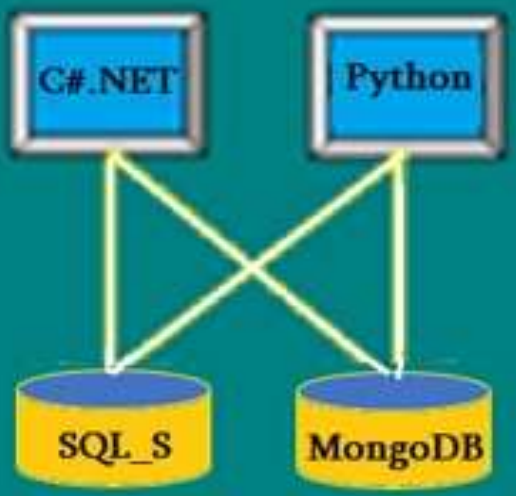
საქართველოს ტექნიკური
უნივერსიტეტი
1922 წლიდან

გია სურგულაძე, ლია ჰეტრიაშვილი

აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი



სტუ-ს IT-კონსალტინგის ცენტრი

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ლილი პეტრიაშვილი

აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი

(ლაბორატორიული პრაქტიკუმის მეთოდური სახელმძღვანელო)

დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ ოქმი N7, 05.02.2022

თბილისი
2022

უაკ 004.5

წარმოდგენილა საქართველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის „ინფორმატიკის“ საბაკალავრო პროგრამის აკადემიური დისციპლინის „აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი“ ლაბორატორიული პრაქტიკუმის ამოცანები და მათი შესრულების დამხმარე სახელმძღვანელო. განხილულია გამოყენებითი პროგრამული სისტემების დეველოპმენტის საკითხები მათი აპლიკაციების სასიცოცხლო ციკლის ეტაპების შესაბამისად, მაიკროსოფთის Visual Studio .NET Framework პლატფორმის გამოყენებით. კერძოდ აღწერილია: Console App, Windows Forms App, ASP.NET Web App, WPF App აპლიკაციებით მომხმარებელთა ინტერფეისების დაპროგრამების და ტესტირების ამოცანები. გამოიყენება ვიზუალური პროგრამირების C# და Python ენები, ხოლო მონაცემთა ბაზები წარმოდგენილია რელაციური SQL Server-ის და NoSQL-ის MongoDB Compass და Atlas (ღრუბლოვანი ტექნოლოგიით) პაკეტებით.

მეთოდური სახელმძღვანელო განკუთვნილია სტუ-ის ინფორმატიკის ფაკულტეტის პროგრამული ინჟინერიის კონცენტრაციის სტუდენტებისათვის.

რეცენზენტი:

ე. თურქია (საქართველოს ეროვნული ბანკი, ტ.მ.კ.)

გ. ნარეშელაშვილი (სტუ-ს პროფესორი)

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, ვ. კვარაცხელია, თ. ლომინაძე, ნ. ლომინაძე, ჰ. მელაძე, თ. ობგაძე, გ. სურგულაძე (რეადაქტორი), გ. ჩაჩანიძე, ა. ცინცაძე, ზ. წვერაიძე, ო. შონია

ISBN 978-9941-8-3810-1

სარჩევი

1. სასწავლო კურსის მიზანი

„აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი“-ს აკადემიური კურსი შეასწავლის სტუდენტებს გამოყენებითი პროგრამული სისტემების აგების სასიცოცხლო ციკლის პროგრამული დეველოპმენტის ეტაპის საკითხებს, მათი აპლიკაციების რეალიზაციას მაიკროსოფთის ინსტრუმენტული (ვიზუალური და არავიზუალური) საშუალებებით C#.NET და Python ენების ბაზაზე, მომხმარებელთა ინტერფეისების დაპროგრამებას მონაცემთა რელაციური და NoSQL ბაზებთან სამუშაოდ.

2. საგნის შესწავლის შედეგად მიღებული ცოდნა და შეძენილი უნარები

საგნის შესწავლის შედეგად სტუდენტი ღებულობს ცოდნას და იძენს შემდეგ უნარებს:

1) განსაზღვრავს ობიექტ-ორიენტირებული დაპროგრამების ძირითად პრინციპებს: ინკაფსულაციას, მემკვიდრეობითობას, პოლიმორფიზმსა და აბსტრაქციას;

2) ერთმანეთისგან განასხვავებს პროგრამული აპლიკაციების აგების ფორმებს და სახეებს;

3) აცნობიერებს მომხმარებეთა ინტერფეისების პროგრამირების შესაძლებლობებს მონაცემთა რელაციურ და NoSQL ბაზებთან სამუშაოდ აპლიკაციების აგების მიზნით;

4) იყენებს C# და Python ენების ვიზუალურ და არავიზუალურ კომპონენტებს მომხმარებლის ინტერფეისების ასაგებად;

5) არჩევს მონაცემთა რელაციურ და NoSQL ბაზებთან სამუშაოდ საჭირო ინტერფეისებს, კლასებსა და მეთოდებს დასმული ამოცანის სპეციფიკიდან გამომდინარე;

6) განსაზღვრული მითითებების შესაბამისად დამოუკიდებლად ქმნის პროგრამულ აპლიკაციას, მონაწილეობს პროგრამული სისტემის გუნდური დამუშავების პროცესში.

3. საათების განაწილება (სტუდენტის დატვირთვა)

„აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი“-ს კურსი 5 კრედიტიანია (1 კრ = 25 სთ). სწავლის ფორმებია: ლექცია (15 სთ/სემესტრში), პრაქტიკული მეცადინეობა (30 სთ) ტარდება ლაბორატორიაში და დამოუკიდებელი მუშაობა (77 სთ). აქვს შუასემესტრული (1 სთ) და დასკვნითი გამოცდა (2 სთ).

4. პრაქტიკული მეცადინეობების თემების დასახელება და შინაარსი

„აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი“-ს კურსის პრაქტიკული მეცადინეობები ტარდება კომპიუტერულ ლაბორატორიაში. მასალა მოიცავს შემდეგ თემებს და საკითხებს:

1. *პროგრამული აპლიკაციების აგების პლატფორმები და ფრეიმვორკები:* მაიკროსოფთის Visual Studio.NET პლატფორმის სამუშაო გარემოში, Framework, Core და Code ვერსიების გაცნობა, გამოყენებითი სფეროების საილუსტრაციო აპლიკაციების მიმოხილვა;

2. *კომპიუტერული პროგრამირების თანამედროვე მეთოდები და მეთოდოლოგიები:*

პროგრამირების პარადიგმები და პროგრამული აპლიკაციების აგების ინსტრუმენტები; პროგრამირების მეთოდი, სტილი, მოდელი, ალგორითმი და ენა – საილუსტრაციო მაგალითების განხილვა და მარტივი პროგრამული კოდების აწყობა C# და Python ენების საფუძველზე

3. პროგრამირების ინსტრუმენტული საშუალებები, გამოყენების პრინციპები, შეფასება და ხარისხი: ობიექტ-ორიენტირებული ენების საფუძველზე (C# და Python) კლასების და ობიექტების დაპროგრამება, მათი თვისებებით და ფუნქციებით; ექსპერიმენტის ჩატარება პროგრამული კოდების მუშაობის მახასიათებლების ანალიზის მიზნით

4. გამოყენებითი პროგრამული აპლიკაციის ინტერფეისის დაპროექტება (დიზაინი) და დაპროგრამება: მომხმარებლის ინტრფეისის აგების პროცესის დაგეგმვა. პროგრამული უზრუნველყოფის აგების სასიცოცხლო ციკლის დეველოპმენტის ეტაპის შერულება. აგებული C# და Python პროგრამული კოდების ანალიზი, ოპტიმიზაცია (კოდის მოცულობის, სწრაფქმედების და სხვ.) და მათი შედარება.

5. პროგრამული აპლიკაციის ინფორმაციული უზრუნველყოფის აგება რელაციური და NoSQL ბაზების საფუძველზე და მათი შედარებითი ანალიზის ჩატარება;

– Ms SQL Server Management Studio სამუშაო გარემოში ექსპერიმენტული ბაზის სტრუქტურისა და ცხრილების აგება;

– მოთხოვნების დამუშავება და შედეგების ანალიზი

6. მომხმარებლის ინტერფეისის აგება ობიექტ-ორიენტირებული კომპონენტებით C#.NET ენისა და MsSQL Server ბაზების პაკეტის საფუძველზე:

– C#-ის Console Appliocation რეჟიმში აგებული პროგრამის მუშაობა (წინა პრაქტიკულზე აგებულ) ექსპერიმენტულ SQL Server მონაცემთა ბაზასთან;

7. მომხმარებლის ინტერფეისის აგება ვიზუალური კომპონენტებით C#.NET ენისა და MsSQL Server ბაზების პაკეტის საფუძველზე:

– C#-ის Windows Forms Appliocation რეჟიმში აგებული პროგრამის მუშაობა (წინა ლაბორატორიაზე აგებულ) ექსპერიმენტულ SQL Server მონაცემთა ბაზასთან, ADO.NET დრაივერის DataSet და სხვ. ობიექტების საფუძველზე;

– ექსპერიმენტის ჩატარება C# ენის ობიექტ-ორიენტირებული და ვიზუალური მეთოდებით მონაცემთა განახლების ოპერაციების დაპროგრამების მიზნით (Insert, Update, Delete);

8. მომხმარებლის ინტერფეისის აგება ობიექტ-ორიენტირებული და ვიზუალური კომპონენტებით C#.NET ენისა და MongoDB Compass ბაზის საფუძველზე:

– MongoDB Compass პაკეტის (ინსტალაცია) საფუძველზე მონაცემთა ექსპერიმენტული ბაზის აგება;

– C#-ის კონსოლური პროექტის მუშაობის მიზნით (წინა ლაბორატორიაზე აგებულ) ექსპერიმენტულ MongoDB ბაზასთან MongoDB.Driver დრაივერის დაყენება პაკეტების მენეჯერის NuGet-ის დახმარებით;

– ბაზასთან მუშაობის ბიბლიოთეკების MongoDB.BSON.dll, MongoDB.Driver и MongoDB.Driver.Core გაცნობა და ექსპერიმენტული გამოყენება C# და MongoDB ტანდემის ერთობლივი მუშაობისათვის

9. MongoDB ბაზის დოკუმენტებისა და კოლექციების განახლება: MongoDB ბაზაში დოკუმენტების ამორჩევის, ფილტრაციის, განახლების C# მეთოდების (ოპერაციების) დაპროგრამება. კოლექციის ელემენტების რედაქტირების მეთოდებით UpdateOneAsync(), UpdateManyAsync() და ReplaceOneAsync() დოკუმენტების განახლების პროგრამული პროექტის აგება და ფუნქციონიების მახასიათებლების კვლევა.

10. მომხმარებლის ინტერფეისის აგება Python ენაზე ობიექტ-ორიენტირებული კომპონენტებით MsSQL Server ბაზებთან ADO.NET დრაივერის გამოყენებით:

– Python-ის Console Appliocation რეჟიმში აგებული პროგრამის მუშაობა (წინა ლაბორატორიაზე აგებულ) ექსპერიმენტულ SQL Server მონაცემთა ბაზასთან;

– CRUD ოპერაციების შესრულება Python SQL ბიბლიოთეკასთან SQL სერვერისთვის (Create, Read, Update, Delete ოპერაციების პროგრამირება);

11. მომხმარებლის ინტერფეისის აგება Python ენის ობიექტ-ორიენტირებული კომპონენტებით MongoDB ბაზასთან:

– MongoDB მონაცემთა ბაზის შექმნა MongoDB Atlas-ის გამოყენებით;

– PyMongo დრაივერის დაყენება (დააინსტალირება).

12. MongoDB ბაზის დოკუმენტებისა და კოლექციების განახლება:

– MongoDB-სთან დაკავშირება. მისი კოლექციების და დოკუმენტების გაცნობა;

– ძირითადი CRUD-ოპერაციების (შექმნა, წაკითხვა, განახლება და წაშლა) შესრულება PyMongo-ს გამოყენებით;

13. პროგრამული ვებ-აპლიკაციის აგება C#-ით, ASP.NET MVC პლატფორმაზე MongoDB ბაზისთვის: MVC (Model, View, Controller) პატერნის გამოყენება, მოდელირების, ასახვის და პროგრამირების კომპონენტები მომხმარებელის ინტერფეისის ასაგებად;

14. პროგრამული ვებ-აპლიკაციის აგება Python-ით MVC პატერნით Django Framework-ის პლატფორმაზე: Python ენის შესაძლებლობები MVC (Model, View, Controller) რეალიზაციის მიზნით. Django ფრეიმვორკის გამოყენებით;

15. პროგრამული აპლიკაციების მოდულური ტესტირება:

– Unit-ტესტირება .NET პლატფორმაზე C#.NET პროგრამებისათვის;

– Unit-ტესტირება .NET პლატფორმაზე Python პროგრამებისათვის;

ტესტირების პროცესების შეფასება და ცვლილებების განხორციელება ოპტიმალური შედეგების მისაღებად.

5. აპლიკაციათა დაპროგრამების სახეები

პროგრამული აპლიკაციების ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო (Desktop) აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [1]. არსებობს პროგრამირების ჰიბრიდული ტექნოლოგიებიც, რომელთაც აქვს შესაძლებლობა ააგონ როგორც ვინდოუს, ასევე ვებ აპლიკაციები. ჩვენ აქ განვიხილავთ ყველა მათგანს.

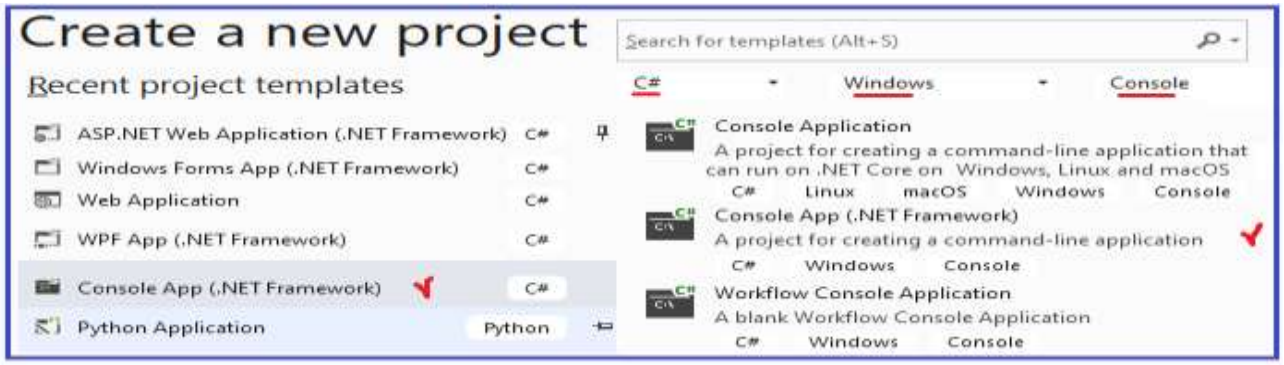
დანართები (აპლიკაციები) იქმნება .NET Framework -ის სხვადასხვა პაკეტით. პირველი - Windows Forms კომპონენტებით და მეორე ASP.NET-ის საშუალებით. ორივეს აქვს თავისი უპირატესობები და ნაკლოვანებანი. კერძოდ, სამაგიდო დანართები ძალზე მოქნილი და რეაქციულია, ხოლო Web-დანართები ინტერნეტის საშუალებით იძლევა დისტანციური წვდომის საშუალებას ერთდროულად მრავალი მომხმარებლისთვის. მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები სულ უფრო და უფრო იშლება [2].

ჰიბრიდული პროგრამირებიდან ჩვენ გავეცნობით მაიკროსოფთის WPF (Windows Presentation Foundation), WF (Worlflow) და WCF (Windows Communication Foundation) ტექნოლოგიებს, რომლებიც Visual Studio.Net პაკეტის კუთვნილებაა [3-5].

6. პრაქტიკული სამუშაოს შესრულების ნიმუშები

6.1. აპლიკაციების დაპროგრამება კონსოლის რეჟიმში (Console App)

Ms Visual Studio.NET პლატფორმის ბოლო ვერსიები (2019, 2022) ითვალისწინებს კონსოლური აპლიკაციების შექმნის სხვადასხვა ინსტრუმენტულ საშუალებას, რომლებიც დაპროგრამების ენაზე ან ტექნოლოგიაზეა დამოკიდებული (ნახ.6.1).



ნახ.6.0. Console App -ის შექმნის ვარიანტები

ჩვენ განვიხილავთ კონკრეტულ ამოცანებს და მათ პროგრამულ რეალიზაციას.

6.1.1. ლაბ-1: პროგრამული აპლიკაციის აგება დაპროგრამების რამდენიმე ენის საფუძველზე (.dll ფაილების შექმნა)

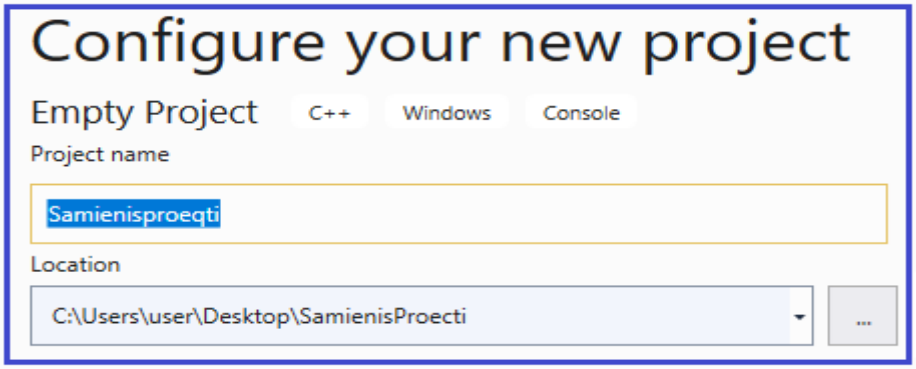
მიზანი: .NET -ის კონცეფციის რეალიზაცია .dll კოდების საფუძველზე. ობიექტ-ორიენტირებული დაპროგრამების ენების: C++, Visual Basic და C# გამოყენებით ერთი პროგრამული აპლიკაციის (პროექტის) აგება და მულტიპროგრამულ .NET პლატფორმაზე [6].

- *სამუშაო გეგმა:*
  - შეიქმნას C++ -ის საბაზო კლასი;
  - შეიქმნას Visual Basic.NET კლასი, რომელიც იქნება C++ - კლასის მემკვიდრე;
  - შეიქმნას C# კლასი, რომელიც კონსოლის აპლიკაციაში შექმნის Visual Basic.NET კლასის ეგზემპლარს და გამოიძახებს მის მეთოდს.

- *პრაქტიკული ნაწილი:*

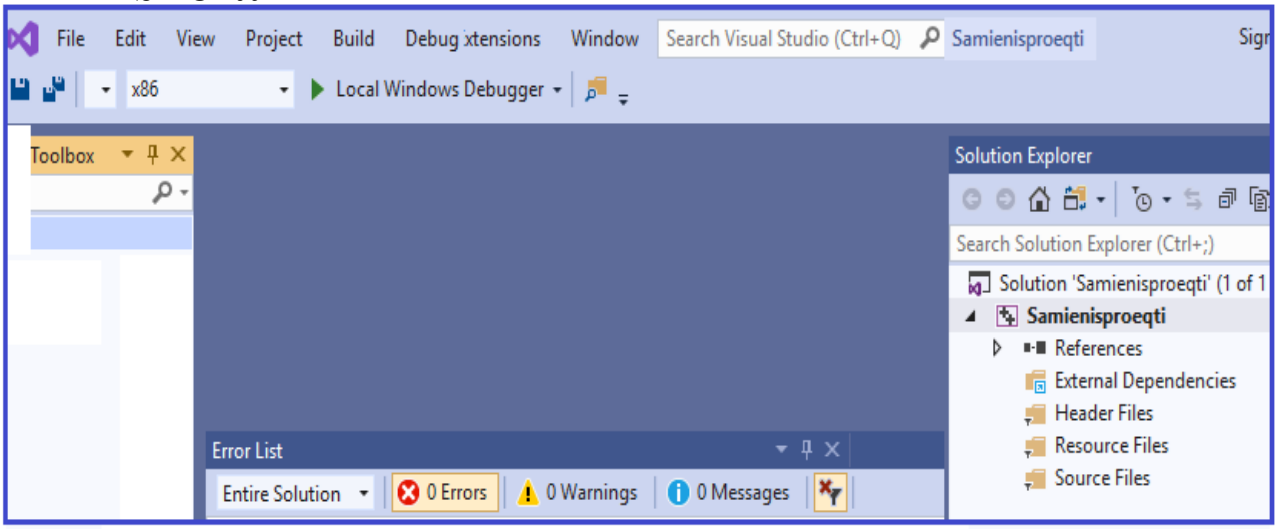
1. შევქმნათ ახალი პროგრამული პროექტი: **Create a new Project -> Next**

მიიღება ფანჯარა, სადაც განისაზღვრება პროექტის სახელი, მაგალითად, Samieenisproeqti და შენახვის ადგილი - Location (ნახ.6.2).



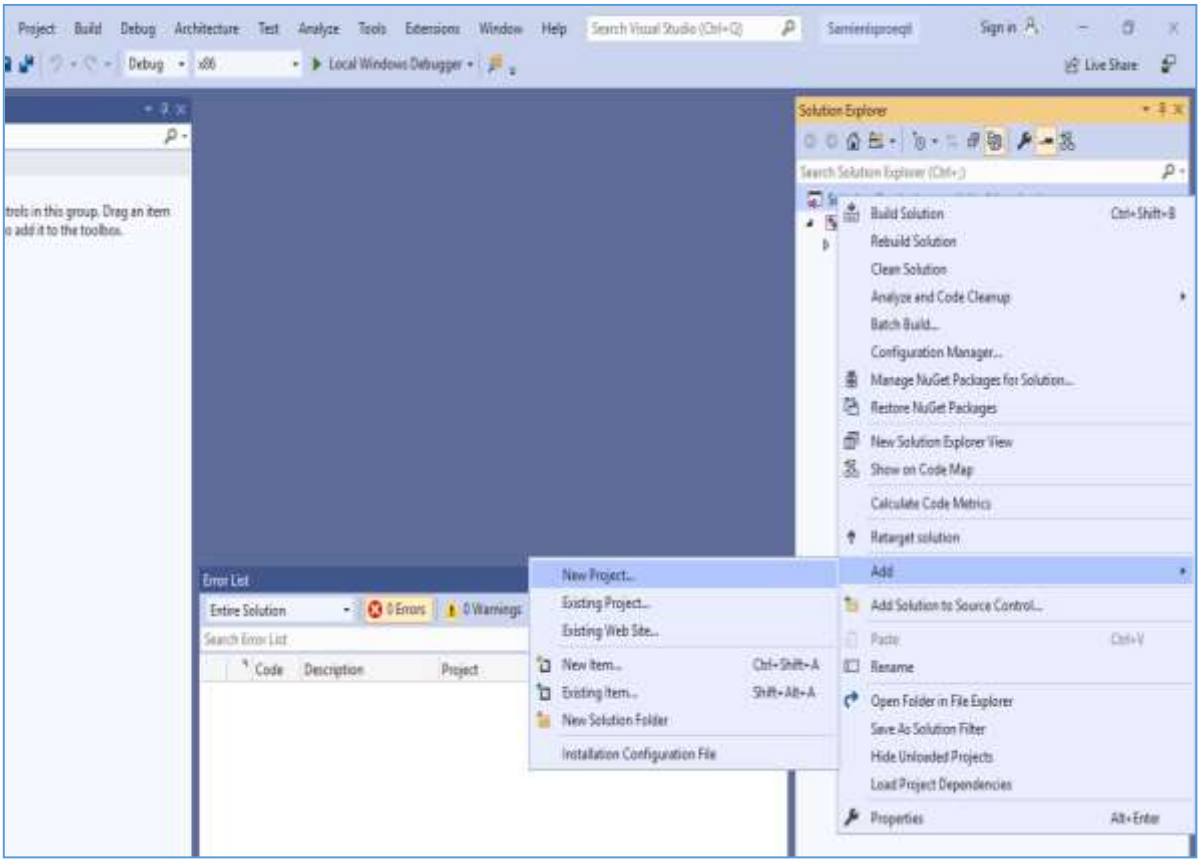
ნახ.6.2

2. მიიღება ფანჯარა (ნახ.6.3).



ნახ.6.3

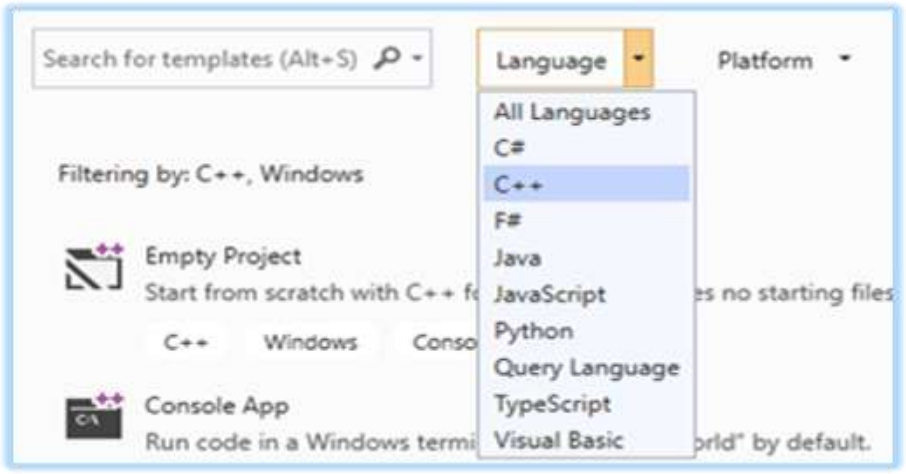
Solution Explorer - ის არეში ვამატებთ ახალ პროექტს. ვქმნით კლასს C++ ენის ბაზაზე. ამისათვის მოვნიშნავთ Solution SamienisProeqti-ს კონტექსტური მენიუდან ნიმუშის შესაბამისად ვამატებთ ახალ პროექტს (ნახ.6.4).



ნახ.6.4

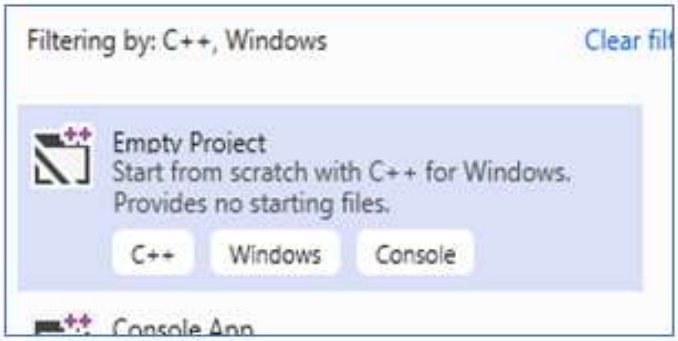
3. პროექტში შევქმნათ C++.NET ენის კლასი.

ცარიელ პროექტს (Solution ‘SamienisProjecti’) დავამატოთ ახალი (ქვე)პროექტი მასზე მარჯვენა ღილაკით და შემდეგ **Add -> New Project** არჩევით, მიიღება ფანჯარა, საიდანაც Lenguage ჩამონათვალიდიან, ვირჩევთ C++ ენას (ნახ.6.5).



ნახ.6.5

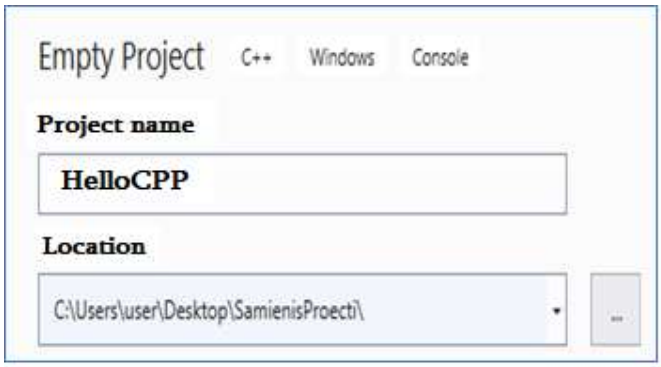
პროექტისთვის კი ვირჩევთ empty Project-ს (ნახ.6.6-ა).



ნახ.6.6-ა

შენიშვნა! წინა ვერსიებისგან განსხვავებით Blank Solution-ის ნაცვლად VS2019-ში გვაქვს empty Project.

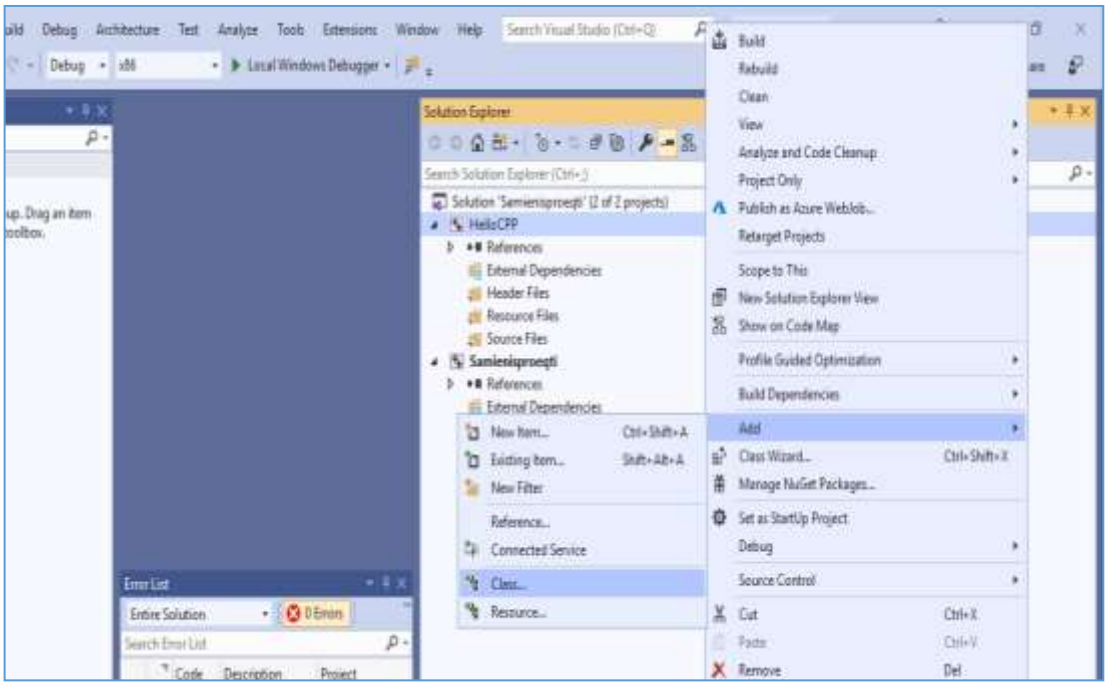
Project Name - ში ვწერთ სახელს, ხოლო ლოკაცია გააქტიურდება ავტომატურად (რომლის შეცვლაც საჭიროების შემთხვევაში შესაძლებელია) (ნახ.6.6-ბ).



ნახ.6.6-ბ

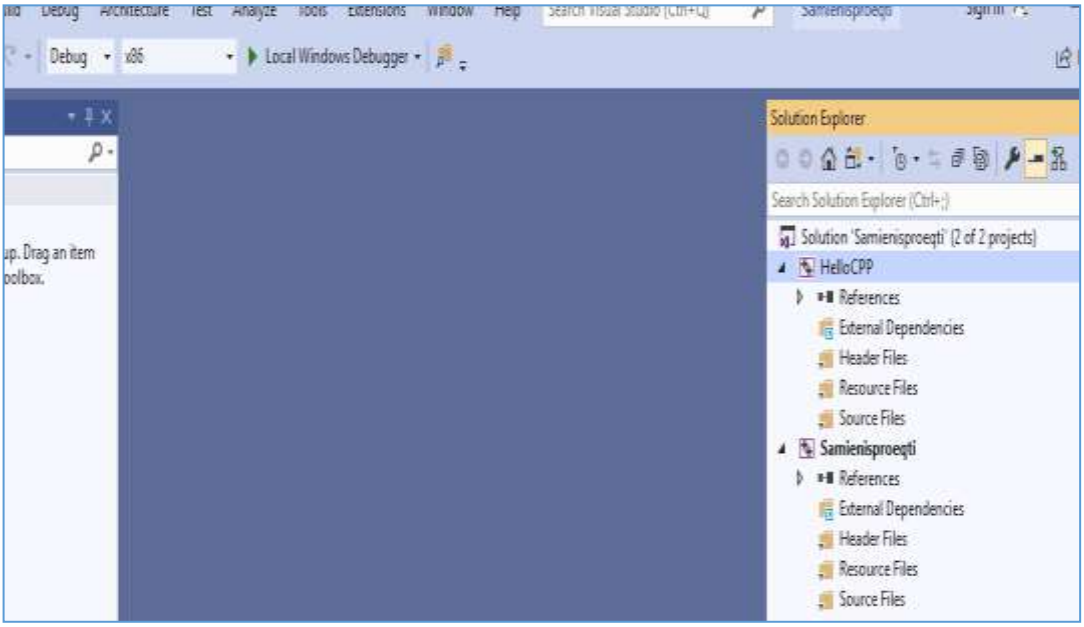
Create ღილაკის გააქტიურების შემდეგ შეიქმნება აპლიკაცია, რომელიც ნეიტრალური იქნება დაპროგრამების ენებისადმი.

შედეგად მივიღებთ 6.7 ნახაზზე ნაჩვენებ ფანჯარას.



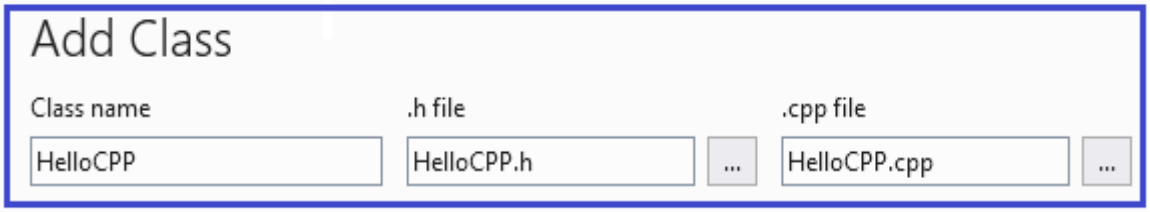
ნახ.6.7

კლასის შესაქმნელად მოვნიშნავთ HelloCPP და ნიმუშის შესაბამისად (ნახ.6.8), ვააქტიურებთ ბრძანებებს: Add → Class.

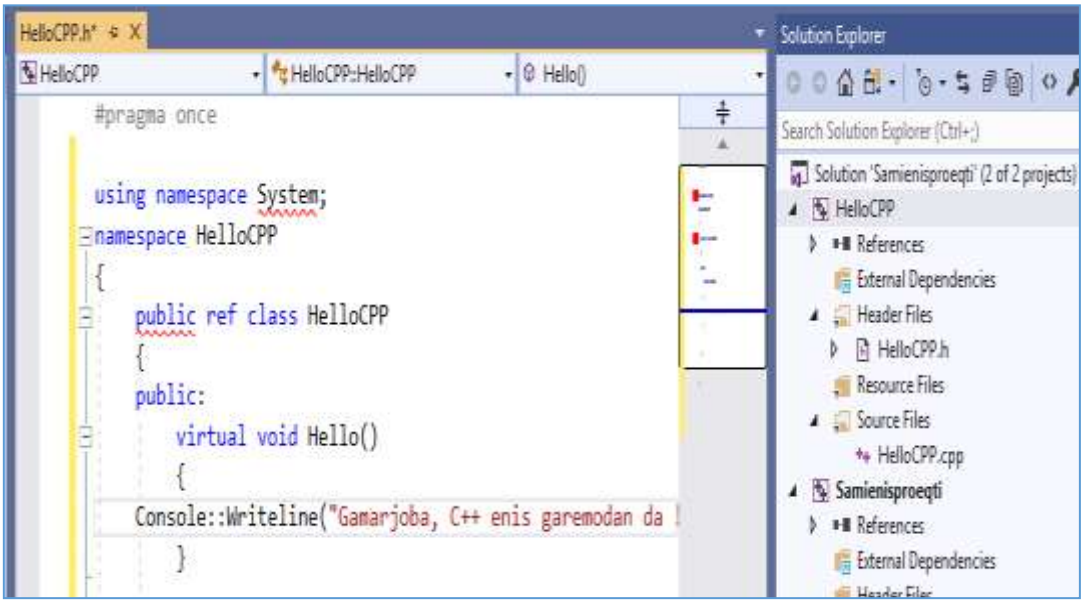


ნახ.6.8

შედეგად მიიღება ფანჯარა, სადაც Class name-ში უნდა ჩავწეროთ კლასის დასახელება (ნახ. 6.9).



ნახ.6.9



ნახ.6.10

გავხსნათ Hello.CPP.h ფაილი. მისი კოდი მოცემულია 6.1 ლისტინგში.

```
// ---- ლისტინგი_6.1 -- შეცვლილი HelloCPP.h ----
#pragma once
using namespace System;
namespace HelloCPP
{
   public ref class HelloCPP
    {
        // --- აქ ჩაემატება კლასის მეთოდი ----
       public:
    virtual void Hello()
     {
        Console::WriteLine("Mogesalmebit C++ enis garemodan da !\n viZaxeb
Visual Basics !\n\n“);
      }
    };
}
```

HelloCPP::Hello() მეთოდი გამოიყენებს .NET Framework კლასების ბიბლიოთეკიდან System::Console::WriteLine() ფუნქციას, რათა კონსოლზე გამოიტანოს მისალმება C++ კოდიდან.

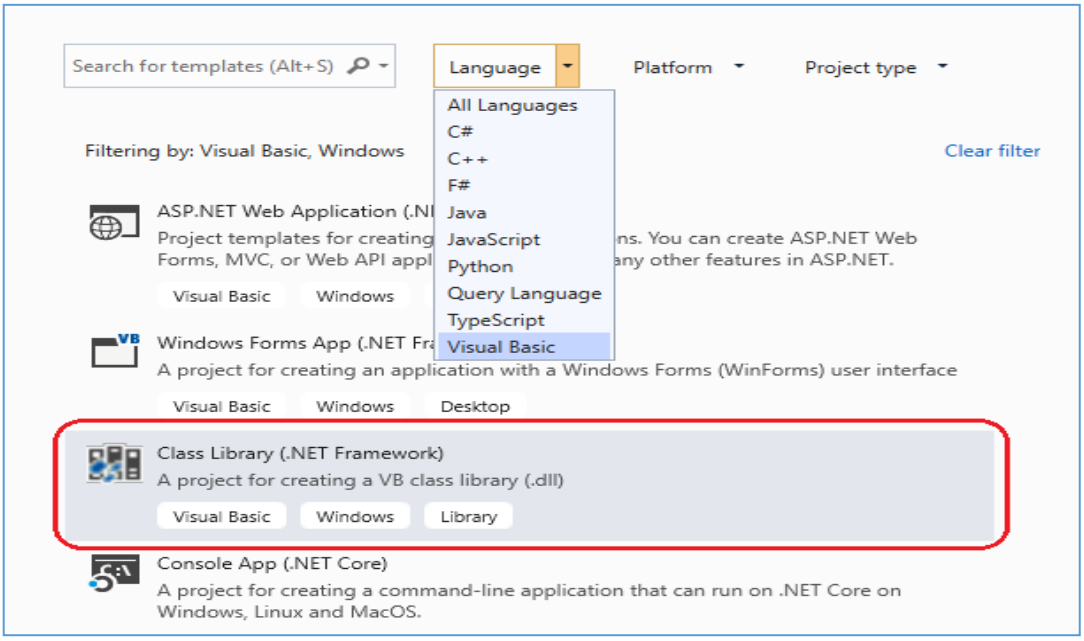
პროექტი გავუშვათ სინტაქსური შეცდომების გასასწორებლად, თუ ასეთი იქნება. იგი ჯერ არ უნდა ავამუშავოთ შესრულებაზე.

4. პროექტში კლასის შექმნა Visual Basic.NET ენაზე

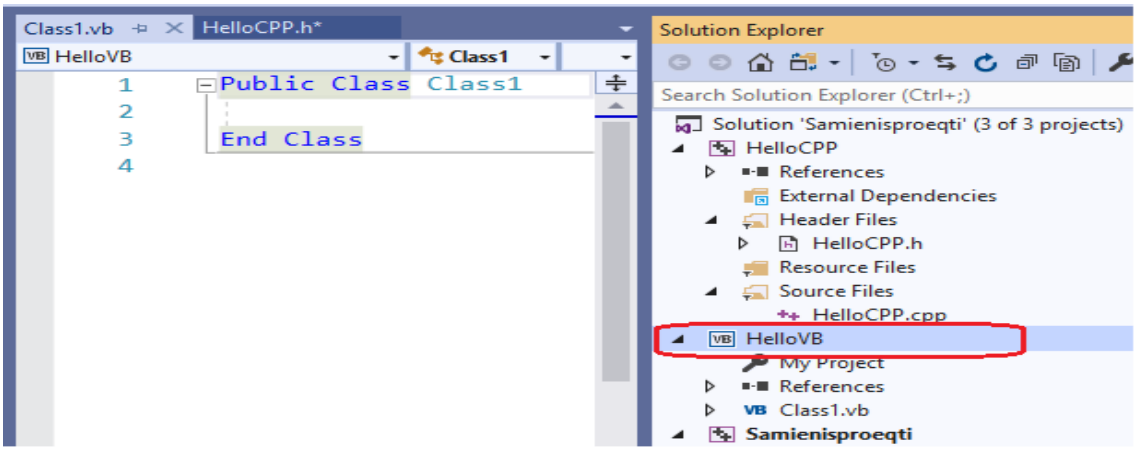
დავამატოთ Solution-ში ახალი პროექტი და მივუთითოთ, რომ იგი შეიქმნას Visual Basic .NET -ში, ავირჩიოთ Visual Basic ენა და ჩანართი: Class Library;

პროექტი შევინახოთ სახელით Name: HelloVB (ნახ.6.11).

მივიღებთ ასეთ შედეგს (ნახ.6.12). ახალი პროექტი HelloVB დაემატება Solution Explorer პანელზე თავისი შემადგენელი ფაილებით.



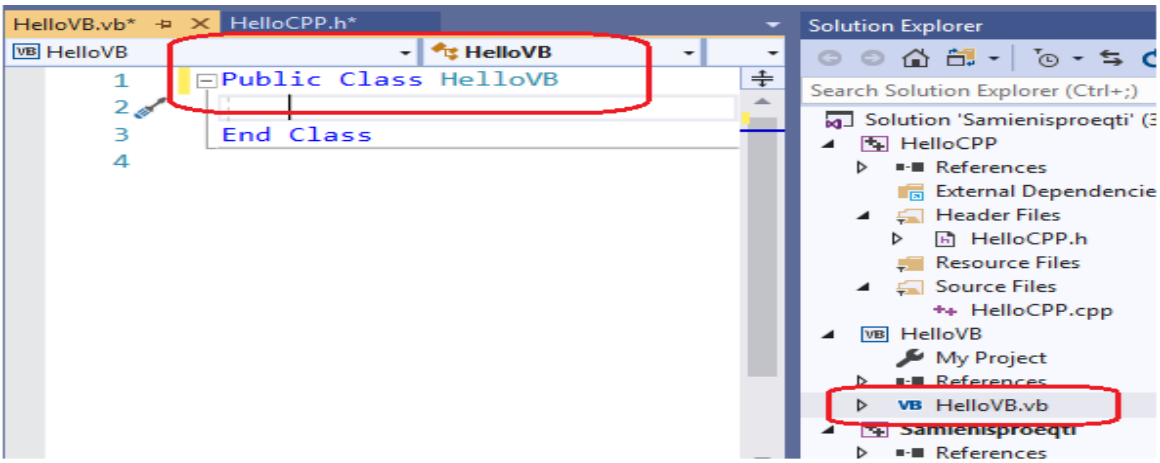
ნახ.6.11



ნახ.6.12

ახლა უკვე გვაქვს HelloCPP და HelloVB ორი პროექტი.

მეორეში ავირჩიოთ Class1.vb ფაილი და შევუცვალოთ სახელი შინაარსის გათვალისწინებით, მაგალითად, HelloVB.vb (ნახ. 6.13),



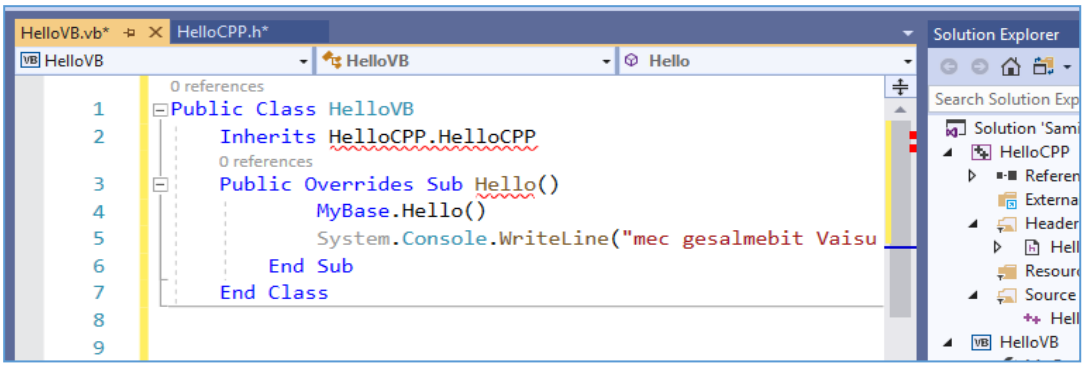
ნახ.6.13

შემდეგ გავხსნათ იგი და ჩავწეროთ ჩვენთვის საჭირო ტექსტი (ლისტინგი 6.2).

```
// -- ლისტინგი_6.2 ---  HelloVB.vb ------------
Public Class HelloVB
  Inherits HelloCPP.HelloCPP
  Public Overrides Sub Hello()
    MyBase.Hello()
    System.Console.WriteLine("Mec mogesalmebiT  Visual Basic.NET-idan !!")
  End Sub
End Class
```

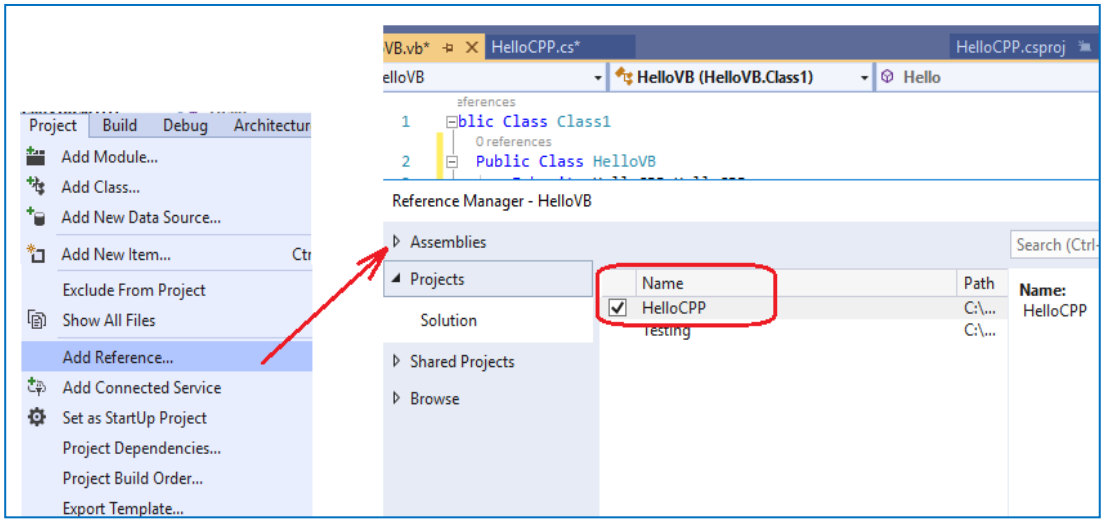
ეს კოდი განსაზღვრავს HelloVB კლასს, რომელიც მემკვიდრეობით იქნა მიღებული C++ ის მმართველი კლასიდან HelloCPP. ამგვარად, HelloVB კლასი ჩაანაცვლებს (Overrides) მშობელი HelloCPP კლასის ვირტუალურ Hello() მეთოდს, ოღონდ ჯერ გამოიძახებს Hello() მეთოდის ვერსიას მშობელი კლასიდან HelloCPP, შემდეკ კი გამოყავს ეკრანზე თავისი მისალმება.

საყურაღებოა, რომ კოდის რეადაქტორში საბაზო კლასის სახელი (HelloCPP.HelloCPP) და ჩასანაცვლებელი მეთოდის სახელი Hello() ტალღისებური ხაზითაა. ეს ნიშნავს, რომ კოდის რედაქტორი ამ სახელებს ვერ ხედავს და წინასწარ იძლევა სინტაქსურ შეცდომას. თუ კურსორს დავაყენებთ ამ ფრაგმეტზე, იგი მოგვცემს შეცდომის მნიშვნელობას (ნახ.6.14).



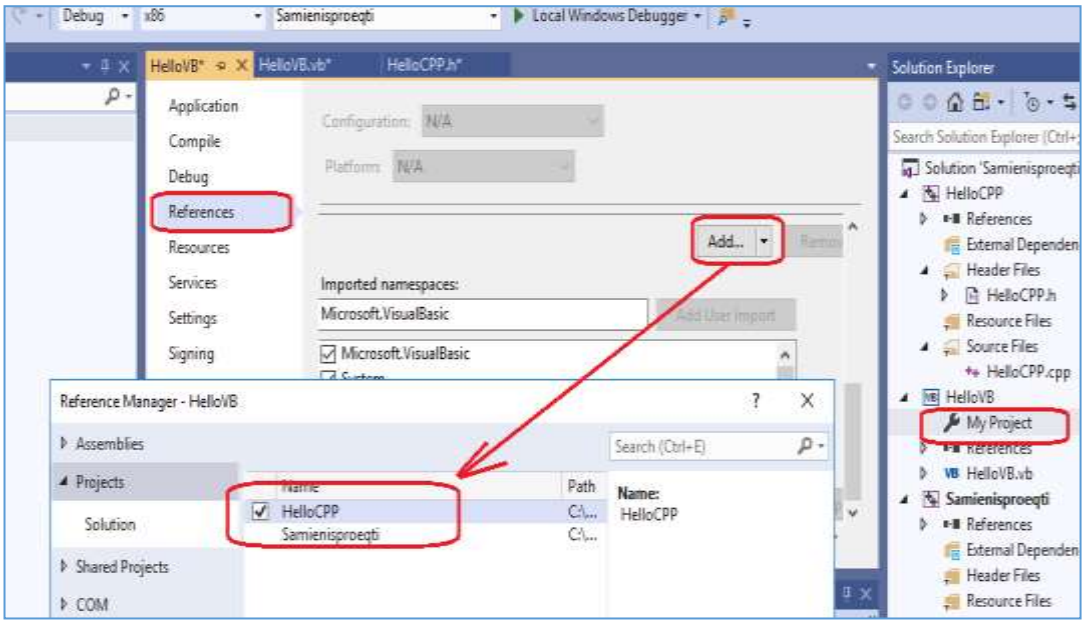
ნახ.6.14

აუცილებელია Visual Basic-ის კომპილატორს მიეთითოს თუ სად იმყოფება ეს ნაკრები (assembly), რომელიც განსაზღვრავს მოცემულ ტიპს. ამისათვის მთავარი მენიუს Project-> Add Reference->Projects-დან ავირჩევთ HelloCPP-ს და OK. (ნახ. 6.15).



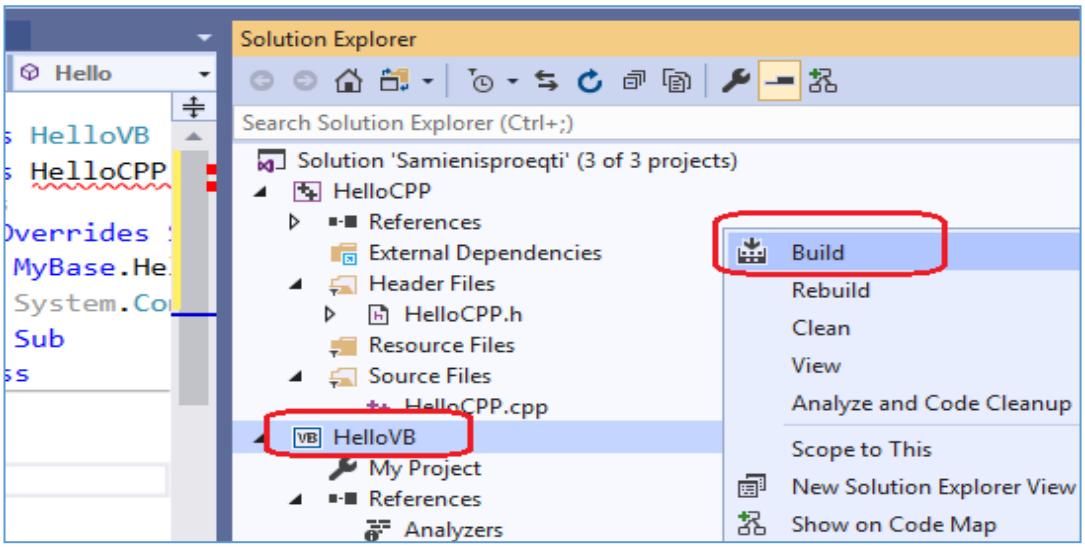
ნახ.6.15

ამის შემდეგ საჭიროა, ჯერ Solutio Explorer ის ფანჯრიდან მოინიშნოს ჩანართი My Project და გააქტიურდეს (ორჯერ დაქლიქვით) მიღებული ფანჯრის Referance - ით გაიხნება 6.16 ნახაზის მსგავსი ფანჯარა, რომელშიც Add→Reference ღილაკით ვამატებთ HelloCPP და ვადასტურებთ პროექტის დამატებას.



ნახ.6.16

ამის შემდეგ HelloVB პროექტზე მარჯვენა ღილაკით გამოვიტანოთ (ნახ. 6.17) კონტექსტური მენიუ და ავირჩიოთ Build (ეს პროექტი ტრანსლატორით ამუშავდება და სინტაქსური გამართვის შედეგს მოგვცემს).



ნახ.6.17

როგორც წესი, თუ შეცდომები არაა HelloVB პროექტში, მაშინ მის შესაბამის ფოლდერის bin->Debug -ში უნდა გამოჩნდეს დაკავშირებული Helli.CPP.dll კლასის ფაილი (ნახ.6.18).

ნახ.6.18

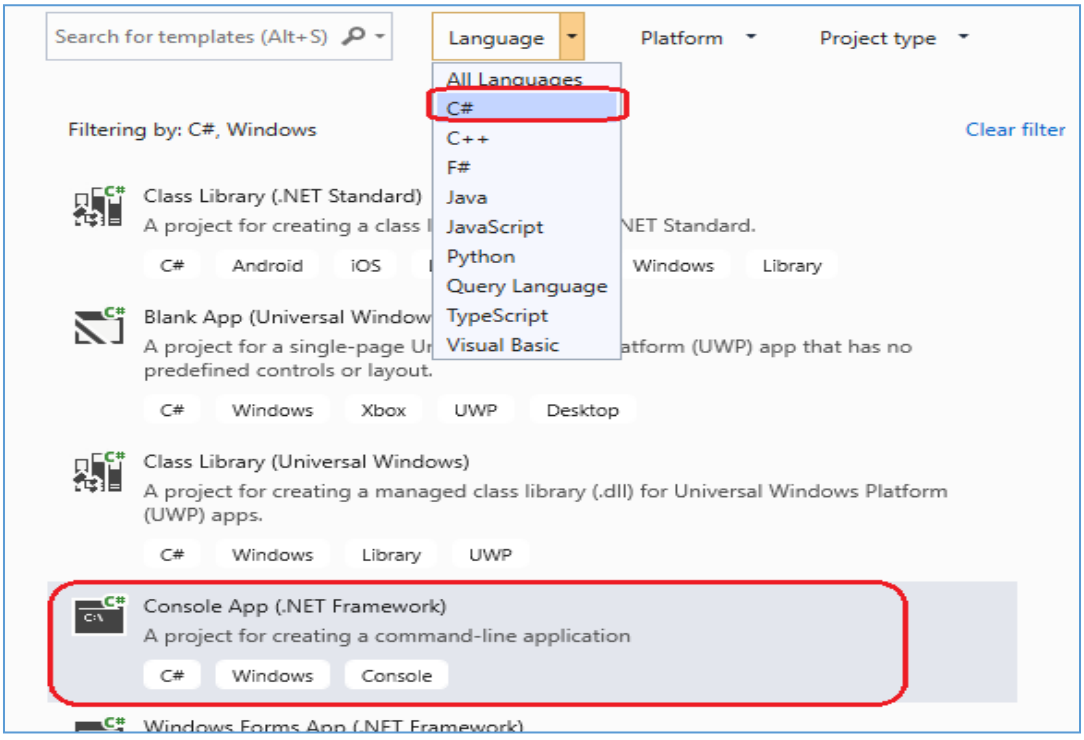
ამგვარად, Hello.CPP პროექტი მიეერთდება გადაწვეტას (Solution-ში) და გამოჩნდება მომავალში HelloVB-ში, ნახაზზე ნაჩვენები „ქვეშგახაზული“ შეცდომების აღნიშვნები HelloVB.vb პროგრამის ტექსტიდან გაქრა ! პროგრამის საბოლოო ტექსტი მოცემულია 6_3 ლისტინგში:

```
' --ლისტინგი_6.3 ----- HelloVB.vb ---------------
Public Class HelloVB
  Inherits HelloCPP.HelloCPP
  Public Overrides Sub Hello()
    MyBase.Hello()
 System.Console.WriteLine("Mec mogesalmebiT Visual Basic.NET -idan !!!")
  End Sub
End Class
```

5. სასტარტო პროექტის დამატება Solution-ში C#.NET ენაზე

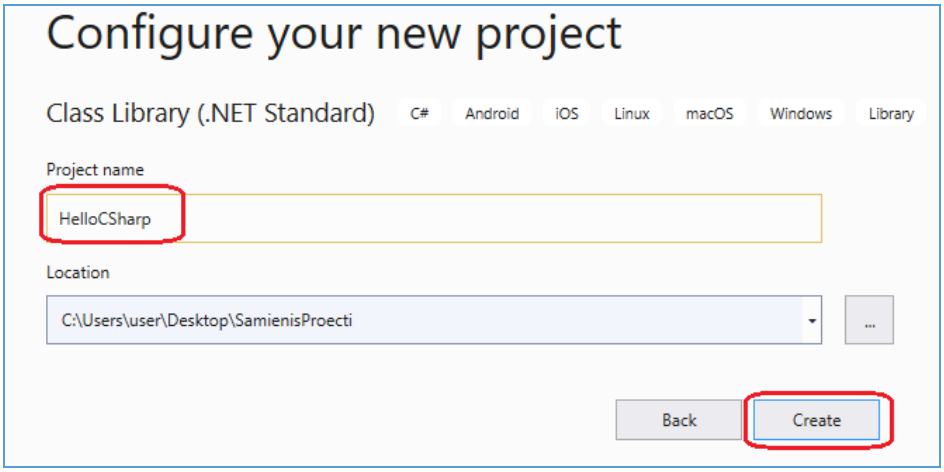
დასკვნით ფაზაში ჩვენი გადაწყვეტისათვის (Solution-ში) SamiEnisProjecti უნდა შევქმნათ მესამე, მთავარი სასტარტო პროექტი, რომელიც აგებული იქნება C#.NET ენაზე Console Application შაბლონის სახით. დავარქვათ მას HelloCSharp. ამგვარად, ვამატებთ პროექტს (ნახ.6.19-ა,ბ,გ) `HelloCSharp` სახელით.

წინა შემთხვევების მსგავსად აქაც ვირჩევთ ახალ პროექტს და ენის C# არჩევის შემდეგ კლასების ბიბლიოთეკა შეგვიძლია ავირჩიოთ როგორც უახლესი მოდელი - Universal Windows, ასევე - Console Application (ჩვენს შემთხვევაში ვირჩევთ მეორეს - ა).



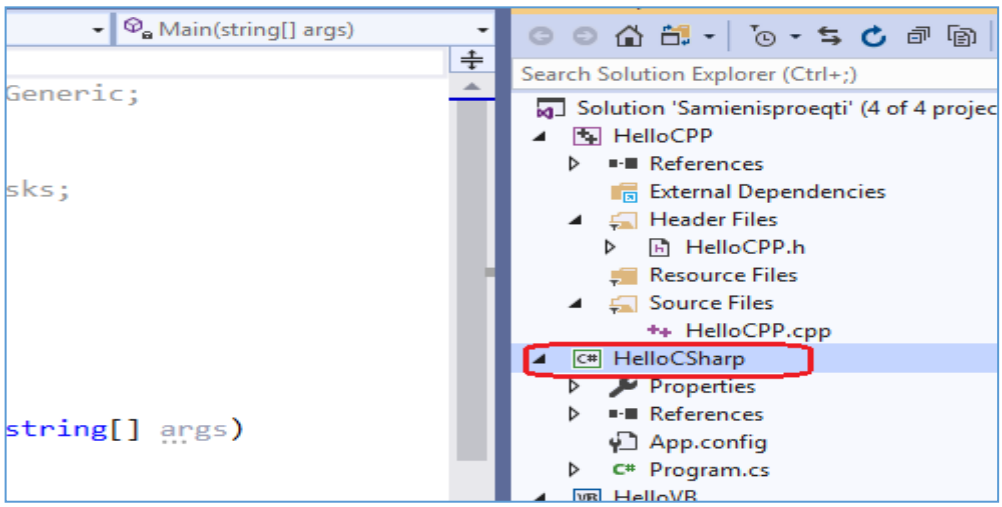
ნახ.6.19-ა

შედეგად მიიღება ფანჯარა, სადაც პროექტს ვარქმევთ სახელს HelloCSharp და ვადასტურებთ Create ღილაკით ნახ.6.19-ბ:



ნახ. 6.19-ბ

Solution - ფანჯარას დაემატება ახალი HelloCSharp პროექტი, თავისი შემადგენელი ყველა კომპონენტით (ნახ.6.19-გ).

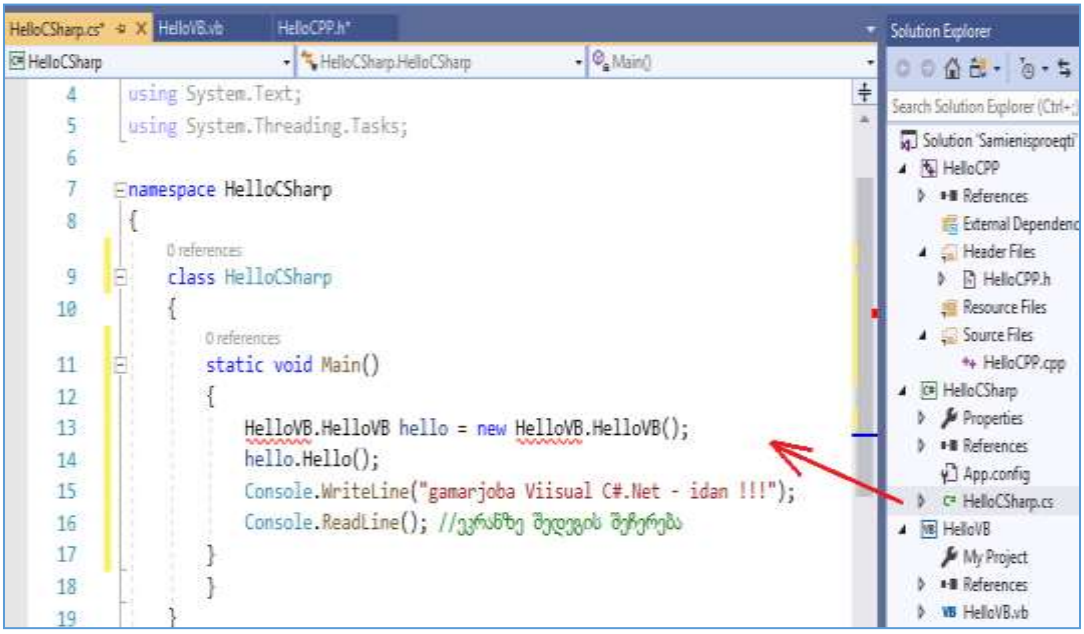


ნახ.6.19-გ

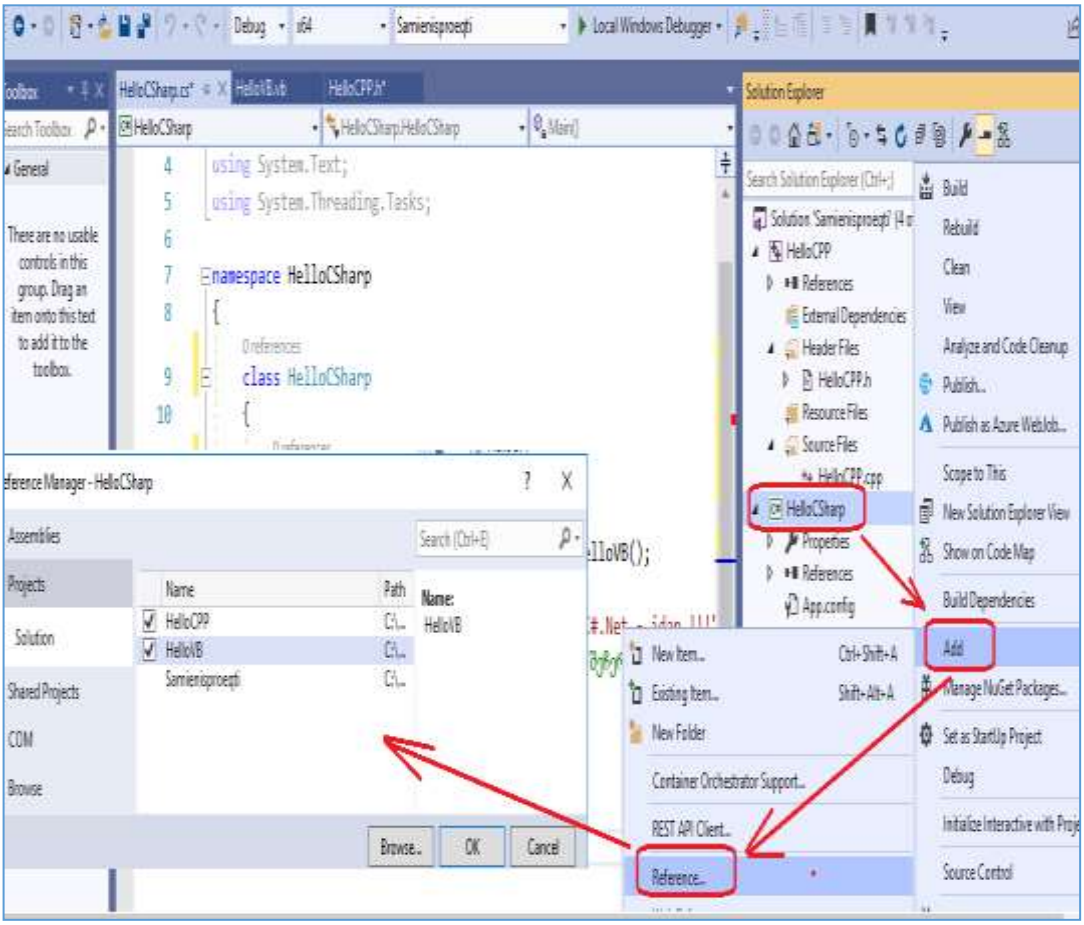
გადავარქვათ სახელი Program.cs ფაილს ჩვენი პროექტის შინაარსის შესაბამისად: HelloCSharp.cs. გამოვიტანოთ ეს ფაილი რედაქტირების ფანჯარაში და ჩავამატოთ Main()-ში შესაბამისი სტრიქონები. ასევე შეიძლება Main() -ის არგუმენტების წაშლა, აქ არ გვჭირდება. რედაქტირების შემდეგ ტექსტი მოცემულია 6.20 ნახაზზე.

პროგრამაში Main() სტატიკური მეთოდი არის აპლიკაციაში შესვლის წერტილი. ეს მეთოდი ქმნის HelloVB კლასის ახალ ობიექტს hello და მისთვის იძახებს Hello() მეთოდს, რომელშიც ინახება ორი შეტყობინება. ერთი CPP.NET-იდან, მეორე VB.BET- დან, რომლებიც ადრე მოვამზადეთ. მესამე შეტყობინებას თვით C#.NET გამოიტანს, დამშვიდობებასთან ერთად. ახლა საჭიროა HelloCSharp პროექტში ჩავამატოთ მიმთითებლები (კავშირები) HelloCPP და HelloVB პროექტებზე. მაშინ 6.20 ნახაზზე ნაჩვენები შეცდომები (ქვეშახაზგასმული HelloVB) გასწორდება.

ამისათვის ვირჩევთ Reference…-ს და ნახაზზე მოყვანილი ნიმუშის შესაბამისად ვააქტიურებთ მითითებებს (ნახ.6.21).

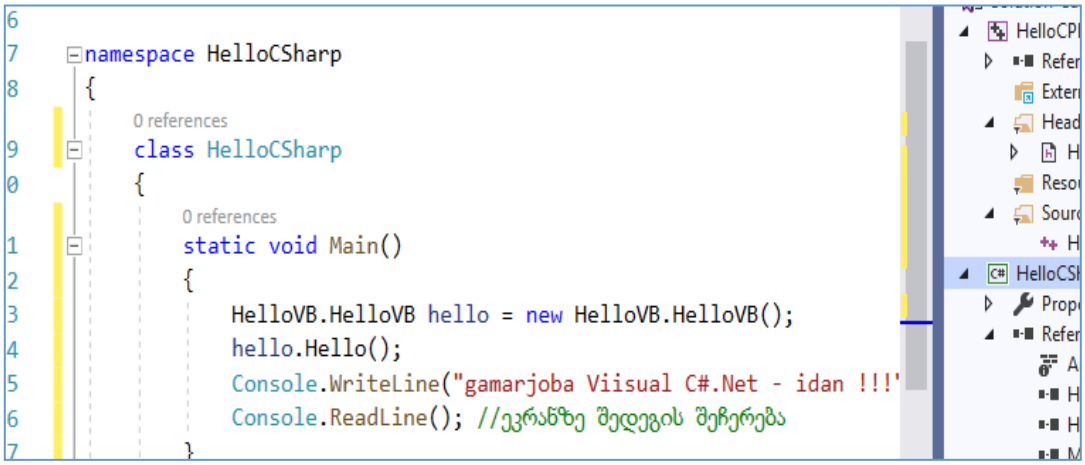


ნახ.6.20



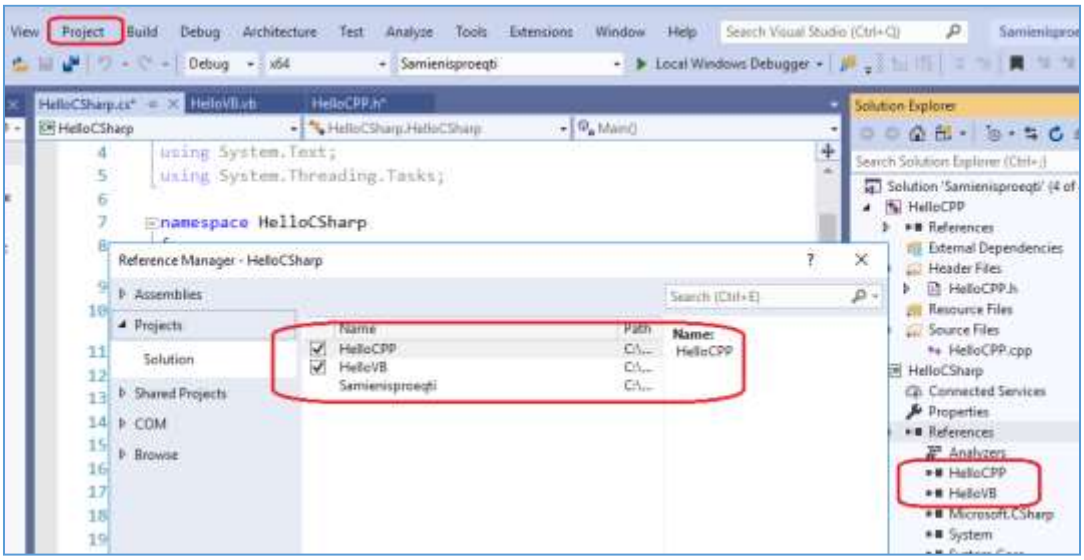
ნახ.6.21

HelloCPP და HelloVB პროექტებთან კავშირების შემდეგ, პროგრამულ კოდში სინტაქსური შეცდომის აღმნიშვნელი ინდიკატორები გაქრება ნახ.6.22



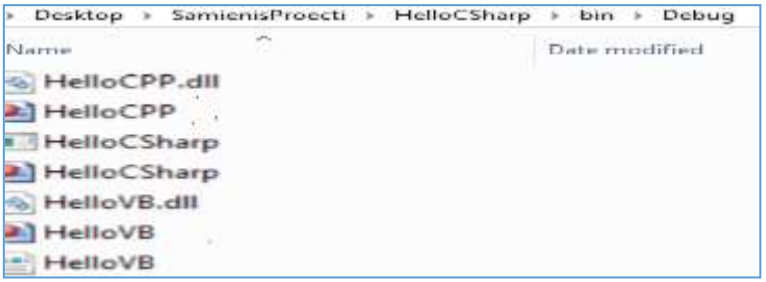
ნახ.6.22

Project-ჩანართიდან ,Add Reference ფანჯარაში გადასვლის შემდეგ მონიშნული HelloCPP და HelloVB პროექტები მიუთითებს იმაზე, რომ კავშირი პროექტებს შორის წარმატებით შესრულდა. ეს შედეგი აისახება Solution- ფანჯარაში HelloCSharp პროექტის References –ში (ნახ.6.23).



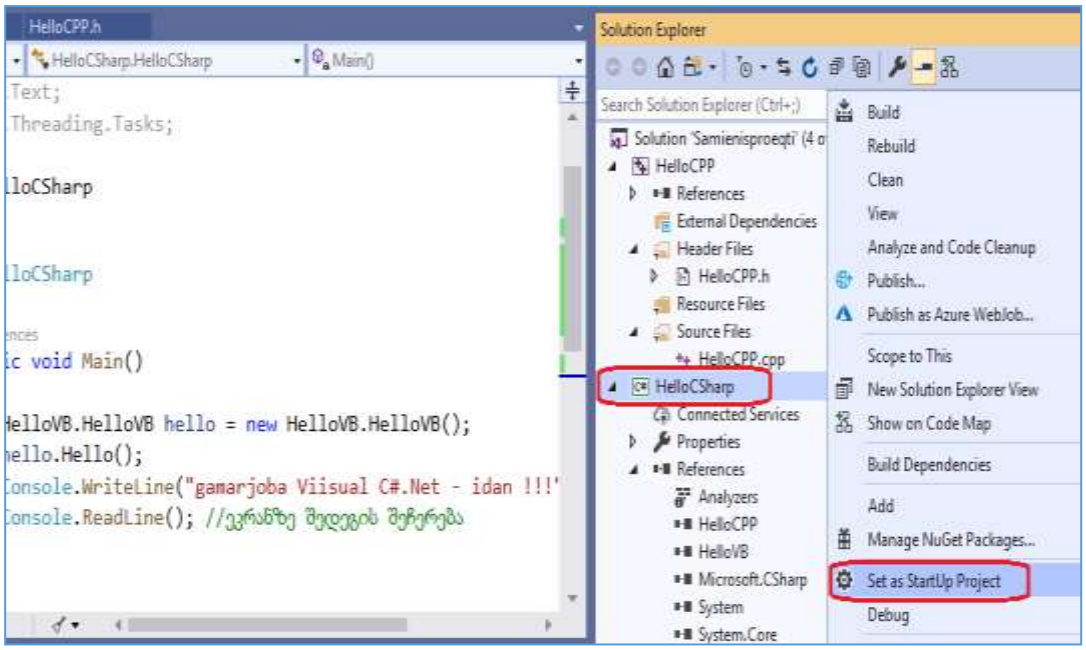
ნახ.6.23

ავამუშავოთ HelloCSharp პროექტი მარჯვენა ღილაკით და build-ით, რათა სინტაქსურად დამუშავდეს იგი. შეცდომების არარსებობის შემთხვევაში მოხდება HelloCPP.dll და HelloVB.dll ფაილების განთავსება HelloCSharp პროექტის bin-> Debug ფოლდერში (ნახ.6.24). აქვეა HelloCSharp.exe აპლიკაციის ფაილიც.



ნახ.6.24

Solution-ში HelloCSharp პროექტი გადავაკეთოთ სასტარტო ფაილად (ნახ.6.25).



ნახ.6.25

ბოლოს, ავამუშავოთ აპლიკაცია და მივიღებთ შედეგს (ნახ.6.26).

ნახ.6.26

რეზიუმე:

აპლიკაციის სხვადასხვა პროექტები დამუშავდა დაპროგრამების სხვადასხვა ენებზე, რომლებიც გამოიყენება .NET გარემოში. მრავალჯერადი გამოყენების ფაილები შემუშავებულ იქნა კლასების დახმარებით და რეალიზებულია დინამიკური .dll - ფაილების სახით.

დავალება:

ააგეთ განხილული პროგრამული პროექტის კოდი და გააანალიზეთ მისი მუშაობის შედეგები..

6.1.2. ლაბ-N2: კლასებისა და ობიექტების დაპროგრამება მართვის ამოცანის მაგალითზე (კონსოლის რეჟიმი)

მიზანი: კონსოლური აპლიკაციების (Console App) პროექტში კლასების, მათი ობიექტების, მეთოდების და კლასთაშორის კავშირების დაპროგრამების შესწავლა (გამეორება, ვისაც არ უსწავლია ან არ ახსოვს). ობიექტორიენტირებული დაპროგრამების ინკაფსულაციის თვისების დეტალიზებული შესწავლა. კლასის განსაზღვრა როგორც მონაცემებისა და მეთოდების ერთობლიობა და მათი პროგრამული რეალიზაციის განხორციელება [7].

განვიხილოთ აღნიშნული საკითხები პრაქტიკული მართვის ამოცანის საფუძველზე, მათი შემდგომი განზოგადების მიზნით.

ამოცანა_1: ავაგოთ ობიექტორიენტირებული პროგრამის კოდი (კლასებისა და მეთოდების გამოყენებით) მაღლივ შენობაში ლიფტის გადაადგილების პროცესის მოდელირებისათვის.

მაგალითად, შენობა N სართულიანია. მას აქვს ლიფტი. პერსონა, რომელიც შედის ლიფტში (ხდება მგზავრი), ირჩევს მისთვის საჭირო სართულს და მიემგზავრება (ზევით ან ქვევით). საჭიროა ამ პროცესის მოდელირება და დაპროგრამება ისე, რომ დაფიქსირდეს ლიფტის საწყისი მდგომარეობა, ამუშავების შემდეგ მისი საბოლოო მდგომარეობა, გავლილი სართულების რაოდენობა (ერთი ამუშავებისას). საბოლოოდ გამოიცეს რეპორტი, თუ სულ რამდენი სართული გაიარა ლიფტმა ერთი სეანსის (მაგალითად, 24 საათის) განმავლობაში.

2.1) კლასთა მოდელი და მათი აგების პირობები:

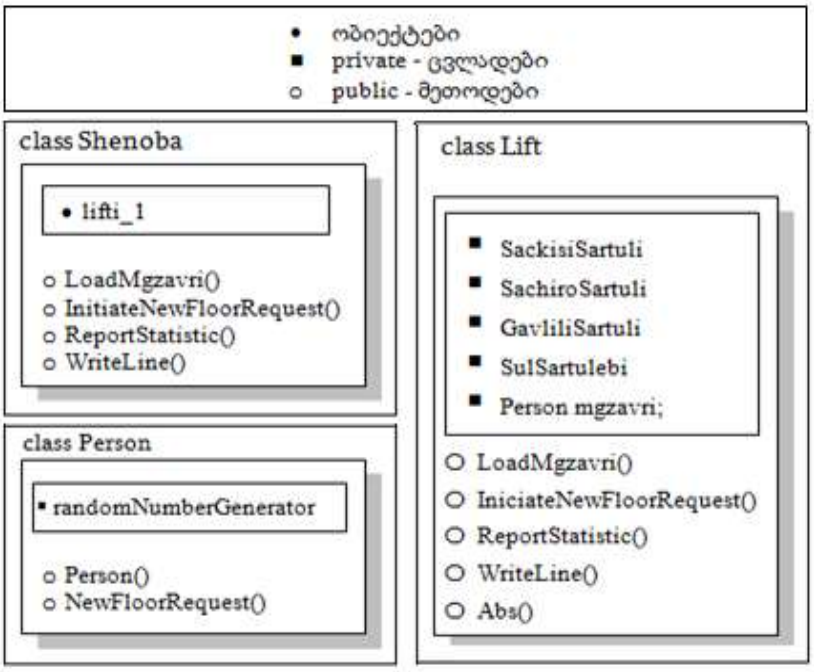
ინკაფსულაციის სქემა მოცემულია 6.27 ნახაზზე.

– პროგრამაში სამი კლასია: Shenoba, Lift და Person;

– Shenoba კლასს აქვს Lift კლასის ერთი ობიექტი, სახელით lifti_1 ;

– Person კლასის ერთი ობიექტი იმყოფება mgzavri – ცვლადის შიგნით, რომელიც იყენებს lifti_1 -ს;

– Lift კლასის ობიექტს შეუძლია გადაადგილება ნებისმიერ სართულზე, ინტერვალში [1,N=60].



ნახ.6.27

– პროგრამაში სართულის არჩევა ხდება მგზავრის მიერ (გამოიყენება „შემთხვევით რიცხვთა გენერატორი“ Random-ფუნქციით);

– lifti_1 ლიფტი მოძრაობს საჭირო სართულისაკენ, რაც აისახება კონსოლზე;

– მგზავრ(ებ)ის ლიფტით მოძრაობის სეანსი შედგება რამდენიმე ეტაპისაგან, რომლბზეც შეირჩევა ახალი მიზნობრივი სართულები;

– სეანსის ბოლოს გამოიცემა ანგარიშის ტექსტი (რეპორტი), თუ ჯამში რამდენი სართული გაიარა ლიფტმა;

– შუალედური და საბოლოო შედეგები გამოიტანება ეკრანზე.

2.2) კლასთა კავშირების აღწერა UML ტექნოლოგიით:

მომხმარებლის სამი კლასი: Shenoba, Lift, Person და ერთი სისტემური კლასი System.Random ამოდელირებს ლიფტის მუშაობის პროცესს:

– Shenoba-კლასი შეიცავს Lift-ობიექტსა და იძახებს მის მეთოდებს;

– Lift-ობიექტი იყენებს Person-ობიექტს, რათა მართოს ლიფტის მოძრაობა;

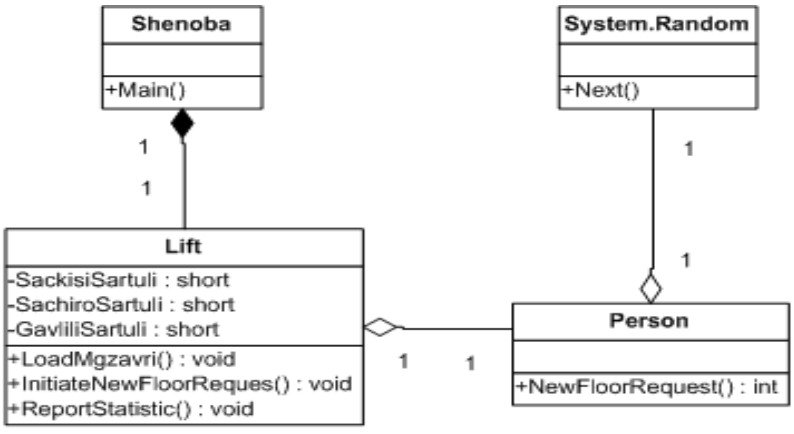
– Person-ობიექტი კი იყენებს System.Random-ობიექტს, რათა აირჩიოს საჭირო სართული შემდგომი გადაადგილებისათვის.

ობიექტორიენტირებული პროგრამული სისტემების დაპროექტებისას კლასებმა მსგავსი ფუნქციები უნდა შეასრულოს. თითოეულს უნდა ჰქონდეს თავისი უნიკალური ფუნქციონალობა და ყველა ერთად უნდა უზრუნველყოფდეს მთლიანი პროგრამის მუშაობას.

ნებისმიერი მაღლივი შენობა, როგორც „მთელი“ შედგება სხვადასხვა ნაწილისაგან: იატაკი, კედლები, ფანჯრები, ლიფტები და ა.შ. ამგვარად, შენობასა და ლიფტს შორის არსებობს დამოკიდებულება „მთელი-ნაწილი“ (აგრეგატული კავშირი UML ენაზე). ამიტომაცაა, რომ Lift-კლასის ობიექტის ცვლადი გამოცხადებულია Shenoba-კლასის შიგნით (ნახ.6.27, lifti_1). პროგრამულად იგი რეალიზებულია სტატიკური ცვლადის სახით (47-ე სტრიქონი, ლისტინგი_2.1): private static Lift lifti_1;

იგი ინახავს Lift კლასის ობიექტსა და შეუძლია მისი მეთოდების გამოძახება. გარდა ამისა, Shenoba-კლასს შეიძლება ჰქონდეს აგრეთვე სხვა ცვლადებიც, რომლებიც აგრეგატულად დაქვემდებარებული კლასების ობიექტების ცვლადები იქნება.

6.28 ნახაზზე მოცემულია ჩვენი მაგალითის კლასებს შორის კავშირების UML დიაგრამა, აგებული Ms Visio პაკეტის გარემოში [9,12].

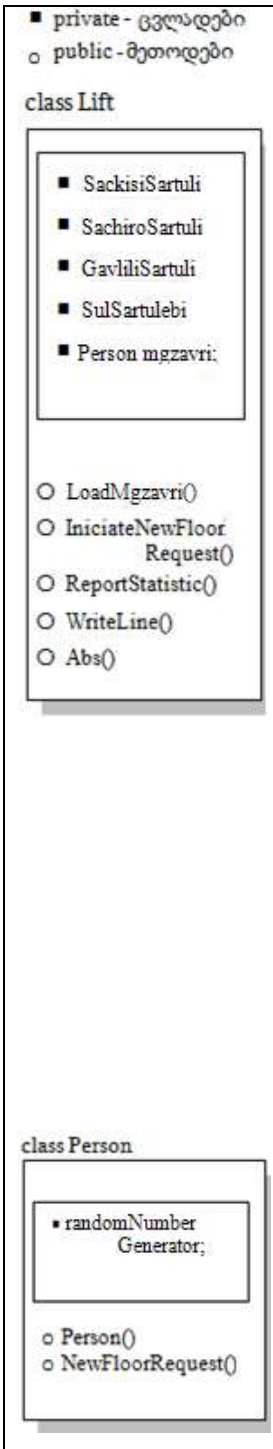


ნახ.6.28. UML-ის კლასების დიაგრამა

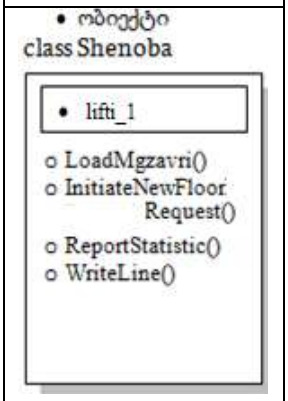
Shenoba და Lift კლასებს შორის აგრეგაციას უწოდებენ კომპოზიციასა და იგი შავი რომბიკით გამოისახება. „1“-ები ნიშნავს, რომ 1 შენობაში არის 1 ლიფტი (ჩვენს შემთხვევაში). Lift და Person, აგრეთვე Person და System.Random კლასებს შორის აგრეგატული დამოკიდებულებაა, მაგრამ არა-კომპოზიციური. ის პატარა რომბითაა ნაჩვენები. განსხვავება ისაა, რომ შენობას ლიფტი ყოველთვის აქვს. ლიფტში კი პერსონა შეიძლება იყოს ან არ იყოს, ლიფტი ისეც მუშაობს.

2.3) პროგრამული რეალიზაცია:

2.1 ლისტინგში მოცემულია განხილული კლასების პროგრამული რეალიზაციის C# კოდის ტექსტი.




```
1  using System;
2  namespace ConsoleApp_Lift
3  {
4   class Lift
5   {
6     private int SackisiSartuli = 1;
7     private int SachiroSartuli = 0;
8     private int GavliliSartuli = 0;
9     private int SulSartulebi = 0;
10    public  int i = 1;
11    private Person mgzavri;
12   public void LoadMgzavri()
13     {
14       mgzavri = new Person();
15     }
16   public void InitiateNewFloorRequest()
17   {
18    SachiroSartuli = mgzavri.NewFloorRequest();
19    GavliliSartuli = Math.Abs(SackisiSartuli -
                                    SachiroSartuli);
20    Console.WriteLine("{0,2}.| Sackisi: {1,2}|
        Sachiro: {2,2}| Gavlili: {3,2} ",  i.ToString(),
        SackisiSartuli.ToString(),
        SachiroSartuli.ToString(),
        GavliliSartuli.ToString());
21   // Math.Abs(SackisiSartuli - SachiroSartuli);
22    SulSartulebi += GavliliSartuli;
23     SackisiSartuli = SachiroSartuli;
24     i++;
25   }
26   public void ReportStatistic()
27   {   Console.WriteLine("\n===>>> Sul gavlili
28                          sartulebi: " + SulSartulebi);
29     }
30    }
31
32  class Person
33   {
       private System.Random
34                    randomNumberGenerator;
35    public Person()   // კონსტრუქტორი
36     {
        randomNumberGenerator = new
37                              System.Random();
38      }
39
```




```
40      public int NewFloorRequest()
41      {
          // აბრუნებს არჩეულ შემთხვევით
42        //  რიცხვს 1-60 დიაპაზონში
43        return randomNumberGenerator.Next(1,60);
44      }
    }
45  class Shenoba   // კლასი შენობა
46  {
47    private static Lift lifti_1;
48    private static void Main()
      {
49      lifti_1 = new Lift();
50      lifti_1.LoadMgzavri();
51      Console.WriteLine("      samgzavro sartulebi \n
52      ============================\n");
53      lifti_1.InitiateNewFloorRequest();
54      lifti_1.InitiateNewFloorRequest();
55      lifti_1.InitiateNewFloorRequest();
56      lifti_1.InitiateNewFloorRequest();
57      lifti_1.InitiateNewFloorRequest();
58      lifti_1.ReportStatistic();
59    }
60  }}
```

ლისტინგი_2.1: პროგრამა Console App კონკრეტული მართვის ამოცანისათვის

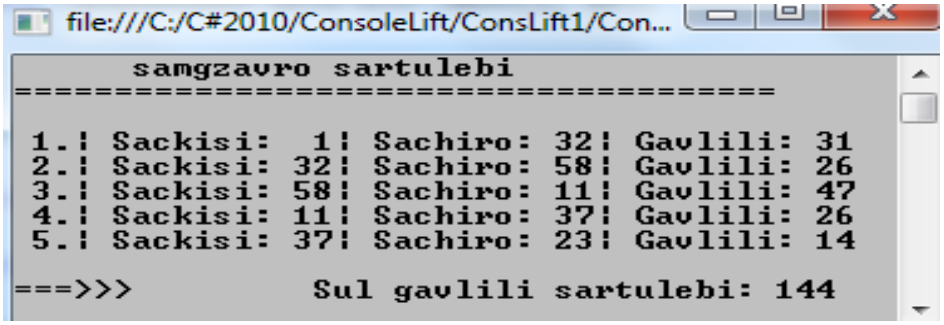
2.4) პროგრამის ლისტინგის ანალიზი (საყურადღებო სტრიქონები)

N	დანიშნულება
4	Lift - კლასის განსაზღვრების დასაწყისი
6	SackisiSartuli - ცვლადის გამოცხადება **int**-ტიპით, private-წვდომის სპეციფიკატორით და საწყისი მნიშვნელობით=1
11	mgzavri - ცვლადის გამოცხადება, რომელიც შეიცავს Person- კლასის ობიექტს. Person- კლასი ასრულებს მგზავრის როლს Lift-კლასთან მიმართებით
12	LoadMgzavri() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
14	Person-კლასის ახალი (**new**) ობიექტის შექმნა. ეს ობიექტი მიენიჭება ცვლადს - mgzavri
16	InitiateNewFloorRequest() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
18	mgzavri-ობიექტისთვის NewFloorRequest() მეთოდის გამოძახება. ამ მეთოდით დაბრუნებული მნიშვნელობა მიენიჭება ცვლადს SachiroSartuli
19	ერთი მგზავრობის შემდეგ იანგარიშება გავლილი სართულების რაოდენობა
20	ეკრანზე გამოიტანება: საწყისი_სართული, საჭირო_სართული, გავლილი_სართულების_რაოდენობა
21	იანგარიშება სულ გავლილი სართულების რაოდენობა ლიფტის მუშაობის მთელი სეანსის დროს
22	ლიფტის საწყისი სართულის მნიშვნელობას ენიჭება მისი ბოლო გაჩერების სართულის მნიშვნელობა
23	ლიფტის ამუშავების მომდევნო ბიჯის ინკრემენტი

26	ReportStatistic() მეთოდის გამოძახებით ეკრანზე გამოიტანება სტატისტიკა SulSartulebi ცვლადით
32	Person კლასის განსაზღვრების დასაწყისი
33	randomNumberGenerator ცვლადის გამოცხადება System.Random კლასის ობიექტის შესანახად
34	სპეციალური მეთოდის (კონსტრუქტორის !) განსაზღვრების დასაწყისი, რომელიც გამოიძახება ავტომატურად Person კლასის ობიექტის შექმნის დროს
36	System.Random-კლასის ახალი ობიექტის შექმნა და მისი მინიჭება randomNumberGenerator - ცვლადზე
39	int ტიპის NewFloorRequest() მეთოდის განსაზღვრა. ის Person-კლასის ინტერფეისის ნაწილია
42	პერსონა (ვირტუალური მგზავრი) ლიფტში ირჩევს საჭირო სართულს (შემთხვევით რიცხვთა გენერატორი ასრულებს ამ ფუნქციას) დიაპაზონში [1-60]
45	Shenoba კლასის აღწერა
47	Shenoba კლასში გამოცხადებულია Lifti ტიპის ცვლადი Lifti_1. Shenoba კლასი კომპოზიციურ კავშირშია Lift კლასთან (ნახ.3.2)
48	Main( ) მეთოდის აღწერის დასაწყისი
49	Lifti კლასის ახალი ობიექტის შექმნა და მისი მინიჭება lifti_1 ცვლადზე
50	lifti_1 ობიექტისთვის LoadMgzavri( ) მეთოდის გამოძახება
51-57	lifti_1 ობიექტისთვის .IniciateNewFloorRequest( ) მეთოდის გამოძახება 5-ჯერ
58	lifti_1 ობიექტისთვის . ReportStatistic() მეთოდის გამოძახება (შედეგების გამოსაცემად ეკრანზე)

2.5) პროგრამის მუშაობის შედეგები:

6.29 ნახაზზე ნაჩვენებია განხილული პროგრამის შესრულების შედეგები კონსოლის რეჟიმში.

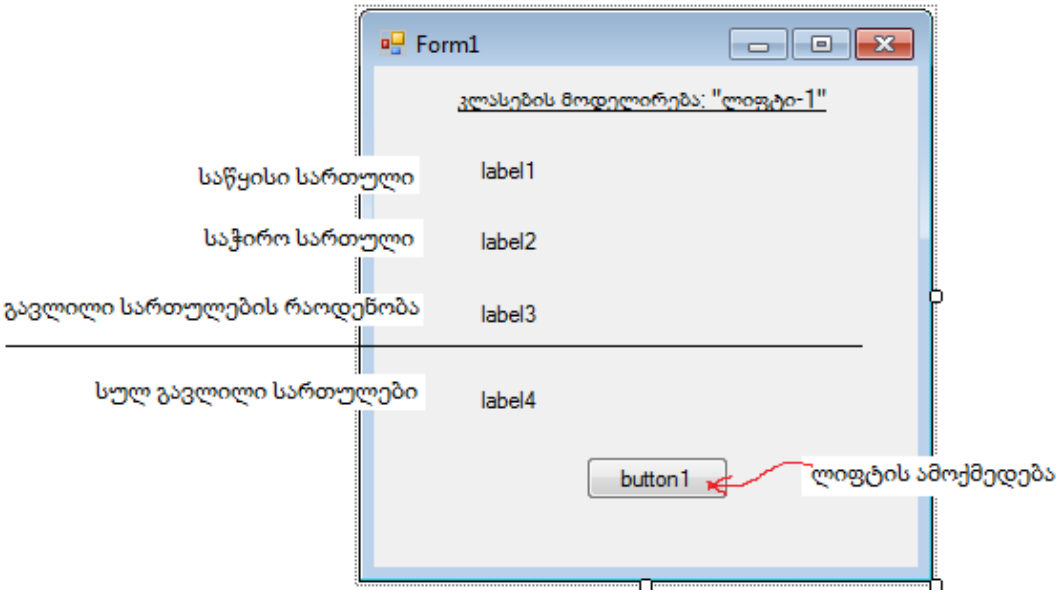


ნახ.6.29. Console App პროგრამის მუშაობის შედეგი

6.2. აპლიკაციის დაპროგრამება ვინ-ფორმის რეჟიმში (Windows Forms App)

განვიხილოთ წინა პარაგრაფში დასმული ამოცანის (ლიფტების მოძრაობის პროცესის მოდელირება და მართვა) Windows Forms Application რეჟიმში.

ამოცანა_2: ავაგოთ პროგრამული კოდი კლასების საფუძველზე ვინდოუსის ფორმის რეჟიმში. 6.30 ნახაზზე ნაჩვენებია სამუშაო ფანჯარა, რომელიც Form კლასის Form1 ობიექტია. მასზე განთავსებულია ოთხი label (1,2,3,4) და ერთი button1, რომლითაც მოდელირდება ლიფტის ამოქმედება (ლიფტის გამოძახება, როცა პერსონა გარეთაა ან სართულის არჩევა ლიფტის ღილაკით, როცა პერსონა ლიფტშია, ანუ მგზავრია).



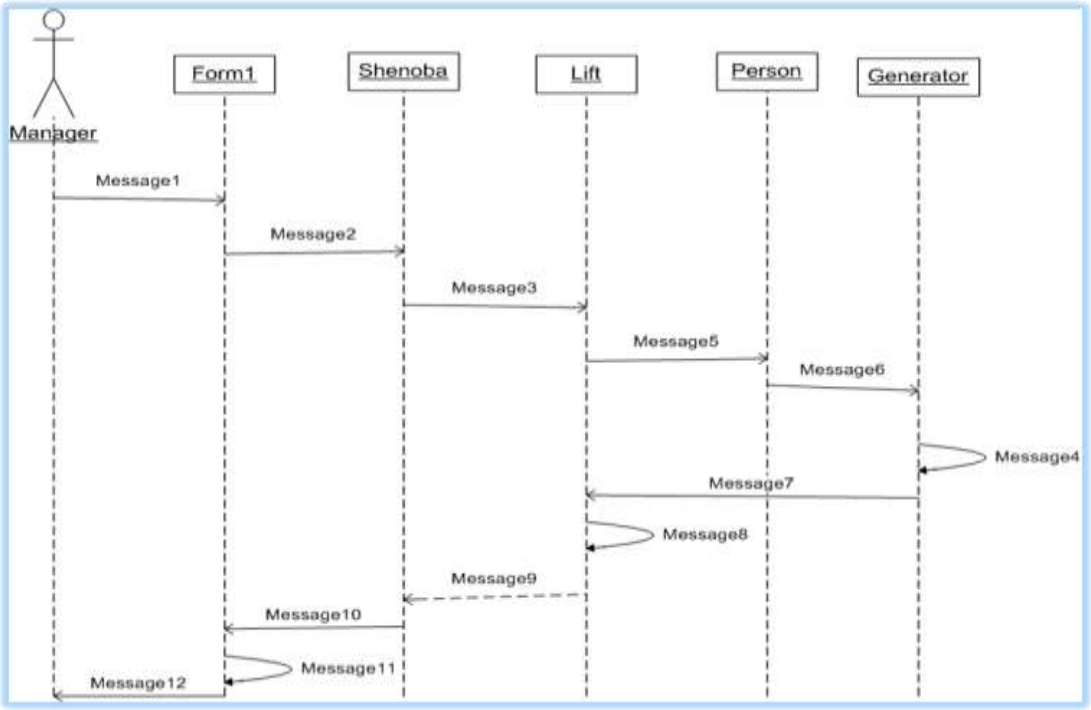
ნახ.6.30. ინტერფეისის ვიზუალური კომპონენტები

2.2 ლისტინგში მოცემულია ინტერფეისის ღილაკის მოქმედების კოდის ტექსტი.

```
// ლისტინგი_2.2--Form1  ელემენტებით და  button1-ის კოდი ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormLift
{
   public partial class Form1 : Form
   {
      Shenoba shenoba_1;    // კლასი Shenoba  უნდა შეიქმნას
      public Form1() { InitializeComponent();   }
      private void button1_Click(object sender, EventArgs e)
      {   // shenoba_1  ობიექტის შექმნა
         shenoba_1 = new Shenoba(label1, label2, label3, label4);
      }
   }
}
```

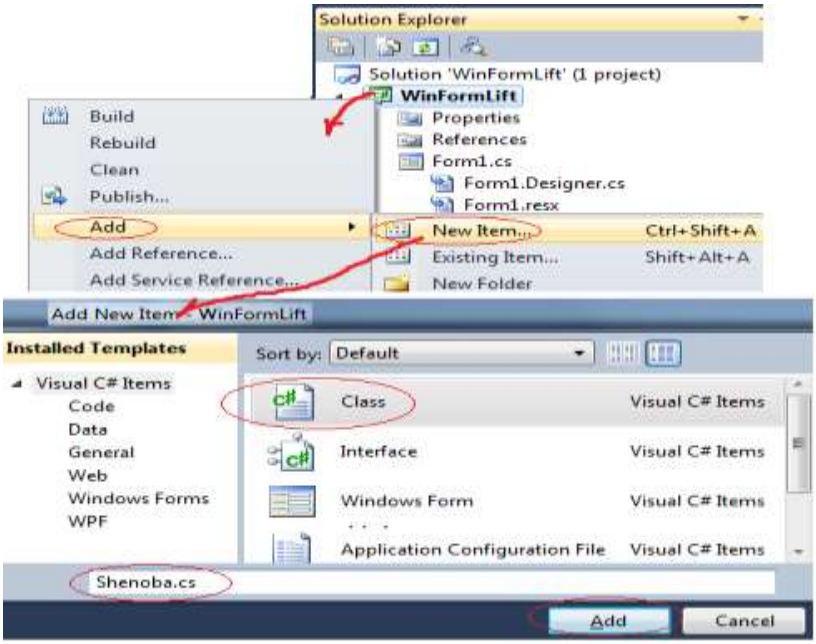
გარდა Form კლასისა (Form1 ობიექტით), რომელიც ასრულებს პროგრამის მომხმარებლის ინტერფეისის ფუნქციას, ლიფტის მუშაობის პროცესის ობიექტორიენტირებული მოდელის ასაგებად საჭიროა კლასები: Shenoba, Lift და Person.

ამ კლასების თვისებები და ფუნქციობა ჩვენ ზემოთ აღვწერეთ. ახლა საჭიროა ავაგოთ სცენარი „ლიფტის მუშაობა“, რომლისთვისაც გამოვიყენებთ UML-ის მიმდევრობითობის (Sequence) დიაგრამას (ნახ.6.31).



ნახ.6.31. UML-ის მიმდევრობითობის დიაგრამა

შევქმნათ დანარჩენი კლასები და აღვწეროთ მათი ფუნქციობები. Solution Explorer-ში დავამატოთ (Add new Items) ახალი კლასები, როგორც ეს 6.32 ნახაზზეა ნაჩვენები.



ნახ.6.32

Shenoba კლასის კოდი მოცემულია 5.3_ ლისტინგში.

```
// ლისტინგი_2.3 --- კლასი Shenoba --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{
  public class Shenoba
 {
   static Lift lift_1 = new Lift();

 public Shenoba(Label label1, Label label2, Label
                                              label3, Label label4)
{
 lift_1.LoadMgzavri();
  lift_1.InitiateNewFloorRequest(label1,label2, label3);
 lift_1.ReportStatistic(label4);
}
}
}
```

using System; Lift კლასის პროგრამული კოდი მოცემულია 5.4_ლისტინგში.

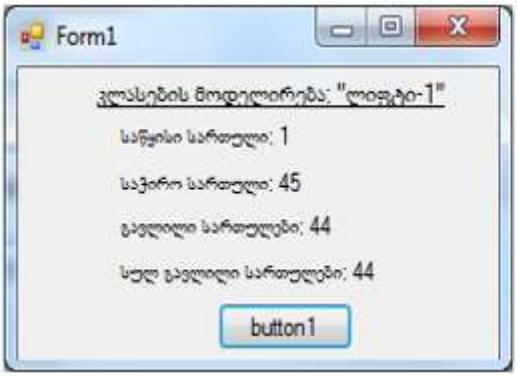
```
// ლისტინგი_2.4 --- კლასი Lift  --------
using System.Windows.Forms;
namespace WinFormLift
{
 public class Lift
  {
    private int SackisiSartuli = 1;
    private int SachiroSartuli = 0;
    private int GavlilSartulebi = 0;
    private int SulSartulebi = 0;
    public int i;
    private Person mgzavri;

  public void LoadMgzavri()
    {
      mgzavri = new Person();
    }
 public void InitiateNewFloorRequest(Label label1,
                                        Label label2, Label label3)
   {
    SachiroSartuli = mgzavri.NewFloorRequest();
    GavlilSartulebi = Math.Abs(SackisiSartuli - SachiroSartuli);
    label1.Text = "საწყისი სართული: " + SackisiSartuli.ToString();
    label2.Text = "საჭირო სართული: " + SachiroSartuli.ToString();
    label3.Text = "გავლილი სართულები: " + GavlilSartulebi.ToString();
    SulSartulebi += GavlilSartulebi;
```

```
    SackisiSartuli = SachiroSartuli;
    i++;
   }
 public void ReportStatistic(Label label4)
  {
   label4.Text = "სულ გავლილი სართულები: " + SulSartulebi;
} } }
```

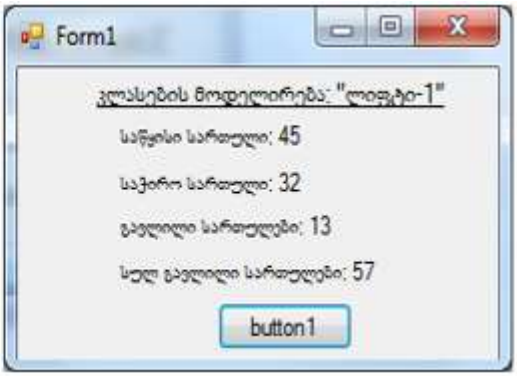
Person კლასის პროგრამა მოცემულია 5.5_ლისტინგში.

```
// ლისტინგი_2.5 --- კლასი Person  --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{  public class Person
  {  private System.Random randomNumberGenerator;
     public Person()    // კონსტრუქტორი
      {
       randomNumberGenerator = new System.Random();  }
  public int NewFloorRequest()
   {  return randomNumberGenerator.Next(1, 60);
} } }
```

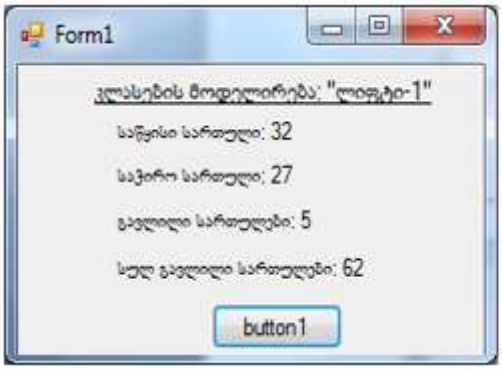
პროგრამის ამუშავების შემდეგ მიიღება 6.34 ნახაზზე ნაჩვენები შედეგები.



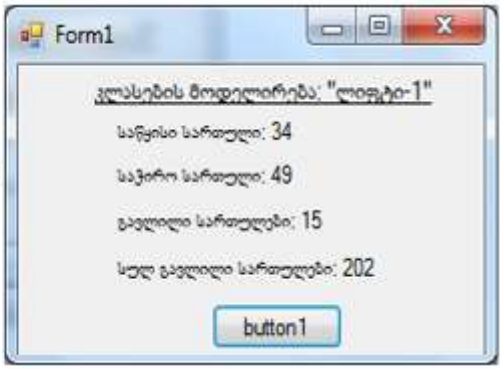
ნახ.6.33-ა. 1-ელი ბიჯი



ნახ.6.33-ბ. მე-2 ბიჯი



ნახ.6.33.-გ. მე-3 ბიჯი



ნახ.6.33-დ. მე-10 ბიჯი,

და ა.შ.

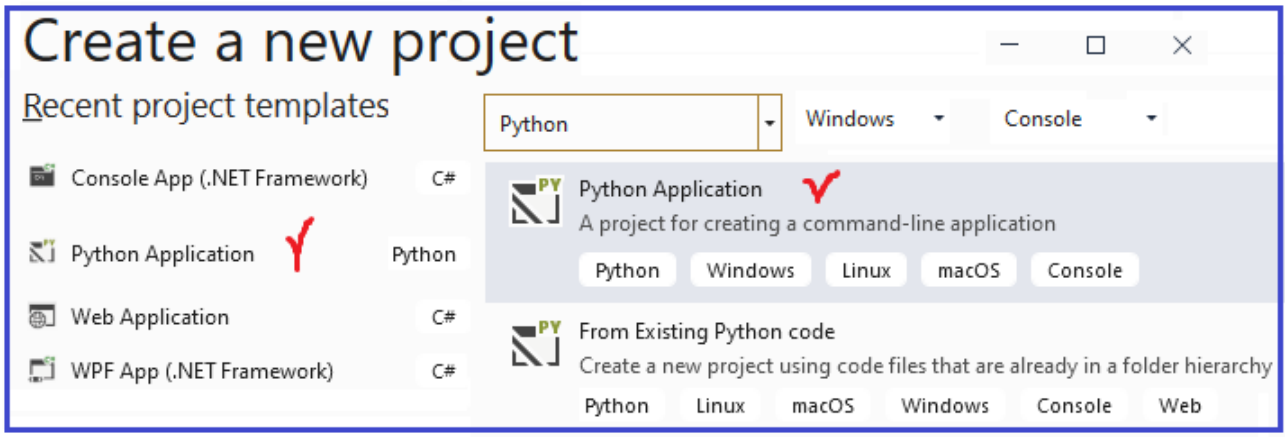
6.3. ლაბ-N4: მართვის ამოცანის აპლიკაციის დაპროგრამება Python ენაზე (Console App)

განვიხილოთ ლიფტის მართვის ამოცანის აპლიკაციის აგების პროცესი დაპროგრამების Python ენაზე (კონსოლის რეჟიმში) [8].

ამოცანის დასმა, პირობები და UML-მოდელირების და კონსოლის აპლიკაციის დაპროგრამების ელემენტები მსგავსია მე-2 ლაბორატორიისა, სადაც ვიყენებდით C#.NET ენას.

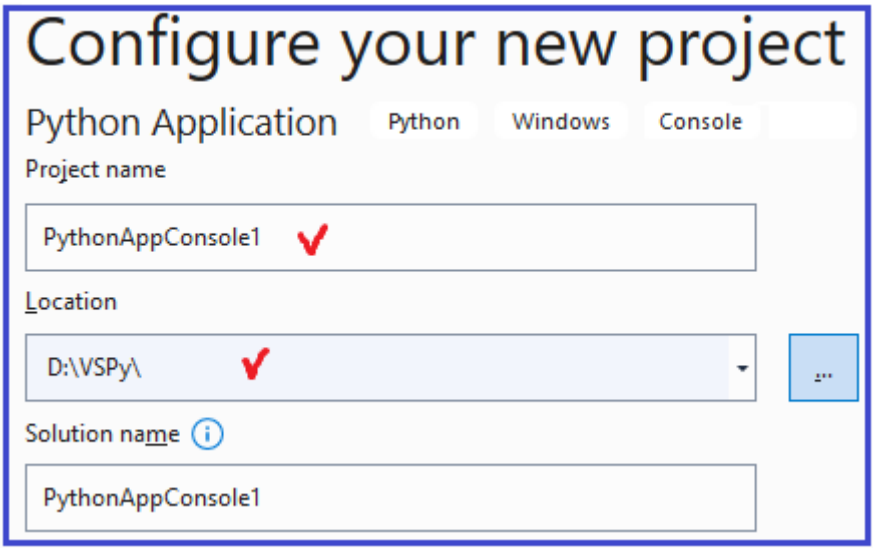
ახლა ავაგოთ კონსოლის აპლიკაცია Visual Studio.NET პლატფორმაზე Python ენის საფუძველზე და შევადაროთ პროგრამული კოდები და ფუნქციონირების პარამეტრები ერთმანეთს.

შევქმნათ ახალი პროექტი Visual Studio.NET-ში Python App-ით (ნახ.6.34).



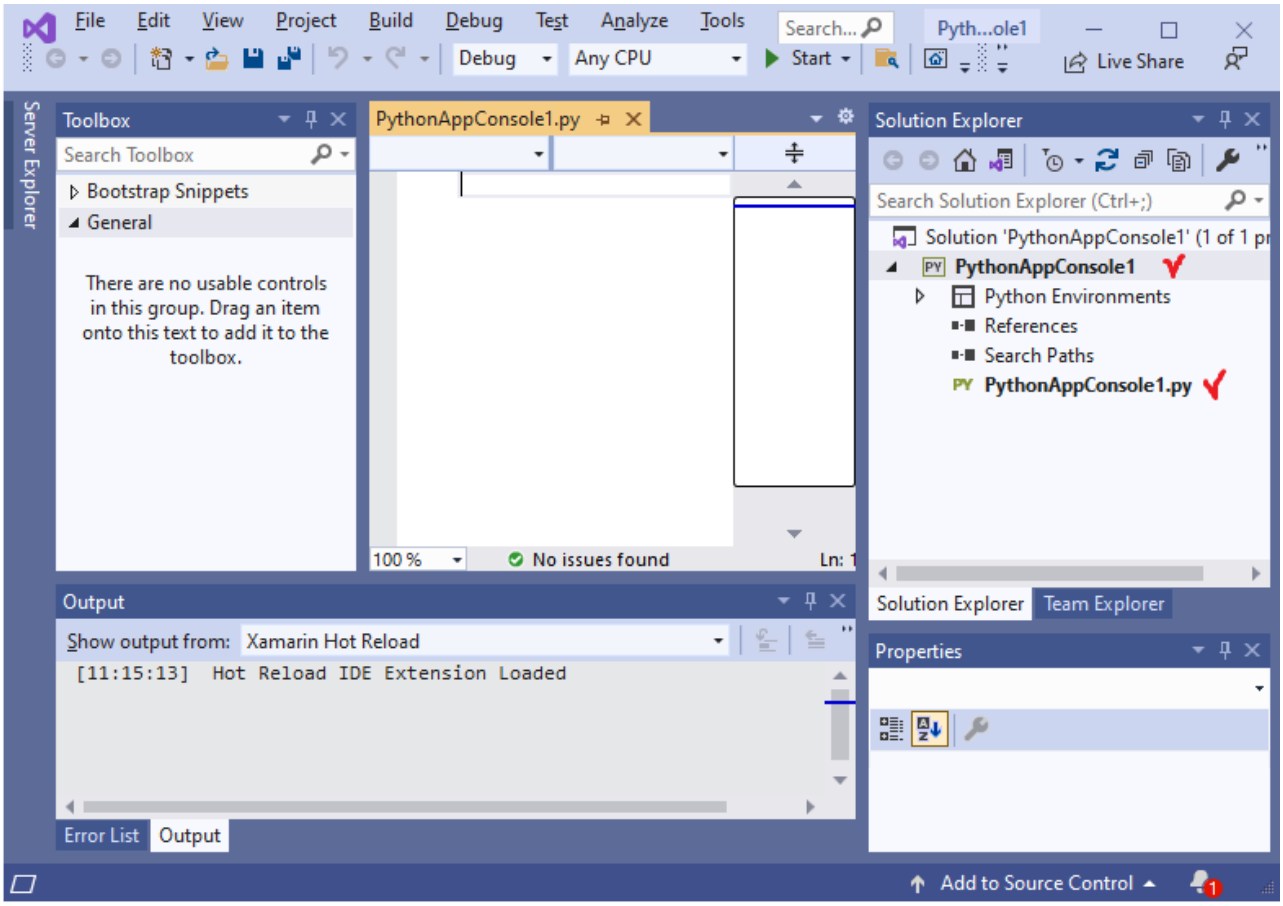
ნახ.6.34

განვსაზღვროთ კონფიგურაცია:



ნახ.6.35. აპლიკაციის სახელი და ლოკაცია

მიიღება 6.36 ნახაზზე ნაჩვენები საწყისი მდგომარეობა. Solution Explorer-ში ჩანს შექმნის პროექტი და მისი რესურსები. ბოლო სტრიქონი პითონის კონსოლური პლიკაციის სახელია.



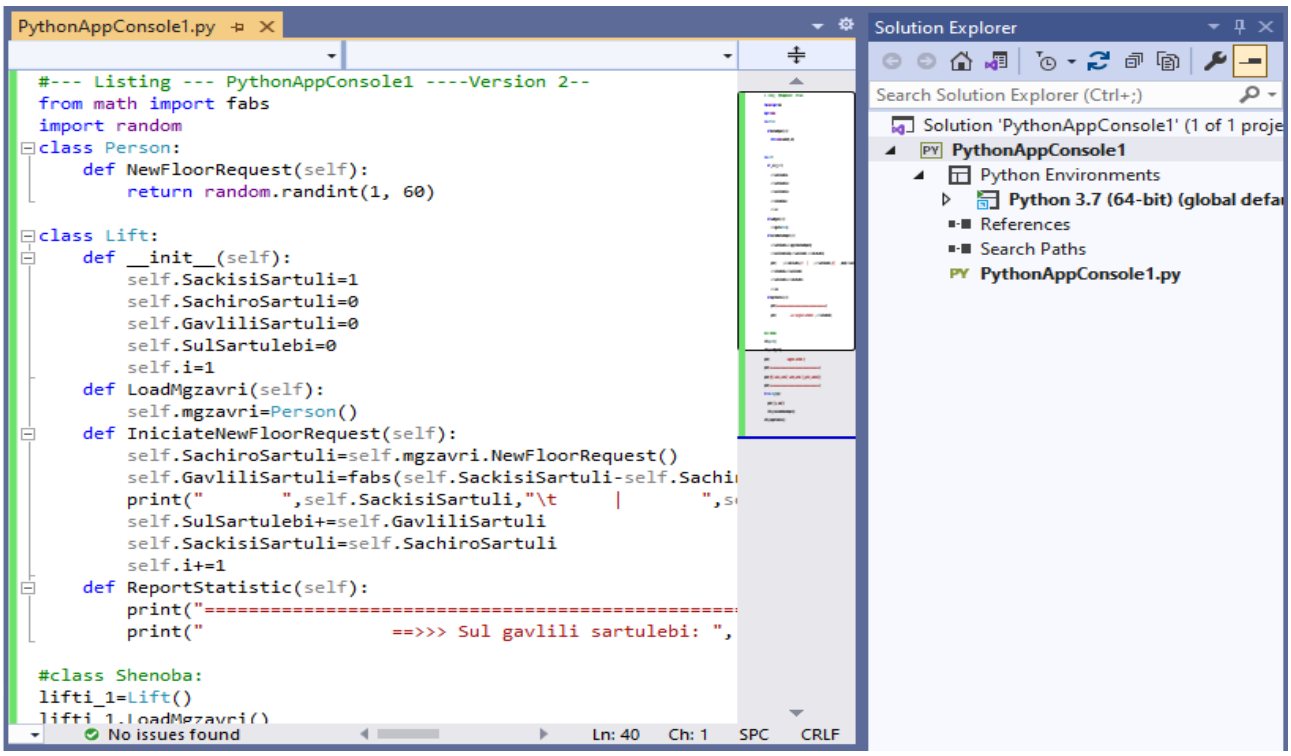
ნახ.6.36. აპლიკაციის აგების სამუშაო გარემო (VS.NET 2029)

შევიტანოთ სამუშაო არეში Python ენაზე დაწერილი კოდი, რომელიც შეესაბამება ლიფტის მოძრაობის პროცესის მართვის ამოცანას (ლაბ.2). მისი ტექსტი მოცემულია 2.6 ლისტინგში. (შეიძლება კოდი იყოს შედგენილი პროგრამისტის მიერ განსხვავებულად, მაგრამ შედეგები იქნება იდენტური ერთნაირი საწყისი პირობების დროს).

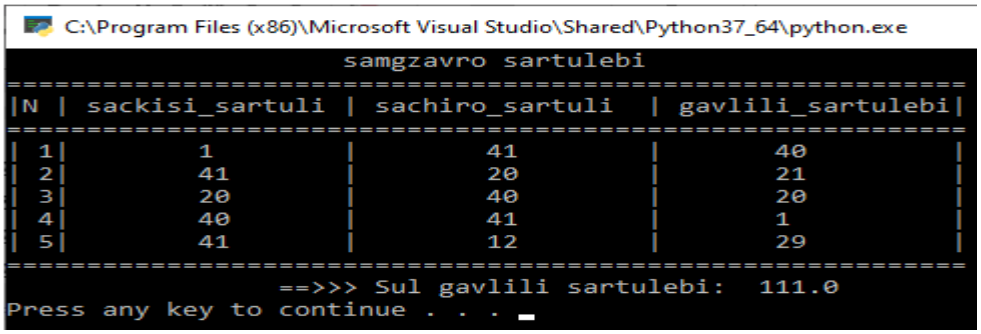
```
#--- ლისტინგი 2.6  --- PythonAppConsole1 ----Version 2--
from math import fabs
import random
class Person:
    def NewFloorRequest(self):
        return random.randint(1, 60)
class Lift:
    def __init__(self):
        self.SackisiSartuli=1
        self.SachiroSartuli=0
        self.GavliliSartuli=0
        self.SulSartulebi=0
        self.i=1
    def LoadMgzavri(self):
        self.mgzavri=Person()
    def IniciateNewFloorRequest(self):
        self.SachiroSartuli=self.mgzavri.NewFloorRequest()
        self.GavliliSartuli=fabs(self.SackisiSartuli-self.SachiroSartuli)
        print("      ",self.SackisiSartuli,"\t   |      ",self.SachiroSartuli,"\t|      ",round(self.GavliliSartuli),"\t  |")
        self.SulSartulebi+=self.GavliliSartuli
        self.SackisiSartuli=self.SachiroSartuli
```

```
        self.i+=1
    def ReportStatistic(self):
        print("============================================================")
        print("              ==>>> Sul gavlili sartulebi: ", self.SulSartulebi)
#class Shenoba:
lifti_1=Lift()
lifti_1.LoadMgzavri()
print("                 samgzavro sartulebi ")
print("============================================================")
print("|N | sackisi_sartuli | sachiro_sartuli  | gavlili_sartulebi|")
print("============================================================")
for x in range(1,6):
    print("|",x, end='|')
    lifti_1.IniciateNewFloorRequest()
lifti_1.ReportStatistic()
```

3.37 ნახაზზე ნაჩვენებია Visual Studio.NET გარემოში ჩაწერილი კოდი და მისი ამუშავებით მიღებული შედეგები (ნახ.6.38).



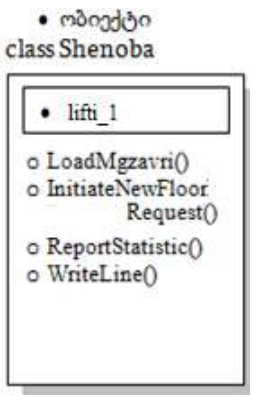
ნახ. 6.37



ნახ. 6.38. შედეგი Python აპლიკაციისთვის

6.39 ნახაზზე ნაჩვენებია ლიფტის მოძრაობის ამოცანის კლასების დიაგრამათა პროგრამული რეალიზაციის კოდები Python-ზე (C# კლასების მსაგვსად 2.1 ლისტინგში)


```
1  from math import fabs
2  import random
3
4  class Lift:
5      def __init__(self):
6          self.SackisiSartuli=1
7          self.SachiroSartuli=0
8          self.GavliliSartuli=0
9          self.SulSartulebi=0
10         self.i=1
11     def LoadMgzavri(self):
12         self.mgzavri=Person()
13     def IniciateNewFloorRequest(self):
14
15 self.SachiroSartuli=self.mgzavri.NewFloorRequest()
16         self.GavliliSartuli=fabs(self.SackisiSartuli-
17 self.SachiroSartuli)
18         print("საწყისი: ",self.SackisiSartuli,"\tსაჭირო:
19 ",self.SachiroSartuli,"\tგავლილი:
   ",round(self.GavliliSartuli))
20         self.SulSartulebi+=self.GavliliSartuli
           self.SackisiSartuli=self.SachiroSartuli
           self.i+=1
       def ReportStatistic(self):
           print("===>>> სულ გავლილი სართულები: ",
21 self.SulSartulebi)
22
23
24
25
26
27
28
29
30
31
32 class Person:
33     def NewFloorRequest(self):
           return random.randint(1, 60)
34
35
36
37
38
39
40
41
```




```
42
43
44
45  #class Shenoba:
46  lifti_1=Lift()
47  lifti_1.LoadMgzavri()
48  print("      სამგზავრო სართულები")
    print("====================================")
49  for x in range(1,6):
50      print(x, end=' ')
51      lifti_1.IniciateNewFloorRequest()
52  lifti_1.ReportStatistic()
53
54
55
56
57
58
59
60
```

ნახ.6.39

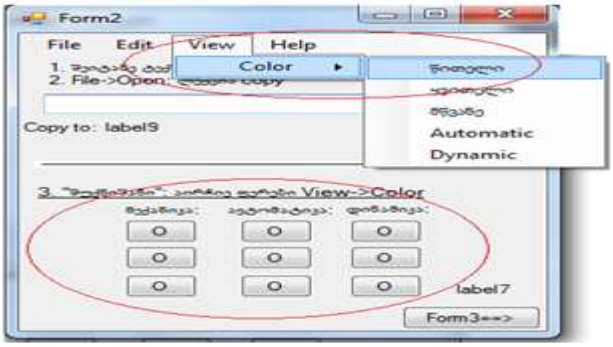
2.1 დ1 2.6 ლისტინგების შედარებით ჩანს, რომ კოდის სიგრძე (სტრიქონებში) Python ენის შემთხვევაში საგრძნობლად ნაკლებია. კოდების მუშაობის სწრაფქმედების დასადგენად საჭიროა ექსპერიმენტები ჩატარდეს შედარებით გრძელი პერიოდების განმავლობაში, კერძოდ ციკლურად შესაძლებელია ვამუშაოთ აპლიკაციები და ვაფიქსიროთ მუშაობის დაწყებისა და დასრულების დროები (გასათვალისწინებელია, რომ Python ინტერპრეტატორია, C# კი კომპილატორი).

6.4. გრაფიკული აპლიკაციების დაპროგრამება

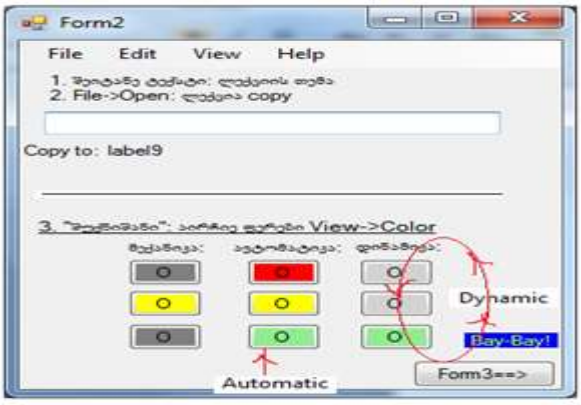
მიზანი: პროგრამული აპლიკაციების აგების გრაფიკული ელემენტების შესწავლა Visual Studio .NET Framework გარემოში [9].

6.4.1. ლაბ-N5: გრაფიკული აპლიკაცია „შუქნიშანი“

ამოცანა_1: ფორმაზე დალაგებულია მართვის ელემენტები (მაგალითად, ბუტონები), რომლებიც განასახიერებს „შუქნიშანს“ სამი ფერით: წითელი, ყვითელი და მწვანე. საჭიროა მთავარი მენიუს View -> Color პუნქტის გასწვრივ (ნახ.6.40-ა), ავირჩიოთ რომელიმე ფერი, რომელიც შეაფერადებს შესაბამის O-ბუტონს, ან ავირჩიოთ ამ ქვემენიუს სტრიქონი Automatic ან Dynamic. „ავტომატიკას“ გამოაქვს ერთბაშად სამივე ფერი, ხოლო „დინამიკას“ - ციკლში ფერთა მონაცვლეობით (ნახ.6.40-ბ).



ნახ.6.40-ა



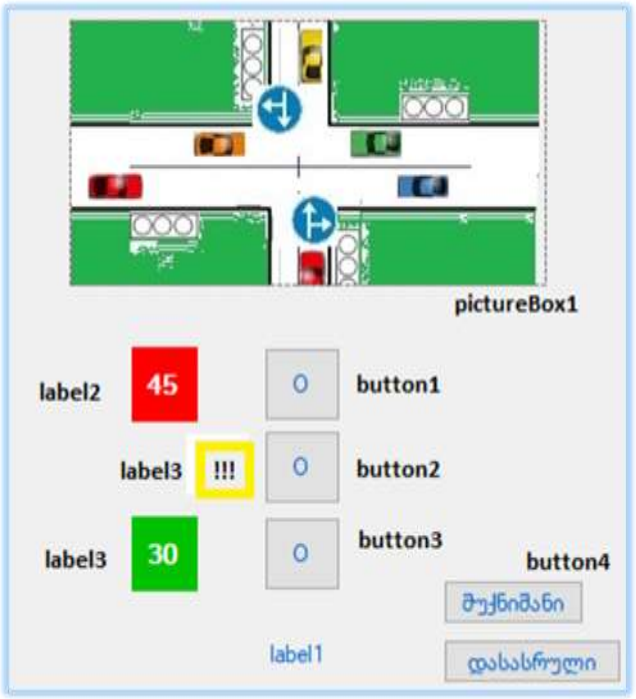
ნახ.6.40-ბ

ამოცანის გადაწყვეტის კოდი მოცემულია 5.1 ლისტინგში.

```
// ლისტინგი_5.1 ---- მთავარი მენიუ->View->Color -------
private void წითელიToolStripMenuItem_Click(object sender, EventArgs e)
 {
   button1.BackColor = Color.Red;
   button2.BackColor = Color.Gray;
   button3.BackColor = Color.Gray;
 }
private void ყვითელიToolStripMenuItem_Click(object sender, EventArgs e)
 {
   button1.BackColor = Color.Gray;
   button2.BackColor = Color.Yellow;
   button3.BackColor = Color.Gray;
 }
private void მწვანეToolStripMenuItem_Click(object sender, EventArgs e)
 {
   button1.BackColor = Color.Gray;
   button2.BackColor = Color.Gray;
   button3.BackColor = Color.LightGreen;
 }

private void automaticToolStripMenuItem_Click(object sender, EventArgs e)
 {
   button4.BackColor = Color.Red;
   button5.BackColor = Color.Yellow;
   button6.BackColor = Color.LightGreen;
 }
private void dynamicToolStripMenuItem_Click(object sender, EventArgs e)
 {
  for (int j = 1; j < 5; j++)
  {
   for (int i = 1; i < 4; i++)
   {
   switch (i)
    {
     case 1: button7.BackColor = Color.Red;
             button8.BackColor = Color.LightGray;
             button9.BackColor = Color.LightGray;
             break;
     case 2: button7.BackColor = Color.LightGray;
             button8.BackColor = Color.Yellow;
             button9.BackColor = Color.LightGray;
             break;
     case 3: button7.BackColor = Color.LightGray;
             button8.BackColor = Color.LightGray;
             button9.BackColor = Color.LightGreen;
             break;
     default: break;
    }
   button7.Refresh();  button8.Refresh(); button9.Refresh();
   Thread.Sleep(1500);
  }
 }
```

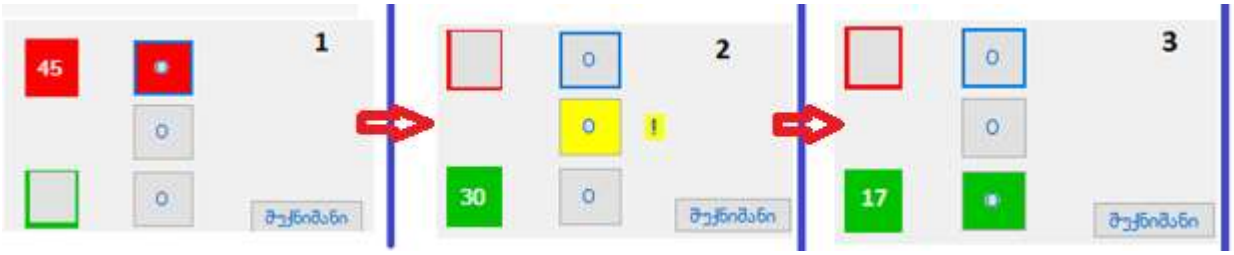
```
 label7.BackColor = Color.Blue;
 label7.ForeColor = Color.GreenYellow;
 label7.Text = "Bay-Bay!";
}
```

ამოცანა_5.2: ფორმაზე Form3 ავაგოთ მოდელი „შუქნიშანი“, რომელიც იმუშავებს რეალური შუქნიშნის მსგავსად ფერთა მონაცვლეობის პრინციპით დროის გარკვეული ინტერვალებით. 6.41 ნახაზზე ნაჩვენებია ასეთი პროექტის ერთ-ერთი ვარიანტი.



ნახ. 6.41

ღილაკი „შუქნიშანი“ რეალიზებულია ფერთამონაცვლეობის დინამიკით და დროითი დაყოვნებების გათვალისწინებით. ეს ფუნქციონალობა ასახულია 5.2 ლისტინგში, ხოლო შედეგები 6.42 ნახაზზე.



ნახ. 6.42. შუქნიშნის მუშაობის მოდელის ეტაპები

```
// ლისტინგი_5.2 --- „შუქნიშანი“ --------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
namespace WinFormAppShuqnishani
{
```

```
public partial class Form1 : Form
{
   public Form1()
   {     InitializeComponent();    }

   private void Button4_Click(object sender, EventArgs e)
   {
      { for (int j = 1; j < 3; j++)
         { for (int i = 1; i < 3; i++)
            {
             switch (i)
              {
                 case 1:
                    label2.Visible = true;
                    label3.Visible = false;
                    button1.BackColor = Color.Red;
                    button2.BackColor = Color.LightGray;
                    button3.BackColor = Color.LightGray;
                    button1.Refresh();
                    button2.Refresh();
                    button3.Refresh();
                    for (int k = 45; k >= 0; k--)
                    {
                       label2.Text = k.ToString();
                       label2.Refresh();
                       Thread.Sleep(500);
                    }
                    label2.Visible = false;
                    label3.Visible = false;
                    button1.BackColor = Color.LightGray;
                    button2.BackColor = Color.Yellow;
                    button3.BackColor = Color.LightGray;
                    //tm.Enabled = true;
                    //tm.Tick += new EventHandler(tm_tick);
                    button1.Refresh();
                    button2.Refresh();
                    button3.Refresh();
                    Thread.Sleep(800);
                    break;
                 case 2:
                    label2.Visible = false;
                    label3.Visible = true;
                    button1.BackColor = Color.LightGray;
                    button2.BackColor = Color.LightGray;
                    button3.BackColor = Color.Green;
                    button1.Refresh();
                    button2.Refresh();
                    button3.Refresh();
                    for (int k = 30; k >= 0; k--)
                    {
                       label3.Text = k.ToString();
                       label3.Refresh();
                       label2.Refresh();
```

```
                    Thread.Sleep(500);
                }
                label2.Visible = false;
                label3.Visible = false;
                button1.BackColor = Color.LightGray;
                button2.BackColor = Color.Yellow;
                button3.BackColor = Color.LightGray;
                // tm.Enabled = true;
                //tm.Tick += new EventHandler(tm_tick);
                button1.Refresh();
                button2.Refresh();
                button3.Refresh();
                Thread.Sleep(250);
                // 3 secondiani  ----------------
                label2.Visible = false;
                label3.Visible = true;
                button2.BackColor = Color.Yellow;
                button3.BackColor = Color.Green;
                button2.Refresh();
                button3.Refresh();
                for (int k = 3; k >= 0; k--)
                {
                    label3.Text = k.ToString();
                    label3.Refresh();
                    Thread.Sleep(500);
                }
                Thread.Sleep(250);
                break;
              default: break;
            }
            Thread.Sleep(500);
          }
        }
        button2.BackColor = Color.LightGray;
        button3.BackColor = Color.LightGray;
        button2.Refresh();
        button3.Refresh();
        label3.Visible = false;
        label1.BackColor = Color.Blue;
        label1.ForeColor = Color.GreenYellow;
        label1.Text = "Bye-Bye!";
      }
    }

    private void Button5_Click(object sender, EventArgs e)
    {
      Close();
    }
  }
}
```

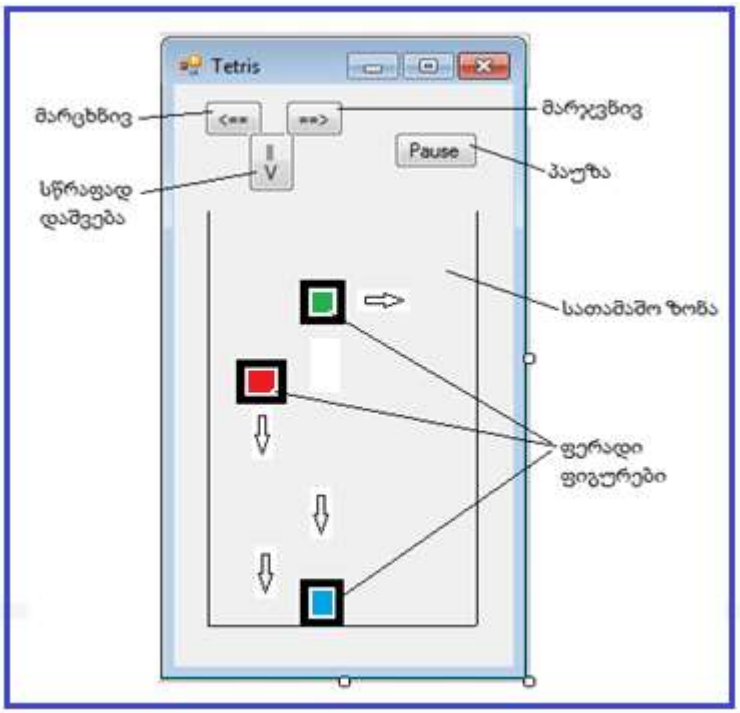
ახლა შესაძლებელია პროგრამის ამუშავება შესრულებაზე.

6.4.2. სათამაშო ანიმაციური პროგრამის აპლიკაცია (“Tetris”)

კომპიუტერული სათამაშო აპლიკაციების შექმნა მომგებიანი მიმართულებაა პროგრამული ინდუსტრიის ბიზნესის სფეროში. ჩვენი მიზანია აქ წარმოვადგინოთ ერთ-ერთი უძველესი (1984 წ. შექმნილი), მაგრამ დღესაც ძალზე პოპულარული პროგრამული თამაში - “Tetris”. არსებობს ცნობილი ფირმების მიერ შექმნილი ამ თამაშის მრავალი ინტერპრეტაციული ვერსია [90- Tetris. Internet resource: https://en.wikipedia.org/wiki/Tetris] [9].

განვიხილოთ ამ კომპიუტერული თამაშის პროგრამული აპლიკაციის შექმნის პრინციპები და რეალიზაციის C# -კოდები .NET პლატფორმაზე.

თამაშის ინტერფეისი 6.43 ნახაზეა მოცემული.



ნახ.6.43

თამაშის მიზანი: არ გაივსოს ჭურჭელი ფერადი ფიგურებით !

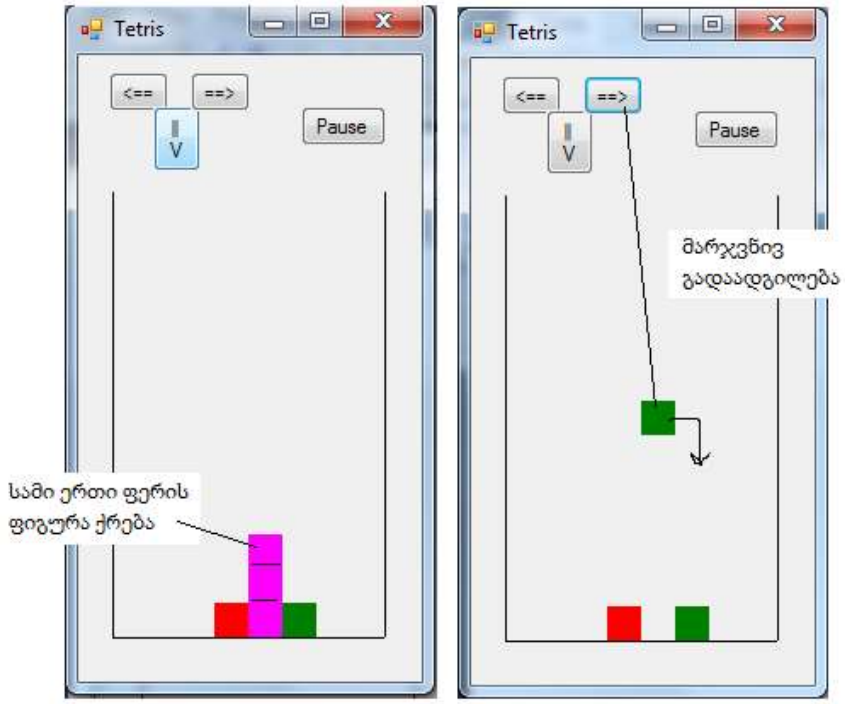
ტაქტიკა: ერთფეროვანი ფიგურები დალაგდეს „ერთ ვერტიკალში“ ან „ერთ ჰორიზონტალში“. როცა რამდენიმე ერთფეროვანი ფიგურა მოგროვდება (მაგალითად, 3), მაშინ სამივე ქრება, ანუ ჭურჭელი თავისუფლდება ახალი ფიგურებისთვის.

ფერადი ფიგურები ჩნდება შემთხვევით რიცხვთა გენერატორით და ეშვება ქვემოთ. „მარცხენა“ და „მარჯვენა“ ღილაკებით ჩვენ უნდა გადავაადგილოთ სასურველ ვერტიკალში (ნახ.6.44). თუ ორი ერთნაირი ფერის ფიგურაზე დაჯდა სხვა ფერის ფიგურა, მაშინ ის იბლოკება მანამ, სანამ ზედა ახალი ფერის ფიგურები არ გაქრება.

თამაშის ტემპი (ფიგურების დაშვების სისწრაფე ჭურჭლის ფსკერზე) პირველ ეტაპზე ნორმალურია. როცა მოთამაშე კარგად ართმევს თავს ამ ეტაპს, ანუ ფერადი ფიგურებით არ ივსება ჭურჭელი, მაშინ პროგრამა *უმატებს* ახალი ფერადი ფიგურების გაჩენის და ფსკერზე დაშვების *სისწრაფეს* (რთულდება პროცესის მართვა). თუ ამასაც გაართვა თავი მოთამაშემ, მაშინ კიდევ ახალი ეტაპი იწყება და ა.შ.

პროგრამის ასაგებად უნდა გამოვიყენოთ:

- ორგანზომილებიანი ველი;
- დროითი Timer;
- შემთხვევით რიცხვთა გენერატორი;
- ფერადი ფიგურების (კვადრატები) გაჩენა და გაქრობა მუშაობის პროცესში;
- ღილაკების (მოვლენების) მართვის ორგანიზება „ახალგაჩენილ“ ფერადი ფიგურებისთვის.



ნახ.6.44. ფიგურების მართვა

პროგრამაში თვით ფერადი ფიგურა რეალიზებულია Panel-ტიპის ელემენტით, რომელსაც შეუძლია რვა ფერიდან ერთ-ერთის მიღება.

თამაშის პროგრამული აპლიკაციის მთლიანი კოდი მოცემულია ლისტინგში.

```
// ლისტინგი_15.1 ---  “Tetris” C# პროგრამის კოდი ------
using System;
using System.Collections;
using System.Drawing;
using System.Windows.Forms;
namespace Tetris
{
   public partial class Form1 : Form
   {
     public Form1() { InitializeComponent();  }
      // აქტუალური პანელის ინდექსი Index
      int PX;
      //  სათამაშო ბილიკების რაოდენობა
      int[,] F = new int[15, 10];
      // სტრიქონები და სვეტები აქტუალური პანელის
      int PZ, PS;
      // სირთულის დონე
      int Stufe;
      // სათამაშო პანელების საწყისი ცარიელი სია
      ArrayList PL = new ArrayList();
      //   პანელების ფერთა სიმრავლე
      Color[] FarbenFeld = {Color.Red,
```

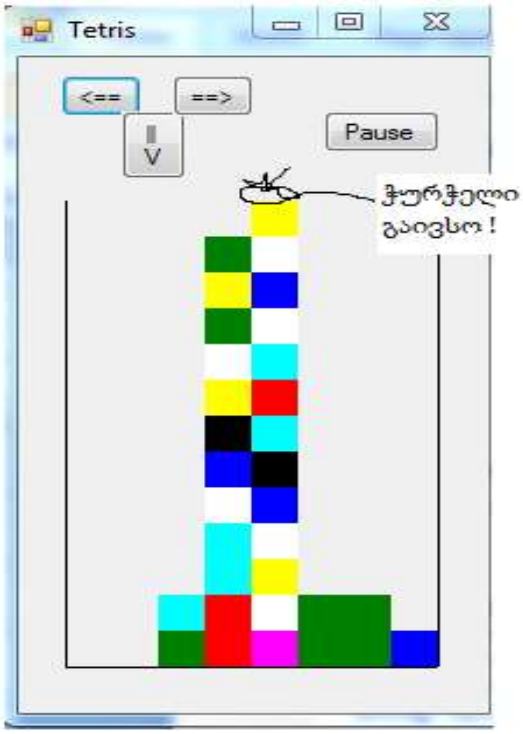
```
        Color.Yellow, Color.Green, Color.Blue,
        Color.Cyan, Color.Magenta, Color.Black,
        Color.White};
    // ბილიკის სტატუსის კონსტანტები
    const int Leer = -1;
    const int Rand = -2;
    //  შემთხვევით რიცხვთა გენერატორი: გაჩენა და გააქტიურება
    Random r = new Random();

    private void Form1_Load(object sender, EventArgs e)
    {
        int Z, S;
        // ბილიკი დაკავებულია
        for (Z=1; Z<14; Z++)
        {
            F[Z, 0] = Rand;
            for (S=1; S<9; S++)
                F[Z, S] = Leer;
            F[Z, 9] = Rand;
        }
        for (S=0; S<10; S++)
            F[14, S] = Rand;
        // ინიციალიზირება
        Stufe = 1;
        NächstesPanel();
    }

    private void NächstesPanel() // შემდეგი პანელი
    {
        int Farbe;
        Panel p = new Panel();
        //  ახალი პანელის შეტანა მასივის სიაში
        PL.Add(p);
        // ”დაკლიკვის”* მოვლენის დამუშავება
        p.Click += new EventHandler(PanelClickReaktion);
        // ახალი პანელის ადგილზე ჩასმა
        p.Location = new Point(100, 80);
        p.Size = new Size(20, 20);
        // ახალი პანელისთვის ფერის შერჩევა
        Farbe = r.Next(0,8);
        p.BackColor = FarbenFeld[Farbe];
        // ახალი პანელის ფორმაზე შეტანა
        Controls.Add(p);
        // მომავალი წვდომის ინდექსის განსაზღვრა
        PX = PL.Count - 1;
        // პანელის საინფორმაციო ინდექსის დაფიქსირება
        p.Tag = PX;
        // აქტუალური სტრიქონი და სვეტი
        PZ = 1;
        PS = 5;
    }
private void PanelClickReaktion(object sender, EventArgs e)
    {
```

```
    // რომელი პანელი დაიკლიკა
    Panel p = (Panel) sender;
    // დაკლიკული პანელის თვისებების შეცვლა
    lblPNr.Text = "P " + p.Tag;
    p.BorderStyle = BorderStyle.FixedSingle;
}

private void timT_Tick(object sender, EventArgs e)
{
    //  თუ დასრულდა ან აღარ მუშაობს
    if (F[PZ + 1, PS] != Leer)
    {
        // მიღწეულია ზედა სტრიქონი
        if (PZ == 1)
        {
            timT.Enabled = false;
            MessageBox.Show("აბა რა !");
            return;
        }
        F[PZ, PS] = PX;       // დაკავება
        AllePrüfen();
        NächstesPanel();
    }
    else
    {
        // თუ გრძელდება თამაში კიდევ
        Panel p = (Panel) PL[PX];
        p.Top = p.Top + 20;
        PZ = PZ + 1;
    }
}

private void AllePrüfen()
{
    int Z, S;
    bool Neben, Über;
    Neben = false;
    Über = false;
    // სამი ერთფეროვანი პანელი გვერდი-გვერდაა ?
    for(Z=13; Z>0; Z--)
    {
        for(S=1; S<7; S++)
        {
            Neben = NebenPrüfen(Z, S);
            if (Neben) break;
        }
        if (Neben) break;
    }
    // სამი ერთფეროვანი პანელი ერთმანეთზეა  ?
    for(Z=13; Z>2; Z--)
    {
        for(S=1; S<9; S++)
        {
```

```
            Über = ÜberPrüfen(Z, S);
            if (Über) break;
        }
        if (Über) break;
    }
    if (Neben || Über)
    {
        // უფრო სწრაფად
        Stufe = Stufe + 1;
        timT.Interval = 5000 / (Stufe + 9);
        // ალბათ შესაძლებელია კიდევ ერთი რიგის ამოგდება
        AllePrüfen();
    }
}
// თუ სამი ბილიკი ერთმანეთის გვერდით დაკავებულია
private bool NebenPrüfen(int Z, int S)
{
    int ZX, SX;
    bool ergebnis = false;
    if (F[Z, S] != Leer &&
            F[Z, S + 1] != Leer &&
            F[Z, S + 2] != Leer)
    {
        Panel p = (Panel) PL[F[Z, S]];
        Panel p1 = (Panel) PL[F[Z, S + 1]];
        Panel p2 = (Panel) PL[F[Z, S + 2]];
        /* თუ სამი ერთნაირი ფერია
        if (p.BackColor == p1.BackColor &&
                p.BackColor == p2.BackColor)
        {
            for(SX=S; SX<S+3; SX++)
            {
                /* PL წაიშალოს ფორმიდან
                Control c = (Control) PL[F[Z, SX]];
                Controls.Remove(c);
                /* ბილიკი თავისუფლდება */
                F[Z, SX] = Leer;
             // ზედა პანელი ათავისუფლებს ქვედა დაბლოკილს
                ZX = Z - 1;
                while (F[ZX, SX] != Leer)
                {
                    Panel px =
                        (Panel) PL[F[ZX, SX]];
                    px.Top = px.Top + 20;
                    // ბილიკი კვლავ კავდება
                    F[ZX + 1, SX] = F[ZX, SX];
                    F[ZX, SX] = Leer;
                    ZX = ZX - 1;
                }
            }
            ergebnis = true;
        }
    }
```

```
        return ergebnis;
    }
    // თუ სამი ერთფეროვანია ერთმანეთზე
    private bool ÜberPrüfen(int Z, int S)
    {
        int ZX;
        bool ergebnis = false;

        if (F[Z, S] != Leer && F[Z - 1, S] != Leer &&
            F[Z - 2, S] != Leer)
        {
            Panel p = (Panel) PL[F[Z, S]];
            Panel p1 = (Panel) PL[F[Z - 1, S]];
            Panel p2 = (Panel) PL[F[Z - 2, S]];
            // თუ სამი ფერი ერთნაირია
            if (p.BackColor == p1.BackColor &&
                p.BackColor == p2.BackColor)
            {
                // სამი პანელი ქრება
                for (ZX=Z; ZX>Z-3; ZX--)
                {
                    // PL იშლება ფორმიდან
                    Control c = (Control) PL[F[ZX, S]];
                    Controls.Remove(c);
                    // ბილიკი თავისუფლდება
                    F[ZX, S] = Leer;
                }
                ergebnis = true;
            }
        }
        return ergebnis;
    }

private void cmdLinks_Click(object sender, EventArgs e)
    {
        if (F[PZ, PS - 1] == Leer)
        {
            Panel p = (Panel) PL[PX];
            p.Left = p.Left - 20;
            PS = PS - 1;
        }
    }
private void cmdRechts_Click(object sender, EventArgs e)
    {
        if (F[PZ, PS + 1] == Leer)
        {
            Panel p = (Panel) PL[PX];
            p.Left = p.Left + 20;
            PS = PS + 1;
        }
    }
private void cmdUnten_Click(object sender, EventArgs e)
    {
```

```
            while (F[PZ + 1, PS] == Leer)
            {
               Panel p = (Panel) PL[PX];
               p.Top = p.Top + 20;
               PZ = PZ + 1;
            }
            F[PZ, PS] = PX;       // დაკავება
            AllePrüfen();
            NächstesPanel();
        }

    private void cmdPause_Click(object sender, EventArgs e)
        {
            timT.Enabled = !timT.Enabled;
        }
    }
}
```



ნახ.6.45. ჭურჭლის გავსებით სრულდება თამაშის სეანსი

დავალება:

1. შექმენით ახალი Windows Forms App - ის პროექტი სათამაშო აპლიკაციისთვის;
2. გადაიტანეთ თამაშის კოდი აპლიკაციის პროექტში
3. გამართეთ და აამუშავეთ აპლიკაცია
4. გააანალიზეთ ექსპერიმენტის შედეგები რამდენიმე ვარიანტისთვის;

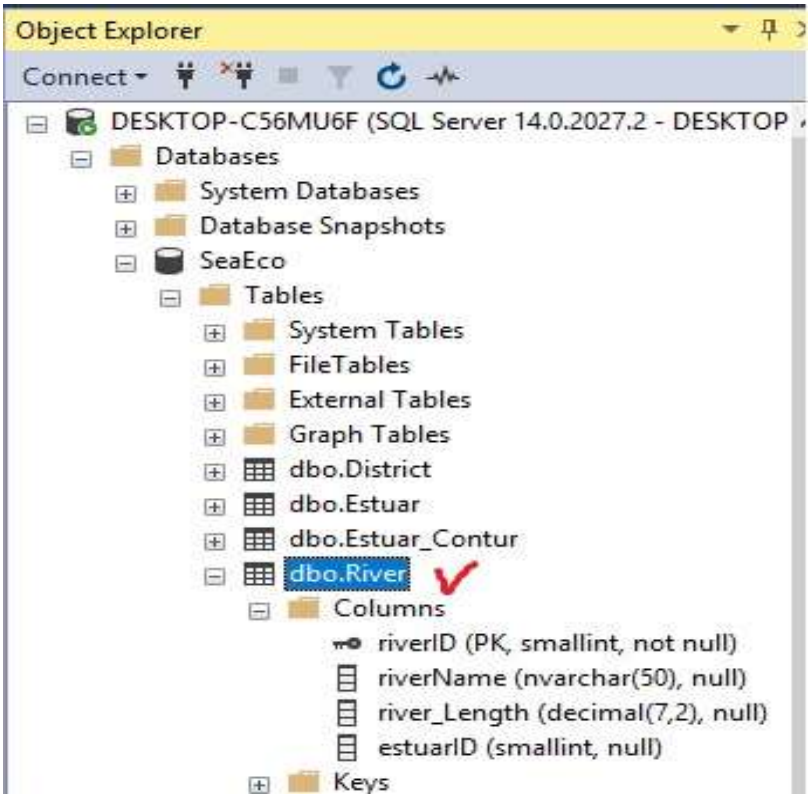
6.5. ლაბ-N7: მომხმარებლის ინტერფეისის აპლიკაცია SQL Server ბაზასთან სამუშაოდ

მიზანი: პროგრამული აპლიკაციების აგების შესწავლა C# ენის, Ms SQL Server მონაცემთა ბაზების და ADO.NET დრაივერის ერთობლივი გამოყენებით. ADO.NET-ის დანიშნულების და მისი ძირითადი ობიექტების გაცნობა, რომელთა საშუალებითაც ხორციელდება კავშირი C# ენაზე დაწერილ ინტერფეისსა და მონაცემთა ბაზას შორის.

ამოცანა_1: მოცემულია SQL Server მონაცემთა ბაზა (მაგალითად, შავი ზღვის ეკოლოგიური მონიტორინგის სისტემა [1,10]), რომლის ერთ-ერთი ცხრილი (Table) არის River.dbo (საქართველოს მდინარეები). საჭიროა ავაგოთ Windows Forms აპლიკაცია (მომხმარებლის ინტერფეისი), რომელიც მონაცემთა ბაზიდან ამოიღებს მდინარეების მონაცემებს DataGridView ცხრილში, შეძლებს Insert, Update და Delete ოპერაციების განხორციელებას. გამოყენებულ უნდა იქნას ADO.NET დრაივერის საშუალებები.

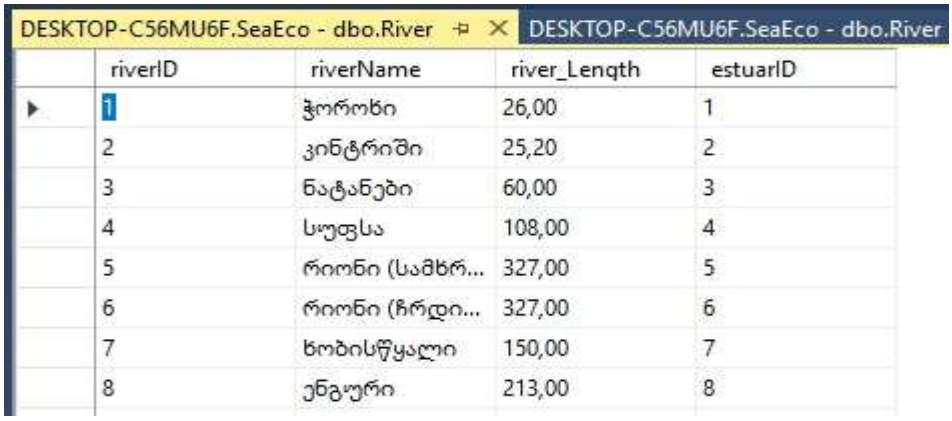
- **ექსპერიმენტული მონაცემთა ბაზის მომზადება SQL Server პაკეტით**

6.46-ა ნახაზზე ნაჩვენებია საწყისი მონაცემთა ბაზა, რომელიც Ms SQL Server პაკეტითაა რეალიზებული. (ა) შეესაბამება ბაზის ცხრილების იერარქიასა და აქვე ჩანს River ცხრილის სტრუქტურაც (ბ), მონაცემთა შესაბამისი ტიპებით. (გ)-ზე მოცემულია ჩანაწერები, რომელთა შეტანა მოხდა წინასწარ (თუმცა ჩვენი პროგრამისთვის არაა აუცილებელი მისი არსებობა. შესაძლებელია იგი შეივსოს ინტერფეისიდან).



ნახ.6.46-ა. მონაცემთა ბაზა Ms SQL Server-ში

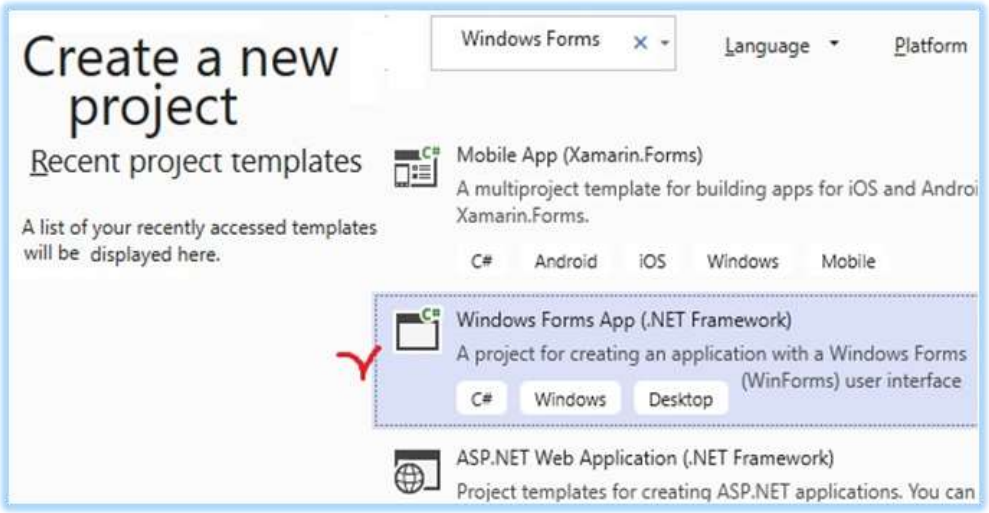
ნახ.6.46-ბ. River (მდინარეების) ცხრილის სტრუქტურა
Ms SQL Server-ში



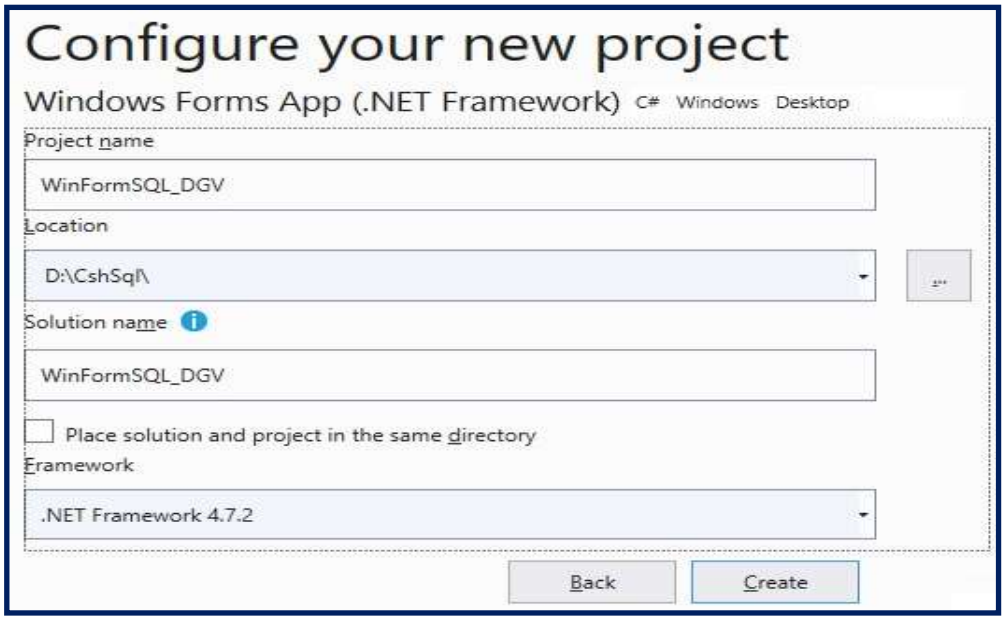
ნახ.6.46-გ. River (მდინარეების) საწყისი ცხრილი
Ms SQL Server-ში

- **ახალი პროექტის შექმნა .NET პლატფორმაზე**
  C# ენის გამოყენებით

დავიწყოთ ახალი პროექტის აგება Ms Visual Studio.NET სამუშაო გარემოში. 6.47 ნახაზზე ნაჩვენებია პროექტის შექმნის პროცედურა.



ნახ.6.47-ა. WinFormSQL_DGV პროექტის შექმნა

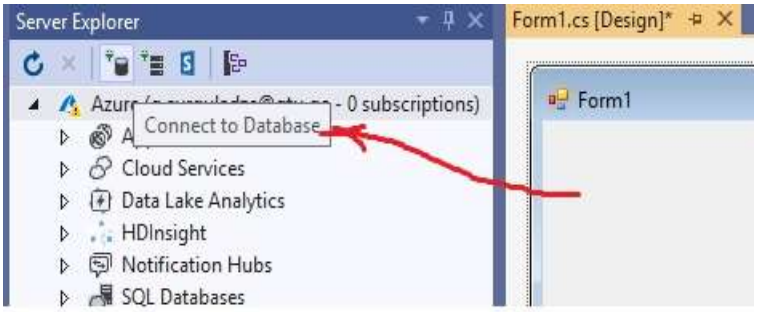


ნახ.6.47-ბ. WinFormSQL_DGV პროექტის შექმნა და განთავსება სისტემის კატალოგში

ვირჩევთ პროექტის სახელს (Name), მისი შენახვის ადგილს (Browse-ს დახმარებით) და Solution name-ს. შემდეგ OK.

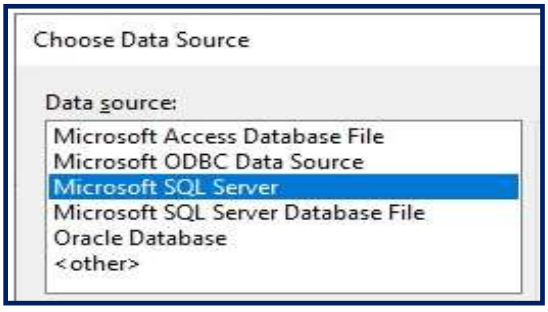
➢ **პროექტთან მონაცემთა ბაზის მიერთება**

საჭიროა პროექტისთვის განვახორციელოთ მონაცემა ბაზის მიერთება (Connect to Database), რაც 6.48 ნახაზზეა ნაჩვენები.

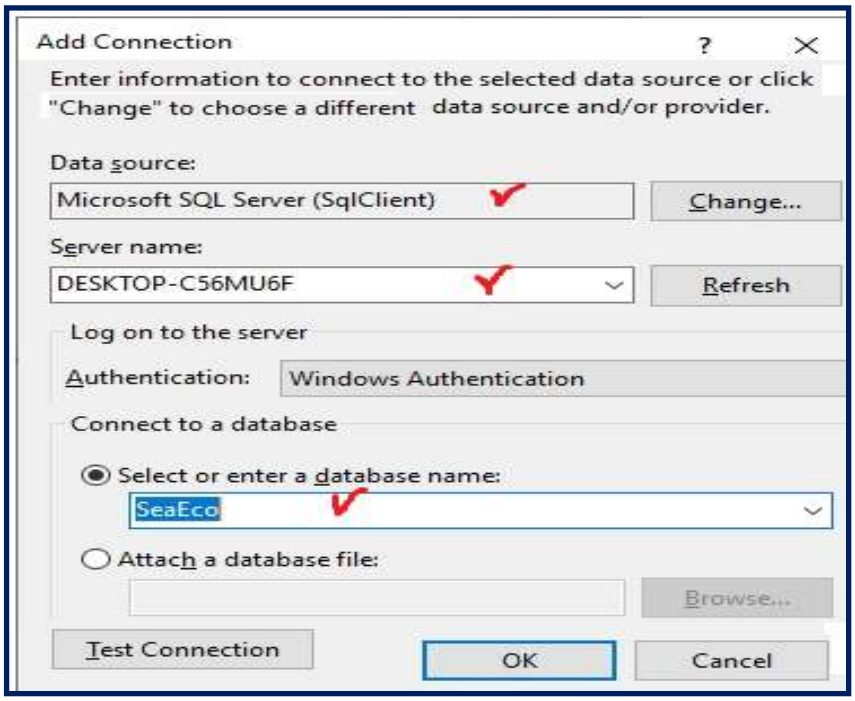


ნახ.6.48. Connect Database ამოქმედება

გამოიტანება ახალი ფანჯარა (ნახ.6.49), რომელშიც უნდა შეირჩეს შესაბამისი წყარო (Data Source), სერვერი (Server name) და მონაცემთა ბაზა (Database name).



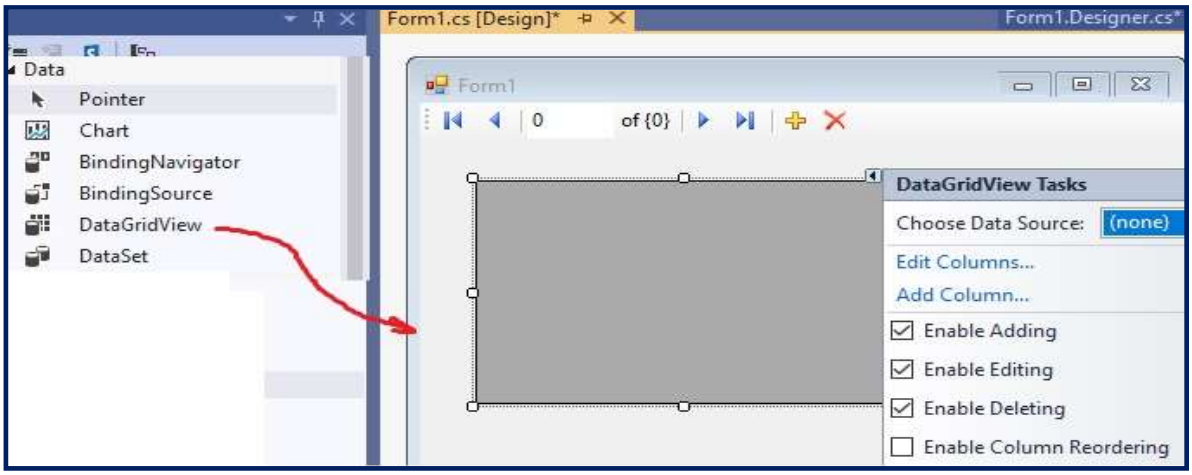
ნახ. 6.49-ა. მბ-ის წყაროს არჩევა



ნახ.6.49-ბ. მბ-ის სერვერის და ბაზის სახელის არჩევა

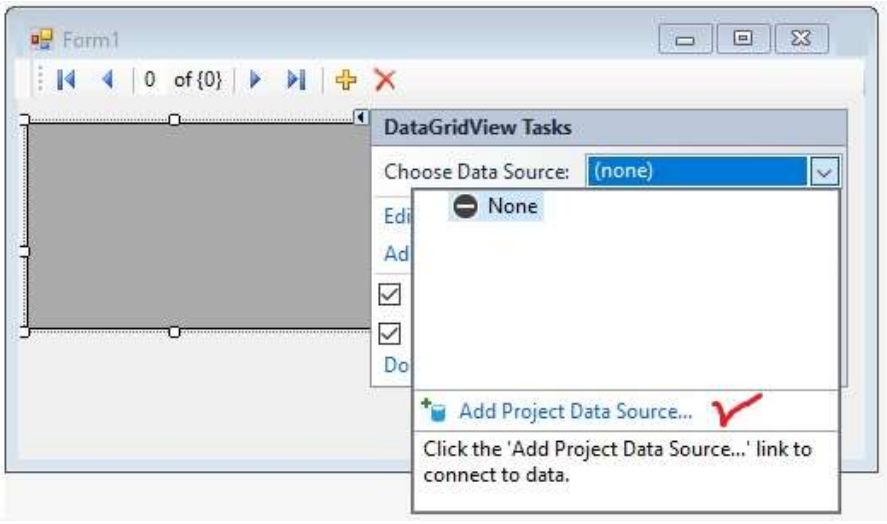
- **DataGridView ელემენტის გააქტიურება და ცხრილის პარამეტრების განსაზღვრა**

ინსტრუმენტების პანელიდან ფორმაზე გადავიტანოთ DataGridView ელემენტი და ზედა მარჯვენა კუთხე პატარა ისრით ავამოქმედოთ. მივიღებთ 6.50 ნახაზს.



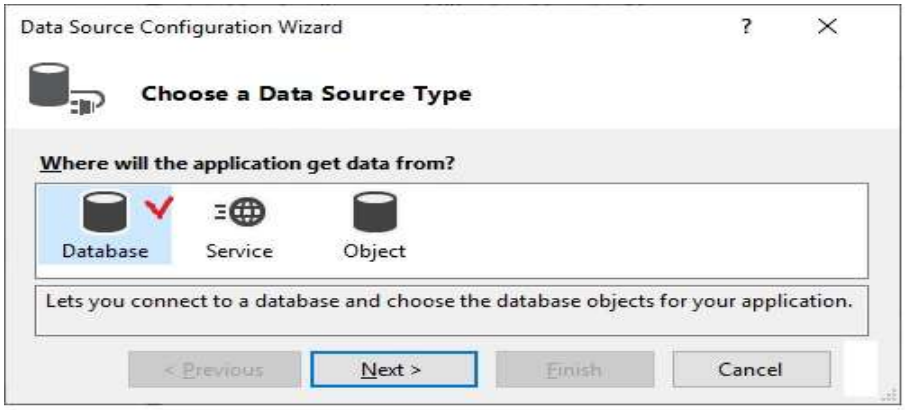
ნახ.6.50. პარამეტრების განსაზღვრა

ნახაზზე ჩანს, რომ ჩამატების, რედაქტირებისა და წაშლის ოპერაციები ნებადართულია (ჩეკბოქსები მონიშნულია). ავირჩიოთ Choose Data Source კომბობოქსის ღილაკი, მივიღებთ 6.51 ნახაზს.



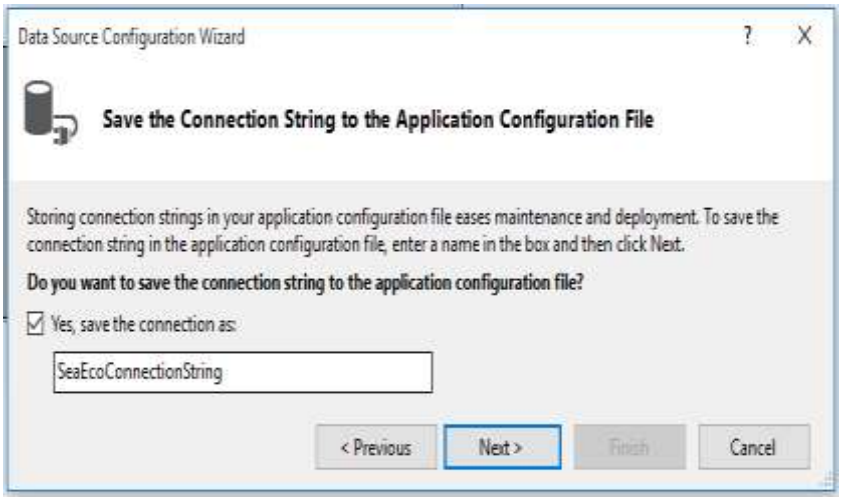
ნახ.6.52. მონაცემთა წყაროს პროექტის დამატება

ავამოქმედოთ Add Project Data Source და გადავიდეთ 6.53 ნახაზზე.



ნახ.6.53. მონაცემთა წყაროს ტიპის არჩევა

ვირჩევთ Database-ს და Next. მივიღებთ 6.54 ნახაზზე მოცემულ ფანჯარას.

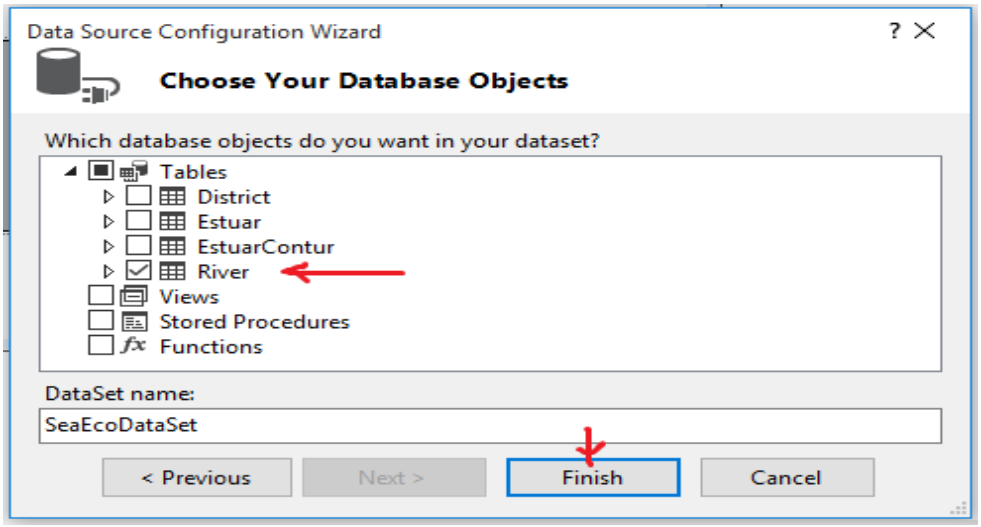


ნახ.6.54. SeaEco Connection (მიერთების) შერჩევა

Connection პარამეტრი ყველა კომპიუტერს ექნება თავისი. მისი განსაზღვრა შესაძლებელია Server Explorer-იდან (ჩვენ შემთხვევაში იგი არის: DESKTOP-C56MU6F.SeaEco.dbo). ბოლოს Next და გადავალთ 6.55 ნახაზზე.

ნახ.6.55. Connection String-ის შენახვა აპლიკაციის კონფიგურაციის ფაილში

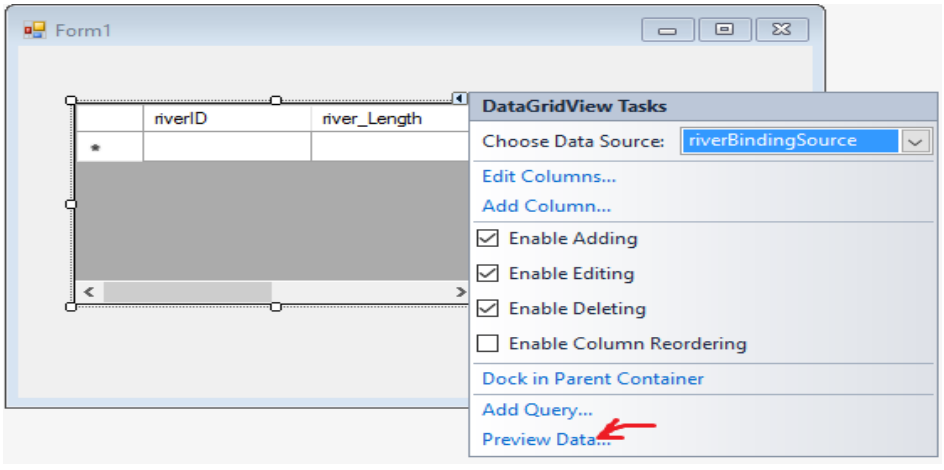
შემდეგ გამოიძახება მონაცემთა ბაზის ობიექტების არჩევის ფანჯარა (ნახ.6.56).



ნახ.6.56. ობიექტების მონიშვნა (მაგალითად, River)

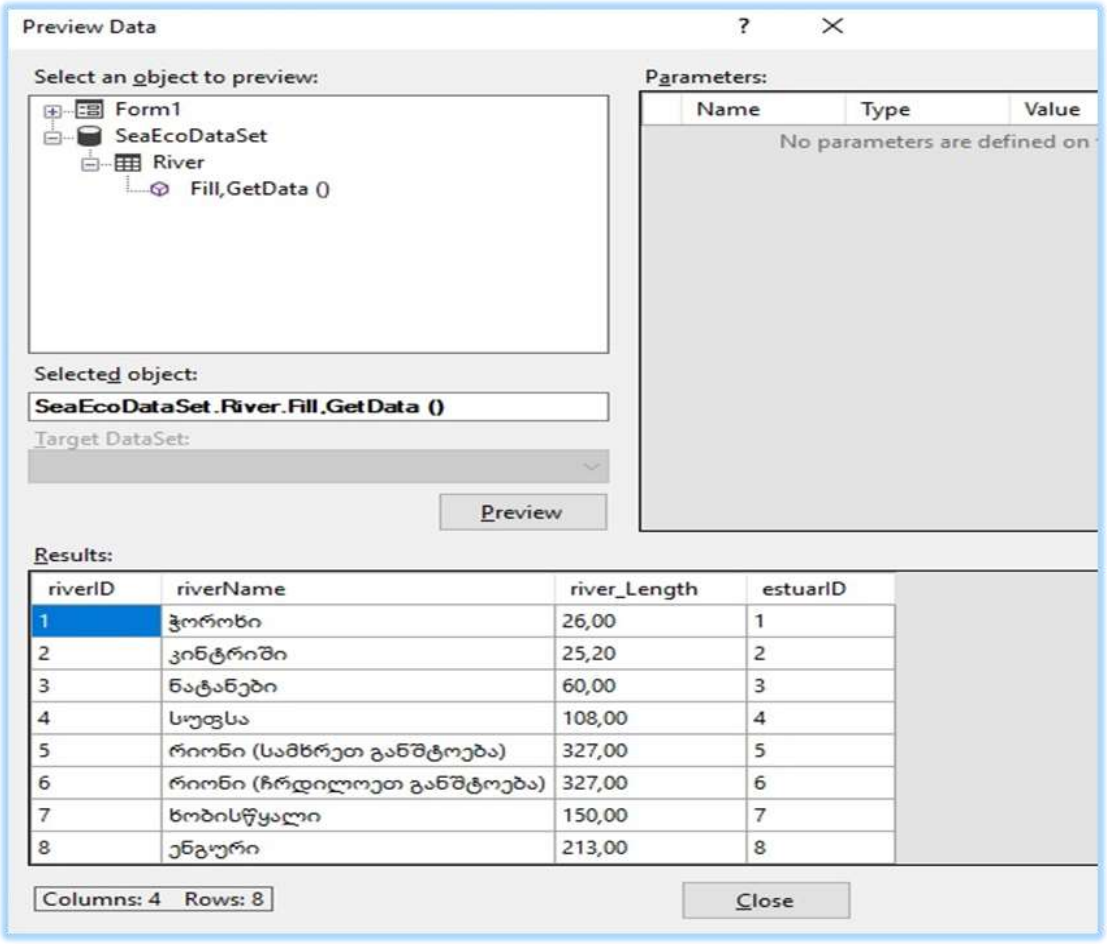
6.57 ნახაზზე ნაჩვენებია მონაცემთა წყაროს (Data Source) განსაზღვრის შედეგის მნიშვნელობა, ჩვენ შემთხვევაში იგი არის:

riverBindingSource.



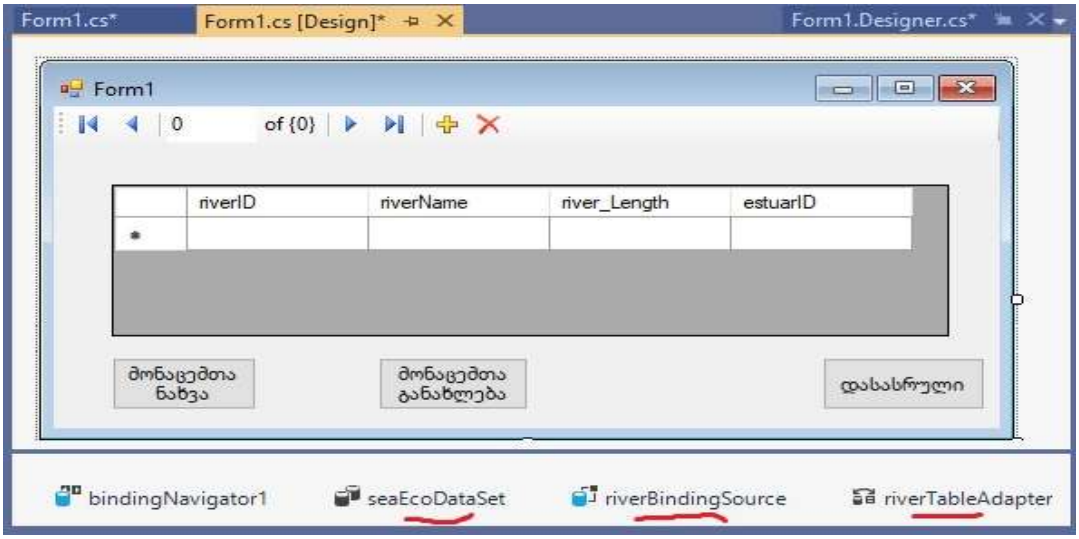
ნახ.6.57. Data Source შედეგი: “riverBindingSource”

აქვე შეიძლება გამოვიყენოთ Preview Data ლინკი და დავათვალიეროთ წინასწარ მიერთებული ბაზის ცხრილის ჩანაწერები (ნახ.6.58).



ნახ.6.58. Preview Data ცხრილი

ბოლოს, Form1-ს Properties-ში შევუცვალოთ სახელი „შავი ზღვის მდინარეები (საქართველოს საზღვრებში)“, ინსტრუმენტების პანელიდან გადმოვიტანოთ სამი ღილაკი (Button1,2.3), დავარქვათ სახელები. მიიღება 6.59 ნახაზი.



ნახ.6.59. პროექტის ძირითადი ინტერფეისი

➢ **პროგრამული რეალიზაციის საკითხები**

ახლა გადავიდეთ სისტემის ინტერფეისის ღილაკების ფუნქციების დაპროგრამებაზე, ანუ უნდა განხორციელდეს SQL Server-ის შესაბამისი ბაზის ცხრილის ჩანაწერების დათვალიერება, ჩამატება, შეცვლა და წაშლა (CEUD ოპერაციები).

– თავდაპირველად ჩავამატოთ სახელსივრცის სტრიქონი:

```
using System.Data.SqlClient;
```

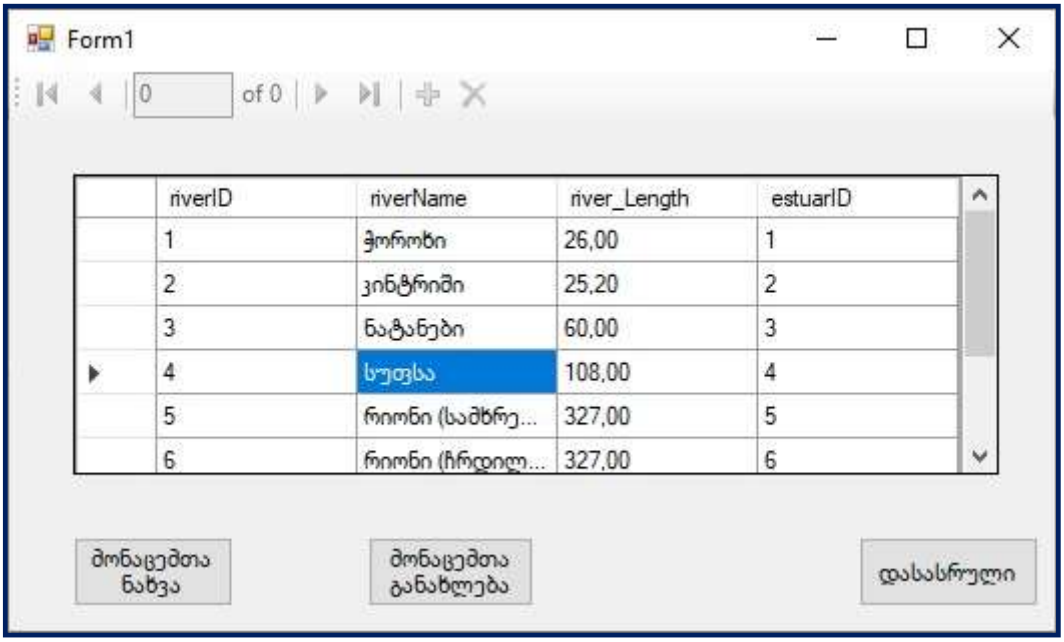
– შემდეგ გამოვაცხადოთ გლობალური ცვლადები:

```
SqlDataAdapter sda;
SqlCommandBuilder scb;
DataTable dt;
```

„მონაცემთა ნახვის“ ღილაკისათვის (button1) გვექნება შემდეგი კოდი:

```
private void Button1_Click(object sender, EventArgs e)
 {
   SqlConnection con = new SqlConnection("Data Source=DESKTOP-C56MU6F;
                       Initial Catalog = SeaEco; Integrated Security = True");
   sda = new SqlDataAdapter(@"SELECT riverID,
                      river_Length, riverName,
                       estuarID from River", con);
    dt = new DataTable();
    sda.Fill(dt);
    dataGridView1.DataSource = dt;    }
```

პროგრამის ამუშავებით, თუ მასში არაა შეცდომები, მიიღება 6.60 ნახაზი.



ნახ.6.60. „მონაცემთა ნახვის“ (DataShow) ღილაკის ამოქმედებით მიღებული შედეგი

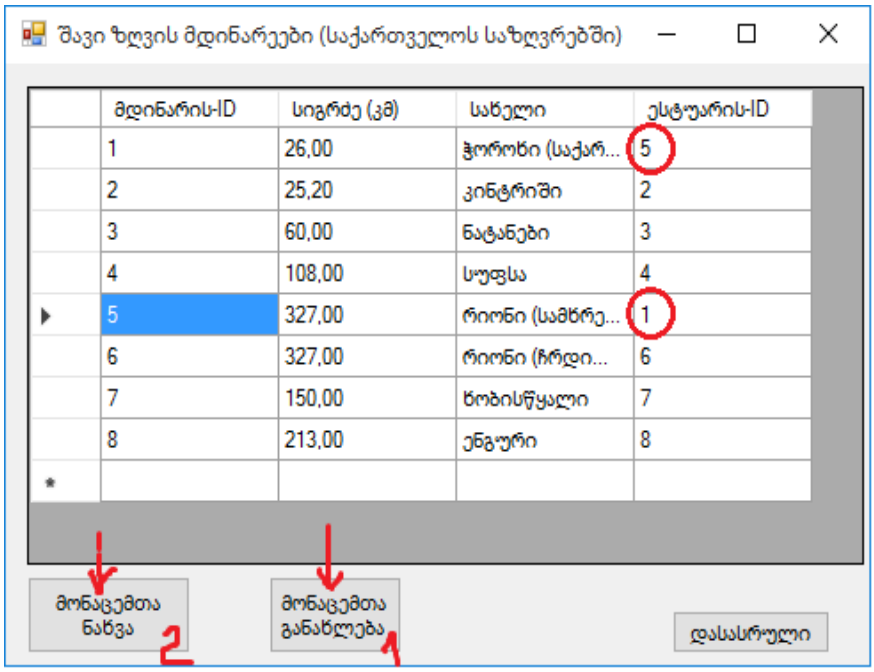
„მონაცემთა განახლების“ ღილაკის კოდი (Insert, Update, Delete) ნაჩვენებია ქვემოთ, Button2-ის ლისტინგში:

```
private void Button2_Click(object sender, EventArgs e)
 {
   scb = new SqlCommandBuilder(sda);
   sda.Update(dt);
 }
```

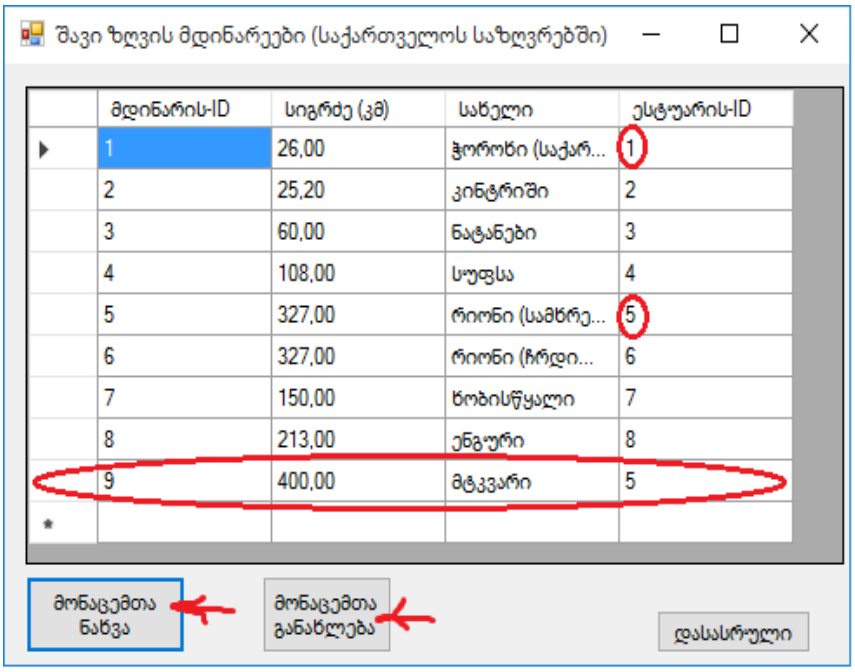
მთლიანი პროგრამის კოდი მოცემულია ლისტინგში.

```
// --- ლისტინგი 6.1 --- Insert, Update, Delete for DataGridView_SQL----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient; // !!!
namespace WinFormSQL_DGV
{
    public partial class Form1 : Form
    {
        SqlDataAdapter sda;
        SqlCommandBuilder scb;
        DataTable dt;
        public Form1()
        { InitializeComponent(); }

       private void Form1_Load(object sender, EventArgs e)
        {
          this.riverTableAdapter.Fill(this.seaEcoDataSet.River);
        }
        // მონაცემთა ნახვა ---------
        private void Button1_Click(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection("Data Source=DESKTOP-C56MU6F;
                    Initial Catalog = SeaEco; Integrated Security = True");
            sda = new SqlDataAdapter(@"SELECT riverID,
                                river_Length, riverName,
                                estuarID from River", con);
            dt = new DataTable();
            sda.Fill(dt);
            dataGridView1.DataSource = dt;
        }

        // განახლება --------
        private void Button2_Click(object sender, EventArgs e)
        {
            scb = new SqlCommandBuilder(sda);
            sda.Update(dt);
        }

        private void Button3_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

6.61 ნახაზზე მოცემულია არსებულ ცხრილში ორი მნიშვნელობის (ესტუარის ID) შეცვლა (ახალი რიცხვები ნაჩვენებია წრეში), შემდეგ 1-ელი და მე-2 ღილაკების ამოქმედებით ბაზაში ჩაიწერება ეს შეცვლილი მნიშვნელობები (შეიძლება დავხუროთ პროგრამა და თავიდან ავამუშავოთ).

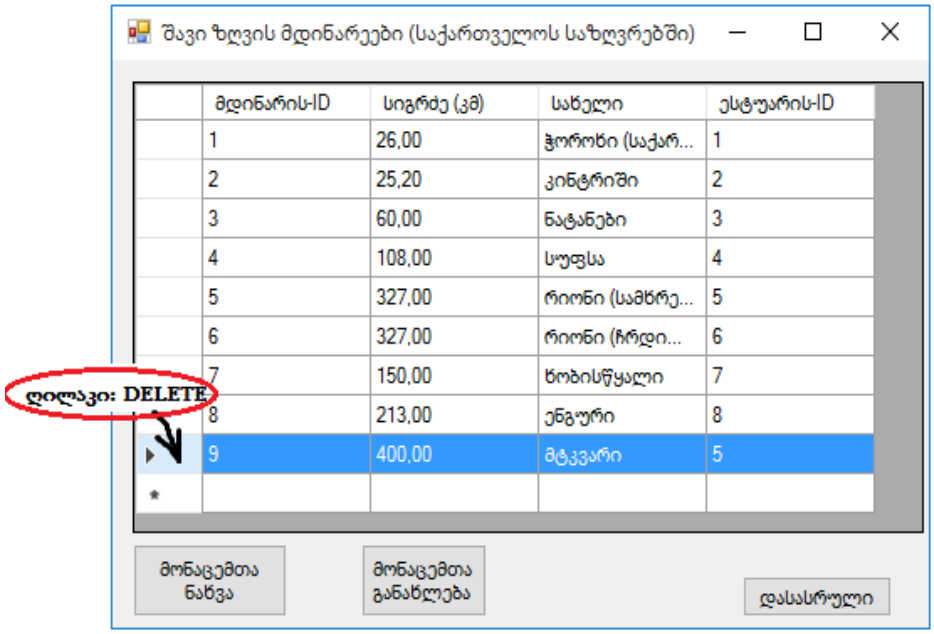


ნახ.6.61-ა. Update ცვილების განხორციელება



ნახ.6.61-ბ. Insert ოპერაციის განხორციელება

ბოლოს ნაჩვენებია სტრიქონის წაშლის (Delete) ოპერაცია. იგი ხორციელდება მაგალითად, „**მდინარის-ID**“ შესაბამისი სტრიქონის მონიშვნით და შემდეგ კომპიუტერის კლავიატურის Delete- ღილაკის ამოქმედებით (ნახ.6.62).



ნახ.6.62. Delete ოპერაციის განხორციელება

ამით დავასრულეთ ვიზუალური C# ელემენტების საფუძველზე მონაცემთა ბაზასთან დაკავშირების რეალიზაციის ამოცანის განხილვა, რომლის შედეგადაც აგებულ იქნა შავი ზღვის ეკომონიტორინგის ინფორმაციული სისტემის ფრაგმენტი.

6.6. პროგრამული აპლიკაციის მოდულების ტესტირება (Unit Testing)

6.6.1. ლაბ-N8: C# პროგრამული მოდულების ტესტირება

მიზანი: პროგრამული კოდების ტესტირების პროცესის შესწავლა Visual Studio.NET ინტეგრირებულ გარემოში [2, 11].

1. თეორიული ნაწილი

Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში.

Unit testing - ანუ მოდულური ტესტირება დაპროგრამების პროცესია, რომლის საშუალებითაც მოწმდება საწყისი კოდის ცალკეული მოდულების კორექტულობა. ასეთი ტესტირების იდეა მდგომარეობს იმაში, რომ ყოველი არატრივიალური ფუნქციის ან მეთოდისათვის დაიწეროს ტესტი. ეს უზრუნველყოფს კოდის სწრაფად შემოწმებას, ხომ არ მიიყვანა კოდის ბოლო ცვლილებამ პროგრამა რეგრესიამდე ანუ შეცდომების გაჩენამდე პროგრამის უკვე ტესტირებულ ნაწილებში [11].

Coded UI ტესტი კი ავტომატურად იწერს, შესრულებაზე უშვებს და ამოწმებს ტესტ - ქეისებს.

ასეთი ტესტების წერა შესაძლებელია C# ან Visual Basic-ზე Visual Studio გარემოში. Unit testing ტესტირების ტექნოლოგია განვიხილოთ ვირტუალური ობიექტის, მაგალითად, ფინანსური ობიექტის, ბანკის მაგალითზე.

- დასატესტი პროგრამის პროექტის შექმნა: Visual Studio–ში საჭიროა ავირჩიოთ:

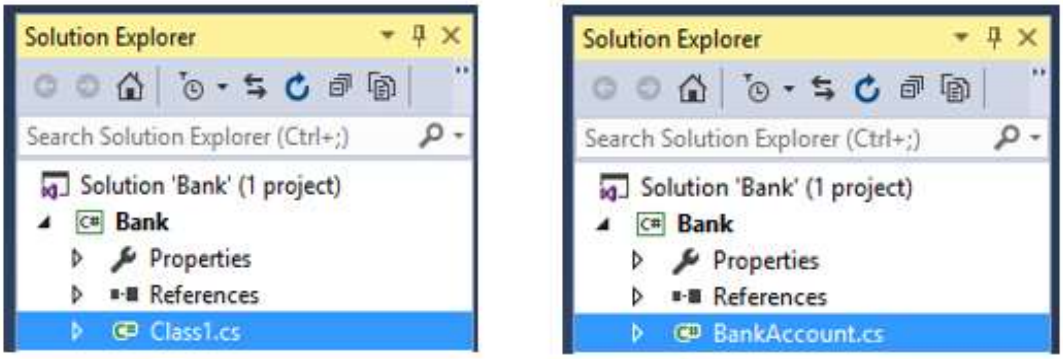
File -> New -> Project.

შედეგად გამოჩნდება დიალოგური ფანჯარა, სადაც ავირჩევთ:

Visual C# => ClassLibrary

პროექტი სახელით Bank.

მიიღება 6.63 ნახაზზე ნაჩვენები Solution Ex[plorer ფანჯარა. აქ Class1.cs სახელი შევცვალოთ BankAccount.cs -ით.



ნახ.6.63. Class1 - > BankAccount

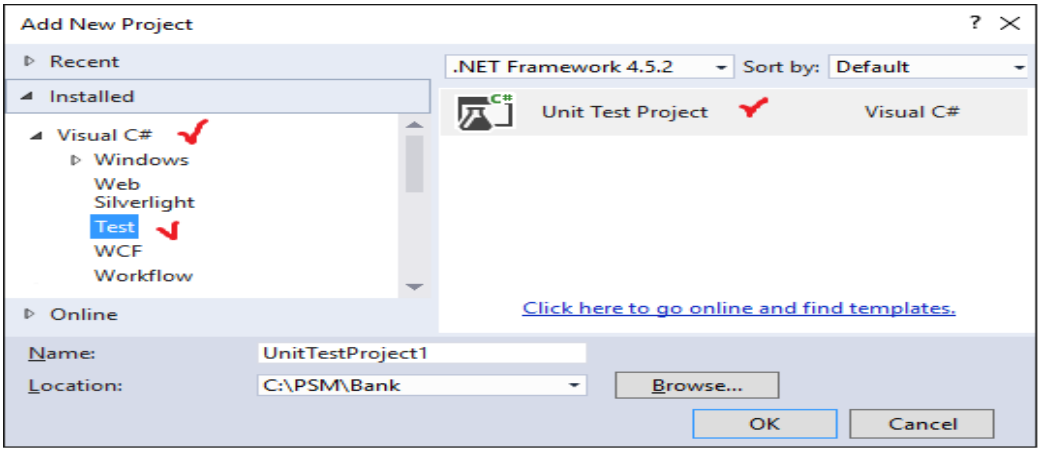
შემდეგ BankAccount.cs-ის ტექსტი რედაქტორის არეში შევცვალოთ ჩვენი დასატესტი პროგრამის კოდით.

ეს საწყისი ტექსტი, მაგალითად, მოცემულია 7.1 ლისტინგში.

```
//-- ლისტინგი_7.1 --- BankAccount.cs --------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BankAccountNS
{
 public class BankAccount
  {
    private string m_customerName;
    private double m_balance;
    private bool m_frozen = false;
   private BankAccount()
    {
    }
   public BankAccount(string customerName, double balance)
    {
      m_customerName = customerName;
      m_balance = balance;
    }
   public string CustomerName
    { get { return m_customerName; }
    }
   public double Balance
    { get { return m_balance; }
    }
   public void Debit(double amount)
    {
      if (m_frozen)
```

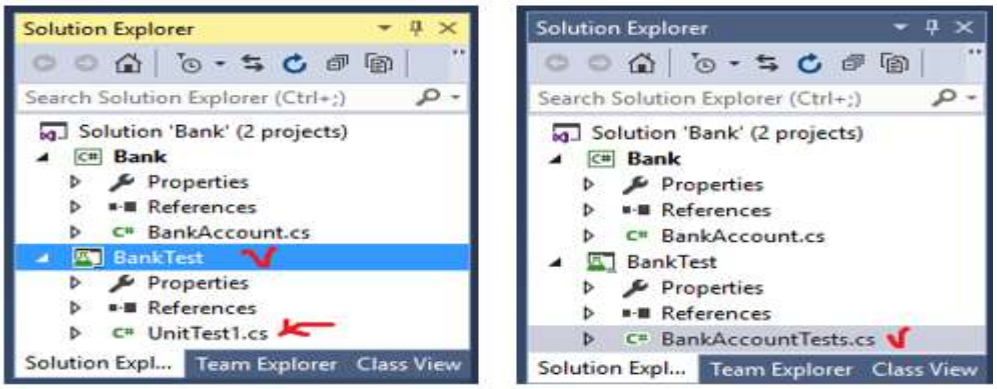
```
        {
          throw new Exception("Account frozen");
        }
       if (amount > m_balance)
        {
           throw new ArgumentOutOfRangeException("amount");
         }
        if (amount < 0)
         {
           throw new ArgumentOutOfRangeException("amount");
         }
         m_balance += amount; // განზრახ არასწორი კოდი
         // m_balance -= amount;   // გასწორებული
     }
    public void Credit(double amount)
     {
       if (m_frozen)
         {
          throw new Exception("Account frozen");
         }
       if (amount < 0)
        {
          throw new ArgumentOutOfRangeException("amount");
        }
        m_balance += amount;
     }
    private void FreezeAccount()
     {
        m_frozen = true;
     }
    private void UnfreezeAccount()
     {
        m_frozen = false;
     }
    public static void Main()
     {
      BankAccount ba = new BankAccount("Mr.Bryan Walton", 11.99);
      ba.Credit(5.77); ba.Debit(11.22);
      Console.WriteLine("Current balance is ${0}", ba.Balance);
     }
}}
```

- ტესტ-ფაილის პროექტის აგება (Unit Test Project):



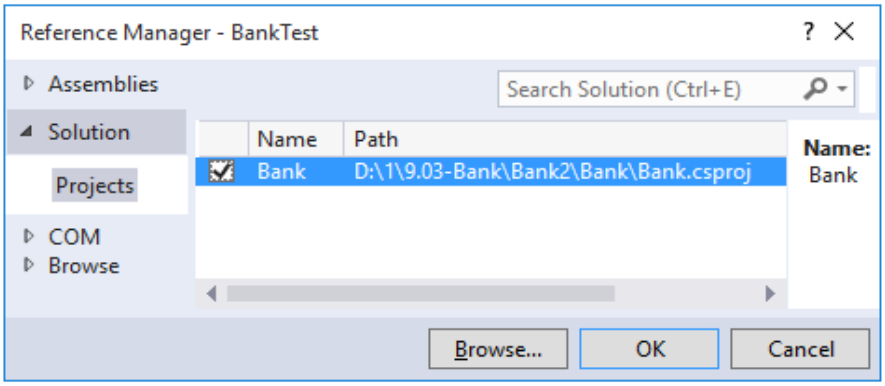
ნახ. 6.64. Unit Test პროექტის შექმნა

მივიღებთ 6.65 ნახაზზე ნაჩვენებ სურათს BankTest პროექტით. მარჯვენა სურათზე შეცვლილია UnitTest კლასის სახელი BankAccountTests-ით.



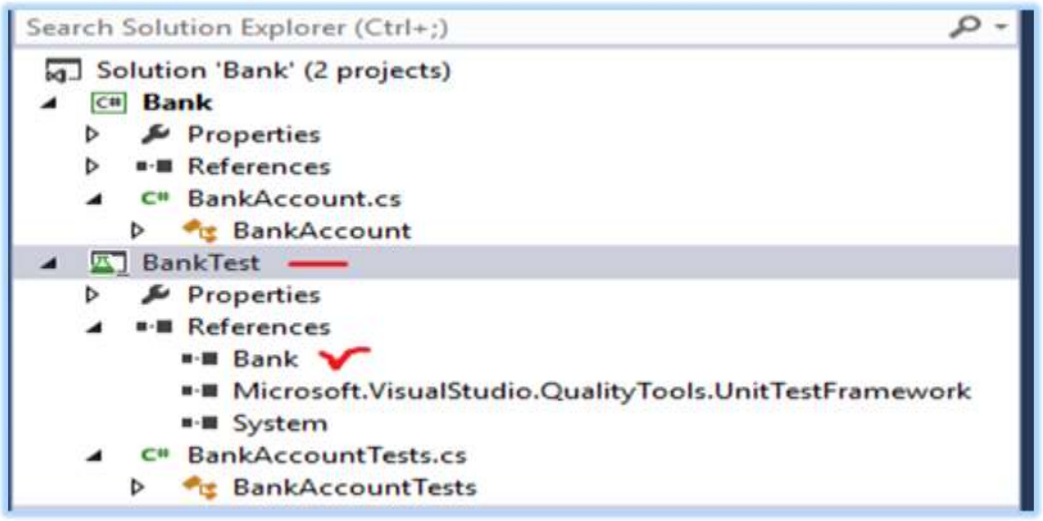
ნახ. 6.65

BankTests პროექტში დავამატოთ reference **Bank** solution-იდან. ამისათვის **BankTests**-ზე მაუსის მარჯვენა ღილაკით ავირჩიოთ **Add Reference** და მივიღებთ 6.66 ნახაზზე ნაჩვენებ ფანჯარას. აქ Solution სტრიქონში ვირჩევთ Projects და Bank-ის ჩეკბოქსს მოვნიშნავთ.



ნახ. 6.66

მივიღებთ 6.67 ნახაზზე ნაჩვენებ შედეგს.



ნახ. 6.67

ჩავამატოთ BankAccountTests პროგრამაში სახელსივრცე Bank-ის პროექტიდან: `using BankAccountNS;`

ამგვარად, BankAccountTests.cs ფაილს ექნება 7.2 ლისტინგზე ნაჩვენები სახე.

```
//-- ლისტინგი_7.2 -----  BankAccountTests.cs -----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
   [TestClass]
   public class BankAccountTests
   {
      [TestMethod]
      public void TestMethod1()
      {
      }
   }
}
```

ახლა შევქმნათ პირველი ტესტ - მეთოდი. ამ პროცედურაში, დაიწერება უნიტ - ტესტის მეთოდები BankAccount class-ის Debit მეთოდის ქცევის ვერიფიკაციისათვის. ეს მეთოდები ზემოთაა ჩამოთვლილი.

დასატესტი მეთოდების ანალიზის გზით გაირკვა, რომ საჭიროა მინიმუმ სამი ქცევის შემოწმება:

1) მეთოდი ქმნის ArgumentOutOfRangeException – გამონაკლისს, თუ კრედიტის ჯამი გადააჭარბებს ბალანსს;

2) იგი ქმნის ArgumentOutOfRangeException – გამონაკლისს მაშინაც, როცა კრედიტის ზომა უარყოფითია;

3) თუ 1 და 2 პუნქტები წარმატებით დასრულდა, მაშინ მეთოდი ითვლის ჯამს ბალანსის ანგარიშიდან.

პირველ ტესტში შევამოწმოთ, რომ კრედიტის დასაშვები მნიშვნელობისათვის (როცა დადებითი მნიშვნელობისაა და ბალანსის ანგარიშზე ნაკლებია) ანგარიშიდან მოიხსნება საჭირო თანხა.

- დავამატოთ BankAccountTests კლასს შემდეგი მეთოდი:

```
//---- unit test code -----
[TestMethod]
public void Debit_WithValidAmount_UpdatesBalance()
{
  // arrange
   double beginningBalance = 11.99;
   double debitAmount = 4.55;
   double expected = 7.44;
   BankAccount account = new BankAccount("Mr. Dito", beginningBalance);
   // act
   account.Debit(debitAmount);
   // assert
   double actual = account.Balance;
   Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly");
}
```

მეთოდი საკმაოდ მარტივია. ჩვენ ვქმნით ახალ BankAccount ობიექტს საწყისი ბალანსით და შემდეგ ვაკლებთ სწორ ოდენობას. ჩვენ ვიყენებთ Microsoft-ის unit-ტესტის ფრეიმვორკს მართვადი კოდის AreEqual მეთოდისათვის, რათა მოხდეს საბოლოო ბალანსის ვერიფიკაცია - არის ის, რასაც ჩვენ ველით.

ტესტ-მეთოდის მოთხოვნები ასეთია:

1) მეთოდი მონიშნული უნდა იყოს [TestMethod] ატრიბუტით;
2) მეთოდმა უნდა დააბრუნოს void;
3) მეთოდს არ შეიძლება ჰქონდეს პარამეტრები.

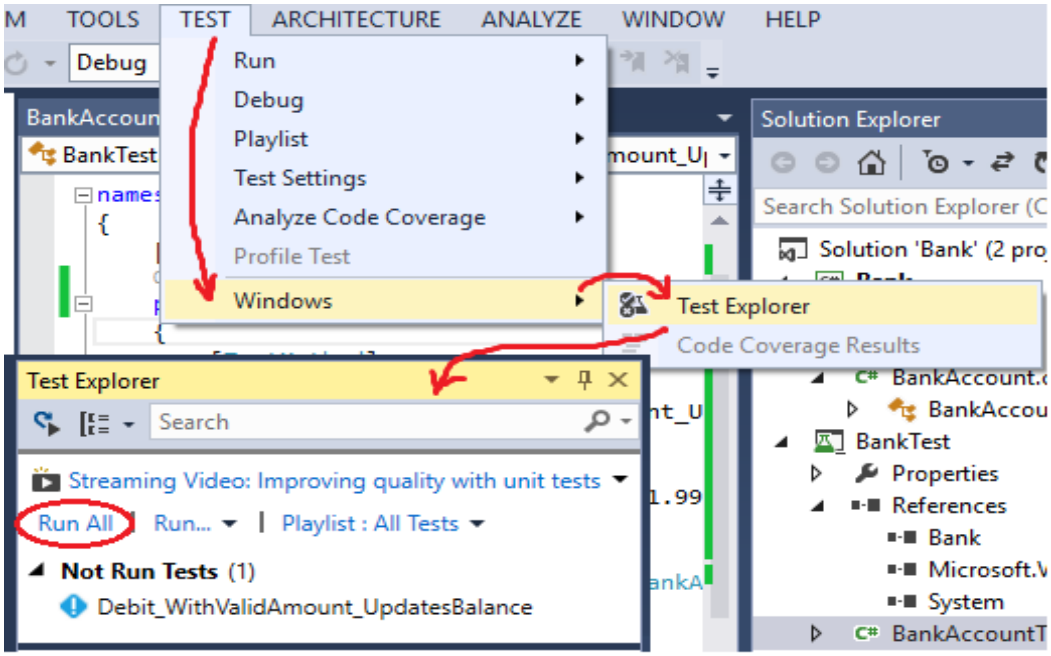
- ტესტის აგება და ამუშავება:

მთლიანი ტესტის კოდი მოცემულია 7.3 ლისტინგში.

```
//-- ლისტინგი_7.3  ----- BankAccountyTests.cs ----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
  [TestClass]
  public class BankAccountTests
  {
    [TestMethod]
    public void Debit_WithValidAmount_UpdatesBalance()
    {
      // arrange
      double beginningBalance = 11.99;
      double debitAmount = 4.55;
      double expected = 7.44;
      BankAccount account = new BankAccount("Mr. Dito",
                                beginningBalance);
      // act
      account.Debit(debitAmount);

      // assert
      double actual = account.Balance;
      Assert.AreEqual(expected, actual, 0.001, "Account
                          not debited correctly");
    }
  }
}
```

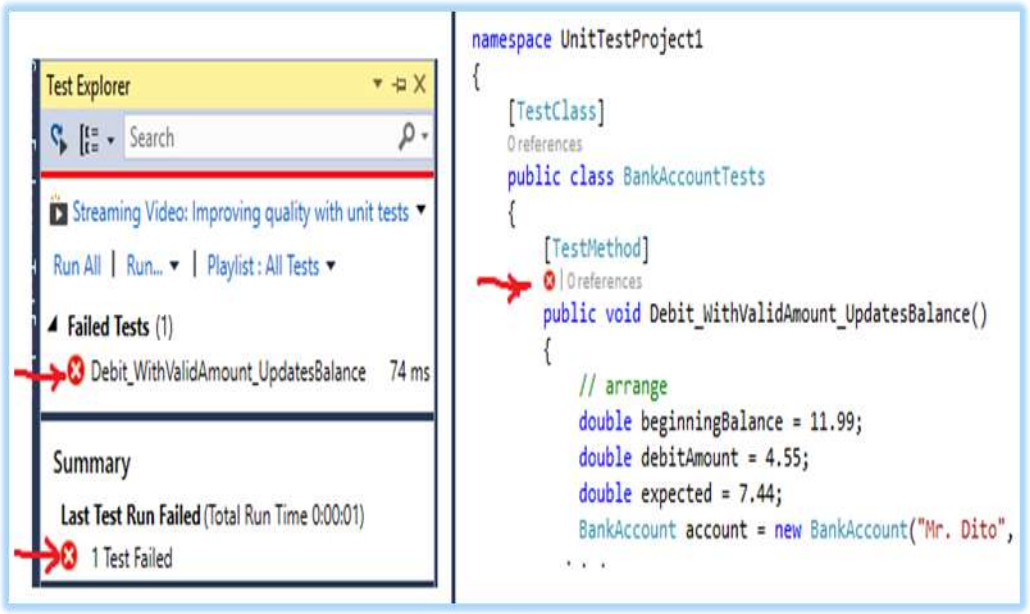
- BUILD მენიუდან ვირჩევთ Build Solution;

- TEST მენიუდან ვირჩევთ Windows და Test Explorer პუნქტებს. იხსნება Test Explorer ფანჯარა (ნახ.6.68).



ნახ.6.68

აქ ვირჩევთ Run All - სა და ვიღებთ შედეგს (ნახ.6.69).



ნახ.6.69

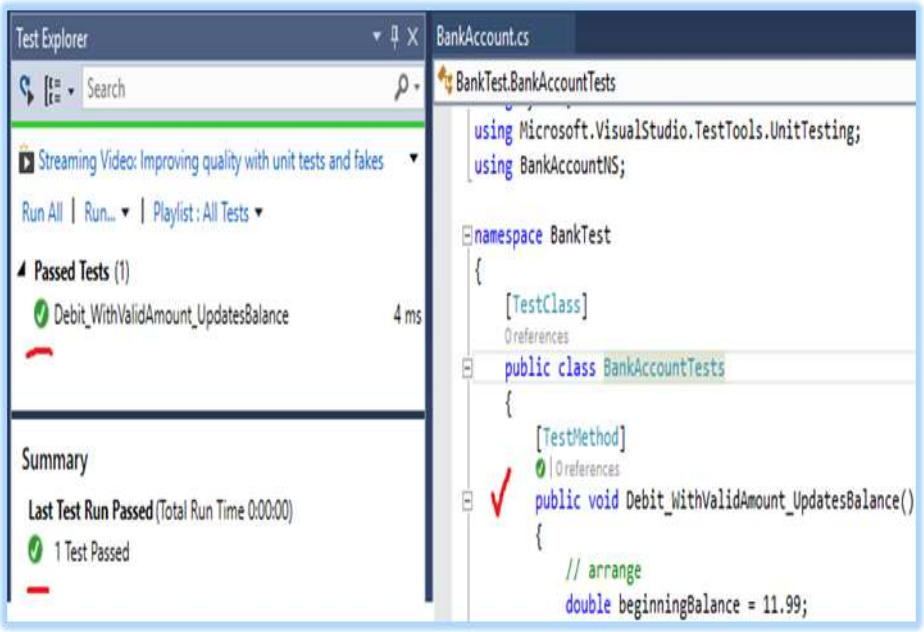
ნახაზზე მითითებული „x“-სიმბოლოები წითელ წრეშია მოთავსებული, ე.ი. ტესტირებამ აღმოაჩინა შეცდომები და კოდი წარმატებით ვერ შესრულდა.

თუ მეთოდი წარმატებით ჩაივლიდა, მაშინ მივიღებდით მწვანე ფერის x-სიმბოლოებს.

შემდეგი ეტაპი კოდის გასწორება და ხელახალი ტესტირებაა. დასატესტ პროგრამაში შევცვალოთ სტრიქონში „ + „ ნიშანი „ – „ -ით.

```
// m_balance += amount; // intentionally incorrect code
m_balance -= amount; // intentionally correct code
```

ტესტის თავიდან ამუშავებით ვიღებთ წარმატებულ შედეგს ანუ მიიღება მწვანე ფერის სიმბოლოები (ნახ.6.70).



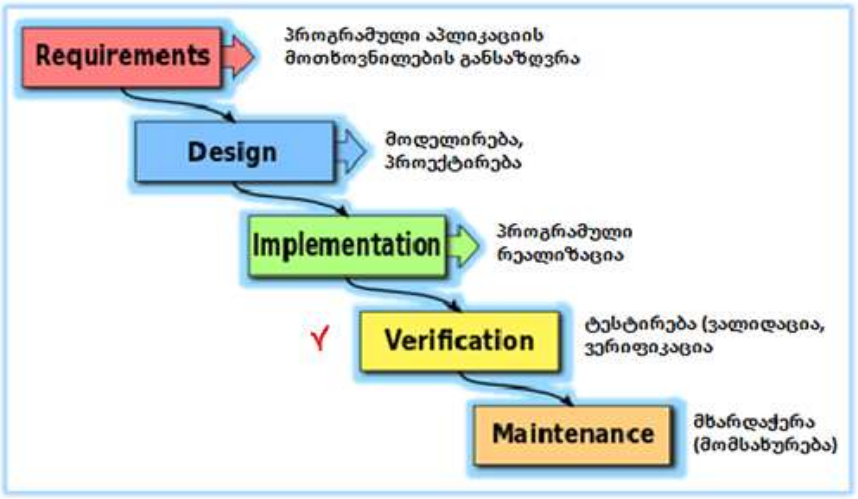
ნახ.6.70. სწორი შედეგი

დავალება:

1. შექმენით ახალი პროექტი WinFormsApp_Test, რომელიც იქნება დასატესტი პროგრამა;
2. ჩაწერეთ პროექტის პროგრამულ ნაწილში დასატესტი კოდი (განხილული ამოცანის შესაბამისად) - BankAccount.cs;
3. ააგეთ ტესტ-ფაილის პროგრამის პროექტი (Unit Test Project);
4. მოაწესრიგეთ BankAccountTests კოდი შესაბამისი სახელსივრცისა და Reference-ს დამატებით;

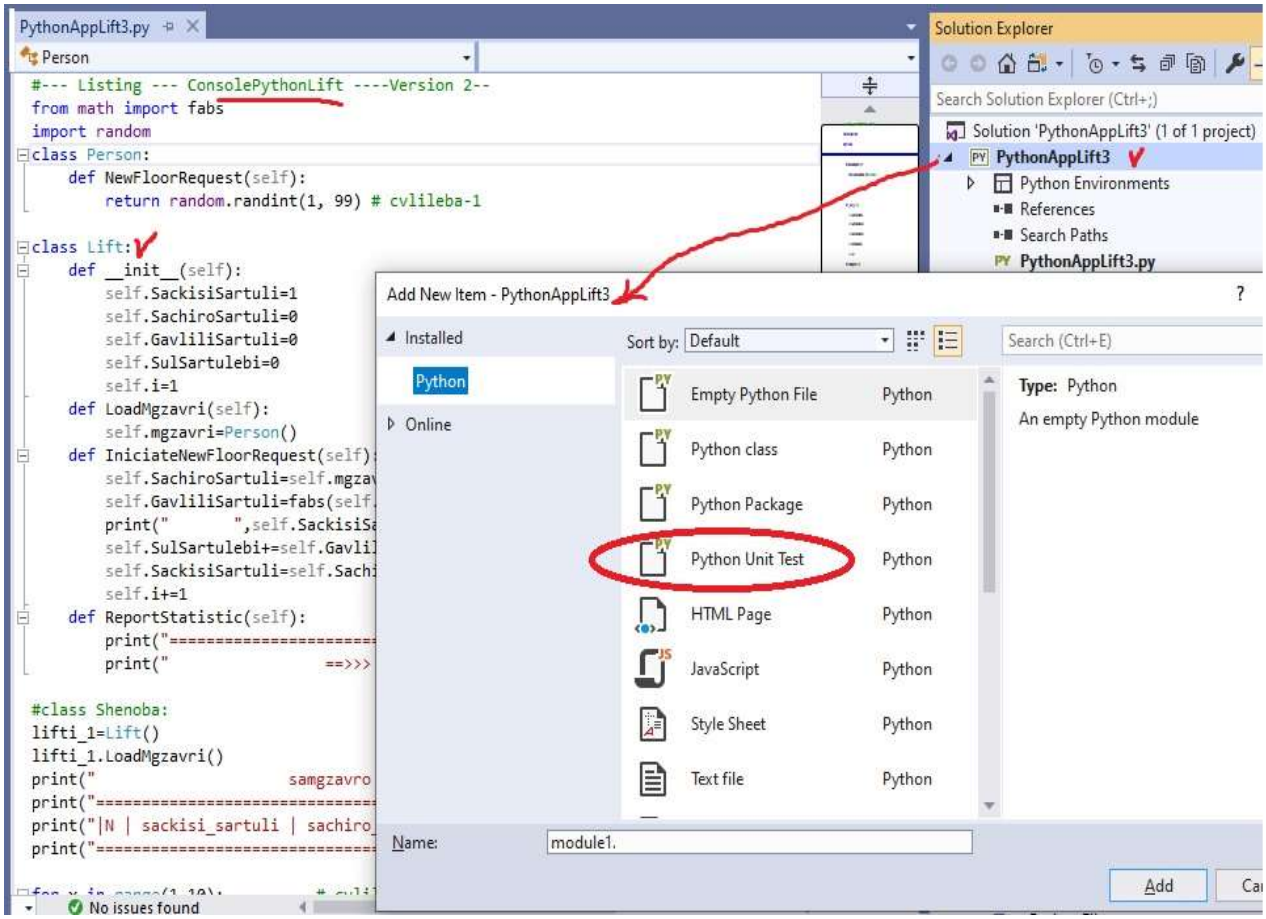
6.6.2. ლაბ-N9: Python პროგრამული მოდულების ტესტირება

6.71 ნახაზზე ნაჩვენებია ტესტირების ეტაპის ადგილი პროგრამული აპლიკაციის შექმნის სასიცოცხლი ციკლის „ჩანჩქერულ“ მოდელში.



ნახ. 6.71

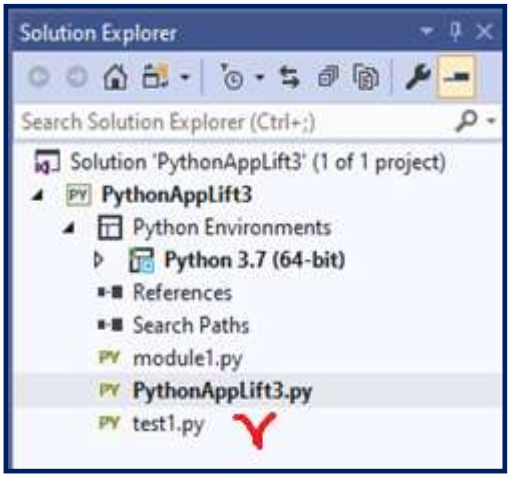
განვიხილოთ Python-ის პროგრამისთვის (აპლიკაცია) Unit ტესტირების საკითხი მართვის ამოცანის მაგალითზე (პარაგაფი 6.3, ლიფტის მართვის ამოცანა). 6.72 ნახაზზე ნაჩვენებია Visual Studio.NET პლატფორმაზე აგებული PythonAppLift3 პროექტის ტესტირების პროცესის დასაწყისი. გავხსნათ პროგრამის პროექტი (ნახ.6,72) [12].



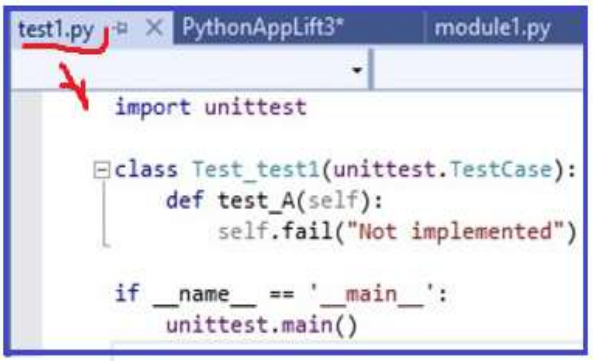
ნახ. 6.72

როგორც ნახაზიდან ჩანს, თავიდან საჭიროა Solution Explorer-ში PythonAppLift3 პროექტზე მაუსის მარჯვენა ღილაკით Add New Item -ის გამოძახება და Python Unit Test -ის არჩევა.

Solution Explorer-ში გამოჩნდება ახალი შექმნილი test1.py ფაილი (ნახ.6.73-ა). მისი ტექსტი ნაჩვენებია 6,73-ბ ნახაზზე.



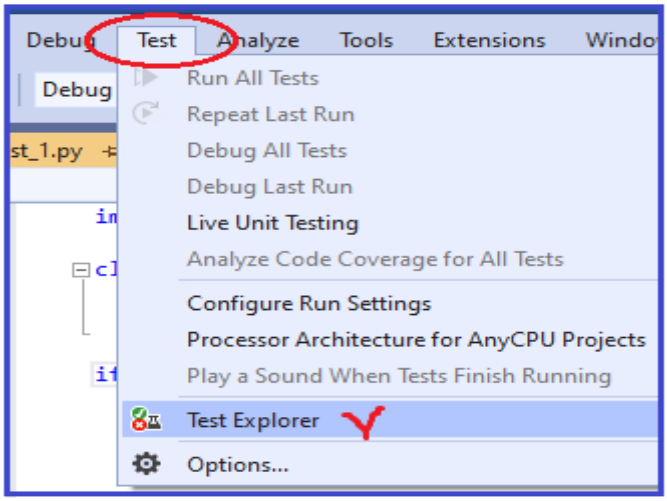
ნახ.6.73-ა



ნახ.6.73-ბ

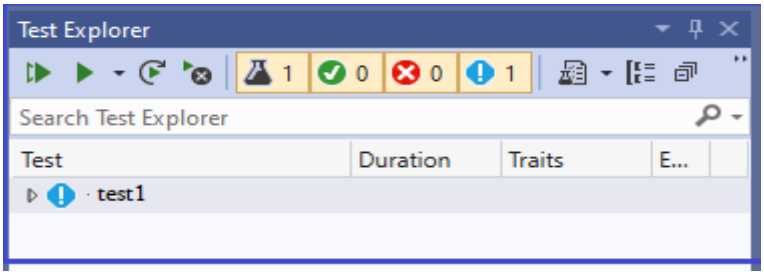
test1.py ფაილის კოდი იმპორტირებას უკეთებს unittest სტანდარტულ მოდულს, ქმნის ტესტურ კლასს unittest.TestCase-დან და იძახებს unittest.main().

მომდევნო ბიჯზე Visual Studio.NET-ის მთავარ მენიუდან ავირჩიოთ: Test -> Test Explorer



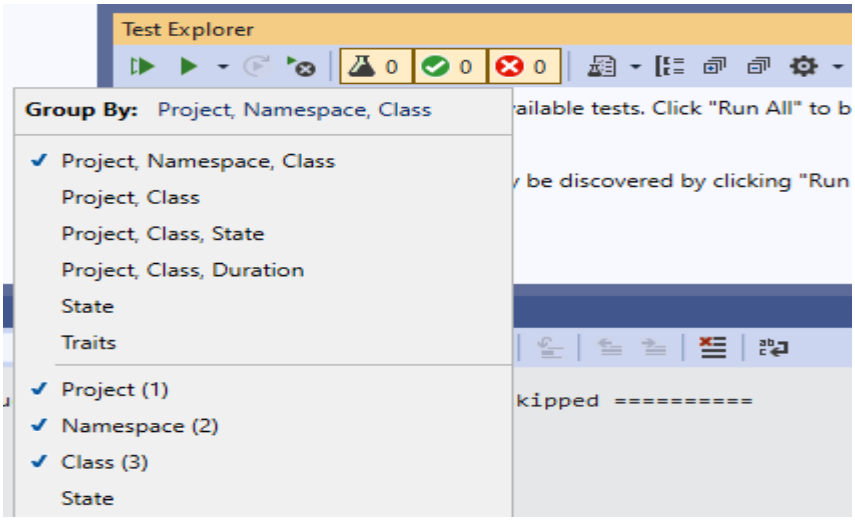
ნახ. 6.74

Test Explorer -ის დანისნულებაა აპლიკაციის პროექტში ტესტების მოძიება და ეკრანზე ასახვა (ნახ. 6.75). მაუსით ტესტზე (test1) ორჯერ დაწკაპუნებით იხსნება მისი საწყისი ფაილი.



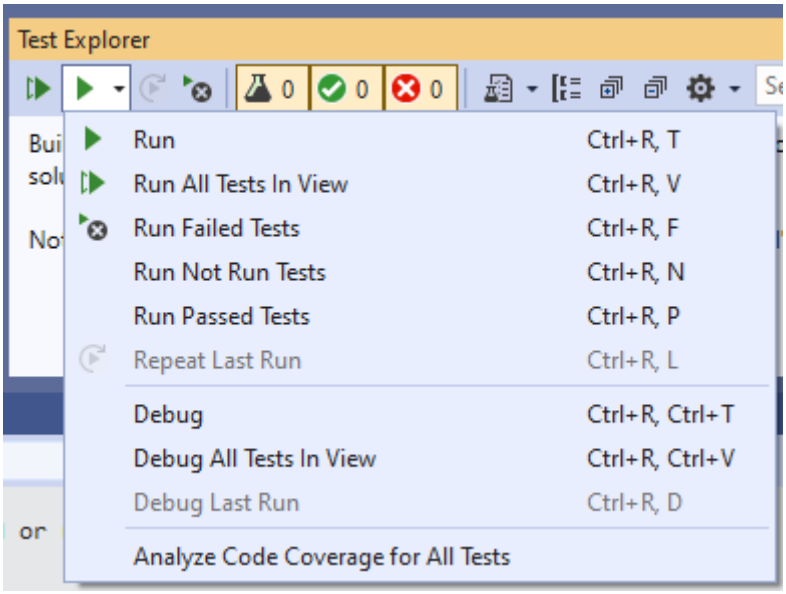
ნახ. 6.75

ტესტების დამატების შემთხვევაში (თუ რამდენიმე ტეასტია აპლიკაციისთვის), მაშინ შესაძლებელია მენიუდან მათი მოწესრიგების ფუნქციის “Group By” გამოყენება (ნახ 6.77).



ნახ. 6.77

ტესტის ამუშავებისათვის გამოიყენება ისევ Test Explorer. 6.78 ნახაზზე ნაჩვენებია ეს შესაძლებლობები.

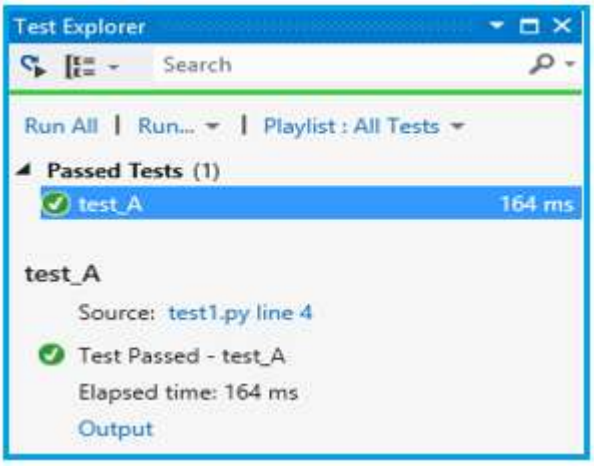


ნახ. 6.78. ტესტის ამოქმედება

- **Run** – ჯგუფში წარუმატებლად გავლილი ან გაუვლელი ტესტების ამუშავების ბრძანებების მოწოდება;
- **Run All** – მკაფიოდ აამოქმედებს ყველა ნაჩვენებ ტესტს (ფილტრების გათვალისწინებით);
- **Run Selected Tests** – მაუსის მარჯვენა ღილაკით შეიძლება ამოირჩეს ერთი ან რტამდენიმე ტესტი, რომელთა ტესტირებაც გვჭირდება.

ტესტების მუშაობის შემდეგ ეკრანზე აისახება მწვანე „ალამი“ და ტესტირების დრო (ნახ.6.79).

წარუმატებლად დასრულებული ტესტები აისახება წითელი ფერის „ჯვრით“ (ნახ.6.80).



ნახ.6.79

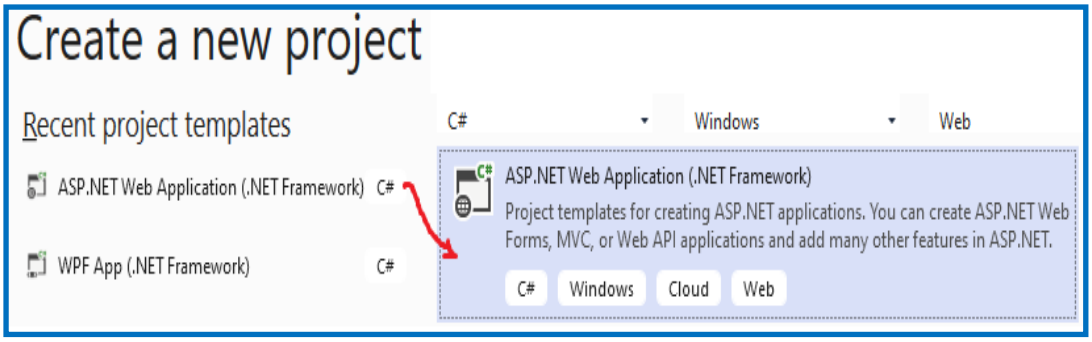
ნახ.6.80

7. პრაქტიკული სამუშაოს შესრულების ნიმუშები (ნაწ.2)

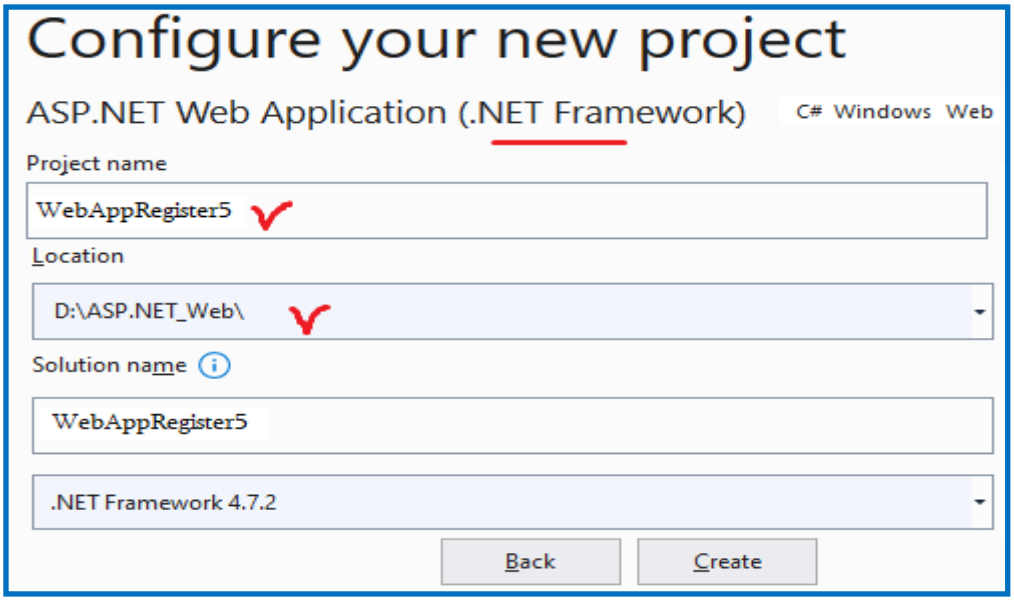
7.1. ლაბ-N10: ASP.NET Web აპლიკაცია – ინტერაქტიული Web-გვერდის შექმნა

ამოცანა: Ms Visual Studio .NET პლატფორმაზე ASP.NET-ის გამოყენებით ახალი Web-გვერდის აგება, რომელზეც მომხმარებელი შეიტანს საკუთარ მონაცემებს და გადააგზავნის სერვერზე [1,6-8].

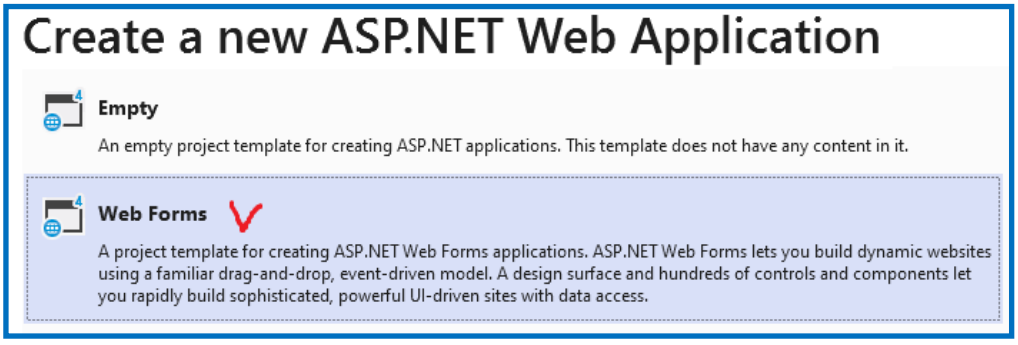
– შევქმნათ ახალი ASP.NET აპლიკაცია პროექტის სახელით WebAppRegister5 (ნახ.7.1, 7.2-ა,ბ);



ნახ.7.1. ახალი ASP.NET პროექტის შექმნა .NET Framework პლატფორმაზე

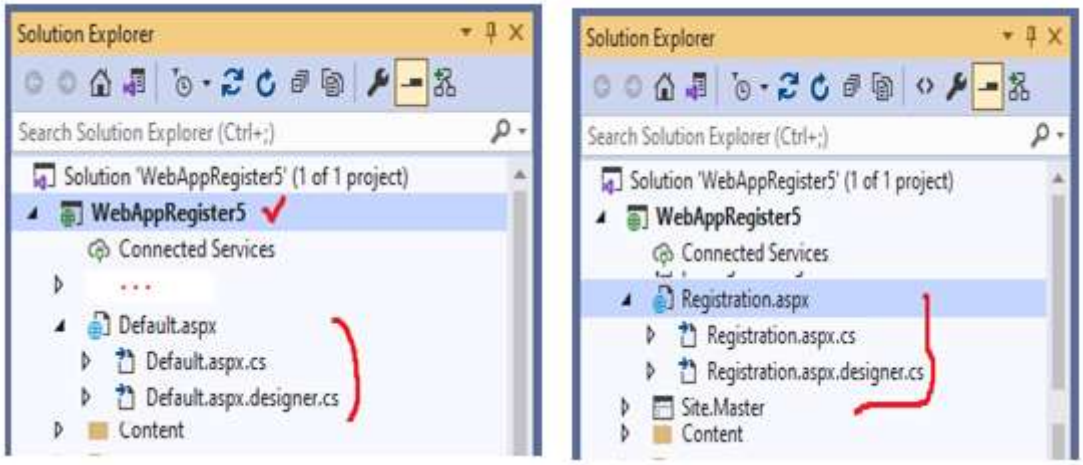


ნახ.7.2-ა



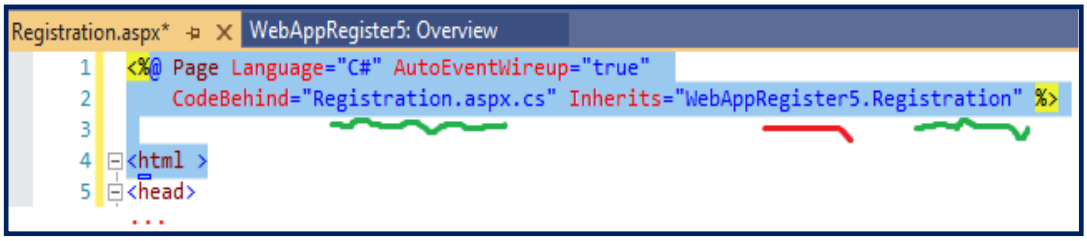
ნახ.7.2-ბ

– მიიღება Solution Explorer (ნახ.7.3 - მარცხენა). შევცვალოთ Default სახელი ახლით, კერძოთ, Registration-ით (ნახ.7.3 - მარჯვენა);



ნახ.7.3

– Registration.aspx ფაილის 1-ელ სტრიქონს ექნება 7.3 ნახაზზე ნაჩვენები სახე, სადაც ასახულია პროექტის და .aspx და .aspx.cs პროგრამების სახელები;

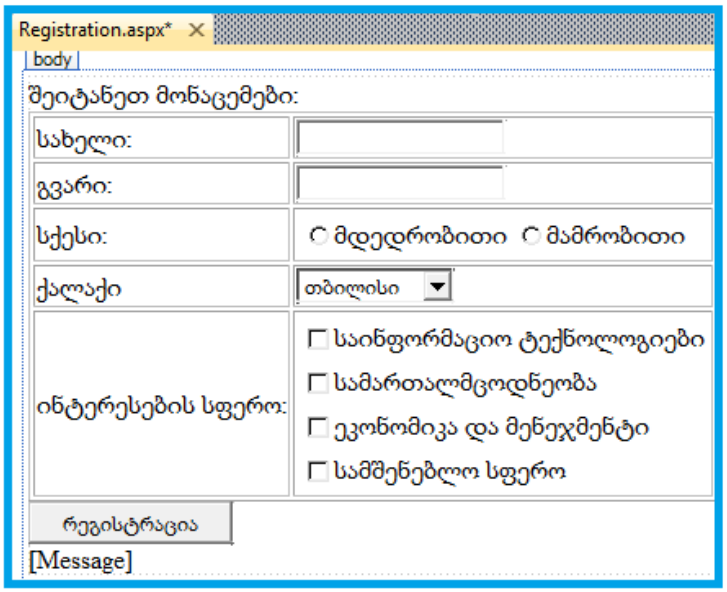


ნახ.7.4

– Registration.aspx.cs ფაილის საწყისი ტექსტის ლისტინგი ასე გამოიყურება;

```
// ----- ლისტინგი_7.1  Registration.aspx.cs.------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace WebAppRegister5
{
   public partial class Registration : Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
      }
   }
}
```

– Web-გვერდის სარეგისტრაციო ფორმის მაკეტი ნაჩვენებია 7.5 ნახაზზე.



ნახ.7.5

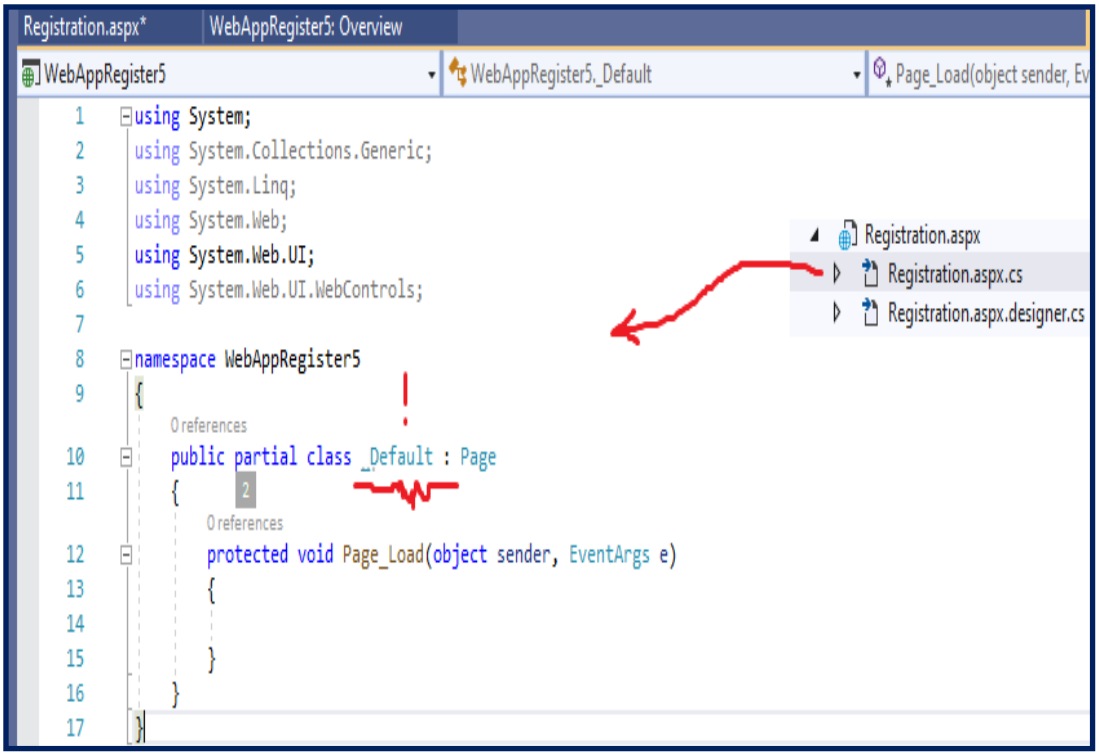
ფორმაზე მოთავსებულია სერვერული მართვის ელემენტები form, asp:TextBox, asp:DropDownList, asp:CheckBoxList, asp:Button, asp:Label და ა.შ., რომლებიც ასახულია Registration.aspx ფაილის 7.2 ლისტინგში. გავხსნათ Registration.aspx და შევიტანოთ შემდეგი კოდი (ან გამოვიყენოთ წიგნიდან მისი შესაბამისი პროტოტიპი):

```
<!— ლისტინგი_7.2 ———— –>
    <%@ Page Language="C#" AutoEventWireup="true"
      CodeBehind="Registration.aspx.cs" Inherits="WebAppRegister8.Registration" %>
    <html >
    <head>
     <title>რეგისტრაციის ფორმა</title>
      <style type="text/css">
         .auto-style1 {
            width: 253px;
         }
      </style>
    </head>
     <body>
      <form method="post" runat="server" id="registration">
      შეიტანეთ მონაცემები:
      <table border="1">
       <tr>
        <td>სახელი:</td>
        <td class="auto-style1">
         <asp:TextBox id="FirstName" runat="server"></asp:TextBox></td>
       </tr>
       <tr>
        <td>გვარი:</td>
        <td class="auto-style1">
         <asp:TextBox id="LastName" runat="server"></asp:TextBox></td>
       </tr>
```

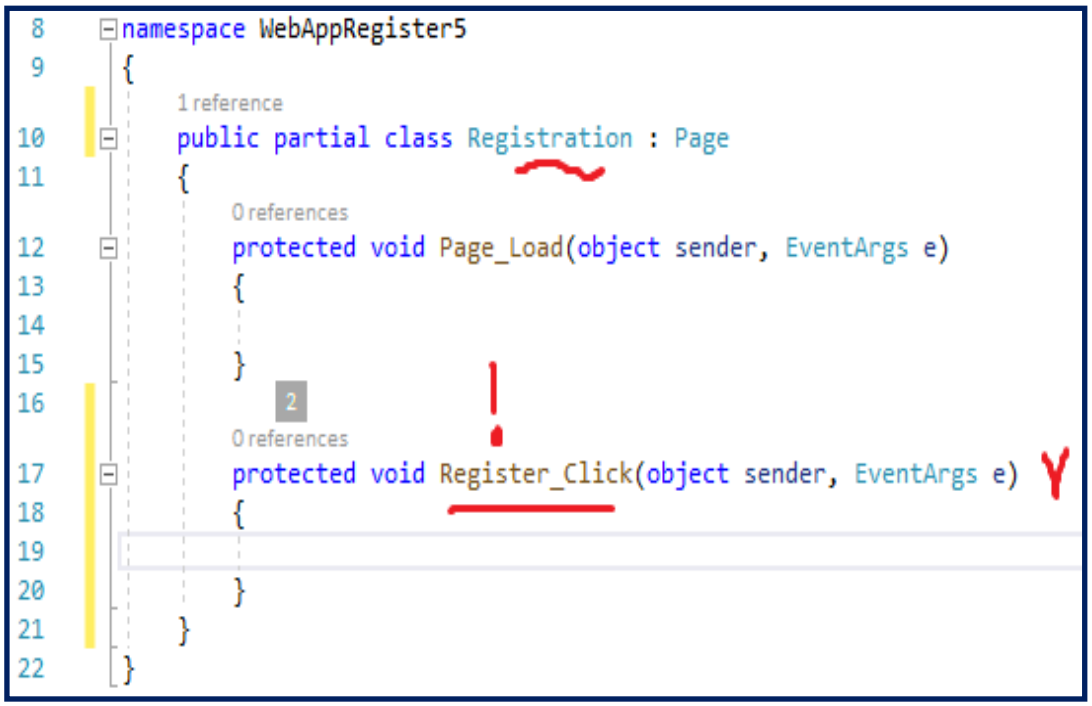
```
<tr>
 <td>სქესი:</td>
 <td class="auto-style1"><asp:RadioButtonList id="Sex" runat="server" RepeatDirection ="Horizontal">
  <asp:ListItem Value="მდედრობითი"></asp:ListItem>
  <asp:ListItem Value="მამრობითი"></asp:ListItem>
 </asp:RadioButtonList></td>
</tr>
<tr>
 <td>ქალაქი</td>
 <td class="auto-style1"><asp:DropDownList id="City" runat="server">
  <asp:ListItem Value="თბილისი"></asp:ListItem>
  <asp:ListItem Value="ქუთაისი"></asp:ListItem>
  <asp:ListItem Value="რუსთავი"></asp:ListItem>
  <asp:ListItem Value="გორი"></asp:ListItem>
  <asp:ListItem Value="ბათუმი"></asp:ListItem>
  <asp:ListItem Value="თელავი"></asp:ListItem>
 </asp:DropDownList></td>
</tr>
<tr>
    <td>ინტერესების სფერო:</td>
 <td class="auto-style1">
  <asp:CheckBoxList id="Interests" runat="server">
  <asp:ListItem Value="საინფორმაციო ტექნოლოგიები"></asp:ListItem>
  <asp:ListItem Value="სამართალმცოდნეობა"></asp:ListItem>
  <asp:ListItem Value="ეკონომიკა და მენეჯმენტი"></asp:ListItem>
  <asp:ListItem Value="სამშენებლო სფერო"></asp:ListItem>
 </asp:CheckBoxList></td>

</tr>
</table>
<asp:Button id="Register" runat="server" Text="რეგისტრაცია" OnClick="Register _Click"></asp:Button>
<br />
<asp:Label id="Message" runat="server"></asp:Label>
</form>
</body>
</html>
```

– Web-გვერდზე „რეგისტრაცია“ Button-ის დაკლიკვით უნდა გადავიდეთ C# კოდში... მაგრამ შეიძლება ეს ვერ მოხერხდეს... რადგანაც .aspx და .aspx.cs ფაილებს შორის კავშირი არაა გასწორებული... კერძოდ Registration.aspx.cs გადასვლით (ნახ. 7.6-ა,ბ) ჩანს რომ მე-10 სტრიქონში დარჩენილია საწყისი კლასის სახელი _Default : Page.

Web-გვერდის ჩატვირთვის და მონაცემების შევსების შემდეგ „რეგისტრაცია“ Button-ის დაჭერისას გამოიძახება OnClick მოვლენაზე მიბმული მეთოდი Register_Click. ის აღიწერება C# კოდში, რომლის 7.3 ლისტინგი მოცემულია ქვემოთ. გავხსნათ Registration.aspx.cs ფაილი და შევიტანოთ შემდეგი კოდი:

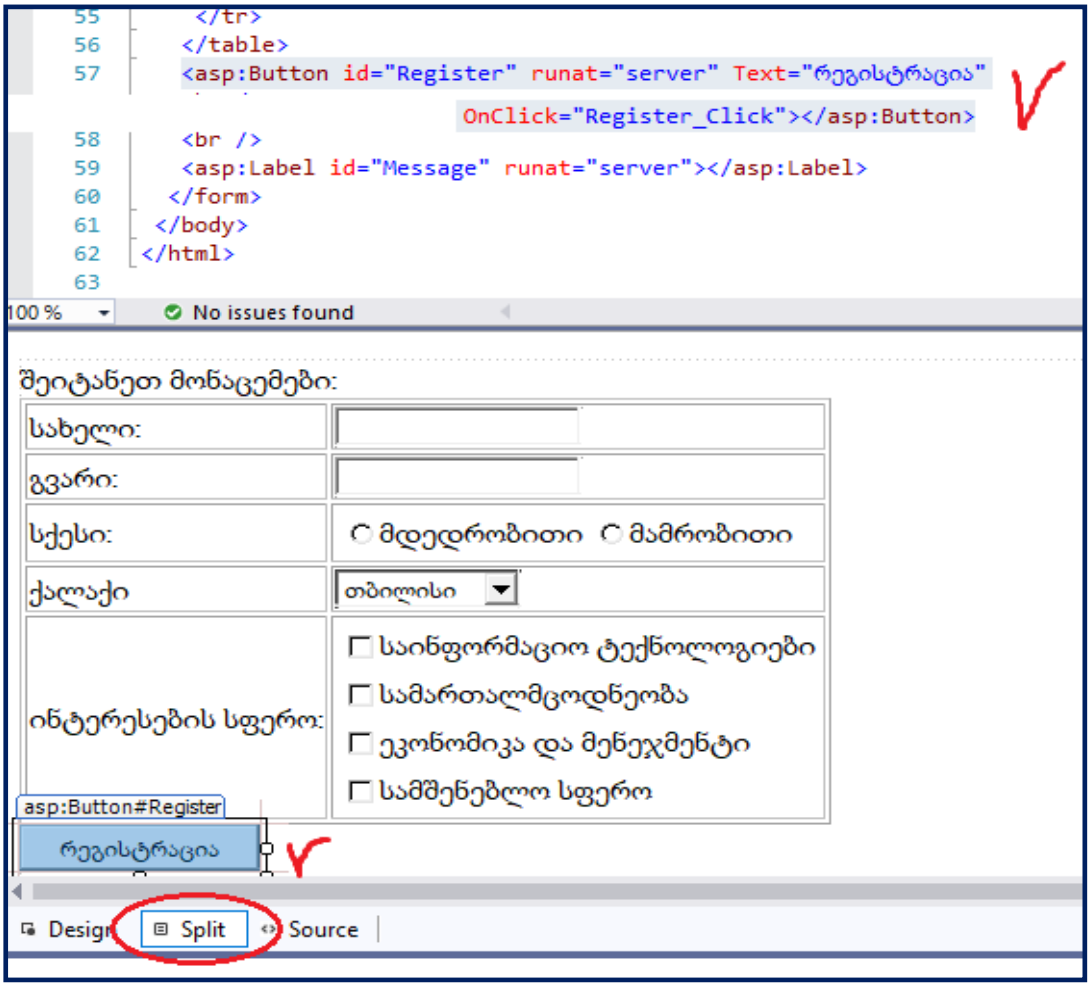


ნახ.7.6-ა. _Default -ის შეცვლა Registration-ით



ნახ.7.6-ბ. შეცვლის შედეგი

ამის შემდეგ Registration.aspx დიზაინის „რეგისტრაცია“ ღილაკი ამუშავდება. Split რეჟიმში ჩანს ფორმის დიზაინიც და შესაბამისი .aspx პროგრამის ტექსტის ფრაგმენტიც (ნახ.7.7). აქ გამუქებულია <asp:Button id …>.



ნახ.7.7.

– „რეგისტრაცია“ ღილაკის ამოქმედებით სისტემა გადაგვიყვანს C# -ის კოდში, Registracion.aspx.cs-ის „Register_Click“ - მოვლენაზე. აქ უნდა ჩავამატოთ ლოგიკის ტექსტი, რომელიც 7.3 ლისტინგშია აღწერილი.

```
// ── ლისტინგი_7.3 ──────────
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
    namespace WebAppRegister5
    {
       public partial class Registration : Page
       {
          protected void Page_Load(object sender, EventArgs e)
          {

          }

          protected void Register_Click(object sender, EventArgs e)
          {
```

```
            System.Text.StringBuilder sb = new System.Text.StringBuilder();
            sb.Append("თქვენი გადაცემული მონაცემები:<br>");
            sb.AppendFormat("სახელი: {0}<br>", FirstName.Text);
            sb.AppendFormat("გვარი: {0}<br>", LastName.Text);
            sb.AppendFormat("სქესი: {0}<br>", Sex.SelectedValue);
            sb.AppendFormat("ქალაქი: {0}<br>", City.SelectedValue);
            sb.Append("ინტერესები: ");

            foreach (ListItem item in Interests.Items)
            {
               if (item.Selected)
                  sb.AppendFormat("{0}, ", item.Value);
            }
            sb.Append("<br>გმადლობთ რეგისტრაციისთვის");
            Message.Text = sb.ToString();
         }
      }
   }
```

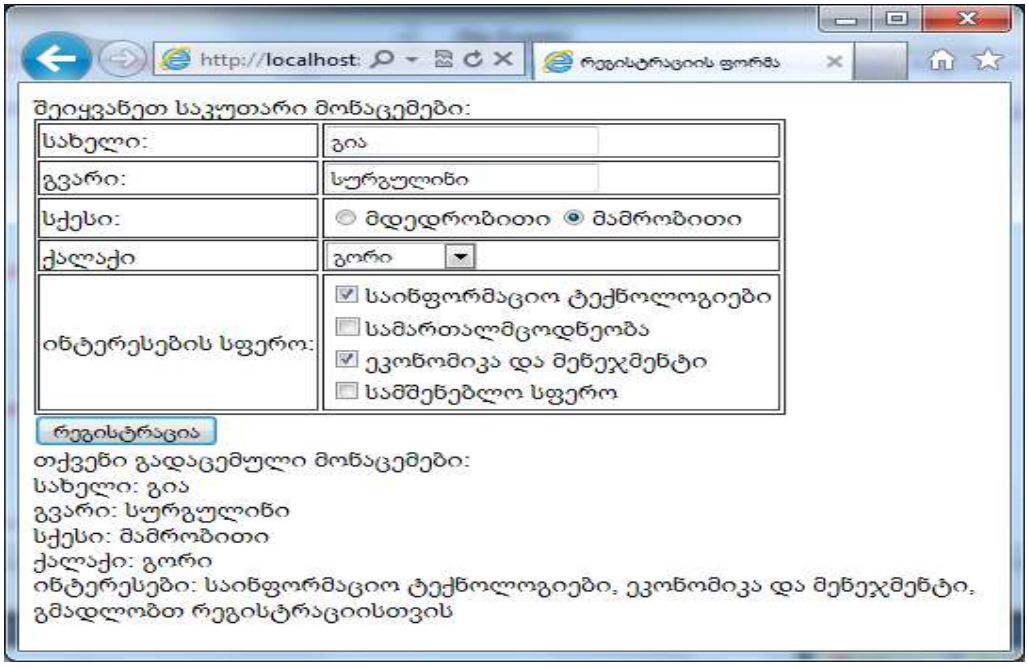
– ავამუშავოთ პროგრამა შესრულებაზე;

– Web-გვერდი, მომხმარებლის მიერ შეტანილი მონაცემებით, „რეგისტრაცია“ ღილაკზე დაკლიკვის შემდეგ, შეასრულებს C# კოდის Register_Click -ში ჩაწერილ ოპერაციებს.

სინტაქსური და სისტემური შეცდომების არარსებობის შემთხვევაში შედეგებს ექნება 7.8 ნახაზზე მოცემული სახე.

სინტაქსური შეცდომების (Errors) არსებობის დროს ისინი უნდა გავასწოროთ და შევამოწმოთ მუშაობისუნარიანობა.

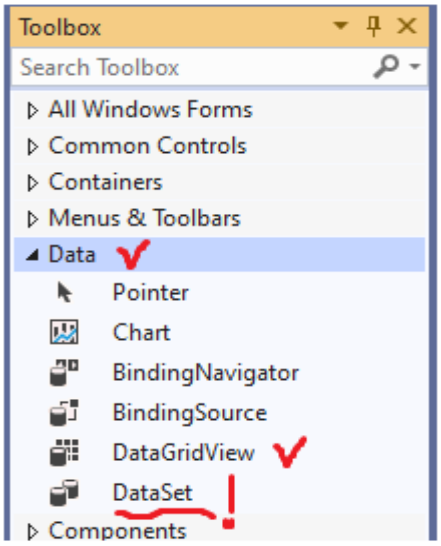
ზოგჯერ შეიძლება იყოს ლოგიკური ან სისტემური შეცდომები, რაც პროგრამის შინაარსში ან სისტემური ცდომილების გარკვევას მოითხოვს.



ნახ.7.8. სწორი შედეგით დასრულება

7.2. ლაბ-11: Ms SQL Server-თან სამუშაოდ C# და DataGridView კლასით პროგრამული აპლიკაციის აგება (DataSet-ის გამოყენება)

Visual Studio.NET პლატფორმაზე C# ენაზე დაწერილი მომხმარებლის ინტერფეისის სამუშაოდ SQL Server ბაზასთან იყენებენ DataGridView კლასს. იგი ინსტრუმენტების პანელზე Data-ში მოთავსებული მართვის ელემენტია, რომელშიც შესაძლებელია მონაცემთა ბაზის ცხრილ(ებ)იდან საჭირო სვეტების (Columns) მონაცემთა გადმოტანა ეკრანზე (ნახ.7.9).



ნახ.7.9

ინტერფეისისა და ბაზის დასაკავშირებლად კი საჭიროა ADO.NET დრაივერის DataSet ობიექტის გამოყენება. იგი უზრუნველყოფს სერვერიდან ბაზის ნაწილის (რომელიც მომხმარებელს სჭირდება) გადმოტანას მის კომპიუტერზე.

გამოყოფილი ბაზის ნაწილი, მაგალითად რომელიმე ცხრლი (Table) შეუძლია მომხმარებელს გადაამუშაოს (შეცვალოს, ჩაამატოს, წაშალოს და ა.შ.) და ბოლოს განახლებული ვერსია დააბრუნოს სერვერზე. მთელი ამ პროცესის განმავლობაში ის ხელს არ უშლის სხვა მომხმარებლებს, ანუ სერვერის ბაზას არ ბლოკავს.

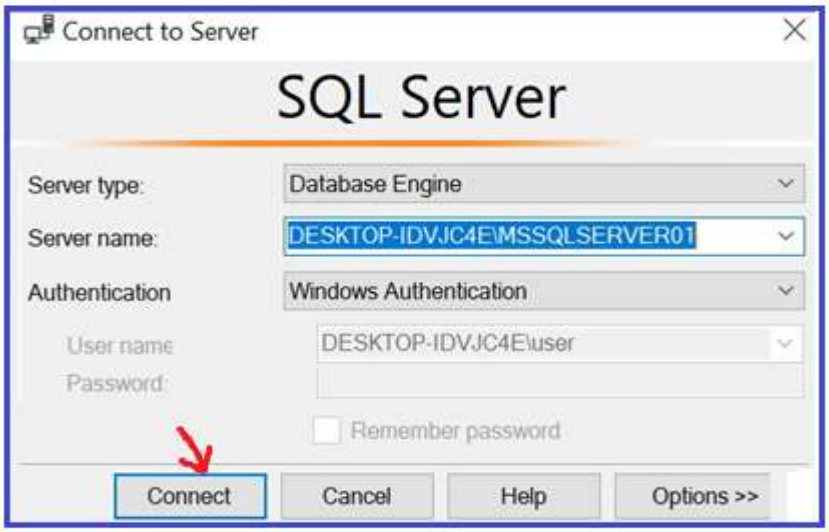
ცალკევებული DataSet გამოიყენება ბაზის მონაცემებთან წვდომისათვის უწყვეტი კავშირის გარეშე და იძლევა მონაცმთა მოდიფიცირების შესაძლებელობას მხოლოდ კლიენტის მხარეს.

განცალკევებული (არადაკავშირებული) DataSet ნიშნავს, რომ მონაცემთა წყაროსთან მონაცემების წვდომისთვის არ არის საჭირო უწყვეტი კავშირი. ამ დროს, მონაცემთა ნაკრების კლასი მონაცემებს ინახავს კლიენტის მანქანაში.

განვიხილლოთ პროგრამული აპლიკაციის აგება ასეთი ამოცენისთვის.

SQL Server Management Studio-ს საშუალებით შევქმნათ მონაცემთა წყაროდ (DataSource) მარტივი მონაცემთა ბაზა, ხოლო აპლიკაცია შევასრულოთ Visual Studio.NET - ში.

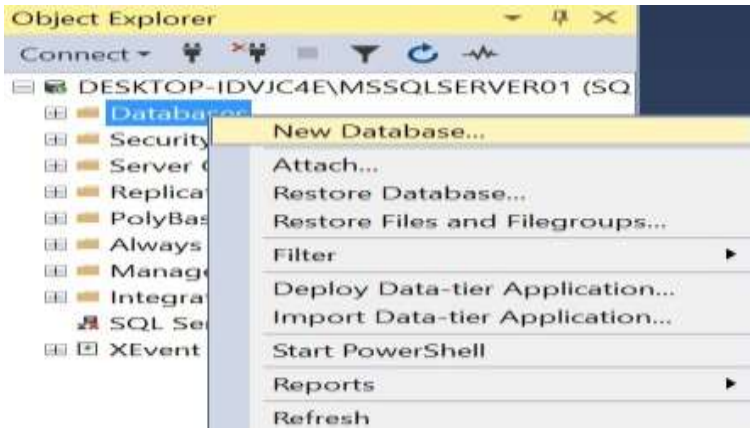
- შევქმნათ ახალი ბაზა და ცხრილი ჩვენი ექსპერიმენტისთვის (ნახ.7.10).



ნახ.7.10

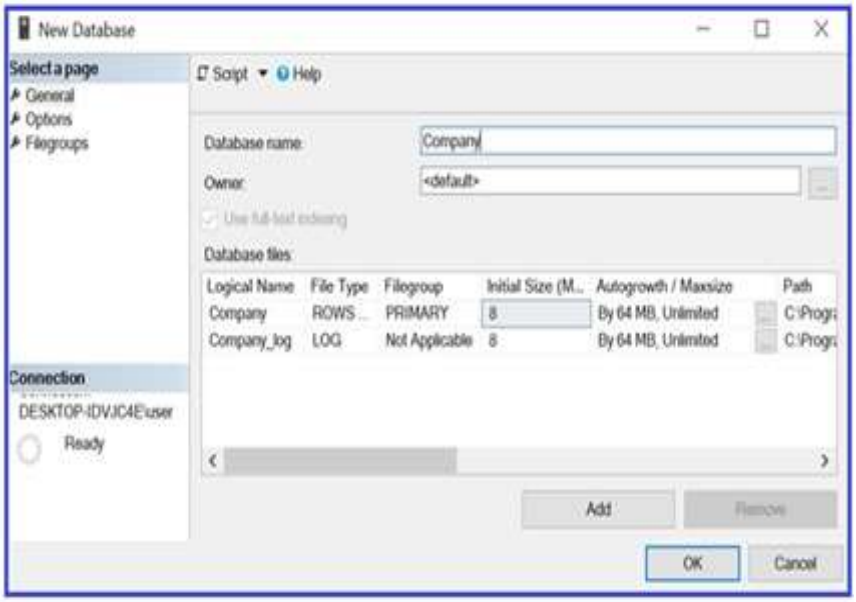
შენიშვნა: Server name - ყველას თავისი კომპიუტერის შესაბამისი ექნება.

Object Explorer-ში მარჯვენა ღილაკით ავირჩიოთ Databases -> New Database (ნახ.7.11).



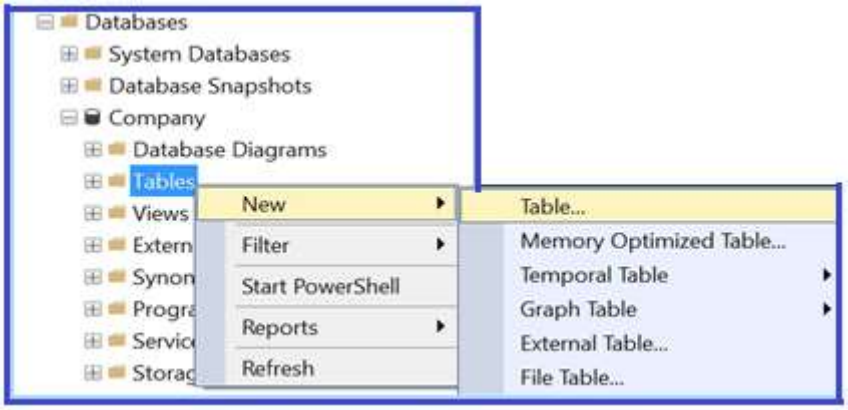
ნახ. 7.11

მონაცემთა ბაზის სახელში ჩავწეროთ Company და დავაჭიროთ ღილაკს"Ok".



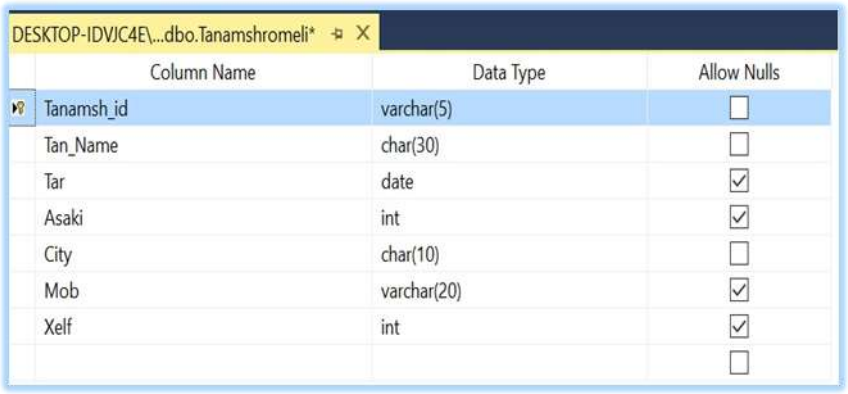
ნახ. 7.12

Company მონაცემთა ბაზაში შევქმნათ Tanamshromeli1 ცხრილი. Databases-ზე მაუსის მარჯვენა ღილაკი და ვირჩევთ: Tables -> New -> Table.



ნახ. 7.13

შევიტანოთ ცხრილის სტრუქტურის ატრიბუტები და შევინახოთ სახელით Tanamshromeli1.



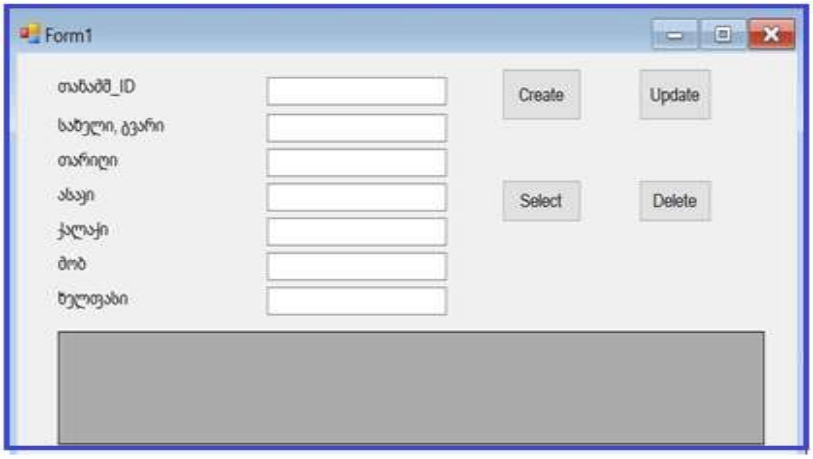
ნახ. 7.14

მარჯვენა ღილაკის საშუალებით გავხსნათ მონაცემების შეტანის ფანჯარა "Edit Top 200 Rows" და დავამატოთ რამდენიმე ჩანაწერი (7.15).

ნახ. 7.15

- *<u>ამოცანა-1:</u> შევქმნათ აპლიკაცია გამოყოფილი Dataset-ისთვის:*

გავხსნათ Visual Studio და შევქმნათ პროექტი (Windows Forms App .NET Framework). ფორმაზე განვათავსოთ ელემენტები ინსტრუმენტების პანელიდან (ნახ.7.16).



ნახ.7.16

დავამატოთ CRUD (Create, Read, Update and Delete) ფუნქციათა ღილაკების შესაბამისი კოდები.

```
// -- Form1.cs-ლისტინგი --------
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace TanamshDataSet
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         Sqlda = new SqlDataAdapter("Select * from Tanamshromeli1", ConnectionString);
         dataSet = new DataSet();
         Sqlda.Fill(dataSet, "Emp");
      }
      public string ConnectionString
      {
         get;
         set;
      } = @"Data Source=DESKTOP-IDVJC4E\MSSQLSERVER01;Initial Catalog=Company;Integrated
Security=True;";
      SqlDataAdapter Sqlda;
      DataSet dataSet;
  }
}
```

1) კოდში თავდაპირველად ვამატებთ System.Data და System.Data.SqlClient სახელსივრცეებს, რათა შევძლოთ DataSet კლასის გამოყენება.

2) ვქმნით თვისებას ConnectionString, რომელიც მოიცავს სამ მნიშვნელობას:
   - Data Source - სერვერის დასახელება;
   - Initial Catalog - მონაცემთა ბაზის დასახელება;
   - Integrated Security;

3) ვაცხადებთ SqlDataAdapter და DataSet კლასების ორ ობიექტს, Form1() {...} -ში ინიცირებით.

4) ვიყენებთ of DataSet Fill() მეთოდს Tanamshromeli1 ცხრილის მისაღებად. ჩვენ შეგვიძლია გამოვიყენოთ მარტივი კოდი of DataSet Fill(), როდესაც მხოლოდ ერთი ცხრილი (Table) გვაქვს. იმ შემთხვევაში თუ DataSet შედგება რამდენიმე ცხრილისგან, მაშინ ვსაზღვრავთ ცხრილის სახელს (მაგალითად, "Emp") ან ვიყენებთ ინდექსს თითოეული ცხრილისთვის ([0], [1], ...), რაც კარგ პრაქტიკად ითვლება.

```
//-- Create ღილაკის ლისტინგი:
```

```
private void button1_Click(object sender, EventArgs e)
    {
       DataRow dr = dataSet.Tables["Emp"].NewRow();
       dr[0] = Convert.ToInt32(textBox1.Text);
       dr[1] = textBox2.Text;
       dr[2] = textBox7.Text;
       dr[3] = Convert.ToInt32(textBox4.Text);
       dr[4] = textBox5.Text;
       dr[5] = textBox6.Text;
       dr[6] = Convert.ToInt32(textBox7.Text);

       dataSet.Tables["Emp"].Rows.Add(dr);
       MessageBox.Show("შეიქმნა ჩანაწერი..");
       Sqlda.Fill(dataSet);
    }
```

- ვქმნით DataRow კლასის "dr" ობიექტს DataSet-ის Tanamshromeli1 ცხრილში ველის დასამატებლად;
- "dr" ობიექტის შექმნის შემდეგ ვაკავშირებთ TextBox-ების მნიშვნეობებს თითოეული ველის ინდექსთან;
- Add() მეთოდით ვამატებთ "dr" ობიექტს "Emp" ცხრილს.

```
//--- Select ღილაკის ლისტინგი -------------
private void button2_Click(object sender, EventArgs e)
{
 dataGridView1.DataSource = dataSet.Tables["Emp"];
}
```

ცხრილის (Table) მონაცემების გამოსატანად DataGridView-ს ვაკავშირებთ DataSet-ის "Emp" ცხრილთან, როგორც მონაცემთა წყაროსთან.

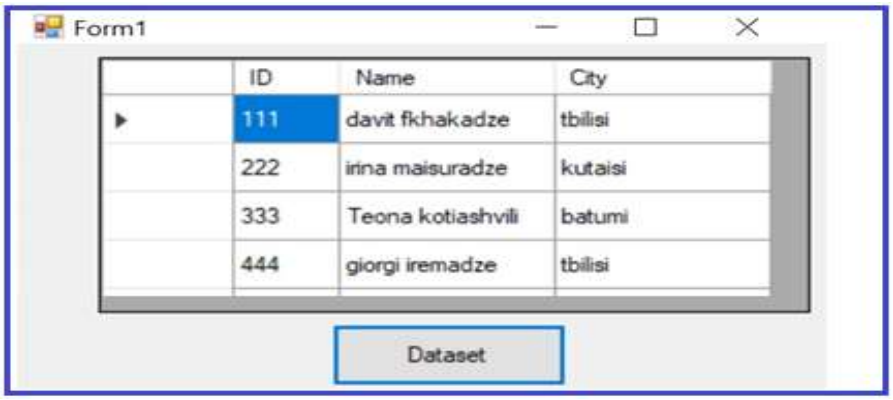
```
// ---- Update - ღილაკის ლისტინგი -------
private void button2_Click(object sender, EventArgs e)
    {
       dataGridView1.DataSource = dataSet.Tables["Emp"];
    }
    private void button3_Click(object sender, EventArgs e)
    {
      foreach (DataRow dr in dataSet.Tables["Emp"].Rows)
       { if (dr[0].ToString() == textBox1.Text)
          {
           try
            {
              dr[0] = Convert.ToInt32(textBox1.Text);
              dr[1] = textBox2.Text;
              dr[2] = textBox7.Text;
              dr[3] = Convert.ToInt16(textBox4.Text);
              dr[4] = textBox5.Text;
              dr[5] = textBox6.Text;
              dr[6] = Convert.ToInt32(textBox7.Text);
```

```
            MessageBox.Show("მონაცემები წარმატებით განახლდა ცხრილში. ბაზაში ჩაწერა ღილაკით – ბაზის განახლება..");
            }
            catch (Exception ex)
            {
               MessageBox.Show(ex.Message);
            }
            break;
         }
      }
   }
// ---- Delete - ღილაკის ლისტინგი ---------
private void button4_Click(object sender, EventArgs e)
   {
      foreach (DataRow dr in dataSet.Tables["Emp"].Rows)
      {
         if (dr[0].ToString() == textBox1.Text)
         {
            try
            {
               dataSet.Tables["Emp"].Rows.Remove(dr);
               MessageBox.Show("მონაცემი წარმატებით წაიშალა..");
            }
            catch (Exception ex)
            {
               MessageBox.Show(ex.Message);
            }
            break;

         }
      }
   }
```

- *<u>ამოცანა-2:</u> განვიხილოთ აპლიკაცია, როდესაც DataSet-ის მონაცემებს არ ვიღებთ მონაცემთა ბაზიდან და თავად ვქმნით DataSet-ის ცხრილს (Table), სვეტებს და ვავსებთ ველებს მონაცემებით.*

```
// ---- DataSet - ღილაკის ლისტინგი ----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp2
{
```

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            DataTable table1 = new DataTable();
            DataTable table2 = new DataTable();

            DataColumn dc11 = new DataColumn("ID", typeof(Int32));
            DataColumn dc12 = new DataColumn("Name", typeof(string));
            DataColumn dc13 = new DataColumn("City", typeof(string));
            table1.Columns.Add(dc11);
            table1.Columns.Add(dc12);
            table1.Columns.Add(dc13);

            table1.Rows.Add(111, "davit fkhakadze", "tbilisi");
            table1.Rows.Add(222, "irina maisuradze", "kutaisi");
            table1.Rows.Add(333, "Teona kotiashvili", "batumi");
            table1.Rows.Add(444, "giorgi iremadze", "tbilisi");

            DataSet dset = new DataSet();
            dset.Tables.Add(table1);
            dataGridView1.DataSource = dset.Tables[0];

        }
    }
}
```

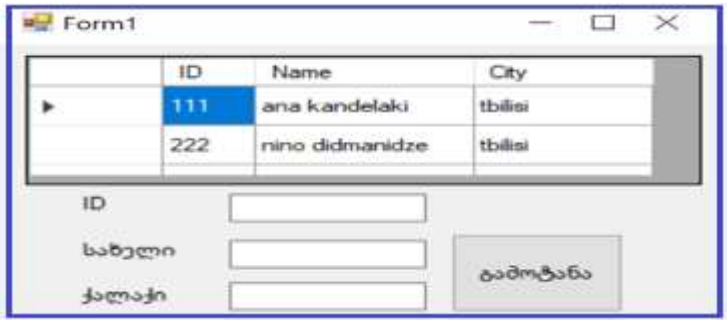


ნახ.7.17

როგორც ვიცით, DataSet ობიექტი მონაცემებს ინახავს სვეტებისა და სტრიქონების ფორმატში, ამიტომ მონაცემების გამოსატანად ვიყენეთ მათ ინდექსებს.

„გამოტანა“ ღილაკით DataSet-დან პირველი ცხრილის მონაცემები გამოვიტანოთ TextBox-ებში, ხოლო DatagridView-ში გამოჩნდეს ცხრილის ყველა ჩანაწერი ფორმის ჩატვირთვისთანავე.

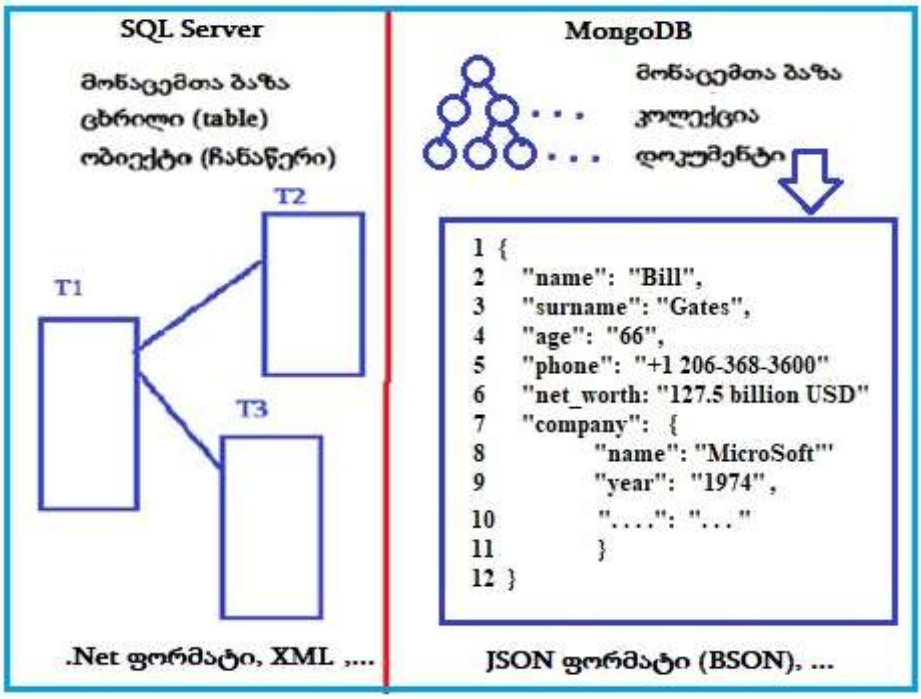
```
// ---- „გამოტანა“ ღილაკის ლისტინგი -------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;
using System.Data.SqlClient;
namespace datasetproject3
{
   public partial class Form1 : Form
   { public Form1()
     {  InitializeComponent();   }
     DataTable EMPLOYEE;
     DataSet dset;
    private void Form1_Load(object sender, EventArgs e)
     {  EMPLOYEE = new DataTable();
        DataColumn dc11 = new DataColumn("ID", typeof(Int32));
        DataColumn dc12 = new DataColumn("Name", typeof(string));
        DataColumn dc13 = new DataColumn("City", typeof(string));
      EMPLOYEE.Columns.Add(dc11);  EMPLOYEE.Columns.Add(dc12);
      EMPLOYEE.Columns.Add(dc13);  EMPLOYEE.Rows.Add(111, "ana kandelaki", "tbilisi");
        EMPLOYEE.Rows.Add(222, "nino didmanidze", "tbilisi");
        dset = new DataSet();
        dset.Tables.Add(EMPLOYEE);
        dataGridView1.DataSource = dset.Tables[0];
     }
    private void button1_Click(object sender, EventArgs e)
     {
        textBox1.Text = dset.Tables[0].Rows[0][0].ToString();
        textBox2.Text = dset.Tables[0].Rows[0][1].ToString();
        textBox7.Text = dset.Tables[0].Rows[0][2].ToString();
     }
}
```



ნახ.7.18

7.3. NoSQL მონაცემთა ბაზები: კონცეფცია და გამოყენება

დღეისათვის გამოყენებაშია მონაცემთა ბაზების სხვადასხვა ტიპები და პაკეტები, როგორიცაა კლასიკური რელაციური (SQL-ტიპის) და არარელაციური (NoSQL-ტიპის). მათ შორის არის საკმაო განსხვავება ტერმინოლოგიის და გამოყენების თვალსაზრისით [14]. ამის საილუსტრაციოდ წარმოდგენილია 7.20 ნახაზი.



ნახ. 7.20. ცხრილური და იერარქიული მოდელების განსხვავება

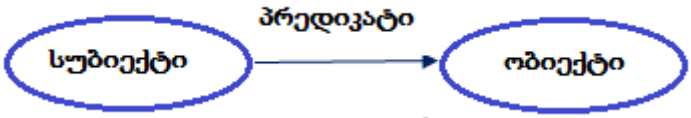
დიდი მონაცემებისა და ღრუბლოვანი ტექნოლოგიების აპლიკაციების თვალსაზრისით და განვითარების დინამიკის გათვალისწინებით უპირატესობა NoSQL-ის მხარეზეა, თუმცა რელაციური ბაზები ინარჩუნებს პრივილეგიებს კორპორაციულ სისტემებში. უმეტეს შემთხვევაში მსოფლიოს პრიორიტეტული კორპორაციები და ორგანიზაციები მონაცემთა ბაზების ორივე ტიპის აქტიურად გამოყენებას უჭერს მხარს (გადასაწყვეტი ამოცანების მიზნებიდან გამომდინარე).

NoSQL მონაცემთა ბაზის ტიპებისთვის ძირითადად დამახასიათებელი JSON ფორმატის გამოყენება, რომელიც „key : value “ (გასაღები : მნიშვნელობა) სახითაა წარმოდგენილი [14,15]. ასეთი ფორმა უზრუნველყოფს დიდი მოცულობის მონაცემთა შენახვას, რაც სხვა ტიპის მონაცემთა ბაზებში წარმოუდგენელია. წყვილი „გასაღებურ-მონაცემთა“ გამოყენების კარგი მაგალითებია, სარეკლამო აპლიკაციები, კომპიუტერული თამაშები და IoT დანართები. აღსანიშნავია ის ფაქტიც, რომ ნებისმიერი მასშტაბის მონაცემთა დამუშავებისას საკმარისია რამდნიმე მილიწამი;

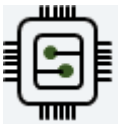
აპლიკაციის კოდში დოკუმენტი ხშირად წარმოდგენილია, როგორც ობიექტი ან დოკუმენტი JSON-ის ფორმატით, რადგან ის დეველოპერბისთვის წარმოადგენს მონაცემთა ეფექტურ ინტუიციურ მოდელს. დოკუმენტების მონაცემთა ბაზები დეველოპერებს ეხმარება მონაცემთა შენახვასა და დამუშავებაში, იმავე დოკუმენტის

მოდელების გამოყენებით, რომელსაც იყენებდენენ კოდის ჩაწერისას. დოკუმენტების ნახევრად სტრუქტურირებული, იერარქიული თვისება საშუალებას იძლევა დოკუმენტების ფორმირება მოხდეს მოთხოვნის შესაბამისად. დოკუმენტური მოდელი ეფექტურად მუშაობს კატალოგებთან, სამომხმარებლო ინტერფეისებთან და სისტემის კონტეტის მართვასთან, სადაც ყოველი დოკუმენტი არის უნიკალური და იცვლება პერიოდულად.

გრაფული მონაცემთა ბაზები ამარტივებს მონაცემთა დამუშავებას და რთული კავშირების აპლიკაციების ამუშავებას. ასეთი ბაზების საფუძველია სამეული: „სუბიექტი-პრედიკატი-ობიექტი“ (ნახ.7.21). მაგალითად, „დილიძე არის სტუდენტი“.



გრაფული ბაზებისს გამოყენების მაგალითია, სოციალური ქსელები, სარეკომენდაციო სერვისები, თაღლითური ქცევები და ცოდნის გრაფები. ისეთი გიგანტი კომპანია, როგორიცაა Amazon Neptune – უსერვერო გრაფული მონაცემთა ბაზა მაღალი მასშტაბირებით და წვდომადობით [16].



მონაცემთა ბაზა მეხსიერებაში. ხშირად სათამაშო და სარეკლამო დანართებში იყენებენ ე.წ. ლიდერბორდებს, სადაც ასახულია ჩაწერილი სესიების, რეალურ დროში ჩატარებული ანალიზის შედეგები.

7.3.1. ლაბ-N12: MongoDB Compass – ინსტალაცია და სამუშაო გარემო

MongoDB Compass-ის ჩამოწერა და ინსტალირება, შესაბამისი ინსტრუქციებით, შესაძლებელია ამ მისამართით:

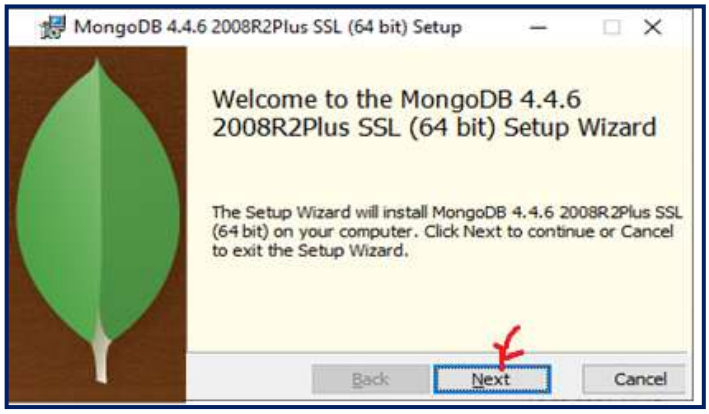
https://www.mongodb.com/docs/compass/current/install/

აქ მოკლედ გვაქვს მოცემული (გამეორების მიზნით) ინსტალაციის ბიჯები.

1) ჩამოვტვირთოთ MongoDB Community Server. ჩვენ დავაინსტალირებთ 64-ბიტიან ვერსიას Windows-ისთვის;

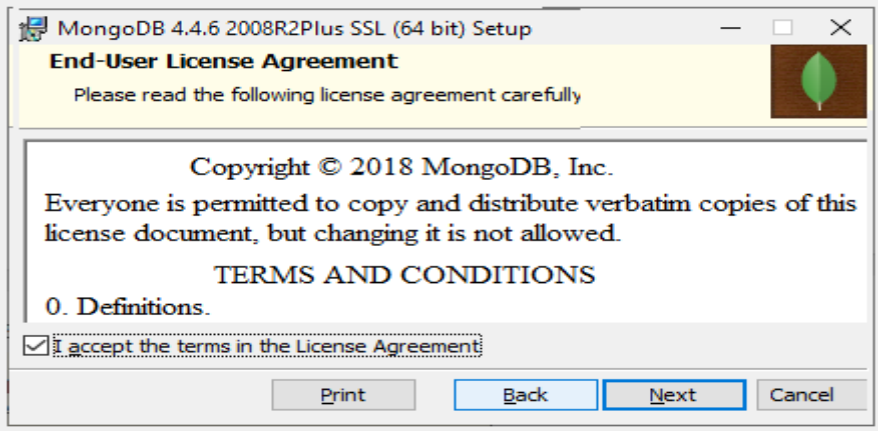
ნახ.7.21

2) ჩამოტვირთვის შემდეგ, გავხნათ msi ფაილი და next (ნახ.7.22).



ნახ. 7.22

3) მომხმარებლის ლიცენზიის სეთანხმების შემდეგ - ღილაკი Next (ნახ.7.23).

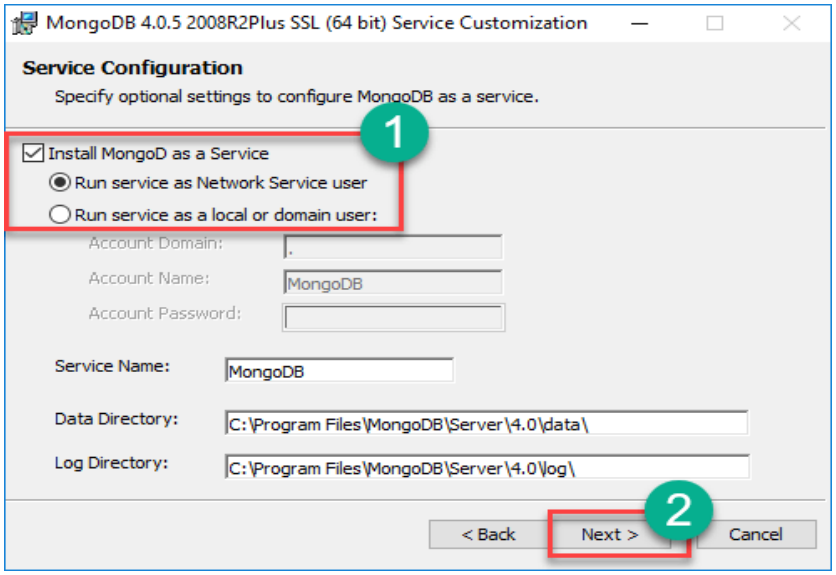


ნახ.7.23

4) სრული ვერსიის დაყენება - „complete“ ღილაკით. მომხმარებლის სურვილით არჩევა კი - „Custom“ - ით.

ნახ.7.24

5) ავირჩიოთ „სერვისის გაშვება, როგორც ქსელის სერვისის მომხმარებელი“. ჩავინიშნოთ მონაცემთა დირექტორია და Next (ნახ.7.25).

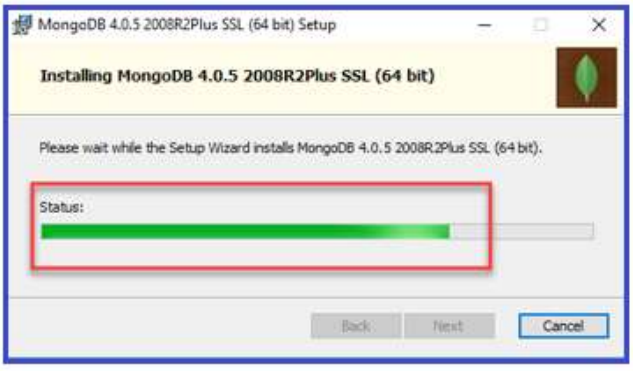


ნახ. 7.25

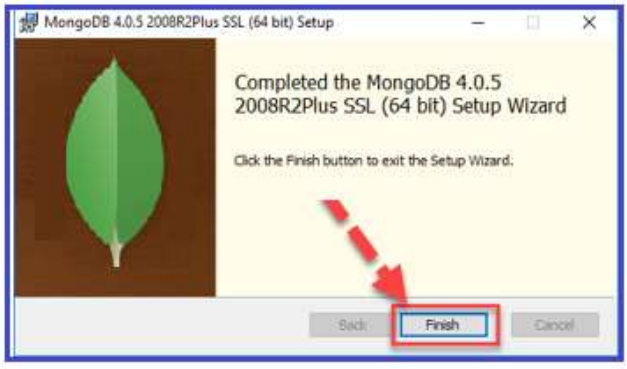
6) ვაჭერთ Install - ღილაკს (ნახ.7.26).

ნახ.7.26

7) ინსტალაციის პროცესი მიმდინაეროებს (ნახ.7.27-ა)
8) ინსტალაციის წარმატებით დასასრული - Finish ღილაკით (ნახ.7.27-ბ).



ნახ.7.27-ა



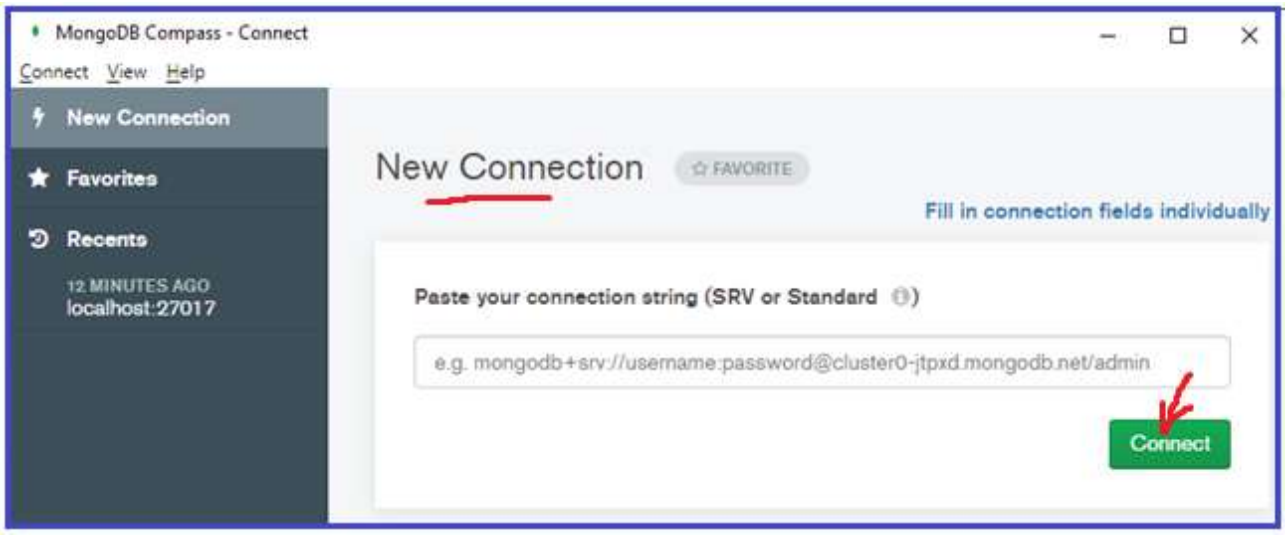
ნახ.7.27-ბ

ახლა დავაინსტალიროთ MongoDB Compass, რომელიც არის ამ ბაზის მენეჯმენტის ინსტრუმენტი, მომხმარებლის გრაფიკული ინტერფეისი. ჩამოსატვირთად გამოიყენება ლინკი:

https://www.mongodb.com/try/download/compass

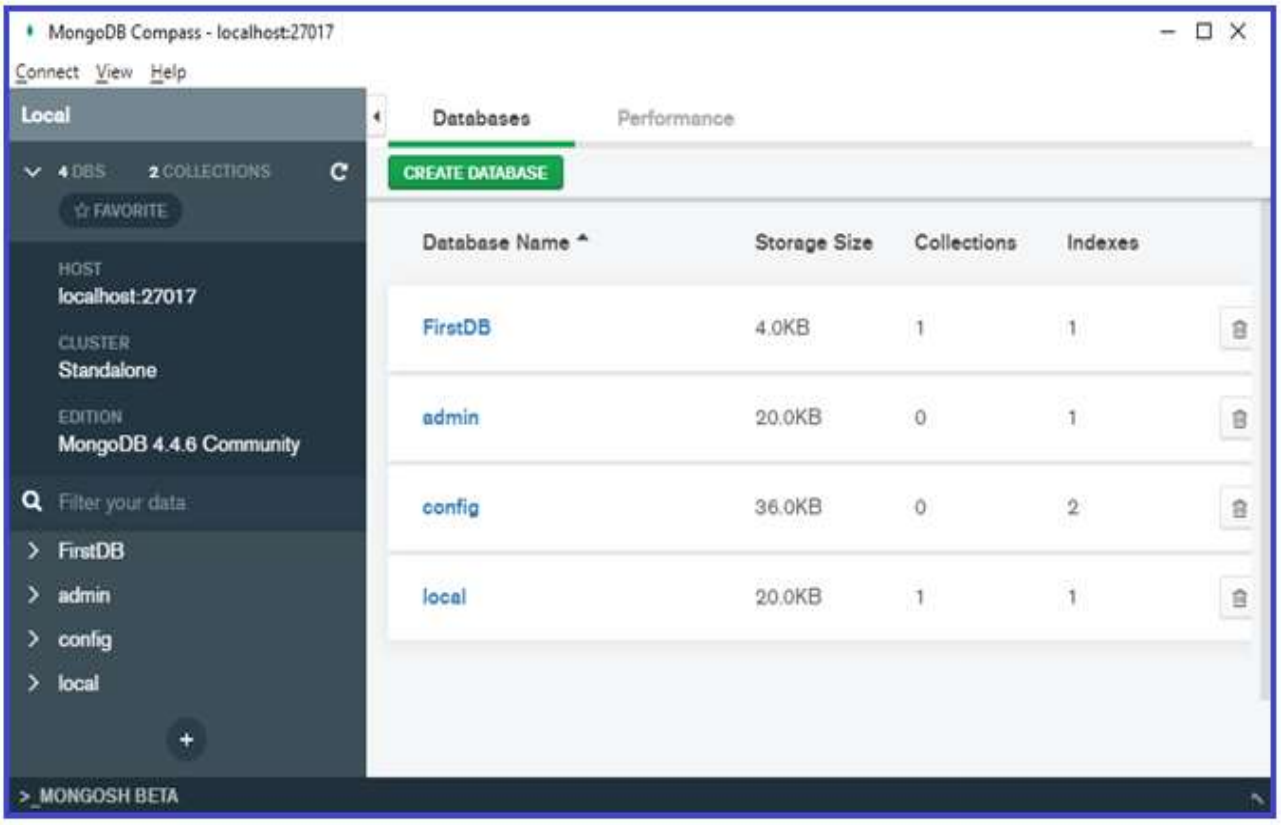
Compass- ის დაყენებით დესკტოპზე გაჩნდა ფოთლის პიქტოგრამა.

მისი ამუშავებით კომპიუტერის ეკრანზე გამოიტანება MongDB-ს საწყისი გვერდი New Connection (ბაზასთან მისაერთებლად) (ნახ.7.28).



ნახ.7.28

მიერთების შემდეგ გამოჩნდება ფანჯარა, რომელზეც ჩანს საწყისი ბაზა და რესურსი. ასევე მწვანე ღილაკი CREATE DATABASE, რომლითაც იწყება ახალი ბაზის შექმნის პროცესი (ნახ.7.29).



ნახ.7.29

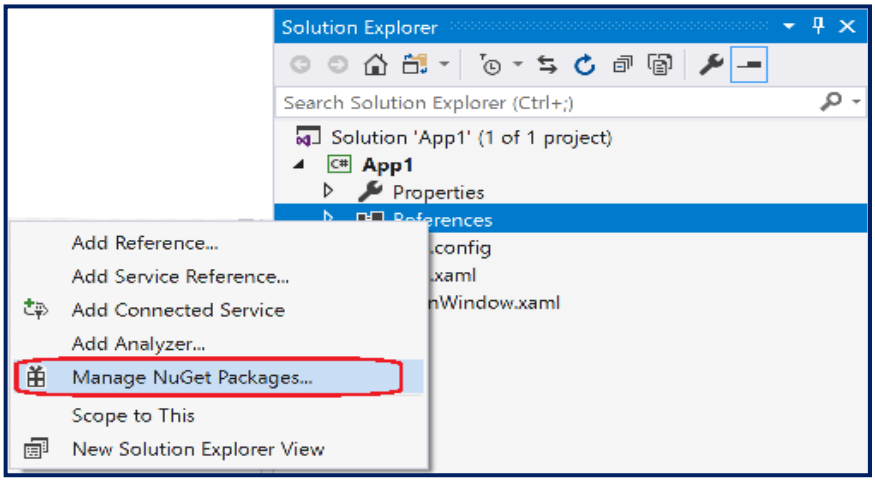
ამავე ბმულზე MongoDB Compass-ის გარდა არის MongoDB Shell. ესაა ინტერაქტიული JavaScript ინტერფეისი MongoDB-სთვის (სწრაფად ასრულებს ოპერაციებს). შესაძლებელია მისი გამოყენება მონაცემების მოთხოვნისა და განახლებისთვის, ასევე ადმინისტრაციული ოპერაციების შესასრულებლად. 7.3.3 პარაგრაფში ჩვენ განვიხილავთ მისი გამოყენების მაგალითებს.

7.3.2. ლაბ-13: NuGet პაკეტების მენეჯერი – პროექტში საჭირო დრაივერის ფაილისა და ბიბლიოთეკის ინსტალაციისთვის

NuGet არის პაკეტების მენეჯერის სამომხმარებლო ინტერფეისი, რომელიც Ms Visual Studio-ში Windows-სთვის საშუალებას იძლევა მარტივად განვახორციელოთ პროექტებსა და აპლიკაციებში დავაინსტალიროთ, წავშალოთ ან განვაახლოთ პაკეტები.

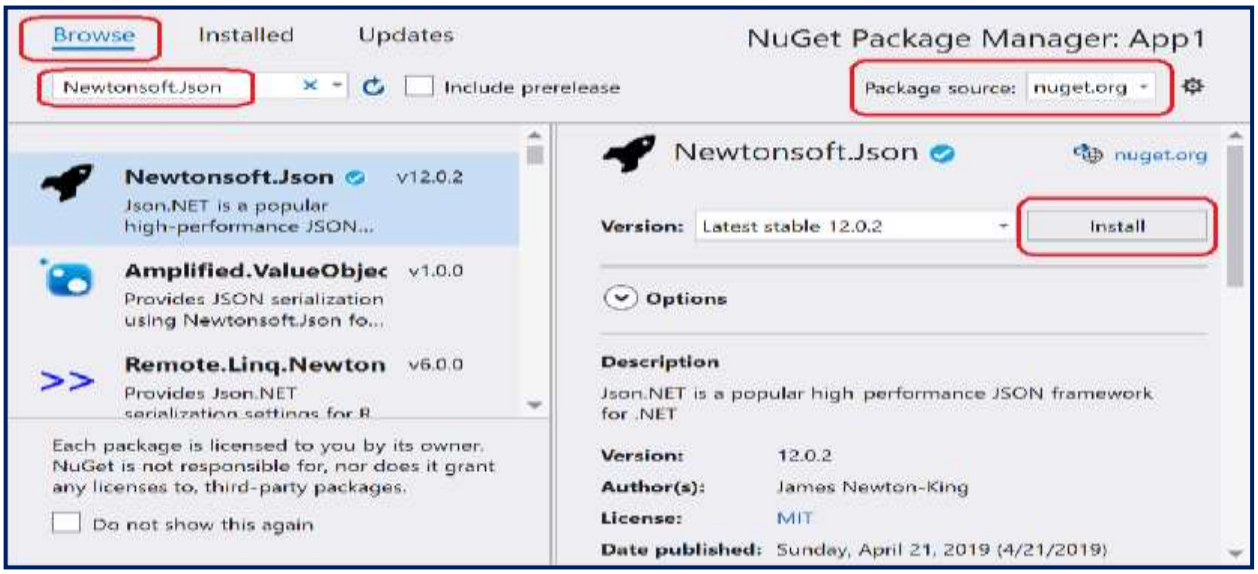
განვიხილოთ ამოცანა: როგორ გავააქტიუროთ NuGet პაკეტების მენეჯერი ჩვენი პროგრამული აპლიკაციისთვის.

- Solution Explorer-ში მაუსის მარჯვენა ღილაკით References-ზე და ავირჩიოთ Manage NuGet Packages (ნახ.7.30);



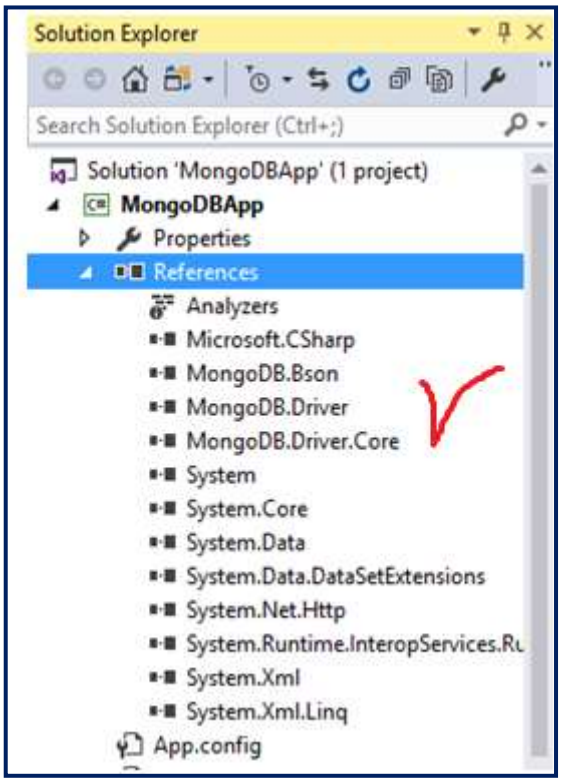
ნახ. 7.30

- პაკეტის წყაროდ ავირჩიოთ "nuget.org", ჩანართი "Browse", ვიპოვოთ Newtonsoft.Json, სიიდან ავირჩიოთ ეს პაკეტი და ბოლოს ღილაკი "Install" (ნახ.7.31).



ნახ. 7.31

- ამის შემდეგ პროექტს (Solution Explorer-ში) დაემატება ბიბლიოთეკები: MongoDB.BSON.dll, MongoDB.Driver და MongoDB.Driver.Core (ნახ.7.32).



ნახ.7.32

– MongoDB.BSON.dll ბიბლიოთეკა შეიცავს დოკუმენტის კოდს და BSON მნიშვნელობებს, ასევე კოდს BSON დოკუმენტების C # კლასის ობიექტებზე ასახვისთვის;

– MongoDB.Driver.Core.dll ბიბლიოთეკა შეიცავს სერვერთან დაკავშირების ფუნქციონირებას;

– MongoDB.Driver.dll ბიბლიოთეკა უზრუნველყოფს მსუბუქი შეფუთვას C # კოდის MongoDB სერვერთან ურთიერთქმედებისათვის.

შემდეგ, პროგრამის კოდში, ჩვენ შეგვიძლია ჩავრთოთ ყველა საჭირო სახელსივრცე:

1) using MongoDB.Bson;
2) using MongoDB.Driver;

- მონაცემთა ბაზასთან დასაკავშირებლად საჭიროა გამოვიყენოთ შემდეგი კლასები:

1) string connectionString = "mongodb://localhost:27017";
2) MongoClient client = new MongoClient(connectionString);
3) IMongoDatabase database = client.GetDatabase("test");

- კავშირის დასამყარებლად, უპირველეს ყოვლისა, უნდა გამოვიყენოთ MongoClient კლასი, რომლის კონსტრუქტორში უნდა გადაეცეს მიერთების სტრიქონი:

1) string connectionString = "mongodb://localhost:27017 "; // სერვერის მისამართი
2) MongoClient client = new MongoClient(connectionString);

- კავშირის სტრიქონი ასე გამოიყურება:

mongodb://[username:password@]hostname[:port][/[database][?options]].

თუ პორტი არ არის მითითებული, მაშინ სტანდარტულად გამოიყენება პორტი 27017.

ეს მეთოდი აბრუნებს IMongoDatabase ობიექტს:

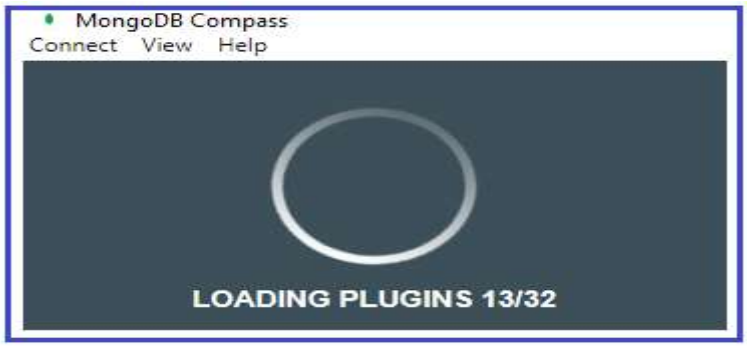
```
IMongoDatabase database = client.GetDatabase("test");
```

მონაცემთა ბაზის ობიექტის მიღების შემდეგ, ჩვენ შეგვიძლია მივიღოთ კონკრეტული კოლექციები და შევასრულოთ სხვადასხვა ოპერაციები მონაცემებზე.

7.3.3. ლაბ-N13: MongoDB Compas კოლექციების და დოკუმენტების შექმნა

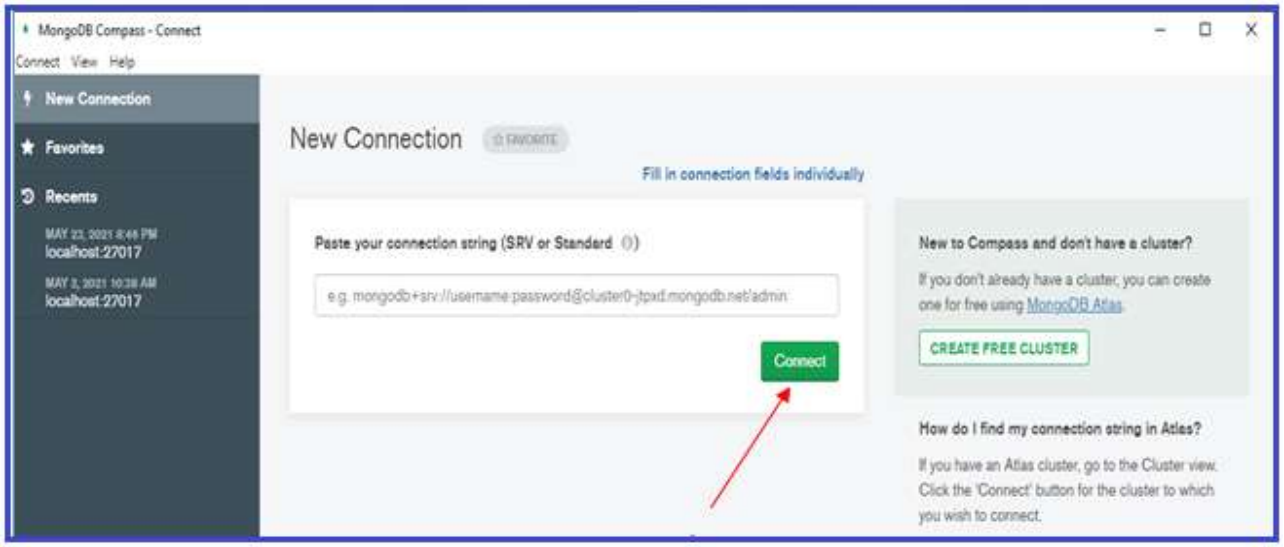
MongoDB Compass სპეციალური აპლიკაციაა (დანართია), რომელიც გვეხმარება შევქმნათ მომხმარებელზე ორიენტირებული ინტერფეისის ბაზა და მოვახდინოთ მონაცემთა ვიზუალიზაცია. შესაძლებელია ბაზაში ჩავწეროთ, განვაახლოთ ან წავშალოთ მონაცემები [16].

MongoDB პლატფორმის გამშვები ღილაკის გააქტიურების შემდეგ იხსნება ფანჯარა სადაც ხდება „პლაგინების“ ჩატვრთვა (ნახ.7.33)



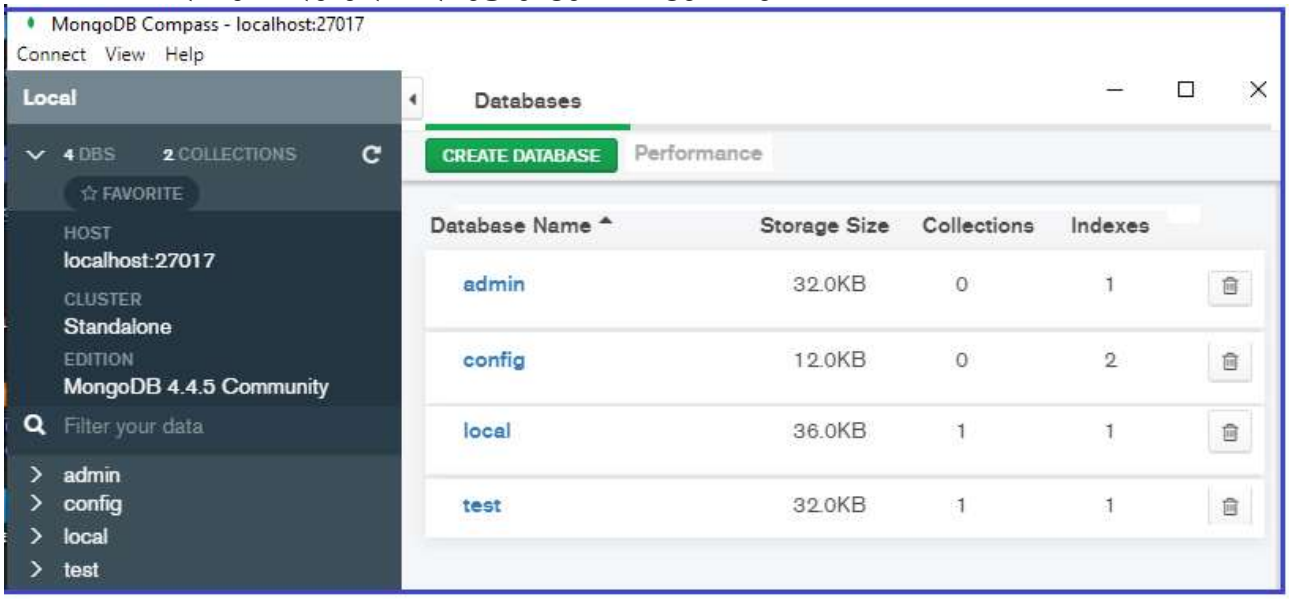
ნახ. 7.33

შემდეგ მიიღება ფანჯარა, რომლის მარცხენა მხარეს წარმოდგენილია ბოლოს შექმნილი ბაზების დასახელება, ახალი კავშირის უზრუნველყოფა და არჩეული ბაზების გამოყოფა, ხოლო შუა ნაწილში მოთავსებული Connect ღილაკის გააქტიურებით ხდება სერვერთან დაკავშირება (ნახ.7.34).



ნახ. 7.34

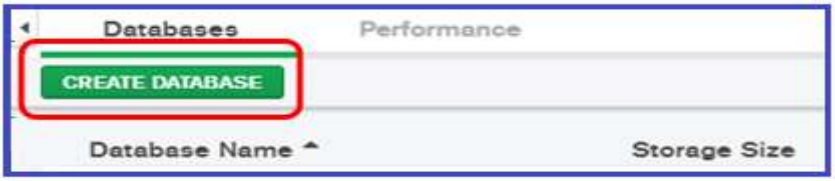
Connection-ის შედეგად მიიღება ფანჯარა საიდანაც იქმნება ახალი ბაზა (CREATE DATABASE) და შესაძლებელია დოკუმენტების ჩატვირთვა (ნახ.7.35).



ნახ. 7.35

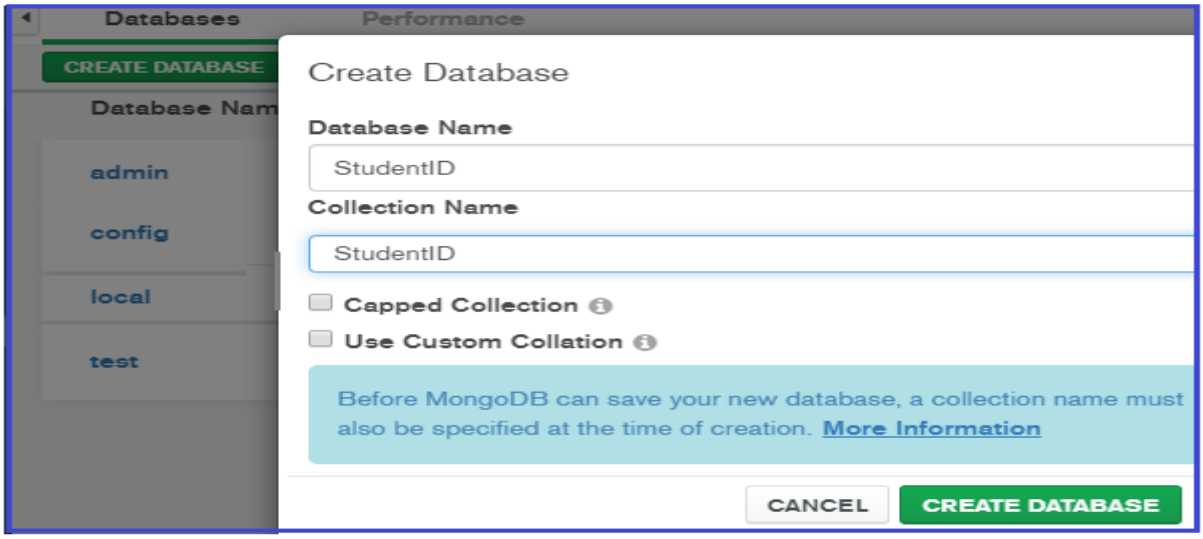
სერვერთან დაკავშირების შემდეგ, როგორც სურათიდან ჩანს ავტომატურ რეჟიმში გამოტანილია ოთხი ბაზა, რომელთა წაშლა შესაძლებელია მარჯვენა სვეტში მოცემული წაშლის ღილაკით, თუმცა მოცემულ ბაზებს აქვთ მხოლოდ დემონსტრირების დატვირთვა.

ახალი ბაზის შესაქმნელად ვააქტიურებთ ღილაკს Create Database (ნახ.7.36).



ნახ. 7.36

მიიღება დიალოგური ფანჯარა, სადაც ველში: Database Name შეგვაქვს ბაზის დასახელება, მაგალითად, StudentID და კოლექციის დასახელება, ამ შემთხვევაში იგივე სახელს, თუმცა არ არის ამის აუცილებლობა (ნახ.7.37)

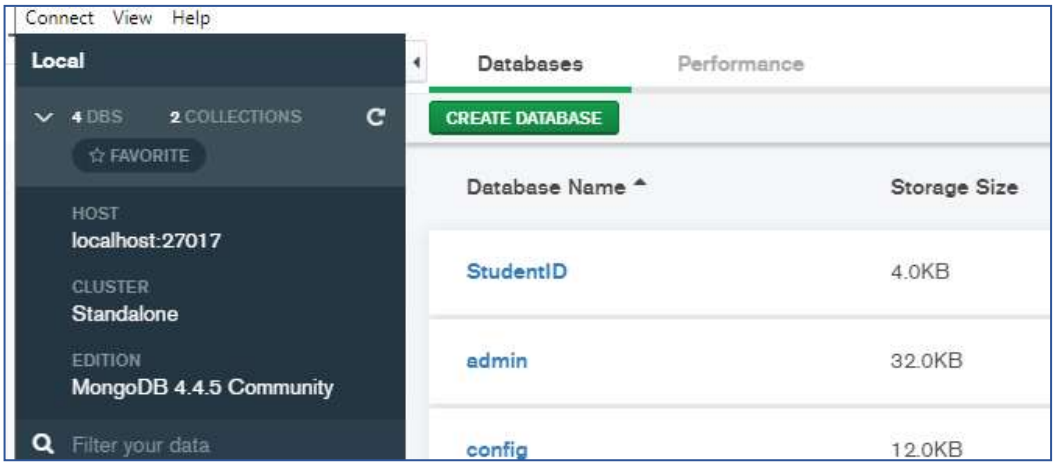


ნახ. 7.37

როგორც მოცემული ფანჯრიდან ჩანს, ბაზის შექმნისას ვუთითებთ, მონაცემთა ბაზის და კოლექციის დასახელებას. კოლექცია მსგავსია რელაციურ ბაზებში არსებული ცხრილის შინაარსობრივი არსისა, თუმცა აქ ცხრილის ნაცვლად მათ ვუწოდებთ ობიექტებს, სადაც წარმოდგენილია განსაზღვრული ინფორმაცია კონკრეტული ობიექტის შესახებ.

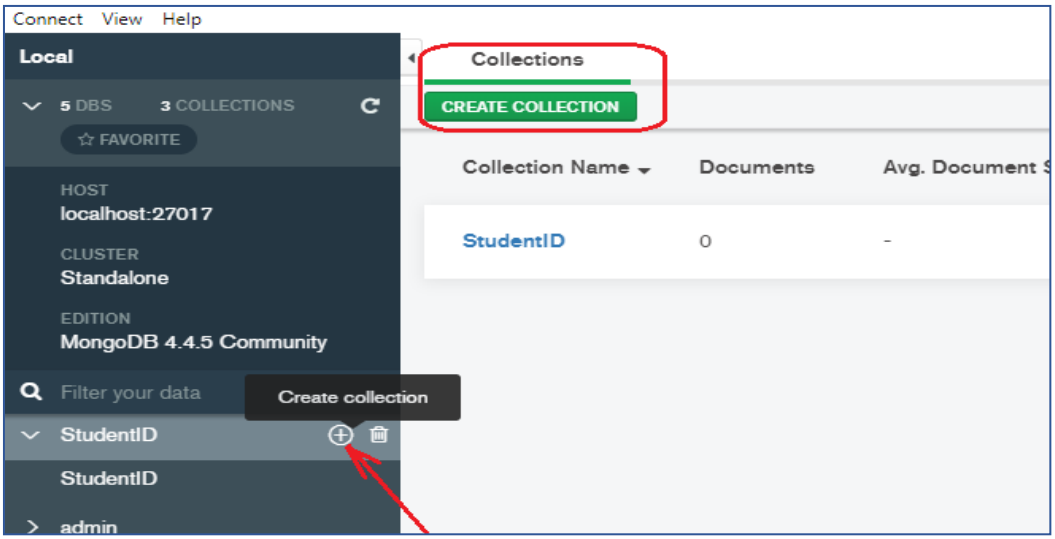
ჩვენი მაგალითის შესაბამისად, თუ ობიექტი იქნება სტუდენტი, ყველა სახის ინფორმაცია უნდა აისახოს კოლექციაში სტუდენტების შესახებ. კოლექციაში შეტანილ მონაცემებს კი უწოდებენ *დოკუმენტებს*, რომელთაც ავტომატურად ენიჭება საიდენტიფიკაციო უნიკალური კოდი. ასე, რომ კოლექციაში შესაძლებელია გვქონდეს რამდენიმე დოკუმენტი, რომელიც უშუალოდ აღწერს ობიექტის შესახებ ინფორმაციას.

ზედა მარჯვენა კუთხეში მოთავსებული Create Database ღილაკის გააქტიურების შემდეგ დავინახავთ რომ ჩვენს მიერ შექმნილი ბაზა დაემატება ჩამონათვალს (ნახ.7.38).



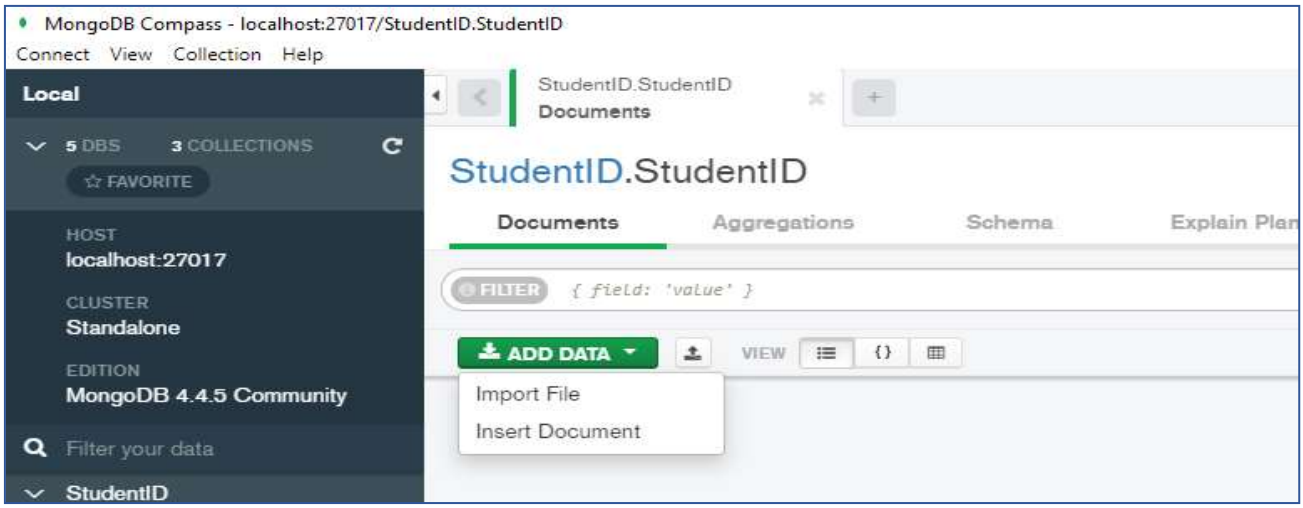
ნახ.7.38

ჩვენს მიერ შექმნილი ბაზის StudentID გააქტიურების შემდეგ მიიღება კოლექციის ფანჯარა, საიდანაც შეგვიძლია შევქმნათ ახალი კოლექცია, ღილაკის CREATE COLLETION (ან „+“ ნიშანის) გამოყენებით ან ვიწყებთ დიკუმენტების ჩატვირთვას. კოლექციაში დამატებული ნებისმიერი ობიექტი ეს არია JavaScript ობიექტი (ნახ.7.39).



ნახ.7.39

დოკუმენტების შესატანად მოვნიშნოთ კოლექციის დასახელება, შედეგად მიიღება, ფანჯარა (ნახ.7.40).



ნახ.7.40

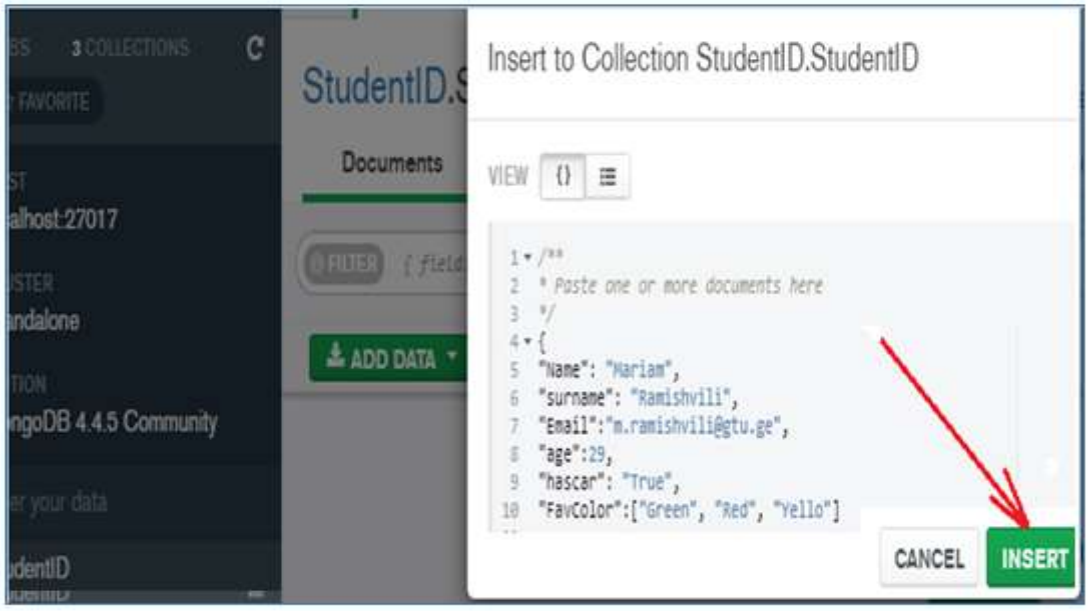
მოცემული ADD DATA ღილაკის ჩამოშლით მიღებულ, Import File ჩანართის გამოყენებით შესაძლებელია, სხვა გარემოდან მოვახდინოთ ფაილების იმპორტი, ან Insert Document-ის გამოყენებით, დავამატოთ დოკუმენტები. Insert Document ჩანართიდან გადავდივართ JSON ფორმატის გარემოში, სადაც ვწერთ ობიექტის შესახებ ინფორმაციას.

მაგალითად, სტუდენტს აქვს:

```
{
“Name”: “Mariami”,
“surname”: “Ramishvili”,
“Email”: m.ramishvili@gtu.ge,
“age”: 29 ,
“hasCar”: true,
“favColors”:[“green”, “yello”],
}
```

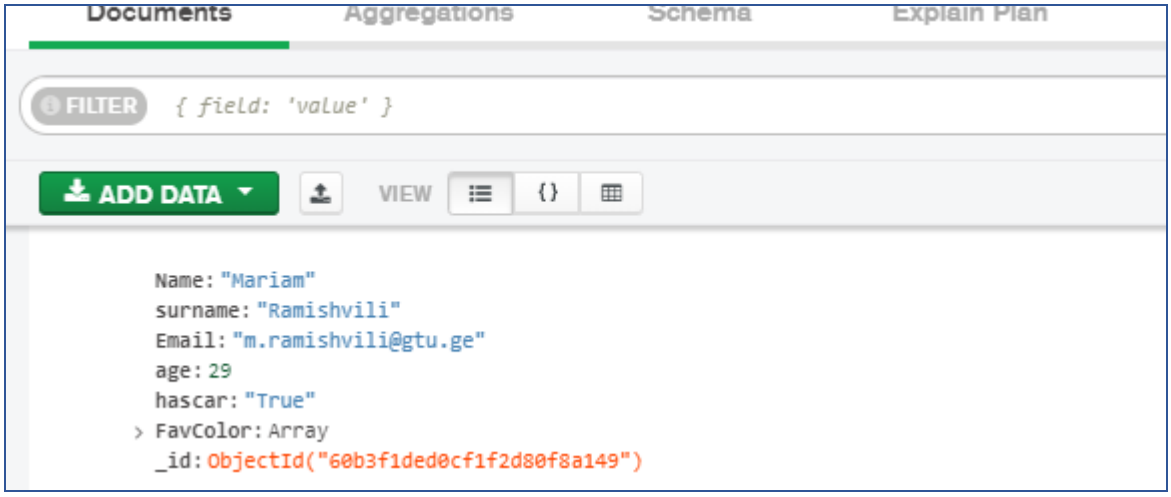
გასათვალისწინებელია შემდეგი ფაქტორები: JSON მეთოდით დოკუმენტის ჩაწერისას, აუცილებლად ტექსტური მონაცემები იწერება ორმაგ ბრჭყალებში, ჩანაწერები ერთმანეთისგან გამოყოფილია მძიმით და მთელი დოკუმენტის უნდა ჩაიწეროს ფიგურულ ფრჩხილებში.

7.41 ნახაზზე წარმოდგენილია MongoDB-ში რეალიზებული ჩვენს მიერ მოყვანილი მაგალითის ამსახველი ჩანაწერი.



ნახ.7.41

Insert ღილაკის გააქტიურებით, დოკუმენტი ემატება კოლექციას და ღებულობს შემდეგ სახეს (ნახ.7.42).



ნახ.7.42

როგორც ჩანს ბოლო სტრიქონად ავტომატურად დაემატა ჩანაწერი, რომელიც შეესაბამება საიდენტიფიკაციო კოდს, ანუ გასაღებურ ჩანაწერს. გარდა ავტომატურად დამატებისა, მომხმარებელსაც აქვს შესაძლებლობა თვითონ ცაწეროს ხელით უნიკალური კოდი.

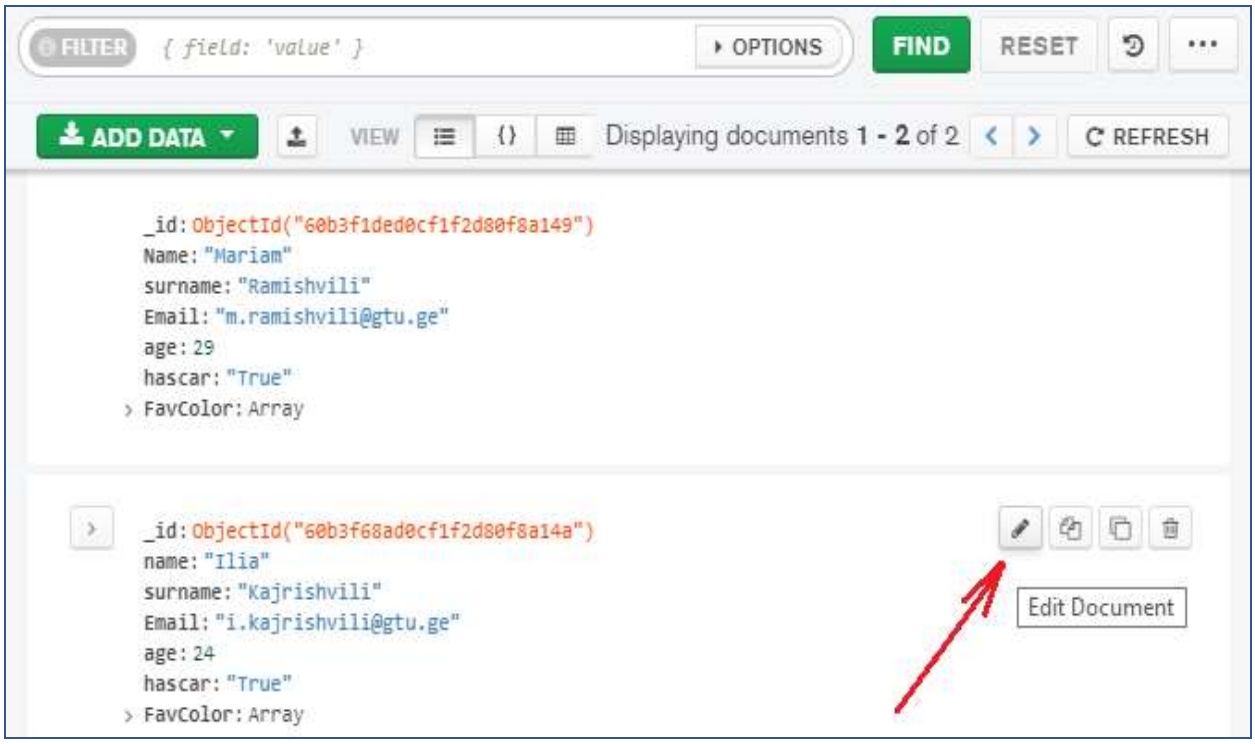
ბაზას დავამატოთ მეორე ჩანაწერი ქვემოთ მოცემული ინფორმაციის შესაბამისად:

{
“Name”: “Ilia”,
“surname”: “Kajriishvili”,
“Email”: i.kajrishvili@gtu.ge,
“age”: 24 ,
“hasCar”: true,
“favColors”:[“Red”, “Green”],
}

ცხრილურ რეჟიმში გადართვის შემდეგ ჩანაწერი ღებულობს შემდეგ სახეს (ნახ.7.43).

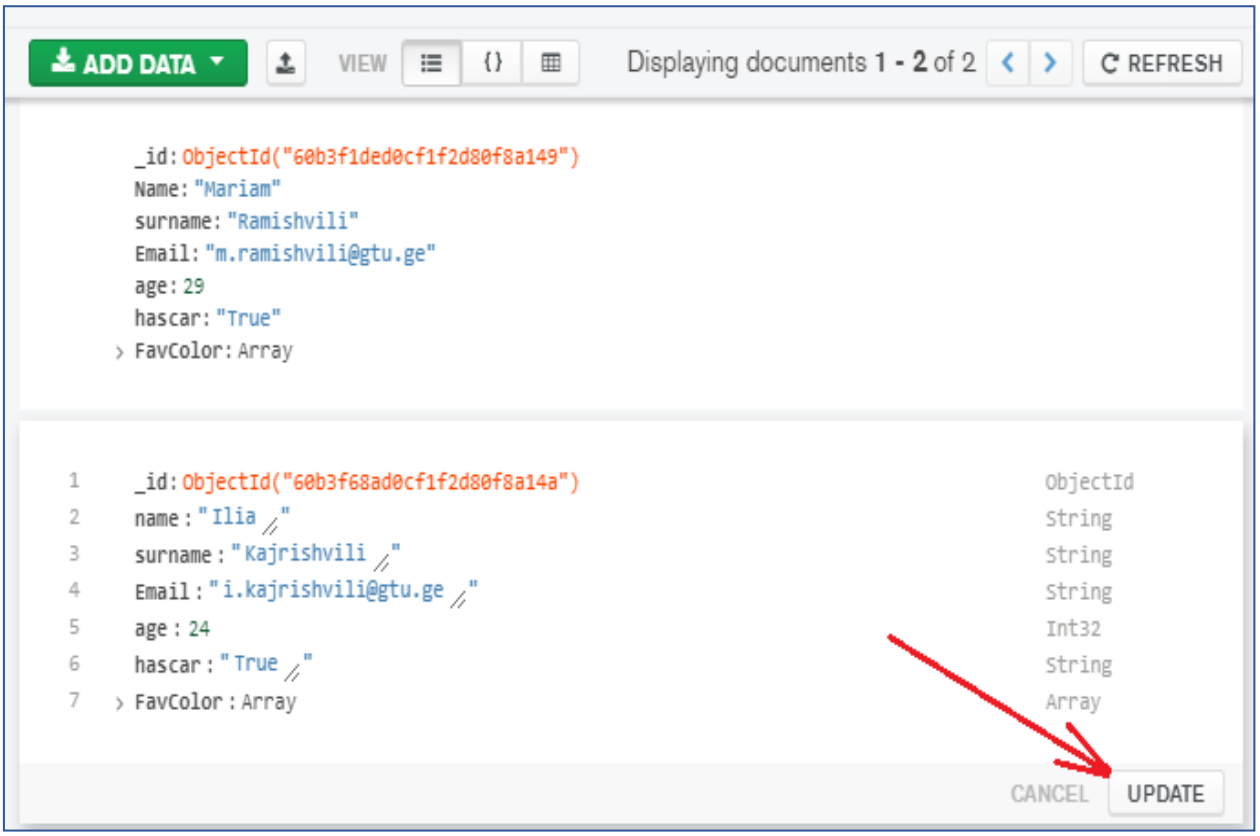
ნახ.7.43

როგორც ჩანაწერიდან ჩანს მეორე სტრიქონში მონაცემის შეტანისას გამორჩენილია სახელის მნიშვნელობა, სახელის ჩასამატებლად ან არსებულის შესაცვლელად გადავდივართ, დოკუმენტის რედაქტირების (Edit) რეჟიმში (ნახ.7.44).



ნახ.7.44

ფანჯრის (ნახ. 7.45) მარჯვენა მხარეს გამოჩნდება მონაცემთა ტიპებიც.

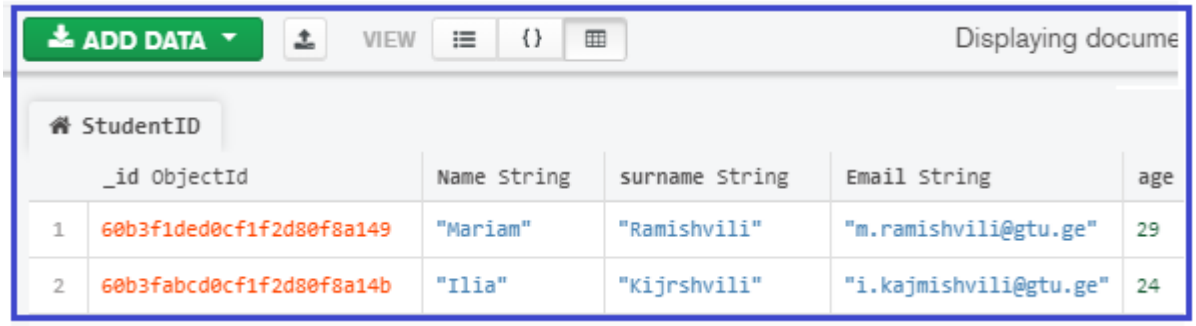


ნახ.7.44

რედაქტირებული ჩანაწერის განახლების და შენახვისთვის ვააქტიურებთ UPDATE ღილაკს.

➢ **დამატებითი ინფორმაცია MongoDB Compass -ის შესახებ:**

ქვემოთ მოცემული ფანჯარაში ობიექტები წარმოდგენილია ცხრილური სტრუქტურით, აქ შესაძლებელია, როგორც მნიშვნელობების ასევე მონაცემთა ტიპების რედაქტირებაც, შესრულებული რედაქტირების შემდეგ, უნდა გააქტიურდეს მონაცემთა განახლების UPDATE ღილაკი, შედეგად კი მივიღებთ განახლებულ ჩანაწერს (ნახ.7.45).



ნახ.7.45

როდესაც ხდება ერთი და იგივე ობიექტის შესახებ სხვადასხვა მნიშვნელობების შეტანა, მაგალითად, მონაცემები შეგვაქვს 25 სტუდენტის შესახებ, პროგრამა გვათავაზობს მონაცემთა შეტანის გამარტივებულ ვარიანტს. შესაძლებელია დავაკოპიროთ ერთხელ შეტანილი მონაცემები, ან მოვახდინოთ კლონირება, რომელთაც საერთო ველები აქვს და შევცვალოთ მნიშვნელობები (ნახ.7.46).

ნახ.7.46

რედაქტირების ფანჯარაში ვახდენთ მონაცემთა ცვლილებას და Insert ღილაკით კოლექციაში დამატებას (ნახ.7.47).

ნახ.7.47

აღსანიშნავია ის ფაქტორი, რომ MongoDB - ში ბაზების ფორმირებისას არა არის აუცილებელი ერთ ობიექტს ქონდეს მონაცემთა შეტანის ერთი და იმავე რაოდენობის ველის მნიშვნელობა, შეიძლება ერთ შემთხვევაში გვქონდეს ინფორმაცია მაგ. არჩეული ფერის (FavColor) შესახებ, ხოლო სხვა სტუდენტის შემთხვევაში არ იყოს ცნობილი ინფორმაცია, რასაც ჩვეულებრივად გამოვტოვებთ, შედეგად ჩანაწერი მიიღებს სახეს (ნახ.7.48).

StudentID

	_id ObjectId	Name String	surname String	Email String	age Int32	hascar String	FavColor Array
1	60b3f1ded0cf1f2d80f8a149	"Mariam"	"Ramishvili"	"m.ramishvili@gtu.ge"	29	"True"	[] 3 elements
2	60b3fabcd0cf1f2d80f8a14b	"Ilia"	"Kijrshvili"	"i.kajmishvili@gtu.ge"	24	"True"	[] 3 elements
3	60b40148d0cf1f2d80f8a14c	"Lily"	"Dolidze"	"l.dolidze@gtu.ge"	52	"False"	No field

ნახ.7.48

ასევე შესაძლებელია კონკრეტული ობიექტისთვის დავამატოთ ისეთი არტიბუტი, რაც მანამდე არ გვქონდა გამოყენებული, მაგალითად, შემოვიტანოთ სტუდენტის შესახებ ინფორმაცია რომელსაც აქვს სერტიფიკატი, შედეგად მიიღება (ნახ.7.49).

StudentID

	_id ObjectId	Name String	surname String	Email String	age Int32	hascar String	FavColor Array	hasCertifikat String
1	60b3f1ded0cf1f2d80f8a149	"Mariam"	"Ramishvili"	"m.ramishvili@gtu.ge	29	"True"	[] 3 elements	No field
2	60b3fabcd0cf1f2d80f8a14b	"Ilia"	"Kijrshvili"	"i.kajmishvili@gtu.g	24	"True"	[] 3 elements	No field
3	60b40148d0cf1f2d80f8a14c	"Lily"	"Dolidze"	"l.dolidze@gtu.ge"	52	"False"	No field	No field
4	60b5ab8d8acb0a30a488593e	"Giorgi"	"Jgenti"	"g.jgenti@gtu.ge"	No field	No field	[] 1 elements	"True"

ნახ.7.49

დოკუმენტში ერთი რომელიმე ობიექტის მნიშვნელობის განსაზღვრისას შესაძლებელია შემოვიტანოთ სხვადასხვა ტიპის მონაცემები, მაგალითად, სქტრიქონი, მასივი, რიცხვი, თარიღი, ობიექტი და ა.შ. (ნახ.7.50).

```
1  _id: ObjectId("60b40148d0cf1f2d80f8a14c")                ObjectId
2  Name : "Lily //"                          სტრიქონი       String
3  surname : "Dolidze //"                    სტრიქონი       String
4  Email : "l.dolidze@gtu.ge //"                            String
5  age : 52                                  რიცხვი         Int32
6  hascar : "False //"                                      String
7  "Birth Date" : 1969-02                    თარიღი         Date
8  "Child" :  {"name":"ana", "age":12} //"   ობიექტი       String
9  "FavColpr" :  ["red", "Green", "Yello"] //" მასივი       String
```

ნახ.7.50

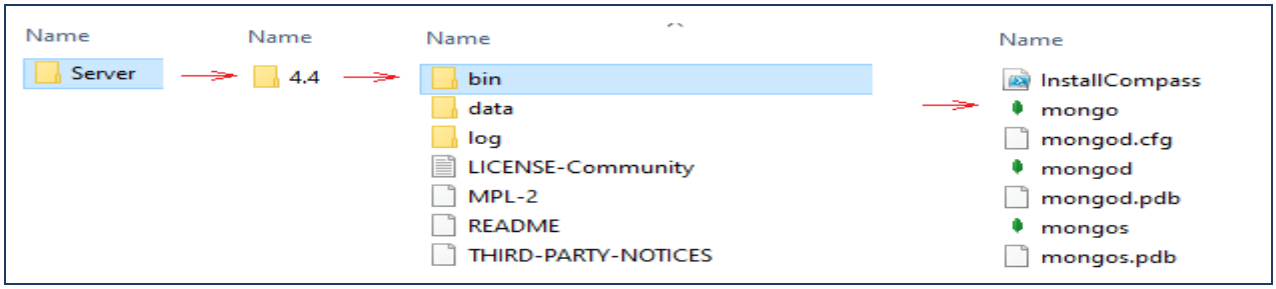
7.3.4 ლაბ-14: Mongo Shell ინტერფეისი და მისი ფუნქციები

MongoDB – ში სამუშაოდ გამოიყენება, Mongo_Compass და Mongo_Shell ინტერფეისები. პირველი ჩვენ უკვე განვიხილეთ. ახლა Mongo_Shell -ს გავეცნოთ. ჩამოტვირთვის მისამართია:

https://www.mongodb.com/try/download/shell

Mongo_shell არის JavaScript -ზე ორიენტირებული ინტერაქტიული ინტერფეისი, სადაც კოდის ჩაწერის შედეგად შესაძლებელია მონაცემთა მოთხოვნის და განახლების ოპერაციების (სწრაფად !) შესრულება, რაც MongoDB Compass -ზე ასევე აისახება ვიზუალურად.

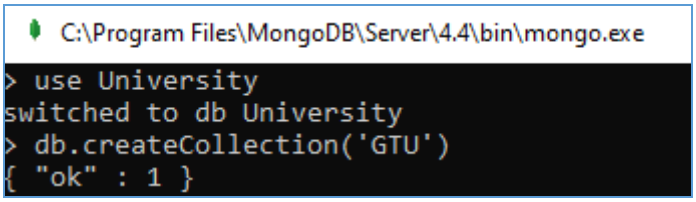
Mongo shell გარემოს გასააქტიურებლად, განვსაზღვრავთ MongoDb -ს ლოკაციის ადგილს, თუ რომელ დისკზე გვაქვს ჩვენს ლოკალურ კომპიუტერში, მოვძებნით ფოლდერს „Server” და ქვემოთ მოცემული ნიმუშის შესაბამისად გავააქტიურებთ shell ინტერფეისს (ნახ.7.51).



ნახ.7.51

Mongo shell გარემოში კოლექციის შექმნა შესაძლებელია შემდეგი კოდის ჩაწერის შედეგად: **db.createCollection(‘კოლექციის დასახელება’)**

მაგალითად, Mongo shell -ში ჩავწეროთ კოდი, რომლითაც ვქმნით ბაზას „University” და კოლექციას GTU, კოდს ექნება სახე (ნახ.7.52).



ნახ.7.52. ok:1 ნიშნავს, რომ დაემატა ერთი კოლექცია

MongoDB Compass-ზე განახლების ღილაკის გააქტიურებით დავინახავთ, რომ შეიქმნება ახალი ბაზა კოლექციით GTU (ნახ.7.53).

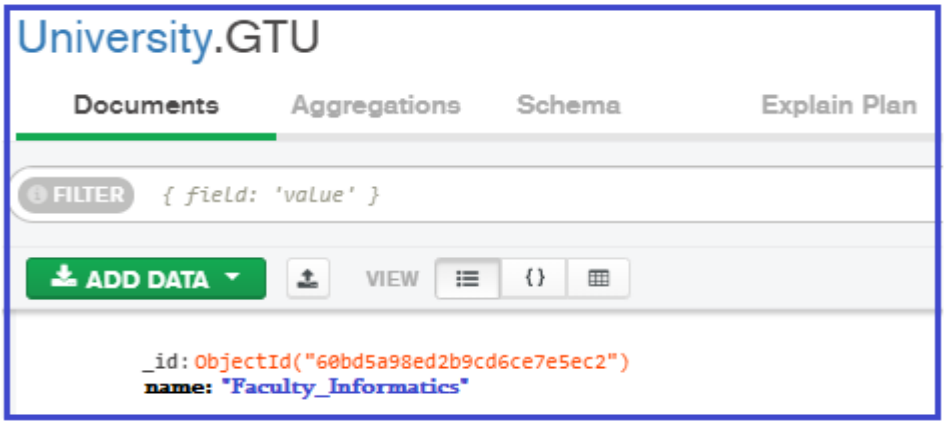
ნახ.7.53

- **insert( ), insertOne( ), insertMany( ) - ფუნქციები**

დოკუმენტების დამატებისთვის გამოიყენება, **Insert() ფუნქცია.** იმისათვის, რომ insert ფუნქციის გამოყენებით, GTU-კოლექციაში დავამატოთ დოკუმენტი, Faculty_Informatics სახელით, Shell გარემოში უნდა ჩავწეროთ კოდი შემდეგი სახით:

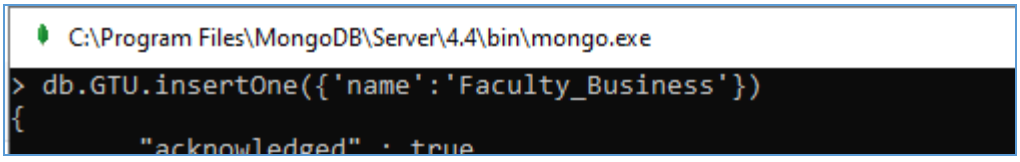
```
> db.GTU.insert({'name':'Faculty_Informatik'})
WriteResult({ "nInserted" : 1 })
>
```

Compass გარემოს განახლების შემდეგ ჩანაწერი აისახება დამატებულ დოკუმენტებში (ნახ.7.54).



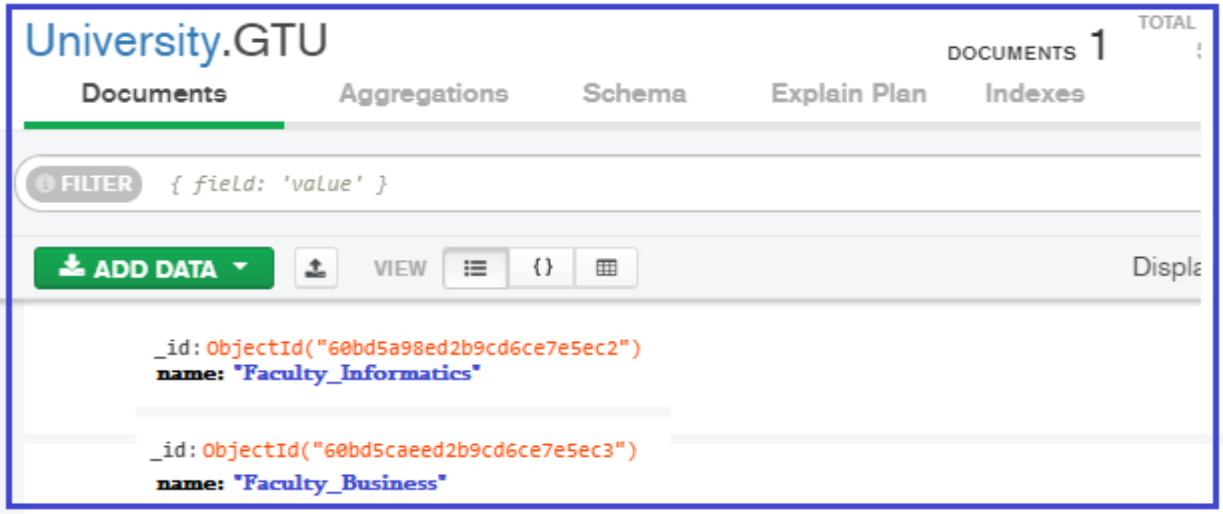
ნახ.7.54

შინაარსობრივად InsertOne ფუნქცია მსგავსია Insert ფუნქციისა, სინტაქსს აქვს ანალოგიური სახე (ნახ.7.55).



ნახ.7.55

შედეგად დაემატება კიდევ ერთი დოკუმენტი (ნახ.7.56).



ნახ.7.56

InsertMany -ფუნქციის გამოყენებით შესაძლებელია ერთდროულიად რამდენიმე ჩანაწერის დამატება. განსხვავებით წინა მაგალითებისა. InsertMany - ში გამოიყენება მასივის წარმოდგენის ფიგურული ფრჩხილები („{ }“).

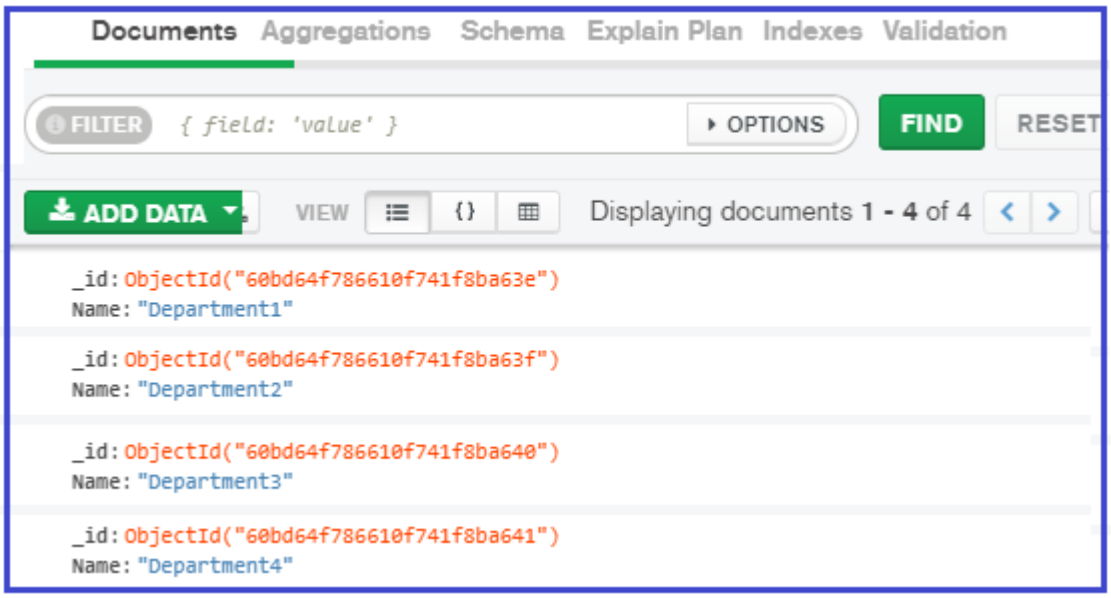
მაგალითად, ბაზას დავამატაოთ ახალი კოლექცია „Department” და ინფორმაცია ოთხი დეპარტამენტის შესახებ, სინტაქსს ექნება შემდეგი სახე:

```
> db.Department.insertMany([{'Name':'Department1'},{'Name':'Department2'}
,{'Name':'Department3'},{'Name':'Department4'}])
```

კოდის გაშვების შემდეგ ჩანაწერი მიიღებს სახეს:

```
> db.Department.insertMany([{'Name':'Department1'},{'Name':'Department2'}
,{'Name':'Department3'},{'Name':'Department4'}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("60bd64f786610f741f8ba63e"),
                ObjectId("60bd64f786610f741f8ba63f"),
                ObjectId("60bd64f786610f741f8ba640"),
                ObjectId("60bd64f786610f741f8ba641")
```

შედეგად ერთდროულად მოხდება ოთხი ჩანაწერის დამატება (ნახ.7.67).



ნახ.7.57

- **Update(..), UpdateOne(..), UpdateMany(…) - ფუნქციები**

MongoShell გარემოში მონაცემთა განახლებისთვის ვირჩევთ კოლექციას რომლის დოკუმენტის განახლებაც გვინდა და ფუნქციების **UpdateOne(..), UpdateMany(…),** გამოყენებით ვახდენთ განახლებას.

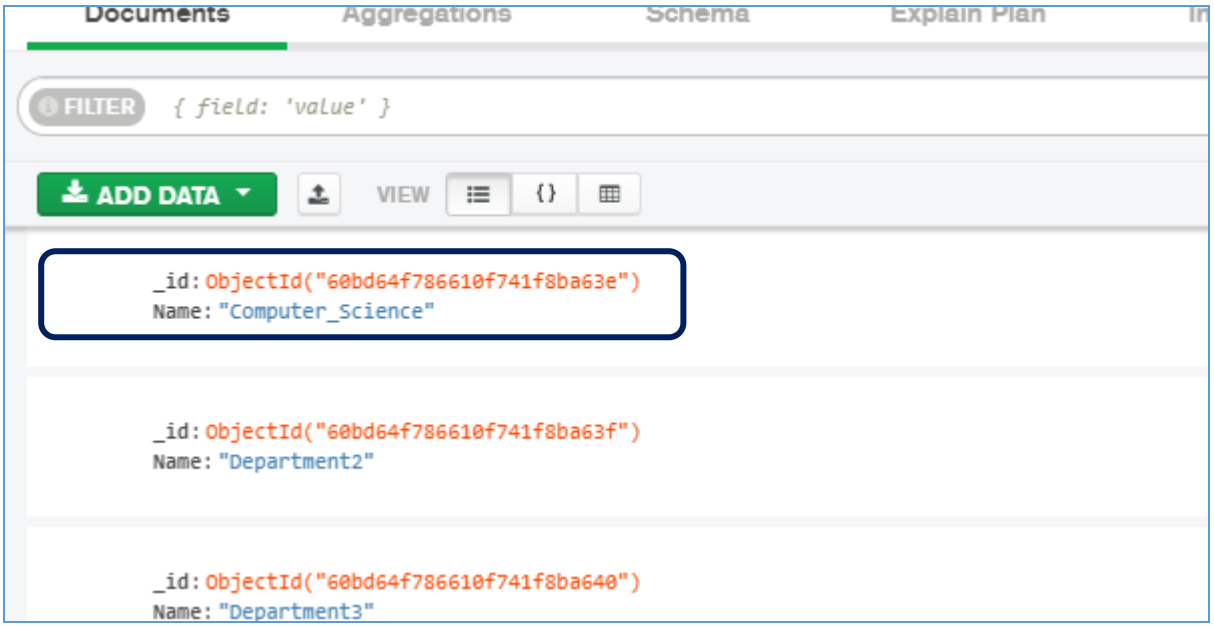
როდესაც გვინდა ერთი ჩანაწერის განახლება, ვიყენებთ **Update(..)** ან **UpdateOne(..)** ფუნქციას, რომლის შესაბამის კოდს აქვს შემდეგი სახე:

```
> db.Department.update({'Name':'Department1'},{$set:{'Name':'Computer_Science'}})
```

კოდის გაშვების შემდეგ მიიღება განახლების დამადასტურებელი შეტყობინება:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> _
```

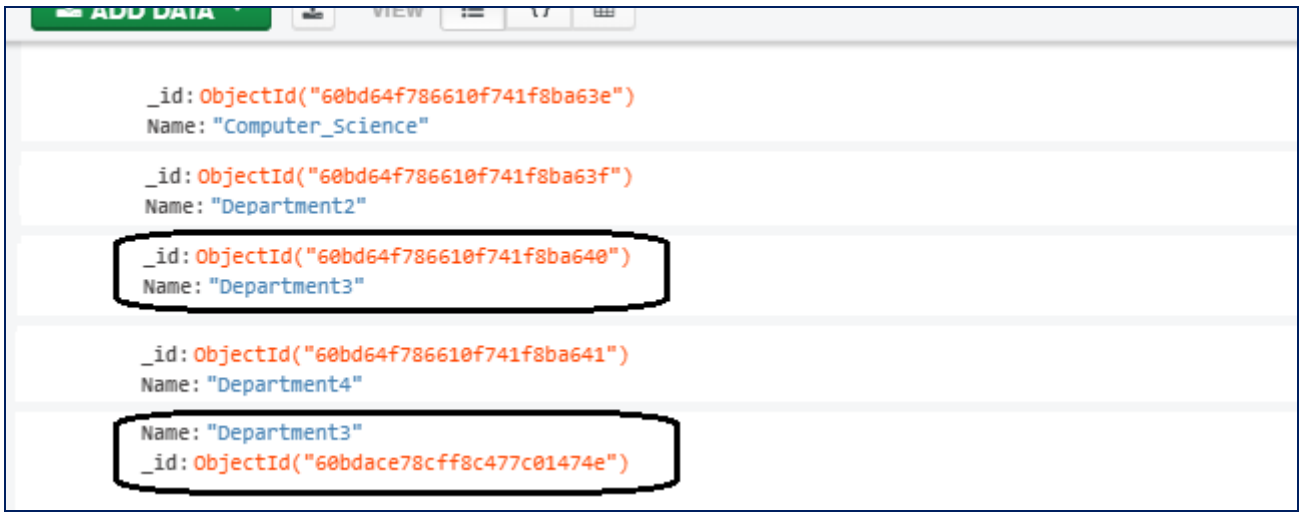
განახლდება ჩანაწერი MongoCompass გარემოშიც და ჩვენს მიერ შეტანილი დეპარტამენტის დასახელება - Department1, შეიცვლება სახელით, Computer_Science (ნახ.7.58).



ნახ.7.58

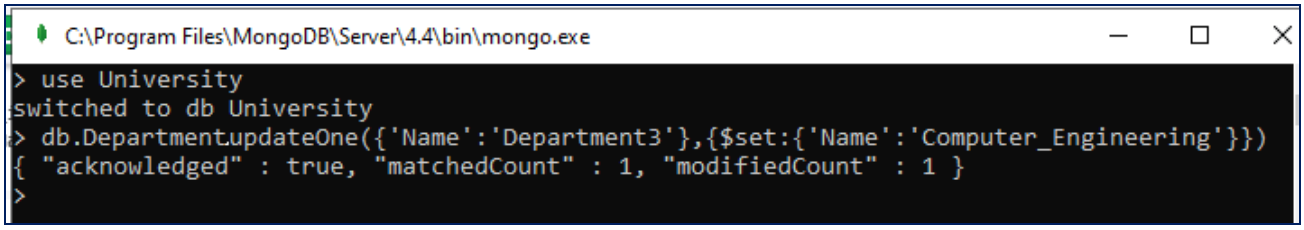
აღსანიშნავია ის ფაქტი, რომ როდესაც კოლეციაში არის ერთი და იგივე დასახელების ერთზე მეტი დოკუმენტი და გვინდა მთელ კოლექციაში განახლდეს მოცემული მნიშვნელობის ყველა ჩანაწერი, გამოიყენება ფუნქცია UpdateOne.

ნიმუშზე მოცემულ კოლექციაში მეორდება დოკუმენტი, Department3 (ნახ.7.59).



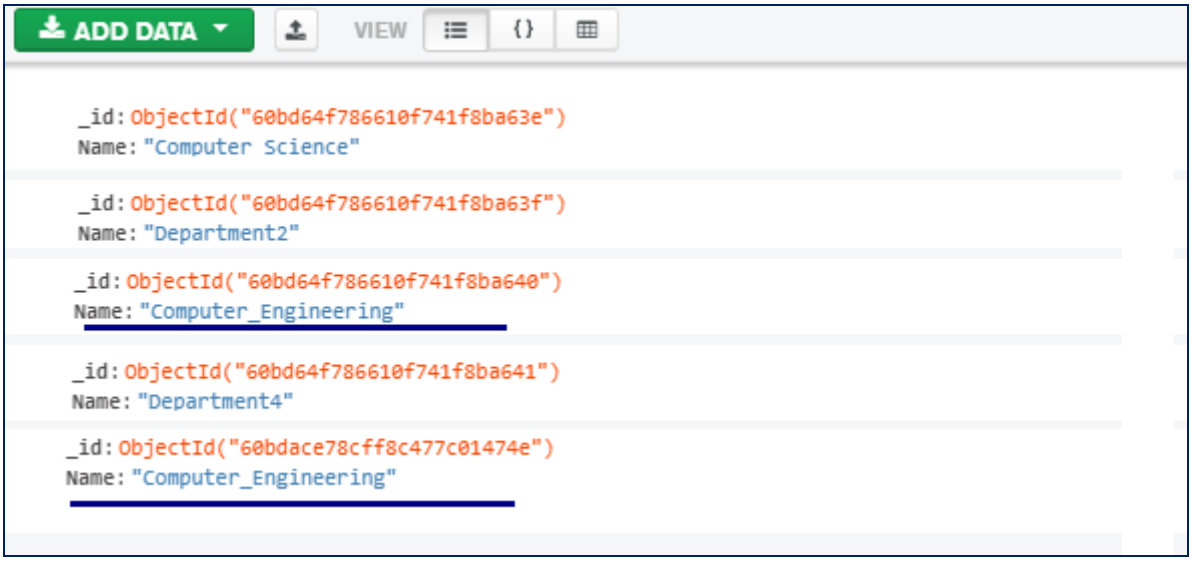
ნახ.7.59

UpdateOne ფუნქციის კოდი ჩაიწერება შემდეგი სახით (ნახ.7.60).



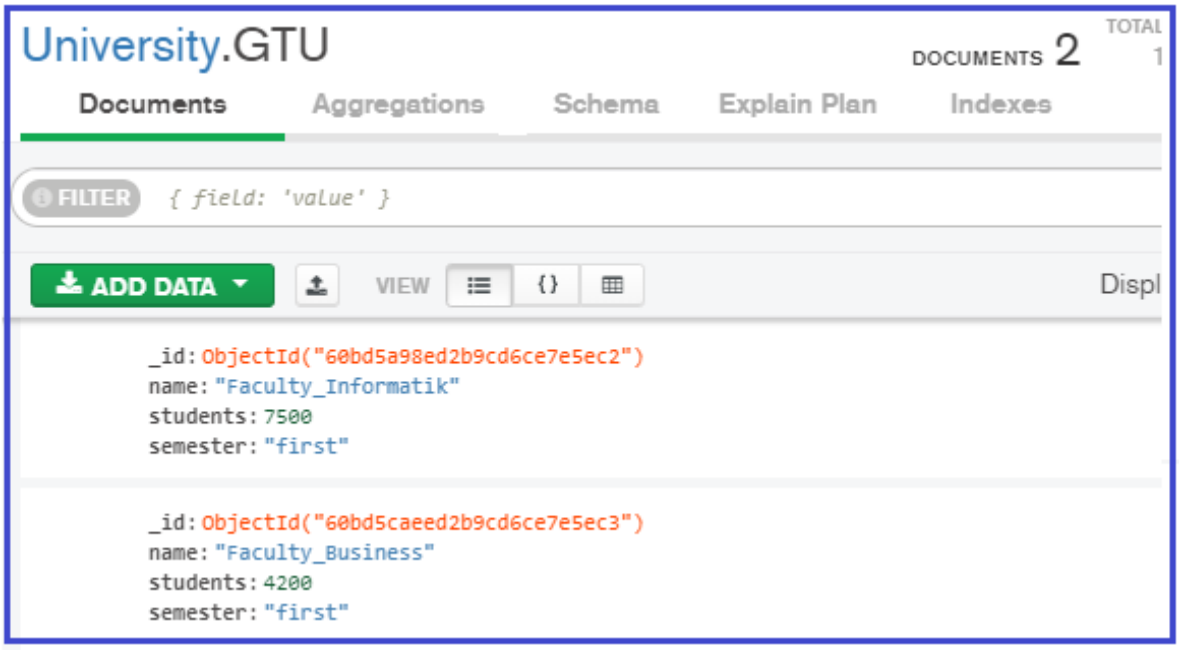
ნახ. 7.60

შედეგად განახლდება ერთიდაიგივე დასახელების ორივე ჩანაწერი (ნახ.7.61).



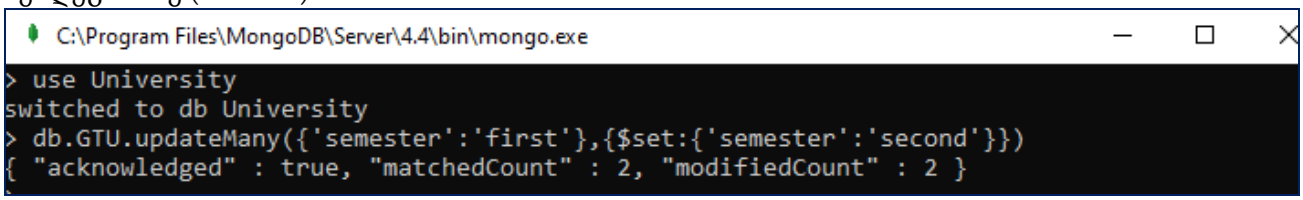
ნახ.7.61

ერთდროულად რამდენიმე ჩანაწერის განახლების შემთხვევაში, გამოიყენება ფუნქცია updateMany, ჩვენს მიერ შექმნილ კოლექციას GTU, დოკუმენტში, სადაც ფაკულტეტების შესახებ გვაქვს ინფორმაცია დავამატოთ ახალი ველი, სტუდენტები (students) და სემესტრი (semester) ნიმუშის შესაბამისად ჩავწეროთ შესაბამისი მნიშვნელობები (ნახ.7.62).



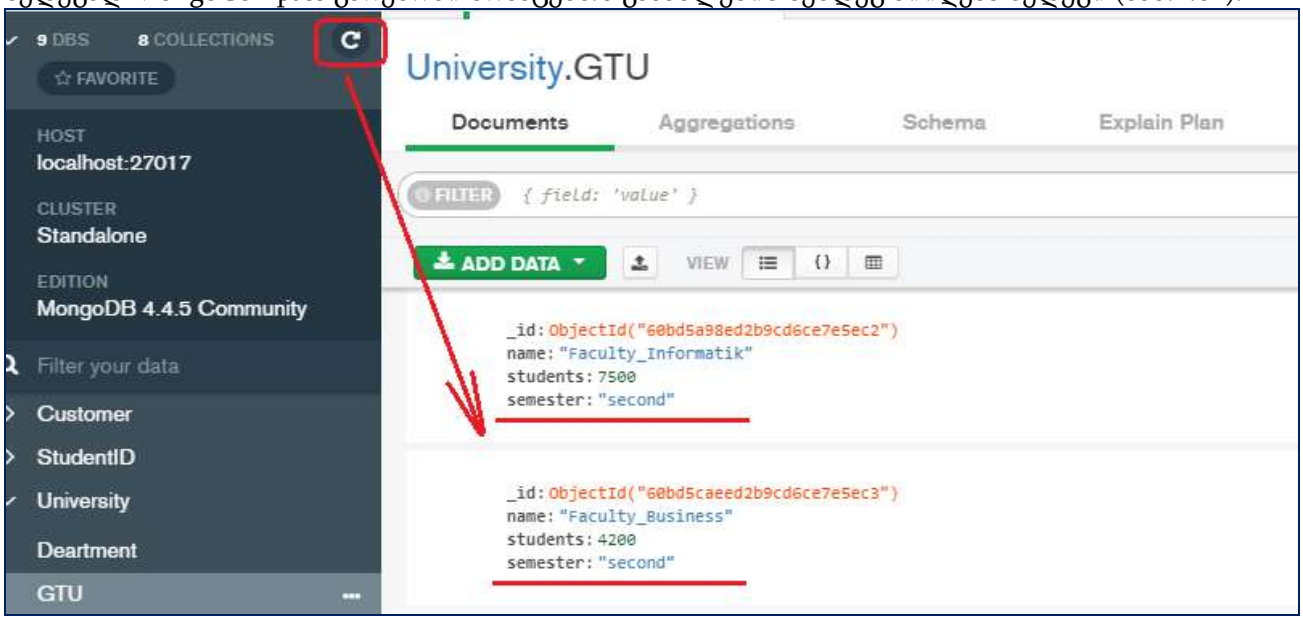
ნახ.7.62

განვაახლოთ მონაცემები, პირობით, ორივე ფაკულტეტზე პირველის ნაცვლად გამოიტანოს მეორე (second) სემესტრი. რომლის შესაბამისი კოდის ჩაწერის სინტაქსს აქვს შემდეგი სახე (ნახ.7.63).



ნახ.7.63

როგორც ბოლო სტრიქონზე გამოტანილი ჩანაწერიდან ჩანს განახლდა 2 მნიშვნელობა, შედეგად MongoCompass გარემოში მონაცემთა განახლების შემდეგ მიიღება შედეგი (ნახ.7.64).

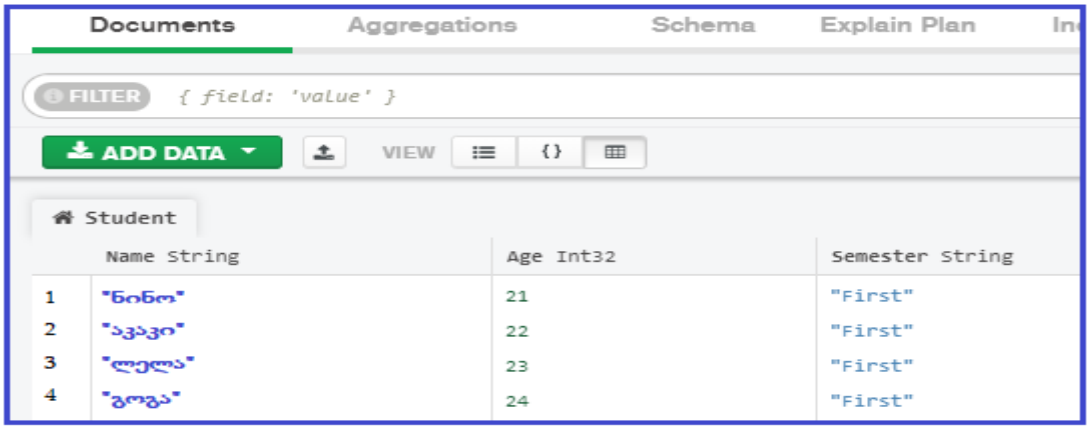


ნახ.7.64

- **delete( ), deleteMany( ) - ფუნქციები**

MongoDB Shell ინტერფეისზე ობიექტების წაშლისთვის გამოიყენება ორი ძირითადი ფუნქცია: deletOne() და deletMany(). ამ შემთხვევაშიც ლოგიკა იგივეა, deleteOne()-ის გამოყენებით წაიშლება ერთი ობიექტი, ხოლო deletMany () შლის რამდენიმე ობიექტს.

კოდის ჩაწერისას წარმოდგენილი უნდა იყოს კოლექციის დასახელება და ობიექტის ის მნიშვნელობა რომლის წაშლაც გვინდა, მაგალითად, კოლექციიდან რომლის სახელიცაა “Student” გვინდა წავშალოთ ობიექტი, რომლის სახელიცაა „კკკი“ (ნახ.7.65).

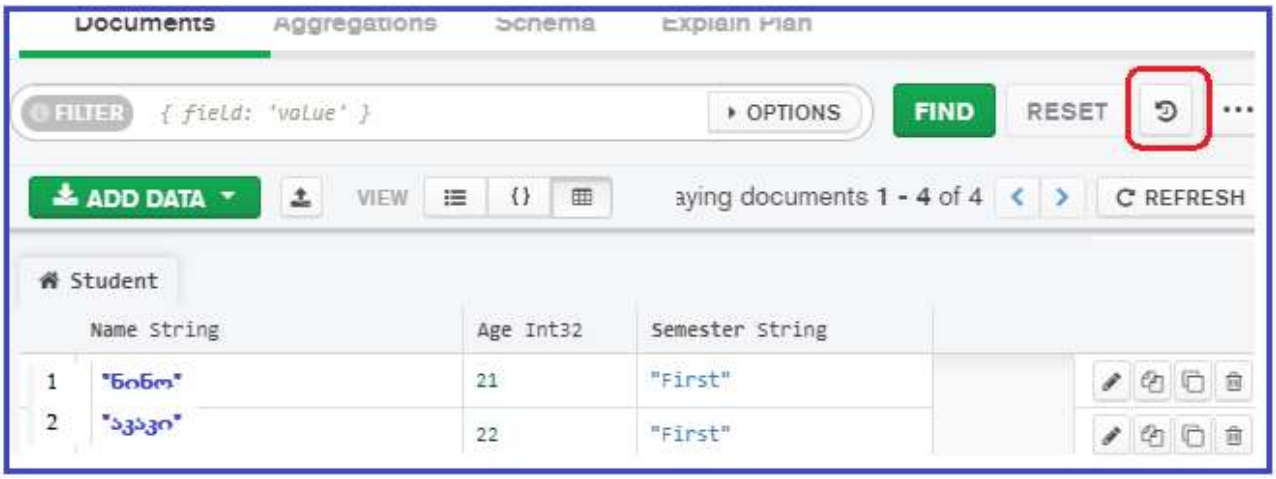


ნახ. 7.65

წაშლის კოდი ასე ჩაიწერება:

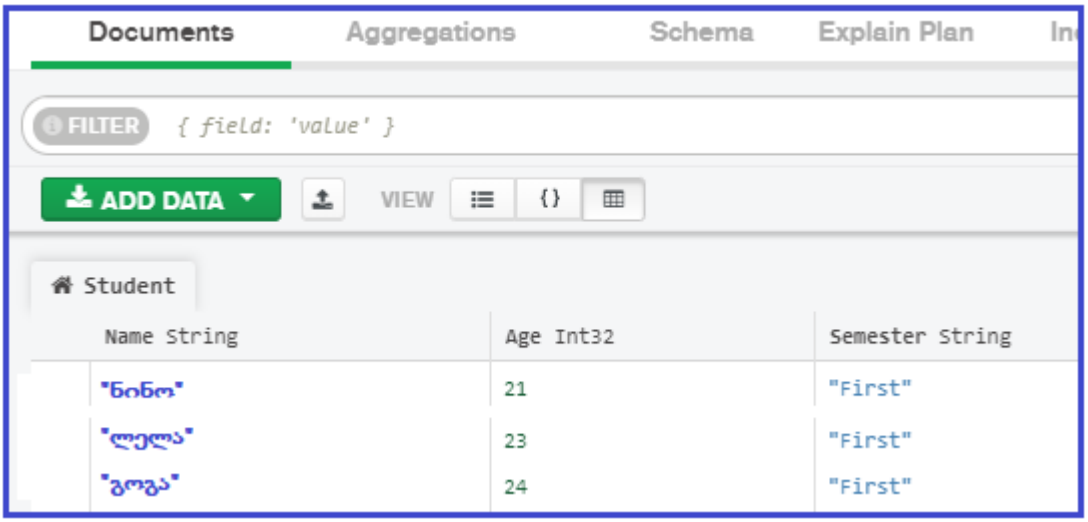
```
> db.Student.deleteOne({Name:'Akaki'})
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

მიღებული შეტყობინების შემდეგ: **{ "acknowledged" : true, "deletedCount" : 1 },** გადავდივართ Compass ინტერფეისზე და ვააქტიურებთ განახლების ღილაკს (ნახ.7.66).



ნახ.7.66

შედეგად წაიშლება ობიექტი, რომლის სახელია „*აკაკი*” (ნახ.7.67).



ნახ.7.67

რამდენიმე ობიექტის წაშლის შემთხვევაში გამოიყენება ფუნქცია deleteMany(). მაგალითად, თუ გვინდა კოლექციიდან ერთდროულად წავშალოთ ობიექტები, რომელთა ასაკი მეტია 20 -ზე და ნაკლებია 24, კოდი ჩაიწერება შემდეგი სახით:

```
> db.Student.deleteMany({Age:{$gt:20, $lt:24}})
{ "acknowledged" : true, "deletedCount" : 2 }
>
```

კოდში მოცემული ჩანაწერი, **$gt** ნიშნავს **(Greater Than)** მეტია მითითებულ მნიშვნელობაზე , ხოლო - **$lt (Less Than)** ნაკლებია მითითებულ მნიშვნელობაზე. Compass ინტერფეისზე მონაცემთა განახლების შემდეგ ვნახავთ, რომ ობიექტები წაშლილია (ნახ.7.68).

ნახ.7.68

7.3.5. ლაბ-N15: MongoDB Shell მონაცემთა ძებნა და ფილტრაცია (Find ფუნქცია და ლოგიკური ოპერატორები)

Find ფუნქციის გამოყენება მომხმარებელს აძლევს საშუალებას, მოიძიოს როგორც კოლექციაში წარმოდგენილი ობიექტები, ასევე დოკუმენტებში ობიექტების შესახებ არსებული ინფორმაცია.

როდესაც MongoDB Shell ინტერფეისზე გვინდა აისახოს კოლექციაში მოძებნილი მონაცემების სრული ჩანაწერი, საინტაქსს აქვს შემდეგი სახე:

db. (კოლექციის დასახელება). find()

რაც ილუსტრირებულია 7.66 ნახაზზე.

ჩანაწერით **use Cells** - ვააქტიურებთ ბაზას, რომლის კოლექციიდან გვინდა ინფორმაციის გამოტანა.

db.Staff.find() – Staff (კოლექციის დასახელება), კოლექციიდან, როდესაც გვინდა სრული ჩანაწერის გამოტანა find - ში არგუმენტს ვტოვებთ ცარიელს.

ნახ. 7.69

როგორც სურათიდან ჩანს კოლექციაში Staff - წარმოდგენილი იყო ექვსი დოკუმენტი, რაც მთლიანად აისახა ინტერფეისზე.

იმ შემთხვევაში თუ გვინდა, რომ გამოვიტანოთ არა სრული ჩანაწერი არამედ პირველი სამი ლიმიტირებული ჩანაწერი, შესაბამისი კოდი ჩაიწერება შემდეგი სახით:

db.Staff.find().limit(3)

```
> db.Staff.find().limit(3)
{ "_id" : ObjectId("60c4e8a42c6520a6174be352"), "Name" : "Giorgi", "Age" : 45, "Salary" : 2890
, "Country" : "Germany" }
{ "_id" : ObjectId("60c4e9412c6520a6174be353"), "Name" : "Mariami", "Age" : 29, "Salary" : 345
0, "Country" : "Georgia" }
{ "_id" : ObjectId("60c4ea242c6520a6174be354"), "Name" : "Ana", "Age" : 41, "Salary" : 2385, "
Country" : "Georgia" }
```

საკმაოდ დიდი ადგილი უკავია საიდენტიფიკაციო (ID) ჩანაწერს, რაც მომხმარებლისთვის ნაკლებად საინტერესოა. ინტერფეისი გვაძლევს საშუალებას, ჩავწეროთ კოდი ისეთი სახით, რომ არ გამოიტანოს ID. კოდს ექნება შემდეგი სახე:

db.Staff.find({}, { _id: 0}), limit(3)

კოდის გაშვების შემდეგ მიიღება შედეგი:

```
> db.Staff.find({}, {_id:0}).limit(3)
{ "Name" : "Giorgi", "Age" : 45, "Salary" : 2890, "Country" : "Germany" }
{ "Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
{ "Name" : "Ana", "Age" : 41, "Salary" : 2385, "Country" : "Georgia" }
>
```

ფუნქცია Find-ი ძებნის გარდა გამოიყენება მონაცემთა სორტირებისა და ფილტრაციისთვის. მაგალითად, თუ გვინდა ობიექტები დავალაგოთ ხელფასის (Salary) შესაბამისად ზრდადობით, კოდი ჩაიწერებათ შემდეგი სახით:

db. Staff.find({}, {_id: 0}).sort({Salary: 1})

როგორც ქვემოთ სურათზეა მოცემული, ხელფასის შესაბამისი მნიშვნელობები დალაგდა ზრდადობით, რაც სორტირების ფუნქციაში „1“-ანის ჩაწერით განხორციელდა, ხოლო კლებადობით დალაგებისთვის უნდა ჩაიწეროს „-1“.

```
> db.Staff.find({},{_id:0}).sort({Salary:1})
{ "Name" : "Lika", "Age" : 28, "Salaty" : 2550, "Country" : "Italy" }
{ "Name" : "Ana", "Age" : 41, "Salary" : 2385, "Country" : "Georgia" }
{ "Name" : "Tamara", "Age" : 32, "Salary" : 2650, "Country" : "Spain" }
{ "Name" : "Giorgi", "Age" : 45, "Salary" : 2890, "Country" : "Germany" }
{ "Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
{ "Name" : "Nika", "Age" : 23, "Salary" : 4125, "Country" : "France" }
>
```

იმ შემთხვევაში თუ გვინდა მონაცემები დავალაგოთ ანბანის მიხედვით მოწესრიგებულად (ა-ჰ) და ასევე ასაკის მიხედვით კლებადობით, კოდი ჩაიწერება შემდეგი სახით:

db. Staff.find({}, {_id: 0}). sort({Age: -1, Name: 1})

```
> db.Staff.find({}, {_id:0}).sort({Name:1, Age:-1})
{ "Name" : "Ana", "Age" : 41, "Salary" : 2385, "Country" : "Georgia" }
{ "Name" : "Giorgi", "Age" : 45, "Salary" : 2890, "Country" : "Germany" }
{ "Name" : "Lika", "Age" : 28, "Country" : "Italy", "Salary" : 2550 }
{ "Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
{ "Name" : "Nika", "Age" : 23, "Salary" : 4125, "Country" : "France" }
{ "Name" : "Tamara", "Age" : 32, "Salary" : 2650, "Country" : "Spain" }
>
```

როგორც აქედან ჩანს, სახელები დალაგდა ანბანის მიხედვით, და მათი შესაბამისი ასაკი კლებადობით.

ფუნქცია Find ასევე გამოიყენება ფილტრაციის პირობის შესრულებისას, მაგალითად, თუ გვინდა გამოვიტანოთ ობიექტი, რომელსაც ქვია „**მარიამი**“ და არის **29** წლის, კოდი ჩაიწერება შემდეგი სახით:

db. Staff.find({Name: ‘Mariami’, Age: 29}, {_id: 0})

შედეგად მიიღება:

```
> db.Staff.find({Age:29, Name:'Mariami'}, {_id:0})
{ "Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
>
```

გამოიტანა მხოლოდ ერთი ჩანაწერი, რადგან ფილტრაციის პირობას მხოლოდ ერთი შეესაბამებოდა.

ლოგიკური ოპერატორი **Or** გამოიყენება, როდესაც მოცემული მნიშვნელობებიდან ერთ-ერთი გვინდა შედეგზე გამოვიტანოთ. მაგალითად, გამოიტანოს ჩანაწერი რომელსაც ჰქვია **გიორგი** და ასევე ის თანამშრომელი, რომელიც არის **29** წლის.

კოდში ლოგიკური ოპერატორის გამოყენებისას წინ იწერება დოლარის (**$**) ნიშანი. კოდს ექნება შემდეგი სახე:

db. Staff.find({ $or: [{ Age: 29}, {Name: ‘Giorgi’ }]}, {_id: 0})

```
> db.Staff.find({$or: [{Age:29}, {Name:'Giorgi'}]}, {_id:0})
{ "Name" : "Giorgi", "Age" : 45, "Salary" : 2890, "Country" : "Germany" }
{ "Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
```

ფილტრაციის პირობების განსაზღვრისას საყურადღებოა ის ფაქტი, რომ კოდში შესაძლებელია გამოვიტანოთ სხვადასხვა კრიტერიუმი, მაგალითად, ნაკლებია რომელიმე კონკრეტულ მნიშვნელობაზე, ჩაიწერება სიმბოლოებით - **$lt** (ნიშნავს, *Less Than,* ე.ი. გამოიტანოს ყველა ის თანამშრომელი რომლის ასაკი ნაკლებია 40 წელზე. შესაბამის კოდს ექნება შემდეგი სახე:

db. Staff.find({ $or: [{ Age: {$lt: 40}}, {Name: ‘Giorgi’ }]}, {_id: 0})

```
db.Staff.find({$or: [{Age: {$lt:40}}, {Name:'Giorgi'}]}, {_id:0})
"Name" : "Giorgi", "Age" : 45, "Salary" : 2890, "Country" : "Germany" }
"Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
"Name" : "Lika", "Age" : 28, "Country" : "Italy", "Salary" : 2550 }
"Name" : "Nika", "Age" : 23, "Salary" : 4125, "Country" : "France" }
"Name" : "Tamara", "Age" : 32, "Salary" : 2650, "Country" : "Spain" }
```

როგორც მიღებული შედეგიდან ჩანს გამოიტანა ყველა თანამშრომლის შესაბამისი ინფორმაცია, რომელიც იყო 40 წელზე ნაკლები ასაკის, გარდა „**გიორგი**“-სა, რადგან იგი კოდის პირობაში წერია, რომ უნდა გამოიტანოს თანამშრომელი სახელად - გიორგი.

ფილტრაციის სხვა მნიშვნელობის გამოტანისას გამოიყენება აღნიშვნები:

- მეტია - **$gt** (Greate Then).
- მეტია ან ტოლი - **$gte (**Greate Then or Equal To)
- ნაკლებია ან ტოლი - **$lte (**Less Then or Equal To)
- ტოლია - **$eq** (Equal)
- არ უდრის - **$ne** (Does Not Equal)

ლოგიკური ოპერატორი **in** გამოიყენაბა იმ შემთხვევაში, როდესაც მოცემული მნიშვნელობებიდან გვინდა გამოვყოთ რამდენიმე (მაგალიტად, თანამშრომელი) (ნახ.7.70).

Name String	Age Int32	Salary Int32	Country String
"Giorgi"	45	2890	"Germany"
"Mariami"	29	3450	"Georgia"
"Ana"	41	2385	"Georgia"
"Lika"	28	2550	"Italy"
"Nika"	23	4125	"France"
"Tamara"	32	2650	"Spain"

ნახ. 7.70

მოთხოვნა: გამოვიტანოთ თანამშრომლები, რომლებიც ცხოვრობენ საქართველოში, გერმანიასა და იტალიაში. კოდს ექნება შემდეგი სახე:

db.Staff.find ({ Country: {$in: ['Georgia', 'Germany', 'Italy']}}, {_id:0})

```
> db.Staff.find({Country:{$in:['Georgia', 'Germany', 'Italy']}}, {_id:0})
{ "Name" : "Giorgi", "Age" : 45, "Salary" : 2890, "Country" : "Germany" }
{ "Name" : "Mariami", "Age" : 29, "Salary" : 3450, "Country" : "Georgia" }
{ "Name" : "Ana", "Age" : 41, "Salary" : 2385, "Country" : "Georgia" }
{ "Name" : "Lika", "Age" : 28, "Country" : "Italy", "Salary" : 2550 }
>
```

შედეგად გამოიტანა ორი ობიექტი საქართველოდან, ხოლო დანარჩენი ის ობიექტები, ვინც არიან გერმანიიდან და იტალიიდან, ხოლო კოდში თუ ჩავწერთ **$nin**, ეს ნიშნავს უარყოფას, ანუ გამოიტანოს ის ობიექტები, რომლებიც არ არიან ამ კონკრეტული ქვეყნებიდან.

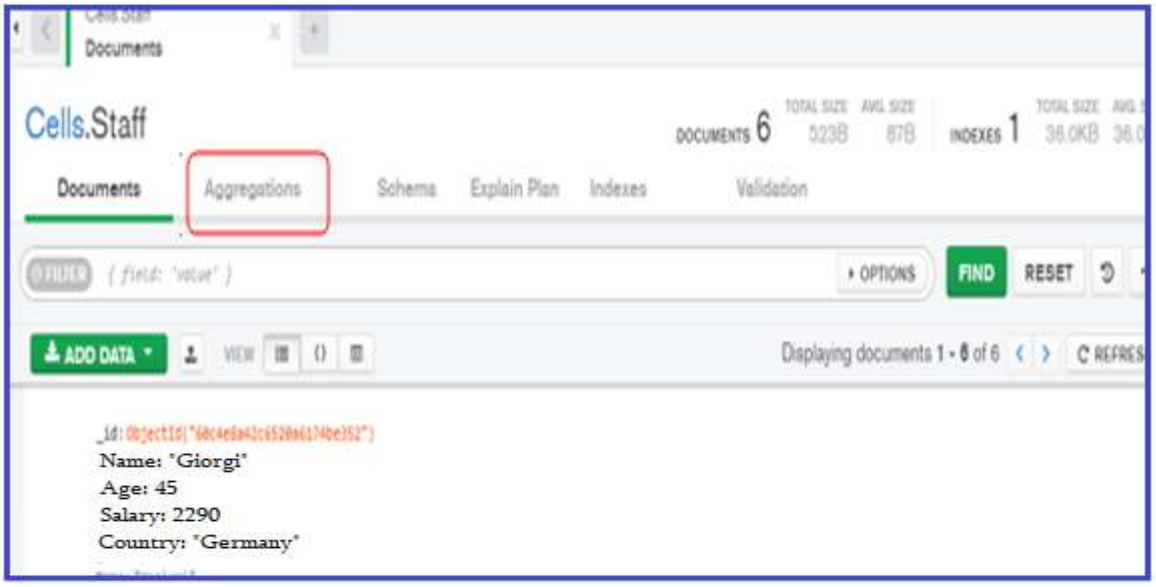
ლოგიკური ოპერატორი and – გამოიყენება, როდესაც მოცემული ობიექტებიდან ზუსტად ხდება განსაზღვრა რომელი მნიშვნელობის გამოტანა გვინდა. მაგ. გამოვიტანოთ ობიექტი, რომელიც არის 28 წლის და ქვია ლიკა. შესაბამის კოდს აქვს შემდეგი სახე:

db. Staff.find({ $and: [{ Age: 28}, {Name: ‘Lika’ }]}, {_id: 0})

```
> db.Staff.find({$and: [{Age:28}, {Name:'Lika'}]}, {_id:0})
{ "Name" : "Lika", "Age" : 28, "Country" : "Italy", "Salary" : 2550 }
>
```

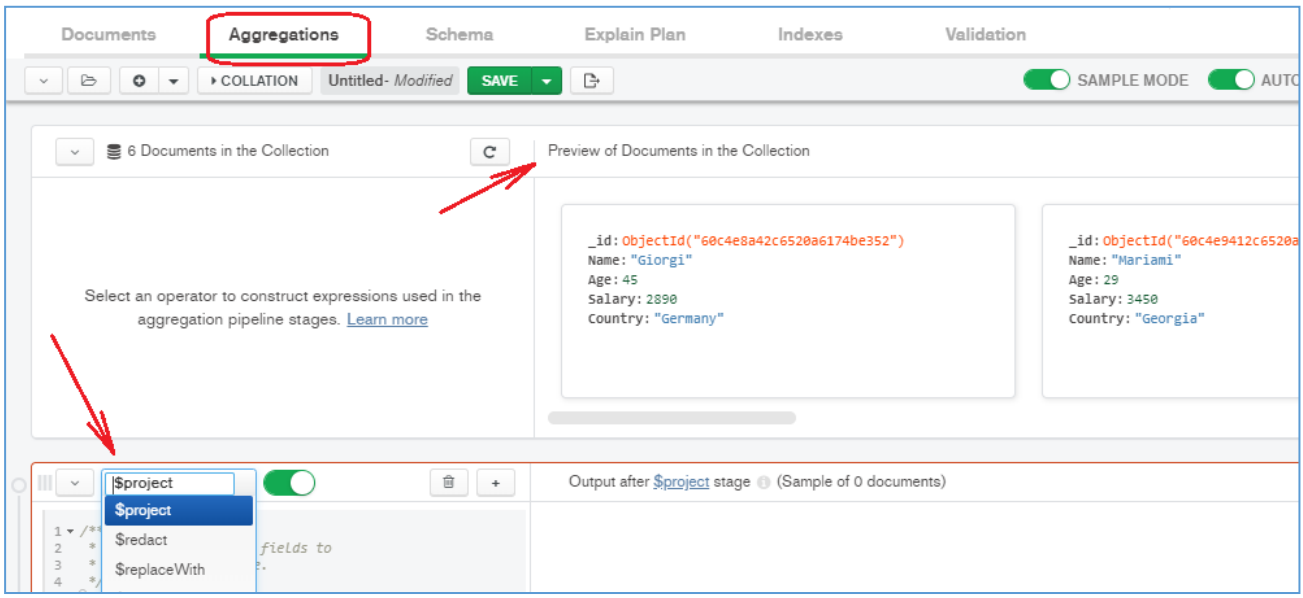
- **Compass ინტერფეისი (ძებნა, ფილტრაცია, სორტირება)**

MongoDB Compass-ის გრაფიკულ ინტერფეისით შესაძლებელია აგრეთვე ძებნის, ფილტრაციის და სორტირების პირობების შესრულება. Compass ინტერფეისზე მოცემულია ჩანართი - Aggregatios (ნახ.7.71).



ნახ.7.71

მისი გააქტიურების შემდეგ გადავდივართ ე.წ. კონვეიერში, სადაც სხვადასხვა ოპერატორის და ფუნქციის გამოყენებით, შესაძალებელია კოლექციაში მოცემული ობიექტების შესაბამისი ველების გამოტანა. ფანჯრის მარჯვენა მხარეს, პირველ სტრიქონზე ჰორიზონტალურად გამოტანილია კოლექცია და მასში წარმოდგენილი დოკუმენტები, ხოლო ქვედა მარცხენა მახარეს, ოპერატორების მითითებების შემდეგ, გამოჩნდება ის ობიექტები, რომლის შესახებაც გვინდა კონკრეტული ინფორმაციის წარმოდგენა (ნახ.7.72).



ნახ.7.72

$project მრავალფუნქციური ოპერატორია, რომლის დახმარებით შეიძლება გამოვიტანოთ კონკრეტული ველები. ველი რომელიც არ გვინდა გამოჩნდეს, ვანიჭებთ მნიშვნელობას - 0, ხოლო 1 -ის შემთხვევაში გამოიტანს შესაბამის მნიშვნელობას.

მაგალითად, თუ გვინდა გამოვიტანოთ ინფორმაცია მხოლოდ თანამშრომელთ ასაკის და ხელფასის შესახებ და ამავე დროს არ გვინდა გამოჩნდეს ID, მოთხოვნა ჩაიწერება შემდეგი სახით (ნახ.7.73).

ნახ.7.73

მონაცემთა აგრეგაციასთან დაკავშირებულ სხვა პარამეტრებს განვიხილავთ მომდევნო სამუშაოზე.

7.3.6. ლაბ-N16: MongoDB მონაცემთა აგრეგაცია

აგრეგაციის ოპერაციები ამუშავებს დოკუმენტებში წარმოდგენილ მონაცემებს და აბრუნებს გამოთვლილ შედეგებს. იგი მოთხოვნის შესაბამისად აჯგუფებს სხვადსხვა დოკუმენტებში არსებულ მონაცემებს და შედეგად გამოიტანს ერთი ჩანაწერის სახით.

MongoDBაგრეგაციის პროცესის შესასრულებლად მომხმარებელს სთავაზობს სამ გზას:

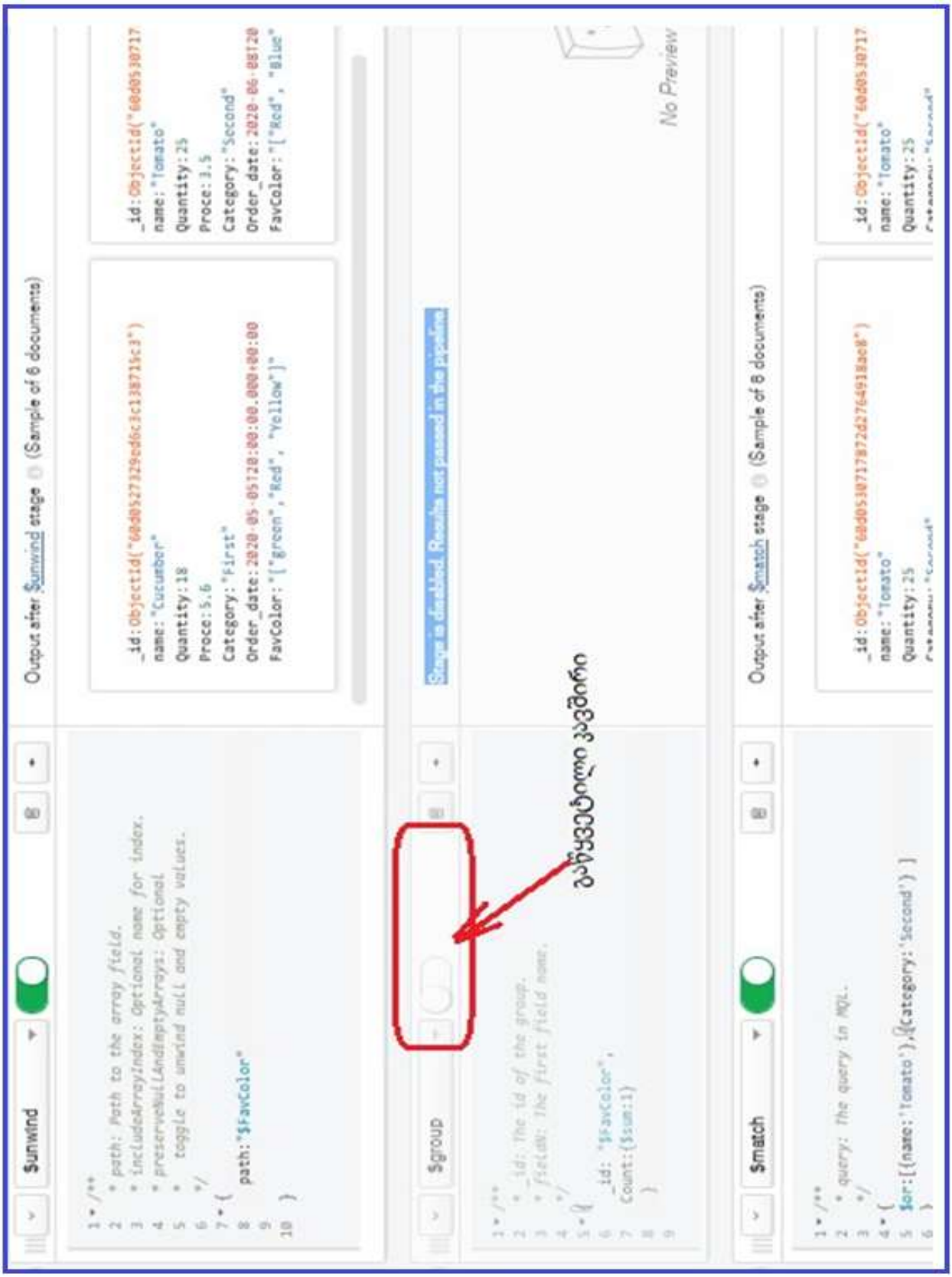
1) აგრეგაციის „მილი“ (pipeline);

2) მაპ-რედუს ფუნქცია და

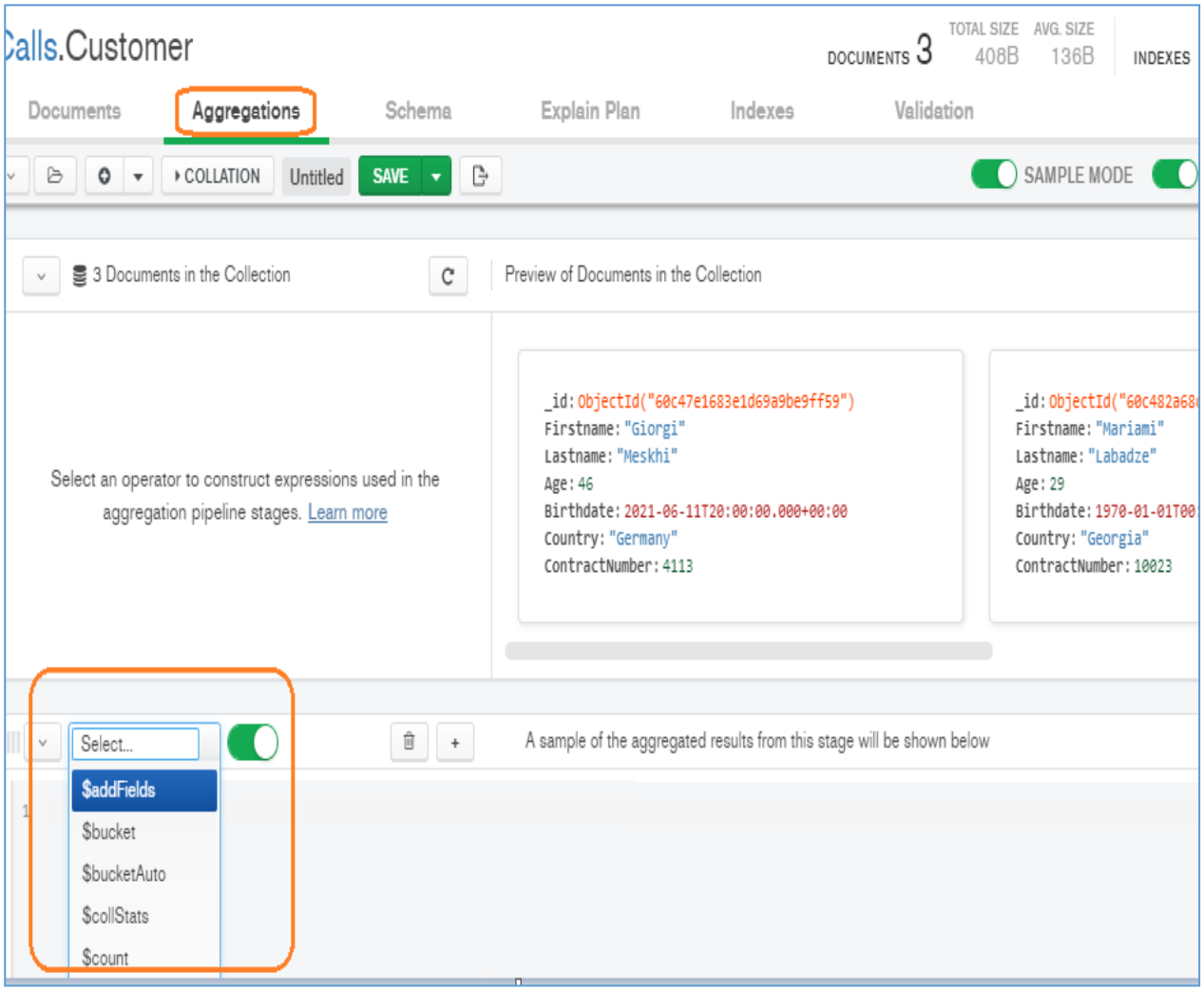
3) ერთი დანიშნულების აგრეგაციის მეთოდები.

მილისებური აგრეგაცია ნიშნავს, რომ წინა აგრეგირებული მონაცემები აქტუალურია მომდევნოაგრეგაციისთვის, მილისებური სწორედ მაგიტომ ჰქვია, რომ მონაცემთა აგრეგირება ხდება მიმდევრობით, თუ არ მოხდა ხელით მათ შორის კავშირის გაწყვეტა (ნახ.7.74).

აგრეგაცია არის მონაცემთა ფორმირების ეტაპები, რომელთა რეალიზება შესაძლებელია სხვადასხვა ოპერატორის გამოყენებით. ქვემოთ მოცემულ ინტერფეისზე წარმოდგენილია აგრეგაციის ჩანართი, რომლის $select ველიდან შესაძლებელია ოპერატორების გამოყენება და მონაცემთა ფორმირება (ნახ.7.75).


$unwind
Output after $unwind stage (Sample of 6 documents)
$group
Stage is disabled. Results not passed in the pipeline
$match
Output after $match stage (Sample of 8 documents)
No Preview
ველების კავშირი

ნახ.7.74



ნახ. 7.75

ოპერატორები ერთიანდება ჯგუფებად, არის ოპერატორები, რომლებიც გამოიყენება რაოდენობრივი, ტექსტური და ზოგადი მნიშვნელობების ფორმირებისთვის.

7.1 ცხრილში მოცემულია აგრეგაციის ფუნქციები მათი დანიშნულება და ჩაწერის სინტაქსი.

დოკუმენტების აგრეგაციისთვის გამოიყენება შემდეგი კონცეფციები:

- **$project** - გამოიყენება კოლექციიდან რამდენიმე სპეციალური ველის ამორჩევისთვის
- **$match** -ფილტრაციის ოპერაცია , ახდენს ფილტრაციას სხვადასხვა კრიტერიუმების საფუძველზე
- **$group** - მონაცემთა დაჯგუფება
- **$sort** - დოკუმენტების სორტირება
- **$skip** - აღნიშნული ბრძანებით შესაძლებელია დოკუმენტთა სიმრავლიდან რამდენიმეს უგულებელყოფა
- **$limit** - ზღუდავს დოკუმენტების რაოდენობას, დოკუმენტების გამოტანას ახდენს მითითებული მნიშვნელობის შესაბამისად საწყისი პოზიციიდან.
- **$unwind** - გამოიყენება ისეთ დოკუმენტებში, სადაც წარმოდგენილია მასივები და ახდენს მასივთა მნიშვნელობების გამოტანას.

აგრეგაციის ფუნქციები და მათი დანიშნულება ცხრ.7.1

ფუნქცია	აღწერა	გამოყენების ფორმა
$sum	აჯამებს კოლექციაში ყველა დოკუმენტში არსებულ მითითებულ მნიშვნელობას	db. კოლექციის დასახელება. aggregate([{$group : {_id : “$title”, total_salary : {$sum : “$salary”}}}])
$avg	გამოითვლის კოლექციაში ყველა დოკუმენტში მითითებული ველის საშუალო მნიშვნელობას	db. კოლექციის დასახელება.aggregate([{$group : {_id : “$title”, avg_salary : {$avg : “$salary”}}}])
$min	მიიღება დოკუმენტებში მოცემული ერთეულების მინიმალური მნიშვნელობა	db. კოლექციის დასახელება. aggregate([{$group : {_id : “$title”, min_salary : {$min : “$salary”}}}])
$max	მიიღება დოკუმენტებში მოცემული ერთეულების მაქსიმალური მნიშვნელობა	db. კოლექციის დასახელება. aggregate([{$group : {_id : “$title”, max_salary : {$max : “$salary”}}}])
$push	მოცემულ დოკუმენტში სვავს მასივს	db. კოლექციის დასახელება. ggregate([{$group : {_id : “$title”, skills : {$push: “$skills”}}}])
$addToSet	მოცემულ დოკუმენტში, სვავს მასივს, მაგრამ არ ქმნის დუბლიკატს.	db. კოლექციის დასახელება. aggregate([{$group : {_id : “$title”, skills : {$addToSet : “$skills”}}}])
$first	გამოიყენება სორტირებისას, მოთხოვნაში, მოცემული დოკუმენტების ჩამონათვალიდან მიიღება პირველი დოკუმენტი.	db. კოლექციის დასახელება. ggregate([{$group : {_id : “$title”, first_skill : {$first : “$skills”}}}])
$last	გამოიყენება სორტირებისას, მოთხოვნაში, მოცემული დოკუმენტების ჩამონათვალიდან მიიღება ბოლო დოკუმენტი.	db. კოლექციის დასახელება. aggregate([{$group : {_id : “$title”, last_skill : {$last : “$skills”}}}])

მონაცემთა აგრეგაციის მარტივი ოპერატორია $sample, რომლის Size კრიტერიუმში მნიშვნელობის მინიჭებისას გამოიტანს შემთხვევითად იმდენ ობიექტს, რამდენ მნიშვნელობა-საც მივანიჭებთ, მაგალითად, თუ Size = 3, გამოიტანს სამ შემთხვევით ობიექტს (ნახ.7.76).

ნახ.7.76

ოპერატორი $match უზრუნველყოფს მოთხოვნის შესრულებას რამდენიმე პარამეტრის გათვალისწინებით. მას ფილტრაციის ოპერატორსაც უწოდებენ, იგი ამცირებს დოკუმენტების რაოდენობას მათ შემდეგ ბიჯზე აგრეგაციისთვის. მაგალითად, თუ გვინდა მოვიპოვოთ ინფორმაცია ობიექტებზე, რომელთა ხელფასი მეტია 2100-ზე და ცხოვრობს საქართველოში, მოთხოვნა $match ოპერატორის გამოყენებით ჩაიწერება შემდეგ სახით (ნახ.7.77).

ნახ.7.77

$match ოპერატორი ასევე გამოიყენება ლოგიკურ or და and ოპერატორებთან. მაგალითად თუ გვინდა გამოვიტანოთ name, როელიც შეესაბამება Tomato, ან კატეგორია არის Second, მოთხოვნა ჩაიწერება შემდეგი სახით (ნახ.7.78).

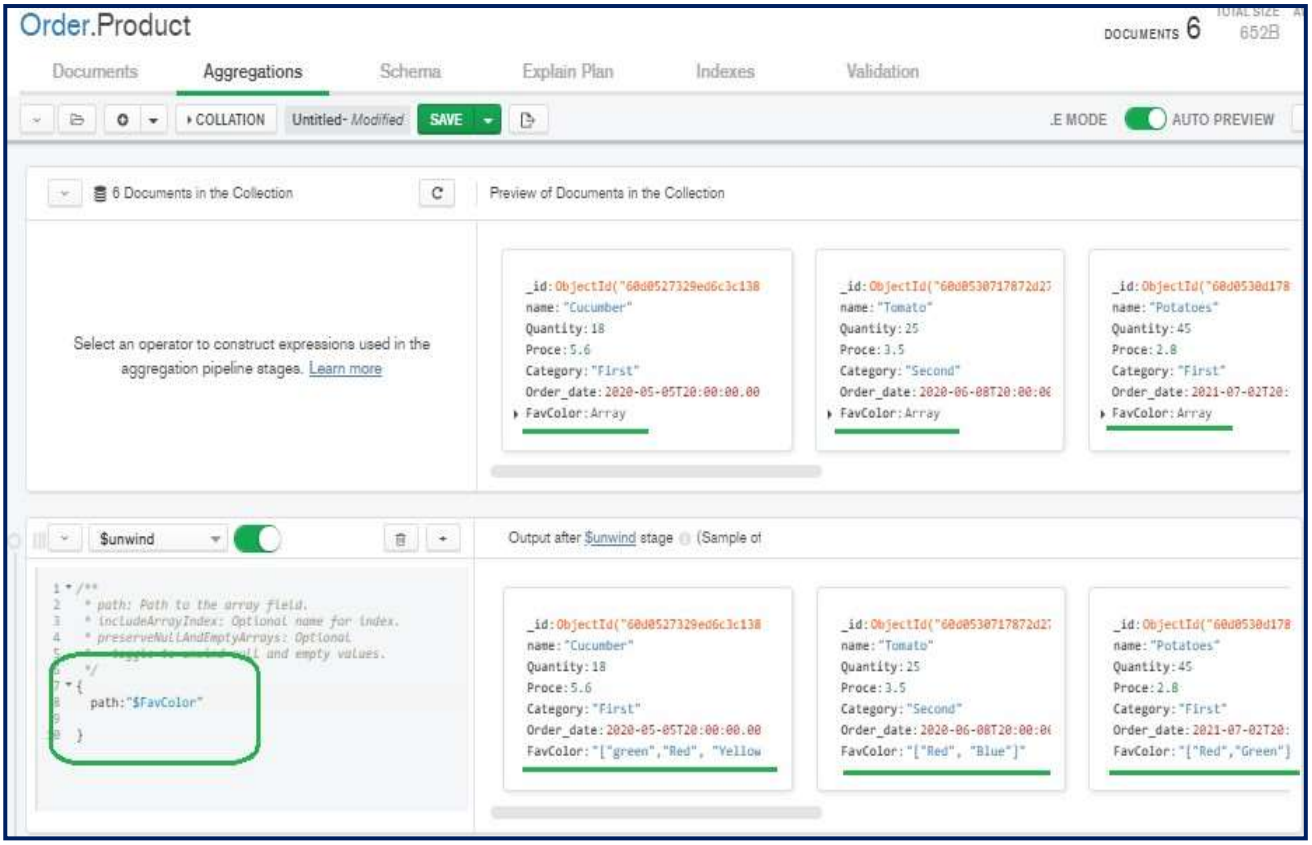
ნახ.7.78

შედეგად კი მიიღება, ყველა დოკუმენტი სადაც სახელი არის Tomato და კატეგორია Second (ნახ.7.79).

ნახ.7.79

იგივე პრინციპით ხდება მოთხოვნის ჩაწერა and ოპერატორის გამოყენებისას

ოპერატორი $unwind გამოიყენება მასივის ცალკეული ელემენტების წარმოდგენისა და მასივის ყოველი ელემენტისთვის ცალკეული “output” დოკუმენტების შესაქმნელად. მაგალითად, მასივის სახით, დოკუმენტებში არის წარმოდგენილი ინფორმაცია, თანამშრომლების რჩეული ფერების შესახებ, **$unwind** ოპერატორის გამოყენებით შეიძლება გამოვიტანოთ თანამშრომლების რჩეული ფერების ჩამონათვალი (ნახ.7.80).

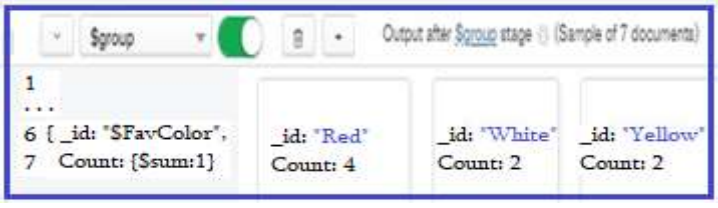


ნახ.7.80

ოპერატორი $group ერთ-ერთი ყველაზე გამოყენებადია, რომელიც იხმარება დაჯგუფების სხვადასხვა მოთხოვნის ჩაწერისას. მისი გამოყენებით შესაძლებელია მონაცემები დაჯგუფდეს კონკრეტული კრიტერიუმების შესაბამისად. მაგალითად, პერიოდის ან სახეობის მიხედვით. ასევე შესაძლებელია გამოვიტანოთ პროდუქციის რაოდენობის ან ღირებულების ჯამური მნიშვნელობა. მისი ჩაწერის სტრუქტურაა (ნახ.7.81).

```
{ $group:
   { _id: <expression>, // Group By Expression
      <field1>: { <accumulator1> : <expression1> },
       ...
   }
}
```

თუ გვინდა თანამშრომლთა არჩეული ფერების დაჯგუფება და გამოტანა, რამდენჯერ მოხდა ერთიდაიმავე ფერის არჩევა, მოთხოვნა უნდა ჩაიწეროს შემდეგი სახით (ნახ.7.81).



ნახ. 7.81

გამოყენებულია რაოდენობრივი ფუნქცია, რომლის არგუმენტში მითითითებულია ჯამის ფუნქცია, იგი აჯამებს ერთნაირ ფერებს. შედეგად მიიღება

როგორც შედეგიდან ჩანს „Red” აირჩია 4-მა, „White” და „Yelow” 2-მა და ა.შ.

7.4. ღრუბლოვანი ტექნოლოგია და MongoDB Atlas პლატფორმა

XXI საუკუნის დასწაყისიდან განსაკუთრები აქტუალური გახდა ღრუბლოვანი გამოთვლების (Cloud Computing) და, ზოგადად, ღრუბლოვანი ტექნოლოგიების (Cloud Tecchnology) მიმართულება. აქტიურად დაიწყო განვითარება ღრუბლოვანმა ტექნოლოგიებმა ვირტუალიზაციისა და კლასტერიზაციის თანამედროვე საშუალებების გამოყენებით, დიდ მონაცემთა ეკოსისტემებმა და მონაცემთა დამუშავების ცენტრების ეფექტური არქიტექტურების კვლევამ [13,17,18].

აპლიკაციის შექმნისას არსებითად მნიშვნელოვანია მონაცემთა ბაზის მოქნილი სერვისის შერჩევა. ბიზნეს პროცესების მართვისას ძალიან ეფექტურია ღრუბლოვანი NoSQL მონაცემთა ბაზის გამოყენება, რადგან უზრუნველყოფილია არარელაციური მონაცემთა ბაზის ყველა მახასიათებელის ხელმისაწვდომობა, არის მასშტაბური და მოქნილი.

ღრუბლოვანი პლატფორმის, როგორც ინტეგრირებული პორტალის გამოყენება უზრუნველყოფს სხვადასხვა აპლიკაციების დაკავშირებას მონაცემთა ბაზასთან, რომელიც შესაძლებელია რეალურ დროში მყისიერად განახლდეს და ადვილად მისაწვდომი გახდეს მომხმარებლისთვის.

- **ღრუბლოვანი მონაცემთა ბაზა MongoDB Atlas**

MongoDB Atlas პირველი ღრუბლოვანი ვერსიაა ამ ტიპის მონაცემთა ბაზებიდან. იგი მომხმარებელს აძლევს საშუალებას წარადგინოს ერთდროულად რამდენიმე აპლიკაცია მსხვილ ღრუბლოვან პროვაიდერებთან. MongoDB-ს მონაცემთა ბაზა შესაძლებელია განთავსდეს ისეთ ღრუბლოვან სერვერებზე, როგორიცაა AWS (Amazon Web Services), Azure ან GCP [18-21].

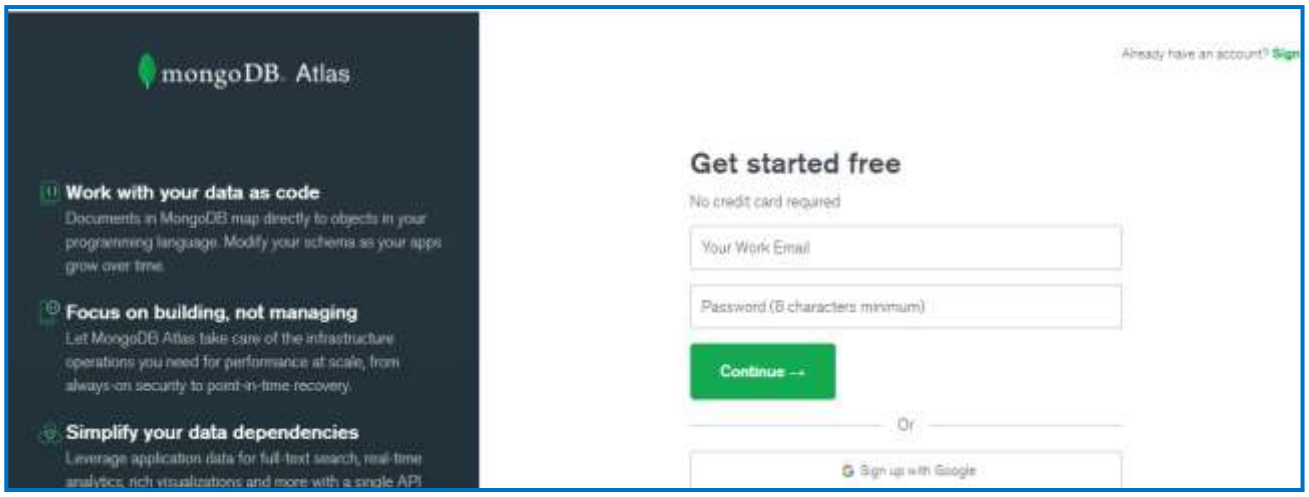
ღრუბლოვანი MongoDB Atlas, კლასტერიზაციის საშუალებით დეველოპერებს სთავაზობს ისარგებლონ, ყოველი პროვაიდერის მიერ წარმოდგენილი საუკეთესო აპლიკაციით, რაც უმარტივებთ და ეფექტურს ხდის მათ მუშაობას.

მულტიღრუბლოვანი კლასტერები კომპანიებს სთავაზობს მონაცემთა მიწოდების უპრეცედენტოდ მოქნილ და ეფექტურ სერვისს. ღრუბლოვან პროვაიდერებს შორის მონაცემთა გაცვლა და ბიზნეს მოთხოვნების დაკმაყოფილება საკმაოდ მნიშვნელოვანი და ძვირად ღირებული პროცესია, რასაც MongoDB Atlas პლატფორმა ეფექტურად ართმევს თავს.

MongoDB Atlas — როგორც აღვნიშნეთ არის გლობალური ღრუბლოვანი მონცემთა ბაზების სერვისი, რომელიც ეფექტურად გამოიყენება დანართებთან / აპლიკაციებთან სამუშაოდ. მისი დახმარებით მონაცემთა ბაზები იქმნება ძალიან სწრაფად, მათი დამუშავების და მართვისთვის საჭიროა მცირე დრო და რაც მთავარია, დაცულია უსაფრთხოების მაღალი სტანდარტით.

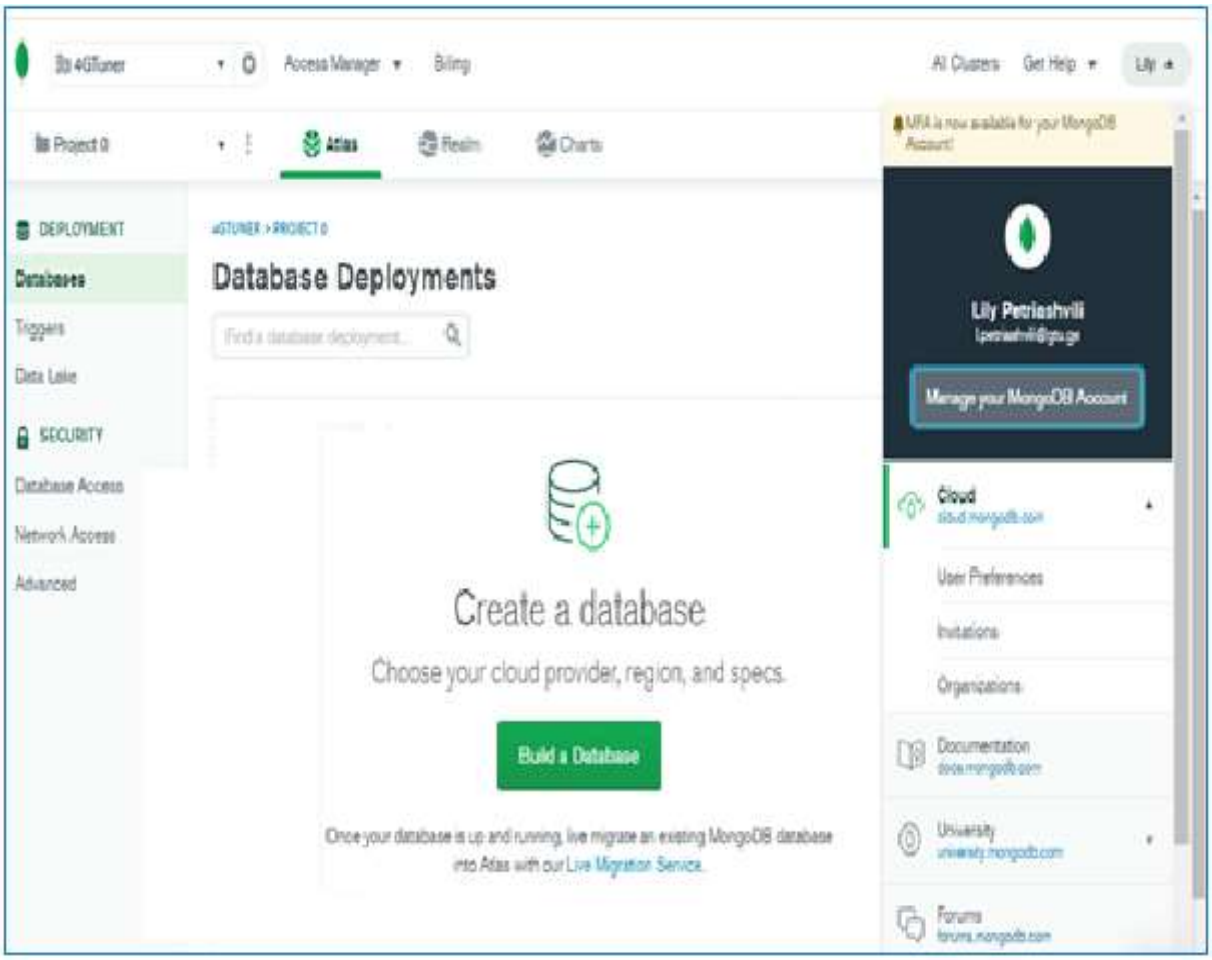
MongoDB Atlas პლატფორმა არის ე.წ. MongoDB მონაცემთა ბაზის სერვისი, რაც იმას ნიშნავს, რომ სერვისი ავტომატურად აკონფიგურირებს მონაცემთა ბაზას და განათავსებს მას როგორც უნიკალურს კონკრეტული მონაცემების განთავსებისთვის. აღნიშნული სერვისი მომხმარებელს ათავისუფლებს NoSQL მონაცემთა ბაზის დაპროექტების და მართვისთვის საჭირო სამუშაოების შესრულებისგან და საშუალებას აძლევს კონცენტრირდეს უშულოდ აპლიკაციის მართვაზე.

MongoDB Atlas პლატფორმის გამოყენებისთვის მომხმარებელი პირველ რიგში უნდა დარეგისტრირდეს თავის სამომხმარებლო ანგარიშით (ელ. ფოსტის მისამართი) და შეავსოს სარეგისტრაციო ველები. (https://www.mongodb.com/cloud/atlas) (ნახ.7.82).



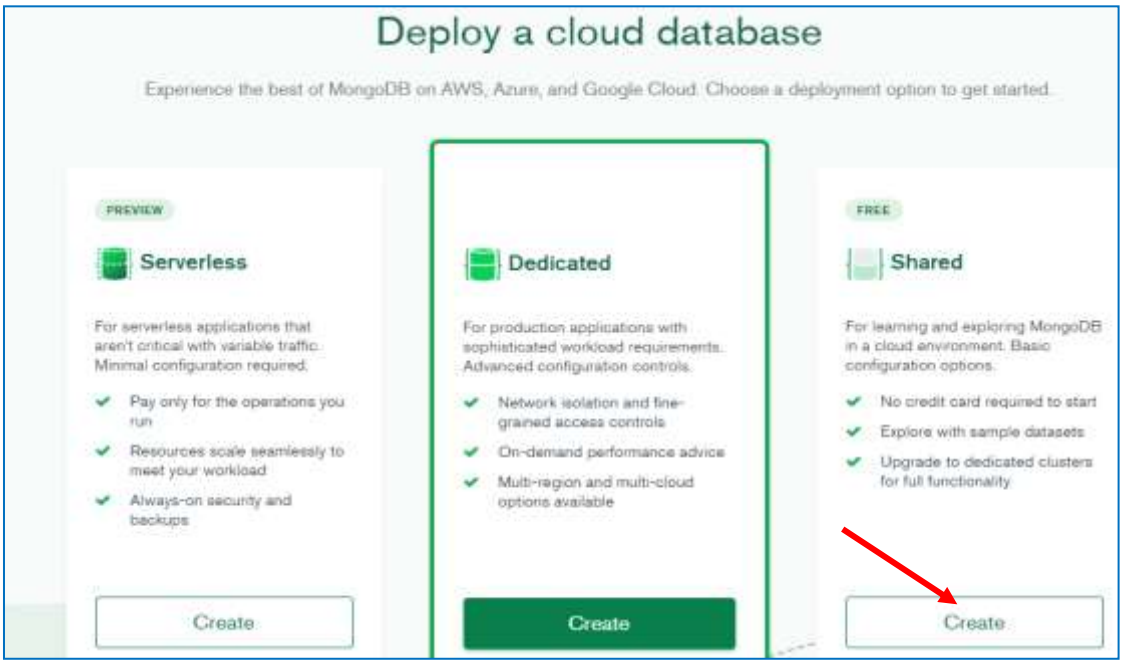
ნახ.7.82

რეგისტრაციის შემდეგ, როგორც 7.83 ნახაზზეა ნაჩვენები, მიიღება ფანჯარა საიდანაც ხდება მონაცემთა ბაზის აგება.



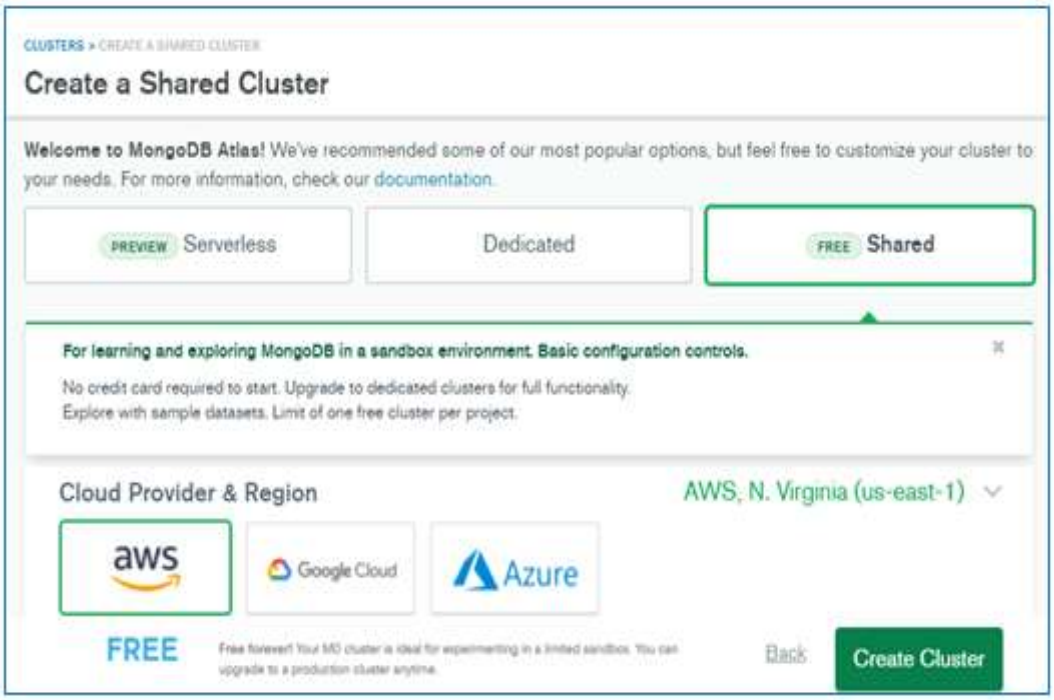
ნახ.7.83

ღილაკის **Build a Database** გადავდივართ გარემოში, სადაც შესაძლებელია შევარჩიოთ პლატფორმის გამოყენების სხვადასხვა სერვისები, ჩვენს შემთხვევაში ვაკეთებთ თავისუფალ არჩევანს, რომელიც განკუთვნილია არაკომერციულ სასწავლო პროცესისთვის და ნიმუშის შესაბამისად არჩევანს ვადასტურებთ ღილაკით Create (ნახ.7.84).



ნახ.7.84

მოცემული ჩანართის არჩევის შემდეგ გადავდივართ კლასტერის შექმნის და ფორმირების ეტაპზე. მიღებულ ფანჯარას აქვს შემდეგი სახე (ნახ.7.85).

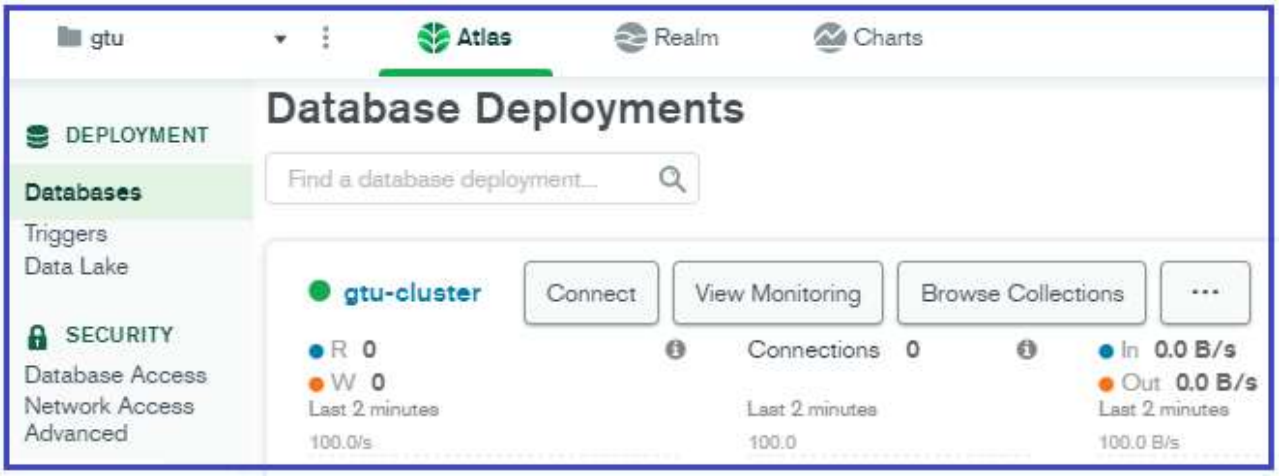


ნახ.7.85

გასათვალისწინებელია, კლასტერის შექმნის ღილაკის გააქტიურებამდე უნდა ავირჩიოთ ღრუბლოვანი პროვაიდერი, AWS (ამაზონის ვებ სერვისი), Google ან Azur, ჩვენს შემთხვევაში ვირჩევთ AWS, შემდეგ ფანჯრის ბოლოს ველში Cluster Name ჩავწეროთ კლასტერის დასახელება მაგ. gtu-claster (საყურადღებოა, კლასტერს სახელს ვარქმევთ სისტემის მიერ შემოთავაზებული რეკომენდაციის შესაბამისად მაგ. უნდა გამოვიყენოთ დაბალი რეგისტრის სიმბოლოები და რიცხვები, გამოტოვების გარეშე) (ნახ.7.86).

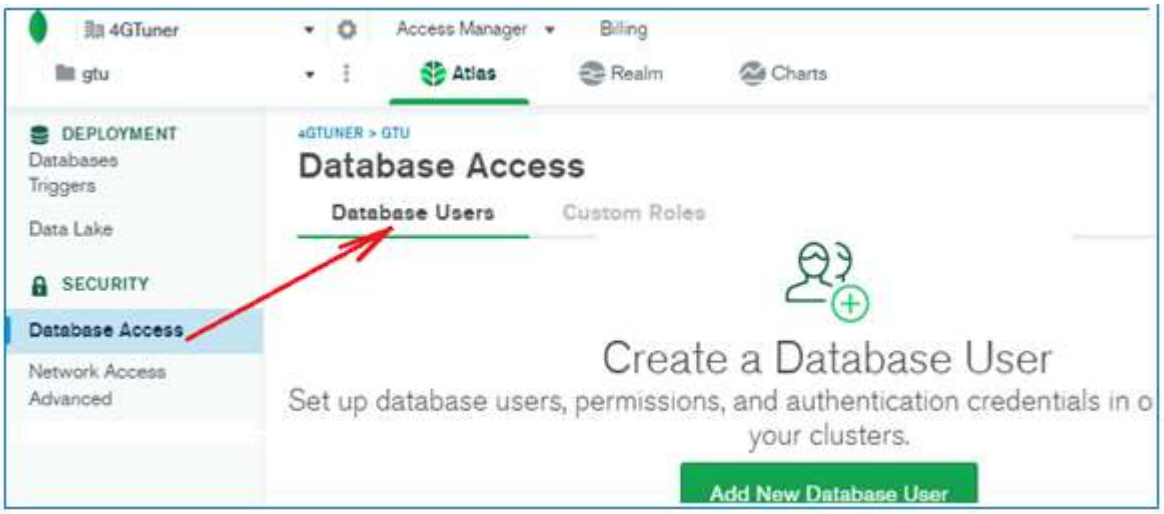
ნახ.7.86

Create Claster ღილაკის გამოყენებით გადავდივართ ფანჯარაში, სადაც გამოჩნდება ჩვენს მიერ ფორმირებული კლასტერი, სახელად „gtu-cluster” (ნახ.7.87).



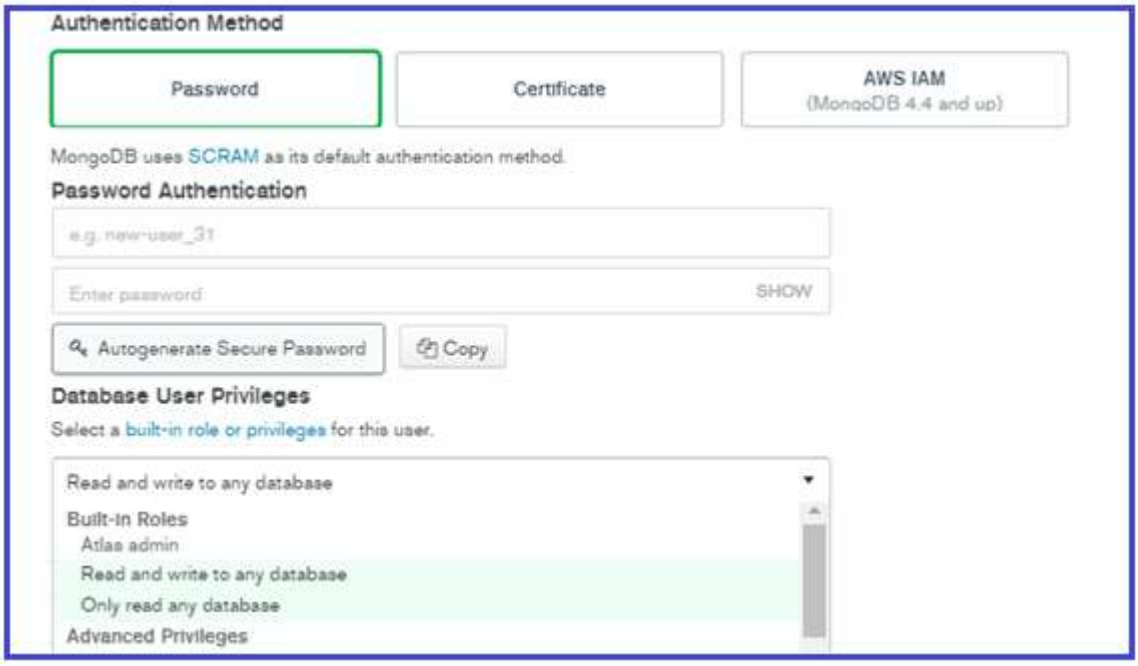
ნახ.7.86

კლასტერის შექმნის შემდეგ ფანჯრის მარცხენა მხარეს მოთავსებული **SECURITY** – **Database Access** ჩანართიდან უნდა განისაზღვროს სისტემის მომხმარებელთა ჩამონათვალი, ვისაც შეეძლებათ ინფორმაციის მიღება და ჩაწერა. აღნიშნული ჩანართის გააქტიურებისას ქვემოთ მოცემულ ფანჯარაში ვამატებთ მომხმარებლებს (ნახ.7.87).



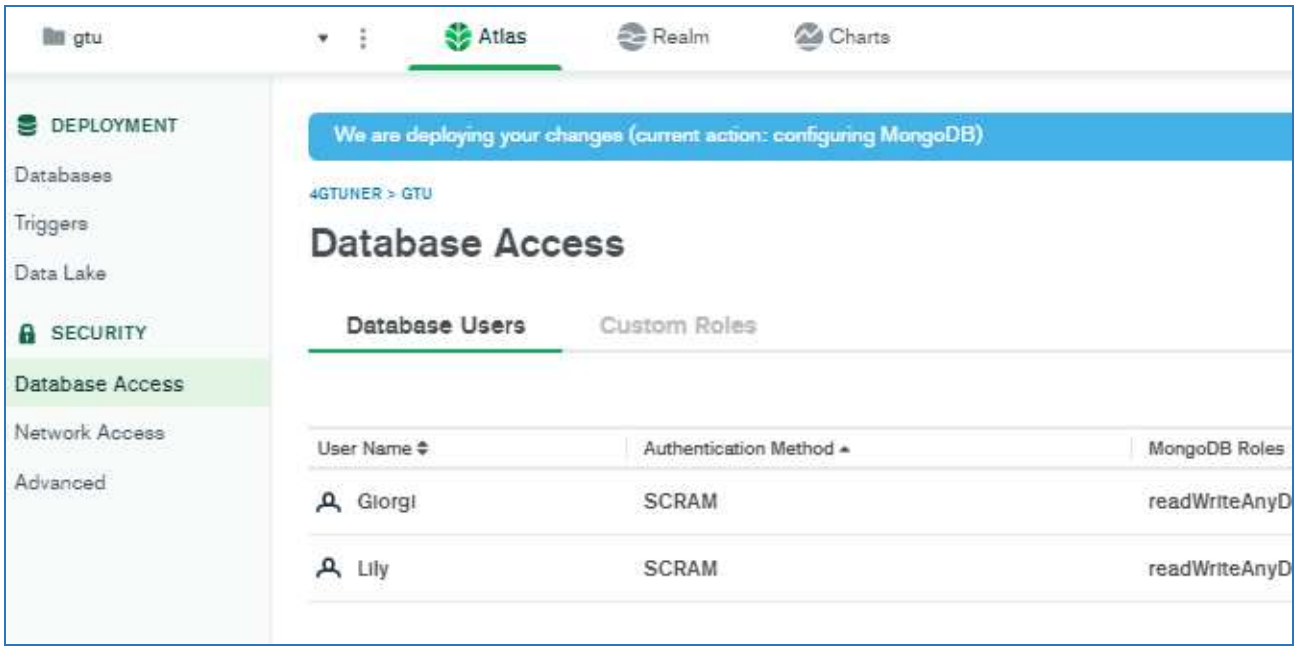
ნახ.7.87

ახალი მომხმარებლების შესახებ მონაცემთა განსაზღვრისას უნდა გავააქტიუროთ ქვემოთ მოცემული ღილაკი Add New Database User, რის შედეგადაც მიიღება ფანჯარა, რომლის ველებშიც შეგვაქვს მომხმარებლის სახელი და პაროლი (ნახ.7.88).



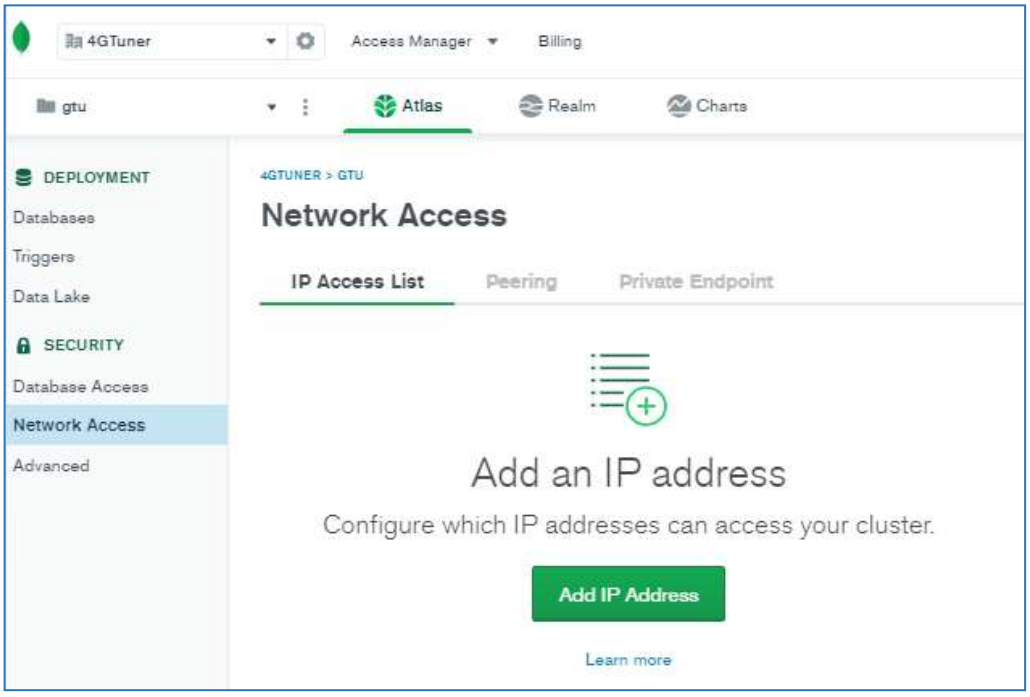
ნახ. 7.88

მოცემული ფანჯრიდან, ჩანართიდან Database User Privileges ყოველ მომხმარებელს შესაძლებელია მივანიჭოთ ან შევუზღუდოთ უფლებები, აღნიშნულ ჩანართში მოცემული შეთავაზებების შესაბამისად. მაგალითად, ქვემოთ მოცემულ სურათზე წარმოდგენილია ორი მომხმარებელი, რომელთაც წვდომა ექნება ჩვენს ღრუბლოვან მონაცემთა ბაზაზე (ნახ.7.89).



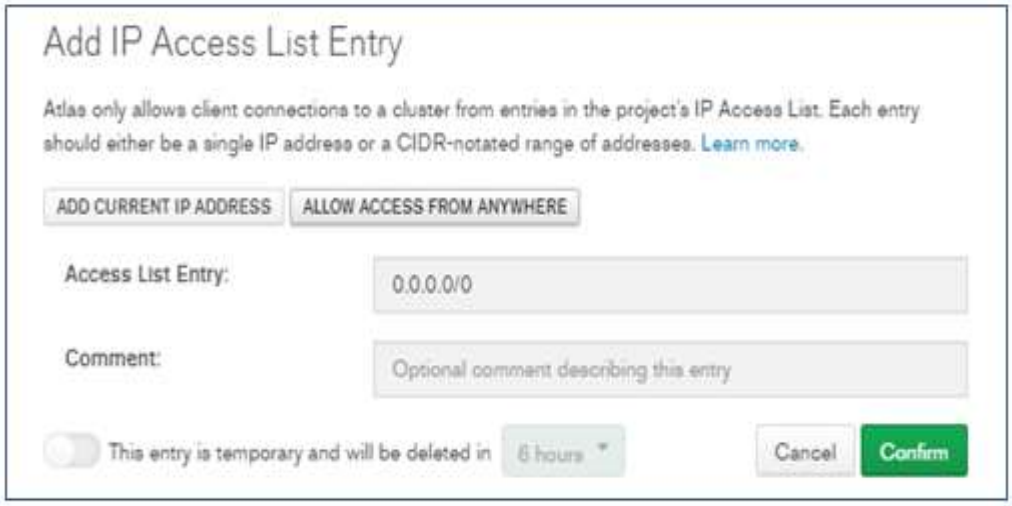
ნახ.7.89

ინტერფეისზე მომდევნო ჩანართი არის **Network Access,** სადაც უნდა განისაზღვროს მომხმარებლის IP მისამართები ჩანართის გააქტიურების შედეგად მიიღება შემდეგი ფანჯარა (ნახ.7.90).



ნახ. 7.90

ღილაკის Add IP Addresse გააქტიურებით, მიიღება ფანჯარა სადაც ემატება ის IP მისამართები, რომელთაც Atlas - ი აძლევს სისტემასთან წვდომის უფლებას (ნახ.7.91).



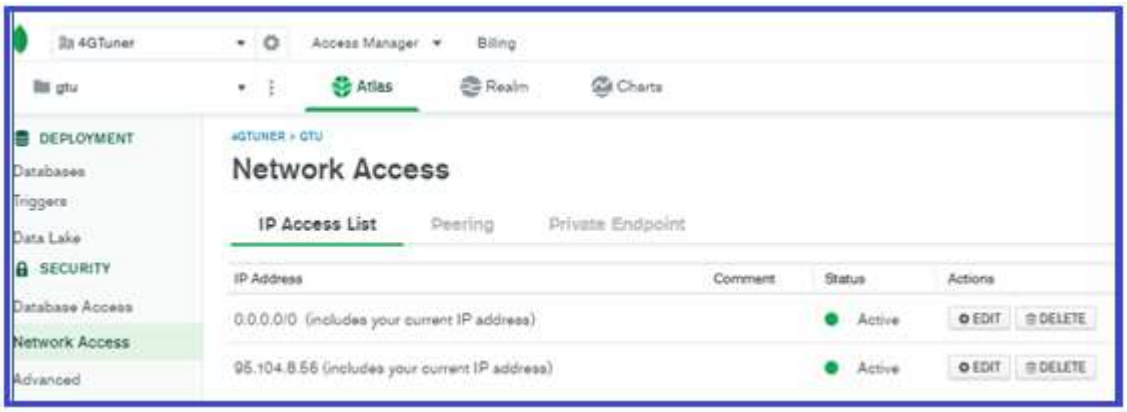
ნახ.7.91

ამისათვის კონკრეტულ მომხმარებელთა მისამართებს ვამატებთ ღილაკით ADD CURRENT IP ADDRESS, ხოლო თუ არ არის ამის აუცილებლობა, მაშინ ვააქტიურებთ ღილაკს ALLOW ACCESS FROM ANYWERE**,** ვადასტურებთ **Confirm** ღილაკით**.**

დავამატეთ ორი IP მისამართი რომელიც არის აქტიური და აქვს წვდომა Atlas პლატფორმაზე, რომლის საილუსტრაციო მაგალითი წარმოდგენილია 7.92 ნახაზზე.

ამგვარად, ჩვენ შევქმენით კლასტერი, დავამატეთ მომხმარებლები და განვსაზღვრეთ IP მისამართები.

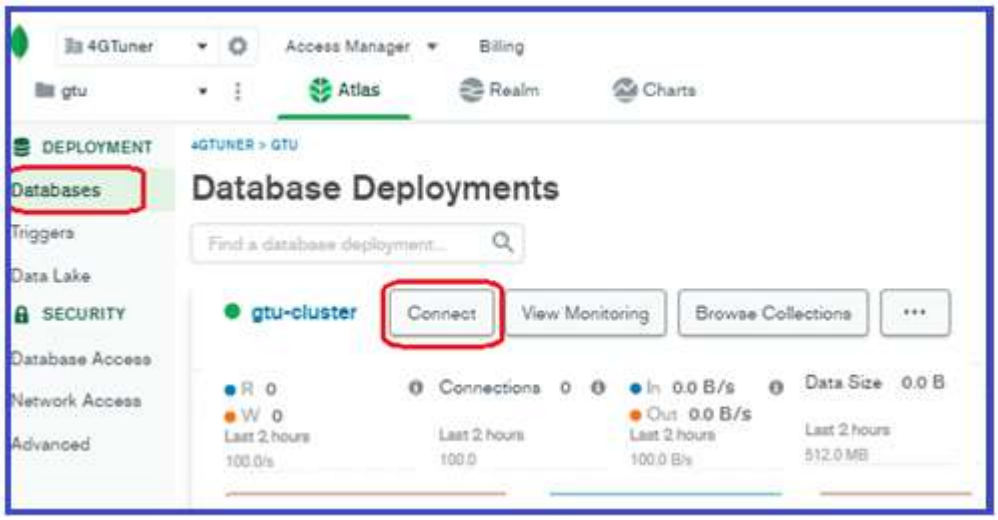
შემდეგი ეტაპი არის MongoDB - ბაზასთან დაკავშირება და მონაცემებთან წვდომა.



ნახ.7.92

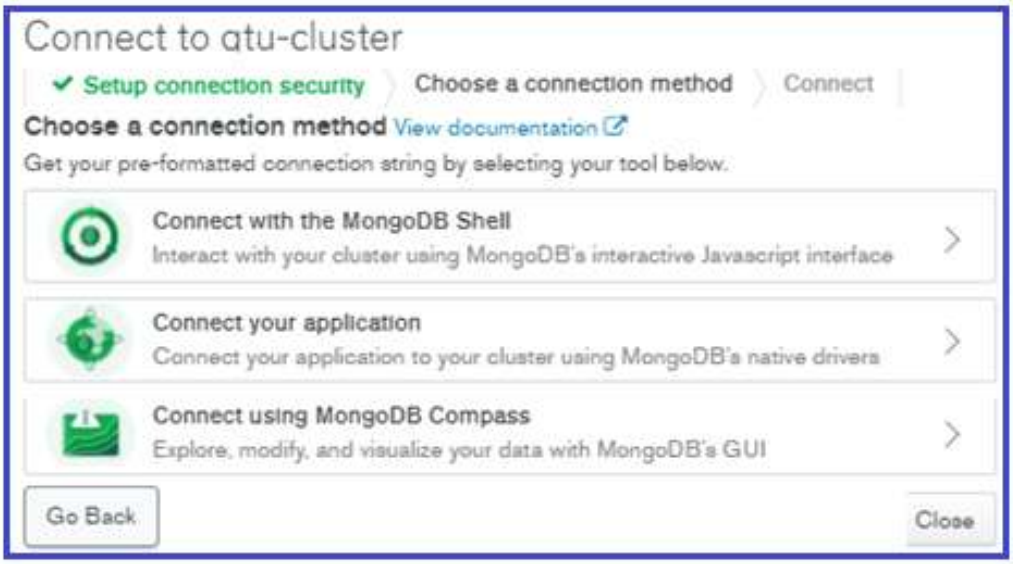
➢ **დაკავშირება მონაცემთა ბაზასთან.**

მონაცემთა ბაზასთან დასაკავშირებლად ვააქტიურებთ, ჩანართს Databases და შემდეგ ვირჩევთ Connect (ნახ.7.93).



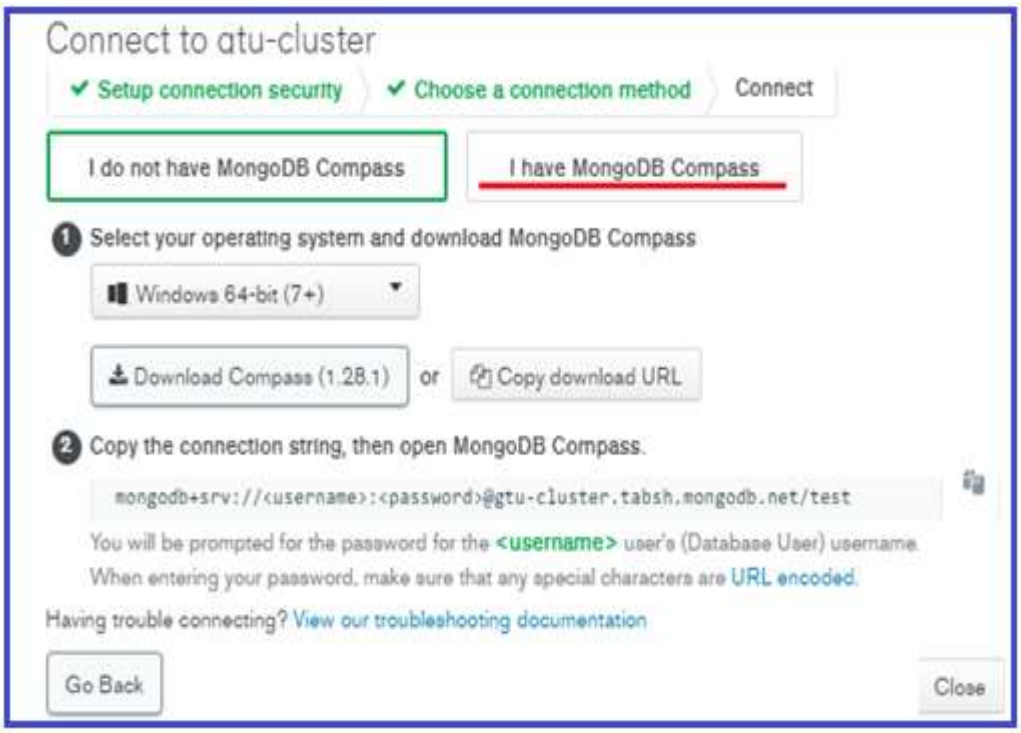
ნახ.7.93

მიიღება ფანჯარა, საიდანაც კლასტერის დაკავშირება შესაძლებელია, როგორც **MongoDB Shell**-ის, ასევე ჩვენს მიერ შექმნილ აპლიკაციასთან და **MongoDB Compass**-გარემოსთან (ნახ.7.94).



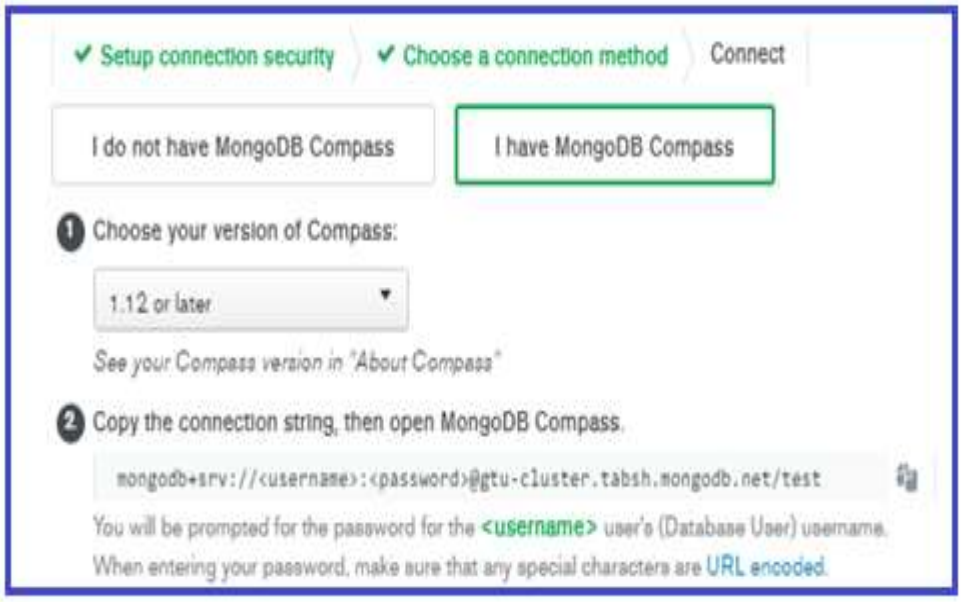
ნახ.7.94

კლასტერი დავაკავშიროთ ჩვენს მიერ შექმნილ MongoDB Compass - თან, ამისათვის, როგორც სისტემაშია მოცემული გავააქტიუროთ ღილაკი - **Connect using MongoDB Compass**, შედეგად მიიღება ფანჯარა, ორი ღილაკით, პირველი, როდესაც მომხმარებელს არა აქვს MongoDB Compass და შეუძლია ჩამოტვირთოს და ლოკალურად დააყენოს თავის პერსონალურ კომპიუტერში და მეორე, როდესაც გვაქვს და შესაძლებელია MongoDB Compass ით სარგებლობა, ჩვენს შემთხვევაში მოვნიშნოთ ღილაკი I have MongoDB Compass



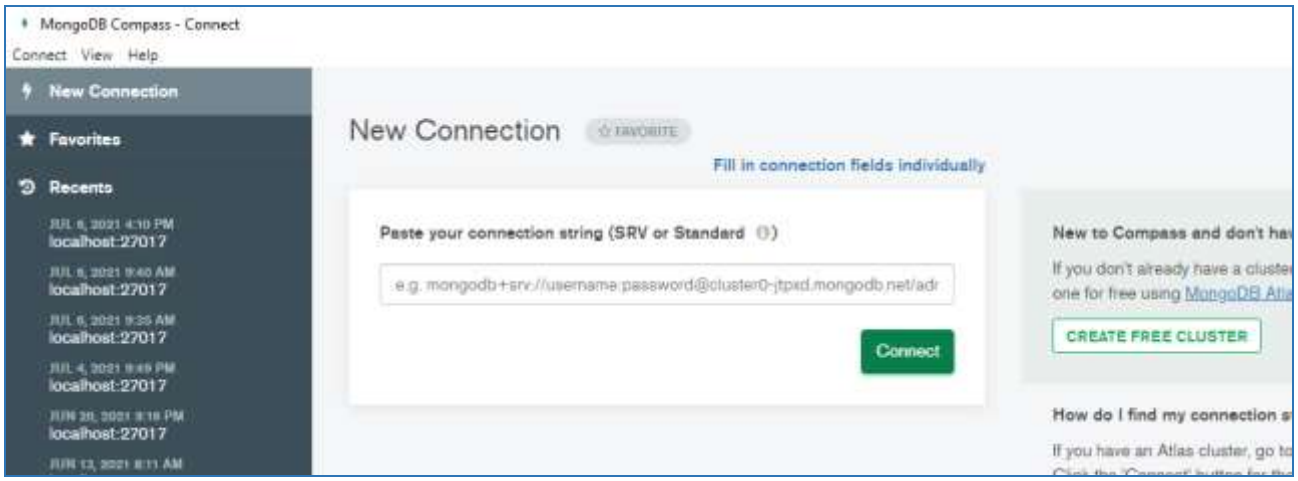
ნახ. 7.95

შედეგად მიიღება ფანჯარა, საიდანაც ვაკოპირებთ სტრიქონს MongoDB Compass ისთვის (ნახ.7.96).



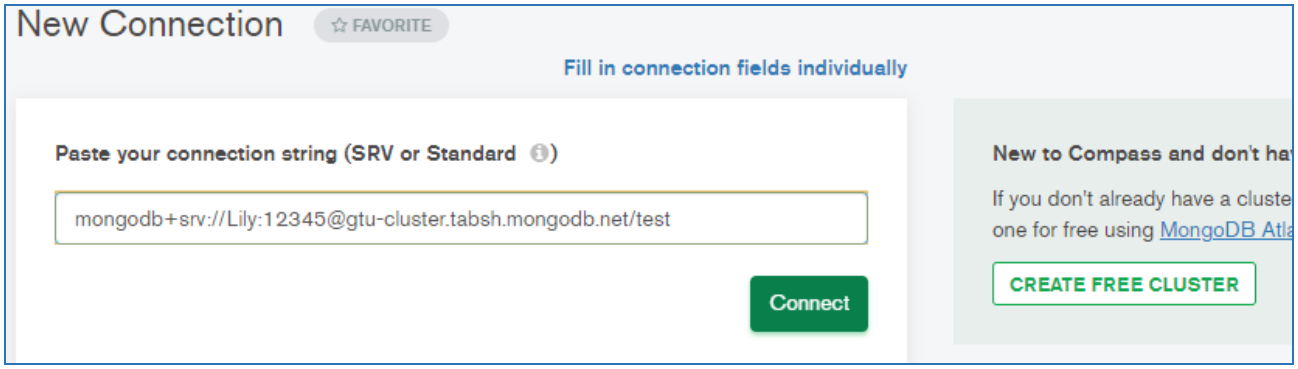
ნახ. 7.96

პარალელურად ლოკალურად ვხსნით ჩვენს MongoDB Compass გარემოს (ნახ.7.97).



ნახ.7.97

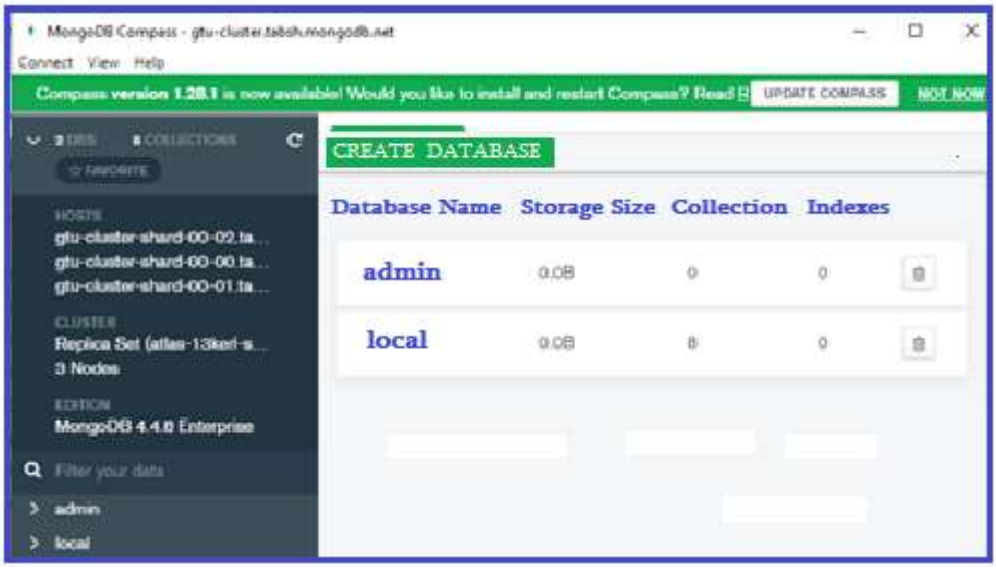
დაკოპორებულ სტრიქონს ჩავსვამთ სპეციალურად გამოყოფილ არეში (ნახ.7.98).



ნახ.7.98

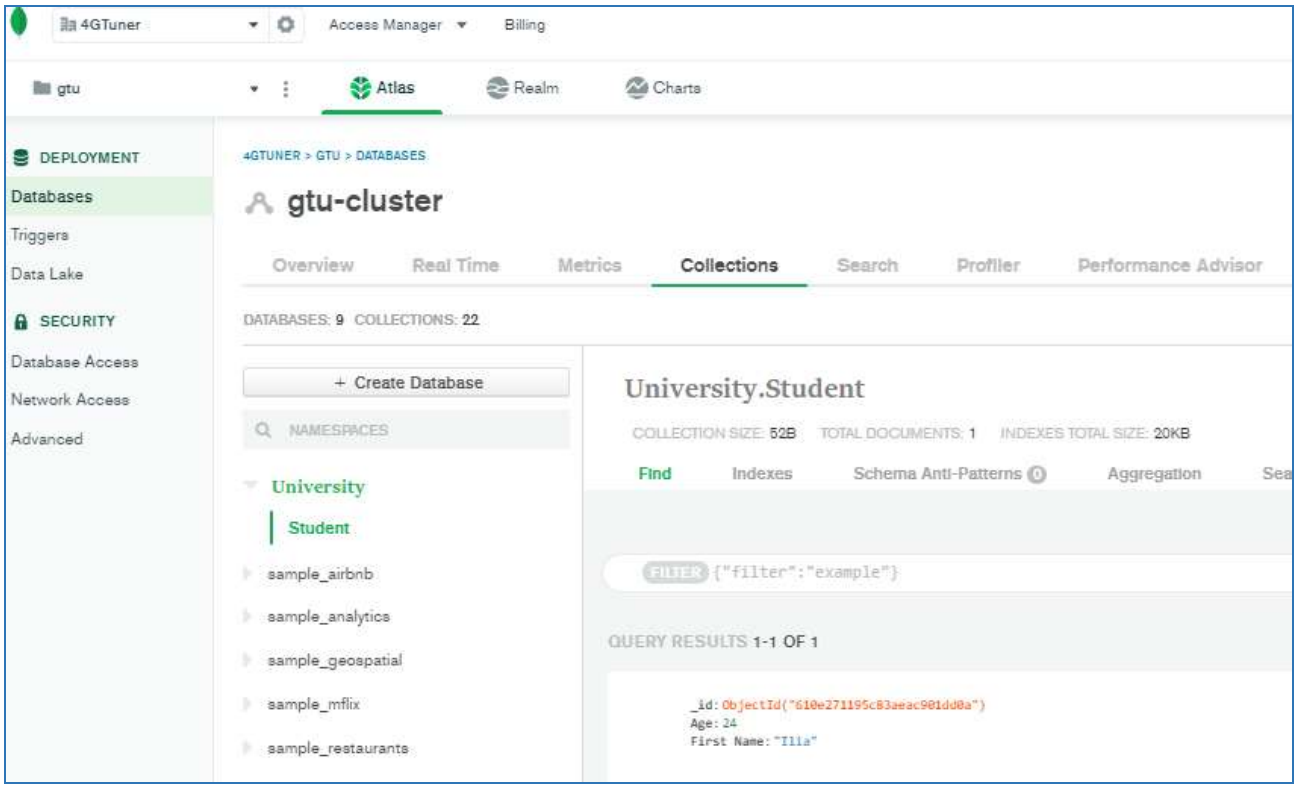
დავაკორექტირებთ მომხმარებლის სახელს და ჩავწერთ შესაბამის პაროლს, რასაც ვადასტურებთ Connect ღილაკით.

შედეგად მოხდება დაკავშირება, მიიღება ფანჯარა, სადაც ვამატებთ ახალ ბაზას. ვქმნით კოლექციებს (ნახ.7.99).



ნახ.7.99

MongoDB Atlas გარემოში განახლების ღილაკის გააქტიურების შემდეგ აისახება ჩვენს მიერ შექმნილი, მაგალითად, University ბაზა, რომლის კოლექციის დასახელებაა Student (ნახ.7.100).



ნახ.7.100

მიღებული შედეგის თანახმად ლოკალური Compass გარემოდან ღრუბლოვან მონაცემთა ბაზაში ავსახეთ ჩვენს მიერ ფორმირებული მონაცემთა ბაზა, რომლის ანალიზის და დამუშავებისთვის შესაძლებელია გამოყენებული იქნას ყველა თანამედროვე ტექნოლოგია და ინსტრუმენტული საშუალება.

7.5. Web-გვერდების აპლიკაციის შექმნა WPF ტექნოლოგიით

მაიკროსოფთის Visual Studio.NET პლატფორმის ტექნოლოგია Windows Presentation Foundation (WPF) გამოიყენება დესკტოპ- და ვებ-აპლიკაციების შესაქმნელად. იგი ჰიბრიდული ტექნოლოგიაა მომხმარებელთა ინტერფეისების ვიზუალური მართვის ელემენტებით [1, 22].

განვიხილოთ ერთი ამოცანა, კერძოდ, ვებ-გვერდების აპლიკაციის აგება ნავიგაციის ელემენტებით.

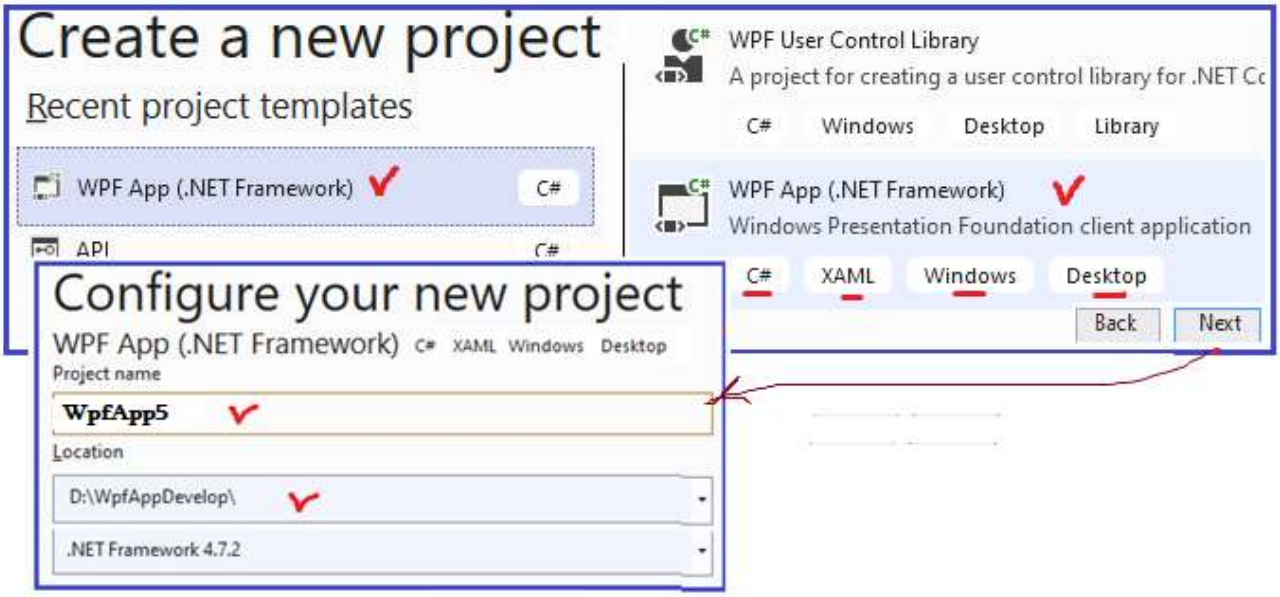
საჭიროა C# და XAML (Extended Application Markup Language). ვისაც HTML ან XML სკრიპტულ ენაზე უმუშავია, მას არ გაუჭირდება ასეთი ამოცანის გარჩევა. საფუძვლიანად კი ამ ტექნოლოგიას მე-7 სემესტრში შევისწავლით.

ამგვარად, WPF-ტექნოლოგიას აქვს ვებ-გვერდების შექმნის საშუალებები. ასეთი გვერდების გამოყენება ხშირად მომხმარებლისთვის უფრო მოსახერხებელია, ვიდრე ფანჯრული (Windows Forms App) ორგანიზაცია. შედეგები აისახება რომელიმე ბრაუზერში.

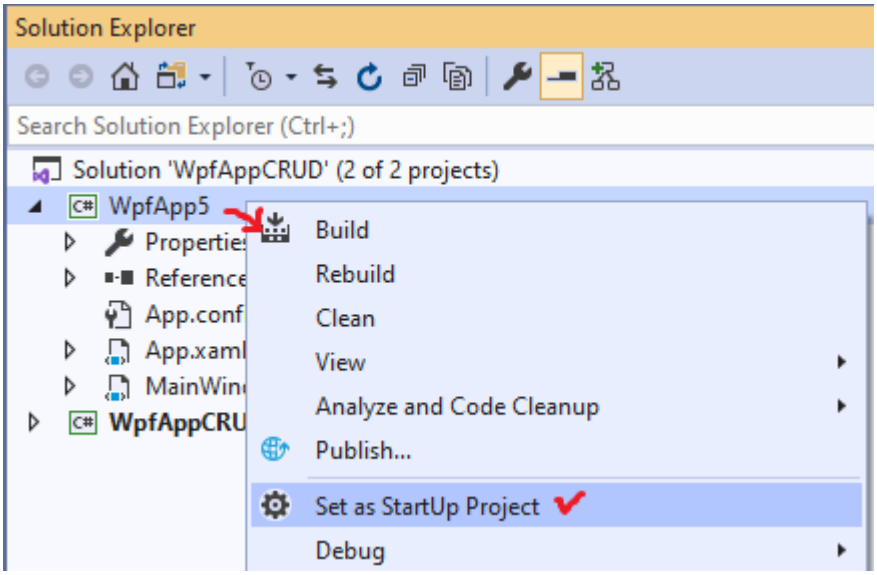
გვერდული აპლიკაციის შიგთავსი ჩაშენდება სპეციალურ ნავიგაციურ კარკასში, რომელსაც აქვს ნავიგაციური კავშირები და ნავიგაციური ჟურნალი.

მისი ძირითადი (ფესვური) კლასია NavigationWindow, რომელიც ამატებს აპლიკაციისთვის *ნავიგაციის სტანდარტულ ინტერფეისს* და მისთვის საჭირო ინფრასტრუქტურას. NavigationWindow კლასი წარმოებულია Window-კლასიდან და აქვს წვდომა დანართის იმავე საშუალებებზე.

1. შევქმნათ ახალი პროექტი WpfApp5 სახელით (ნახ.7.101) და (ან დავამატოთ Solution Exp[lorer-ში ახალი პროექტი და გავხადოთ „სასტარტო“ (ნახ.7.102).



ნახ. 7.101. ახალი პროპექტის შექმნა



ნახ. 7.102. პროექტის დამატება Solution Explorer-ში

2. წავშალოთ ავტომატურად შექმნილიMainWindow.xaml ფაილი SolutionExplorer-ში და ჩავამატოთ მის ნაცვლად Window(WPF)-შაბლონით ახალი ფაილი NavExample.xaml სახელით.

3. გავხსნათ App.xaml ფაილი და შევცვალოთ მასში ატრიბუტი:

StartupUri=“NavExample.xaml“

4. ავამუშავოთ პროექტი WpfApp5. დავრწმუნდეთ, რომ არაა შეცდომები.

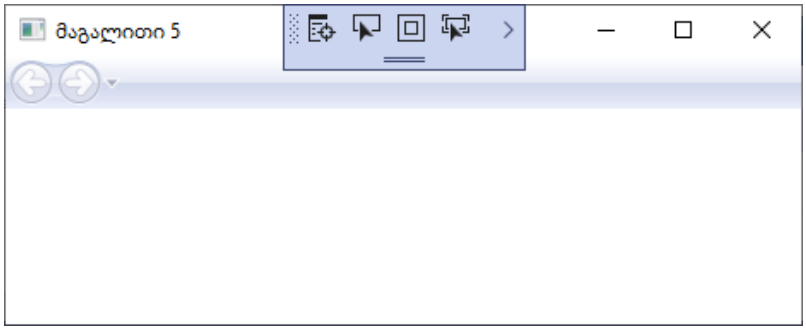
5. გავხსნათ ფაილი NavExample.xaml და შევასწოროთ მასში დესკრიპტორული კოდი:

```
<NavigationWindow x:Class="WpfApp5.NavExample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="მაგალითი 5"
    Height="300"
    Width="300"
    WindowStartupLocation="CenterScreen"
        >
</NavigationWindow>
```

6. გავხსნათ NavExample.xaml.cs ფაილი და შევასწოროთ C# კოდის ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

using System.Windows.Navigation;

namespace WpfApp5
{
    public partial class NavExample : NavigationWindow
    {
        public NavExample()
        {
            InitializeComponent();

            //this.Navigate(new Page1());
        }
    }
}
```

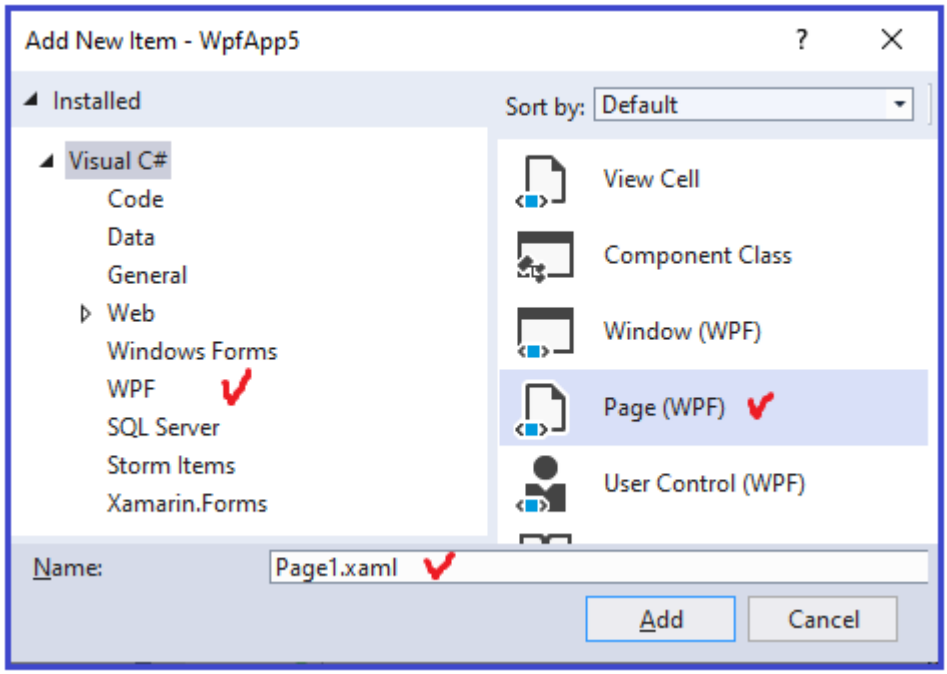
ავამუშავოთ პროექტი, შედეგი იქნება ცარიელი გვერდი (ნახ.7.103) ნავიგაციური ელემენტით.



ნახ.7.103

7. ნავიგაციური კვანძის შიგთავსი წარმოდგენილი უნდა იყოს კლასით, რომელიც წარმოებულია ბიბლიოთეკის Page-კლასისგან. შევქმნათ სამი გვერდი და მივაბათ ნავიგაციის კვანძს Navigate() მეთოდის დახმარებით.

- დავამატოთ პროექტს ახალი Page ელემენტი Page1.xaml სახელით (ნახ.7.104).



ნახ.7.104. ახალი გვერდის დამატება

8. შევასწოროთ Page1.xaml ფაილის ტექსტი:

```
<Page x:Class=“WpfApp5.Page1“
xmlns=“http://schemas.microsoft.com/winfx/2006/xaml/presentation“
   xmlns:x=“http://schemas.microsoft.com/winfx/2006/xaml“
      >
   <StackPanel>
      <TextBlock TextAlignment=”Center” FontSize=”24”>გვერდი 1</TextBlock>
      <TextBlock></TextBlock>
        <TextBlock>
     <Hyperlink Click=”LinkClicked”>მე-2 გვერდზე</Hyperlink>
      </TextBlock>
   </StackPanel>
</Page>
```

9. შევასწოროთ Page1.xaml.cs ფაილის კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
```

```
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }

    private void LinkClicked(object sender, RoutedEventArgs e)
        {
            this.NavigationService.Navigate(page2);
        }
    }
}
```

10. დავამატოთ პროექტს ახალი Page ელემენტი Page2.xaml სახელით. შევასწოროთ xaml-კოდი:

```
<Page x:Class=”WpfApp5.Page2”
xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”
    xmlns:x=”http://schemas.microsoft.com/winfx/2006/xaml”
    Title=”Page2”
        >
    <StackPanel>
        <TextBlock TextAlignment=”Center” FontSize=”24”>გვერდი 2</TextBlock>
    </StackPanel>
</Page>
```

11. NavExample.xaml ფაილში დავამატოთ Source-ატრიბუტი, რომელიც მიუერთდება კარკასის გაშვებისას Page1.xaml საწყისი გვერდის შიგთავსს.

```
<NavigationWindow x:Class=“WpfApp5.NavExample“
    . . .
    WindowStartupLocation=”CenterScreen”
    Source=”Page1.xaml”
        >
</NavigationWindow>
```

ავამუშავოთ პროექტი და შევამოწმოთ ღილაკების მუშაობისუნარიანობა.

დასკვნა:

ამგვარად, ჩვენ შევქმენით კარკასი და ორი ცარიელი გვერდი, რომლებიც არაფერს არ აკეთებს. თითოეული გვერდი ავტონომიურია, შეიძლება მათი შევსება ტულბოქსის ელემენტებით.

გვერდებს შორის გადასვლისას საჭიროა ვიცოდეთ ინფორმაციის გადაცემა ერთი გვერდიდან მეორეში. უნდა არსებობდეს საერთო საფოსტო ყუთი, რომელიც არ იქნება დამოკიდებული გვერდებზე.

WPF-ში მონაცემების გადასაცემად გვერდებს შორის იყენებენ ლექსიკონს (წყვილების მასივი: „გასაღები-მნიშვნელობა“) Application.Current.Properties, ან ინფორმაციის „ჩაკერვას“ უშუალოდ ახალი გვერდის ობიექტში.

12. Page1-ზე დავამატოთ სახელმინიჭებული ტექსტური ველი შემდეგი სახით:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Page1"
      >
   <StackPanel>
      <TextBlock TextAlignment="Center"
                        FontSize="24">გვერდი 1</TextBlock>
      <TextBlock></TextBlock>
      <Label>შეიტანეთ თქვენი სახელი: </Label>
      <TextBox Name="nameBox" Width="200"></TextBox>
      <TextBlock></TextBlock>
      <TextBlock>
    <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
      </TextBlock>
   </StackPanel>
</Page>
```

TextBox-ობიექტს მივანიჭეთ სახელი, რათა შეიძლებოდეს მასზე მიმართვა კოდიდან.

13. შევცვალოთ Page1 გვერდის კოდი შემდეგი სახით;

```
using System;
. . .
namespace WpfApp5
{
  public partial class Page1 : Page
   {
    public Page1()
      {
        InitializeComponent();
      }
 private void LinkClicked(object sender, RoutedEventArgs e)
      {
        Page2 page2 = new Page2();
        page2.Message = nameBox.Text + " !!!"; // ინფორმაციის
                                                    //„ჩაკერვა“ ობიექტში
        this.NavigationService.Navigate(page2);
      }
   }
}
```

14. დავამატოთ Page2 გვერდზე სახელმინიჭებული ტექსტური ჭდე და ჰიპერლინკი Page3-ზე გადასასვლელად. აგრეთვე მომამზადეთ გვერდის მოვლენა Loaded და მოვლენა Click ჰიპერლინკისთვის.

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Page2"
   Loaded="Page_Loaded"
      >
```

```
  <StackPanel>
    <TextBlock TextAlignment=”Center”
                    FontSize=”24”>გვერდი 2</TextBlock>
    <TextBlock></TextBlock>
    <TextBlock>მოგესალმებით </TextBlock>
    <Label Name=”nameLabel”></Label>
    <TextBlock Margin=”0,10”>  <!—Отступ сверху→
<Hyperlink Click=”LinkClicked”>მე-3 გვერდზე</Hyperlink>
    </TextBlock>
  </StackPanel>
</Page>
```

Label ობიექტს მივანიჭეთ სახელი, რათა კოდიდან შეიძლებოდეს მასზე მიმართვა.

15. დავამატოთ Page2-ის კოდს public თვისება, ტექსტურ ჭდეზე გადაცემული ტექსტის მისანიჭებელი კოდი Loaded-მოვლენაში და შემდეგ გვერდზე გადასვლის კოდი.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
    public partial class Page2 : Page
    {
        public Page2()
        {
            InitializeComponent();
        }

        string message;
        public string Message
        {
            set { message = value; }
        }
   private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            nameLabel.Content = message;
        }
   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            Page3 page3 = new Page3();
            this.NavigationService.Navigate(page3);
        }
```

```
    }
}
```

16. ამოუშავეთ აპლიკაცია. ინფორმაცია გადაეცემა, ოღონდ ღილაკის ამუშავებით.

(!) ნავიგაციის ღილაკით ახალი ინფორმაცია ტექსტური ველიდან არ გადაიცემა. ინფორმაცია აიღება ისტორიის ჟურნალიდან.

17. Page3 გვერდისთვის შეავსეთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Page3">
   <StackPanel>
      <!--გვერდის კონტექსტის სათაური →
      <TextBlock TextAlignment="Center"
                                FontSize="24">გვერდი 3
         <TextBlock.Margin>0,0,0,10</TextBlock.Margin>
      </TextBlock>
    <!--პირველი ღილაკი→
      <Button Content="Push Me!" FontSize="22" Width="175"
                                          Height="50" Click="Button_Click">
         <Button.Effect>
            <DropShadowEffect />
         </Button.Effect>
      </Button>
    <!--მეორე ღილაკი→
      <Button FontSize="22" Height="50" Width="175"
                                        Margin="0,10" Click="Button_Click">
         "დამკლიკე"
         <Button.Effect>
            <DropShadowEffect />
         </Button.Effect>
         <Button.Foreground>
      <LinearGradientBrush StartPoint="1,0" EndPoint="0,0">
            <GradientStop Color="Red" Offset="0" />
            <GradientStop Color="Orange" Offset=".17" />
            <GradientStop Color="Yellow" Offset=".33" />
            <GradientStop Color="Green" Offset=".5" />
            <GradientStop Color="CornflowerBlue" Offset=".67" />
           <GradientStop Color="Blue" Offset=".84" />
          <GradientStop Color="BlueViolet" Offset="1" />
          </LinearGradientBrush>
         </Button.Foreground>
         <Button.Background>
      <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Color="Red" Offset="0" />
           <GradientStop Color="Orange" Offset=".17" />
```

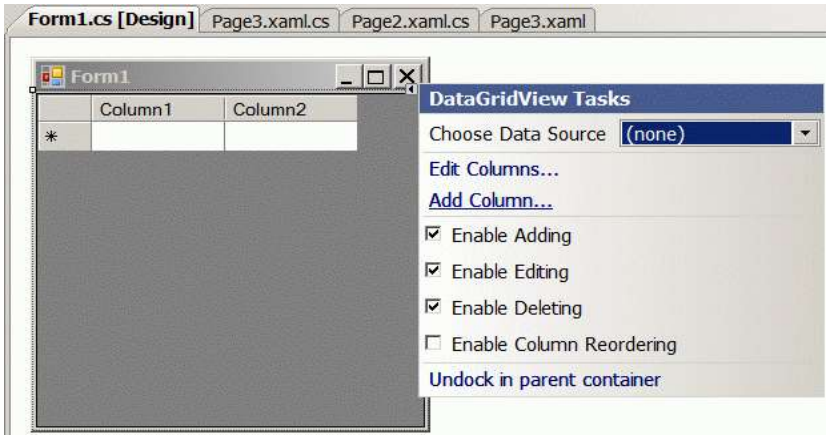
```
            <GradientStop Color="Yellow" Offset=".33" />
            <GradientStop Color="Green" Offset=".5" />
            <GradientStop Color="CornflowerBlue" Offset=".67" />
            <GradientStop Color="Blue" Offset=".84" />
          <GradientStop Color="BlueViolet" Offset="1" />
          </LinearGradientBrush>
        </Button.Background>
      </Button>
    </StackPanel>
</Page>
```

Click - მოვლენის დამმუშავებელი უნდა ჩაიწეროს ხელით, რათა ის შეიქმნას კოდის ნაწილში.

18. სანამ შევავსებთ Page3 გვერდის კოდის ნაწილს, საჭიროა პროექტს დაემატოს ახალი ფორმა. ამით შესაძლებელია WPF და Windows Forms ტექნოლოგიების ერთობლივად მუშაობის დემონსტრირება. ღილაკებზე დავდოთ ფორმის ამუშავების კოდი.

19. დავამატოთ WpfApp5-ში ახალი ფორმა Form1.cs

20. Form1 ფორმაზე ტულბოქსიდან დავდოთ DataGridView ელემენტი. დავაყენოთ მისი თვისება Dock=Fill და დავამატოთ ინტელექტუალური დესკრიპტორიდან (SmartTag) ორი სვეტი



ნახ.10

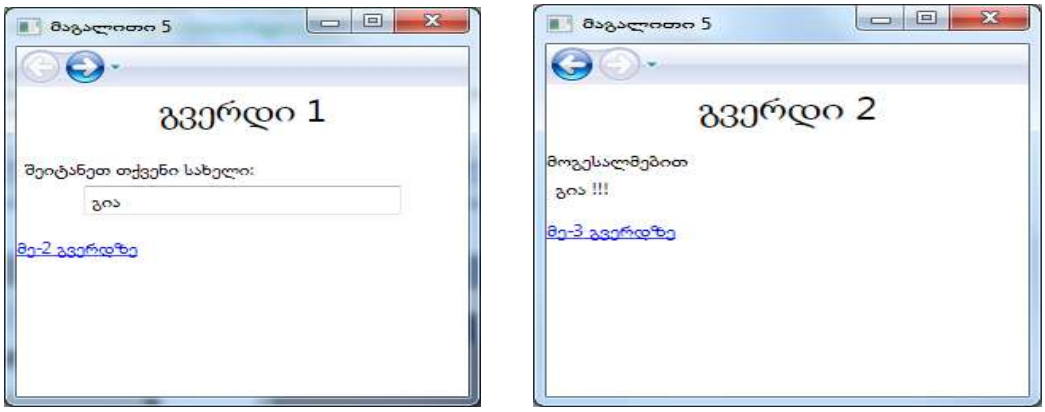
21. ღილაკების საერთო დამმუშავებელი Page3 კოდის ნაწილში შევავსოთ ასე:

```
//--- ლისტინგი -------
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
```

```
    public partial class Page3 : Page
    {
        public Page3()
        {
            InitializeComponent();
        }
private void Button_Click(object sender, RoutedEventArgs e)
        {
            Form1 frm = new Form1();
            frm.ShowInTaskbar = false; // ფორმის ღილაკი არ
                                       //გამოჩნდეს ამოცანების პანელზე
            frm.Show();
        }
    }
}
```

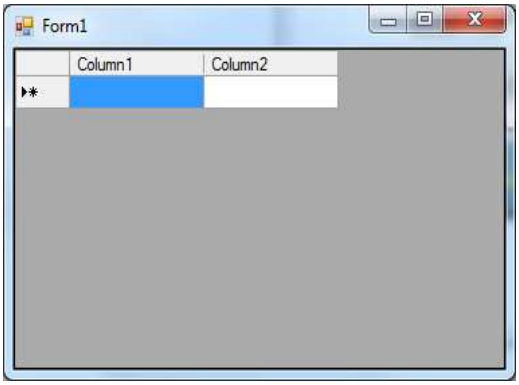
22. ავამუშავოთ პროექტი. დავაკვირდეთ ახალი გვერდის დიზაინს. იხსნება Form1 -იც, რაც ადასტურებს WPF და Windows Form ტექნოლოგიების ერთობლივი მუშაობის შესაძლებლობას.

შედეგები (ნახ.11–ა,ბ,გ):



ნახ.11–ა

ნახ.11–ბ



ნახ.11–გ

გამოყენებული ლიტერატურა:

1. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. მონოგრ., ISBN 978-9941-20-790-7. სტუ, „ტექნიკ.უნივ.“, თბ., 2017. -1001 გვ., ბიბლ.ინდ. 004.42/7

2. ჩოგოვაძე გ., სურგულაძე გ., გულიტაშვილი მ., დოლიძე ს. პროგრამული აპლიკაციების ხარისხის მართვა: ტესტირება და ოპტიმიზაცია. მონოგრაფია. ISBN 978-9941-8-0629-2. სტუ. „ITკონსალტინგ ცენტრი". თბ., 2020. -365 გვ. https://gtu.ge/book/Surgu_SoftwareQuality.pdf

3. სურგულაძე გ., კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი (WCF ტექნოლოგია). ISBN 978-9941-0-7878-1. სტუ. „IT-კონსალტინგ ცენტრი“. თბ.,2015. -154 გვ.

4. სურგულაძე გ., კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი (WPF ტექნოლოგია). ISBN 978-9941-0-7103-4, 978-9941-0-7104-1. სტუ. „IT-კონსალტ. ცენტრი“. თბ.,2014.-202 გვ.

5. სურგულაძე გ., კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი (WF ტექნოლოგია). ISBN 978-9941-0-7520-9. სტუ. „IT-კონსალტინგ ცენტრი“. თბ.,2015. -136 გვ.

6. სურგულაძე გ., პეტრიაშვილი ლ. ვიზუალური დაპროგრამება C# ენის ბაზაზე ინფორმაციულ სისტემებისათვის (Visual StudioNET 2019 პლატფორმაზე). ISBN 978-9941-8-1708-3. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2019. -200 გვ. https://gtu.ge/book/GiaSurg_Csh_IS_2019.pdf

7. სურგულაძე გ., თურქია ე.პროგრამული სისტემების მენეჯმენტის საფუძვლები. ISBN 978-9941-20-651-1. სტუ. „ტექნიკ.უნივ.“, თბ., 2016. -350 გვ.

8. გაჩეჩილაძე ლ. დაპროგრამების ენა Python. სტუ, გამომც. „ტექნიკური უნივერსიტეტი". თბ., 2018. 157 გვ.

9. სურგულაძე გ., ვიზუალური დაპროგრამება C#_2010. ISBN978-9941-20-021-2. სტუ. „ტექნიკ.უნივერს.“, თბ., 2011 -445 გვ.

10. სურგულაძე გ., თოფურია ნ., გავარდაშვილი ა. შავი ზღვის ეკოლოგიური მონიტორინგისა და კვლევის საინფორმაციო სისტემა. ISBN 978-9941-8-0624-7. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2018. -206 გვ. https://gtu.ge/book/Surgul_EcoBlackSea.pdf

11. სურგულაძე გ., გულიტაშვილი მ., კივილაძე ნ. Web-აპლიკაციების ტესტირება, ვალიდაცია და ვერიფიკაცია. ISBN 978-9941-0-7682-4. სტუ. „IT-კონსალტ.ცენტრი“. თბ., 2015. -205 გვ.

12. Set up unit testing for Python code 2022, Visual Studio, Microsofr.com. Internet resource: https://learn.microsoft.com/en-us/visualstudio/python/?view=vs-2022.

13. პეტრიაშვილი ლ., სურგულაძე გ. მონაცემთა მენეჯმენტის თანამედროვე ტექნოლოგიები (Oracle, MySQL, MongoDB, Hadoop). ISBN 978-9941-27-176-2. სტუ. „ITC-ცენტრი”, თბ., 2017. 202 გვ.

14. სურგულაძე გ., კივილაძე გ. (2017). შესავალი NoSQL მონაცემთა ბაზებში. ISBN 978-9941-0-9642-6. სტუ. „ტექნიკ.უნივ.“. თბ., – 152 გვ.

15. Install MongoDB Compass- MongoDB Management Tool. https://www.guru99.com/installation-configuration-mongodb.html#1. (15.12.21)

16. Download and Install Compass. Internet resource: https://www.mongodb.com/docs/compass/current/install/ (15.12.22)

17. სურგულაძე გ., გულუა დ., კახელი ბ. პროგრამული აპლიკაციების აგება ვირტუალიზაციის პირობებში. ISBN 978-9941-8-0627-4. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2019. -159 გვ. ბიბილ. ინდექსი 004.92/8. . https://gtu.ge/book/SurGuluaKakheli_Virtualization.pdf

18. სურგულაძე გ., გულუა დ. ქსელური არქიტექტურები ბიზნესისათვის. ISBN 978-9941-0-9842-0. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2017. -269 გვ. https://gtu.ge/book/gia_sueguladze/SurGul_Serv-TechBusiness.pdf

19. About AWS . https://aws.amazon.com/about-aws/

20. Gao J. Windows Azure and SQL Database (formerly SQL Azure) tutorials. Step-by-step. Copyright © 2012 by Microsoft Corporation.

21. Google Cloud to Azure services comparison. Microsoft. 2022. https://learn.microsoft.com/en-us/azure/architecture/gcp-professional/services

22. სურგულაძე გ., კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი (WPF ტექნოლოგია). ISBN 978-9941-0-7103-4, 978-9941-0-7104-1. სტუ. „IT-კონსალტინგ ცენტრი“. თბ.,2014. -202 გვ.

გადაეცა წარმოებას 20..01.2022. ხელმოწერილია დასაბეჭდად 01.02.2022. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 6. ტირაჟი 50 ეგზ.

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი“,
თბილისი, მ.კოსტავას 77