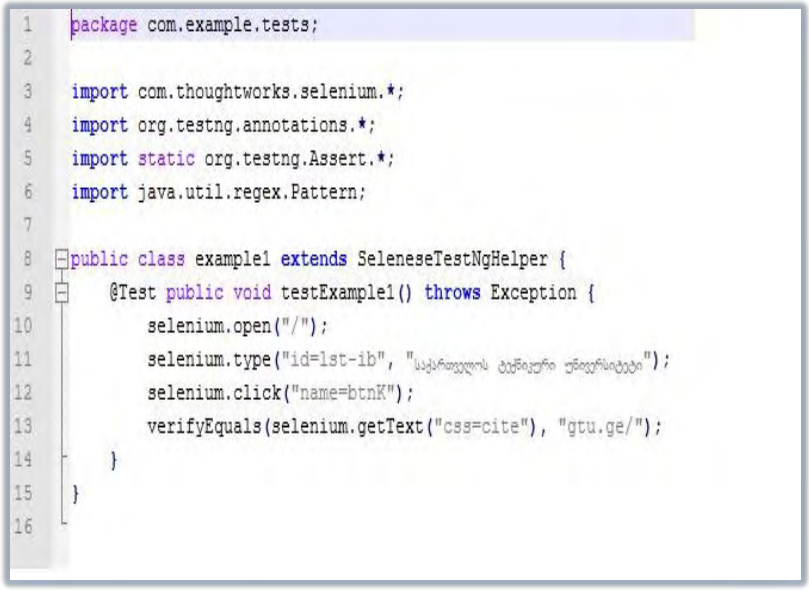
Step (ბიჯი): მისი საშუალებით შესაძლებელია ტესტ-ქეისის გაშვება ბიჯებით, ანუ ყოველ ბიჯზე სრულდება ერთი Selenium ბრძანება. იგი გამოიყენება დებაგის დროს;

Record (ჩამწერი): იწერს მომხმარებლის მიერ ინტერნეტ–ბრაუზერში გაკეთებულ მოქმედებებს;

Selenium IDE–ის საშუალებით შესაძლებელია ჩაწერილი HTML სკრიპტი დავაექსპორტიროთ რომელიმე თანამედროვე პროგრამირების ენაზე. მოცემულ სურათზე (ნახ.3.2) ასახულია Java–ზე ექსპორტი, ხოლო მომდევნო სურათზე (ნახ.3.3) გენერირებული Java კოდი [6].



ნახ.3.2. ტესტ-სკრიპტის Java-ზე ექსპორტი



```
package com.example.tests;

import com.thoughtworks.selenium.*;
import org.testng.annotations.*;
import static org.testng.Assert.*;
import java.util.regex.Pattern;

public class example1 extends SeleneseTestNgHelper {
    @Test public void testExample1() throws Exception {
        selenium.open("/");
        selenium.type("id=lst-ib", "საქართველოს ტექნიკური უნივერსიტეტი");
        selenium.click("name=btnK");
        verifyEquals(selenium.getText("css=cite"), "gtu.ge/");
    }
}
```

ნახ.3.3. ექსპორტის შედეგად მიღებული
Java ფაილი

მას შემდეგ რაც დავაინსტალირეთ Selenium IDE, მისი საშუალებით შეგვიძლია გავაანალიზოთ ტესტირების პროცესი. ზოგადად არსებობს რამდენიმე წესი, რომელიც უნდა გავით-ვალისწინოთ ტესტის შექმნისას. ეს წესები ვრცელდება ყველა ავტომატურ ტესტზე, რომლებიც განკუთვნილია სამომხმარებლო ინტერფეისებისთვის:

- ტესტს ყოველთვის განსაზღვრული უნდა ჰქონდეს **საწყისი წერტილი**. ჩვენ შემთხვევაში ეს ნიშნავს გვერდის გახსნას, საწყისი URL - ის მითითებას, რომ დაიწყოს პროცესი;
- ტესტი არ უნდა იყოს **დამოკიდებული** სხვა ტესტის გაშვების შედეგზე. თუ მოცემული ტესტი ჩავარდა (გავიდა შეცდომაზე), ამან არ უნდა გამოიწვიოს მომდევნო ტესტის ჩავარდნა;

- ტესტმა ერთჯერ უნდა დატესტოს ერთი შემთხვევა;
- ტესტი უნდა გაიწმინდოს თვითონვე.

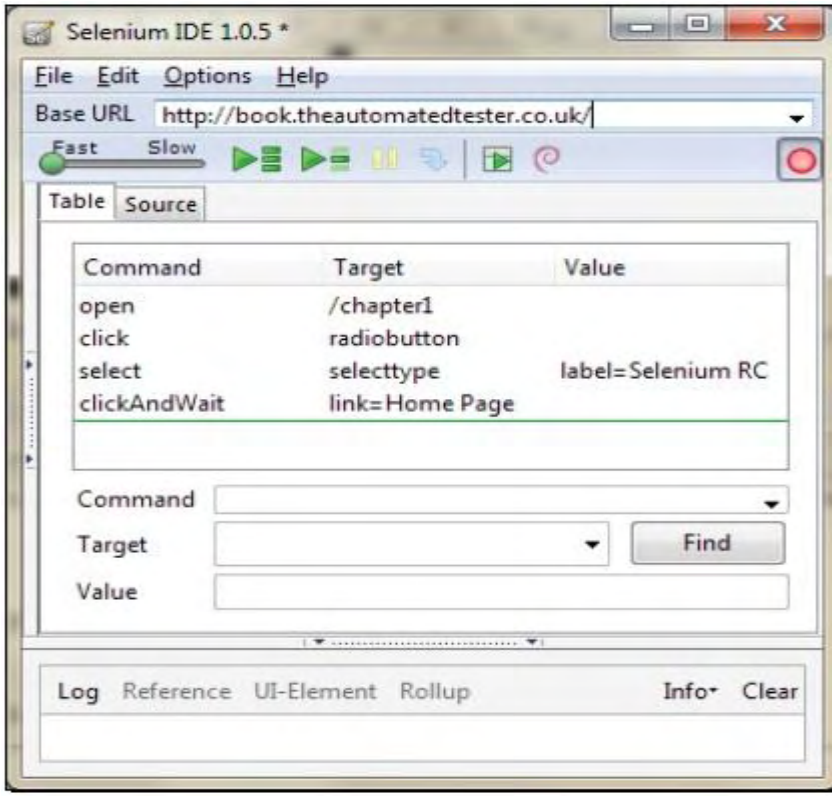
რომელიმე ამ წესთაგანი შესაძლებელია დარღვეულ იქნას, რადგან მათი დარღვევა შეიძლება ნიშნავდეს, რომ თქვენ იპოვნეთ გამოსავალი, მაგრამ დიდი რაოდენობის ტესტების შემთვხვევაში შეიძლება მივიღოთ პირიქით: პატარა ტესტის ჩავარდნამ გამოიწვიოს სცენარის უდიდესი ნაწილის გაწითლება.

ამ წესების შესაბამისად შევქმნათ ტესტი Selenium IDE–ს გამოყენებით. ტესტის ჩასაწერად გავუშვათ Mozilla Firefox და მისი Tools მენიუდან ავირჩიოთ Selenium IDE. შევნიშნოთ რომ ჩაწერის ღილაკი არის ავტომატურად მონიშნული [6].

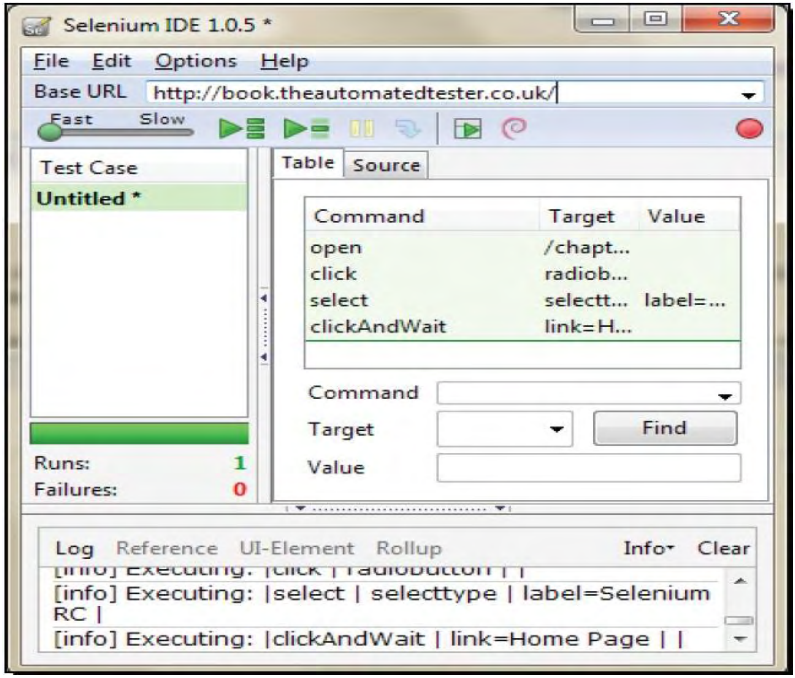
ტესტის ჩაწერისთვის საჭიროა შევასრულოთ შემდეგი ქმედებები:

1. შევცვალოთ **Base URL** შემდეგი მისამართით:
   http://book.theautomatedtester.co.uk/.
2. დავაჭიროთ Radio button-ზე;
3. შევცვალოთ **Select**–ის მნიშვნელობა;
4. გადავიდეთ Home Page ლინკზე;
5. ჩაწერილი ტესტი უნდა გამოიყურებოდეს მოცემული სურათის მსგავსად (ნახ.3.4).
6. დავაჭიროთ შესრულებაზე გაშვების ღილაკს:  .
7. როდესაც ტესტი დასრულდება, იგი უნდა გამოიყურებოდეს შემდეგი სქრინშოტის ანალოგურად (ნახ.3.5).

ამრიგად, ჩვენ წარმატებით ჩავწერეთ პირველი ტესტი და გავუშვით შესრულებაზე. განხილულ ტესტში საწყის წერტილად შეიძლება განვიხილოთ **open** ბრძანება. მას აქვს მინიჭებული საწყისი URL–ი, საიდანაც იგი გადადის /chapter1 ლინკზე, და იწყება ჩაწერის პროცესი [5].



ნახ.3.4. Selenium IDE ჩაწერილი ტესტი



ნახ.3.5. Selenium IDE ტესტის შედეგი

როდესაც ტესტი დაასრულებს მუშაობას, მივიღებთ შესრულებული ქმედებების მიმდევრობას, რომელთაც აქვს მწვანე ფონი. მწვანე ფონი გვიჩვენებს, რომ მოქმედებები დასრულდა წარმატებით. მარცხენა მხარეს მოცემულია **Runs: 1**, ანუ გვაქვს ერთი წარმატებული ტესტი. თუ ტესტი ჩავარდებოდა, მაშინ **Failures** იქნებოდა 1–ის ტოლი.

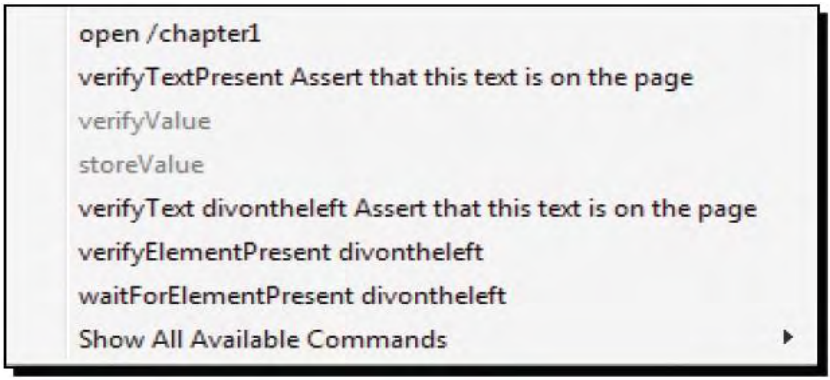
განხილულ მაგალითებში ჩვენ ჩავწერეთ პროცესები, რომლებსაც ახორციელებს მომხმარებელი. შევამოწმეთ ღილაკების და ლინკების მუშაობა. სამწუხაროდ, ჩვენ არ განგვიხილავს შემთხვევა, რომ სხვა რაიმე ელემენტი არსებობს Web-გვერდზე, დამალული ან ცხადი სახით [5].

3.2. Selenium IDE: ვალიდაცია

Selenium-ს გააჩნია ორი ტიპის მექანიზმი web-გვერდზე არსებული ელემენტების შემოწმებისთვის. პირველი მექანიზმია assert: იგი ამოწმებს არის თუ არა ელემენტი მოცემულ გვერდზე. ტესტმა თუ ელემენტი ვერ იპოვნა, ამ შემთხვევაში ტესტი გაჩერდება და დასრულდება.

მეორე არის verify: ისიც აგრეთვე ანალოგიურად ამოწმებს არის თუ არა ელემენტი მოცემულ გვერდზე, მაგრამ თუ ელემენტი არ არსებობს, მაშინ ტესტი აგრძელებს მუშაობას.

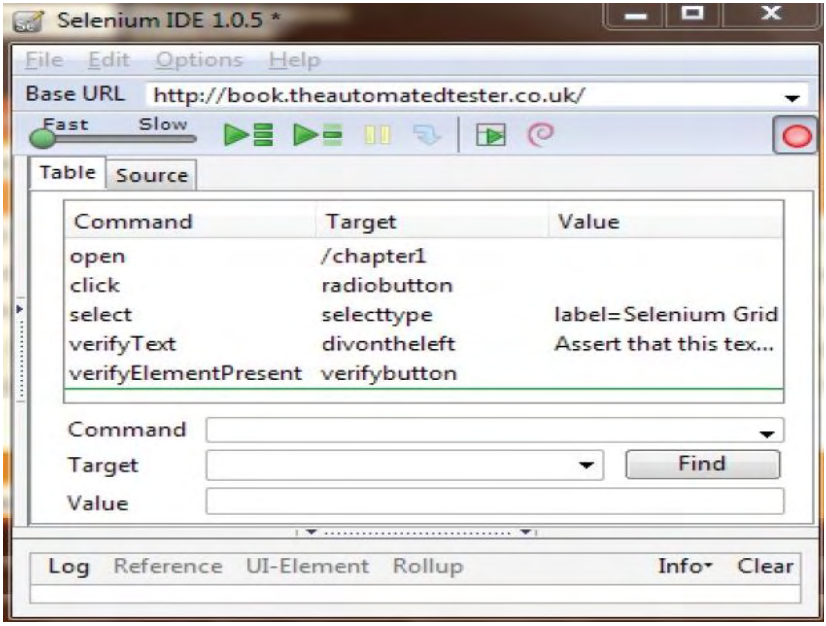
ილუსტრაციისთვის გამოვიყენოთ განხილული ტესტი და დავამატოთ assert ან verify ბრძანებები. ამისთვის საჭიროა Selenium IDE–ს კონტექსტური მენიუს გამოყენება [6]. გვერდზე არსებულ ელემენტთან მივიტანოთ მაუსის კურსორი და დავაჭიროთ მარჯვენა ღილაკს, გამოჩნდება შემდეგი ფორმის კონტექსტური მენიუ (ნახ.3.6).



ნახ.3.6. Selenium IDE კონტექსტური მენიუ

შევცვალოთ განხილული ტესტი: შევამოწმოთ სხვა ელემენტების არსებობა განხილულ გვერდზე.

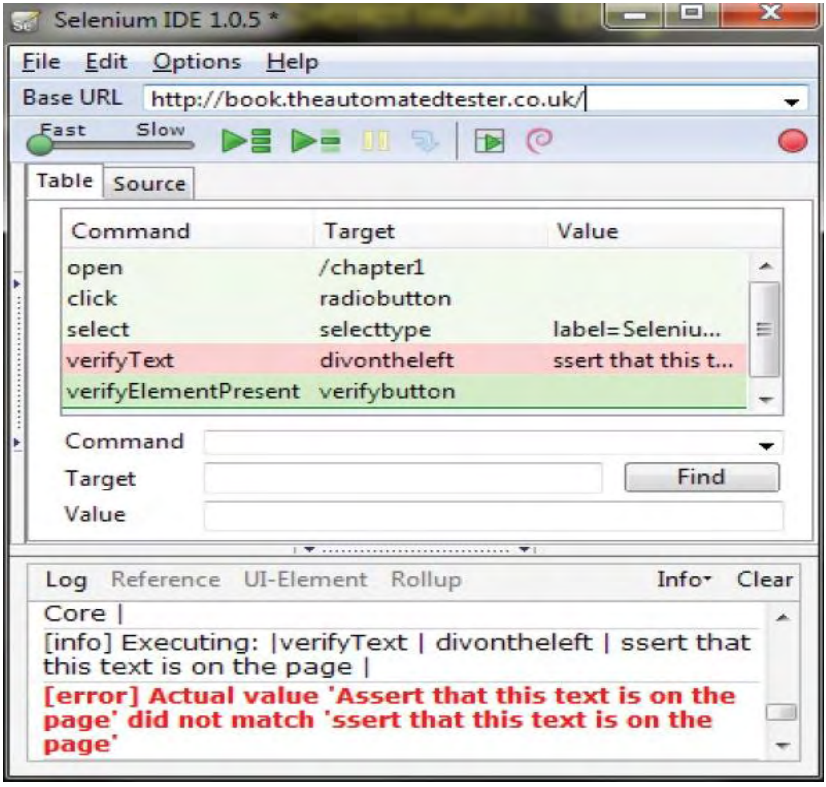
1. გავხსნათ Selenium IDE ჩაწერის რეჟიმში;
2. მივუთითოთ Base URL – http://book.theautomatedtester.co.uk/
3. გადავიდეთ Chapter1-ზე;
4. დავაჭიროთ Radio button-ს;
5. შევცვალოთ Select–ის მნიშვნელობა Selenium Grid-ით;
6. დავამოწმოთ Assert that this text is on the page -ს არსებობა;
7. დავადასტუროთ გვერდზე Verify-ღილაკის არსებობა. გამოვიყენოთ კონტექსტური მენიუ;
8. ყველა პუნქტის შესრულების შემდეგ, Selenium IDE უნდა გამოიყურებოდეს შემდეგნაირად (ნახ.3.7).



ნახ.3.7. Selenium IDE ჩაწერილი ტესტი

თუ გავუშვებთ ტესტს, დავინახავთ, რომ მოხდება ტექსტის და ღილაკის არსებობის შემოწმება. გავითვალისწინოთ, რომ ყველა verify ბრძანებას აქვს მუქი მწვანე ფერი.

რა მოხდებოდა იმ შემთხვევაში თუ verify ბრძანება ვერ იპოვნიდა იმას, რაც აღვწერეთ ? Selenium IDE დააფიქსირებდა შეცდომას, რომ აღწერილი ტექსტი არ არსებობს, მაგრამ იგი გააგრძელებდა ტესტის შესრულებას. ეს შემთხვევა მოცემულია შემდეგ ნახაზზე (ნახ.3.8) [7]:



ნახ.3.8. Selenium IDE გაშვებული ჩავარდნილი ტესტი

verify ბრძანების ნაცვლად თუ გამოვიყენებთ assert ბრძანებას, ამ შემთხვევაში ტესტი დაასრულებს მუშაობას იქ, სადაც დაფიქსირდება შეცდომა. საბოლოოდ ჩვენ ვნახეთ, რომ შეგვიძლია გამოვიყენოთ assert და verify ბრძანებები, რათა შევამოწმოთ არსებობს თუ არა განსაზღვრული ელემენტი მოცემულ გვერდზე.

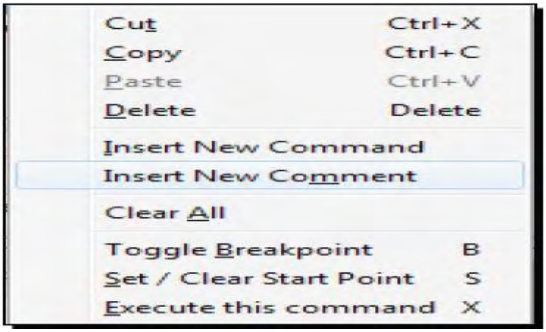
ავტომატურად, Selenium IDE არ იწერს მსგავსი ტიპის ბრძანებებს, ამიტომ ამ სახის ბრძანებები უნდა მივუთითოთ ხელით [6]. საბოლოოდ შეგვიძლია დავასკვნათ, რომ თუ ჩვენ გამოვიყენებთ assert ბრძანებას და ის იპოვნის შეცდომას, ამ შემთხვევაში ტესტი გაჩერდება, verify ბრძანების დროს კი ტესტი აგრძელებს მუშაობას. ეს არის მათი ერთადერთი განმასხვავებელი თვისება. მიუხედავად ამისა ორივე ტიპის ბრძანებებს აქვთ თავისი გამოყენების სფეროები.

ქვემოთ მოცემულია სხვადასხვა verify და assert ბრძანებები:

- **verifications**
  - verifyElementPresent – ელემენტის არსებობის შემოწმება;
  - verifyElementNotPresent – ელემენტის არარსებობის შემოწმება;
  - verifyText – ტექსტის შემოწმება;
  - verifyAttribute – ატრიბუტის შემოწმება;
  - verifyChecked – მონიშვნის შემოწმება (ჩექბოქსი, რადიობოქსი);
  - verifyAlert – ალერტის შემოწმება;
  - verifyTitle – სათაურის შემოწმება;
- **assertions**
  - assertElementPresent - ელემენტის არსებობის შემოწმება;
  - assertElementNotPresent – ელემენტის არარსებობის შემოწმება;
  - assertText – ტექსტის შემოწმება;
  - assertAttribute – ატრიბუტის შემოწმება;
  - assertChecked – მონიშვნის შემოწმება (ჩექბოქსი,რადიობოქსი);
  - assertAlert – ალერტის შემოწმება;
  - assertTitle – სათაურის შემოწმება.

როგორც ჩვენთვის ცნობილია, იმისათვის, რომ გვქონდეს ადვილად აღქმადი პროგრამული კოდი, საჭიროა გამოვიყენოთ კომენტარები, რომლებიც დაგვეხმარება პროგრამული კოდის გარჩევისას. განხილული ფაქტის მსგავსად, კარგი იქნება თუ ჩვენს ავტომატურ ტესტებს დავურთავთ კომენტარებს, რომლებიც მომავალში გამოყენებულ იქნება სხვა ტესტერის მიერ [6].

იმისთვის, რომ დავამატოთ კომენტარი განხილულ ტესტში, საჭიროა: Selenium IDE - ს კონტექსტური მენიუდან ავირჩიოთ Insert New Comment ბრძანება, შედეგად მივიღებთ ცარიელი ადგილს ბრძანებებს შორის (ნახ.3.9).



ნახ.3.9. Selenium IDE კონტექსტური მენიუ

განხილულ სურათზე მოცემულია კომენტარი, რომელიც შექმნილია Selenium IDE-ს დახმარებით. ტესტის კომენტარი ყოველთვის გამოჩნდება მუქი წითელი ფერის ტექსტით.

თანამედროვე ინფორმაციულ ტექნოლოგიებში გავრცელებული Web აპლიკაციები, სამწუხაროდ, ბრაუზერის მხოლოდ ერთ ფანჯარაში არ არსებობს. მაგალითისთვის განვიხილოთ რეპორტების საიტი. თითოეულ რეპორტს აქვს თავისი საკუთარი ფანჯარა, სადაც მომხმარებელი გადაადგილდება ერთი ფანჯრიდან მეორეზე და ა.შ.

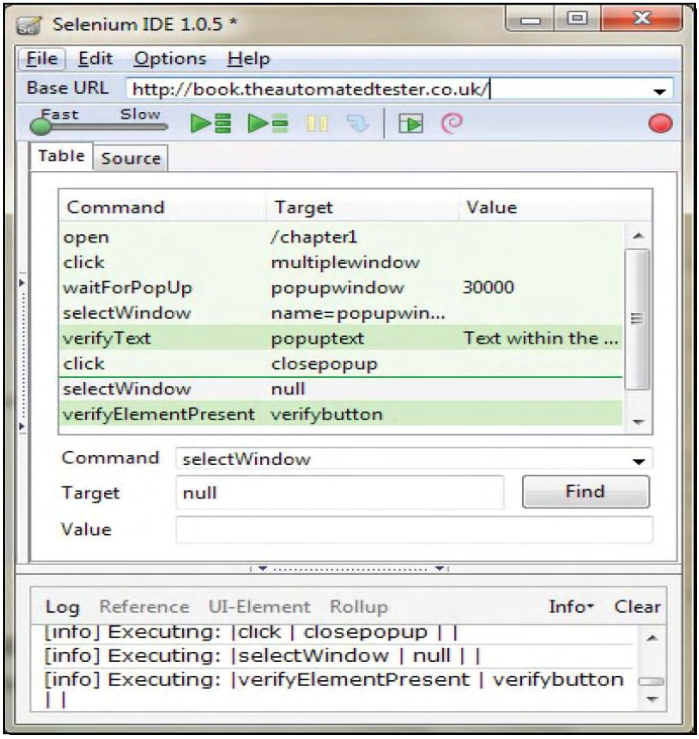
სამწუხაროდ, მრავალფანჯრიანი web გვერდის ტესტირების პროცესში შეიძლება წავაწყდეთ სირთულეებს. ეს არის ის დეტალური საკითხები, რომლებსაც უნდა მივაქციოთ ყურადღება, რათა უპრობლემოდ გადავიდეთ სხვადასხვა ფანჯრებზე.

მრავალფანჯრიანი აპლიკაციების მართვა არის ერთ–ერთი რთული საკითხი Selenium ავტომატური ტესტირების დროს. ამ დროს ბრაუზერის მიერ გახსნილ ყოველ შვილობილ ფანჯარაზე უნდა გადავიდეს Selenium-ი იმდენჯერ, რამდენჯერაც წარმოიქმნება შვილობილი პროცესი.

მრავალფანჯრიანი web გვერდის საილუსტრაციოდ განვიხილოთ მაგალითი, სადაც გვერდის ელემენტზე მაუსით დაკლიკვა წარმოქმნის ახალ ფანჯარას. თუ ჩართულია ბრაუზერის pop-up blocker, გავთიშოთ იგი, რათა მოხდეს ახალი pop-up ფანჯრის ჩამოტვირთვა [5].

1. ავამუშავოთ Selenium IDE და გადავიდეთ Chapter1 გვერდზე;
2. დავკლიკოთ ელემენტი, რომლის ტექსტია **Click this link to launch another**, შედეგად წარმოიქმნება პატარა ფანჯარა;
3. ფანჯარის ჩამოიტვირთვის შემდეგ დავამატოთ verify ბრძანება ელემენტისთვის წარწერით – Text within the pop up window;
4. დავაჭიროთ **Close the window** ლინკს;
5. დავადასტუროთ, რომ დავბრუნდით საწყის გვერდზე (ნახ.3.10).

შევამჩნევთ, რომ განხილულ ტესტში Selenium აჭერს ლინკს, რომელიც ხსნის ახალ ფანჯარას და Selenium ავტომატურად სვამს **waitForPopUp** ბრძანებას. მაშასადამე, ტესტი ხვდება, რომ იგი უნდა დაელოდოს web სერვერს, რათა მიიღოს პასუხი მოთხოვნაზე და საბოლოოდ ბრაუზერმა გახსნას ახალი გვერდი. ნებისმიერი ბრძანება, რომელიც ითხოვს ახალი გვერდის ჩამოტვირთვას web სერვერიდან, იქნება ***waitFor*** ტიპის ბრძანება.



ნახ.3.10. Selenium IDE მრავალფანჯრიანი სისტემა

განვიხილოთ ბრძანება **selectWindow**. იგი ატყობინებს Selenium-ს, რომ მას სჭირდება გადავიდეს სხვა ფანჯარაზე, რომლის სახელია **popupwindow** და მომდევნო ბრძანებები შეასრულოს აღნიშნულ ფანჯარაზე მანამ, სანამ გადავმისამართდებით კიდევ სხვა ფანჯარაზე და ა.შ.

როდესაც ტესტი დაასრულებს მუშაობას pop-up ფანჯარაზე, შემდეგ საჭიროა დავბრუნდეთ მშობელ ფანჯარაზე. ამისთვის საჭიროა განვსაზღვროთ ფანჯარა როგორც **null**. ეს უზრუნველყოფს ტესტის უკან, მშობლიურ ფანჯარაზე დაბრუნებას.

3.3. Selenium ტესტები AJAX აპლიკაციებისთვის

AJAX (Asynchronous JavaScript and XML) არის Web ტექნოლოგია, რომელიც უზრუნველყოფს სწრაფი, დინამიკური გვერდების შექმნას. AJAX ტექნოლოგიის დროს Web აპლიკაციას შეუძლია გააგზავნოს და მიიღოს მონაცემები სერვერიდან ასინქრონულად, ანუ განაახლოს Web გვერდის რაღაც ნაწილი მთლიანი გვერდის განახლების გარეშე.

AJAX ტექნოლოგიით დაწერილი აპლიკაციების ავტომატური ტესტირებისას, შესაძლებელია ტესტის ჩავარდნის მიზეზი გახდეს ელემენტის არარსებობა მიმდინარე გვერდზე – როდესაც ბრაუზერი განახლების რეჟიმშია და ელოდება გაგზავნილი მოთხოვნის პასუხს სერვერიდან. შესაძლებელია აპლიკაციის მუშაობის სიჩქარე დავარდნილი იყოს სხვადასხვა ფაქტორების გამო: ტექნიკური მოწყობილობის სიმძლავრე, ქსელის პრობლემა, გადატვირთული რეჟიმი და სხვა.

არსებული პრობლემის გადაჭრა შესაძლებელია waitFor ტიპის ბრძანებების მეშვეობით. ასეთი ტიპის ბრძანებებს Selenium გადაჰყავს ლოდინის რეჟიმში. waitFor ბრძანებები გამოიყენება იმ შემთხვევაში, როდესაც საჭიროა Selenium გადავიდეს ლოდინის რეჟიმში. აღნიშნული ბრძანებების დროს Selenium-ის ლოდინის დრო არის მაქსიმუმ 30 წამი „გამოუცხადებლად", მაგრამ შეგვიძლია აღნიშნული დროის ცვლილება setTimeout ბრძანების საშუალებით. მაშასადამე, setTimeout ბრძანება ზოგადად, განსაზღვრავს waitFor ტიპის ბრძანებების მაქსილამურ ლოდინის დროს.

არსებობს Selenium-ის სხვადასხვა ბრძანებები, რომლებიც გამოიყენება web ელემენტების ლოდინისთვის. ქვემოთ ჩამოთვლილია waitFor ტიპის ბრძანებები [6]:

- WaitForAlertPresent – დაელოდე ალერტის წარმოქმნას;
- waitForAlertNotPresent – დაელოდე ალერტის არწარმოქმნას;

- waitForElementPresent – დაელოდე ელემენტის გამოჩენას;
- waitForElementNotPresent – დაელოდე ელემენტის არგამოჩენას;
- waitForTextPresent – დაელოდე ტექსტის გამოჩენას;
- waitForTextNotPresent – დაელოდე ტექსტის არგამოჩენას;
- waitForPageToLoad – დაელოდე გვერდის ჩამოტვირთვას;
- waitForFrameToLoad – დაელოდე ფრეიმის ჩამოტვირთვას.

ზოგიერთი ზემოთ ჩამოთვლილი ბრძანება შეიძლება გაეშვას სხვა ბრძანების შესრულებისას ავტომატურად. მაგალითად, განვიხილოთ clickAndWait ბრძანება. მისი შესრულებისას ხდება ორი ბრძანების გაშვება: იგი ჯერ აკეთებს ჩვეულებრივ click–ს და შემდგომ ასრულებს waitForPageToLoad ბრძანებას. ასევე ჩვენთვის ნაცნობი open ბრძანება, რომელიც სრულდება მაშინ, როდესაც გვერდი მთლიანად ჩამოიტვირთება.

ზოგჯერ საჭიროა web გვერდის ელემენტების შენახვა ტესტში, რათა მოგვიანებით გამოვიყენოთ ისინი. ეს ნიშნავს იმას, რომ ტესტში შევინახოთ web გვერდზე არსებული მონაცემი შემდგომი გამოყენების მიზნით. Selenium ტექნოლოგიით მონაცემების შენახვა ხდება store ტიპის ბრძანებით. ტესტებში ცვლადების გამოყენების ფორმატია:

storedVars['variableName'];

ჩვენ შევქმენით რამდენიმე ტესტი, წარმატებით გავუშვით შესრულებაზე, ვნახეთ თუ როგორ მუშაობს Selenium ტესტი AJAX აპლიკაციებზე, მაგრამ სამწუხაროდ ცოტა რთულია ისეთი ტესტის შექმნა, რომელიც გაეშვება ბოლომდე. ამიტომ, ავტომატური ტესტების შექმნისას საჭიროა შექმნილი ტესტის დებაგი, რომ ვნახოთ დაწერილი ტესტ-სრიპტის შეცდომები [6].

თუ ტესტი გაშვებულია **Pause** ღილაკის საშუალებით, შესაძლებელია გავაჩეროთ მისი შესრულება. როდესაც ტესტის შესრულება გაჩერდება, **Step** ღილაკის საშუალებით შესაძლებელია მომდევნო ბრძანებების მიმდევრობით თითო–თითოდ შესრულება.

თუ მიმდინარე web გვერდზე გვაქვს ელემენტის პოვნის პრობლემა, შეგვიძლია ჩავწეროთ ელემენტის დასახელება Target ველში, დავაწვეთ Find ღილაკს და web გვერდის ელემენტი წამიერად განათდება. თუ მიმდინარე გვერდზე არცერთი ელემენტი არ განათდა, მაშინ მსგავსი სახელის ელემენტი არ არსებულა web გვერდზე. საბოლოოდ, ჩვენ განვიხილეთ ტესტ–სკრიპტების ჩაწერა Selenium IDE-ის საშუალებით და ჩაწერილი ტესტები წარმატებით გავუშვით შესრულებაზე.

სამწუხაროდ, თანამედროვე Web ტექნოლოგიებში არსებობს ისეთი მოქმედებები, რომელთაც Selenium ვერ ასრულებს. როგორც ვიცით, Selenium–ი შექმნეს JavaScript-ზე, ამის გამო იგი ცდილობს განახორციელოს ისეთი მოქმედებები, რაც შესაძლებელია JavaScript–ით. სხვა სიტყვებით რომ ვთქვათ, იგი შემოიფარგლება JavaScript ტექნოლოგიის შესაძლებლობებით.

ქვემოთ ჩამოთვლილია Selenium IDE-ს არსებული „სუსტი წერტილები":

- **Serverlight და Flex / Flash** აპლიკაციების მოქმედებები შეუძლებელია ჩაიწეროს Selenium IDE-ით. აღნიშნული ორივე ტექნოლოგია მოქმედებს თავის საკუთარ ელემეტებზე და არ იმართება DOM ტექნოლოგიით. DOM (Document Object Model) არის პლატფორმისგან, პროგრამული ენისგან დამოუკიდებელი მოდელი, რომელიც უზრუნველყოფს ობიექტების წარმოდგენას HTML, XHTML, XML დოკუმენტებში. იგი HTML დოკუმენტს წარმოადგენს ხისებური სტრუქტურით;

- **HTML5** ტექნოლოგია არ არის Selenium IDE-ს მიერ ბოლომდე მხარდაჭერილი. აღნიშნული პრობლემის კარგ მაგალითს წარმოადგენს HTML5 ელემენტი, რომელსაც აქვს ატრიბუტი contentEditable=true. თუ განვიხილავთ type ბრძანებას, რათა ჩავწეროთ რამე ტექსტი მოცემულ ელემენტზე, დავინახავთ, რომ ტესტი წარმატებით დაასრულებს მუშაობას, მაგრამ UI (user interface) ელემენტში ტექსტი არ ჩაიწერება [5,6];
- Selenium IDE არ მუშაობს **Canvas** ელემენტებთან, მაშასადამე შეუძლებელია 2D ფიგურების მართვა;

Selenium IDE–ის არ შეუძლია ფაილების ატვირთვა, რადგან Selenium-ით შეუძლებელია Windows ფანჯრების მართვა.

3.4. ელემენტის ლოკატორები და მათი კლასიფიკაცია

ავტომატური ტესტების წარმატებით მუშაობა დამოკიდებულია GUI (Graphical Use Interface) ელემენტების იდენტიფიკაციაზე, მათი განლაგების განსაზღვრაზე, რათა ტესტის შესრულებისას მათზე განხორციელდეს სხვადასხვა ოპერაციები. Selenium–ს გააჩნია სხვადასხვა ინსტრუმენტი რომლებიც განსაზღვრავს ელემენტებზე წვდომის განსხვავებულ გზებს: Name, ID, Link, CSS და Xpath.

Web დეველოპმენტის პროექტში რეკომენდებულია GUI ელემენტებისთვის აღიწეროს ისეთი ატრიბუტები, როგორიცაა: ID, Name, Class. აღნიშნული ატრიბუტები იძლევა იმის შესაძლებლობას, რომ აპლიკაცია გახდეს უფრო ტესტირებადი და სტანდარტული გზებით მოხდეს ელემენტებზე წვდომა.

მიუხედავად ამისა პრაქტიკაში გვხვდება ისეთი შემთხვევები, სადაც შეუძლებელია მოცემული ატრიბუტების განსაზღვრა, ან

ელემენტს უბრალოდ არ აქვს აღწერილი ID, Name. მსგავს შემთხვევებში იყენებენ CSS და XPath ლოკატორებს [6].

Web გვერდზე არსებული ელემენტის ლოკატორი არის მისი მისამართი, რომლითაც მივმართავთ ელემენტს: ID, name, link, XPath, CSS. ლოკატორები გამოიყენება ტესტის წერისას, რათა მოვახდინოთ საჭირო ელემენტის იდენტიფიკაცია და განლაგების განსაზღვრა. Selenium ტესტში ლოკატორის საშუალებით მივმართავთ ელემენტს მასზე გარკვეული ურთიერთქმედების მიზნით.

ჩვენ უკვე გავუშვით წარმატებული ტესტები, რომლებიც იყენებდა ლოკატორებს. ავტომატური ტესტირებისას, იმ HTML ელემენტებს, რომლებთანაც ვურთიერთქმედებთ ავტომატურად, უმჯობესია მინიჭებული ჰქონდეს ID და Name [5].

შემდგომ ჩვენ განვიხილავთ:

- ელემენტების განლაგებას ID-ით;
- ელემენტების განლაგებას Name–ით;
- ელემენტების განლაგებას Link-ით;
- ელემენტების განლაგებას XPath-ით;
- ელემენტების განლაგებას CSS–ით.

CSS (Cascading Style Sheet) არის ტექსტის სტილური გაფორმების ენა, რომლის საშუალებით განსაზღვრავენ Web გვერდზე HTML ელემენტების წარმოდგენის სტილს. CSS-ის საშუალებით HTML დოკუმენტი შეგვიძლია წარმოვადგინოთ სხვადასხვა სტილით. იგი საშუალებას იძლევა დავწეროთ ტექსტის ფორმატირების, ელემენტების წარმოდგენის სტილი ცალკე .css ფაილში. ზოგადად, Web გვერდზე ელემენტების წარმოდგენის სტილი შენახულია გარე .css ფაილებში. ტექსტის გარე გაფორმება საშუალებას გვაძლევს მარტივად ვცვალოთ მთლიანი საიტის სტილი, ერთი მარტივი ფაილის ცვლილებით.

XPath არის XML ტეგების ენა, რომელიც გამოიყენება კვანძების მოსანიშნად XML-ში. სხვა სიტყვებით რომ ვთქვათ, იგი არის ენა, რომელიც გამოიყენება XML დოკუმენტიდან ინფორმაციის ამოსაღებად. მას გააჩნია თავისი სინტაქსი რომლის საშუალებითაც მოძრაობს XML ხისებური სტრუქტურის დოკუმენტებში. Xpath-ს, მსგავსად XML-ისა, აქვს ხისებური სტრუქტურა, რათა იმოძრაოს XML ხის კვანძებზე.

განვიხილოთ მარტივი XML დოკუმენტი (ნახ.3.11):

```
<PostAdr residential="true">
  <name title="Mr.">
    <first>Aaron</first>
    <last>Bartell</last>
  </name>
  <street>123 Center Rd</street>
  <cty>Mankato</cty>
  <state>MN</state>
  <zip>56001</zip>
  <phone>123-123-1234</phone>
  <phone>321-321-4321</phone>
</PostAdr>
```

ნახ.3.11. XML დოკუმენტი

აქ მოცემულ XML დოკუმენტში first ელემენტის შესაბამისი Xpath-ი იქნება: **/PostArd/name/first**. როგორც ვხედავთ XPath-ის აგებისას, მშობელი კვანძიდან შვილობილ კვანძზე გადასვლა ხდება "/" სიმბოლოთი. თუ XML დოკუმენტში გვაქვს იდენტური ტეგები (მაგალითად, phone), მაშინ იდენტიფიკაცია ხდება ტეგების ინდექსირების საშუალებით. ინდექსირება იწყება 1-დან. მოცემულ მაგალითში 321-321-4321 ტელეფონის ნომრის შესაბამისი XPath არის: **/PostArd/phone** [6].

განვიხილოთ რამდენიმე პროგრამული ინსტრუმენტი, ინტერნეტ ბრაუზერის პლაგინები. განხილული ინსტრუმენტები

დაგვეხმარება იმის გაგებაში, თუ როგორაა განსაზღვრული ელემენტები და მათი ატრიბუტები, DOM (Document Object Model) სტრუქტურა, CSS სტილის ატრიბუტები და სხვ.

კომპიუტერზე დავაყენოთ ისეთი პროგრამული საშუალებები, რომლებიც დაგვეხმარება ლოკატორის აგებაში:

- **Firebug:** https://addons.mozilla.org/en-US/firefox/addon/firebug/
  - Firebug ფაქტობრივად არის Web დეველოპერების პროგრამული ინსტრუმენტი. იგი საშუალებას იძლევა ვიპოვნოთ Web გვერდის ელემენტები Find ფუნქციის გამოყენებით. Firebug არის Firefox ბრაუზერის პლაგინი;
  - Firebug-ის საშუალებით შესაძლებელია HTML დოკუმენტის ხისებური სტრუქტურით წარმოდგენა;
  - მას აქვს JavaScript-ის REPL (Read Eval Print Loop), რომელიც საშუალებას გვაძლევს გავუშვათ და შევამოწმოთ JavaScript კოდი.
- **Firefinder:** https://addons.mozilla.org/en-US/firefox/addon/ firefinder-for-firebug/

პროგრამული ისტრუმენტი, Web გვერდზე XPath და CSS ტესტირებისთვის. იგი წარმოადგენს Firebug -ის დანამატს.

- **IE Developer.** იგი Firebug-ის ანალოგია, რომელიც ინტეგრირებულია Internet Explorer-ში და მისი გაშვება შესაძლებელია F12 ღილაკით.
- **Google Chrome Developer**

არის Firebug-ის ანალოგი, Google Chrome ბრაუზერის ინსტრუმენტი, რომლის საშუალებითაც შესაძლებელია ელემენტების პოვნა web გვერდზე, მათი Xpath-ის განსაზღვრა და ა. შ. [5].

ჩამოთვლილი ინსტრუმენტები გვეხმარება ელემენტის ლოკატორის დეტალების განსაზღვრასა და ატრიბუტების პოვნაში. მოცემული ინსტრუმენტები Web გვერდზე არსებულ ინფორმაციას წარმოადგენს იერარქიულ ხეში. მათი საშუალებით შესაძლებელია ელემენტის ნებისმიერი ტიპის ლოკატორის განსაზღვრა ტესტ–

სკრიპტებში გამოყენების მიზნით. როგორც უკვე განვიხილეთ, თუ მიმდინარე web გვერდზე გვაქვს ელემენტის პოვნის პრობლემა, Selenium IDE-ის Target ველში შეგვიძლია ჩავწეროთ ელემენტის დასახელება (ელემენტის ლოკატორი), დავაჭიროთ Find ღილაკს და web გვერდის ელემენტი წამიერად განათდება.

განვიხილოთ მაგალითი თუ როგორ შეიძლება ელემენტის ლოკატორის პოვნა მოცემულ გვერდზე Firebug-ის საშუალებით. ცხადია Firefox-ზე დაინსტალირებული უნდა გვქონდეს Firebug. განვიხილოთ Firebug-ის გამოყენების მაგალითი:

1. Firefox-ში გაუშვით Firebug – დავაკლიკოთ მარჯვენა ზედა კუთხეში მოთავსებულ „ხოჭოს" პიკტოგრამას;
2. გამოსულ ფანჯარაზე დვააჭიროთ შემდეგ ღილაკს:  ;
3. მივიტანოთ მაუსის კურსორი აქტიური გვერდის რომელიმე ელემენტთან;
4. გადავატაროთ მაუსი სხვადასხვა ელემენტებს.

როგორც ვხედავთ Firebug-ი ანათებს მაუსით მითითებული ყველა ელემენტის მონაცემებს, სადაც შეგვიძლია ვნახოთ არჩეული ელემენტის სხვადასხვა ატრიბუტები: ID, Name, class, value, text და სხვ. ელემენტის ატრიბუტების გამოყენებით შესაძლებელია ლოკატორის აგება ტესტ-სკრიპტებისთვის, რათა მათი გამოყენებით მოხდეს ელემენტების იდენტიფიკაცია [5].

ტესტ–სკრიპტში ლოკატორად ID ატრიბუტის გამოყენება არის ყველაზე კარგი არჩევანი, რათა განისაზღვროს ელემენტის ადგილმდებარეობა მიმდინარე web გვერდზე. რეკომენდებულია, რომ დეველოპერებმა უზრუნველყონ ელემენტის id ატრიბუტის გაწერა, რომელიც იქნება უნიკალური ყველა ელემენტისთვის. ელემენტისთვის უნიკალური id ატრიბუტის არსებობა უზრუნველყოფს მის ზუსტ და საიმედო განლაგებას.

DOM-ის დამუშავების დროს ბრაუზერი იყენებს id–ის, როგორც პრივილეგირებულ გზას, რომ მოხდეს ელემენტის მიკვლევა, ამიტომ იგი არის ყველაზე სწრაფი ლოკატორი. თუ web ელემენტს გადავაადგილებთ მიმდინარე გვერდზე და ლოკატორად განსაზღვრულია id, ტესტი მაინც იმუშავებს. ტესტი იმუშავებს იმიტომ, რომ განხილულ ელემენტს ვწვდებით არა მისი ადგილ-მდებარეობის მიხედვით, არამედ id-ით.

ელემენტის ლოკატორად id ატრიბუტის გამოყენება არის ყველაზე კარგი სტრატეგია, მაგრამ ხშირ შემთხვევაში შეუძლებელია id–ის გამოყენება შემდეგი მიზეზების გამო:

- გვერდზე მოცემულ ყველა ელემენტს არ აქვს id ატრიბუტი;
- id ატრიბუტის მნიშვნელობა გენერირდება დინამიურად.

აღნიშნულ შემთხვევებში იყენებენ name ლოკატორებს. განსხვავებით id-სგან, name ატრიბუტი შეიძლება არ იყოს უნიკალური. მიმდინარე გვერდზე შესაძლებელია არსებობდეს რამდენიმე ერთნაირი name ატრიბუტის მქონე ელემენტი. მსგავს სიტუაციაში Selenium-ი მიმართავს პირველ ნაპოვნ ელემენტს, რომელიც შეიძლება არ იყოს ტესტის სამიზნე ელემენტი. ეს გამოიწვევს ტესტის ჩავარდნას.

შევნიშნოთ, რომ არ არის აუცილებელი ელემენტს გააჩნდეს name ატრიბუტი, ისევე როგორც id.

id ლოკატორის ანალოგიურად, თუ მოცემულ ელემენტს გადავაადგილებთ მიმდინარე web გვერდზე, ტესტი მაინც იმუშავებს. ტესტი იმუშავებს იმიტომ, რომ გვერდის ელემენტს მივმართავთ არა მისი მდებარეობის მიხედვით, არამედ name ატრიბუტით.

როგორც უკვე აღვნიშნეთ, მოცემულ გვერდზე შეიძლება არსებობდეს ორი ერთიდაიმავე name-ს მქონე ელემენტი. მოცემულ სიტუაციაში ვიყენებთ ე.წ. ფილტრს, რომლითაც ვახდენთ საჭირო

ელემენტის იდენტიფიკაციას: Target ველში ვწერთ ისეთ ლოკატორს, რომელიც განსხვავებულია მეორე მსგავსი სახელის მქონე ელემენტის ლოკატორისგან. მაგალითად [5]:

name=verifybutton value=chocolate;

Selenium ტექნოლოგიით Web გვერდზე არსებულ ლინკზე მიმართვა ხდება ლინკის ტექსტის საშუალებით. თუ დავაკვირდებით ზემოთ განხილულ მაგალითებს, დავინახავთ, რომ ლინკის ლოკატორს აქვს შემდეგი ფორმატი: link=LinkText, მაგალითად, LinkText=Chapter2.

არსებობს ლინკები, რომელთა ტექსტი გენერირდება დინამიკურად. ასეთ შემთხვევაში ლინკის ლოკატორს აგებენ ნაწილობრივი ტექსტით, კერძოდ, ლინკის ტექსტის სტატიკური ნაწილით. თუ დინამიკურად გენერირებად ლინკის არ გააჩნია სტატიკური ტექსტი, მაშინ ლოკატორად იყენებენ სხვა ალტერნატიულ საშუალებებს: id, name, CSS, Xpath [5].

როგორც უკვე აღვნიშნეთ, XPath არის XML ტეგების ენა, რომელიც გამოიყენება კვანძების მოსანიშნად XML–ში. ყველა თანამედროვე ინტერნეტ ბრაუზერს აქვს Xpath-ის მხარდაჭერა, რადგან DOM-ში HTML გვერდი წარმოდგება როგორც XHTML დოკუმენტი. XPath ენა ბაზირებულია XML ხისებურ სტრუქტურაზე და გააჩნია შესაძლებლობა იმოძრაოს ხის გარშემო, მონიშნოს კვანძები სხვადასხვა კრიტერიუმების საშუალებით.

ელემენტის XPath ლოკატორი ძალიან მოქნილია და წარმოადგენს ერთ–ერთ ბოლო შემოთავაზებულ ლოკატორ–სტრატეგიას, თავისი ნელი შესრულებით. მისი ნელი მუშაობა განპირობებულია იმით, რომ ელემენტის ძებნისას DOM იერარქიულ ხეში შვილ კვანძზე გადასვლა ხდება მშობელი კვანძის გამოყენებით და ეს პროცესი გრძელდება მანამ, სანამ არ მივაღწევთ საძიებო კვანძს [10,11].

შესაძლებელია xpath ლოკატორის ოპტიმიზაცია მის კვანძებში ელემენტის web ატრიბუტების მითითებით.

Selenium ელემენტის xpath-ს აქვს ფორმატი: /xpath. მაშასადამე Selenium-ში xpath ლოკატორი მიიღება ჩვეულებრივ xpath-ს წინ დამატებული '/' სიმბოლო. ანუ Firebug-ის მიერ გენერირებულ XPath-ს წინ უნდა ემატება ერთი დახრილი ხაზი '/'.

Web ტექნოლოგიებში არის შემთხვევები, როდესაც შეუძლებელია ელემენტს მივანიჭოთ სტატიკური ატრიბუტები, მაგალითად, თუ მის დასახელებას ვაგებთ მონაცემთა ბაზიდან მიღებული მონაცემით, ატრიბუტების მნიშვნელობები გაწერილია დინამიკურად და იცვლება გვერდის განახლებასთან ერთად და სხვა. არის შემთხვევები, სადაც ელემენტის დასახელების მხოლოდ ნაწილი არის დინამიკური. XPath ტექნოლოგიას შეუძლია მიმართოს გვერდის ელემენტს ატრიბუტის ნაწილობრივი მნიშვნელობით. განვიხილოთ XPath ფუნქციები, რომლებიც უზრუნველყოფს ელემენტებზე წვდომას ნაწილობრივი მნიშვნელობებით: starts-with(), ends-with() და contains().

starts-with() ფუნქცია აიდენტიფიცირებს ელემენტს საწყისი ლოკალური მნიშვნელობით. მაგალითად, თუ ელემენტის ID არის ctrl_12, სადაც ctrl არის ლოკალური და 12 დინამიკური მნიშვნელობა, აღნიშნული ფუნქცია განსაზღვრავს ელემენტის ადგილმდებარეობას ctrl მნიშვნელობით შემდეგნაირად:

```
//input[starts-with(@id, 'ctrl_')];
```

ends-with() ფუნქცია აიდენტიფიცირებს ელემენტს ბოლო ლოკალური მნიშვნელობით. მაგალითად, თუ ელემენტის ID არის a_1_userName, სადაც userName არის ლოკალური, მოცემული ფუნქცია განსაზღვრავს ელემენტის ადგილმდებარეობას _ userName მნიშვნელობით:

```
//input[ends-with(@id, '_userName')];
```

contains() ფუნქცია აიდენტიფიცირებს ელემენტს ატრიბუტის დასახელების შემცველი მნიშვნელობით, ქვე–ტექსტით. მაგალითად, თუ ელემენტის ID არის panel_login_userName_textfield, მოცემული ფუნქცია განსაზღვრავს ელემენტს userName ქვე–ტექსტით:

```
//input[contains(@id, 'userName')];
```

საბოლოოდ შევნიშნოთ რომ XPath–ს გააჩნია მრავალი წესი, ფუნქცია, სინტაქსი, რომ მიმართოს გვერდის ნებისმიერ ელემენტს. ერთიდაიგივე ელემენტის xpath შეიძლება აიგოს მრავალნაირად, თითოეულ კვანძში ატრიბუტების დამატებით, კვანძების ინდექსირებით, ქვე–ტექსტებით და სხვა.

სხვა ლოკატორებთან შედარებით Selenium-ის XPath ლოკატორის ერთადერთ ნაკლად შეიძლება განვიხილოთ მისი ნელი შესრულება. ეს ნაკლი განსაკუთრებით შესამჩნევია IE-ბრაუზერზე (Internet Explorer).

CSS ლოკატორით ელემენტების ადგილმდებარეობის განსაზღვრა ხდება ელემენტის სტილით. ანალოგიურად XPath–ისა, DOM იერარქიულ ხეში CSS ლოკატორის აგება ხდება ხის კვანძების გამოყენებით. მშობელი კვანძიდან შვილობილ კვანძზე გადავდივართ '>' სიმბოლოთი. ერთიდაიმავე ელემენტის CSS ლოკატორი შეიძლება აიგოს მრავალნაირად, მაგალითად:

```
css=input[id='but1'];
css=input#but1;
css=input [id='but1'][value='Button with ID'];
```

CSS ლოკატორებს აქვს ნაკლი, რაც ვლინდება მათ ელემენტის სტრუქტურისადმი დამოკიდებულებაში. თუ სტრუქტურა შეიცვლება, მსგავსად xpath–სა, ლოკატორი ვეღარ იპოვნის ელემენტს და ტესტი ჩავარდება [5].

3.5. Selenium ტესტებში JavaScript-ის გამოყენება

JavaScript (JS) არის მსოფლიოში ერთ–ერთი ყველაზე პოპულარული ენა [96]. JavaScript არის სკრიპტების ენა, რომლის საშუალებით შესაძლებელია დინამიკური სკრიპტების დაწერა HTML კოდში. მას აღიქვავს ყველა თანამედროვე ინტერნეტ–ბრაუზერი. JavaScript და JAVA არის ორი ერთმანეთისგან აბსოლიტურად განსხვავებული ენა თავისი დიზაინით და კონცეფციით [60]. მათ აქვთ ერთმანეთისგან მთლიანად განსხვავებული სემანტიკა. JS სინტაქსი უფრო ახლოს დგას C ენასთან. JAVA არის უფრო კომპლექსური პროგრამირების ენა, რომელიც Sun-ის მიერაა შექმნილი. ორიგინალური JavaScript–ი შექმნილია Netscape-ს მიერ.

JavaScript-ში, ისევე როგორც უმრავლეს სკრიპტულ ენებში, ცვლადის ტიპს განსაზღვრავს მისი მნიშვნელობა. მაგალითად, ცვლადი x შეიძლება განისაზღვროს როგორც number მასზე რამე რიცხვის მინიჭებით, შემდეგ იგივე ცვლადი გახდეს string ტიპის, მასზე ტექსტური მნიშვნელობის მინიჭებით და ა.შ. [6].

ქვემოთ განვიხილავთ თუ როგორ შეიძლება გამოვიყენოთ JavaScript ტესტ-სკრიპტებში. Selenium ტესტში რომ განვსაზღვროთ რაიმე JavaScript კოდი, ამისთვის უნდა გამოვიყენოთ შემდეგი აღწერის ბლოკი.

JavaScript { javascript operators; **}**

სისტემურ ფრჩხილებში { } თავსდება JavaScript ოპერატორი, ან ოპერატორთა მიმდევრობა, ერთმანეთისგან წერტილ-მძიმით გამოყოფილი. Selenium ტესტებში JS ოპერატორები სრულდება ჩვეულებრივი მიმდევრობით და შედეგი ბრუნდება ბოლო ოპერატორის საშუალებით, მაგალითად:

```
var  a, b ;
a = 2 ; b = 5 ;
a + b ;
```

განხილული ფრაგმენტი Selenium-ს შედეგად დაუბრუნებს 7-ს.

Selenium სისტემა დაწერილია JavaScript ტექნოლოგიით. ამის გამო შესაძლებელია web გვერდზე მანიპულირება JS სტანდარტული ფუნქციებით. ეს გულისხმობს იმას, რომ თუ Selenium ინსტრუმენტი არ გვთავაზობს საჭირო ფუნქციას, შესაძლებელია ტესტებში წმინდა JavaScript-ის გამოყენება.

Selenium ტესტ-სკრიპტებში JS ფუნქციების გამოყენებას მიმართავენ მაშინ, როდესაც დასატესტირებელია რთულად ტესტირებადი სცენარები. თუ ტექნოლოგია არ არის თავსებადი JavaScript-თან, მაშინ Selenium უძლურია მოცემული სცენარის რეალიზაციისთვის, ამ შემთხვევაში მიმართავენ უფრო მძლავრი ინსტრუმენტების გამოყენებას, როგორებიცაა: CodedUI და Test Complete [8].

3.6. Selenium ბრძანებების გაფართოება და დამატებითი ფუნქციები

განვიხილოთ თუ როგორ შეიძლება Selenium ბრძანებათა სისტემის გაფართოება, რადგან Web ტესტირებისას არის შემთხვევები სადაც Selenium სტანდარტული ბრძანებები არ არის საკმარისი.

Selenium ტექნოლოგია დეველოპერებს აძლევს შესაძლებლობას, რომ განსაზღვრონ და შექმნან ახალი ფუნქციები – მსგავს სამ სვეტიან ფორმატში. იმის გამო რომ Selenium სისტემა დაწერილია JavaScript-ზე, ამიტომ დამატებითი ფუნქციების რეალიზება შესაძლებელია მხოლოდ JavaScript-ით.

წარმოვიდგინოთ შემთხვევა, როდესაც ვიყენებთ ერთიდაიმავე JavaScript კოდის ფრაგმენტს სხვადასხვა ტესტში:

type | locator | javascript{ ... }

ვთქვათ სცენარის შესრულებისას გვინდა მიმდინარე თარიღის გამოთვლა და საიტის რამდენიმე გვერდზე ჩაწერა

"dd/mm/yyyy" ფორმატით. დასმულ ამოცანას გადავჭრით JavaScript ოპერატორების გამოყენებით, ტესტ–სკრიპტში JS პროგრამული ფრაგმენტის რეალიზებით. თუ მოგვიანებით აღმოვაჩენთ, რომ JS ფრაგმენტის რეალიზაციისას დაშვებულია შეცდომა, მაშინ საჭირო იქნება ყველა იმ ტესტის ცვლილება, სადაც გამოყენებულია მოცემული JS ფრაგმენტი. ეს მიდგომა, როგორც პროგრამული დეველოპმენტიდან არის ცნობილი, არის მცდარი, რადგან ამ დროს მიმდინარეობს ერთიდაიგივე შესწორების მრავალჯერადი გამეორება [6].

აღნიშნული პრობლემის გადასაჭრელად Selenium გვთავაზობს ფუნქციების აღწერას, რომელებსაც გამოვიძახებთ ნებისმიერ ტესტ-სკრიპტში. წარმოდგენილი პრაქტიკული მაგალითისთვის აღვწერთ ფუნქცია typeCurrentDate-ს, რეალიზაციას გავაკეთებთ JavaScript ოპერატორებით და ტესტ-სკრიპტში გამოვიძახებთ მისი სახელის (typeCurrentDate) საშუალებით. აღწერილი სტრატეგია გაცილებით მოქნილია, რადგან დეფექტის პოვნის შემთხვევაში საჭირო იქნება მხოლოდ აღწერილი ფუნქციის ერთჯერადი გასწორება და არა სხვადასხვა ადგილას ერთიდაიმავე შესწორებების შეტანა [6].

მომხმარებლის მიერ რეალიზებულ ფუნქციებს უწოდებენ Selenium გაფართოებად ბრძანებებს. ისინი ინახება ცალკე ფაილში. გაფართოებადი ბრძანებების გამოყენების დროს საჭიროა მათი ფაილის მისამართი მივაბათ Selenium IDE-ს ან Selenium RC-ს. აღნიშნული ფაილები წარმოადგენს JavaScript ფაილებს .js გაფართოებით. js ფაილში იწერება მომხმარებლის მიერ განსაზღვრული ფუნქცია, რეალიზებული JavaScript სინტაქსით.

Selenium გაფართოებადი ბრძანებები იწერება პროტოტიპული ენით. იმისთვის რომ განვსაზღვროთ დამატებითი ფუნქცია, მისი აღწერის ბლოკი უნდა იყოს შემდეგი სახის:

```
Selenium.prototype.doFunctionName = function()
{
  . . .
}
```

"do" იწერება ფუნქციის დასახელების წინ, რომელიც Selenium-ს მიუთითებს რომ აღწერილი ფუნქცია შეიძლება გამოყენებულ იქნას როგორც ჩვეულებრივი Selenium ბრძანება.

წარმოვიდგინოთ, რომ გვინდა გავტესტოთ აპლიკაციის რაიმე ფუნქცია, რომელიც მოითხოვს შემთხვევით აღებული რიცხვების ჩაწერას „ტექსტბოქსში". ზოგადად, შემთხვევით აღებულ რიცხვებს, ე.წ Random-ით მიღებულ რიცხვით მნიშვნელობებს დიდი გამოყენება აქვს ტესტირებისას. მაგალითად, კალკულატორის ტესტირების დროს, პროგრამაში შესატანი მნიშვნელობების განსაზღვრის დროს და ა.შ. მოცემული ამოცანა რომ გავამარტივოთ, ამისათვის საჭიროა შევქმნათ Selenium-ის დამატებითი ფუნქცია (მაგალითად, storeRandom) და მისი მუშაობის შედეგი შევინახოთ რაიმე Selenium ცვლადში [5].

აღნიშნული დამატებითი ბრძანების რეალიზაციისას დაგვჭირდება ფუნქციის არგუმენტების გამოყენება. ფუნქციის დასახელებაში მოცემულ ცვლადებს ეწოდება ფუნქციის არგუმენ-ტები. ფუნქციის განსაზღვრისას მის დასახელებაში მოცემული პირველი არტგუმენტი ასოცირდება „Target" ველთან, ხოლო მეორე არგუმენტი „Value" ველთან.

განვიხილოთ აღწერილი ამოცანის რეალიზება დამატებითი ფუნქციის საშუალებით:

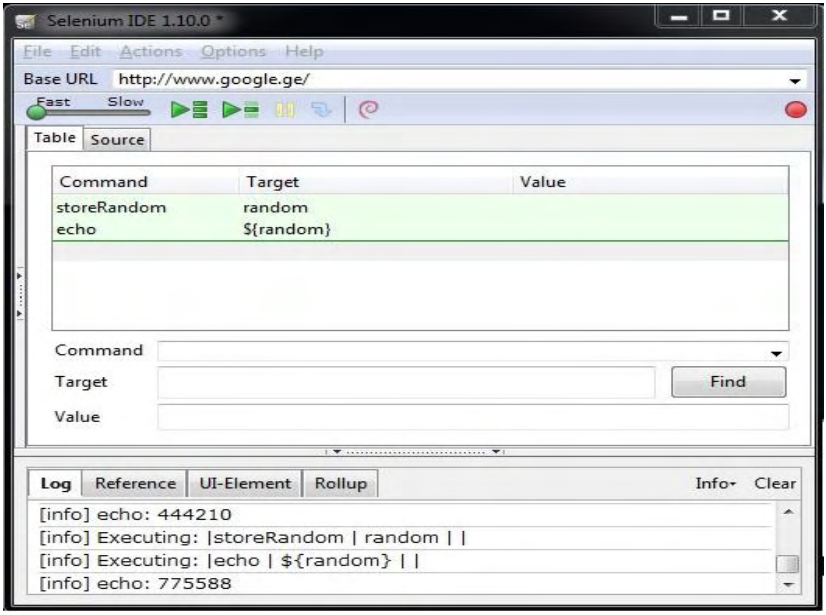
1. ავამუშავოთ ტექსტური რედაქტორი და გავხსნათ უკვე შექმნილი user-extensions.js ფაილი;

2. შევქმნათ ახალი ფუნქცია სახელწოდებით storeRandom. storeRandom ფუნქცია უნდა გამოიყურებოდეს შემდეგნაირად:

```
Selenium.prototype.doStoreRandom = function(variableName)
  {
    random = Math.floor(Math.random() * 1000000);
    storedVars[variableName] = random;
  }
```

3. შევინახოთ ფაილი და გადავტვირთოთ Selenium IDE;
4. Selenium IDE-ში შევქმნათ ბრძანება storeRandom და გადავცეთ რაიმე ცვლადი, რომელსაც მიენიჭება ჩვენს მიერ განსაზღვრული ფუნქციის მუშაობის შედეგი;
5. შევქმნათ echo ბრძანება, რათა დავინახოთ ფუნქციის მუშაობის შედეგი.

3.12 ნახაზზე მოცემულია დამატებითი ფუნქციის მაგალითი Selenium სისტემაში.



ნახ.3.12. Selenium IDE: დამატებითი ფუნქციის მაგალითი

განხილულ მაგალითში ვნახეთ თუ როგორ შეიძლება აღვწეროთ დამატებითი Selenium ფუნქცია, რომელიც მუშაობის შედეგს დააბრუნებს არგუმენტად გადაცემულ ცვლადში. აღწერილი ფუნქცია Selenium ცვლადზე წვდომისთვის იყენებს storedVars ოპერატორს. თუ დავაკვირდებით ამ ნახაზს, დავინახავთ რომ დამატებთი ბრძანების მუშაობის შედეგი იწერება Log-ში echo ბრძანების საშუალებით [6].

შევნიშნოთ, რომ, თუ დამატებით ფუნქციის აღწერისას javascript კოდში დავუშვით შეცდომა, Selenium IDE-ს გადატვირთვის დროს ამოვარდება შეცდომის აღწერის შეტყობინება.

3.7. Selenium Remote Control (RC) სერვერი

ახლა განვიხილოთ Selenium ტექნოლოგიის სერვერი – **Selenium Remote Control**, მისი დაყენების და გაშვების ინსტრუქცია, არგუმენტები, ზოგადი მახასიათებლები და ა.შ. **Selenium Remote Control (შემოკლებით RC) არის ერთ–ერთი ყველაზე პოპულარული, გამოყენებადი Selenium პროდუქტი**, რომელსაც დიდი პრაქტიკული გამოყენება აქვს Web აპლიკაციების ავტომატური ტესტირებისას. Selenium RC დეველოპერებს საშუალებას აძლევს ტესტები დაწერონ მათთვის სასურველ დაპროგრამების ენაზე. აქ განვიხილავთ Windows ოპერაციულ სისტემაზე Selenium RC-ს ინსტალაციის ინსტრუქციას, Selenium IDE-ს მიერ შექმნილი სცენარის გაშვებას RC სერვერზე და სხვა [6-13].

Selenium Remote Control არის დაწერილი Java ტექნოლოგიით, არის jar გაფართოების ფაილი. მაშასადამე, რადგან Java არის პლატფორმისგან **დამოუკიდებელი** პროგრამირების ენა, ამიტომ RC-ს გაშვება შესაძლებელია ნებისმიერ თანამედროვე პლატფორმაზე: Mac, Linux, Windows, Solaris და სხვა [5,6].

Selenium IDE მუშაობს მხოლოდ Firefox (FF) ბრაუზერზე. IDE წარმოადგენს FF ბრაუზერის პლაგინს, რომლის საშუალებით შესაძლებელია ტესტ-სკრიპტების ჩაწერა ავტომატურ რეჟიმში. ჩვენ, როგორც Web დეველოპერებმა, ან ტესტერებმა ვიცით, რომ Web აპლიკაციის მომხმარებლები არ იყენებენ მხოლოდ ერთ ბრაუზერს. ისინი Firefox-ის გარდა შეიძლება იყენებდნენ Internet Explorer-ს, Google chrome-ს, Safari–ს და ა. შ.

აღნიშნულიდან გამომდინარე შეგვიძლია ვთქვათ, რომ Selenium IDE ინსტრუმენტის ძირითად ნაკლოვანებას წარმოადგენს მისი ბრაუზერზე **დამოკიდებულება** (მუშაობს მხოლოდ Firefox-ზე).

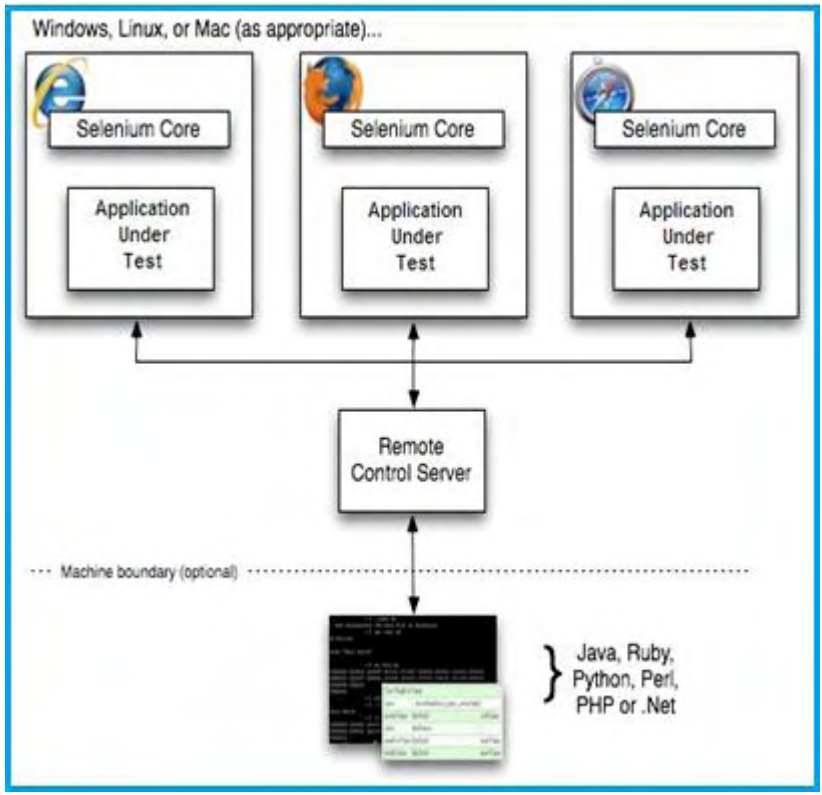
Selenium RC ტესტირების ინსტრუმენტის შექმნის მიზანი იყო ტესტების გაშვება სხვადასხვა ბრაუზერზე. ფაქტობრივად, იგი შექმნეს როგორც proxy შუალედური სერვერი აპლიკაციასა და ტესტ-სცენარს შორის. RC შეიცავს Selenium Core-ს, რომელიც უშვებს ბრაუზერს, კითხულობს და ასრულებს **Selense** ბრძანებებს, ინახავს ტესტირების საშედეგო ფაილს (რეპორტს) და სცენარის დასრულე-ბისას „კლავს" ბრაუზერის სესიას.

Selenium RC შეიცავს თანამედროვე პროგრამული ენების ბიბლიოთეკას, რომლის გამოყენებით შესაძლებელია ტესტების წერა ისეთ პოპულარულ ენებზე როგორიცაა: **Java, C#, Ruby, Python, Perl, PHP** და სხვა.

ახლო წარსულში დეველოპერები ტესტებს ქმნიდნენ ძირითადად Java და C# ობიექტორიენტირებულ ენებზე. რაც გამოწვეული იყო იმით, რომ აპლიკაციები იქმნებოდა ერთ-ერთ მათგანზე.

ბოლო დროს მიმდინარეობს დინამიკური ენების მძლავრი პოპულარიზაცია, დეველოპერები გადადიან მსგავსი ტიპის ენების გამოყენებაზე. Ruby და Python ყველაზე პოპულარული თანამე-დროვე ენებია, რომლებზეც მიმდინარეობს პროგრამისტების მიგრაცია [7].

პროგრამული ენის გამოყენება HTML ფორმატის ტესტ–სკრიპტების ნაცვლად უზრუნველყოფს აზრიან, გასაგებ, პროგრამულად მოქნილ, კარგად სტრუქტურირებად ტესტ-სკრიპტის შექმნას. დაპროგრამების ენით დაწერილ ტესტებში შესაძლებელია ციკლების, პირობითი ოპერატორების გამოყენება, რაც ამარტივებს სკრიპტის შექმნას და გაცილებიდ კითხვადს ხდის ტესტს [6-14]. მოცემულ ნახაზზე წარმოდგენილია Selenium RC სერვერის არქიტექტურის დიაგრამა (ნახ.3.13) [6]:



ნახ.3.13. Selenium RC არქიტექტურის დიაგრამა

განვიხილოთ Windows ოპერაციულ სისტემაზე Selenium RC-ს ინსტალაციის ინსტრუქცია. შევნიშნოთ, რომ RC სერვერის გასაშვებად Windows სისტემაზე საჭიროა ჯავას JDK პლატფორმის ინსტალაცია, რადგან RC დაწერილია Java-ზე და წარმოდგენილია jar ფაილად. თუ კომპიუტერზე არ აყენია Java, წარმოდგენილი ინსტრუქციის შესრულებამდე გადმოწერეთ და დააყენეთ ბოლო ვერსიის Java JDK პლატფორმა.

1. ჩამოწერეთ Selenium RC მისამართიდან:

   http://docs.seleniumhq.org/download/

2. ჩამოწერილი selenium-server-standalone.jar ფაილი შეინახეთ ხისტ დისკოზე თქვენთვის სასურველ დირექტორიაში;

3. გაუშვით Windows ბრძანებათა ველი - cmd და კონსოლიდან მიმართეთ იმ დირექტორიას, სადაც შეინახეთ Selenium RC-ს jar ფაილი;

4. cmd-ში ჩაწერეთ და გაუშვით ბრძანება: **java -jar selenium-server-standalone.jar;**

5. მე-4 ბიჯზე გამოყენებული ბრძანებით ამუშავდება RC სერვერი (ნახ.3.14).

ნახ.3.14. cmd: Selenium RC-ს გაშვება

მაშასადამე, გადმოვწერეთ Selenium RC სერვერი და გავუშვით Windows ბრძანებათა ველიდან. იგი არის JAR გაფართოების ფაილი, რომლის გასაშვებად გამოიყენება ჩვეულებრივი jar ფაილის გამშვები ბრძანება: **java -jar FileName.jar;**

განხილული მასალიდან ვიცით, რომ Selenium IDE წარმოადგენს Firefox ბრაუზერის პლაგინს და მისი საშუალებით აპლიკაციების ტესტირება შესაძლებელია მხოლოდ Firefox-ზე.

თანამედროვე ტექნოლოგიებში Web-აპლიკაციებთან ურთიერთობისთვის მომხმარებლები იყენებენ სხვადასხვა ინტერნეტ ბრაუზერებს. ბრაუზერისა და ოპერაციული სისტემის კომბინაცია შესაძლოა ნიშნავდეს, რომ პროგრამისტმა ან ტესტერმა გაუშვას ტესტი 9-ჯერ ან მეტჯერ, რათა დარწმუნდეს, რომ აპლიკაცია მუშაობს ყველა პოპულარულ ბრაუზერისა და ოპერაციული სისტემის კომბინაციაზე [6].

წარმოვიდგინოთ რომ საჭიროა ტესტ-სცენარების გაშვება კომპიუტერზე, სადაც არ არის დაინსტალირებული Selenium IDE, ან საერთოდ არ გვინდა, რომ Firefox გამოვიყენოთ მატესტირებელ ბრაუზერად. მოცემული ამოცანის გადასაწყვეტად საჭიროა Selenium Remote Control სერვერის გამოყენება.

Selenium RC-ზე ტესტ-სუიტის (სცენარის) გასაშვებად უნდა გამოვიყენოთ –**htmlsuite** არგუმენტი. მისი საშუალებით Selenium-ს ვატყობინებთ, რომ გახსნას html ტიპის ტესტ-სუიტი. გაშვების დროს ასევე საჭიროა მივუთითოთ ტესტ-სუიტის ადგილმდებარეობა დისკოზე, საშედეგო ფაილის დასახელება და ჩაწერის ადგილი. ასევე საჭიროა საწყისი URL მისამართისა და ბრაუზერის დასახელების განსაზღვრა. საბოლოოდ, Selenium RC სერვერის გამშვები ბრძანება cmd-ში შეიძლება წარმოვადგინოთ შემდეგი ფრაგმენტით:

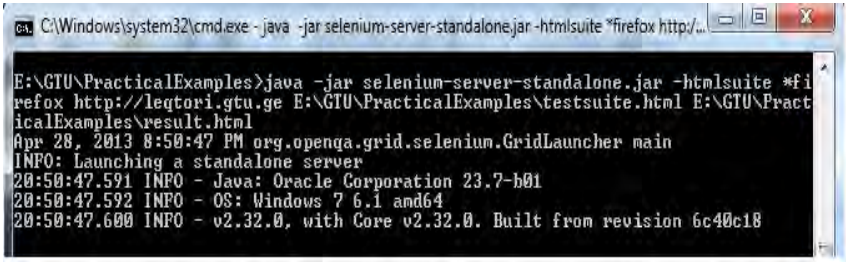
java -jar selenium-server-standalone.jar -htmlsuite *firefox http://book.theautomatedtester.co.uk c:\path\to\testuite.html c:\path\to\resultName.html

განვიხილოთ Selenium IDE-ს საშუალებით შექმნილი რომელიმე სცენარის Selenium RC-ზე გაშვების პრაქტიკული მაგალითი:

1. გაუშვით Windows ბრძანებათა ველი (cmd) და კონსოლიდან მიმართეთ იმ დირექტორიას, სადაც შეინახეთ Selenium RC-ს jar ფაილი;
2. cmd–ში ჩაწერეთ შემდეგი ბრძანება:

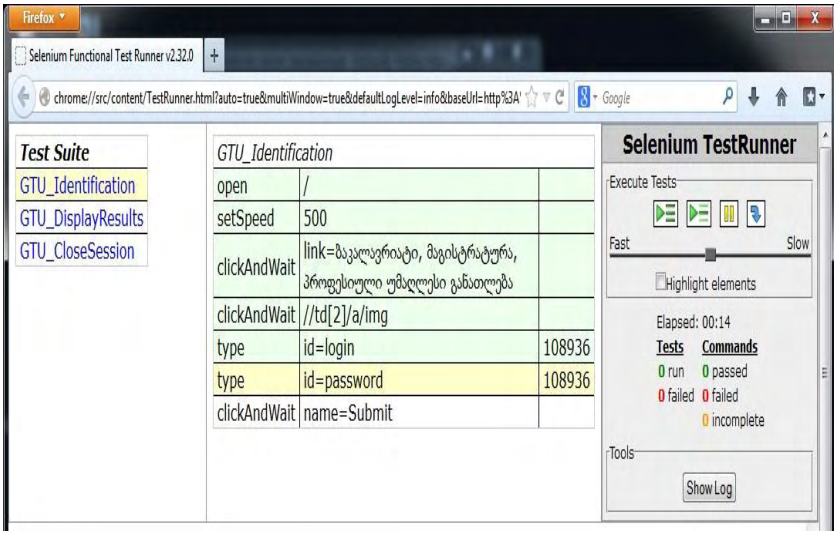
java -jar selenium-server-standalone.jar -htmlsuite *firefox http://leqtori.gtu.ge E:\GTU\PracticalExamples\testsuite.html E:\GTU\PracticalExamples\result.html

მოცემული ბრძანების შესრულების შედეგად გაეშვება Selenium RC სერვერი (ნახ.3.15) [6]:



ნახ.3.21. cmd: Selenium RC-ზე სცენარის გაშვება

მოცემული ტესტი დაიწყებს გაშვებას 2 ფანჯარაში. პირველი ფანჯარა არის Selenium Core ფრეიმვორკი, რომლის მარცხენა მხარეს მოთავსებულია ჩვენი ტესტ-სუიტი (ტესტ-სკრიპტების ჩამონათვალი), ფანჯრის ცენტრალურ ნაწილში წარმოდგენილია ტესტ-სკრიპტში მოცემული ბიჯები მიმდევრობით, ხოლო მის მარჯვენა მხარეს მოცემულია ტესტის მუშაობის შედეგი (ნახ. 3.16).

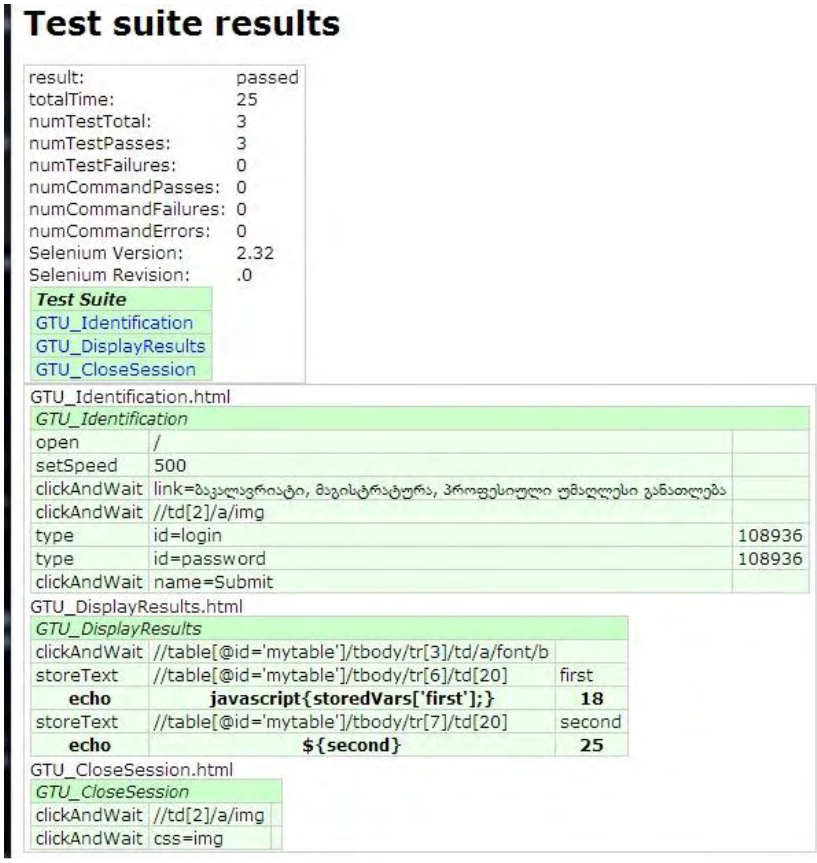


ნახ.3.16. Selenium Core: ტესტ–სკრიპტის შესრულების პროცესი

მეორე ფანჯარა ჩვეულებრივი Firefox ბრაუზერია, რომელშიც ავტომატურად სრულდება სცენარში გაწერილი მოქმედებები.

განხილულ მაგალითში Selenium IDE ტესტები გავუშვით Firefox ბრაუზერზე. cmd-ში გამოყენებული ბრძანებით განვსაზღვრეთ ბრაუზერი (Firefox) და გადავეცით საწყისი URL მისამართი (http://leqtori.gtu.ge). ასევე გადავეცით საშედეგო ფაილის ადგილმდებარეობა, თუ სად და რა დასახელებით უნდა შეინახოს საშედეგო (რეპორტ) ფაილი.

როდესაც ტესტი დაასრულებს მუშაობას, Selenium RC დააგენერირებს HTML ფორმატის საშედეგო ფაილს, რომელშიც ასახული იქნება თუ რომელმა სკრიპტმა გაიარა ტესტი და რომელი სკრიპტი „ჩავარდა“. 3.17 ნახაზზე წარმოდგენილია Selenium RC-ს მიერ გენერირებული საშედეგო ფაილი.



Test suite results

result:	passed
totalTime:	25
numTestTotal:	3
numTestPasses:	3
numTestFailures:	0
numCommandPasses:	0
numCommandFailures:	0
numCommandErrors:	0
Selenium Version:	2.32
Selenium Revision:	.0

Test Suite
GTU_Identification
GTU_DisplayResults
GTU_CloseSession

GTU_Identification.html

GTU_Identification		
open	/	
setSpeed	500	
clickAndWait	link=ბაკალავრიატი, მაგისტრატურა, პროფესიული უმაღლესი განათლება	
clickAndWait	//td[2]/a/img	
type	id=login	108936
type	id=password	108936
clickAndWait	name=Submit	

GTU_DisplayResults.html

GTU_DisplayResults		
clickAndWait	//table[@id='mytable']/tbody/tr[3]/td/a/font/b	
storeText	//table[@id='mytable']/tbody/tr[6]/td[20]	first
echo	**javascript{storedVars['first'];}**	**18**
storeText	//table[@id='mytable']/tbody/tr[7]/td[20]	second
echo	**${second}**	**25**

GTU_CloseSession.html

GTU_CloseSession	
clickAndWait	//td[2]/a/img
clickAndWait	css=img

ნახ.3.17. Selenium RC: ტესტის რეპორტ ფაილი

შევნიშნოთ, რომ თუ რომელიმე სკრიპტი „ჩავარდა“, ანუ ვერ გაიარა ტესტი, იგი საშედეგო ფაილში მოიცემა წითელი ფერით.

Internet Explorer არის ერთ-ერთი ყველაზე პოპულარული ბრაუზერი მომხმარებლებთა შორის. იგი მოყვება Microsoft ოპერაციულ სისტემებს. მიმდინარე მომენტისთვის Windows ოპერაციულ სისტემებს იყენებს მსოფლიოს კომპიუტერების 90%.

აღნიშნულიდან გამომდინარე საჭიროა ჩვენ, როგორც ტესტერებმა ან პროგრამისტებმა უზრუნველვყოთ პროგრამული კოდის მუშაობა ბოლო ვერსიის IE ბრაუზერზე მაინც. თუმცა არსებობენ ადამიანები, რომლებიც ჯერ კიდევ იყენებენ ძველი ვერსიის ბრაუზერს – IE5.

თანამედროვე ინტერნეტ სამყაროში ერთ-ერთი გავრცელებადი და პოპულარული ბრაუზერია Google Chrome. ლოგიკურია პრაქტიკაში გაჩნდეს რომელიმე Web აპლიკაციის ტესტირების მოთხოვნა ისეთ პოპულარულ ბრაუზერებზე, როგორიცაა Google Chrome და Internet Explorer [4-15].

Selenium RC სერვერის საშუალებით შესაძლებელია Google Chrome-ს გაშვება Firefox-ის ანალოგიურად, თუ გამშვებ ბრძანებაში *firefox-ის ნაცვლად დავწერთ *googlechrome-ს.

მაშასადამე, გამშვები ბრძანების მარტივი ცვლილებით შესაძლებელია სცენარების სხვადასხვა ბრაუზერზე გაშვება. Chrome ბრაუზერზე სცენარის გაშვება განხორციელდება ტესტის ცვლილების გარეშე.

ეს არის Selenium ტექნოლოგიის ერთ-ერთი მძლავრი მახასიათებელი, რის გამოც იგი ითვლება ერთ-ერთ ფავორიტ უფასო მატესტირებელ ტექნოლოგიად.

ზუსტად ანალოგიურად, Selenium RC სერვერის საშუალებით შესაძლებელია IE-ის გაშვება Google Chrome-ის ანალოგიურად, თუ გამშვებ ბრძანებაში *googlechrome-ის ნაცვლად დავწერთ *iexplore-ს.

მაშასადამე შეგვიძლია ვთქვათ, რომ Selenium RC სერვერის გამოყენებით ტესტები გავუშვით ისეთ თანამედროვე ბრაუზერებზე როგორიცაა: FF, IE და Google Chrome [6].

განხილული ინტერნეტ ბრაუზერების გარდა არსებობს სხვა ბრაუზერებიც, მაგალითად, Opera, Konquerer, Safari და სხვა.

RC-ზე რომ გავუშვათ სხვა ბრაუზერები, უნდა ვიხელმძღვანელოთ შემდეგი სიით:

- ***firefox**
- ***mock**
- ***firefoxproxy**
- ***pifirefox**
- ***chrome**
- ***iexplore**
- ***firefox3**
- ***safariproxy**
- ***googlechrome**
- ***konqueror**
- ***firefox2**
- ***safari**
- ***piiexplore**
- ***firefoxchrome**
- ***opera**
- ***iehta**
- ***custom**

თუ რომელიმე ინტერნეტ ბრაუზერი არ არის მოცემული სიაში, მაშინ მისი გაშვება შესაძლებელია *custom ბრძანებით და ბრაუზერის გამშვები გზის მითითებით [6].

განვიხილოთ Selenium RC-ს კიდევ ერთი პოპულარული არგუმენტი –**port**. რადგან Selenium RC მოქმედებს როგორც proxy სერვერი ტესტ-სცენარსა და აპლიკაციას შორის, ამიტომ მას ესაჭიროება რაღაც პორტი რომ დაელოდოს ბრძანებებს. ამისთვის იგი „გაჩუმების“ პრინციპით (default) იყენებს 4444 პორტს. იმ შემთხვევაში თუ მოცემული პორტი დაკავებულია, ან სერვისი ჩარჩენილია, შესაძლებელია სხვა პორტის გამოყენება შემდეგი ფრაგმენტით: **-port <port number>,** რაც საშუალებას გვაძლევს გავუშვათ RC სერვერი სხვ პორტზე პრობლემების გარეშე [5]. გარდა ჩამოთვლილი არგუმენტებისა, არსებობს RC-ს კიდევ მრავალი სხვა არგუმენტი:

- **-setTimeOut <timeout value>** მოცემული არგუმენტით განისაზღვრება Selenium RC-ს მუშაობის მაქსიმალური დრო მილისეკუნდებში (1 წმ = 1000 მს). გაჩუმების პრინციპით RC–ის მაქსიმალური მუშაობის დრო არის 30 წუთი.
- **-singleWindow** ეს არგუმენტი უზრუნველყოფს Selenium RC-ს გაშვებას ერთ ფანჯარაში. მოცემული RC-ს გაშვების ერთფანჯრიანი

რეჟიმი გაცილებით სწრაფია, ამიტომ ძირითადად მას იყენებენ აპლიკაციების IE-ზე ტესტირებისას. IE-ზე ტესტირება გაცილებით ნელია ვიდრე FF–ზე.

ზოგადად Selenium RC სერვერის არგუმენტთა სიის ნახვა შესაძლებელია შემდეგი cmd ბრძანებით: **java -jar selenium-server-standalone.jar -h**

Windows ოპერაციულ სისტემაზე Selenium RC სერვერის გაშვება შესაძლებელია batch გამშვები ფაილის განსაზღვრითაც. მსგავსი ტიპის ფაილებს აქვს .bat გაფართოება. bat ფაილის განსასაზღვრად იყენებენ ნებისმიერ ტექსტურ რედაქტორს და მასში წერენ ჩვეულებრივ RC-ს გამშვებ ბრძანებას (cmd ბრძანება). სწორად განსაზღვრული bat ფაილის გაშვება შესაძლებელია მაუსის ორი კლიკის საშუალებით, რომლის შედეგად გაეშვება Selenium RC სერვერი.

ამგვარად, ჩვენ განვიხილეთ Selenium RC სერვერი, რომელიც არის დაწერილი Java ტექნოლოგიით, ე.ი არის პლატფორმისგან **დამოუკიდებელი**.

RC-ს საშუალებით შესაძლებელია ტესტ-სკრიპტების წერა ნებისმიერ თანამედროვე პროგრამირების ენაზე, მაშასადამე იგი არის **დაპროგრამების ენისგან დამოუკიდებელი**.

ასევე RC სერვერით შესაძლებელია ნებისმიერი ბრაუზერის გაშვება, ანუ იგი ასევე არის **ბრაუზერისგან დამოუკიდებელიც**.

ჩამოთვლილი უპირატესობები განაპირობებს მის პოპულარობას, თუმცა მასაც გააჩნია ტექნოლოგიური ნაკლოვანებები, რომელთა თავიდან აცილების მიზნით შეიქმნა RC-ს შემდგომი განვითარება Selenium WebDriver [5].

3.8. Selenium WebDriver ინსტრუმენტი

თანამედროვე ინფორმაციულ ტექნოლოგიებში მიმდინარეობს Web აპლიკაციების ყოველღიური განვითარება. იქმნება მრავალი სამომხმარებლო Web აპლიკაცია, რაც განპირობებულია Web სისტემების პოპულარობით. Web ტექნოლოგიების განვითარების პარალელურად ვითარდება ავტომატური მატესტირებელი ტექნოლოგიებიც. იქმნება ავტომატური მატესტირებელი ფრეიმვორკები, რომლებიც უზრუნველყოფს საიტების მუშაობის შემოწმებას, ტესტირებას.

Selenium IDE და Selenium RC წლების განმავლობაში გამოიყენება ბრაუზერის მართვის ავტომატიზაციისთვის. როდესაც Selenium შეიქმნა „Jason Huggins"-ის მიერ, წყვეტდა პრობლემებს, რომლებიც მიიღებოდა ბრაუზერში მომხმარებლის ურთიერთქმედებით. იგი არის კარგი ავტომატური ტესტირების ფრეიმვორკი, მაგრამ შემოიფარგლება JavaScript–ით.

მობილურ მოწყობილობებზე და HTML5 ტექნოლოგიაზე გადასვლის შემდეგ, ცხადი გახდა რომ Selenium RC სერვერი ვეღარ ასრულებდა თავის მოთხოვნას: ბრაუზერის ავტომატიზაცია, მომხმარებლის მოქმედებების შესრულება.

Simon Stewart-ს სურდა შეექმნა განსხვავებული ტექნოლოგია, რომელიც გადაწყვეტდა აღნიშნულ პრობლემებს. კომპანია ThoughtWorks-ში ყოფნისას მან დაიწყო WebDriver პროექტზე მუშაობა, რომლის პირველი რეალიზაცია განხორციელდა 2007 წელს [11,62].

WebDriver-ის რეალიზაციის განხორციელების შემდეგ ცხადი გახდა მნიშვნელოვანი სხვაობა მას და Selenium RC-ს შორის, რაც გამოიხატებოდა მათ API-ში (Application Programming Interface). Selenium RC-ს ჰქონდა ლექსიკონზე ბაზირებული API, ხოლო WebDriver–ს კი – უფრო ობიექტზე ორიენტირებული API.

თავდაპირველად WebDriver გვთავაზობდა მხოლოდ Java–ს მხარდაჭერას, ხოლო RC გვთავაზობდა მრავალი ენის მხარდაჭერას. მათ შორის ასევე იყო მნიშვნელოვანი ტექნოლოგიური სხვაობა: Selenium Core (რაზეც RC არის დაფუძნებული) რეალიზებულია JavaScript-ით, რომელიც ეშვება ბრაუზერში, ხოლო WebDriver ბრაუზერს მართვას ოპერაციული სისტემის დონეზე [4-11].

2009 წლის აგვისტოში გამოაცხადეს ამ ორი პროექტის გაერთიანება, რომელსაც უწოდეს Selenium WebDriver (იგივე Selenium-2). ამჟამად WebDriver-ს გააჩნია შემდეგი ენების მხარდაჭერა: Java, C#, Python და Ruby. იგი გვთავაზობს Google Chrome, Firefox, Internet Explorer, Opera, Android და iPhone ბრაუზერების მხარდაჭერას.

WebDriver მუშაობს ოპერაციული სისტემის დონეზე, რაც იმას ნიშნავს, რომ იგი ინტერნეტ ბრაუზერს ბრძანებებს უგზავნის ოპერაციული სისტემიდან. ეს ფრეიმვორკი არ შემოიფარგლება JavaScript–ით, როგორც ეს იყო Selenium–თან მიმართებაში. მისი საშუალებით შესაძლებელია მივწვდეთ ფაილურ სისტემას, მაშასადამე შესაძლებელია ფაილის ატვირთვის ტესტის რეალიზება. ტესტების გაშვებისას WebDriver არ მოითხოვს პროქსი სერვერის დაყენებას, როგორც ეს იყო Selenium RC-ს დროს.

WebDriver-ის უარყოფით მხარედ უნდა აღინიშნოს ის, რომ იგი მოითხოვს ახალ რეალიზაციას ბრაუზერის ყოველი ახალი ვერსიის გამოსვლის პარალელურად. ეს არის გამოწვეული იმით, რომ ყოველ ბრაუზერს გააჩნია თავის გამშვები წერტილი ოპერაციულ სისტემაში [5].

მაშასადამე, Selenium WebDriver არის Web აპლიკაციების ავტომატური ტესტირების ფრეიმვორკი, რომელსაც შეუძლია გაუშვას ტესტები სხვადასხვა ბრაუზერზე (ნახ.3.18) და სხვადასხვა მოწყობილობაზე.

ნახ.3.18. WebDriver–ის ბრაუზერებთან თავსებადობა

WebDriver აგრეთვე საშუალებას გვაძლევს შევქმნათ ტესტები თანამედროვე პროგრამირების ენის გამოყენებით, მაშასადამე:

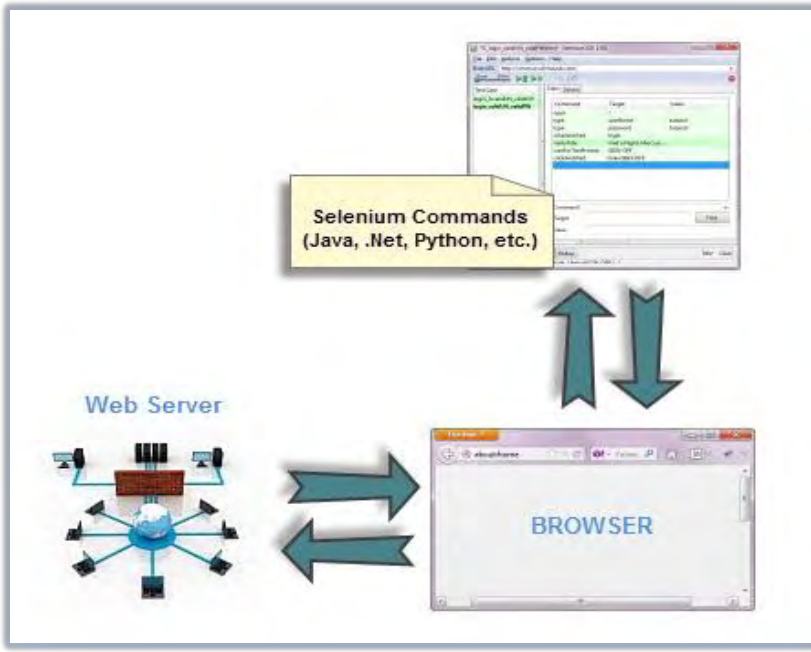
- შესაძლებელია პირობითი ოპერატორების გამოყენება: if-then-else ან switch-case;
- შესაძლებელია ციკლების გამოყენება: for, while, do-while

WebDriver-ის მიერ მხარდაჭერილი პროგრამირების ენებია: Java, .Net, PHP, Python, Perl და Ruby. ჩამოთვლილთაგან არ არის საჭირო ყველა ენის ცოდნა. საკმარისია ჩამოთვლილთაგან ვიცოდეთ ერთი რომელიმე ენა მაინც. ჩვენ განვიხილავთ Java პროგრამირების ენას და Eclipse პროგრამირების გარემოს.

თუ დავაკვირდებით **WebDriver**-ს და **Selenum RC**-ს, ორივეს გააჩნია სხვადასხვა ბრაუზერის და სხვადასხვა პროგრამირების ენის მხარდაჭერა. მაშასადამე რით განსხვადება ისინი ?

1. **არქიტექტურა**

WebDriver–ის არქიტექტურა Selenium RC-ს არქიტექტურასთან შედარებით გაცილებით მარტივია. იგი ბრაუზერის მართვას ახორციელებს ოპერაციული სისტემის საშუალებით. მის გასაშვებად საკმარისია პროგრამირების რომელიმე გარემო და ინტერნეტ ბრაუზერი. WebDriver-ის არქიტექტურის დიაგრამა ნაჩვენებია 3.19 ნახაზზე [5]:

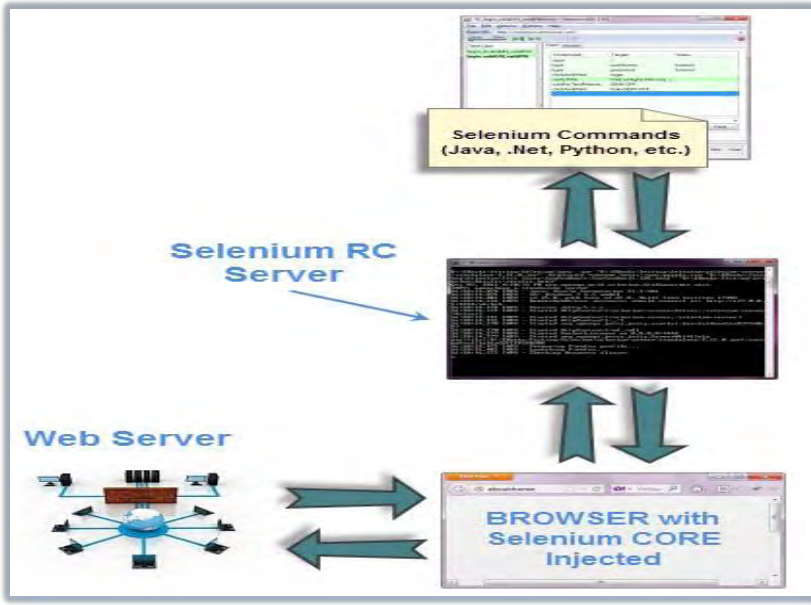


ნახ.3.19. WebDriver–ის არქიტექტურის დიაგრამა

Selenium RC არის უფრო რთული არქიტექტურის:

- თავდაპირველად, ტესტირების დაწყებისთვის საჭიროა ცალკე აპლიკაციის გაშვება, რომელსაც ეწოდება Selenium RC სერვერი;
- Selenium RC სერვერი მოქმედებს როგორც „შუაკაცი" Selenium ბრძანებებსა და ბრაუზერს შორის;
- როდესაც დაიწყება ტესტირება, Selenium RC სერვერს ბრაუზერში „შეჰყავს" Selenium Core;
- ინსტრუქციების მიღების შემდეგ Selenium Core უშვებს მათ როგორც Javascript ბრძანებებს.

Selenium RC სერვერის არქიტექტურის დიაგრამა შეიძლება წარმოვადგინოთ 3.20 ნახაზით.



ნახ.3.20. Selenium RC–ის არქიტექტურის დიაგრამა

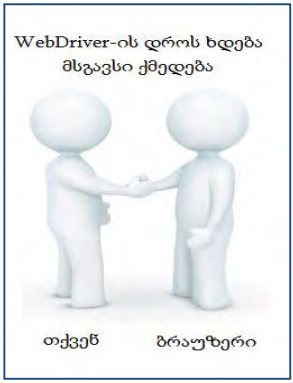
2. სიჩქარე

WebDriver არის გაცილებით სწრაფი ვიდრე Selenium RC, რადგან იგი პირდაპირ „ელაპარაკება" ბრაუზერს.

Selenium RC არის უფრო ნელი, ტესტების შესრულებისას იგი იყენებს დამატებით Javascript პროგრამას, რომელსაც ეწოდება Selenium Core. Selenium Core არის ინსტრუმენტი, რომელიც მართავს ბრაუზერს [5].

3. ურთიერთქმედება

WebDriver გვერდის ელემენტებთან ურთიერთქმედებს უფრო რეალური გზით. მაგალითად, თუ სატესტირებელ გვერდზე გვაქვს disabled „ტექსტ-ბოქსი", WebDriver-ს, ადამიანის მსგავსად არ შეუძლია მასში ტექსტის ჩაწერა.



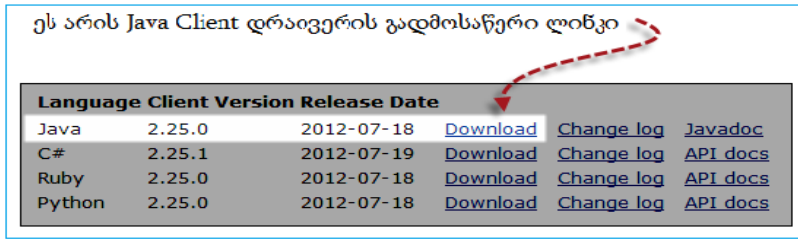
ნახ.3.21. ურთიერთქმედება-1

Selenium Core, ისევე როგორც JavaScript კოდი წვდება disabled ელემენტებს (ნახ.3.22). Selenium ტესტერები წარსულში გამოთქვამდნენ უკმაყოფილებას, რადგან Selenium Core-ს შეეძლო მნიშვნელობის შეყვანა disabled ელემენტში [5,6].

ნახ.3.22. ურთიერთქმედება-2

განვიხილოთ WebDriver–ის ინსტალაცია, მოვახდინოთ მისი კონფიგურაცია Eclipse ჯავა პროგრამირების გარემოში.

ვიგულისხმოთ, რომ უკვე დაინსტალირებული გვაქვს Java პლატფორმა (JDK) და Eclipse. Selenium-ის ოფიციალური საიტიდან გადმოვიწეროთ Java Client დრაივერი (ნახ.3.23).

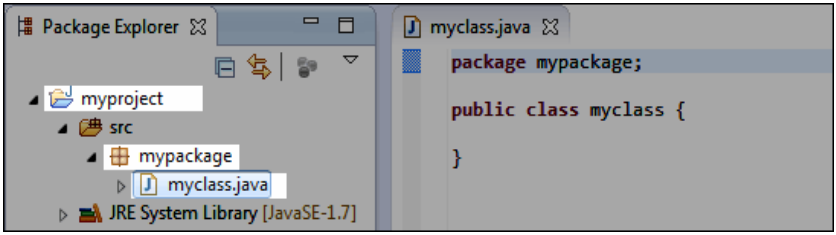


ნახ.3.23. Java Client დრაივერის ლინკი

გადმოწერის შედეგად მივიღებთ ZIP ფაილს სახელწოდებით selenium-<version number>.zip. სიმარტივისთვის ამოვაარქივოთ ZIP ფაილი C დისკზე, რის შედეგადაც მიიღებთ შემდეგ დირექტორიას C:\selenium-<version number>\. ეს დირექტორია შეიცავს JAR ფაილებს, რომლებსაც დავაიმპორტირებთ Eclipse-ში.

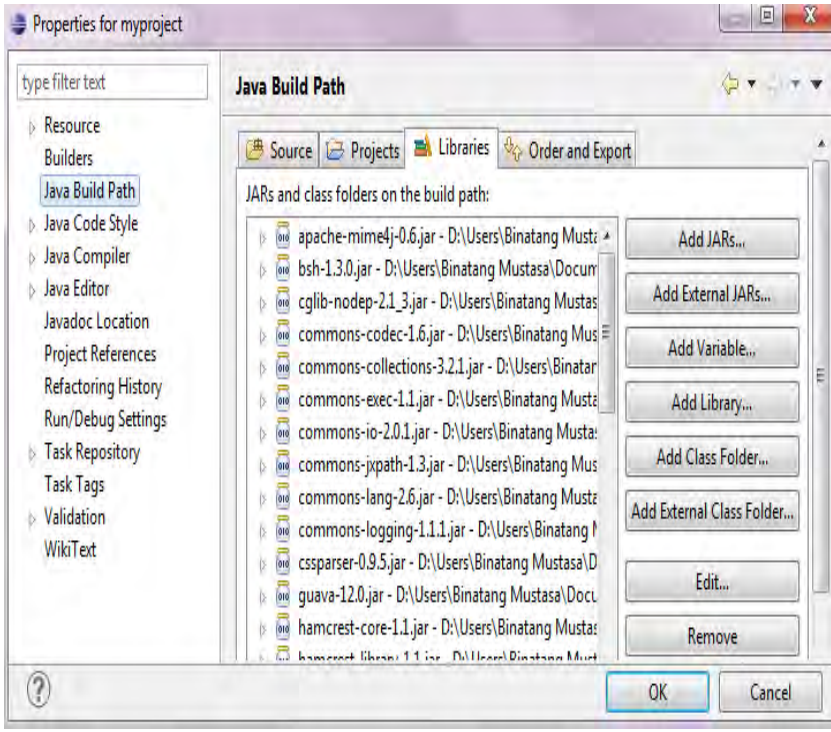
➢ **WebDriver-ის კონფიგურაცია Eclipse ინსტრუმენტში:**

1. გაუშვით პროგრამა Eclipse;
2. შექმენით ახალი პროექტი File -> New -> Java Project. პროექტს სახელად დაარქვით "myproject";
3. ახლად შექმნილ პროექტზე მაუსის მარჯვენა კლიკის საშუალებით გადადით New->Package. პეკიჯს დაარქვით სახელი "mypackage";
4. mypackage-ში შექმენით ახალი Java კლასი მასზე მაუსის მარჯვენა კლიკის საშუალებით და აირჩიეთ New -> Class. კლასის სახელი იყოს "myclass". თქვენი Eclipse ინსტრუმენტი დაახლოებით უნდა გამოიყურებოდეს.3.24 ნახაზის მსგავსად.



ნახ.3.24. Eclipse გარემო

5. myproject-ზე მაუსის მარჯვენა კლიკით - Properties-ზე;
6. დიალოგურ ფანჯარაზე მოვნიშნოთ „Java Build Path";
7. გადავიდეთ Libraries ტაბზე და შემდეგ დავაჭიროთ „Add External JARs...";
8. მივმართოთ C:\selenium-<version number>\ დირექტორიას (სადაც შევინახეთ ამოარქივებული Java Client დრაივერი);
9. მოცემული დირექტორიიდან დავამატოთ ყველა JAR ფაილი. ჩვენი Properties დიალოგური ფანჯარა უნდა გამოიყურებოდეს შემდეგნაირად (ნახ.3.25);
10. დასასრულს დააჭირეთ OK ღილაკს.



ნახ.3.25. პროექტის თვისებები

ამრიგად, აღწერილი ინსტრუქციით ჩვენს Eclipse პროექტში დავაიმპორტირეთ Selenium ბიბლიოთეკები.

WebDriver-ს აქვს ე.წ. „უჩინარი" ბრაუზერი, სახელწოდებით **HtmlUnit,** რომელიც ყველაზე სწრაფი ბრაუზერია.

იგი არის Java-ზე შექმნილი ბრაუზერი გრაფიკული ინტერფეისის გარეშე. სწორედ ინტერფეისის არარსებობა განაპირობებს ტესტების მაღალი სიჩქარით შესრულებას: Selenium აღარ ელოდება გვერდის ჩამოტვირთვას, ლინკზე გადასვლას, ელემენტის გამოჩენას და სხვ.

HTMLUnit და **Firefox** არის ორი ბრაუზერი, რომელთა ავტომატიზაციას WebDriver ახდენს პირდაპირ, რაც იმას ნიშნავს, რომ არ არის საჭირო ცალკეული დამატებითი კომპონენტების ინსტალაცია. სხვა ბრაუზერებისთვის საჭიროა დამატებითი კომპონენტები, რომელთაც ეწოდება Driver Server [9].

Driver Server განსხვავებულია ყოველი ბრაუზერისთვის. მაგალითად, Internet Explorer-ს აქვს თავის driver server, რომელსაც ვერ გამოვიყენებთ სხვა ბრაუზერისთვის. ქვემოდ მოცემულია ბრაუზერების სია და მათი შესაბამისი driver server:

- Chrome ChromeDriver
- Opera OperaDriver
- Safari SafariDriver
- Internet Explorer Internet Explorer Driver Server

შევქმნათ პირველი WebDriver სკრიპტი. გამოვიყენოთ ჩვენს მიერ შექმნილი კლასი „myclass" და დავწეროთ სკრიპტი რომელიც უზრუნველყოფს:

1. Google საწყისი გვერდის გახსნას;
2. მისი სახელწოდების შემოწმებას;
3. შემოწმების შედეგის ბეჭდას;
4. გაშვებული ბრაუზერის დახურვას.

ქვემოთ მოცემულია აღწერილი სცენარის შესაბამისი ლისტინგი WebDriver ჯავა კოდისთვის:

```
package mypackage;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class myclass {
   public static void main(String[] args) {
// declaration and instantiation of objects/variables
                WebDriver driver = new FirefoxDriver();
                String baseUrl = "https://www.google.com/";
```

```
        String expectedTitle = "Google";
        String actualTitle = "";
        // launch Firefox and direct it to the Base URL
        driver.get(baseUrl);
        // get the actual value of the title
        actualTitle = driver.getTitle();
        if (actualTitle.contentEquals(expectedTitle)) {
            System.out.println("Test Passed!");
          }
      else {
            System.out.println("Test Failed");
        }
        // close Firefox
        driver.close();
        // exit the program explicitly
        System.exit(0);
    }
}
```

- **განვიხილოთ რეალიზებული Java კოდი:**

import org.openqa.selenium.WebDriver; აიმპორტებს WebDriver კლასს, რომელიც საჭიროა driver ობიექტის აღწერისთვის.

import org.openqa.selenium.firefox.FirefoxDriver; აიმპორტებს FirefoxDriver კლასს, რომელიც ინიციალიზაციას უკეთებს driver ობიექტს.

თუ თქვენს მიერ აღწერილი ტესტი ასრულებს რთულ მოქმედებებს, უკავშირდება სხვა კლასებს, აკეთებს ბრაუზერის სქრინს ან მანიპულირებს გარე ფაილებზე, აუცილებლად დაგჭირდებათ სხვა პეკიჯების იმპორტი.

WebDriver driver = new FirefoxDriver(); მისი საშუალებით ხდება driver ობიექტის აღწერა, FirefoxDriver კლასი პარამეტრის

გარეშე ნიშნავს, რომ გაეშვება Firefox–ის default პროფილი. default Firefox პროფილი იგივეა, რაც Firefox ბრაუზერის გაშვება safe mode-თი (არ შეიცავს არანაირ გაფართოებას).

driver.get (baseUrl); WebDriver-ის **get()** მეთოდი გამოიყენება ბრაუზერის ახალი სესიის წარმოსაქმნელად, რომლის საწყისი URL-ი განისაზღვრება baseUrl ცვლადით.

actualTitle = driver.getTitle(); WebDriver კლასს აქვს **getTitle()** მეთოდი, რომელიც გამოიყენება მიმდინარე გვერდის სათაურის გასაგებად.

```
if (actualTitle.contentEquals(expectedTitle))
    {
            System.out.println("Test Passed!");
    }
  else {
        System.out.println("Test Failed");
     }
```

მოცემული პირობითი ოპერატორი უზრუნველყოფს მიმდინარე სათაურის შედარებას წინასწარ აღწერილ სათაურთან და შედეგად ბეჭდავს შემოწმების შედეგს.

driver.close(); close() მეთოდი გამოიყენება ბრაუზერის ფანჯრის დასახურად.

System.*exit*(0); ასრულებს Java პროგრამის შესრულებას**.** თუ თქვენ გამოიყენებთ მოცემულ ბრძანებას ბრაუზერის დახურვის ბრძანებამდე, მაშინ თქვენი Java პროგრამა დასრულდება და ბრაუზერი დარჩება გახსნილი.

რეალიზებული ტესტის გასაშვებად საჭიროა Eclipse-ს მენიუდან Run -> Run ბრძანებაზე გადასვლა. თუ ყველაფერი შესრულებულია სწორად, Eclipse გაუშვებს ბრაუზერს, გაივლის სცენარს და კონსოლში გამოიტანს "Test Passed!" შედეგს.

// იგივე ტესტის გაშვება IE-ზე:

```
package mypackage;
import java.io.File;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class myclass {
     public static void main(String[] args) {
      // declaration and instantiation of objects/variables
     File file = new File("C:\\iedriver\\IEDriverServer.exe");
     System.setProperty("webdriver.ie.driver", file.getAbsolutePath());
     WebDriver driver = new InternetExplorerDriver();
      //WebDriver driver = new InternetExplorerDriver();
            String baseUrl = "https://www.google.com/";
            String expectedTitle = "Google";
            String actualTitle = "";
              // launch Firefox and direct it to the Base URL
            driver.get(baseUrl);
             // get the actual value of the title
            actualTitle = driver.getTitle();
            if (actualTitle.contentEquals(expectedTitle)) {
                                    System.out.println("Test Passed!");
            }
              else {
                        System.out.println("Test Failed");
                  }
                        // close Firefox
                        driver.close();
                        // exit the program explicitly
                        System.exit(0);          }
}
```

Web გვერდის ელემენტებზე წვდომის გზა (ლოკატორი) მკვეთრად შეცვლილია Selenium2-ში. WebDriver-ს აქვს გაცილებით მდიდარი შესაძლებლობები გვერდის ელემენტის საპოვნელად. იგი ელემენტებს მიმართავს **By** სტატიკური კლასის საშუალებით. By კლასს გააჩნია შემდეგი მეთოდები:

- By.id("ifOfObject")
- By.linkText("TextUsedInTheLink")
- By.partialLinkText("partOfTheLink")
- By.tagName("theTHMLNodeType")
- By.className("cssClassOnTheElement")
- By.cssSelector("cssSelectorToTheElement")
- By.xpath("//Xpath/to/the/element")
- By.name("nameOfTheElement")

როგორც მოცემული სიიდან ჩანს, WebDriver განიხილავს 8 სახის ლოკატორს, მაშასადამე Selenium1–ის ლოკატორთა სიას დაემატა 3 სახის ლოკატორი: **tagName**, **className** და **partialLinkText.**

ბრაუზერის დასახურად WebDriver იყენებს 2 სახის ბრძანებას: close() და quit().

ნახ.3.26. close() და quit() ბრძანებები

close() ბრძანება დახურავს WebDriver ტესტის მიმდინარე ფანჯარას, ხოლო **quit()** ბრძანება უზრუნველყოფს WebDriver ტესტის მიერ გახსნილი ყველა ფანჯრის დახურვას. quit() ბრძანება ძირითადად გამოიყენება სცენარით გახსნილი pop-up ფანჯრების დასახურად. მაშასადამე, პირველი სურათი ახდენს close() ბრძანების იმიტაციას, ხოლო მეორე quit() ბრძანებისას [5].

განვიხილოთ ფაილის ატვირთვის ავტომატიზაცია bin.ge სერვერზე. ვთქვათ C: დისკოზე გვაქვს შენახული newhtml.html ფაილი, რომელიც WebDriver-ის საშუალებით უნდა ავტვირთოთ http://bin.ge სერვერზე. აღწერილი სცენარის შესაბამისი Java კოდი მოცემულია ქვემოთ:

```
package mypackage;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Upload {
   public static void main(String[] args) {
                String baseUrl = "http://bin.ge/";
                WebDriver driver = new FirefoxDriver();
                driver.get(baseUrl);
       WebElement uploadElement =
                driver.findElement(By.id("uploadfile_0"));
 // enter the file path onto the file-selection input field
   uploadElement.sendKeys("C:\\newhtml.html");
// check the "I accept the terms of service" check box
   driver.findElement(By.id("terms")).click();
   // click the "UploadFile" button
   driver.findElement(By.name("send")).click();
   }
}
```

ქვემოთ აღწერილი კოდით შესაძლებელია youtube web გვერდის ავტომატიზაცია:

```
package mypackage;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
public class Upload {
    private WebDriver driver;
    private String baseUrl;
    @BeforeClass
    public void setUp() throws Exception {
      driver = new FirefoxDriver();
      baseUrl = "http://www.youtube.com";
    }
    @Test
    public void test() throws Exception {
      driver.get(baseUrl + "/");
  driver.findElement(By.id("masthead-search-term")).clear();
      driver.findElement(By.id("masthead-search-term"))
     .sendKeys("louis armstrong what a wonderful world");
  driver.findElement(By.id("search-btn")).click();
  driver.findElement(By.xpath("//a[contains(text(),'Louis
          Armstrong  What A  Wonderful World')]")).click();
   Thread.sleep(180000);
 }
    @AfterClass
```

```
    public void tearDown() throws Exception {
      driver.quit();
    }
  }
```

3.9. მესამე თავის დასკვნა

განხილულ იქნა მართვის ავტომატიზებული სისტემების პროგრამული უზრუნველყოფის, Web-აპლიკაციების ავტომატური ტესტირება Selenium ტექნოლოგიით; ავტომატური ტესტირების გამოყენების სფეროები და მისი შედარება ხელოვნურ ტესტირებასთან, უპირატესობები და ნაკლოვანებები.

Selenium ტექნოლოგიის ზოგადი აღწერა, მისი შექმნის ისტორია და განვითარება. ფართოდ მიმოხილულია Selenium ტექნოლოგიის ძირითადი შემადგენელი მატესტირებელი ინსტრუმენტები ამოცანების პრაქტიკული რეალიზებით.

Selenium RC სერვერი, შექმნილი Java ტექნოლოგიით, არის პლატფორმისგან დამოუკიდებელი, დაპროგრამების ენებისგან დამოუკიდებელი და ბრაუზერისგან დამოუკიდებელი. ჩამოთვლილი უპირატესობები განაპირობებს მის პოპულარობას, თუმცა მასაც გააჩნია ტექნოლოგიური ნაკლოვანებები, რომელთა თავიდან აცილების მიზნით შეიქმნა RC-ს შემდგომი განვითარება Selenium WebDriver.

WebDriver მუშაობს ოპერაციული სისტემის დონეზე, იგი ინტერნეტ ბრაუზერს ბრძანებებს უგზავნის ოპერაციული სისტემიდან. ეს ფრეიმვორკი არ შემოიფარგლება JavaScript–ით, როგორც ეს იყო Selenium-თან მიმართებაში. მისი საშუალებით შესაძლებელია მივწვდეთ ფაილურ სისტემას, მაშასადამე შესაძლებელია ფაილის ატვირთვის ტესტის რეალიზება. ტესტების გაშვებისას WebDriver არ მოითხოვს პროქსი სერვერის დაყენებას, როგორც ეს იყო Selenium RC-ს დროს.

IV თავი
IT-სერვისების პარალელური ტესტირება

4.1 Selenium Grid: პარალელური ტესტირება

განვიხილავთ Selenium ტექნოლოგიის კიდევ ერთ ინსტრუმენტს, როგორიცაა **Selenium Grid**. ზოგადად დავახასიათებთ მის შემადგენელ კომპონენტებს, აღვწერთ თუ როგორ ხდება Selenium Grid ჰაბის (ცენტრის) ოპერაციულ სისტემაზე დაყენება, Selenium RC სერვერების განსაზღვრა და ტესტების გაშვება Grid-ზე, ტესტების პარალელურ რეჟიმში მუშაობა და ა.შ.

Selenium Grid არის ავტომატური ტესტების გასაშვები ინსტრუმენტი, რომლის საშუალებით შესაძლებელია ტესტების გაშვება განსხვავებულ მანქანებზე, სხვადასხვა ბრაუზერებზე, სხვადასხვა ოპერაციულ სესტემებზე პარალელურად. ანუ იგი იძლევა დროის ერთიდაიმავე მომენტში რამდენიმე ტესტის გაშვების შესაძლებლობას სხვადასხვა მანქანაზე, სხვადასხვა ბრაუზერზე და ოპერაციულ სისტემაზე. აგრეთვე Grid-ს შეუძლია განაწილებული ტესტების გაშვება.

ზოგადად ამბობენ, რომ არსებობს ორი ძირითადი მიზეზი რატომაც შეიძლება დაგვჭირდეს Selenium Grid-ის გამოყენება:

- ავტომატური ტესტები რომ გაეშვას სხვადასხვა ბრაუზერზე, ბრაუზერის სხვადასხვა ვერსიებზე, და თვითონ ბრაუზერები გაეშვას განსხვავებულ ოპერაციულ სისტემებზე;
- რომ მოხდეს ავტომატური სცენარების მუშაობის დროის შემცირება.

Selenium Grid გამოიყენება ტესტირების შესრულების დროის დასაჩქარებლად. იგი შესრულების სიჩქარეს ზრდის რამდენიმე მანქანის გამოყენებით და მათზე პარალელური ტესტების რეალიზებით. მაგალითად, თუ გვაქვს შესასრულებელი 100 ტესტი, და ჩვენ დავაყენებთ Selenium Grid–ს 4 სხვადასხვა მანქანაზე, მოცემული

ტესების გასაშვებად დაგვჭირდება 4-ჯერ ნაკლები დრო, ვიდრე ტესტების ერთ მანქანაზე მიმდევრობით გაშვებისას. დიდი ტესტ-სცენარებისთვის და ხანგრძლივი ტესტებისთვის, როგორიცაა მაგალითად, ბევრი მონაცემების ვალიდაცია, პარალელური ტესტირება შეიძლება იყოს მნიშვნელოვანი, დროის დაზოგვის თვალსაზრისით [5].

Selenium Grid-ის სისტემაზე დასაყენებლად საჭიროა დამატებითი ინსტრუმენტის **Apache Ant** -ის გამოყენება. აღნიშნულის გამო ჯერ მოკლედ განვიხილოთ Ant ინსტრუმენტი, მოვახდინოთ მისი ინსტალაცია და შემდგომ დავუბრუნდეთ Grid-ს.

Ant არის პროგრამული ინსტრუმენტი, რომელიც გამოიყენება პროგრამული უზრუნველყოფის შექმნის პროცესების ავტომატიზაციისთვის. იგი დაწერილია Java ენაზე, იყენებს Java პლატფორმას და არის ერთ-ერთი საუკეთესო ინსტრუმენტი Java პროექტის შესაქმნელად. Ant-ი, პროექტის შექმნის პროცესის განსაზღვრისთვის იყენებს XML ენას. „გაჩუმების“ პრინციპით, XML ფაილის დასახელებაა build.xml.

Ant არის უფასო Apache პროექტი. Windows ოპერაციული სისტემისთვის Ant-ის გადმოწერა შესაძლებელია შემდეგი მისამართიდან:

http://ant.apache.org/bindownload.cgi

Ant ინსტრუმენტის დაყენების შემდეგ შესაძლებელია Selenium Grid-ის ინსტალაცია. ბოლო ვერსიის Grid–ის გადმოსაწერად გამოიყენეთ მისი ოფიციალური საიტის ლინკი:

http://selenium-grid.seleniumhq.org/download.html

გადმოწერილი zip გაფართოების ფაილი ამოაარქივეთ და შეინახეთ თქვენთვის სასურველ მისამართზე, ხისტ დისკოზე.

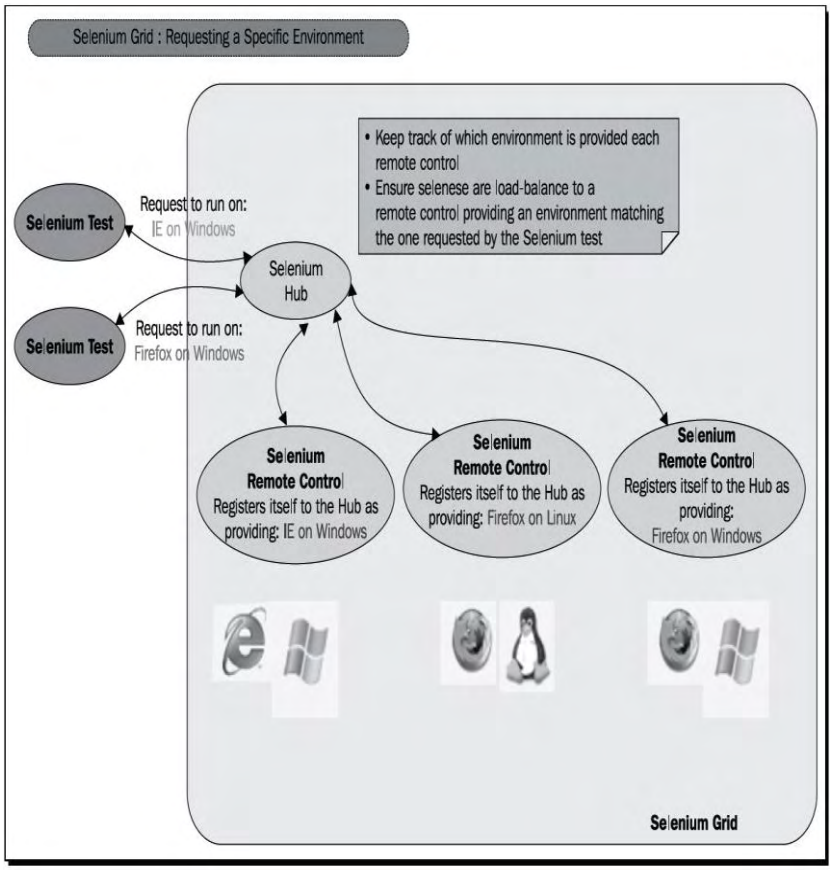
განვიხილოთ Selenium Grid ინსტალაციის ტესტი, რათა შევამოწმოთ რამდენად სწორად გვაქვს შესრულებული ინსტრუქციები და დავრწმუნდეთ, რომ Grid-ის გასაშვებად გვაქვს ყველა საჭირო ინსტრუმენტი. ამისთვის საჭიროა განვახორციელოთ ე.წ. **sanity-check** ტესტი [5].

გახსენით Windows ბრძანებათა ველი (cmd), კონსოლიდან მიმართეთ Grid დირექტორიას და ჩაწერეთ შემდეგი ბრძანება: **ant sanity-check**. მოცემული ბრძანების წარმატებით შესრულების შემთხვევაში მიიღებთ შემდეგს (ნახ.4.1):

ნახ.4.1. cmd: sanity-check ტესტი

Selenium Grid-ს გააჩნია ეგრეთ წოდებული „ცენტრალური საწყისი წერტილი", რომელსაც უკავშირდება ტესტები. ამ

ცენტრალური ნაწილიდან მიეწოდება ბრძანებები Selenium RC სერვერს. ესი ცენტრი წარმოადგენს Grid-ის ე.წ. **ჰაბს** (Hub), ნახ.4.2.

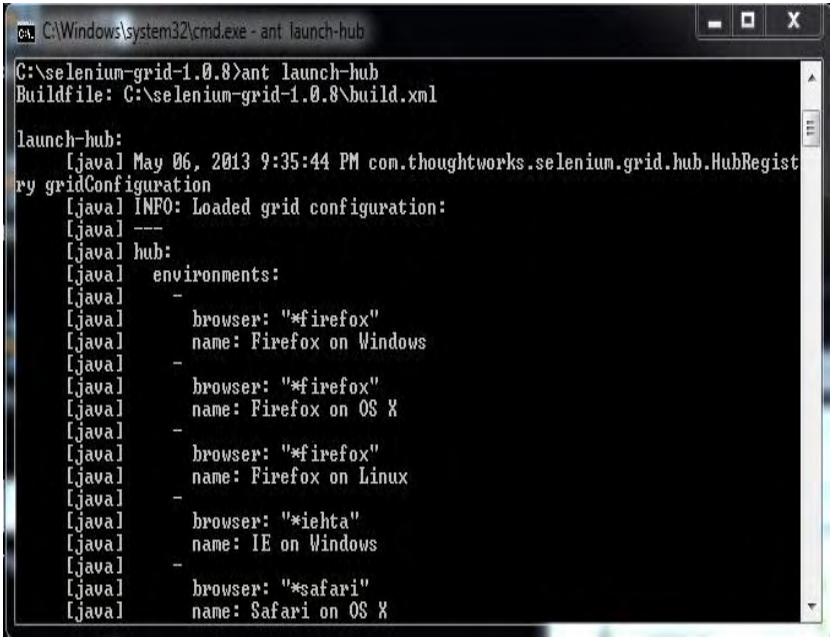


ნახ.4.2. Grid Hub დიაგრამა

ჰაბს აქვს Web ინტერფეისი, სადაც მოიცემა ჰაბთან დაკავშირებული Selenium RC სერვერები და მათი მდგომარეობები – არის თუ არა ისინი ამუშავებული [5].

Selenium Grid ჰაბის გასაშვებად საჭიროა ერთი მარტივი cmd ბრძანება:

1. გავხსნათ cmd ფანჯარა და კონსოლიდან მივმართოთ Grid დირექტორიას;
2. გაუშვით შემდეგი ბრძანება: **ant launch-hub;**
3. მოცემული ბრძანების შესრულებისას უნდა მივიღოთ შემდეგი სურათი (ნახ.4.3).

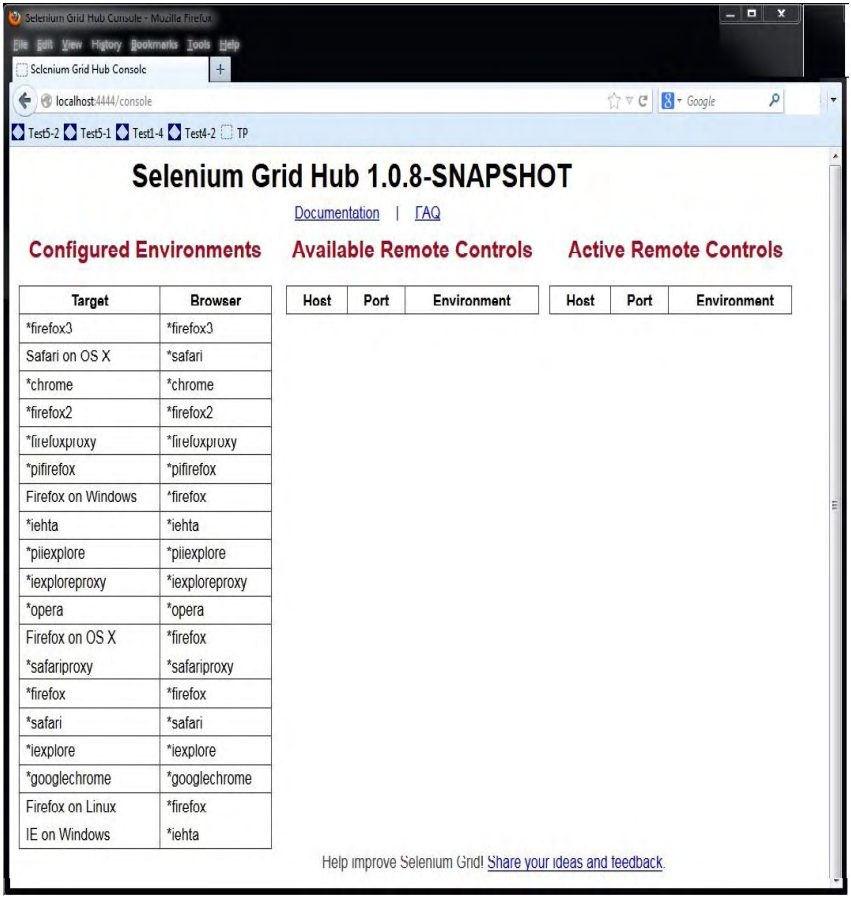


ნახ.4.3. ჰაბის გაშვება

მოცემულ cmd ფანჯრიდან ჩანს, რომ გაშვებულია Selenium Grid Hub–ი. როგორც აღვნიშნეთ, ჰაბს გააჩნია Web ინტერფეისი, ამიტომ მისი გაშვების დანახვა შესაძლებელია ბრაუზერიდანაც.

თუ ბრაუზერში გავხსნით შემდეგ URL–ს: http://localhost:4444/console, დავინახავთ შემდეგ Web ინტერფეისს (ნახ.4.4).

მოცემული ინსტრუქციით გავაკეთეთ Selenium Grid ჰაბის გაშვება. localhost მისამართით ვნახეთ ჰაბის Web ინტერფეისი, რომლის მარცხენა მხარეს მოიცემა „გაჩუმების“ პრინციპით კონფიგურირებული მოწყობილობები, ხოლო მარჯვენა ნაწილში RC კომპონენტების მდგომარეობები – თავისუფალი და აქტიური სერვერები.



ნახ.4.4. Firefox: ჰაბის Web ინტერფეისი

ახლა, როდესაც წარმატებით გავუშვით Selenium Grid Hub, განვიხილოთ როგორ ხდება Selenium RC სერვერების დამატება ჰაბზე, რის განსახორციელებლადაც საჭიროა კვლავ Ant ბრძანების გამოყენება [5].

Selenium Grid ინსტრუმენტიდან განვახორციელოთ Selenium RC სერვერზე კავშირი და დავარეგისტრიროთ იგი ჰაბზე.

სიმარტივისთვის ბრაუზერად განვიხილოთ Firefox და მივიჩნიოთ რომ ჰაბი და RC სერვერი არის ერთიდაიმავე მანქანაზე.

გამშვებ ბრძანებაში ყოველთვის დაგვჭირდება პორტის ნომრის ხელით გაწერა, რადგან Selenium–ი ვერ ხედავს თავისუფალ პორტებს.

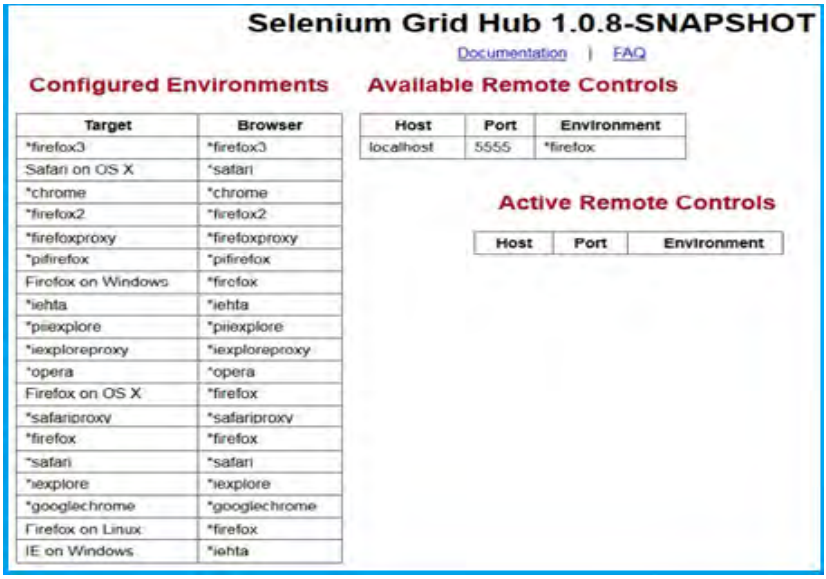
1. გახსენით cmd ფანჯარა და კონსოლიდან მიმართეთ Grid დირექტორიას;
2. გაუშვით შემდეგი ბრძანება: **ant –Dport=5555 launch-remote-control;**
3. მოცემული ბრძანების შესრულების შედეგად მიიღებთ შემდეგ სურათს (ნახ.4.5).

ზემოთ აღვნიშნეთ, რომ Selenium Grid-ს ქსელის საშუალებით შეუძლია სხვადასხვა მანქანაზე განსხვავებულ ოპერაციულ სისტემებზე სცენარების გაშვება. ამის პრაქტიკული რეალიზაციის მიზნით განვიხილოთ თუ როგორ ხდება სხვადასხვა ოპერაციულ სისტემაზე Selenium RC სერვერების გაშვება და მათი მართვა ჰაბის საშუალებით [6].

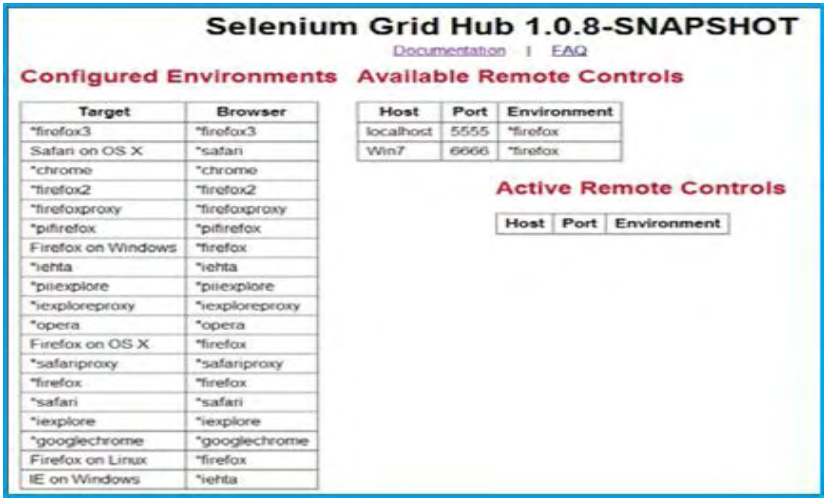
1. გავხსნათ ახალი cmd ფანჯარა და კონსოლიდან მივმართოთ Grid დირექტორიას;
2. გავუშვათ შემდეგი ბრძანება:

   ant –Dport=9999 -DhubURL=http://nameofmachine:port -Dhost=nameofcurrentmachine launch-remote-control;

3. მოცემული ბრძანების შესრულების შედეგად უნდა მივიღოთ შემდეგი სურათი (ნახ.4.6).

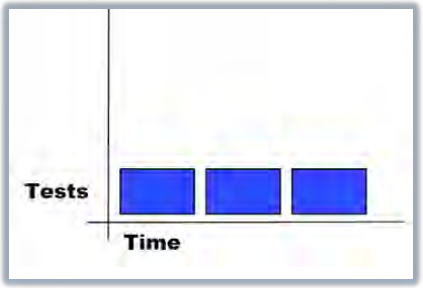


ნახ.4.5. Hub: Selenium RC სერვერის რეგისტრაცია ჰაბზე



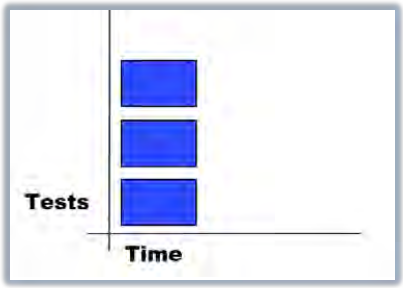
ნახ.4.6. Hub: Selenium RC სერვერის რეგისტრაცია
განსხვავებულ მანქანაზე

ტრადიციულად Selenium ავტომატური ტესტები სრულდება იმ თანმიმდევრობით, როგორ მიმდევრობასაც მატესტირებელი ფრეიმვორკი გადაწყვეტს (ნახ.4.7).



ნახ.4.7. მიმდევრობითი ტესტების დროის დიაგრამა

ახლა განვიხილოთ ტესტების შესრულება პარალელურ რეჟიმში, რაც ნიშნავს ტესტების შესრულების პროცესის დაჩქარებას n–ჯერ, სადაც n არის ბრაუზერების რაოდენობა (ნახ.4.8) [5].



ნახ.4.8. პარალელური ტესტების დროის დიაგრამა

მოცემული დიაგრამების პრაქტიკული რეალიზაციისთვის განვიხილოთ ორი, მიმდევრობითი და პარალელური ტესტირების მაგალითი.

მაგალითი 1: მიმდევრობითი ტესტები

განვიხილოთ Grid ინსტრუმენტში რეალიზებული სადემონ-სტრაციო მაგალითი, სადაც 4 Selenium ტესტი გაეშვება მიმდევრობით RC სერვერზე. ტესტების გაშვება განვიხილოთ Google Chrome ბრაუზერზე [5].

1. გავხსნათ cmd ფანჯარა, მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება: **ant launch-hub;**

2. გავხსნათ ახალი cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Denvironment="*googlechrome" launch-remote-control

3. გავხსნათ მე–3 cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Dbrowser="*googlechrome" run-demo-in-sequence

აღწერილი ინსტრუქციის სწორად შესრულების შემთხვევაში Selenium RC სერვერი გაეშვება 4–ჯერ Google Chrome ბრაუზერზე და თითოეული შეასრულებს განსხვავებულ ძებნის ტესტებს.

ყურადღება უნდა მიექცეს მთლიანი ტესტ-სუიტის გაშვების ხანგრძლივობას. სიმართლე რომ ვთქვათ, მოცემული მაგალითით არაფერი ახალი არ გაგვიკეთებია, მსგავს მიმდევრობით ტესტებს ჩვენ მიერ განხილული Selenium RC სერვერიც აკეთებს. ჩვენთვის საინტერესოა პარალელური ტესტირება, რომელსაც აკეთებს Grid ინსტრუმენტი.

მაგალითი 2: პარალელური ტესტები

განვიხილოთ Grid ინსტრუმენტში რეალიზებული იგივე მაგალითი (მაგალითი 1), სადაც 4 Selenium ტესტი გაეშვება პარალელურად ერთ მანქანაზე. ტესტების გაშვება განვიხილოთ კვლავ Google Chrome ბრაუზერზე.

1. გავხსნათ cmd ფანჯარა, მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება: **ant launch-hub**

2. გავხსნათ ახალი cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Denvironment="*googlechrome" launch-remote-control

3. გავხსნათ სამი ახალი cmd ფანჯარა, ყოველი მათგანიდან მივმართოთ Grid ინსტრუმენტის დირექტორიას და თითოეულში შესაბამისად ჩავწეროთ შემდეგი ბრძანება:

ant -Denvironment="*googlechrome" -Dport=5556 launch-remote-control

ant -Denvironment="*googlechrome" -Dport=5557 launch-remote-control

ant -Denvironment="*googlechrome" -Dport=5558 launch-remote-control

ჩამოთვლილი ბრძანებების გაშვების შემდეგ ჰაბის Web ინტერფეისზე უნდა გამოჩნდეს ოთხი თავისუფალი RC სერვერი http://localhost:4444/console:

Available Remote Controls

Host	Port	Environment
localhost	5555	*googlechrome
localhost	5556	*googlechrome
localhost	5557	*googlechrome
localhost	5558	*googlechrome

4. გავხსნათ კიდევ ერთი cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Dbrowser="*googlechrome" run-demo-in-parallel

შეცდომების არარსებობის შემთხვევაში შესრულებაზე გაეშვება 4 Selenium RC სერვერი ერთდროულად, ანუ შესრულდება პარალელური ტესტირება. დროის ერთიდაიმავე მომენტში გაეშვება სხვადასხვა ტესტი, რაც ცხადია პირველი (მიმდევრობითი) მაგალითის შესრულების სიჩქარეს მნიშვნელოვნად გააუმჯობესებს [5].

4.2. Selenium Grid 2

განვიხილოთ Selenium Grid-ის ახალი ვერსია Grid 2. იგი განსხვავდება ჩვენს მიერ განხილული Grid 1.0 ვერსიისგან. დღეისთვის Grid 1.0 ტექნოლოგია უკვე მოძველებულია, რომელსაც Selenium–ი აღარ ავითარებს. 2.0 ვერსიაში Selenium Grid–ი შეუერთდა Selenium RC სერვერს, რაც ნიშნავს იმას, რომ ერთი .jar ფაილის გადმოწერით თქვენ მიიღებთ Selenium RC სერვერს და Selenium Grid-საც, ორივეს ერთად [6].

Selenium Grid–ი იყენებს **ჰაბ–კვანძების** ცნებებს, სადაც **ჰაბს** წარმოადგენს ის მანქანა რომლიდანაც ხდება ტესტების გაშვება, ხოლო გაშვებული ტესტები სრულდება განსხვავებული მანქანების მიერ, რომელთაც ეწოდება **კვანძები** (ნახ.4.9) [5].

ნახ.4.9. Selenium Grid 2.0

ქვემოთ ჩამოთვლილია Grid 1.0-ის და Grid 2.0–ის ძირითადი განსხვავებები [6]:

1. Grid 1.0-ს აქვს თავის RC სერვერი, რომელიც განსხვავდება Selenium RC სერვერისგან. ისინი წარმოადგენს 2 სხვადასხვა პროგრამას;

ამჟამად Grid 2.0 არის Selenium RC სერვერის JAR ფაილთან შეერთებული;

2. Grid 1.0-ის ინსტალაციამდე საჭიროა Apache Ant–ის კონფიგურაცია;

Grid 2.0 არ საჭიროებს Ant ინსტალაციას;

3. Grid 1.0 თავსებადია მხოლოდ Selenium RC ბრძანებებთან / სკრიპტებთან;

Grid 2.0 თავსებადია როგორც Selenium RC-ს, ასევე WebDriver სკრიპტებთან;

4. Grid 1.0-ის ერთ RC სერვერზე შესაძლებელია მხოლოდ ერთი ბრაუზერის ავტომატიზაცია;

Grid 2.0–ის ერთ RC სერვერს შეუძლია 5 ბრაუზერზე მეტის ავტომატიზაცია.

- **რას წარმოადგენს ჰაბი** ?

- ჰაბი არის ცენტრალური ადგილი, საიდანაც ხდება ტესტების გაშვება;
- Grid-ს აქვს მხოლოდ ერთი ჰაბი;
- ჰაბის გაშვება ხდება მხოლოდ ერთ მანქანაზე, ვთქვათ კომპიუტერზე, რომელზეც აყენია Windows 7 და ბრაუზერი IE;
- ავტომატური ტესტების გაშვება შესაძლებელია განხორციელდეს როგორც ჰაბის შემცველ მანქანაზე, ასევე იმ კვანძებზე რომლებიც დარეგისტრირებულია ჰაბზე.

- **რას წარმოადგენს კვანძები** ?

- კვანძები არის Selenium ის ინსტრუმენტები, რომლებიც შესრულებაზე უშვებს ჰაბიდან მითითებულ ტესტებს;
- Grid-ს შეიძლება გააჩნდეს ერთი ან მეტი კვანძი;
- კვანძები შეიძლება აღიწეროს მრავალ მანქანაზე, განსხვავებულ პლატფორმასა და ბრაუზერზე;
- მანქანებზე დაყენებულ კვანძებს არ მოეთხოვება იყოს გაშვებული ჰაბის მსგავს პლატფორმაზე.

განვიხილოთ Grid 2.0-ის ინსტალაცია. დავუშვათ, რომ გვაქვს ორი მანქანა. პირველი მანქანა – სისტემა სადაც გავუშვებთ ჰაბს, მეორე მანქანა – განვიხილავთ Grid-ის კვანძად [5]. სიმარტივისთვის ჰაბის მანქანას ვუწოდოთ „მანქანა A", ხოლო კვანძის მანქანას ვუწოდოთ „მანქანა B". ვთქვათ A მანქანის IP მისამართია 192.168.8.13, ხოლო B მანქანის 192.168.8.13.

ბიჯი 1: გადმოწერეთ Selenium RC სრვერი Selenium-ის ოფიციალური საიტიდან;

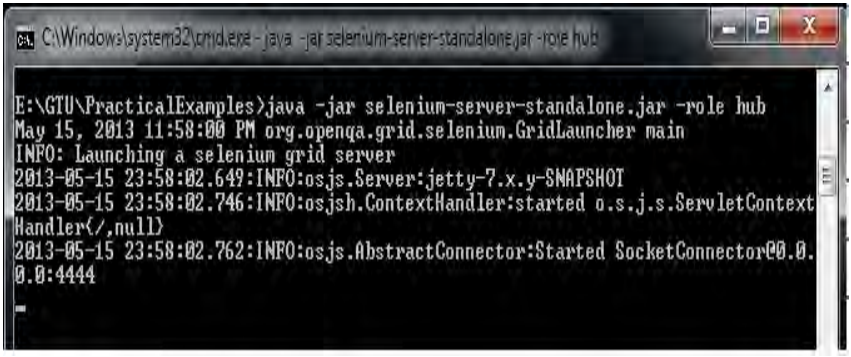
ბიჯი 2: შეინახეთ selenium-server.jar ფაილი სადმე ხისტ დისკოზე. ამით Selenium Grid_2.0-ის ინსტალაცია დასრულდა. მივყვეთ მომდევო პუნქტებს, გავუშვათ ჰაბი და მასზე დავარეგისტრიროთ კვანძი;

ბიჯი 3:

- მანქანა A-ზე გახსენით cmd ფანჯარა და კონსოლიდან მიმართეთ RC–ის დირექტორიას;
- Windows ბრძანებათა ველში გაუშვით შემდეგი ბრძანება:

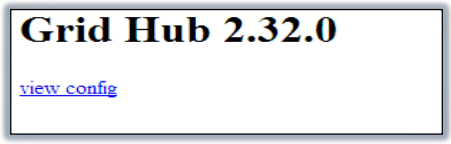
java -jar selenium-server.jar -role hub

- ჰაბი უნდა გაეშვას შესრულებაზე. თქვენი cmd ფანჯარა დაახლოებით უნდა გამოიყურებოდეს შემდეგნაირად (ნახ.4.10).



ნახ.4.10. cmd: გაშვებული ჰაბი

ბიჯი 4: ჰაბის გაშვება შევამოწმოთ ბრაუზერის გამოყენებით. Grid 2 „გაჩუმების“ პრინციპით Web ინტერფეისისთვის იყენებს A მანქანის 4444 პორტს. გახსენით ბრაუზერი და გადადით: http://localhost:4444/grid/console ლინკზე (ნახ.4.11).



ნახ.4.11. ჰაბის Web ინტერფეისი

B მანქანიდანაც არის შესაძლებელი ჰაბის Web ინტერფეისზე კავშირის შემოწმება შემდეგი URL–ით:

http://iporhostnameofmachine:4444/grid/console

სადაც "iporhostnameofmachine" არის A მანქანის (ჰაბის მანქანა) IP მისამართი, ან მისი ჰოსტის სახელი.

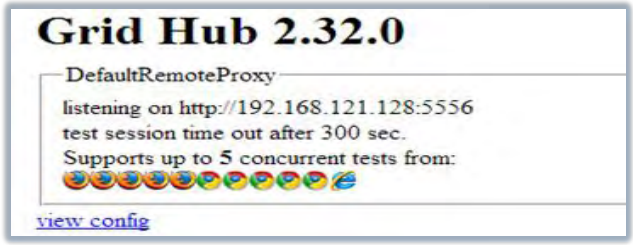
ბიჯი 5:

- მოცემულ ბიჯზე განვსაზღვროთ კვანძი. გადადით მანქანა B-ზე და გახსენით cmd ფანჯარა;
- კონსოლიდან მიმართეთ RC სერვერის დირექტორიას და ჩაწერეთ ქვემოთ აღწერილი ბრძანება:

java -jar selenium-server.jar -role webdriver -hub http://192.168.8.13:4444/grid/register -port 5556

- მოცემული ბრძანების შესრულების შედეგად Grid-ის ჰაბზე მოხდება კვანძის დარეგისტრირება (ნახ.4.12).

ბიჯი 6: Grid–ის Web ინტერფეისის განახლებით მიიღებთ დაახლოებით შემდეგ სურათს:

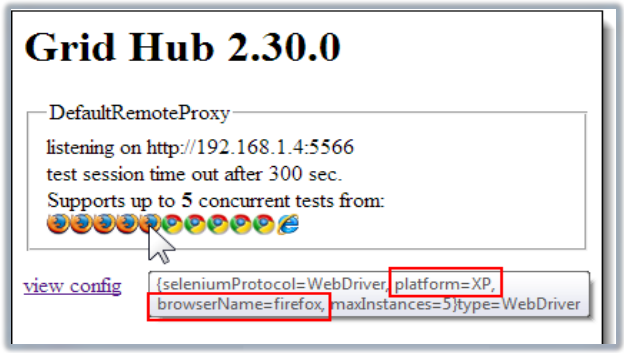


ნახ.4.12. ჰაბზე დარეგისტრირებული კვანძი

ამრიგად, მოცემული ინსტრუქციის საშუალებით დავაყენეთ Selenium Grid 2.0, გავუშვით ჰაბი და მასზე დავარეგისტრირეთ კვანძი.

Grid 2.0-ზე ტესტების გასაშვებად საჭიროა DesiredCapabilities და RemoteWebDriver ობიექტების გამოყენება. DesiredCapabilities ობიექტის საშუალებით განისაზღვრება ბრაუზერისა და ოპერაციული სისტემის ტიპი, ხოლო RemoteWebDriver ობიექტი გამოიყენება იმ კვანძის მისათითებლად, სადაც უნდა გაეშვას ჩვენი ავტომატური ტესტი.

DesiredCapabilities ობიექტის კომპონენტების განსასზღვრის მიზნით იყენებენ Grid 2.0–ის Web ინტერფეისს (ნახ.4.13).



ნახ.4.13. რეგისტრირებული კვანძის ატრიბუტები

მოცემულ შემთხვევაში პლატფორმა არის XP, ხოლო ბრაუზრის დასახელება firefox. აღნიშნული DesiredCapabilities ობიექტის განმსაზღვრელ კოდის ფრაგმენტს ექნება შემდეგი სახე:

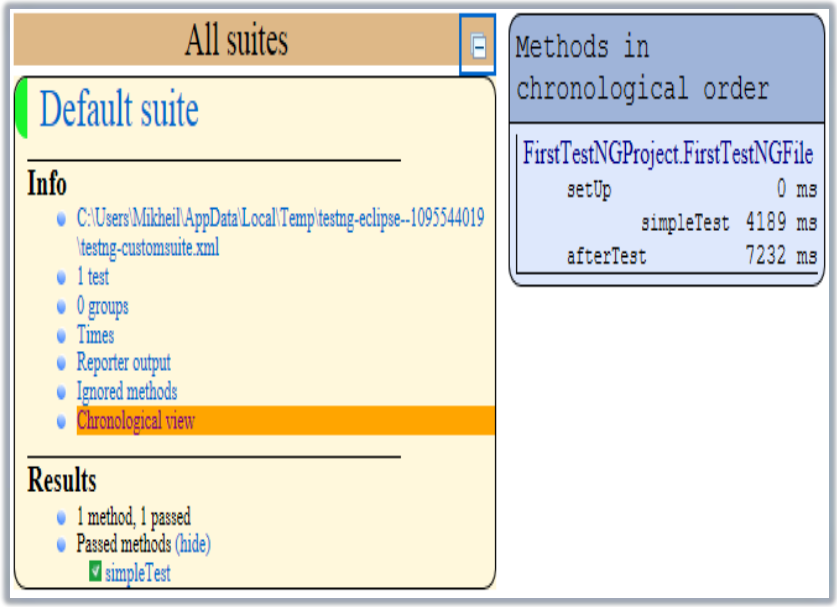
```
DesiredCapabilities capability = DesiredCapabilities.firefox();
capability.setBrowserName("firefox");
capability.setPlatform(Platform.XP);
```

ქვემოდ მოყვანილია მარტივი ტესტის მაგალითი Grid 2.0–ზე. იგი წარმოადგენს TestNG ფრეიმვორკის პროგრამულ კოდს, რომლის გაშვება შესაძლებელია A მანქანაზე, Eclipse გარემოში. მისი გაშვების შემდეგ ტესტის ავტომატიზაცია მოხდება B მანქანის Firefox ბრაუზერზე:

```
import org.openqa.selenium.*;
import org.openqa.selenium.remote.DesiredCapabilities;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Assert;
import org.testng.annotations.*;
public class FirstTestNGFile {
WebDriver driver;
String baseUrl, nodeURL;
@BeforeTest
public void setUp() throws MalformedURLException  {
        baseUrl = "http://newtours.demoaut.com/";
        nodeURL = "http://192.168.121.128:5556/wd/hub/";
DesiredCapabilities capability =
                DesiredCapabilities.firefox();
        capability.setBrowserName("firefox");
        capability.setPlatform(Platform.XP);
driver = new RemoteWebDriver(new URL(nodeURL), capability);
```

```
        }
        @AfterTest
        public void afterTest() {
                driver.quit();
        }
        @Test
        public void simpleTest() {
                driver.get(baseUrl);
        Assert.assertEquals("Welcome: Mercury Tours", driver.getTitle());
        }
}
```

ტესტის მუშაობის შედეგად უნდა მივიღოთ შემდეგი სახის რეპორტი (ნახ.4.14):



ნახ.4.14. Eclipse: TestNG რეპორტ ფაილი

4.3. Selenium სერვერი და მატესტირებელი ფრეიმვორკები: JUnit, TestNG, Screenshots

განვიხილავთ ტესტ-ქეისების შექმნას Java ტექნოლოგიაზე. შევქმნით Java ტესტებს JUnit და TestNG ფრეიმვორკების საშუალებით. ტესტირების პროცესში განვახორციელებთ ბრაუზერში გახსნილი გვერდების სქრინების გაკეთებას და ა.შ. ცხადია ეს მიანიშნებს Selenium ტექნოლოგიის პოპულარობაზე, იგი მხარდაჭერილია მრავალ პლატფორმაზე, Java-ზე, .NET-ზე და სხვა [5].

Java ტესტ-სკრიპტების დასაწერად საჭიროა დაპროგრამების რომელიმე ინსტრუმენტი, Java IDE. თავდაპირველად სიმარტივისთვის ტესტების შესაქმნელად გამოვიყენოთ Intellij IDEA, რადგან იგი შეიცავს ყველა ინსტრუმენტს, რაც საჭიროა წარმატებული ტესტების შესაქმნელად. გამოვიყენებთ Eclipse ინსტრუმენტსაც. ამრიგად, განსახილველი მაგალითების რეალიზაციისთვის დაგვჭირდება ორივე Intellij IDEA და Eclipse ინსტრუმენტები [7].

Java ტესტ-ქეისების დაწერისას დაგვჭირდება JUnit „ფრეიმვორკი", რომლის გადმოსაწერად გამოიყენეთ შემდეგი URL: http://junit.org/.

- JUnit არის მატესტირებელი Java ფრეიმვორკი, რომელიც უზრუნველყოფს განმეორებადი ტესტების შექმნას. იგი ძირითადად გამოიყენება unit (ერთეულოვანი) ტესტების დასაწერად. არის უფასო, open source ტექნოლოგია.

Selenium RC ტექნოლოგიის ერთ-ერთ მთავარ უპირატესობაა პროგრამული ლოგიკის რეალიზება ტესტ–სკრიპტებში. პროგრამული ლოგიკა მსგავსია დაპროგრამების ყველა ენაში. აპლიკაციის შესრულება ძირითადად იმართება პირობითი ოპერატორებისა და ციკლების საშუალებით. პირობითი ოპერატორის მაგალითია **if()** ოპერატორი, ხოლო ციკლისა **for().** Java სინტაქსით Selenium ტესტ-

სკრიპტების რეალიზებით შესაძლებელია ნებისმიერი რთული შემთხვევის ავტომატიზაცია.

განვიხილოთ როგორ ხდება Selenium ტექნოლოგიაში პროგრამული ენის კომბინაციით ტესტირების პრობლემების გადაჭრა. თუ განხილული მარტივი HTML ტესტებიდან გადავალთ რთულ ტესტებზე, სადაც ხდება Web გვერდებზე არსებული დინამიკური ფუნქციების გამოკვლევა და დინამიკურად მიღებული მონაცემების დადასტურება, მაშინ ამ დინამიკური პროცესების ავტომატიზაციისათვის საჭირო იქნება პროგრამული ლოგიკა, რომ დავადასტუროთ მოსალოდნელი შედეგი.

ზოგადად, Selenium IDE-ს არ გააჩნია ციკლების და პირობითი ოპერატორების მხარდაჭერა. შესაძლებელია რაღაც პირობის განსაზღვრა JavaScript კოდის ჩადგმით Selenese ბრძანებებში.

მიუხედავად ამისა იტერაციები (ციკლები) შეუძლებელია და პირობითი ოპერატორების რეალიზებაც დაპროგრამების ენაზე გაცილებით მარტივად და გასაგებად აღიწერება. აგრეთვე, შეცდომების აღმოსაჩენად შეიძლება დაგვჭირდეს გამონაკლისი შემთხვევების მართვა.

ჩამოთვლილი მიზეზების გამო განვიხილოთ პრაქტიკული მაგალითი, სადაც გამოჩნდება პროგრამული ტექნიკის ფართო გამოყენების საჭიროება. ავტომატურ ტესტებში პროგრამული ენის გამოყენება ამდიდრებს ვერიფიკაციის შესაძლებლობებს.

იტერაცია ტესტ-სკრიპტში არის ერთ-ერთი ყველაზე გამოყენებადი ინსტრუმენტი. მაგალითად, ერთიდაიმავე ტესტში შეიძლება დაგვჭირდეს ძებნის ტესტის გაშვება მრავალჯერ.

HTML ფორმატის ტესტ-სკრიპტიდან ცხადია, რომ მსგავსი მოქმედებების შესასრულებლად საჭიროა ერთიდაიმავე კოდის გამეორება ყოველ იტერაციაზე. როგორც ვიცით მსგავსი კოდის მრავალჯერადი კოპირება არ არის კარგი პროგრამული პრაქტიკა.

იგი მოითხოვს უფრო მეტ მუშაობას რეალიზაციისთვის. პროგრამული ენის გამოყენებით შესაძლებელია არსებული პრობლემის გადაჭრა, ძებნის „გაციკლება", უფრო დახვეწილი და მოქნილი გადაწყვეტილების პოვნა. ქვემოთ მოცემულია ტესტი C# დაპროგრამების ენაზე (ნახ.4.15).

```
// Collection of String values.
String[] arr = {"ide", "rc", "grid"};
// Execute loop for each String in array 'arr'.
foreach (String s in arr) {
   sel.open("/");
   sel.type("q", "selenium " +s);
   sel.click("btnG");
   sel.waitForPageToLoad("30000");
   assertTrue("Expected text: " +s+ " is missing on page."
   , sel.isTextPresent("Results * for selenium " + s));
 }
```

ნახ.4.15. C# ტესტ-სკრიპტი

მოცემული ტესტი Java დაპროგრამების ენაზე (ნახ.4.16):

```
// Array of String values.
String[] str = {"ide", "rc", "grid"};
// Execute loop for each String in array ' str'.
for (String s : str) {
   sel.open("/");
   sel.type("q", "selenium " +s);
   sel.click("btnG");
   sel.waitForPageToLoad("30000");
   assertTrue("Expected text: " +s+ " is missing on page."
   , sel.isTextPresent("Results * for selenium " + s));
 }
```

ნახ.4.16. Java ტესტ-სკრიპტი

მოცემული ერთიდაიგივე ტესტ-ქეისის Java და C# პროგრამული ფრაგმენტებიდან ჩანს, რომ დაპროგრამების ენით დაწერილი ტესტ-სკრიპტი პროგრამული თვალაზრისით გაცილებით ელეგნტურია, ვიდრე HTML ფორმატით შექმნილი სკრიპტი.

HTML ტესტში ერთიდაიმავე ფრაგმენტის გადაწერა ხდება 3–ჯერ, ხოლო დაპროგრამების ენით, მოცემული ბიჯი აღვწერეთ ერთხელ და დავატრიალეთ ციკლით.

ტესტ-სკრიპტში პირობითი ოპერატორის გამოყენების საჭიროების საილუსტრაციოდ განვიხილოთ კიდევ ერთი მაგალითი. ტესტირების პროცესის დროს, ძირითადად მოულოდნელი პრობლემა წარმოიშობა მაშინ, როდესაც ტესტ-ქეისში აღწერილი ელემენტი არ მოიძებნება მიმდინარე Web გვერდზე. მაგალითად, როდესაც გაეშვება შემდეგი სტრიქონი [6]:

```
selenium.type("q", "selenium " +s);
```

თუ ელემენტი 'q' არ არის web გვერდზე, მაშინ მივიღებთ შემდეგი ტიპის შეცდომას:

```
com.thoughtworks.selenium.SeleniumException: ERROR:
Element q not found
```

აღნიშნული შემთხვევა გამოიწვევს ტესტის ნაადრევად დასრულებას. მსგავსი რამ სცენარის შესრულებისას არ არის სასურველი, რადგან პრაქტიკაში ტესტ-სცენარს აქვს მრავალი სხვა ტესტ-ქეისების ქვემიმდევრობა შესასრულებელი.

მოცემულ პრობლემას გადავწყვეტთ პირობითი ოპერატორის გამოყენებით, რომლის საშუალებითაც შევამოწმებთ არსებობს თუ არა ელემენტი გვერდზე და მივიღებთ ალტერნატიულ გადაწყვეტილებას ელემენტის არარსებობის შემთხვევაში.

აღნიშნულის საილუსტრაციოდ გამოვიყენოთ Java პროგრამული ფრაგმენტი:

```
// If element is available on page then perform type operation.
        if(selenium.isElementPresent("q")) {
           selenium.type("q", "Selenium rc");
        } else {
  System.out.printf("Element:"+q+"is not available on page.")
        }
```

აღწერილი ფრაგმენტის უპირატესობაზე მეტყველებს ის, რომ მისი შესრულების შემდეგ შესაძლებელია გაგრძელდეს ტესტი, მიუხედავად იმისა, არსებობს თუ არა რაღაც UI (User Interface) ელემენტი მიმდინარე გვერდზე.

მაშასადამე, ტესტ-ქეისების დაპროგრამების Java ენაზე რეალიზებით Selenium მატესტირებელი ტექნოლოგია გახდა უფრო მდიდარი შესაძლებლობების მქონე, ადვილად გასაგები და დინამიკური პროცესების მართვადი ინსტრუმენტი.

ჩვენს მიერ გაშვებული JUnit ტესტები არ აგენერირებს სათანადო საშედეგო ფაილს. საშედეგო ფაილი წარმოადგენს შესრულებული სცენარის რეპორტს, მოცემულმა აპლიკაციამ გაიარა თუ არა ჩვენს მიერ განსაზღვრული სცენარი. ავტომატური ტესტირებისას რეპორტ ფაილის გენერირება მნიშვნელოვანია, რადგან მისი საშუალებით შესაძლებელია მსჯელობა, მუშაობს თუ არა მოცემული აპლიკაცია მოცემული სცენარის მიმართ.

წარმოდგენილ მასალაში განვიხილეთ HTML ფორმატის ტესტ-სცენარის საშედეგო ფაილი, სადაც „ჩავარდნილი“ ტესტები მოიცემოდა წითელ ფერში, ხოლო წარმატებით შესრულებული ტესტები მწვანეში. აღნიშნულ საკითხს გაცილებით კარგად ართმევს თავს Java-ს მატესტირებელი ფრეიმვორკი სახელად **TestNG**.

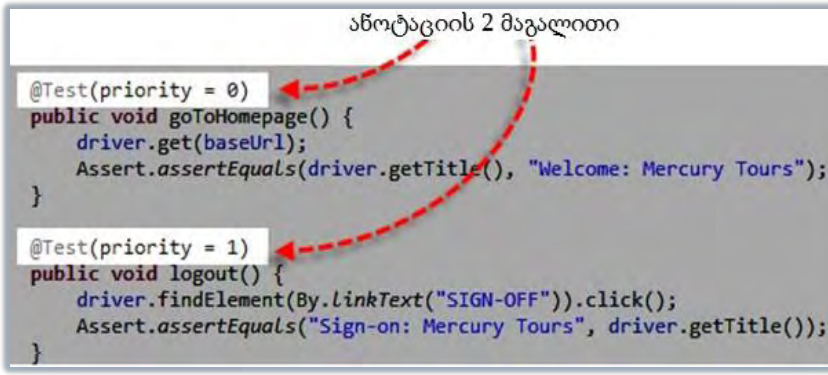
- TestNG არის JUnit–ზე ბაზირებული მატესტირებელი ფრეიმვორკი, რომელიც ახდენს JUnit ფრეიმვორკის ნაკლოვანებების აღმოფხვრას. მის დასახელებაში არსებული NG ნიშნავს "Next

Generation" (შემდგომი გენერაცია). Selenium მომხმარებლების უმრავლესობა უპირატესობას ანიჭებს TestNG-ს, ვდრე JUnit-ს. TestNG-ის გააჩნია მრავალი პოპულარული მახასიათებელი. ჩვენ აქ მიმოვიხილავთ ყველაზე მნიშვნელოვანს მათ შორის, რომლებიც კავშირშია Selenium ტექნოლოგიასთან [5,6].

TestNG-ის JUnit-თან შედარებით გააჩნია 3 ძირითადი უპირატესობა:

1. ანოტაციები არის უფრო ადვილად გასაგები;
2. ტესტ-ქეისები შესაძლებელია დაჯგუფდეს უფრო ადვილად;
3. შესაძლებელია პარალელური ტესტირება.

ანოტაცია წარმოადგენს სტრიქონს პროგრამულ კოდში, რომელიც საზღვრავს მის ქვემოთ აღწერილი მეთოდის გაშვების პრინციპებს. ანოტაციას წინ უძღვის @ სიმბოლო. თვალსაჩინოებისთვის ანოტაციის მაგალითი წარმოდგენილია 4.17 ნახაზზე.

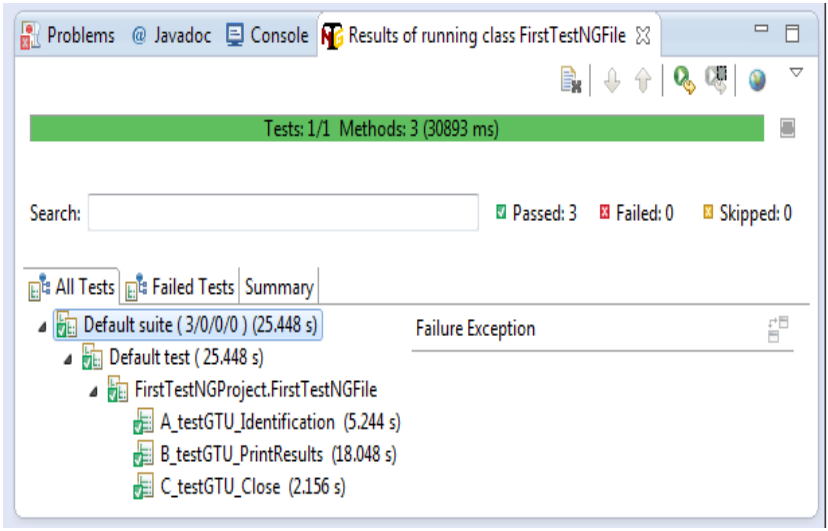


ნახ.4.17. TestNG ანოტაცია

მოცემულ მაგალითში მეთოდი goToHomepage() გაეშვება პირველი, ვიდრე logout(), რადგან მას აქვს უფრო დაბალი პრიორიტეტის მნიშვნელობა. მოცემული სურათიდან ჩანს რომ ანოტაციები TestNG–ში აღიწერება უფრო მარტივად და გასაგებია ვიდრე JUnit–ში [5].

TestNG ფრეიმვორკს აქვს პარალელური ტესტების გაშვების შესაძლებლობა, მაშასადამე Selenium Grid პარალელური ტესტერებისთვის იგი უფრო პრივილეგირებული ტექნოლოგიაა, ვიდრე JUnit-ი. TestNG–ის უპირატესობად უნდა აღინიშნოს ისიც, რომ მას შეუძლია აღუწერავი გამონაკლისი შემთხვევების მართვა, რაც თავიდან გვაციელბს ტესტის ნაადრევად დასრულების შესაძლებლობას.

TestNG, გარდა ხისებური სტილით მოცემული და კონსოლში ჩაწერილი ტექსტური რეპორტისა (ნახ.4.18), აგენერირებს აგრეთვე HTML ფორმატის საშედეგო ფაილს (ნახ.4.19).



ნახ.4.18. Eclipse: TestNG გრაფიკული რეპორტი

პარაგრაფის დასაწყისში ჩვენ შევეხეთ **@Test** ანოტაციის საკითხს. ახლა განვიხილოთ იგი უფრო დაწვრილებით.

ნახ.4.19. Eclipse: TestNG HTML რეპორტი

შესაძლებელია მრავალი @Test ანოტაციის გამოყენება ერთ TestNG ფაილში. @Test–ით ანოტირებული მეთოდები „გაჩუმების“ პრინციპით სრულდება ანბანური თანმიმდევრობით.

მიუხედავად იმისა, რომ მოცემულ ნახაზზე მეთოდები c_test, a_test და b_test კოდში არ არის დალაგებული ანბანურად, ისინი გაეშვებიან ანბანური რიგის მიხედვით (ნახ.4.20) [5].

```
public class FirstTestNGFile {

    @Test
    public void c_test() {
        Assert.fail();
    }

    @Test
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}
```

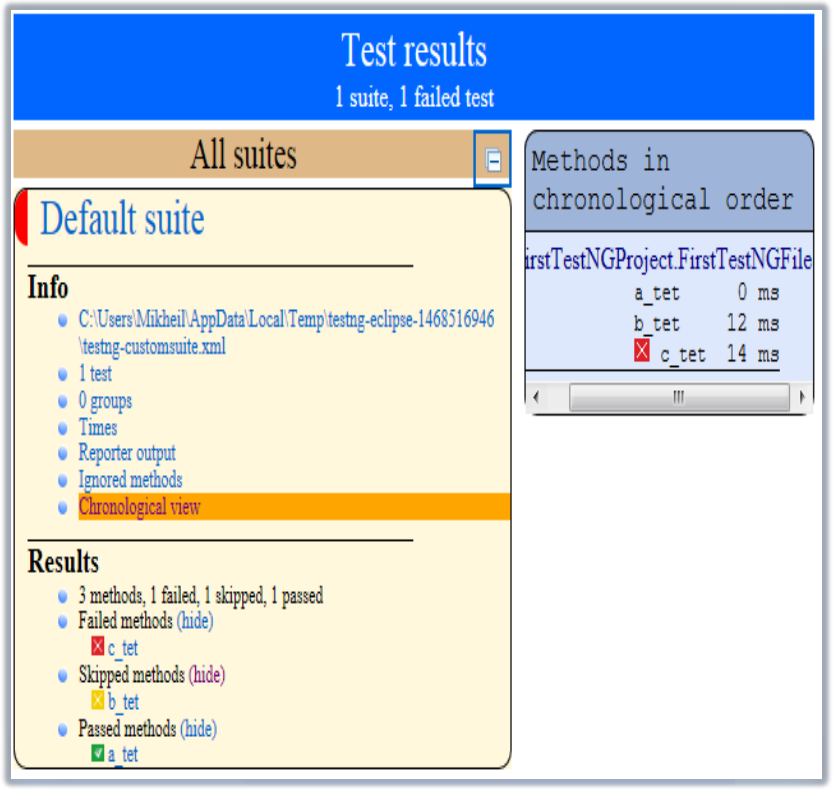
ნახ.4.20. TestNG ანოტაციები

აღნიშნულის სადემონსტრაციოდ გავუშვათ შესრულებაზე მოცემული კოდი და შევხედოთ საშედეგო ფაილს.

4.21 ნახაზიდან ჩანს, რომ ტესტები გაშვებულა ანბანის რიგის მიხედვით. იმისთვის რომ ანოტირებული მეთოდები გაეშვას სხვა მიმდევრობით და არა ანბანური რიგით, იყენებენ პარამეტრს "priority".

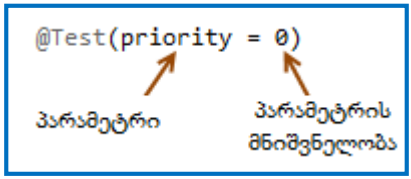
პარამეტრები საკვანძო სიტყვებია, რომლებიც ცვლის ანოტაციების ფუნქციებს [5].

- პარამეტრი მოითხოვს რაღაც მნიშვნელობის მინიჭებას, რაც ხორციელდება ჩვეულებრივი "=" ნიშნით, რომლის შემდეგაც იწერება მნიშვნელობა.



ნახ.4.21. Eclipse: TestNG HTML რეპორტი

- პარამეტრი იწერება მრგვალ ფრჩხილებში, ანოტაციის დასახელების შემდეგ, მაგალითად:



@Test ანოტაციებს TestNG ასრულებს ყველაზე დაბალი პრიორიტეტიდან მაღლისკენ. ამასთანავე არ არის საჭირო პრიორიტეტების მნიშვნელობების თანამიმდევრობის დაცვა, მაგალითად განვიხილოთ შემდეგი კოდის ფრაგმენტი (ნახ.4.22).

```
public class FirstTestNGFile {

    @Test(priority = 3)
    public void c_test() {
        Assert.fail();
    }

    @Test(priority = 0)
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test(priority = 7)
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}
```

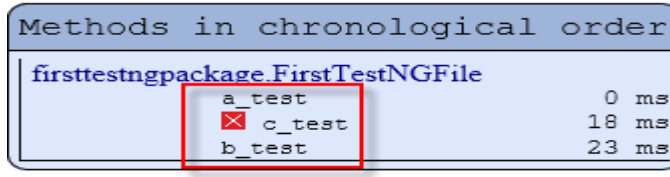
სიდიდით მეორე პრიორიტეტ–მნიშვნელობა აწუ იგი შესრულდება მეორე

ყველაზე დაბალი პრიორიტეტ–მნიშვნელობა აწუ იგი შესრულდება პირველი

ყველაზე მაღალი პრიორიტეტ–მნიშვნელობა აწუ იგი შესრულდება ბოლოს

ნახ.4.22. TestNG: პრიორიტეტებ-განსაზღვრული ტესტ-ქეისები

თუ მოცემულ კოდს გავუშვებთ შესრულებაზე, HTML რეპორტში, დავინახავთ მეთოდების შესრულების თანამიმდევრობას პრიორიტეტების მიხედვით (ნახ.4.23).



ნახ.4.23. ტესტ-ქეისების პრიორიტეტებით განსაზღვრული სია

განვიხილოთ TestNG მატესტირებელი ფრეიმვორკის ძირითადი ანოტაციები, რომლებიც შეიძლება გამოვიყენოთ Selenium ტესტების განსზღვრისას.

@BeforeTest - აღწერილი ანოტაციის ქვემოთ განსაზღვრული მეთოდი შესრულდება პირველი ტესტ–ქეისების გაშვებისას.

@AfterTest - აღწერილი ანოტაციის ქვემოთ განსაზღვრული მეთოდი შესრულდება ყველა ტესტ–ქეისის გაშვების შემდეგ.

მაგალითად განვიხილოთ შემდეგი პროგრამული კოდის ფრაგმენტი (ნახ.4.24).

```
public class FirstTestNGFile {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver = new FirefoxDriver();
    @BeforeTest
    public void launchBrowser() {
        driver.get(baseUrl);
    }
    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }
    @AfterTest
    public void terminateBrowser() {
        driver.quit();
    } }
```

ნახ.4.24. TestNG: ანოტირებული ტესტ-მეთოდები

მოცემულ კოდში აღწერილი მეთოდები შესრულდება შემდეგი თანამიმდევრობით:

პირველი – launchBrowser()

მეორე – verifyHomepageTitle()

მესამე – terminateBrowser()

შევნიშნოთ, რომ კოდში აღწერილი მეთოდის ბლოკის გადაადგილებით შესრულების თანამიმდევრობა არ შეიცვლება.

@BeforeMethod - მოცემული ანოტაციის ქვემოთ აღწერილი მეთოდი გაეშვება ყველაზე პირველი, ნებისმიერი მეთოდის გაშვებამდე.

@AfterMethod - მოცემული ანოტაციის ქვემოთ აღწერილი მეთოდი გაეშვება ყველაზე ბოლოს, ყველა მეთოდის გაშვების შემდეგ.

შევნიშნოთ რომ ერთიდაიმავე ტესტ-ფაილში აღწერილი @BeforeMethod და @BeforeTest ანოტაციებიდან პირველი გაეშვება @BeforeTest, რადგან იგი სრულდება ტესტ-ქეისის გაშვებამდე პირველი.

@BeforeSuite: მისი საშუალებით ანოტირებული მეთოდი გაეშვება პირველი მოცემულ ტესტ-სუიტში.

@AfterSuite: მისი საშუალებით ანოტირებული მეთოდი გაეშვება ყველაზე ბოლოს მოცემულ ტესტ-სუიტში.

@BeforeClass: მისი საშუალებით ანოტირებული მეთოდი გაეშვება პირველი მიმდინარე კლასში.

@AfterClass: მისი საშუალებით ანოტირებული მეთოდი გაეშვება ყველაზე ბოლოს მიმდინარე კლასში.

@Test: მისი საშუალებით ანოტირებული მეთოდი არის ტესტ-ქეისის შემადგენელი ნაწილი.

დასკვნის სახით შეგვიძლია ჩამოვაყალიბოთ TestNG ჯავა მატესტირებელი ფრეიმვორკის ძირითადი მახასიათებლები:

- TestNG არის მატესტირებელი ფრეიმვორკი, რომელსაც შეუძლია Selenium ტესტების შექმნა და ადვილად გასაგები რეპორტის გენერირება;
- TestNG-ის ძირითადი უპირატესობები JUnit–თან მიმართებაში: ადვილად გასაგები ანოტაციები, ტესტების დაჯგუფების უფრო ადვილი შესაძლებლობა და პარალელური ტესტირების მხარდაჭერა;

- Eclipse-ში კონსოლის ფანჯარაზე მოიცემა ტესტირების ტექსტური რეპორტი, ხოლო TestNG ფანჯარაზე გრაფიკული, ხისებური სტრუქტურით წარმოდგება ტესტირების დეტალური შედეგი: ტესტის ხანგრძლივობის დრო, მეთოდების შესრულების ქრონოლოგიური მიმდევრობა;
- TestNG აგენერირებს HTML ფორმატის რეპორტს;
- TestNG ანოტაციებს შეუძლია პარამეტრების გამოყენება ისევე, როგორც ჩვეულებრივ Java მეთოდებს.

ავტომატური ტესტებია ის სცენარები, რომლებიც გაიშვება სერვერებზე და სრულდება დამოუკიდებლად. პროგრამისტი ან ტესტერი მათ შესრულების პროცესში არ ერევა. თუ სცენარი შესრულდა ვინმეს ხელოვნური მანიპულირებით, მაშინ ამბობენ რომ მოცემული ტესტი არის ხელოვნური (manual) ტესტი [5].

ავტომატური ტესტირების დროს შეიძლება დადგეს ისეთი მექანიზმის გამოყენების საჭიროება, რომლითაც შევძლებთ გვერდის სქრინის გაკეთების, რათა სურათზე დავინახოთ ის შეცდომა, რომელმაც გამოიწვია ავტომატური ტესტის „ჩავარდნა“.

პირველი ბრძანება, რომელსაც განვიხილავთ Selenium ტექნოლოგიაზე, არის **captureScreenshot** ოპერატორი. იგი აკეთებს კომპიუტერის მიმდინარე სამუშაო მაგიდის სქრინს და არა ბრაუზერში გახსნილი გვერდის სქრინს. მოცემული ბრძანება გამოიყენება იმ შემთხვევაში, როდესაც გვაინტერესებს იმ მომენტში რა ხდება სამუშაო მაგიდაზე, როცა ტესტი „ვარდება“. შესაძლოა ამ დროს ბრაუზერი საერთოდ არ გამოჩნდეს სურათზე.

აღნიშნულ ბრძანებას აქვს **captureScreenshot(file)** ფორმატი, სადაც file წარმოადგენს გზას და ფაილის დასახელებას, რათა განსაზღვრულ გზაზე მოცემული დასახელებით მოხდეს სქრინის შენახვა. თუ თქვენ მიუთითებთ მხოლოდ ფაილის დასახელებას, იგი სქრინს შეინახავს Selenium RC სერვერის დირექტორიაში, ხოლო

თუ მიუთითებთ ფაილის შესანახ გზას, იგი შექმნის აღნიშნულ სურათს მითითებულ ფოლდერში [5].

დროის თვალსაზრისით ხანგრძლივი სცენარების ავტომატური ტესტირებისას სისტემაში თავდაპირველად იქმნება საწყისი მონაცემები და შემდგომ მათზე განისაზღვრება (აიგება) მომდევნო მოქმედებები. თუ რომელიმე ტესტ-ქეისი „ჩავარდება“, იგი დაფიქსირდება რეპორტ ფაილში. სცენარის დასრულების შემდეგ ხდება რეპორტ ფაილის ანალიზი. ჩავარდნილი სცენარის შემთხვევაში საჭიროა ტესტ-სკრიპტებში გაწერილი ლოგიკის ხელით შესრულება, რომ მივიღოთ ავტომატური ტესტირებისას წარმოქმნილი პრობლემა ბრაუზერში.

ცხადია, ეს მოითხოვს დამატებით რესურსებს, დროის ხარჯვას, სცენარის ანალიზს და ა.შ. მოცემულ შემთხვევაში ძალზე ხელსაყრელია შეცდომის ადგილების სქრინების ავტომატურ რეჟიმში გაკეთება.

პროგრამების ავტომატური ტესტირების დროს სქრინების გაკეთება არის გამოყენებადი, თუმცა ტესტირების პროცესის ვიდეოს ჩაწერა არის გაცილებით საინტერესო საკითხი.

4.4. მეოთხე თავის დასკვნა

ავტომატური ტესტირების ინსტრუმენტი Selenium Grid საშუალებას იძლევა ტესტების გაშვება განვახორციელოთ პარალელურად განსხვავებულ მანქანებზე, ბრაუზერებსა და ოპერაციულ სესტემებზე. მას შეუძლია აგრეთვე განაწილებული ტესტების გაშვება. ტესტირების შესრულების დრო საგრძნობლად მცირდება. მნიშვნელოვანია ტესტ-ქეისების შექმნა Java ტექნოლოგიაზე JUnit და TestNG ფრეიმვორკების საშუალებით. ტესტირების პროცესში შესაძლებელია ბრაუზერში გახსნილი გვერდების სქრინების გაკეთება. ეს მიანიშნებს Selenium ტექნოლოგიის პოპულაროზაზე, იგი მხარდაჭერილია მრავალ პლატფორმაზე.

V თავი
მართვის საინფორმაციო სისტემის დაპროექტება, დაპროგრამება და ტესტირება სერვის-ორიენტირებული არქიტექტურის ბაზაზე

საუნივერსიტეტო მართვის საინფორმაციო სისტემის შექმნა ან არსებული სისტემის მოდიფიკაცია ინტეგრაციის პრინციპების საფუძველზე მოითხოვს ამ საპრობლემო სფეროს ობიექტორიენტირებული, პროცესორიენტირებული და სერვისორიენტირებული მიდგომების კომპლექსურ გამოყენებას [1]. სისტემის შესაბამისი ინფრასტრუქტურის დასამუშავებლად კი აუცილებელია დღეისათვის არსებული ისეთი სტანდარტებისა და მეთოდოლოგიების გამოყენება, როგორიცაა BSI, ITIL, COBIT [2], რაც საბოლოო ჯამში უზრუნველყოფს უსაფრთხო განაწილებული ინფორმაციული სისტემის შექმნას, მის შემდგომ მასშტაბირებას და განვითარებას.

მეორეს მხრივ, ინტეგრაციის პროცესში სისტემის ცალკეული კომპონენტების დასამუშავებლად დროითი პარამეტრების და პროგრამული პროდუქტის ხარისხის გასაუმჯობესებლად აუცილებელი ხდება თანამედროვე CASE ტექნოლოგიების გამოყენება, როგორიცაა მაგალითად, Enterprise Architect (UML-ის ინსტრუმენტული საშუალება), Natural Object Role Modeling Architect (NORMA) და სხვ., რომლებიც მაიკროსოფთის Visual Studio .NET Framework პაკეტის სამუშაო გარემოს თავსებადია [3,31,65].

წინამდებარე თავში გადმოცემულია საუნივერსიტეტო მართვის საინფორმაციო სისტემის საპილოტო ვერსიის მაგალითზე მონაცემთა განაწილებული ბაზის და მომხმარებელთა ინტერფეისების დაპროექტების და პროგრამული რეალიზაციის პროცედურების დეტალური აღწერა ზემოაღნიშნული ახალი ტექნოლოგიებისა და სერვისორიენტირებული არქიტექტურის საფუძველზე.

პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის ეტაპების შესაბამისად (ანალიზი, დაპროექტება, დეველოპმენტი, ტესტირება, დანერგვა), საუნივერსიტეტო მართვის საინფორმაციო სისტემის შექმნა, UML-ტექნოლოგიით, ითვალისწინებს სისტემის ფუნქციონალური და არაფუნქციონალური მოთხოვნილებების განსაზღვრას, ობიექტორიენტირებული ანალიზის და დაპროექტების განხორციელებას, შესაბამისად სისტემასთან მომხმარებელთა მუშაობის ინტერაქტიული სცენარების, კლასთა-ასოსიაციების და მდგომარეობათა დიაგრამების აგებით სხვადასხვა შემთხვევით მოვლენათა შესაბამისად.

ესაა ცოდნა სამართავი ობიექტის შესახებ, მისი სტატიკური (მდგომარეობათა სიმრავლე) და დინამიკური (ქცევათა სიმრავლე) მოდელებით. თუ ობიექტორიენტირებული მოდელირების ტერმინებით ვისარგებლებთ, დასმული ამოცანის გადაწყვეტის „გასაღებს“ კლასების, ობიექტების, კლასთაშორისი კავშირების, ობიექტ-როლური და არსთა-დამოკიდებულების მოდელებისა და სხვა სახის დიაგრამების აგება წარმოადგენს. ხოლო შემდეგ, კლასთა-ასოციაციებისა და არსთა-დამოკიდებულების დიაგრამათა საფუძველზე განხორციელდება მიზნობრივი სისტემის პროგრამული კოდების რეალიზაციის ავტომატიზებული პროცესი ტესტირებით [66,67].

ამგვარად, ჩვენ განვიხილვთ კონკრეტული მართვის საინფორმაციო სისტემის („უნივერსიტეტი“) აგების ამოცანების სპექტრს და ამ პროცესში გამოვყოფთ პროგრამული სისტემის ტესტირების ფუნქციურ ამოცანას და მის რეალიზაციას [68].

5.1. სისტემის ობიექტ-როლური მოდელის დაპროექტება

განვიხილოთ ზოგადად „უნივერსიტეტის“ კლასის ობიექტი-სათვის მონაცემთა განაწილებული ბაზის დაპროექტების ამოცანა. უნივერსიტეტის საპრობლემო სფეროს არაფორმალიზებული აღწერის ობიექტებია (ტერმინთა ლექსიკონი): ფაკულტეტი, დეპარტამენტი, სტუდენტი, ლექტორი, საგანი (აკადემიური დისციპლინები, რომლებიც იკითხება შესაბამისი კათედრებსა და სპეციალობების მიხედვით), სასწავლო გეგმები, სილაბუსები (პროგრამები), ლექციები, პრაქტიკული და ლაბორატორიული სამუშაოები, აუდიტორიები, გამოცდები, ტესტირება და ა.შ.

ობიექტ-როლური მოდელირების თეორიის წესების თანახმად საჭიროა აიგოს უნივერსიტეტის სასწავლო პროცესის შესაბამისი ORM-დიაგრამა. ამისათვის ფაქტ-შეზღუდვების ერთო-ბლიობით (რომელსაც ადგენს სისტემური ანალიტიკოსი და საბოლოო მომხმარებელი), რომლებშიც ასახულია საპრობლემო სფეროს შესახებ ცოდნა (კლასებისა და ობიექტების ძირითადი ტერმინები და ქცევის წესები, დებულებით დადგენილი კანონზო-მიერებები და სხვ.), გადაიტანება Ms_Visual Studio.NET Framework სამუშაო გარემოში, ობიექტ-როლური მოდელის, NORMA-პაკეტის (Natural ORM Architect) ინტერფეისზე [65]. მაგალითად, ასეთი ფაქტები შეიძლება იყოს:

f1 : ლექტორს აქვს გვარი
f2 : ლექტორი მუშაობს დეპარტამენტში
f3 : ლექტორს აქვს თანამდებობა
f4 : ლექტორს აქვს ტელეფონი
f5 : ლექტორს აქვს ელ-ფოსტა
f6 : ლექტორს აქვს ხარისხი
f7 : ლექტორი კითხულობს საგანს

f8 : ლექტორი ასწავლის #-ჯგუფს

f8 : სტუდენტი არის #-ჯგუფში

. . .

f50 : ლექტორი მუშაობს #-დეპარტამენტში

f51 : დეპარტამენტი ეკუთვნის #-ფაკულტეტს

f52 : #-ჯგუფი ეკუთვნის #-დეპარტამენტს

f53 : ფაკულტეტს აქვს დასახელება

. . .

f100 : სტუდენტი არ შეიძლება იყოს ერთზე მეტ ჯგუფში

f101 : ლექტორი არ შეიძლება იყოს სრულ შტატზე ერთზე მეტ დეპარტამენტში

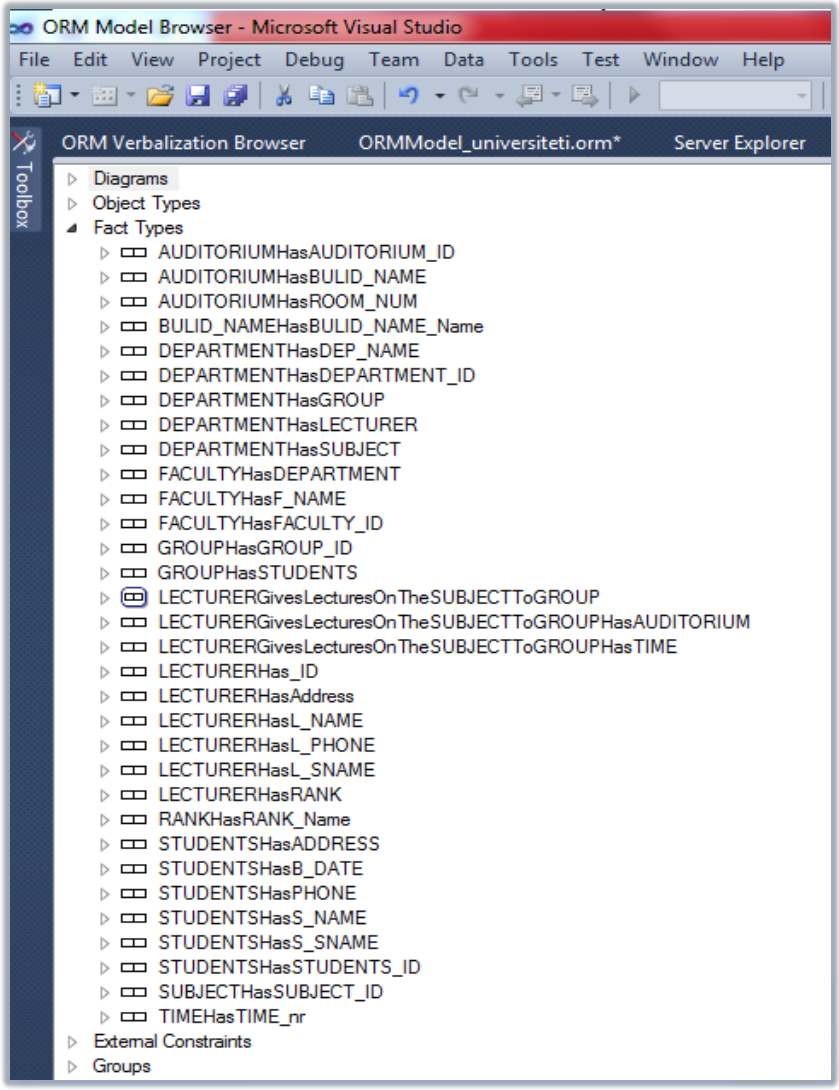
f102 : ლექტორი დროის ერთ მომენტში არ შეიძლება იყოს ორ სხვადასხვა აუდიტორიაში

და ა.შ.

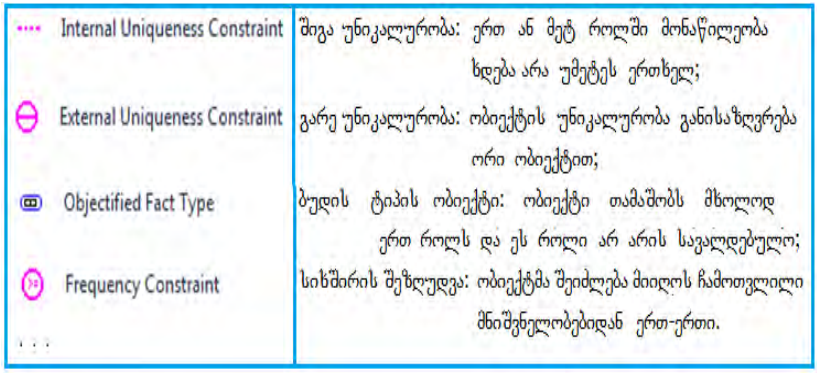
5.1 ნახაზზე მოცემულია Visual Studio.NET სამუშაო გარემოში ORM ინსტრუმენტის გამოყენებით მიღებული შედეგები „უნივერსიტეტის“ ფაქტების შეტანის საფუძველზე.

5.2 ნახაზზე წარმოდგენილია ობიექტ-როლური მოდელირების თეორიაში არსებული ზოგიერთი შეზღუდვის მაგალითი, რომლებიც გამოყენებულია ჩვენ მაგალითში.

შეზღუდვების აღწერა ხდება კატეგორიალური მიდგომის საფუძველზე, რომელიც აერთიანებს სალაპარაკო ენის ფორმალური გრამატიკის და ლოგიკურ-ალგებრულ წესებს. შედეგად მიიღება პრედიკატები, რომლებიც შეიძლება იყოს ერთ-, ორ- ან n-ადგილიანი [67].



ნახ.5.1. „უნივერსიტეტის“ ფაქტების აღწერის ფრაგმენტი



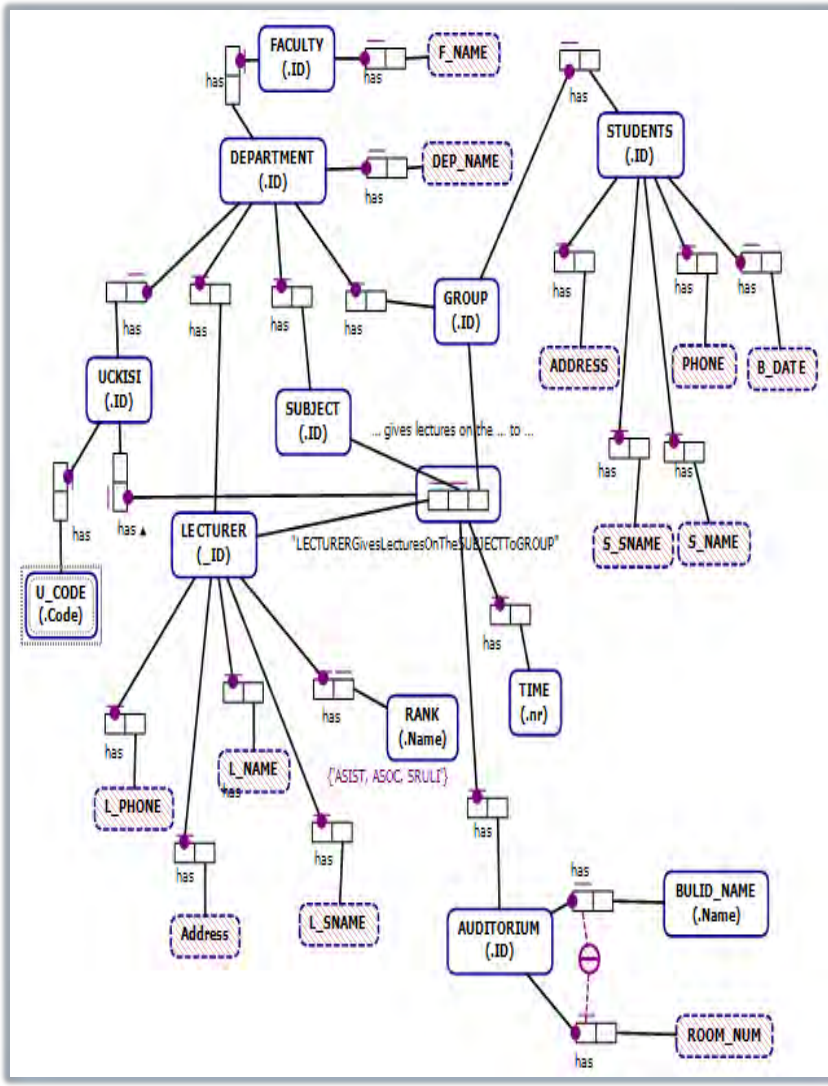
ნახ.5.2. შეზღუდვების აღწერის მაგალითები ORM-დიაგრამაზე

ზემოჩამოთვლილი ფაქტებიდან NORMA ინსტრუმენტი გვაძლევს შემდეგი სახის ORM-დიაგრამას (ნახ.5.3). აქ შესაძლებელია ახალი ფაქტის დამატება, არსებულის მოდიფიკაცია / წაშლა.

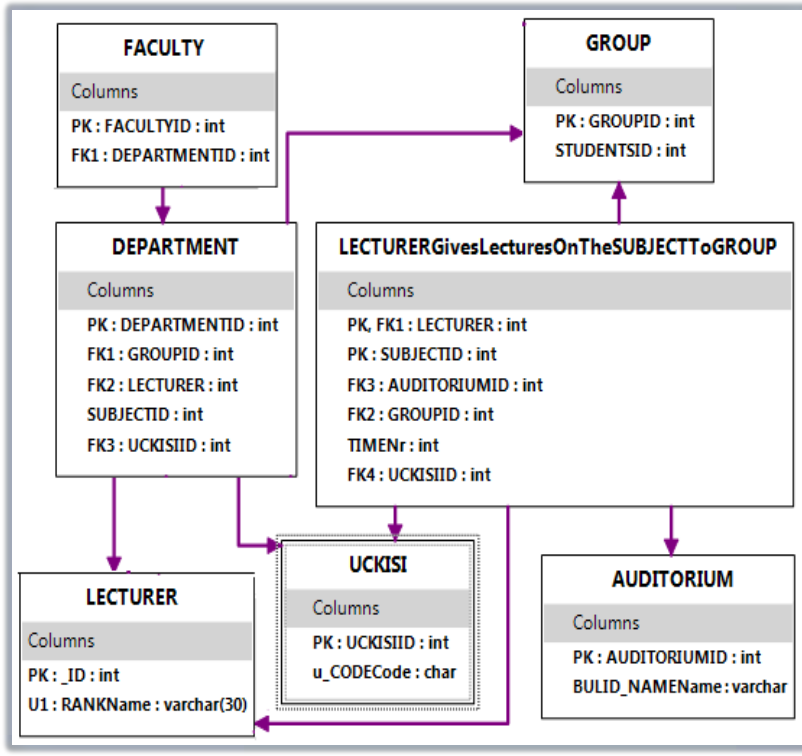
5.2. არსთა-დამოკიდებულების Eღ მოდელის დაპროექტება

მომდევნო ეტაპზე განხორციელდება ობიექტ-როლური მოდელის ავტომატიზებული გადაყვანა არსთა-დამოკიდებულების მოდელში. სისტემის დამპროექტებელი გააქტიურებს NORMA პაკეტის მენიუდან გენერაციის პროცედურას. ORM-დიაგრამიდან გენერირებული ERM-დიაგრამა მოცემულია 5.4 ნახაზზე.

შესაბამისად გამოკვეთილია შვიდი ობიექტი (არსი - Entity): ფაკულტეტი (Faculty), დეპარტამენტი (Department), ჯგუფი (Group), სტუდენტი (Students), ლექტორი (Lecturer), საგანი (Subject), აუდიტორია (Auditorium), საგამოცდო_უწყისი (Uckisi).



ნახ.5.3. „უნივერსიტეტის“ ORM დიაგრამის ფრაგმენტი
(საპრობლემო სფეროს კონცეპტუალური მოდელი-1)

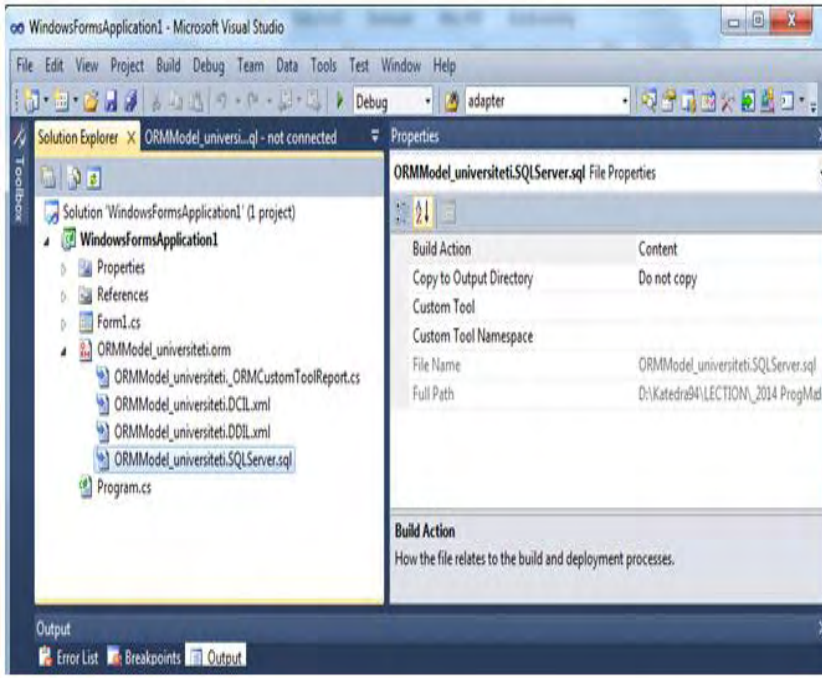


ნახ.5.4. ORM დიაგრამიდან გენერირებული ER-მოდელი (საპრობლემო სფეროს კონცეპტუალური მოდელი-2)

დიალოგში შესაძლებელია ER-მოდელის ობიექტების (Tables) განლაგების შეცვლა, რათა ვიზუალურად უფრო მოხერხებული მდებარეობა მიიღოს თითოეულმა, ობიექტთაშორისი კავშირების რაც შეიძლება ნაკლები გადაკვეთებით. ცხრილებში ჩანს ობიექტის სახელი და ატრიბუტთა დასახელებები, ტიპების მითითებით. აგრეთვე ასახულია პირველადი (PK) და მეორეული (FK) გასაღებური ატრიბუტები და ა.შ.

5.3. მონაცემთა ბაზის სერვერზე განთავსება

მონაცემთა ბაზის კონცეპტუალური ERM სქემის აგების შემდეგ საჭიროა მის საფუძველზე სისტემის მიერ დაიწეროს შუალედური ტექსტური ტიპის DLL–ფაილი, რომელიც მომავალში SQL Server მონაცემთა ბაზების მართვის სისტემამ უნდა გამოიყენოს. ამგვარად, ER-დიაგრამიდან, ცხრილებითა და ატრიბუტებით, ავტომატურად გენერირდება .DDL ფაილები, რომლებიც შემდგომ სტრუქტურულად მოთავსდება Ms SQL Server მონაცემთა განაწილებულ ბაზაში. 5.5 ნახაზზე ნაჩვენებია ამ ეტაპის პროცესის ინიცირების დიალოგური სქემა.



ნახ.5.5. DDL ფაილის გენერაცია ER-მოდელიდან

5.1 ლისტინგში მოცემულია ავტომატურად გენერირებული DDL-ფაილის ტექსტის ფრაგმენტი.

```
-- Listing_5.1 -----DDL file -------------------------
CREATE SCHEMA ORMModel1
GO
GO
CREATE TABLE ORMModel1.FACULTY
(  FACULTYID INTEGER IDENTITY (1, 1) NOT NULL,
   DEPARTMENTID INTEGER NOT NULL,
   CONSTRAINT FACULTY_PK PRIMARY KEY(FACULTYID)  )
GO
CREATE TABLE ORMModel1.DEPARTMENT
(  DEPARTMENTID INTEGER IDENTITY (1, 1) NOT NULL,
   GROUPID INTEGER NOT NULL,
   LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DEPARTMENTID)   )
GO
CREATE TABLE ORMModel1."GROUP"
(  GROUPID INTEGER IDENTITY (1, 1) NOT NULL,
   STUDENTSID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT GROUP_PK PRIMARY KEY(GROUPID)       )
GO
CREATE TABLE ORMModel1.LECTURER
(  "_ID" INTEGER IDENTITY (1, 1) NOT NULL,
   RANKName NATIONAL CHARACTER VARYING(30) NOT NULL,
   CONSTRAINT LECTURER_PK PRIMARY KEY("_ID"),
   CONSTRAINT LECTURER_UC UNIQUE(RANKName),
   CONSTRAINT LECTURER_RANKName_RoleValueConstraint1 CHECK
(RANKName IN (N'ASIST, ASOC, SRULI'))  )
GO
CREATE TABLE
ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
(  LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   AUDITORIUMID INTEGER NOT NULL,
   GROUPID INTEGER NOT NULL,
   TIMENr INTEGER NOT NULL,
   CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_PK
PRIMARY KEY(LECTURER, SUBJECTID)     )
GO
CREATE TABLE ORMModel1.AUDITORIUM
```

```
(  AUDITORIUMID INTEGER IDENTITY (1, 1) NOT NULL,
   BULID_NAMEName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
   CONSTRAINT AUDITORIUM_PK PRIMARY KEY(AUDITORIUMID)
)
GO
ALTER TABLE ORMModel1.FACULTY ADD CONSTRAINT FACULTY_FK
FOREIGN KEY (DEPARTMENTID) REFERENCES ORMModel1.DEPARTMENT
(DEPARTMENTID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT DEPARTMENT_FK1
FOREIGN KEY (GROUPID) REFERENCES ORMModel1."GROUP" (GROUPID)
ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT DEPARTMENT_FK2
FOREIGN KEY (LECTURER) REFERENCES ORMModel1.LECTURER ("_ID")
ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
ADD CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK1
FOREIGN KEY (LECTURER) REFERENCES ORMModel1.LECTURER ("_ID")
ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
ADD CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK2
FOREIGN KEY (GROUPID) REFERENCES ORMModel1."GROUP" (GROUPID)
ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
ADD CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK3
FOREIGN KEY (AUDITORIUMID) REFERENCES ORMModel1.AUDITORIUM
(AUDITORIUMID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
GO
```

შეიძლება ჩვთვალოთ, რომ ამ DDL ფაილის კოპირებით Ms SQL Server–ში შეიქმნება შესაბამისი ბაზა, ცხრილებით, ატრიბუტებით და კავშირებით. რა თქმა უნდა, შესაძლებელია აქაც დამპროექტებლის ჩარევა ბაზის სტრუქტურის ზოგიერთი კომპონენტის შესასწორებლად, საჭიროების შემთხვევაში

5.4. ბიზნესპროცესის სერვისის შექმნა

როგორც ცნობილია, ბიზნესპროცესი შეიძლება განთავსდეს ვებ-სერვისში, რომელიც უზრუნველყოფს ბიზნესპროცესის გადაწყვეტილების (შედეგის) მიწოდებას კლიენტებისთვის (ვებ-აპლიკაციებისთვის). ვებ-სერვისი იღებს მოთხოვნას კლიენტისგან, ასრულებს მის დამუშავებას და უბრუნებს პასუხს. ეს პროცედურები სრულდება Receive და Send ქმედებებით.

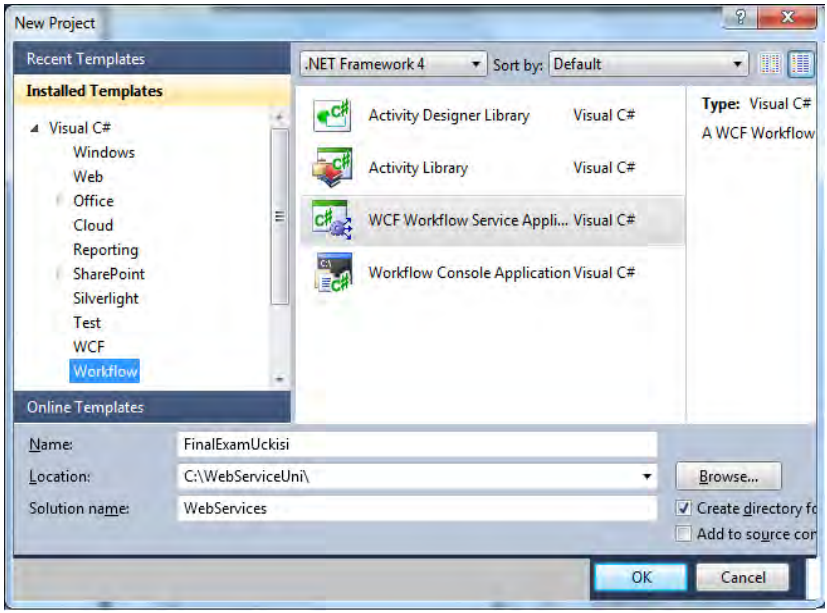
განვიხილოთ ჩვენი მაგალითის რეალიზაცია ჰიბრიდული ტექნოლოგიების, WF (Workflow Foundation) და WCF ბაზაზე (Windows Communication Foundation) [69]. ისევე, როგორც WPF (Windows Presentation Foundation) ტექნოლოგია, Visual Studio .NET Framework 3.0/3.5 გარემოში ქმნის მომხმარებელზე ორიენტირებულ მაღალი ხარისხის დიზაინის აპლიკაციებს C#.NET (ლოგიკის ნაწილი) და XAML (დიზაინის ნაწილი) ენების საფუძველზე [70,71].

ავამუშავოთ Visual Studio და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი FinalExamUckisi და solution-ის სახელი Web-Services (ნახ.5.6).

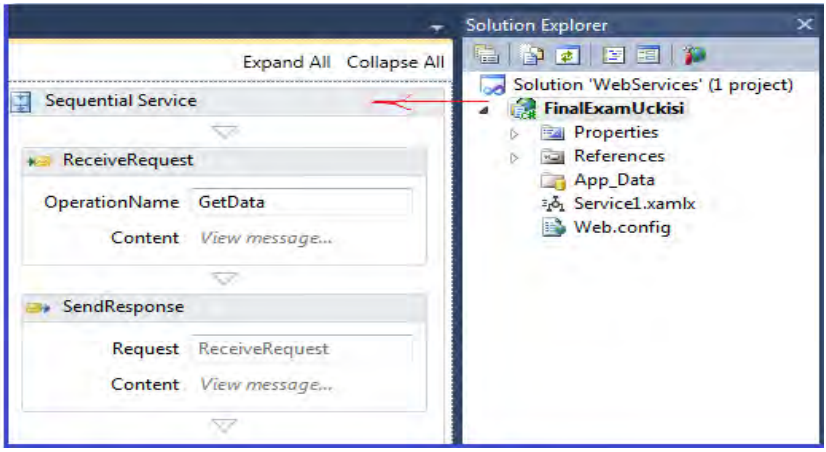
შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 5.7 ნახაზზეა ნაჩვენები.

თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების ბიზნესპროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი შექმნის საწყის ბიზნესპროცესს ფაილში, სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი Final-ExamUckisi.xamlx -ით.



ნახ.5.6. Visual Studio.NET –ში WCF-პროექტის შექმნა



ნახ.5.7. FinalExamUckisi პროექტის workflow Sequence ბლოკი

სერვისი, რომელიც მაგალითის სახით უნდა შევქმნათ, „საფინალო გამოცდისთვის“, ძებნის შესაბამის უწყისებს მითითებული აკადემიური ჯგუფის, ლექტორის და აკადემიური საგნის მიხედვით. ელექტრონული საგამოცდო უწყისები ეკუთვნის ფაკულტეტის დეკანატს, ხოლო მათი დამუშავება ხდება დეპარტამენტთა ლექტორების მიერ”.

5.5. სერვისის კონტრაქტის განსაზღვრა

WCF სისტემებში იყენებენ კონტრაქტების სამ დონეს: მონაცემთა კონტრაქტი, შეტყობინებათა კონტრაქტი და სერვისის კონტრაქტი [72].

მონაცემთა კონტრაქტის დანიშნულებაა შეთანხმება კლიენტსა და სერვისს შორის ერთმანეთთან გასაცვლელ მონაცემებზე (ითვალისწინებენ მონაცემთა სტრუქტურებს, პარამეტრებს, მოწესრიგებას და ა.შ.).

შეტყობინებათა კონტრაქტი უზრუნველყოფს SOAP (Simple Object Access Protocol) შეტყობინებების კონტროლს, რომელიც გამოიყენება ქსელში სხვადასხვა შეტყობინების გასაცვლელად XML ფორმატში. იგი უზრუნველყოფს აგრეთვე ინფორმაციის გაცვლის უსაფრთხოებას შეტყობინებების დონეზე.

სერვისის კონტრაქტი (ანუ კონტრაქტი მომსახურებისთვის) განსაზღვრავს ოპერაციათა სახეებს, რომლებსაც უზრუნველყოფს სერვისი. იგი კლიენტს აწვდის ასევე ინფორმაციას: შეტყობინებაში მონაცემთა ტიპების შესახებ, ოპერაციათა ადგილმდებარეობის შესახებ, ინფორმირების პროტოკოლისა და სერიალიზაციის ფორმატის შესახებ, შეტყობინებათა გაცვლის შაბლონების შესახებ (ცალმხრივი, ორმხრივი ან კითხვა/პასუხის ტიპებით).

ჩვენი პროექტისთვის სერვისის კონტრაქტის ასაგებად უნდა შევქმნათ საგამოცდო უწყისის ინფორმაციის კლასი C# ნაზე -

UckisiInfo.cs. ამისთვის Solution Explorer-ში მარჯვენა ღილაკით FinalExamUckisi პროექტზე ვირჩევთ Add->Cla სახელით UckisiInfo.cs, რომლის ტექსტი მოცემულია 5.2 ლისტინგში.

```
// ----- ლისტინგი_5.2 – Service Contract ---
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace FinalExamUckisi
{  // ---- სერვისის კონტრაქტის განსაზღვრა ---
IfinalExamUckisi,  რომელიც
   // --- შედგება ერთი მეთოდისგან  -  LookupUckisi () ------
    [ServiceContract]
    public interface IFinalExamUckisi
    {   [OperationContract]
        UckisiInfoList LookupUckisi(UckisiSearch request);
}
    //--  მოთხოვნის შეტყობინების განსაზღვრა , UckisiSearch ---
    [MessageContract(IsWrapped = false)]
    public class UckisiSearch
     {  private String _JGUPI;
        private String _Sagani;
        private String _Lectori;
        public UckisiSearch() {  }
        public UckisiSearch(String sagani, String lectori,
String jgupi)
        {
            _Sagani  = sagani;
            _Lectori = lectori;
            _JGUPI   = jgupi;       }
        #region Public Properties
        [MessageBodyMember]
```

```
        public String Sagani
        {   get { return _Sagani; }
            set { _Sagani = value; }    }
        [MessageBodyMember]
        public String Lectori
        {   get { return _Lectori; }
            set { _Lectori = value; }   }
        [MessageBodyMember]
        public String JGUPI
        {   get { return _JGUPI; }
            set { _JGUPI = value; }     }
        #endregion Public Properties
    }

    // --- UckisiInfo კლასის განსაზღვრა -----
        [MessageContract(IsWrapped = false)]
    public class UckisiInfo
    {   private Guid _ExamUckisiID;
        private String _JGUPI;
        private String _Sagani;
        private String _Lectori;
        private String _Status;
        public UckisiInfo() {   }
        public UckisiInfo(String sagani, String lectori,
String jgupi, String status)
        {   _Sagani = sagani;
            _Lectori = lectori;
            _JGUPI = jgupi;
            _Status = status;
            _ExamUckisiID = Guid.NewGuid();    }
        #region Public Properties
        [MessageBodyMember]
        public Guid ExamUckisiID
        {   get { return _ExamUckisiID; }
            set { _ExamUckisiID = value; }     }
```

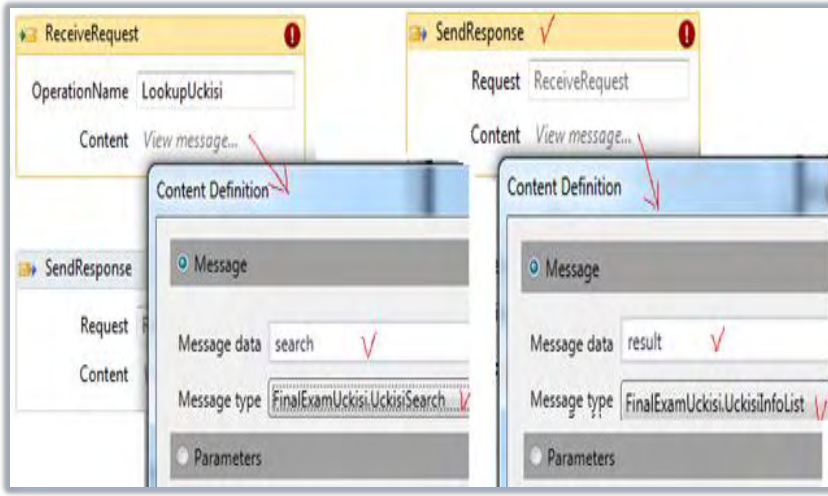
```
        [MessageBodyMember]
        public String Sagani
        {   get { return _Sagani; }
            set { _Sagani = value; }           }
        [MessageBodyMember]
        public String Lectori
        {   get { return _Lectori; }
            set { _Lectori = value; }          }
        [MessageBodyMember]
        public String JGUPI
        {   get { return _JGUPI; }
            set { _JGUPI = value; }            }
        [MessageBodyMember]
        public String status
        {   get { return _Status; }
            set { _Status = value; }           }
        #endregion Public Properties
    }
    // ------ საპასუხო შეტყობინების განსაზღვრა  ---
UckisiInfoList ---
       [MessageContract(IsWrapped = false)]
    public class UckisiInfoList
    {   private List<UckisiInfo> _UckisiList;
        public UckisiInfoList()
        {  _UckisiList = new List<UckisiInfo>();        }
        [MessageBodyMember]
        public List<UckisiInfo> UckisiList
        {    get { return _UckisiList; }          }
    }
}
```

სერვისის კონტრაქტი IFinalExamUckisi შეიცავს ერთადერთ მეთოდს LookupUckisi(). იგი მონაცემებს გადასცემს UckisiSearch კლასს, რომელსაც აქვს სხვადასხვა თვისებები, საჭირო საგამოცდო უწყისის მოსაძებნად, მაგალითად, ლექტორი, საგანი, ჯგუფი. ის

აბრუნებს უკან UckisiInfoList კლასს, რომელიც შეიცავს UckisiInfo კლასების კოლექციას. F6 ამოქმედებით აიგება გადაწყვეტა (solution). ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

5.6. Receive და SendReply კონფიგურირება

შემდეგ ეტაპზე FinalExamUckisi.xamlx-ში ReceiveRequest ქმედებისთვის OperationName თვისებაში ვათავსებთ LookupUckisi სახელს. WorkflowService-დიზაინერში შევქმნით ორ ახალ ცვლადს (Variables): search ცვლადი, შემომავალი შეტყობინების შესანახად და result ცვლადი, გამოსატანი შედეგისთვის (ნახ.5.8). Message ბუტონი უნდა იყოს ჩართული და ტიპებიც არჩეული.



ნახ. 5.8. შემავალი მოთხოვნის შეტყობინების და გამომავალი შედეგის ცვლადების განსაზღვრა

5.7. PerformLookup ქმედების აგება (ძებნის შესრულება)

ძებნის განსახორციელებლად (მონაცემთა ბაზებთან მიმართვის მიზნითაც) ვქმნით ახალ კლასს PerformLookup.cs, რომელიც Solution Explorer-ში პროექტისთვის FinalExamUckisi აირჩევა Add->NewItem, Workflow-კატეგორიაში, Code Activity-ით. ამ ახალი კლასის ტექსტი მოცემულია 5.3 ლისტინგში.

```
// ---- ლისტინგი_5.3 --- PerformLookup ----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
namespace FinalExamUckisi
{
  public sealed class PerformLookup : CodeActivity
   {
    public InArgument<UckisiSearch> Search { get; set; }
   public OutArgument<UckisiInfoList> UckisiList {get;set;}
   protected override void Execute(CodeActivityContext
                                                    context)
    { string lectori = Search.Get(context).Lectori;
      string sagani = Search.Get(context).Sagani;
      string jgupi = Search.Get(context).JGUPI;

      UckisiInfoList l = new UckisiInfoList();

     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "Available"));
```

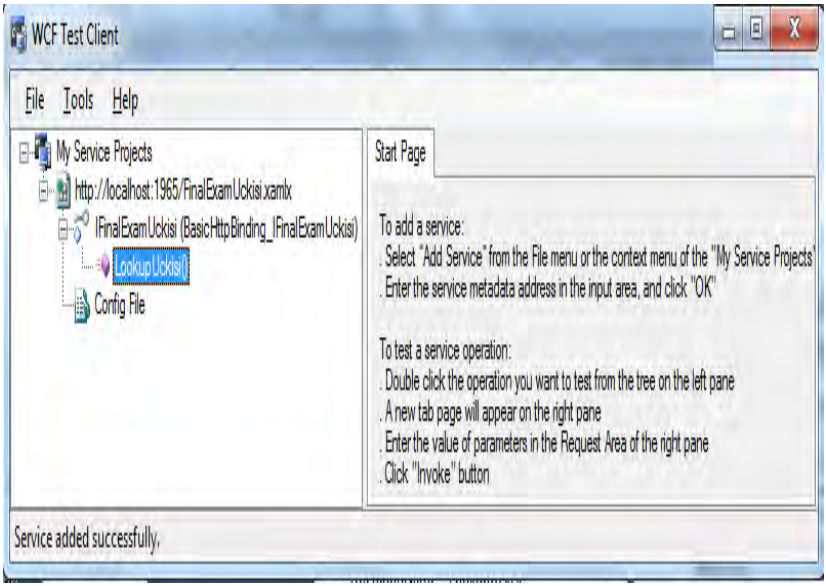
```
      l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "CheckedOut"));
     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "Missing"));
     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "Available"));
       UckisiList.Set(context, l);        }
    }
}
```

PerformLookup ქმედება ჩაემატება ინსტრუმენტების პანელზე. იგი უნდა გადმოვიტანოთ „ReceiveRequest” და „SendResponse” ქმედებებს შორის შესასრულებლად. ამასთანავე მისი Search თვისებისთვის ჩავწერეთ search, ხოლო UckisiList თვისებისთვის კი - result.

5.8. სერვისის ტესტირება

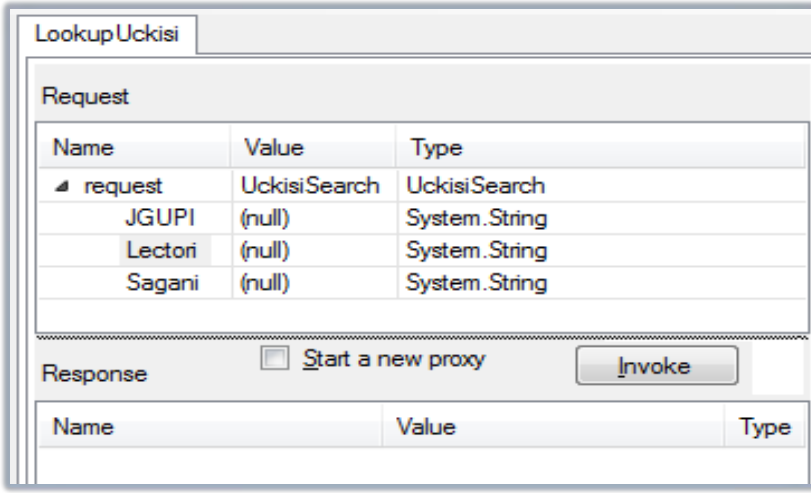
აპლიკაციის საბოლოო გამართვამდე უნდა მოვახდინოთ აგებული სერვისების ტესტირება. F5-ით ავამუშავოთ სერვისის გამართვის პროცედურა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს [107].

ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს მეთოდებს, რომლებიც გათვალისწინებულია. ეს შედეგი ჩანს 5.9 ნახაზის მარცხენა პანელზე.



ნახ.5.9. ტესტირების პროცედურა

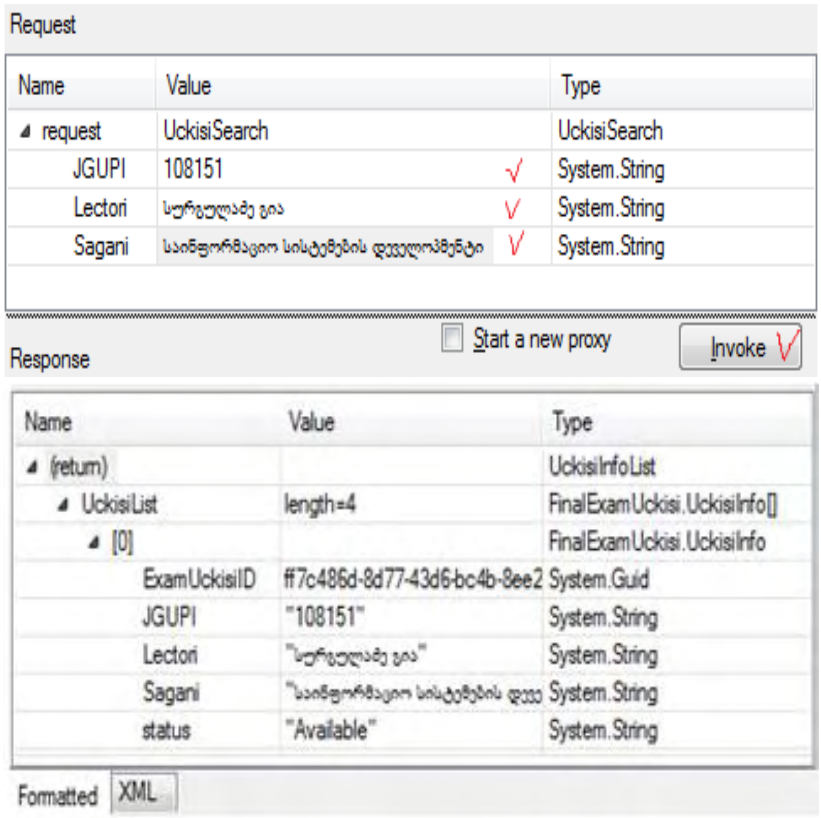
LookupUckisi() მეთოდზე 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად (ნახ.5.10).



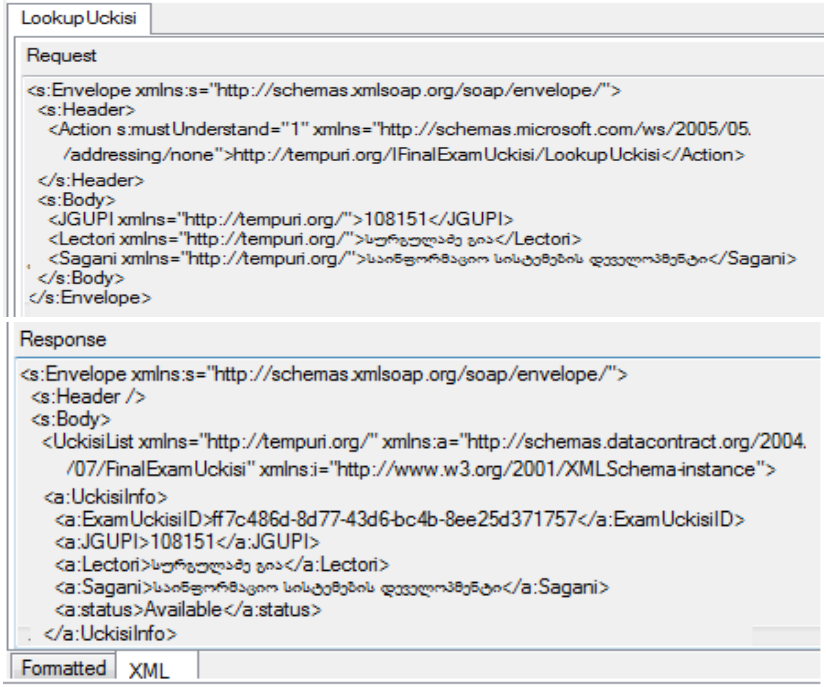
ნახ.5.10. საწყისი მონაცემების შესატანი ფანჯარა

შევიტანოთ კონკრეტული მნიშვნელობები Lectori, JGUPI, Sagani და ავამოქმედოთ Invoke ღილაკი. ცხრილებში გამოჩნდება შედეგები (ნახ.5.11).

5.12 ნახაზზე ნაჩვენებია Request და Response ცხრილების შესაბამისი ფაილები XML ფორმატში.



ნახ.5.11. WCF Client Test უტილიტით სერვისის ტესტირების შედეგების ნახვა



ნახ.5.12. ტესტირების შედეგების XML ტექსტები

5.9. მეხუთე თავის დასკვნა

უნივერსიტეტის მართვის საინფორმაციო სისტემის მაგალითზე შემუშავებულია მონაცემთა განაწილებული ბაზის და მომხმარებელთა ინტერფეისების ავტომატიზებული დაპროექტების და პროგრამული რეალიზაციის პროცესები (ტესტირების პროცედურების გათვალისწინებით), ჰიბრიდული აპლიკაციების (Windows- და Web-სისტემების) აგების ტექნოლოგიების და სერვის-ორიენტირებული არქიტექტურის საფუძველზე. მომხმარებელთა ინტერფეისების Web-სერვისები აგებულია WPF, Workflow და WCF ტექნოლოგიებით.

ლიტერატურა:

1. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., სტუ. თბ., 2012.

2. სურგულაძე გ., ურუშაძე ბ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). სტუ. თბ., 2013.

3. Booch G., Jacobson I., rambaugh J. Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara, 1995.

4. Бек К. Шаблоны реализации корпоративных приложений. Экстремальное программирование: Пер. с англ. М.: Вильямс, 2008.

5. Guru99, new kind of learning experience http://www.guru99.com.

6. Selenium Official Page http://docs.seleniumhq.org.

7. Hunt D., Newhook P., Kumar T., Selenium Documentation from Selenium Project, 2012.

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

9. Sing Li., Article - Automate web application UI testing with Selenium, 2010 http://www.developerfusion.com/article/84484/light-up-your-development-with-selenium-tests.

10. Ashfaque Ahmed, Software testing as a service, 2010.

11. David Burns, Selenium 1.0 Testing Tools, Birmingham 2010

12. "One Stop Testing", testing articles and resources: http://www.onestoptesting.com.

13. სურგულაძე გ., გულიტაშვილი მ., კაკულია ი., ჩერქეზიშვილი გ., ჯავახიშვილი ი. პროგრამული სისტემების სასიცოცხლო ციკლის პროცესის მოდელირება უნივერსალური და ექსტრემალური პროგრამირების პრინციპების კომპრომისული გადაწყვეტით.

სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(8). 2010, გვ.63-70.

14. გულიტაშვილი მ., სურგულაძე გ., ჩერქეზიშვილიბგ. Web აპლიკაციების დამუშავების პროცესის მოდელირება UML/2 ტექნოლოგიით. სტუ-ს შრ.კრ., „მართვის ავტომატიზებული სისტემები“, N 1(10), 2011. გვ.180-183.

15. გულიტაშვილი მ., სურგულაძე გ. Web-აპლიკაციების ავტომატური ტესტირება. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(12), 2012. გვ.41-44.

16. გულიტაშვილი მ. Web აპლიკაციების ავტომატური ტესტირება Selenium ტექნოლოგია. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 2(13), 2012. გვ.199-202.

17. გულიტაშვილი მ., სურგულაძე გ., ჩერქეზიშვილი გ. პროგრამული უზრუნველყოფის ტესტირების ტიპები და სცენარები. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(14), 2013. გვ.95-99.

18. სურგულაძე გ., გულიტაშვილი მ., რამიშვილი ა. Web სისტემების ტესტირების ავტომატიზაცია Open Source ტექნოლოგიების გამოყენებით. ინსო–2013, VI საერთაშ.სამეცნ.კონფ. „ინტერნეტი და საზოგადოებია“. ქუთაისი. 2013. გვ. 15-19.

19. გულიტაშვილი მ., სურგულაძე გ., ჩიხრაძე ბ. Web აპლიკაციების დამუშავების პროცესის მოდელირება UML/2 ტექნოლოგიით. საერთაშ.სამეცნ.კონფერ. მოხს.თეზისები: „მართვის ავტომატიზებული სისტემები და თანამედროვე საინფორმაციო ტექნოლოგიები“. ISBN 978-9941-14-937-4. სტუ, თბ., 2010. გვ.168-169.

20. კივილაძე ნ., სურგულაძე გ., გულიტაშვილი მ. Microsoft-ის ტექნოლოგიების ანალიზი და განვითარების ტენდენციები საინფორმაციო სისტემებისთვის. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(17), 2013. გვ.199-202.

21. ბიტარიშვილი მ. პროგრამული სისტემების მენეჯმენტის ბიზნესპროცესების კვლევა მოქნილი მოდელირების (Agile modeling) მეთოდების და ინსტრუმენტების საფუძველზე. დისერტ. დოქტ.აკად ხარისხ. სტუ, თბ., 2013.

22. Gousset M., Keller B., Woodward M.. Professional Appication Lifecycle Management with Visual Studio 2012.

23. http://en.wikipedia.org/wiki/ჯავა_(პროგრამირების_ენა).

24. http://msdn.microsoft.com/en-us/library/hh191495.aspx

25. http://msdn.microsoft.com/enus/library/dd286680 (v=vs.110) .aspx

26. AutoIT, http://www.autoitscript.com/site/

27. გოგიჩაიშვილი გ., სუხიაშვილი თ. სისტემების ობიექტ-ორიენტირებული ანალიზი და დაპროექტება. სტუ, თბილისი, 2012

28. გოგიჩაიშვილი გ., თურქია ე. პროგრამული უზრუნველყოფის რეალიზაცია Rational Rose ინსტრუმენტის ბაზაზე. სტუ, თბილისი, 2009.

29. Сургуладзе Г.Г., Гогичаишвили Г.Г. Разработка прикладного программного обеспечения интегрированных информационных систем управления на основе UML. Georgian Electronic Scientific Journal, 2002,N1,ст.42-48.

30. Martin C.R., Martin M. Agile Principles, Patterns and Practices in C#. Prentice Hall. 2005.

31. თურქია ე. ბიზნეს–პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., 2010.

32. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო. მართვის ავტომატიზებული სისტემების დაპროექტება და აგება. სტუ, თბ., 2001.

33. ფრანგიშვილი ა., სურგულაძე გ., ვაჭარაძე ი. ბიზნეს-პროგრამების ექსპერტულ შეფასებებში გადაწყვეტილებათა

მიღების მხარდამჭერი მეთოდები და მოდელები. სტუ. მონოგრ., თბილისი, 2009.

34. სურგულაძე გ., კაშიბაძე მ. ორგანიზაციულ სისტემებში ინფორმაციული რესურსების მართვა. სტუ, თბილისი, 2009.

35. სურგულაძე გ., ოხანაშვილი მ., სურგულაძე გრ. მარკეტინგის ბიზნეს-პროცესების უნიფიცირებული და იმიტაციური მოდელირება. სტუ, თბ., სტუ, 2009.

36. სურგულაძე გ., გულუა დ. განაწილებული სისტემების ობიექტ-ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. სტუ. თბ., 2004.

37. Котляров В.П., Коликова Т.В. Основы информационных технологий: Основы тестирования программного обеспечения. Интернет-Университет БИНОМ. – М., Информационные Технологий. Лаб. знаний. www.intuit.ru. 2005.

38. სურგულაძე გ., ბულია ი., თურქია ე.. Web-აპლიკაციების აგება ASP.NET & C# პაკეტების საფუძველზე. სახელმძღვანელო. სტუ, თბ., 2009.

39. msdn Library: http://msdn.microsoft.com/en-us/library/ff536015 (v=vs.95).aspx.

40. Bruce Johnson, Professional Visual Studio 2012.

41. Official documentation of Selenium project: http://docs.seleniumhq.org /docs/, 2008-2013.

42. Guru testing blogs about Selenium technology: http://www.-guru99.com /selenium-tutorial.html.

43. გოგიჩაიშვილი გ., სუხიაშვილი თ. სისტემების ობიექტ-ორიენტირებული ანალიზი და დაპროექტება. სტუ, თბილისი, 2012

44. გოგიჩაიშვილი გ., თურქია ე. პროგრამული უზრუნველყოფის რეალიზაცია Rational Rose ინსტრუმენტის ბაზაზე. სტუ, თბილისი, 2009.

45. Сургуладзе Г.Г., Гогичаишвили Г.Г. Разработка прикладного программного обеспечения интегрированных информационных систем управления на основе UML. Georgian Electronic Scientific Journal, 2002,N1,ст.42-48

46. Chris Anderson. Pro Business Applications with Silverlight 4. (Expert's Voice in Silverlight) – Apress, 2012.

47. msdn Library: http://msdn.microsoft.com/enus/library/ff536015 (v=vs.95).aspx.

48. Bruce Johnson, Professional Visual Studio 2012.

49. Official documentation of Selenium project: http://docs.seleniumhq.org /docs/, 2008-2013.

50. Мак-Дональд М. WPF: Windows Presentation Fundation в .NET 3.5 с примерами на C# 2008 для профессионаюв. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильяме". 2008

51. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი (WPF ტექნოლოგია). სტუ, თბ., 2013.

52. Freeman A. Pro ASP.NET MVC 4. Apress, 2013.

53. Sanderson S. Pro ASP.NET MVC Framework (Expert's Voice in .NET) – Apress, 2009.

54. Msdn Library: https://msdn.microsoft.com/en-us/magazine/-dd942838.aspx.

55. სურგულაძე გ., თურქია ე., ქაჩლიშვილი თ., ფხაკაძე ც. საფინანსო კორპორაციის ბიზნეს-პროცესების მენეჯმენტი ITIL მეთოდოლოგიის საფუძველზე (რეპორტების ავტომატიზაცია). სტუ-ს შრ.კრ. „მას“. 2, 18, თბ., 2014, გვ.51-55.

56. სურგულაძე გ., კივილაძე ნ. Web-აპლიკაციაში რეპორტების ინტეგრაციის მეთოდები. VII-საერთაშ.სამეცნ.-პრაქტ.კონფ. „ინტერნეტი და საზოგადოება“ (INSO 2015). ქუთაისი. 3-5 ივლისი, 2014.

57. Misner S. Microsoft SQL Server 2012. Reporting Services (Developer Reference).

58. MSDN. microsoft.com. library/reportviewer controls (visual studio). 2013.

59. technet.microsoft.com. Library / integrating reporting services into your application. 2013.

60. ბოტჰე კ., სურგულაძე გ., დოლიძე თ., შონია ო., სურგულაძე გ. თანამედროვე პროგრამული პლატფორმები და ენები (Unix, Linux, Windows, C++, Java). სტუ, თბილისი, 2003.

61. JavaScript Language. https://en.wikipedia.org/wiki/JavaScript

62. Stewart S. WebDriver. The 2nd Annual Google Test Automation Conference (GTAC) in our New York office on August 23 and 23. https://www.youtube.com/watch?v=tGu1ud7hk5I

63. Walther St., The Evolution of MVC. http://stephenwalther-.com/archive/2008/08/24/the-evolution-of-mvc. 2008.

64. Msdn Library: ASP.NET MVC Application Using Entity Framework Code First: https://code.msdn.microsoft.com/ASPNET-MVC-Application-b01a9fe8.

65. Halpin T. ORM-2 Graphical Notation. Neumont Univer., 2004. http://www.orm.net/ pdf/ORM2_TechReport1.pdf

66. Surguladze G., Turkia E., Topuria N., Lominadze T., Giutashvili M. Automation of Business-Processes of an Election System. IV-Intern.Conf. “Problems of Cybernetics and Informatics“ (PCI’ 2012). Baku, Azerbaijan, 2012. pp. 16-19

67. ვედეკინდი ჰ., სურგულაძე გ., თოფურია ნ. განაწილე-ბული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. მონოგრ., სტუ, თბ., 2005.

68. სურგულაძე გ., თოფურია ნ., ბაკურია კ., ლომიძე მ. საინფორმაციო სისტემის დაპროექტება ობიექტ-როლური მოდელი-რების და სერვის-ორიენტირებული არქიტექტურის ბაზაზე. სტუ-ს

შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(17), თბილისი, 2014, გვ. 32-44.

69. Collins M.J. Beginning WF: Windows Workflow in .NET 3.0. ISBN-978-1-4302-2485-3 Copyright © 2010. USA. http://www.ebooks-it.net/ebook/beginning-wf.

70. Уотсон К.,Нейгел К., Педерсен Я., ХаммерР., Джон Д., Скиннер М., Уайт Э. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2009

71. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

72. Anurag S. WCF: From a Beginner's perspective & a Tutorial. V, 4 Apr 2013.

73. http://www.codeproject.com/Articles/566691/WCF-From-a-Beginners-perspective-a-Tutorial.

გადაეცა წარმოებას 10.06.2015 წ. ხელმოწერილია დასაბეჭდად 15.06.2015 წ. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 15,25. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის ცენტრი“
(თბილისი, მ.კოსტავას 77)