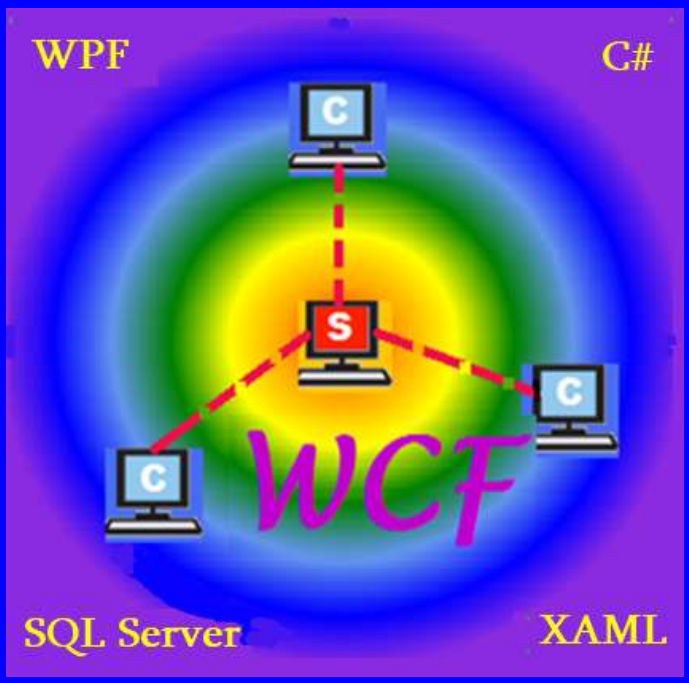
მაგისტრანტის ბიბლიოთეკა

გია სურგულაძე, ლილი პეტრიაშვილი

კორპორაციული მართვის სისტემების პროგრამული დეველოპმენტი (WCF/WPF, SOA)

(საკურსო პროექტის მეთოდური მითითებანი)



„სტუ-ს IT- კონსალტინგის სამეცნიერო ცენტრი”

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ლილი პეტრიაშვილი

კორპორაციული მართვის სისტემების პროგრამული დეველოპმენტი (WCF/WPF, SOA)

(საკურსო პროექტის მეთოდური მითითებანი)

დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ ოქმი N2, 10.02.2021

თბილისი
2021

უაკ 004.5

წარმოდგენილია საქართველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის სამაგისტრო პროგრამის „ინფორმატიკა“ აკადემიური დისციპლინის „კორპორაციული მართვის სისტემების პროგრამული დეველოპმენტი“ სილაბუსის საფუძველზე პრაქტიკული საკითხების და საკურსო პროექტის მომზადების მეთოდური მითითებები (კონცენტრაცია: „მართვის საინფორმაციო სისტემები“ – Management Information Systems). განხილულია საინფორმაციო სისტემების პროგრამული დეველოპმენტის ამოცანები და მათი გადაწყვეტა მაიკროსოფთის Visual Studio .NET Framework პლატფორმის WCF/WPF ტექნოლოგიებით. კერძოდ, საკურსო პროექტის შესრულება ხორციელდება ინდივიდუალურად ან გუნდური პრინციპით. პროგრამული პროდუქტი რეალიზებულია კლიენტ-სერვერული ან სერვის ორიენტირებული არქიტექტურით. განკუთვნილია სტუ-ის „ინფორმატიკის“ სამაგისტრო პროგრამის სტუდენტებისათვის,

რეცენზენტი:

პროფ. ე. თურქია (საქართველოს ეროვნული ბანკის განყოფილების გამგე, ტ.მ.კ.)

პროფ. გ. ნარეშელაშვილი (სტუ)

ISBN 978-9941-8-2725-9

სარჩევი

1. სასწავლო კურსის მიზანი

„კორპორაციული მართვის სისტემების პროგრამული დეველოპმენტი“-ს აკადემიური კურსი შეისწავლის ორგანიზაციული მართვის (მენეჯმენტის) საინფორმაციო სისტემების პროგრამირების ჰიბრიდულ WCF და WPF ტექნოლოგიებს, კლიენტ-სერვერული და ვებ-სერვისებზე ორიენტირებული არქიტექტურებით; სამაგიდო (Desktop) აპლიკაციებითვის მომხმარებელთა ინტერფეისების დიზაინს, პროგრამულ აპლიკაციათა დეველოპმენტს და მათ ტესტირებას; განაწილებული ინფორმაციული ბაზების გამოყენებას და მონაცემთა მენეჯმენტს

2. საგნის შესწავლის შედეგად მიღებული ცოდნა და შეძენილი უნარები

საგნის შესწავლის შედეგად სტუდენტი ღებულობს ცოდნას და იძენს შემდეგ უნარებს:

1. ავლენს კორპორაციული მართვის საინფორმაციო სისტემების ობიექტ-ორიენტირებული მოდელირების და დაპროგრამების თეორიულ ცოდნას და პრაქტიკულ უნარს;
2. იყენებს ორგანიზაციული მართვის სისტემების ასაგებად ბიზნესის, კომერციის, ეკონომიკის, პროდუქციის წარმოების, ჯანმრთელობის, განათლების და სხვა სფეროებში ცოდნის შემოქმედებითი გამოყენების უნარებს;
3. აყალიბებს ორგანიზაციული მართვის (მენეჯმენტის) პროცესების ავტომატიზაციის მნიშვნელობას საინფორმაციო სისტემების ასაგებად.
4. დამოუკიდებლად (ან გუნდში მონაწილეობით) არჩევს პროგრამულ აპლიკაციებს .NET პლატფორმაზე კორპორაციული მენეჯმენტის ამოცანების გადასაწყვეტად;
5. ასაბუთებს დაპროექტების, პროგრამული დეველოპმენტის, ტესტირების და ფუნქციური ამოცანების შესრულების დანერგვის ეტაპებს;
6. აყალიბებს Desktop- და Web- აპლიკაციების საფუძველზე მართვის საინფორმაციო სისტემების აგების პრობლემების გადაჭრის გზებს.

3. საათების განაწილება (სტუდენტის დატვირთვა)

„კორპორაციული მართვის სისტემების პროგრამული დეველოპმენტი“-ს კურსი 5 კრედიტიანია (1 კრ = 25 სთ). სწავლის ფორმებია: ლექცია (15 სთ/სემესტრში), ლაბორატორიული (15 სთ), საკურსო პროექტი (15 სთ) და დამოუკიდებელი მუშაობა (77 სთ). აგრეთვე 1-შუასემესტრული (1 სთ) და დასკვნითი გამოცდა (2 სთ).

4. ლაბორატორიული მეცადინეობების თემების დასახელება და შინაარსი

„კორპორაციული მართვის სისტემების პროგრამული დეველოპმენტი“-ს“ კურსის ლაბორატორიული მეცადინეობების მასალა მოიცავს შემდეგ თემებს და საკითხებს:

1. Visual Studio .NET (2019) Framework 4.7 პლატფორმა: გარემოს გაცნობა და საილუსტრაციო მაგალითების გარჩევა. პირველი საილუსტრაციო პროგრამული მოდულის შექმნა C# და XAML ენების საფუძველზე;
2. WPF ტექნოლოგია: ბიზნესპროცესების დაპროგრამების ვიზუალიზაცია და კომპლიაცია. დაპროგრამების ტექნოლოგიის ქმედებათა (Activities) ფუნქციონალობის შესწავლა;
3. ბიზნესპროცესების აპლიკაცია: ბიზნესპროცესების მართვის საილუსტრაციო პროგრამული აპლიკაციის აგება WPF ტექნოლოგიით;

4. ბიზნესპროცესების პროცედურული ელემენტები: დიზაინერის მართვა და XAML კოდი, ბიზნესპროცესის დიაგრამა;

5. ბიზნესპროცესების დაპროექტება: რთული ბიზნესპროცესის არგუმენტები, გამოსახულებათა ქმედებები;

6. ბიზნესპროცესების დაპროექტება: გამონაკლისების დამუშავება;

7. საპრობლემო სფეროს დაპროექტება: WPF ტექნოლოგიით .NET პლატფორმაზე;

8. XAML ენის სტრუქტურა და საბაზო ფუნქციები: HTML, XML და XAML ენების შედარების საილუსტრაციო პროგრამების განხილვა;

9. WCF ტექნოლოგია: კომუნიკაციის რეალიზება აპლიკაციებს შორის.ინფორმაციის „გადაცემის“ და „მიღების“ ქმედებები. Send და Receive ქმედებების დაპროგრამება.

10. WCF ტექნოლოგია:მომხმარებლის ქმედების დაპროგრამება: პასუხის მიღების ქმედება. სერვერი – მოთხოვნების დამუშავება. მოთხოვნის მიღების ქმედება. მომხმარებლის ქმედება – პასუხის შექმნა. SendReply ქმედება;

11. Host-აპლიკაციის რეალიზაცია: ServiceHost კლასი და WorkflowService-ის შექმნა. სერვისი - WorkflowService კლასი. ბოლო წერტილი (Endpoint);

12. WCF ტექნოლოგია სამუშაო პროცესის გამომძახებელი (WorkflowInvoker): აპლიკაციათა ამუშავებაშედეგის ანალიზი;

13. WCF ტექნოლოგია პროექტისათვის მომხმარებლის (კლიენტის) ფილიალის შექმნა ქვეკატალოგის სახით: კონფიგურაციის ფაილში პორტების განსაზღვრა. აპლიკაციათა ამუშავება ადმინისტრატორის უფლებით და შედეგების ანალიზი;

14. კონკრეტული საპრობლემო სფეროს აპლიკაციის აგება: WPF, WF და WCF ტექნოლოგიების კომპლექსური გამოყენებით Ms SQL Server მონაცემთა ბაზასთან ერთად;

15. Web - სერვისების დაპროგრამება: Receive და SendReply კონფიგურირება. PerformLookup აქტიურობის შექმნა. სერვისის ტესტირება და შედეგების ანალიზი.

5. შესავალი WCF/WPF ტექნოლოგიებში

პროგრამული აპლიკაციების ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო (Desktop) აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [1- 3].

ეს დანართები იქმნება .NET Framework -ის ორი 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 [4].

Web-სამსახურების და WCF (Windows Communication Foundation) სერვის-ორიენტირებული არქიტექტურის აგების საშუალებების გაჩენამ განაპირობა სამაგიდო- და ვებ-დანართების ფუნქციონირების შესაძლებლობა ერთიან განაწილებულ გარემოში, სადაც მონაცემთა გაცვლა ხორციელდება როგორც ლოკალურ, ასევე გლობალურ ქსელებში.

1-ელ ნახაზზე ნაჩვენებია ეს გამაერთიანებელი პროცესი.

ნახ.1. მაიკროსოფთის ჰიბრიდული ტექნოლოგიები

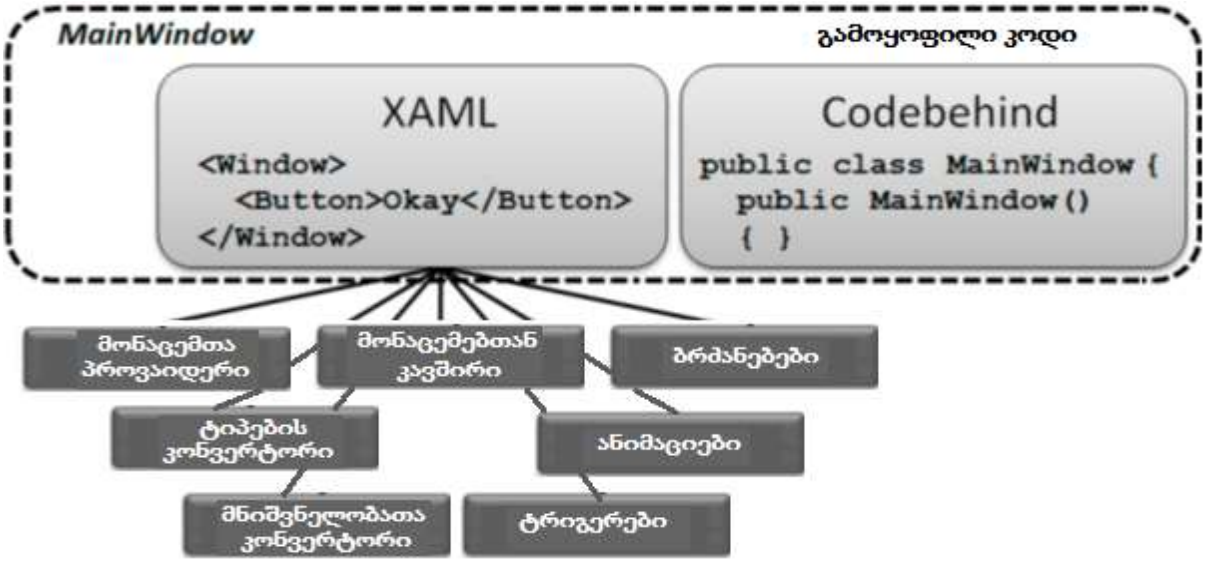
WPF – Windows Presentation Foundation არის ერთ-ერთი ასეთი გამაერთიანებელი ტექ–ნოლოგია. იგი საშუალებას იძლევა აიგოს ისეთი დანართი, რომელშიც გამორიცხულია დაპირისპირება სამაგიდო აპლიკაციასა და ინტერნეტს შორის.

WCF/WPF - დანართს შეუძლია ფუნქციონირება როგორც სამაგიდო აპლიკაციას, ასევე როგორც ვებ-აპლიკაციას ბრაუზერის შიგნით. არსებობს ასევე WPF-ის შეზღუდული ვერსია, სახელით Silverlight [lit].

პრაქტიკული სამუშაოების შესასრულებლად .NET გარემოში საჭიროა შემდეგი რესურსები:

- **ოპერაციული სისტემა** - .Net Framework-ის 4.0 (ან მეტი). .Net Framework 4.5-ის გამოჩენის შემდეგ იგი ბევრად სრულყოფილი გახდა. ახალი ფუნქციონალობის გამოსაყენებლად სასურველია კონფიგურაციის გაუმჯობესება. Windows 7 (ან 8, 10).
- გრაფიკული კარტა DirectX-ის მე-9 ვერსიისთვის;

ინტეგრირებული სამუშაო გარემო: .Net Framework 4.0/4.5 (ან მეტი). მე-2 ნახაზზე ნაჩვენებია ორი ძირითადი პროგრამული ტანდემი (XAML-თავისი მეთოდებით და C#-კოდი).



ნახ.2. ინტეგრირებული გარემო

6. ლაბორატორიული სამუშაოს შესრულების ნიმუშები

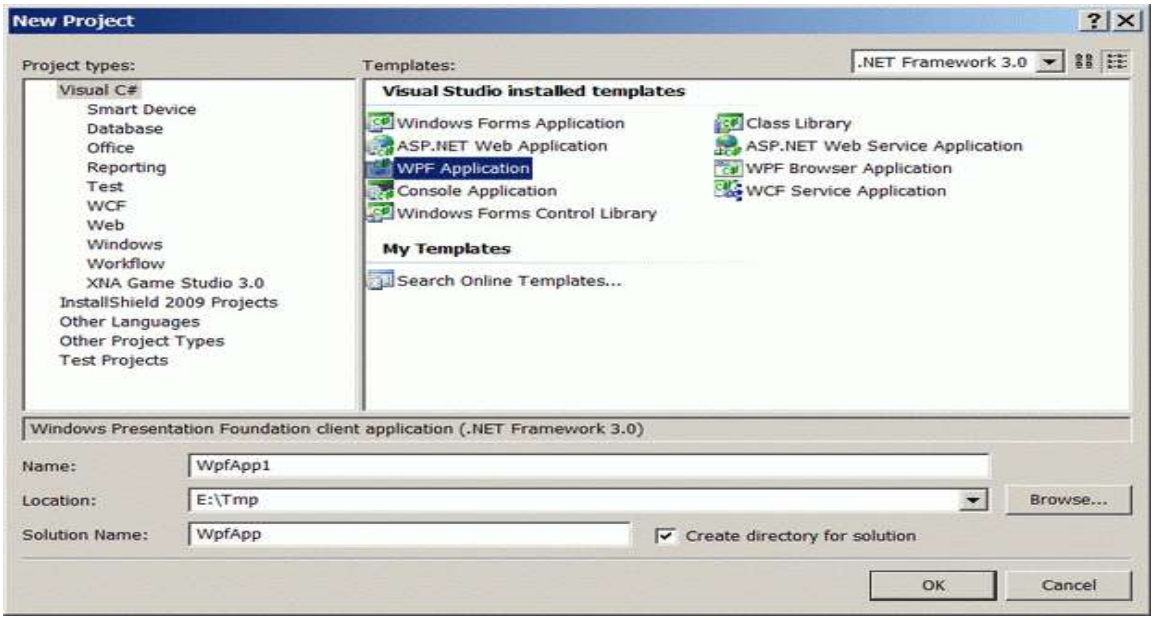
6.1. აპლიკაციების დეველოპმენტი WPF-ის ბაზაზე

6.1.1. ლაბორატორიული სამუშაო N2

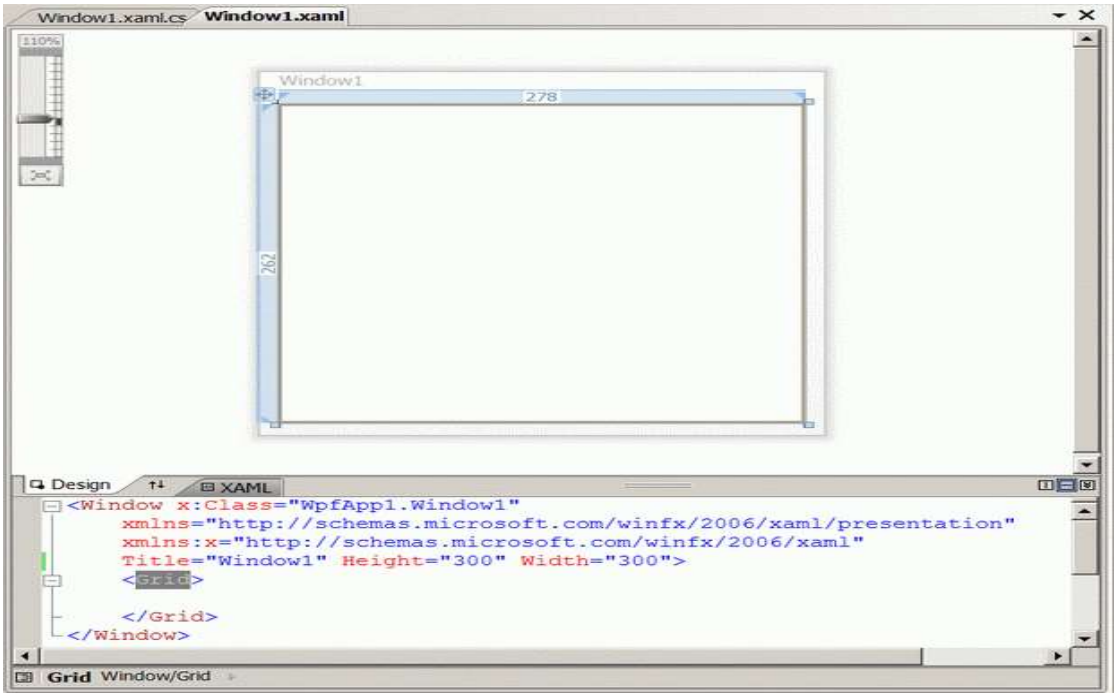
მარტივი ღილაკის დაპროგრამება ანიმაციის ელემენტებით

მიზანი: WPF-ის მულტიმედიალური შესაძლებლობების დემონსტრირება [2].

1. შექმენით პროექტი სახელით: WpfApp1 და Solution-ის სახელით: WpfApp.



ნახ.3-ა. პირველი WPF პროექტის შექმნა



ნახ.3-ბ. შედეგი

2. შეავსეთ XAML კოდი:

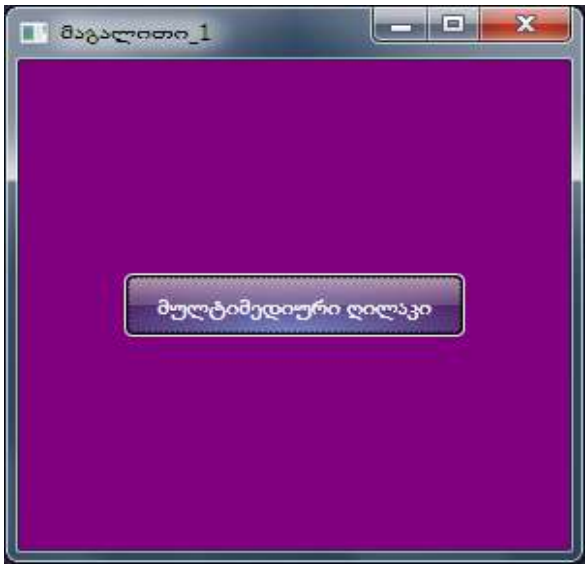
```
<!----WpfApp1----- Window1.xaml ----- >
<!-- მთლიანად ფანჯრის აწყობა: მომხმარებლის ინტერფეისი →
<Window x:Class=“WpfApp1.Window1“
xmlns=“http://schemas.microsoft.com/winfx/2006/xaml/presentation“
    xmlns:x=“http://schemas.microsoft.com/winfx/2006/xaml“
    Width=”300” Height=”300”
        Title=”მაგალითი_1”
        x:Name=”Window”
        Background=”Aqua”>

    <!-- აპლიკაციის ფანჯრის რესურსების შესანახი სექცია: →
    <Window.Resources>
   <!-- იქმნება შაბლონი დასახელებით ClassButton, ღილაკის ტიპის
        ელემენტებისთვის→
    <ControlTemplate x:Key=”ClassButton”
            TargetType=“{x:Type Button}“>
      <!-- ფანჯრის რესურსების შესანახი სექცია ღილაკისთვის →
     <ControlTemplate.Resources>
      <!-- სექციაში Storyboard აღიწერება ანიმაციური ეფექტი, მაგ.,
                    ღილაკის დაჭერა→
   <Storyboard x:Key=“Timeline1“>
        <!-- მითითებულ დროში, მაგ., 0.3 წამი, ღილაკი ხდება
                                                გაუმჭვირვალე →
     <DoubleAnimationUsingKeyFrames BeginTime=“00:00:00“
      Storyboard.TargetName=“glow“
      Storyboard.TargetProperty=“(UIElement.Opacity)“>
        <!-- ანიმაციის ბოლო წერტილი →
      <SplineDoubleKeyFrame KeyTime=”00:00:0.3” Value=”1” />
     </DoubleAnimationUsingKeyFrames>
   </Storyboard>

          <!-- სექცია Storyboard ღილაკის ჩასაქრობად →
   <Storyboard x:Key=”Timeline2”>
<!- მითითებულ დროში, მაგ., 0.3 წამში ღილაკი ხდება გამჭვირვალე→
     <DoubleAnimationUsingKeyFrames BeginTime=”00:00:00”
       Storyboard.TargetName=”glow”
       Storyboard.TargetProperty=”(UIElement.Opacity)”>
         <!-- ანიაციის ბოლო წერტილი →
      <SplineDoubleKeyFrame KeyTime=”00:00:0.3” Value=”0” />
     </DoubleAnimationUsingKeyFrames>
  </Storyboard>
</ControlTemplate.Resources>
      <!-- ღილაკის აღწერის სექცია →
        <!-- გარე საზღვარი - თეთრი →
 <Border BorderBrush=“#FFFFFFFF“ BorderThickness=“1,1,1,1“
                                    CornerRadius=“4,4,4,4“>
        <!-- შიგა საზღვარი - შავი →
 <Border x:Name=”border” Background=”#7F000000” BorderBrush=”#FF000000”
BorderThickness=”1,1,1,1” CornerRadius=”4,4,4,4”>
```

```
      <Grid>
        <Grid.RowDefinitions>
          <!-- ღილაკის ზედა ნახევარი  →
            <RowDefinition Height=”0.5*” />
              <!-- ღილაკის ქვედა ნახევარი →
                <RowDefinition Height=”0.5*” />
        </Grid.RowDefinitions>

        <!-- იხატება ღილაკის განათება →
   <Border Opacity=”0” HorizontalAlignment=”Stretch”
     x:Name=”glow” Width=”Auto”
           Grid.RowSpan=”2” CornerRadius=”4,4,4,4”>
        <Border.Background>
         <!-- მიეწოდება რადიალური გრადიენტი წანაცვლებით →
            <RadialGradientBrush>
              <RadialGradientBrush.RelativeTransform>
                 <TransformGroup>
             <ScaleTransform ScaleX=”1.702” ScaleY=”2.243” />
                    <SkewTransform AngleX=”0” AngleY=”0” />
                    <RotateTransform Angle=”0” />
               <TranslateTransform X=”-0.368” Y=”-0.152” />
                 </TransformGroup>
              </RadialGradientBrush.RelativeTransform>
              <!-- გრადიენტის ფერები ARGB  ფორმატში→
              <GradientStop Color=”#B28DBDFF” Offset=”0” />
              <GradientStop Color=”#008DBDFF” Offset=”1” />
              </RadialGradientBrush>
            </Border.Background>
          </Border>
     <!-- ათინათის  დახატვა →
     <ContentPresenter HorizontalAlignment=”Center” VerticalAlignment=”Center”
Width=”Auto” Grid.RowSpan=”2” />
    <Border HorizontalAlignment=”Stretch” x:Name=”shine”
       Width=”Auto” CornerRadius=”4,4,0,0”>
        <Border.Background>
          <LinearGradientBrush StartPoint=”0.494,0.028”
             EndPoint=”0.494,0.889”>
              <GradientStop Color=”#99FFFFFF” Offset=”0” />
              <GradientStop Color=”#33FFFFFF” Offset=”1” />
             </LinearGradientBrush>
        </Border.Background>
      </Border>
     </Grid>
    </Border>
  </Border>
<!--ტრიგერული მოვლენების დაყენება,  მაუსის დაჭერის რეაქციაზე →
     <ControlTemplate.Triggers>
       <!-- მაუსის ღილაკი დაჭერილია →
     <Trigger Property=”IsPressed” Value=”True”>
  <Setter Property=”Opacity” TargetName=”shine” Value=”0.4” />
```

```
  <Setter Property="Background" TargetName="border"
                                        Value="#CC000000" />
  <Setter Property="Visibility" TargetName="glow"
                                        Value="Hidden" />
   </Trigger>
     <!-- მაუსის კურსორი ობიექტზეა → 
       <Trigger Property="IsMouseOver" Value="True">
    <!-- ობიექტზე შესვლა - ანიმაციის გამოძახება Timeline1 →
         <Trigger.EnterActions>
  <BeginStoryboard Storyboard="{StaticResource Timeline1}" />
         </Trigger.EnterActions>
  <!-- ობიექტიდან გამოსვლა - ანიმაციის გამოძახება Timeline2 →
         <Trigger.ExitActions>
  <BeginStoryboard Storyboard="{StaticResource Timeline2}" />
         </Trigger.ExitActions>
     </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </Window.Resources>
    <Grid>
        <!-- იქმნება ღილაკის ეგზემპლარი ფანჯრის შუაში →
        <Button x:Name="Btn1" HorizontalAlignment="Center"
     VerticalAlignment="Center"  Width="176" Height="34"
    Content="მულტიმედიალური ღილაკი" Foreground="#FFFFFFFF"
      Template="{DynamicResource ClassButton}" />
    </Grid>
</Window>
```

3. ავამუშავოთ პროგრამა, მივიღებთ მე-4 ნახაზზე ნაჩვენებ შედეგს, სადაც შაბიამნისფერის ფონია (ა). ღილაკზე დაჭერით იცვლება ფონი იასამნისფერით (ბ). შემდგომ ფერების შეცვლა მონაცვლეობით ხდება მაუსის დაკლიკვით.

ნახ.4-ა. შაბიამნისფერი ეკრანით



ნახ.4-ბ. იასამნისფერი ეკრანით

4. დავაპროგრამოთ ლოგიკა C# ენაზე:

```
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 WpfApp1
{
  // Interaction logic for Window1.xaml
  public partial class Window1 : Window
  {
    System.Windows.Media.Brush color;
    bool colorFlag = true;
   public Window1()
   {
      //InitializeComponent();
    Application.LoadComponent(this, new Uri("Window1.xaml",
                    UriKind.Relative));

    Btn1.Click += new RoutedEventHandler(Btn1_Click);
    color = this.Window.Background;
   }
   void Btn1_Click(object sender, RoutedEventArgs e)
      {
        // ფინის ფერის შეცვლა
       if (colorFlag)
         this.Window.Background =
             System.Windows.Media.Brushes.Purple;
       else
         this.Window.Background = color;

       colorFlag = !colorFlag;
}}}
```

5. Application - ობიექტი

ნებისმიერი დანართი (აპლიკაცია) იყენებს Application კლასს, რომელიც Run() მეთოდის საშუალებით ამ აპლიკაციას მიუერთებს ოპერაციული სისტემის მოვლენების მოდელს ასამუშავებლად.

Application - ობიექტი მართავს დანართის სიცოცხლის დროს, აკვირდება ხილვად ფანჯრებს, ათავისუფლებს რესურსებს და აკონტროლებს აპლიკაციის გლობალურ მდგომარეობას. Run() მეთოდი აამუშავებს შესრულების გარემოს დისპეტჩერს, რომელიც დაიწყებს მოვლენების და შეტყობინებების გადაგზავნას დანართის კომპონენტებთან.

დროის მოცემულ მომენტში აქტიური შეიძლება იყოს მხოლოდ ერთი Application - ობიექტი, და ის მუშაობს მანამ, სანამ დანართი არ დასრულდება. შესრულებად Application - ობიექტზე მიმართვა შეიძლება დანართის ნებისმიერი ადგილიდან Application.Current სტატიკური თვისების საშუალებით.

WPF დანართის (და Application ობიექტის) სასიცოცხლო დრო შედგება შემდეგი ეტაპებისგან:

1. კონსტრუირდება Application ობიექტი;
2. გამოიძახება მისი Run() მეთოდი;
3. ინიცირდება მოვლენა Application.Startup;
4. მომხმარებლის კოდი აკონსტრუირებს ერთ ან რამდენიმე Window (ან Page) ობიექტს და დანართი მუშაობს;
5. გამოიძახება მეთოდი Application.Shutdown();
6. გამოიძახება მეთოდი Application.Exit().

ჩვენი WPF-დანართის აგებისას სისტემამ თვითონ შექმნა Solution Explorer-ში ორი ფაილი Application-ობიექტთან დაკავშირებული ტიპური სახელებით: App.xaml და App.xaml.cs. მათი ნახვა შესაძლებელია. აქ არ ჩანს ცხადად არც Application და Window ობიექტების შექმნა და არც Run() მეთოდის გამოძახება (!)

WPF-პლატფორმის შემქმნელებმა გადაწყვიტეს, რომ, ვინაიდან ეს სტანდარტული ოპერაციები გამოიყენება ყველა დანართისათვის, არაა საჭირო მისი მომხმარებელზე გადაცემა და თვით კომპილატორი (სისტემა) ავტომატურად გამოიყენებს მათ (!), თვითონ შექმნის Application ობიექტს WPF-პროექტის კომპილაციის დროს, შექმნის ასევე Window (ან Page) ობიექტებს და გადასცემს მათ Application.Run() მეთოდს.

სხვა სიტყვებით რომ ვთქვათ, ჩვენ „ვერ ვხედავთ ცხადად“ კომპილატორის მიერ ასეთი კოდის შესრულების ტექსტს:

```
// ეს ტექსტი მოტანილია საილუსტრაციოდ ---------
public partial class App : System.Windows.Application
{
  public App()
    {   System.Windows.Window win = new System.Windows.Window();
        win.Title = “Hello World”;
        win.Show();
    }
      // Application entry point
    [System.STAThreadAttribute()]
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    public static void Main()
    {
        App app = new App();
        app.Run();
    }
}
```

ჩვენი დანართის მაგალითისათვის App.xaml.cs ფაილი ასე გადავაკეთოთ:

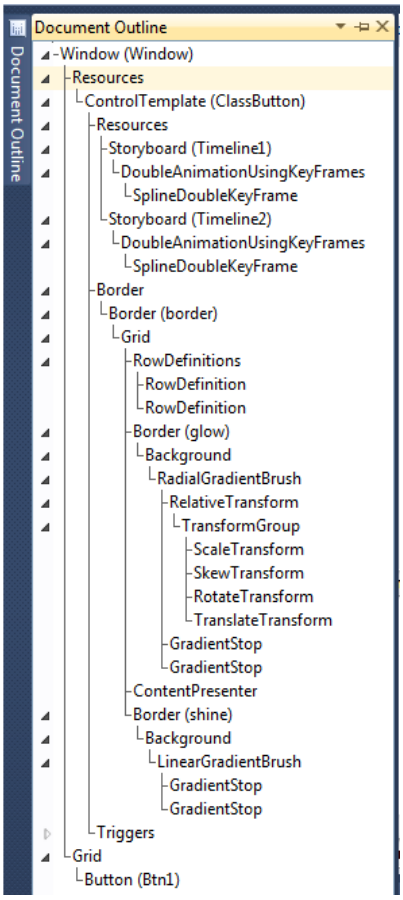
```
// ----- App.xaml.cs  -------------------------
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Windows;

namespace WpfApp1
{
 public partial class App : Application
  {
   public App()
    {
     // გაშვებულია Application ობიექტი
      this.Startup += new StartupEventHandler(App_Startup);
        // მოვლენა დაუმუშავებელი გამოსარიცხი სიტუაციისთვის
  this.DispatcherUnhandledException += App_DispatcherUnhandledException;
    }

  void App_DispatcherUnhandledException(object sender,
      System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)
    {
       e.Handled = true;// შესრულების გაგრძელება
    }

  void App_Startup(object sender, StartupEventArgs e)
    {
       // იქმნება სასტარტო ფანჯრის ობიექტი -----
       Window1 win = new Window1();
       // ფანჯრის ობიექტის აწყობა -----
       win.Title = “ფანჯრის  ახალი სათაური “;
       // ფანჯრის ნახვა ---------
       win.Show();
    }
  }
}
```

ავამუშავოთ დანართი და ვნახოთ შედეგი.

6. XAML-პროგრამის სტრუქტურის სანახავად Document Outline პანელის გამოტანა:

View/Other Windows/Document Outline

მიიღება შემდეგი იერარქიული სურათი (ნახ.5).



ნახ.5

6.1.2. ლაბორატორიული სამუშაო N 6

WPF-ში Web-გვერდების აპლიკაციების შექმნა

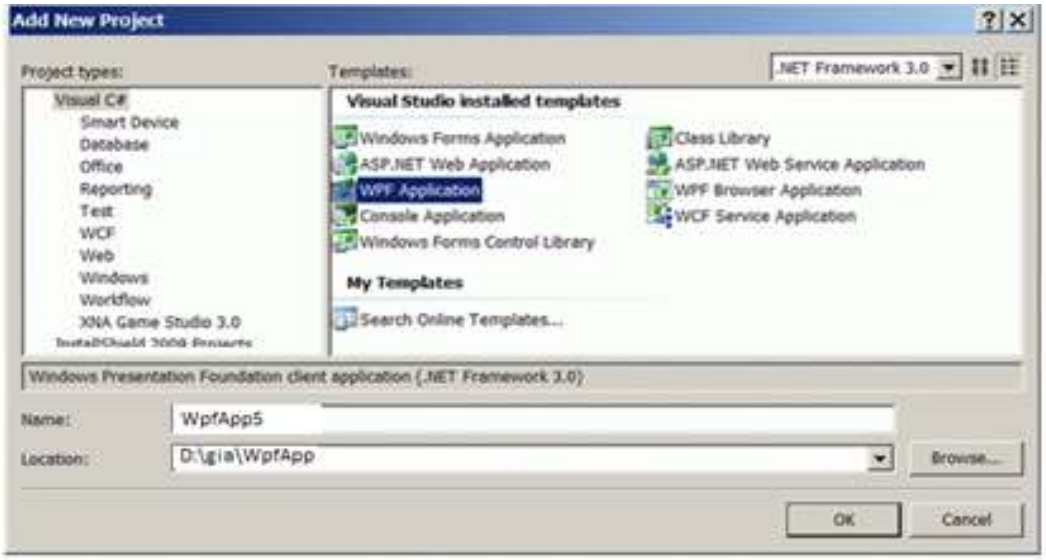
მიზანი: WPF-ის ვებ-აპლიკაციის დამუშავება მომხმარებელთა ინტერფეისების ვიზუალური მართვის ელემენტებით [2].

WPF-პლატფორმას აქვს ვებ-გვერდების შექმნის საშუალებები. ასეთი გვერდების გამოყენება ხშირად მომხმარებლისთვის უფრო მოსახერხებელია, ვიდრე ფანჯრული ორგანიზაცია.

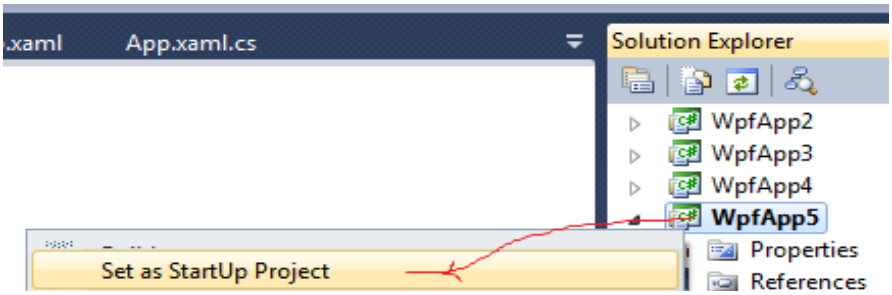
გვერდული დანართის (აპლიკაციის) შიგთავსი ჩაშენდება სპეციალურ ნავიგაციურ კარკასში, რომელსაც აქვს ნავიგაციური კავშირები და ნავიგაციური ჟურნალი.

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. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp5 სახელით (ნახ.6) და გავხადოთ „სასტარტო“ (ნახ.7).



ნახ.6



ნახ.7

2. წავშალოთ ავტომატურად შექმნილი Window1.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.

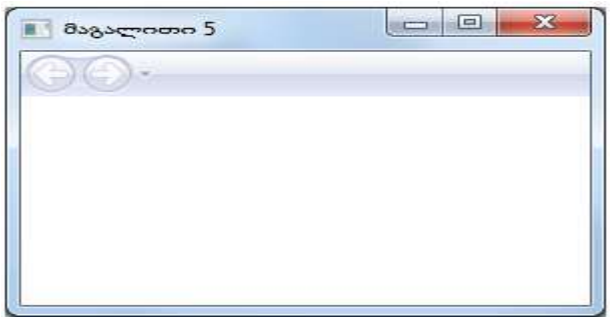
5. გავხსნათ ფაილი NavExample.xaml და შევასწოროთ მასში დესკრიპტორული კოდი:

```
<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());
        }
    }
}
```

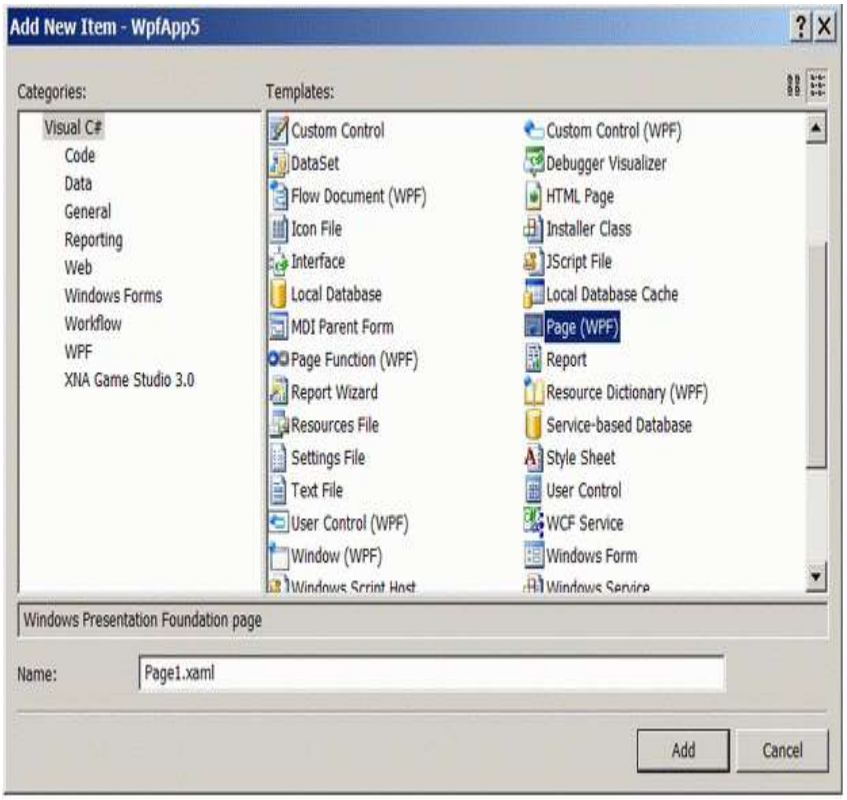
ავამუშავოთ პროექტი, შედეგი იქნება ცარიელი გვერდი (ნახ.8) ნავიგაციური ელემენტით.



ნახ.8

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.9).



ნახ.9. 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>
```

აამუშავეთ პროექტი. შეამოწმეთ 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" />
```

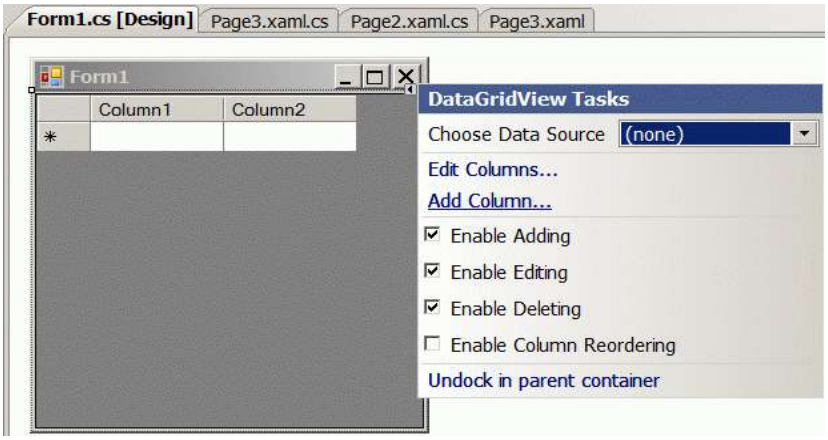
```
        <GradientStop Color="BlueViolet" Offset="1" />
        </LinearGradientBrush>
      </Button.Foreground>
      <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
          <GradientStop Color="Red" Offset="0" />
         <GradientStop Color="Orange" Offset=".17" />
         <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

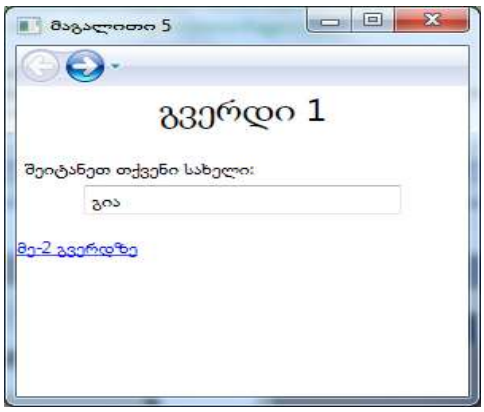
21. ღილაკების საერთო დამმუშავებელი Page3 კოდის ნაწილში შევავსოთ ასე:

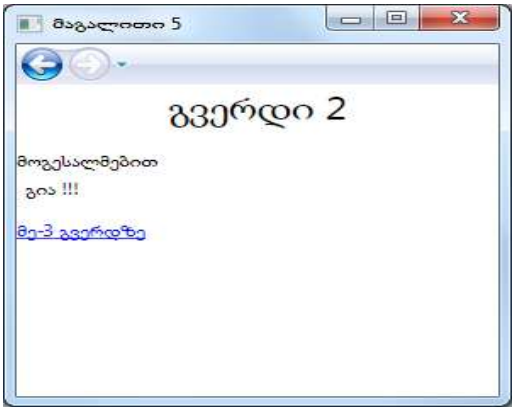
```
//--- ლისტინგი -------
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
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 ადასტურების WPF და Windows Form ტექნოლოგიების ერთობლივი მუშაობის შესაძლებლობას.

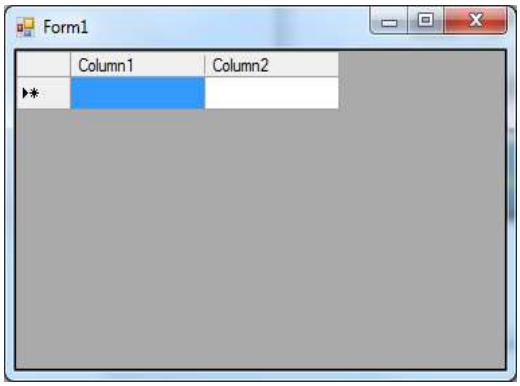
შედეგები (ნახ.11–ა,ბ,გ):





ნახ.11–ა

ნახ.11–ბ



ნახ.11–გ

6.1.3. ლაბორატორიული სამუშაო N 12

WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: StackPanel და DockPanel

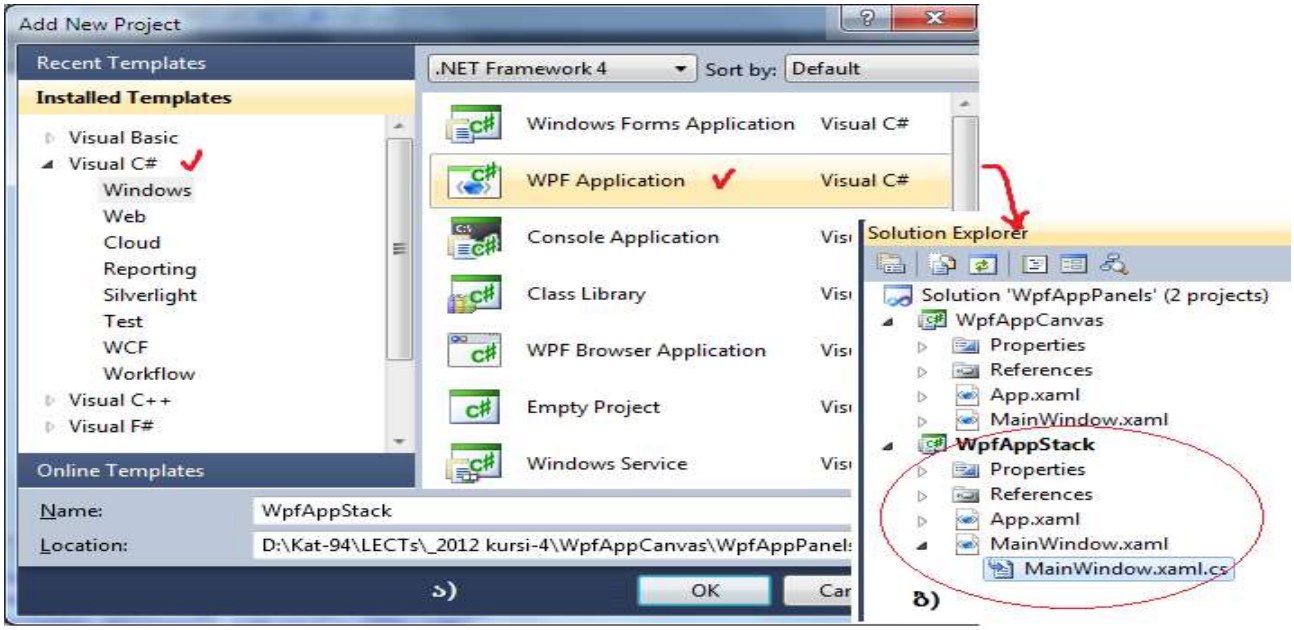
მიზანი: WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის StackPanel და DockPanel ფუნქციონალობის შესწავლა.

პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება შემდეგი პანელები: Canvas, *StackPanel, DockPanel*, UniformGrid და Grid [2].

StackPanel განათავსებს ფანჯარაზე მოთავსებულ ელემენტებს ვერტიკალში სტრიქონების სახით. Orientation თვისებით შეიძლება განლაგება ვერტიკალურად ან ჰორიზონტალურად განხორციელდეს. გამოუცხადებლად, StackPanel იკავებს კლიენტის სამუშაო არეს (მაგ., ფანჯარას) მთლიანად. მის ყოველ შვილობილ ელემენტს გამოუყოფს სლოტს (სასიცოცხლო არე), ხოლო თავის ზომას გაითვლის შვილობილებიან მაქსიმალურის მიხედვით.

გამოყოფილი სლოტის მიხედვით შვილობილ ელემენტს შეუძლია პოზიციონირება თავისი შეხედულებისამებრ, Margin თვისების გამოყენებით. HorizontalAlignment და VerticalAlignment სლოტის შიგა ადგილი. ელემენტებსშორისი მანძილი მოიცემა Padding თვისებით.

1. დავამატოთ ახალი WpfAppStack პროექტი ძველ Solution WpfAppPanels -ში და გავხადოთ იგი სასტარტო (ნახ.12-ა,ბ).



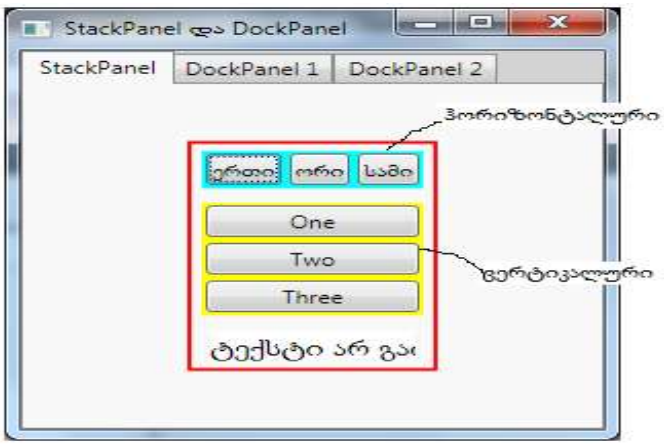
ნახ.12

2. დავამატოთ MainWindow.xaml კოდში შემდეგი ტექსტი (ჯერ StackPanel–ის გამოყენებისთვის):

```
<Window x:Class="WpfAppStack.MainWindow"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Title="StackPanel და DockPanel" Height="300" Width="300"
     Background="LightGray"
```

```
        >
    <!--  StackPanel-ის გამოყენება →
    <TabControl>
        <TabItem Header="StackPanel">
            <Border BorderBrush="Red" BorderThickness="2"
                 HorizontalAlignment="Center"
                 VerticalAlignment="Center">
              <StackPanel Orientation="Vertical">
              <StackPanel Orientation="Horizontal" Margin="5"
                   Background="Aqua">
                <Button Margin="2">ერთი</Button>
                <Button Margin="2">ორი</Button>
                <Button Margin="2">სამი</Button>
              </StackPanel>
              <StackPanel Orientation="Vertical" Margin="5"
                      Background="Yellow">
                <Button Margin="2">One</Button>
                <Button Margin="2">Two</Button>
                <Button Margin="2">Three</Button>
              </StackPanel>
              <StackPanel Orientation="Horizontal" Margin="5"
                      Width="100" Background="White">
                 <TextBlock FontSize="12pt"
                   TextWrapping="Wrap">ტექსტი არ გადაიტანება,
                   რადგან ის ჩატვირთულია StackPanel-ში
                 </TextBlock>
               </StackPanel>
              </StackPanel>
            </Border>
        </TabItem>
    </TabControl>
</Window>
```

3. ავამუშავოთ პროგრამა და მივიღებთ შედეგს (ნახ.13).

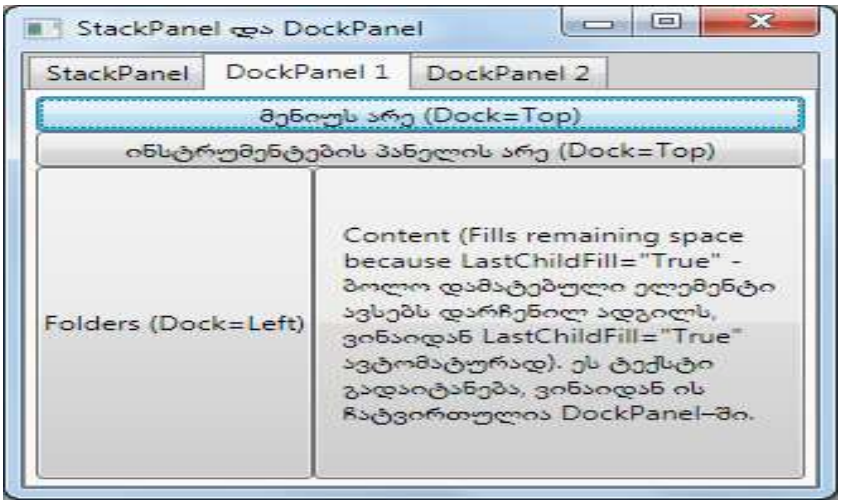


ნახ.13

4. ახლა დავამატოთ MainWindow.xaml ფაილში ახალი ტექსტი DockPanel–ის გამოსაყენებლად, რომელიც აისახება „DockPanel 1“ გვერდზე:

```
<!–Windows Explorer სივრცის იმიტაცია DockPanel–ის გამოყენებით →
   <TabItem Header=”DockPanel 1”>
        <DockPanel>
   <Button DockPanel.Dock=”Top”>მენიუს არე (Dock=Top)</Button>
   <Button DockPanel.Dock=”Top”>ინსტრუმენტების პანელის არე
            (Dock=Top)</Button>
   <Button DockPanel.Dock=”Left”>Folders (Dock=Left)</Button>
   <Button>
     <TextBlock TextWrapping=”Wrap” Padding=”5pt”>
        Content (Fills remaining space because
           LastChildFill=”True” - ბოლო დამატებული ელემენტი
           ავსებს დარჩენილ ადგილს, ვინაიდან
                 LastChildFill=”True” ავტომატურად).
                 ეს ტექსტი გადაიტანება, ვინაიდან ის
                 ჩატვირთულია DockPanel–ში.
    </TextBlock>
   </Button>
  </DockPanel>
</TabItem>
```

5. პროგრამის ამუშავებით მიიღება ასეთი შედეგი (ნახ.14).



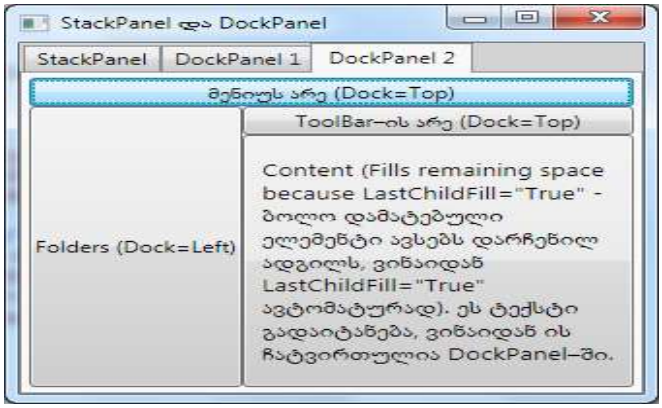
ნახ.14

6. MainWindow.xaml ფაილში დავამატოთ ტექსტი `DockPanel 2` –თვის:

```
 <TabItem Header=“DockPanel 2“>
   <DockPanel LastChildFill=”True”>
     <Button DockPanel.Dock=”Top”>მენიუს არე
              (Dock=Top)</Button>
     <Button DockPanel.Dock=”Left”>Folders
              (Dock=Left)</Button>
     <Button DockPanel.Dock=”Top”>ToolBar–ის არე
              (Dock=Top)</Button>
     <Button>
        <TextBlock TextWrapping=”Wrap” FontSize=”10pt”
            Padding=”5pt”>
          Content (Fills remaining space because
          LastChildFill=”True”-ბოლო დამატებული ელემენტი
```

```
            ავსებს დარჩენილ ადგილს, ვინაიდან
            LastChildFill="True" ავტომატურად).
               ეს ტექსტი გადაიტანება, ვინაიდან ის
               ჩატვირთულია DockPanel-ში.
          </TextBlock>
        </Button>
      </DockPanel>
    </TabItem>
```

7. პროგრამის ამუშავების შედეგად მიიღება (ნახ.15):



ნახ.15

დასკვნა:

1. პირველ გვერდზე იხატება ჩარჩო მოცემული სიგანით და ფერით, რომელშიც თავსდება StackPanel, ჯერ ჰორიზონტალური (ერთი, ორი, სამი), შემდეგ ვერტიკალური ორიენტაციით (One, Two, Three). მათი ფონის ფერები განსხვავებულია. პანელში ქვემოთ მოცემულია ტექსტური ბლოკი, რომელიც სტრიქონს არ გადაიტანს (თუმცა ჩართულია თვისება TextWrapping="Wrap").

2. სტრიქონის გადატანა ტექსტურ ბლოკში შესაძლებელია მაშინ, როცა იგი ჩატვირთულია DockPanel პანელში.

3. DockPanel-ში განლაგების სტრუქტურა დამოკიდებულია ელემენტების დამატების მიმდევრობაზე ამ პანელში. LastChildFill თვისება უფლებას აძლევს ბოლო შვილობილ ელემენტს დაიკავოს მთელი თავისუფალი სივრცე. DockPanel-ის ელემენტებისთვის მოქმედებს მიბმის თვისება, ანუ თავისუფალი ადგილის რომელ მხარეს უნდა მიებას ახალი ელემენტი.

6.2. კომუნიკაცია აპლიკაციებს შორის WCF-ის გაზაზე

6.2.1. ლაბორატორიული სამუშაო N 1

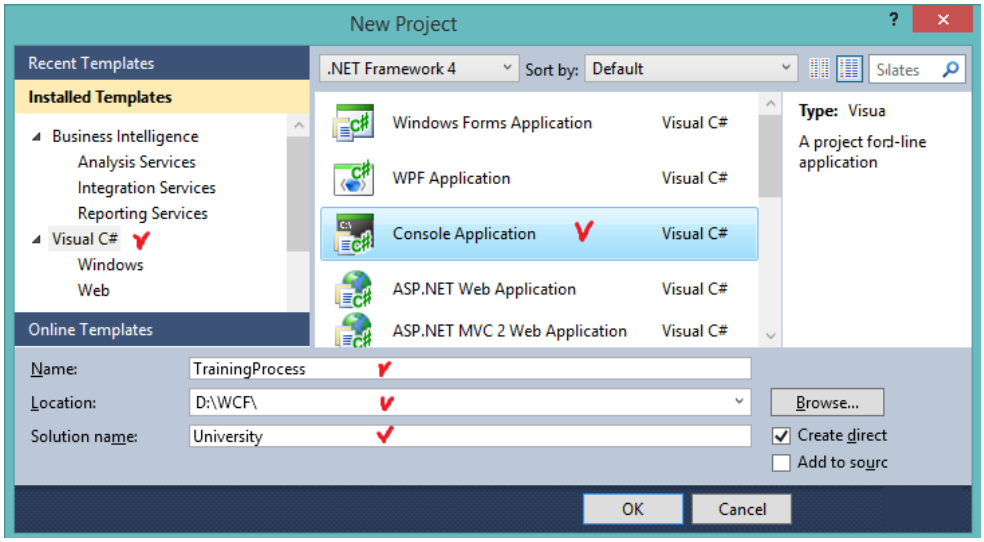
ინფორმაციის „გადაცემის“ და „მიღების“ ქმედებები [1]

მიზანი: Visual Studio.NET პაკეტის WCF-ის სამუშაო გარემოს გაცნობა და პირველი აპლიკაციის აგება. Send და Receive ქმედებების გაცნობა და დაპროგრამება.

მთავარი ქმედებები (Activities), რომლებიც კომუნიკაციისთვის გამოიყენება არის Send (გაგზავნა) და Receive (მიღება). ეს ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply) გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და მონიტორინგის პროცესისთვის.

1. ახალი პროექტის შექმნა კონსოლის რეჟიმში

შევქმნათ ახალი პროექტი TrainingProcess სახელით, Solution name: University და Windows -> Console Application რეჟიმში.

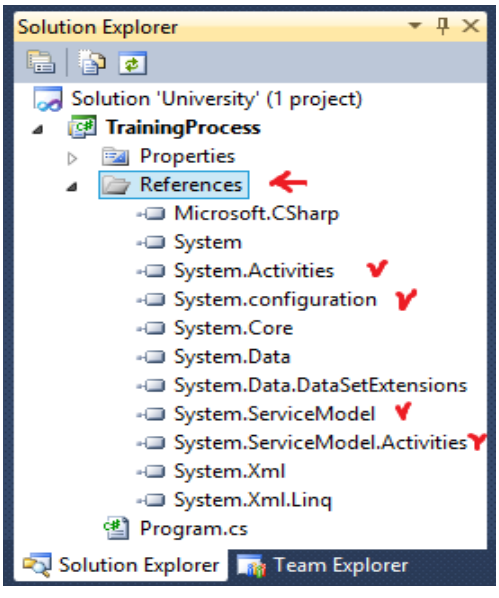


ნახ.16-ა

TrainingProcess პროექტის Solution Explorer–ში მაუსის მარჯვენა ღილაკით ავირჩიოთ Add Reference და .NET tab-დან დავამატოთ შემდეგი სტრიქონები.

- System.Activities
- System.Configuration
- System.ServiceModel
- System.ServiceModel.Activities

მიიღება (ნახ.16-ბ)



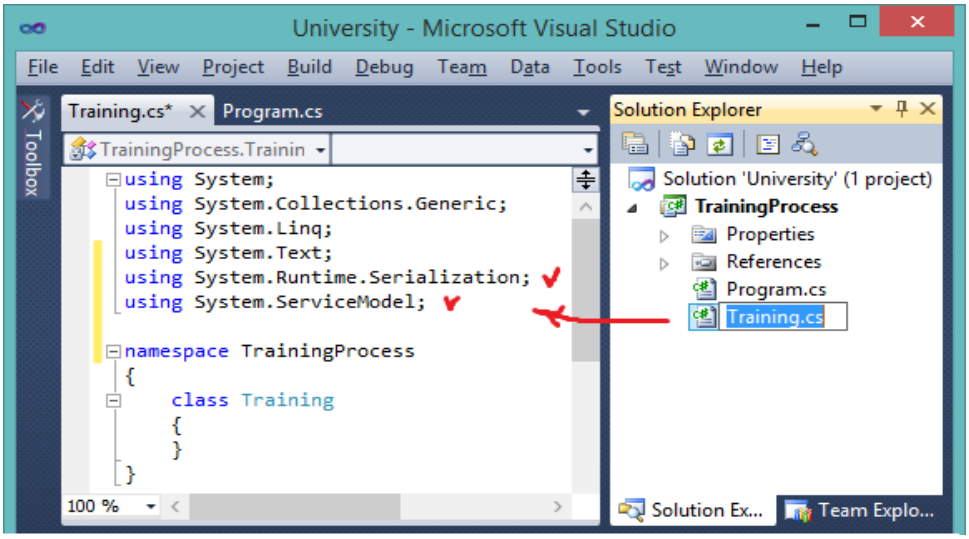
ნახ.16-ბ. შედეგი

2. შეტყობინებათა განსაზღვრა

საჭიროა შეიქმნას კლასი, რომელიც განსაზღვრავს შეტყობინებებს აპლიკაციებს შორის. Solution Explorer-იდან Add -> Class და ჩავწეროთ კლასის სახელი Training.cs. ამ ფაილში დავამატოთ ორი სახელსივრცე (namespaces):

```
using System.Runtime.Serialization;
using System.ServiceModel;
```

მიიღება მე-17 ნახაზზე ნაჩვენები ფანჯარა.



ნახ.17. Training.cs კლასის შექმნა

Training.cs ფაილში განვსაზღვროთ (ჩავამატოთ) სამი კლასი:

- Branch: განსაზღვრავს **მონაცემებს** ტრეინინგ-ცენტრის ფილიალის **მდებარეობის შესახებ;**
- TrainingRequest: განსაზღვრავს ფილიალის **მოთხოვნას** სასწავლო საგნის (კურსის დისციპლინის) მითითებით;
- TrainingResponse: განსაზღვრავს ტრეინინგ-ცენტრის პასუხს მოთხოვნის შესაბამისი ფილიალისთვის.

Branch კლასის განსაზღვრა მოცემულია 1.1 ლისტინგში. შევიტანოთ იგი namespace TrainingProcess-ის შიგნით. წავშალოთ ცარიელი კლასი Training, რომელიც წინა ბიჯზე შეიქმნა.

```
// --- ლისტინგი_ 1 .1 ---
// --- ფილიალის მონაცემთა სტრუქტურის განსაზღვრა ---
public class Branch
{
  public String BranchName { get; set;}
  public String Address { get; set; }
  public Guid BranchID { get; set; }
  #region Constructors
  public Branch()   {  }
  public Branch(String name, String address)
   {
     BranchName = name;
     Address = address;
```

```
     BranchID = Guid.NewGuid();
   }
 public Branch(String name, String address, Guid id)
  {
    BranchName = name;
    Address = address;
    BranchID = id;
  }
public Branch(String name, String address, String id)
  {
    BranchName = name;
    Address = address;
    BranchID = new Guid(id);
  }
#endregion Constructors
}
```

Branch კლასს აქვს სამი დასამახსოვრებელი წევრი“: სახელი, ქსელის მისამართი და უნიკალური იდენტიფიკატორი. ზოგიერთი კონსტრუქტორი შემოტანილია გამოყენების გასამარტივებლად. აქ დამატებულია რეგიონის მარკერები კონსტრუქტორის ირგვლივ, შესაძლებელი რომ იყოს კოდის შეკუმშვა მისი კითხვადობის გასაუმჯობესებლად.

ახლა დავამატოთ TrainingRequest კლასის განსაზღვრება:

```
//---ლისტინგი_1.2  ---
// --- მოთხოვნის შეტყობინების განსაზღვრა: TrainingRequest ---
[MessageContract(IsWrapped = false)]
public class TrainingRequest
    {
        private String _Jgupi;
        private String _Sagani;
        private String _Lector;
        private Guid _RequestID;
        private Branch _Requester;
        private Guid _InstanceID;

        #region Constructors
        public TrainingRequest() { }

public TrainingRequest(String sagani, String lector,
                          String jgupi, Branch requestor)
    {
       _Sagani = sagani;
       _Lector = lector;
       _Jgupi = jgupi;
       _Requester = requestor;
       _RequestID = Guid.NewGuid();
    }
public TrainingRequest(String sagani, String lector,
                  String jgupi, Branch requestor, Guid id)
    {
       _Sagani = sagani;
       _Lector = lector;
       _Jgupi = jgupi;
       _Requester = requestor;
       _RequestID = id;
     }
```

```
    #endregion Constructors

    #region Public Properties
    [MessageBodyMember]
    public String sagani
      {
           get { return _Sagani; }
           set { _Sagani = value; }
      }
    [MessageBodyMember]
     public String Jgupi
      {
           get { return _Jgupi; }
           set { _Jgupi = value; }
      }
     [MessageBodyMember]
     public String Lector
      {
           get { return _Lector; }
           set { _Lector = value; }
      }
     [MessageBodyMember]
     public Guid RequestID
       {
           get { return _RequestID; }
           set { _RequestID = value; }
       }
     [MessageBodyMember]
     public Branch Requester
       {
           get { return _Requester; }
           set { _Requester = value; }
       }
     [MessageBodyMember]
     public Guid InstanceID
       {
           get { return _InstanceID; }
           set { _InstanceID = value; }
       }
       #endregion Public Properties
}
```

TrainingRequest კლასი შედგება Jgupi, Sagani და Lector წევრებისაგან მოთხოვნილი ტრეინინგის კურსის აღსაწერად. იგი შეიცავს ასევე Branch კლასს, რომელიც ასახავს მოთხოვნილი ტრეინინგის განმახორციელებელ ფილიალს.

3. კონტრაქტის შეტყობინება

ვინაიდან TrainingRequest კლასი გამოყენებულ უნდა იყოს გამომავალი შეტყობინების განსაზღვრისთვის, MessageContract ატრიბუტი მიუთითებს, რომ ეს კლასი ჩართულ იქნება SOAP ბარათში. SOAP-ის გამოყენების დროს შეტყობინებები გადაიცემა XML-ის მსგავსი ფორმატირებადი ენით. ეს უზრუნველყოფს კლიენტებსა და სერვერს შორის მაღალი ხარისხის ურთიერთქმედების პლატფორმას. SOAP არის სტანდარტული პროტოკოლი, რომლის მხარდაჭერაც აქვს WCF-ს.

არსებობს აგრეთვე MessageBodyMember ატრიბუტი მის ყოველ პაბლიკ–თვისებაზე. ეს აუცილებელია WCF-ფენისთვის, რათა სწორად დაფორმატდეს SOAP შეტყობინება.

ახლა შევიტანოთ რეალიზაციის კოდი TrainingResponse კლასისითვის, რომელიც 1.3 ლისტინგზეა მოცემული.

```
//--- ლისტინგი_1.3 ---
// პასუხის შეტყობინების განსაზღვრა: ReservationResponse ---
[MessageContract(IsWrapped = false)]
public class ReservationResponse
  {
    private bool _Reserved;
    private Branch _Provider;
    private Guid _RequestID;
   #region Constructors
    public ReservationResponse()  {  }
public ReservationResponse(ReservationRequest request,
                                  bool reserved,Branch provider)
      {
        _RequestID = request.RequestID;
        _Reserved = reserved;
        _Provider = provider;
       }
    #endregion Constructors
     #region Public Properties

 [MessageBodyMember]
 public bool Reserved
  {
    get { return _Reserved; }
    set { _Reserved = value; }
  }
  [MessageBodyMember]
   public Branch Provider
     {
       get { return _Provider; }
       set { _Provider = value; }
     }
   [MessageBodyMember]
    public Guid RequestID
     {
       get { return _RequestID; }
       set { _RequestID = value; }
     }
     #endregion Public Properties
}
```

TrainingResponse კლასი შეიცავს ლოგიკურ ელემენტს (Reserved), რომელიც მიუთითებს იყო თუ არა სატრეინინგო მოთხოვნა ხელმისაწვდომი. ის შეიცავს ასევე Branch კლასს, რომელიც ასახავს იმ ფილიალს, რომელმაც შესრულა ეს მოთხოვნა.

4. სერვისის კონტრაქტი

WCF–ის ბოლო წერტილის განსაზღვრისთვის არსებობს ინფორმაციის სამი პორცია, რომლებიც მითითებულ უნდა იქნას: მიერთება (binding), მისამართი და კონტრაქტი.

მიერთება მიუთითებს იმ პროტოკოლს, რომელიც გამოიყენება (მაგალითად, HTTP, TCP ან სხვ.).

მისამართი მიუთითებს თუ სად უნდა ვიპოვოთ ბოლო წერტილი და მისამართის ტიპს, რომლის გამოყენება დამოკიდებულია მიერთებაზე. მაგალითად, HTTP-მიერთებისას უნდა მიეთითოს URL, ხოლო TCP-თვის მისამართი იქნება სერვერის სახელი ან IP-მისამართი.

კონტრაქტი განისაზღვრება ServiceContract-ით, რომელიც არის ინტერფეისი. იგი განსაზღვრავს მეთოდებს, რომლებიც მიწვდომადია ბოლო წერტილში.

ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

ახლა დავამატოთ ინტერფეისის განსაზღვრება, რომელიც 1.4 ლისტინგზეა მოცემული იმავე Training.cs ფაილში.

```
//--- ლისტინგი_1.4 ---
//--- სერვისის კონტრაქტის განსაზღვრა: ItrainingProcess ---
// შეიცავს ორ მეთოდს: RequestTraining() and RespondToRequest()
[ServiceContract]
    public interface ItrainingProcess
    {
        [OperationContract(IsOneWay = true)]
        void RequestTraining(TrainingRequest request);

        [OperationContract(IsOneWay = true)]
        void RespondToRequest(TrainingResponse response);
    }
```

RequestTraining() მეთოდი გამოძახებულ იქნება კლიენტის მიერ TrainingRequest შეტყობინების გადასაცემად სერვერზე. ამასთანავე RespondToRequest() მეთოდი გააგზავნის TrainingResponse შეტყობინებას უკან – კლიენტისკენ.

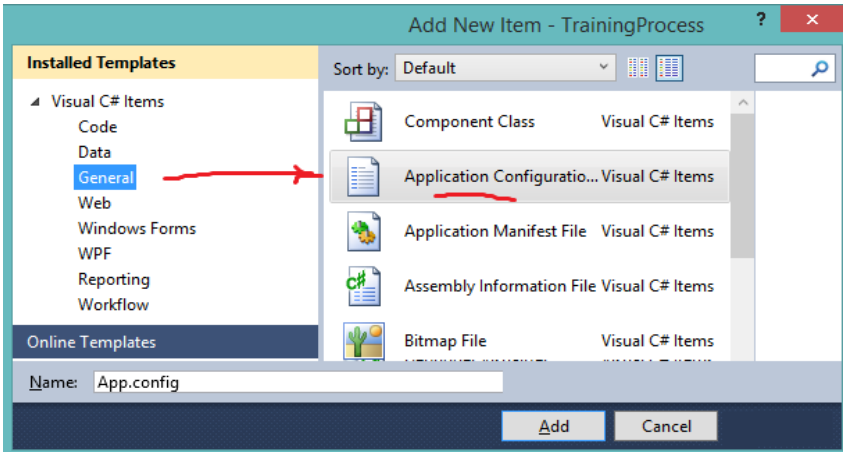
დავაკომპილიროთ პროექტი (F6) და გავმართოთ კოდი.

5. აპლიკაციის კონფიგურაცია

როგორც ადრე აღვნიშნეთ, ჩვენ გვექნება აგებული აპლიკაციის რამდენიმე დუბლი (კოპიო), რომლებიც ასახავს სხვადასხვა ფილიალების განთავსებას. ფილიალთა მონაცემები ინახება კონფიგურაციის ფაილებში. აპლიკაციის ყოველ კოპიოს ექნება თავისი კონფიგურაციის ფაილი, რომელიც შეიცავს თავის სპეციფიკურ ატრიბუტებს.

TrainingProcess პროექტის Solution Explorer–დან ვირჩევთ Add New->Item. დიალოგურ ფანჯარაში General ჯგუფში ვირჩევთ Application Configuration File (ნახ.18).

ფაილის სახელი ავტომატურად არის App.config (). კონფიგურაციის ფაილში შევიტანოთ საჭირო მონაცემები (ლისტინგი_1.5).



ნახ.18. აპლიკაციის კონფიგურაციის შექმნა

```
<!— ლისტინგი_1.5 ———— –>
<?xml version=“1.0“ encoding=“utf-8“ ?>
  <configuration>
   <appSettings>
     <add key=”Branch Name” value=”Central Library”/>
     <add key=”ID” value=”{43E6DADD-4751-4056-8BB7-7459B5C361AB}”/>
     <add key=”Address” value=”8000”/>
     <add key=”Request Address” value=”8730”/>
   </appSettings>
</configuration>
```

<appSettings> სექციას აქვს მნიშვნელობები ფილიალის სახელი, ID (უნიკალური იდენტიფიკატორი) და მისამართი (პორტის ნომერი, რომელსაც აპლიკაცია იყენებს).

მოთხოვნის მისამართი განსაზღვრავს პორტის ნომერს, საითაც იქნება მოთხოვნები გაგზავნილი. შესაძლებელია სხვა პორტების გამოყენებაც, თუ ეს საჭიროა.

6. ბიზნესპროცესების განსაზღვრა

შემდეგი ბიჯი არის კლიენტის და სერვერის სამუშაო პროცესების განსაზღვრა. მათ შორის კომუნიკაცია ხორციელდება Send და Receive ქმედებათა გამოყენებით. ამ პროექტში ჩვენ გამოვიყენებთ Workflow-ის კოდირებას, ნაცვლად დიზაინერისა .xaml ფაილის გენერაციისათვის.

TrainingProcess პროექტის Solution Explorer-ში დავამატოთ კლასი: Add->Class, სახელით TrainingWF.cs და ჩავამატოთ სახელსივრცეები:

```
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
```

7. კლიენტი – მოთხოვნების გაგზავნა

პირველ რიგში უნდა განისაზღვროს სამუშაო პროცესი (workflow), რომლითაც მოთხოვნა გადაეცემა სხვა ფილიალს. TrainingWF კლასის კოდი შევცვალოთ 1.6 ლისტინგის მიხედვით.

```
//--- ლისტინგი_1.6 ------
public sealed class SendRequest : Activity
```

```
{
  // შესატან და გამოსატან არგუმენტთა განსაზღვრა
  public InArgument<string> Title { get; set; }
  public InArgument<string> Author { get; set; }
  public InArgument<string> ISBN { get; set; }
  public OutArgument<ReservationResponse> Response { get; set; }

  public SendRequest()
  {
    // ცვლადების განსაზღვრა ამ პროცესისთვის
    Variable<ReservationRequest> request =
                new Variable<ReservationRequest> { Name = "request" };
    Variable<string> requestAddress =
                    new Variable<string> { Name = "RequestAddress" };
  // გაგზავნის Send ქმედების განსაზღვრა
  Send submitRequest = new Send
   {
     ServiceContractName = "IlibraryReservation",
     EndpointAddress = new InArgument<Uri>
          (env => new Uri("http://localhost:" + requestAddress.Get(env) +
             "/LibraryReservation")),
     Endpoint = new Endpoint
      {  Binding = new BasicHttpBinding()  },
     OperationName = "RequestBook", Content = SendContent.Create
                (new InArgument<ReservationRequest>(request))
    };
    //  SendRequest პროცესის განსაზღვრა
  }
}
```

ამ ბიზნესპროცესს აქვს სამი შემავალი არგუმენტი: Sagani, Lector და Jgupi, რომლებიც განსაზღვრავს მოთხოვნილ სატრეინინგო კურსს. აგრეთვე აქვს გამომავალი არგუმენტი (Response), რომელიც არის საპასუხო შეტყობინება და იგი ადრე იყო განსაზღვრული TrainingResponse კლასით.

კონსტრუქტორი განსაზღვრავს აგრეთვე ზოგიერთ ლოკალურ ცვლადებს. მათი გადაცემა არ ხდება ბიზნესპროცესიდან (ან –ში), ისინი გამოიყენება შიგა სამუშაო პროცესების ქმედებებით. მოთხოვნილი ცვლადი შეიცავს გამომავალ შეტყობინებას, რომელიც არის TrainingRequest კლასი. RequestAddress ცვლადი შეინახავს პორტის ნომერს, რომელიც იმ ფილიალისაა, რომელიც ელოდება მოთხოვნის მიღებას.

შენიშვნა: კოდირებულ მუშა პროცესისთვის ყველა ქმედება შეიქმნა როგორც ჩადგმული ანონიმური კლასი. ეს კარგად მუშაობს ხშირ შემთხვევებში. Send ქმედება აქ შეიქმნა როგორც სახელმინიჭებული კლასი, ვინაიდან იგი საჭიროა მიმართვისათვის ReceiveReply ქმედებიდან. იგი არ განსხვავდება არგუმენტებისა და ცვლადებისგან, რომლებიც შევქმენით. ისინი იქმნება სახელდებული კლასების სახით, რომლებიც მოგვიანებით იქნება გამოყენებული.

8. გაგზავნის ქმედება

submitRequest (მოთხოვნის გაგზავნის) ეგზემპლარი განისაზღვრება როგორც Send ქმედება. იგი იყენებს WCF-ს, რათა გადასცეს შეტყობინება მითითებულ ბოლო წერტილში.

ინფორმაციის სამი ნაწილი გამოიყენება ბოლო წერტილის მისათითებლად:

- ServiceContractName მიეთითება როგორც ItrainingProcess;
- EndpointAddress მიეთითება როგორც URL ცვლადით (პორტის ნომერი);
- Binding მიეთითება BasicHttpBinding კლასით.

ამასთანავე არსებობს კიდევ რამდენიმე თვისება, რომლებიც უნდა განისაზღვროს. OperationName მიუთითებს სერვისის კონტრაქტის სპეციფიკურ მეთოდზე, რომელიც გამოძახებულ უნდა იქნას დანიშნულების ადგილას შეტყობინების მიღების დროს. Content თვისება შეინახავს მიმთითებელს შეტყობინებაზე (TrainingRequest კლასი), რომელიც უნდა გაიგზავნოს.

6.2.2. ლაბორატორიული სამუშაო N 2

მომხმარებლის ქმედების დაპროგრამება [1]

მიზანი: მომხმარებლის ქმედების (activity) შექმნის პროცესის შესწავლა. ეს ქმედება ააგებს შეტყობინებას მოთხოვნაზე ბიზნეს–პროცესით გადმოცემული არგუმენტების საფუძველზე.

/ჩვენ ვაგრძელებთ მუშაობას N1 ლაბორატორიაზე შექმნილ აპლიკაციის კოდთან /

TrainingProcess პროექტის Solution Explorer–ში დავამატოთ კლასი: Add->Class სახელით **CreateRequest.cs**, რომლის რეალიზაცია მოცემულია 1.7 ლისტინგში.

```
//——— ლისტინგი_1.7 ——————————
using System;
using System.Activities;
using System.Configuration;
namespace TrainingProcess
{
 // ეს მომხმარებლის ქმედება ქმნის TrainingRequest კლასს შესატანი
 // პარამეტრებით (Sagani, Lector და Jgupi). ეს უზრუნველყოფილია
 // გამომავალი პარამეტრის მოთხოვნაში. იგი ასევე აბრუნებს ქსელის
 // მისამართს ფილიალისთვის, რომელსაც უნდა გაეგზავნოს მოთხოვნა.
  public sealed class CreateRequest : CodeActivity
  {
    public InArgument<string> Sagani { get; set; }
    public InArgument<string> Lector { get; set; }
    public InArgument<string> Jgupi { get; set; }
    public OutArgument<TrainingRequest> Request { get; set; }
    public OutArgument<string> RequestAddress { get; set; }

    protected override void Execute(CodeActivityContext context)
      {
        // config ფაილის გახსნა და Request Address მიღება (get)
        Configuration config = ConfigurationManager
                .OpenExeConfiguration(ConfigurationUserLevel.None);
```

```
    AppSettingsSection app =
            (AppSettingsSection)config.GetSection("appSettings");
    //  TrainingRequest კლასის შექმნა და მისი შევსება შესატანი
     // არგუმენტებით
    TrainingRequest  r = new TrainingRequest
        (
  Sagani.Get(context),
        Lector.Get(context),
        Jgupi.Get(context),
        new Branch
          {
            BranchName = app.Settings["Branch Name"].Value,
            BranchID = new Guid(app.Settings["ID"].Value),
            Address = app.Settings["Address"].Value
          }
       );
      // მოთხოვნის მოთავსება OutArgument – ში
     Request.Set(context, r);
      // address–ის მოთავსება OutArgument –ში
     RequestAddress.Set(context, app.Settings["Request  Address"].Value);
} } }
```

Execute() მეთოდი პირველად ხსნის აპლიკაციის კონფიგ–ფაილს ფილიალის დეტალების დასადგენად, რომელსაც ესაჭიროება მოთხოვნის ფორმატირება. შემდეგ იგი ქმნის TrainingRequest კლასს ერთ-ერთი ჩვენ მიერ უზრუნველყოფილი კონსტრუქტორით. Branch კლასი, რომელიც არის კონსტრუქტორის ერთ-ერთი პარამეტრი, შექმნილია როგორც ანონიმური კლასი BranchName, BranchID და Address თვისებების მითითებით. შემდეგ TrainingRequest კლასი შეინახავს მოთხოვნის გამომავალ პარამეტრში.

კონფიგურაციის ფაილში Request Address შეიცავს ფილიალის მისამართს (პორტის ნომერს), რომელმაც უნდა მიიღოს მოთხოვნა. იგი შეინახება RequestAddress გამომავალ არგუმენტში, ვინაიდან ის დასჭირდება სამუშაო პროცესს.

დავუბრუნდეთ TrainingWF.cs ფაილს. დავამატოთ ბიზნეს–პროცესის განსაზღვრება 1.8 ლისტინგის შესაბამისად. ეს უნდა ჩაჯდეს იქ სადაც მითითებულია (// SendRequest სამუშაო პროცესის განსაზღვრა).

```
// ——— ლისტინგი_1.8 ———
// Define the SendRequest workflow
this.Implementation = () => new Sequence
{
  DisplayName = „SendRequest“, Variables = { request, requestAddress},
Activities =
    {
    new CreateRequest
     {
      Sagani = new InArgument<string>(env => Sagani.Get(env)),
      Lector = new InArgument<string>(env => Lector.Get(env)),
```

```
    Jgupi = new InArgument<string>(env => Jgupi.Get(env)),
    Request = new OutArgument<TrainingRequest>
        (env => request.Get(env)),
    RequestAddress = new OutArgument<string>
        (env => requestAddress.Get(env))
    },
  new CorrelationScope
    {
      Body = new Sequence
        {
         Activities = {  submitRequest,
          new WriteLine
            {
              Text = new InArgument<string>
                (env => "Request sent; waiting for response"),
            },
       new ReceiveReply
         {
          Request = submitRequest,
          Content = ReceiveContent.Create
              (new OutArgument<ReservationResponse>
                 (env => Response.Get(env)))
         }
        }
      }
    },
     new WriteLine
      {
       Text = new InArgument<string>
          (env => "Response received from " +
              Response.Get(env).Provider.BranchName),
      },
    }
};
```

ქმედების კლასის Implementation თვისება (მითითებით როგორც this. Implementation) შეიცავს შვილ ქმედებას. ამ შემთხვევაში იგი განისაზღვრება როგორც Sequence ქმედება, რომელიც შედგება შვილ ქმედებათა ერთობლიობისგან. ამ ქმედებათა მიერ გამოყენებული ცვლადები უნდა იყოს გამოცხადებული. ესაა მოთხოვნა და requestAddress ცვლადები, რომლებიც განისაზღვრა კონსტრუქტორში.

პირველი ქმედება არის მომხმარებლის CreateRequest ქმედება, რომელიც ახლახანს ავაგეთ. მივაქციოთ ყურადღება, რომ როგორც ჩვენ მივუთითეთ თვისება, ეს იცის Intellisense–მ (შესატანი და გამოსატანი არგუმენტები, რომლებიც განვსაზღრეთ ჩვენს კლასში). Sagani, Lector და Jgupi არის სამუშაო პროცესის შემავალი არგუმენტები. Request და RequestAddress გამომავალი არგუმენტები ინახება სამუშაო პროცესის ცვლადებში.

CorrelationScope ქმედება ამატებს შემდეგს (next), რომელიც შედგება ქმედებათა მიმდევრობისგან. კერძოდ, ის შეიცავს Send და ReceiveReply ქმედებებს, რომელთა მოთავსებით CorrelationScope ქმედებაში შესაძლებელი ხდება შემოსული მოთხოვნისა და საპასუხო შეტყობინების კორელაციის განსაზღვრა მოცემული ბიზნესპროცესისთვის.

WriteLine ქმედება ემატება Send ქმედების შემდეგ, რათა მიეთითოს, რომ მოთხოვნა იქნა გაგზავნილი.

1. პასუხის მიღების ქმედება

ReceiveReply ქმედება ასოცირდება გაგზავნის Send ქმედებასთან. იგი ელოდება პასუხს შეტყობინებაზე, რომელიც გაიგზავნა Send ქმედების მიერ. ამის რეალიზაციის მიზნით Request–ის (მოთხოვნის) თვისებას აქვს Send ქმედების სახელმინიჭებული ეგზემპლარის მნიშვნელობა (submitRequest).

თვისების შინაარსი განსაზღვრავს თუ სად ინახება საპასუხო შეტყობინება (TrainingResponse კლასი). ამით ყენდება სამუშაო ნაკადის Response გამომავალი არგუმენტი. იგი მისაწვდომი იქნება ჰოსტ–აპლიკაციისთვის როცა ვორკფლოვ–პროცესი დასრულდება.

2. სერვერი – მოთხოვნის დამუშავება

განვსაზღვროთ სამუშაო პროცესი, რომელიც სრულდება სრევერზე ფილიალიდან მიღებული მოთხოვნის დასამუშავებლად.

TrainingWF.cs ფაილში დავამატოთ კლასის განსაზღვრა:

```
// ––– ლისტინგი_1.9 ––––––––––––––––––––––––––
public sealed class ProcessRequest : Activity
{
   public ProcessRequest()
   { // ცვლადების განსაზღვრა ამ workflow–ისთვის
      Variable<TrainingRequest> request =
                  new Variable<TrainingRequest> { Name = “request” };
      Variable<TrainingResponse> response =
                new Variable<TrainingResponse> { Name = “response” };
Variable<bool> reserved = new Variable<bool> { Name = “reserved” };
Variable<CorrelationHandle> requestHandle =
       new Variable<CorrelationHandle> { Name = “RequestHandle” };
      // Receive ქმედების შექმნა
      Receive receiveRequest = new Receive
          {
              ServiceContractName = “ItrainingProcess”,
              OperationName = “RequestTraining”,
              CanCreateInstance = true,
              Content = ReceiveContent.Create
                    (new OutArgument<TrainingRequest>(request)),
              CorrelatesWith = requestHandle
          };
       //  ProcessRequest workflow–ის განსაზღვრა
   }
}
```

ამ სამუშაო პროცესს არ აქვს შემავალი და გამომავალი არგუმენტები. აქვს ოთხი ცვლადი, რომლებიც განსაზღვრულია. მოთხოვნის (request) ცვლადი ინახავს შემავალ შეტყობინებებს (TrainingRequest კლასი), ხოლო პასუხის (response) ცვლადი ინახავს გამომავალ პასუხებს (TrainingResponse კლასი). დარეზერვებული ცვლადი მიუთითებს შეიძლება თუ არა კურსის საგანი (sagani) იყოს დაჯავშნული, თუ არა. requestHandle გამოიყენება შემოსული მოთხოვნის და პასუხის კორელაციისთვის.

3. მოთხოვნის მიღების ქმედება

Receive ქმედების სახელმინიჭებული ეგზემპლარი (receiveRequest – მოთხოვნის მიღება) განსაზღვრულია. არაა საჭირო მიერთების ან მისამართის მითითება WCF შეტყობინების მიღების ბოლოს. მაგრამ უნდა განისაზღვროს სერვისის კონტრაქტი. ServiceContractName მიუთითებს, რომ ItrainingProcess სერვისის კონტრაქტი უნდა იქნას გამოყენებული და OperationName თვისება განსაზღვრავს RequestTraining() მეთოდს.

CanCreateInstance დაყენებულია true -ში, ვინაიდან როცა ეს ქმედება სრულდება, იგი შექმნის პროცესის ახალ ეგზემპლარს. იგი მოითხოვს, რომ ეს ქმედება იყოს პირველი სამუშაო პროცესში. Content თვისება უნდა შეიცავდეს შემავალ შეტყობინებას და კონფიგურირდება ისე, რომ შეინახოს იგი მოთხოვნის ცვლადში. CorrelatesWith თვისება იყენებს requestHandle ცვლადს.

4. მომხმარებლის ქმედება – პასუხის შექმნა

სანამ განვსაზღვრავთ სამუშაო პროცესის ქმედებებს, საჭიროა მომხმარებლის ქმედება TrainingResponse კლასის შექმნისათვის. TrainingProcess პროექტის Solution Explorer–დან დავამატოთ ახალი კლასი სახელით CreateResponse.cs, რომლის რეალიზაცია მოცემულია 1.10 ლისტინგზე.

```
// —— ლისტინგი_1.10 ——————
using System;
using System.Activities;
using System.Configuration;
namespace TrainingProcess
{
// ეს custom ქმედება ქმნის ReservationResponse კლასს. საწყისი
// მოთხოვნა უზრუნველყოფილია როგორც InArgument, ასევე
// ლოგიკური (boolean) მითითებით, რომ მოთხოვნა დაკმაყოფილდა
// თუ არა. კლასი უზრუნველყოფილია Response–ში OutArgument-ით
public sealed class CreateResponse : CodeActivity
{
  public InArgument<TrainingRequest> Request { get; set; }
  public InArgument<bool> Reserved { get; set; }
  public OutArgument<TrainingResponse> Response { get; set; }

  protected override void Execute(CodeActivityContext context)
   {
    // config ფაილის გახსნა
    Configuration config = ConfigurationManager
          .OpenExeConfiguration(ConfigurationUserLevel.None);
```

```
        AppSettingsSection app =
              (AppSettingsSection)config.GetSection("appSettings");
        // TrainingResponse კლასის შექმნა და შევსება
        TrainingResponse r = new TrainingResponse
            (   Request.Get(context),
                Reserved.Get(context),
                new Branch
                   {
                     BranchName = app.Settings["Branch Name"].Value,
                     BranchID = new Guid(app.Settings["ID"].Value),
                     Address = app.Settings["Address"].Value
                   }
               );
              // პასუხის შენახვა  OutArgument–ში
                 Response.Set(context, r);
               }
             }
          }
```

CreateResponse ქმედება ძალზე ჰგავს CreateRequest-ეს. ჯერ იგი ხსნის აპლიკაციის კონფიგ–ფაილს, რათა მიიღოს ფილიალის შესახებ დეტალები. შემდეგ TrainingResponse კლასი იქმნება ერთ-ერთი მოცემული კონსტრუქტორით და შეინახება Response გამომავალ არგუმენტში.

დავბრუნდეთ TrainingWF.cs კლასში შევიტანოთ სამუშაო პროცესის განსაზღვრება, როგორც 1.11 ლისტინგშია (იგი უნდა ჩაემატოს //---სამუშაო პროცესის განსაზღვრა ProcessRequest--- ში).

```
// ——— ლისტინგი_1.11 ——————————
// ProcessRequest workflow-ის განსაზღვრა
this.Implementation = () => new Sequence
{   DisplayName = „ProcessRequest“,
    Variables = { request, response, reserved, requestHandle },
    Activities =
    { receiveRequest,
      new WriteLine
       {
       Text = new InArgument<string>(env => "Got request from: " +
            request.Get(env).Requester.BranchName),
       },
       new WriteLine
       {     Text = new InArgument<string>(env => "Requesting: " +
             request.Get(env).Title),
        },
       new Assign
       {    To = new OutArgument<Boolean>(reserved),
            Value = new InArgument<Boolean>(env => true)
       },
       new Delay
```

```
    {
        Duration = TimeSpan.FromSeconds(2)
    },
    new CreateResponse
    {   Request = new InArgument<ReservationRequest>(env =>
            request.Get(env)),
        Response = new OutArgument<ReservationResponse>
            (env => response.Get(env)),
        Reserved = new InArgument<bool>(env => reserved.Get(env)),
    },
    new WriteLine
    {   Text = new InArgument<string>(env => "Sending response to: " +
            request.Get(env).Requester.BranchName),
    },
    new SendReply
    {   Request = receiveRequest,
        Content = SendContent.Create
            (new InArgument<ReservationResponse>(response))
    }
    }
};
```

ეს ოთხი ცვლადი, რომლებიც კონსტრუქტორშია განსაზღვრული, გამოცხადებულია სამუშაო პროცესის კოდის ტანში. Receive ქმედება (receiveRequest) არის პირველი ქმედება მუშა პროცესში. იგი, ორ WriteLine ქმედებასთან თანხლებით: პირველს ეკრანზე გამოაქვს ფილიალის სახელი, რომელმაც მოთხოვნა გამოაგზავნა, და მეორე გვიჩვენებს საგნის დასახელებას (sagani), რომელიც მოითხოვება.

Assign ქმედება უბრალოდ აყენებს დარეზერვებულ ცვლადებს true-ში. ამ მაგალითში ჩვენ დავუშვით, რომ საგნის დასახელება არის მიწვდომადი. Delay ქმედება აყოვნებს სამუშაო პროცესს 2 წამით (დამუშავების პროცესის იმიტაცია). შემდეგ მომხმარებლის CreateResponse ქმედება სრულდება TrainingResponse კლასის შექმნის მიზნით, რომელიც შენახულ იქნება response ცვლადში. ბოლო WriteLine ქმედება მიუთითებს, რომ პასუხი გაიგზავნა.

5. SendReply ქმედება

SendReply ქმედება ასოცირდება (კავშირშია) Receive ქმედებასთან. ეს ხორციელდება Request თვისების მითითებით როგორც მაჩვენებლისა Receive ქმედებაზე (receiveRequest). თვისების შინაარსი განსაზღვრავს შეტყობინებას, რომელიც იქნება გაგზავნილი უკან საპასუხოდ. ესაა პასუხის ცვლადის მნიშვნელობა.

ჩვენი სამუშაო პროცესი დასრულდა. საფინალო რეალიზაცია TrainingWF.cs ფაილისთვის მოცემულია 1.12 ლისტინგში.

```
// ── ლისტინგი_1.12 ──────────
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
```

```
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;

namespace TrainingProcess
{
  public sealed class SendRequest : Activity
  {
   // შესატან და გამოსატან არგუმენტთა განსაზღვრა
   public InArgument<string> Sagani { get; set; }
   public InArgument<string> Lector { get; set; }
   public InArgument<string> Jgupi { get; set; }
      public OutArgument<TrainingResponse> Response {get;set;}

  public SendRequest()
  {
  // ცვლადების განსაზღვრა ამ პროცესისთვის
  Variable<TrainingRequest> request = new
                Variable<TrainingRequest> { Name = "request" };
  Variable<string> requestAddress = new Variable<string> {
                Name = "RequestAddress" };

  // გაგზავნის Send ქმედების განსაზღვრა
   Send submitRequest = new Send
    {
     ServiceContractName = "ItrainingProcess",
     EndpointAddress = new InArgument<Uri>
         (env => new Uri("http://localhost:" +
                requestAddress.Get(env) + "/TrainingProcess")),
     Endpoint = new Endpoint
       {
          Binding = new BasicHttpBinding()
       },
     OperationName = "RequestTraining",
     Content = SendContent.Create(new
                        InArgument<TrainingRequest>(request))
    };
    //  SendRequest პროცესის განსაზღვრა
    // ——— ლისტინგი_1.8 ———
    this.Implementation = () => new Sequence
     {
      DisplayName = "SendRequest",
     Variables = { request, requestAddress },
     Activities = { new CreateRequest {
        Sagani = new InArgument<string>(env => Sagani.Get(env)),
         Lector = new InArgument<string>(env =>
                                Lector.Get(env)),
```

```
            Jgupi = new InArgument<string>(env =>
                                        Jgupi.Get(env)),
          Request = new OutArgument<TrainingRequest> (env =>
                                        request.Get(env)),
             RequestAddress = new OutArgument<string>(env =>
                                    requestAddress.Get(env))
        },
       new CorrelationScope
       {
        Body = new Sequence
        {
          Activities = { submitRequest,  new WriteLine
          {
            Text = new InArgument<string>(env =>
                            "Request sent; waiting for response"),
          },
       new ReceiveReply
        {
         Request = submitRequest,
          Content = ReceiveContent.Create(new
                                    OutArgument<TrainingResponse>
              (env => Response.Get(env)))
        }
       }
       }
    },
    new WriteLine
     {
      Text = new InArgument<string>
      (env => "Response received from " +
                      Response.Get(env).Provider.BranchName),
        },
       }
      };
     }
   }
public sealed class ProcessRequest : Activity
 {
   public ProcessRequest()
   { // ცვლადების განსაზღვრა ამ workflow–ისთვის
    Variable<TrainingRequest> request =
      new Variable<TrainingRequest> { Name = "request" };
   Variable<TrainingResponse> response =
     new Variable<TrainingResponse> { Name = "response"};
   Variable<bool> reserved =
       new Variable<bool> { Name = "reserved" };
   Variable<CorrelationHandle> requestHandle =
```

```
        new Variable<CorrelationHandle> { Name =
            “RequestHandle” };

// Receive ქმედების შექმნა
Receive receiveRequest = new Receive
 {
   ServiceContractName = “ItrainingProcess”,
   OperationName = “RequestTraining”,
   CanCreateInstance = true,
   Content = ReceiveContent.Create
          (new OutArgument<TrainingRequest>(request)),
   CorrelatesWith = requestHandle
 };

 //  ProcessRequest workflow–ის განსაზღვრა
this.Implementation = () => new Sequence
{
 DisplayName = “ProcessRequest”,
 Variables = { request, response, reserved, requestHandle },
 Activities =
 {
    receiveRequest,
    new WriteLine
     {
Text = new InArgument<string>(env=> “Got request from: “ +
                request.Get(env).Requester.BranchName),
     },
    new WriteLine
     {
    Text = new InArgument<string>(env => “Requesting: “ +
                 request.Get(env).Sagani),
     },
  new Assign
  {
     To = new OutArgument<Boolean>(reserved),
     Value = new InArgument<Boolean>(env => true)
  },
  new Delay
  {
     Duration = TimeSpan.FromSeconds(2)
  },
  new CreateResponse
  {
     Request = new InArgument<TrainingRequest>(env =>
                request.Get(env)),
     Response = new OutArgument<TrainingResponse>
                (env => response.Get(env)),
```

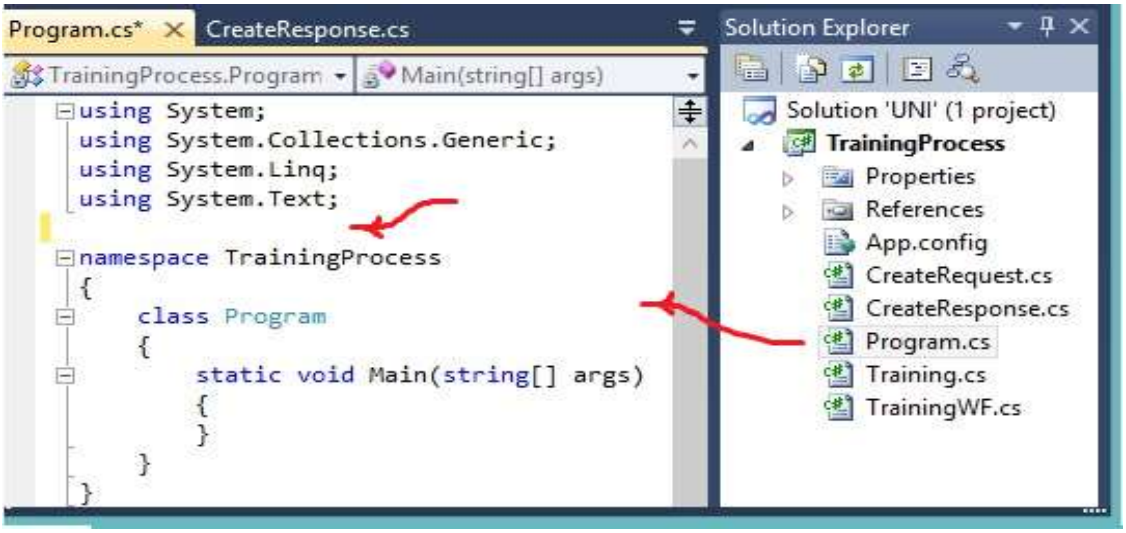
```
    Reserved = new InArgument<bool>(env =>
            reserved.Get(env)),
    },
  new WriteLine
    {
Text=new InArgument<string>(env=>"Sending response to: "+
            request.Get(env).Requester.BranchName),
    },
  new SendReply
    {
      Request = receiveRequest,
      Content = SendContent.Create
          (new InArgument<TrainingResponse>(response))
            }
          }
        };
      }
    }
}
```

6.2.3. ლაბორატორიული სამუშაო N 3

Host-აპლიკაციის რეალიზაცია

მიზანი: ასაგები სისტემის ჰოსტ–აპლიკაციის რეალიზაციის საფუძვლების შესწავლა.

ბოლო ბიჯი პროექტის ასაგებად არის ჰოსტ–აპლიკაციის რეალიზაცია. უნდა გამოვიყენოთ კონსოლის აპლიკაცია (Program.cs), რომელიც გენერირებულ იქნა შაბლონის (template) სახით (ნახ.19).



ნახ.19

აპლიკაცია ასრულებს მოთხოვნის ინიციალიზებას და დამუშავებას, ამიტომ საჭირო იქნება ორივესთვის ლოგიკის დაწერა. ჯერ უნდა აეწყოს აპლიკაცია მოთხოვნების მისაღებად და დასამუშავებლად, შემდეგ კი უნდა მოხდეს ახალი მოთხოვნის ინიცირება და გაგზავნა სხვა აპლიკაციისთვის.

დავამატოთ Program.cs ფაილში რამდენიმე სახელსივრცე:

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;
```

1. ServiceHost კლასი და WorkflowServiceHost –ის შექმნა

შემავალი მოთხოვნების მისაღებად გამოიყენება ServiceHost კლასი, რომელიც WCF ტექნოლოგიაშია. WF 4.0 უზრუნველყოფს WorkflowServiceHost კლასს, რომელიც ახდენს ServiceHost-ის რეალიზაციას, ოღონდ აინიცირებს სამუშაო პროცესს, როცა შეტყობინება მიღებულია.

შვიტანოთ 1.13 ლისტინგის კოდი როგორც main() ფუნქციის რეალიზაცია Program კლასისთვის.

```
// --ლისტინგი 1-13. ნაწილობრივი რეალიზაცია  main() ფუნქციის –
//  config ფაილის გახსნა და  ფილიალის სახელის (name) და
// ქსელის მისამართის (adress) მიღება
Configuration config = ConfigurationManager
       .OpenExeConfiguration(ConfigurationUserLevel.None);
AppSettingsSection app = (AppSettingsSection)config.GetSection(“appSettings”);

string adr = app.Settings[“Address”].Value;
Console.WriteLine(app.Settings[“Branch Name”].Value);

// სერვისის შექმნა შემოსული მოთხოვნების დასამუშავებლად
WorkflowService service = new WorkflowService
 {
   Name = „LibraryReservation“,
   Body = new ProcessRequest(),
   Endpoints =
     {
       new Endpoint
       {
           ServiceContractName = “IlibraryReservation”,
           AddressUri = new Uri(“http://localhost:” + adr +
               „/LibraryReservation“),
           Binding = new BasicHttpBinding(),
       }
   }
};
//  WorkflowServiceHost –ის შექმნა შემოსული მოთხოვნების მისაღებად
System.ServiceModel.Activities.WorkflowServiceHost wsh =
       new System.ServiceModel.Activities.WorkflowServiceHost(service);
wsh.Open();
```

ეს კოდი ჯერ ხსნის აპლიკაციის კონფიგურაციის ფაილს და იღებს მისამართის პარამეტრებს. ის განსაზღვრავს პორტის ნომერს, რომლითაც აპლიკაცია იღებს შემავალ მოთხოვნას. აგრეთვე ღებულობს ფილიალის სახელს, რომელიც ეკრანზე გამოიტანება. ვინაიდან ჩვენ საქმე გვაქვს რამდენიმე აპლიკაციასთან, ეს მექანიზმი საშუალებას მოგვცემს თვალყური ვადევნოთ თუ რომელი რომელია.

2. სერვისი - WorkflowService კლასი

1.13 ლისტინგის ბოლოში ნაჩვენებია WorkflowService კლასის შქმნის კოდი. Body თვისებისთვის ის იყენებს ProcessRequest კლასის ახალ ეგზემპლარს, რომელიც განსაზღვრავს პროცესს შემავალი მოთხოვნების დასამუშავებლად.

3. ბოლო წერტილი

სერვისის კლასი განსაზღვრავს აგრეთვე ბოლო წერტილს (Endpoint), კონტრაქტის ItrainingProcess სერვისის გამოყენებით, URL–ით, რომელიც შეიცავს პორტის ნომრის ცვლადს და BasicHttpBinding კლასს. დასასრულ, WorkflowServiceHost კლასი კონკრეტიზირდება განსაზღვრული სერვისკლასის გამოყენებით. შემდეგ ის იხსნება Open() მეთოდის გამოძახებით. ამ მომენტში აპლიკაცია უთვალთვალებს შემავალ შეტყობინებებს. როცა ის მიღებულ იქნება, ProcessRequest მუშა პროცესის ეგზემპლარი ამუშავდება მოთხოვნის დასამუშავებლად.

4. სამუშაო პროცესის გამომძახებელი [WorkflowInvoker]

ახლა საჭიროა კოდის დამატება მოთხოვნის ინიციალიზებისათვის. შევიტანოთ 1.14 ლისტინგის კოდი, ოღონდაც wsh.Open() მეთოდის გამოძახების შემდეგ.

```
// ლისტინგი 1-14.  main() ფუნქციის რეალიზების დარჩენილი ნაწილი – – –
Console.WriteLine
  (“Waiting for requests, press ENTER to send a request.”);
Console.ReadLine();

//ლექსიკონის შექმნა შემავალი არგუმენტებით სამუშაო პროცესისთვის-
Idictionary<string, object> input =
            new Dictionary<string, object>
    {
      { “Sagani” , “Distributed DBS Management” },
      { “Lector”, “Gia Surguladze” },
      { “Jgupi”, “151100108350” }
    };

//  SendRequest სამუშაო პროცესის გამოძახება
Idictionary<string, object> output =
         WorkflowInvoker.Invoke(new SendRequest(), input);
TrainingResponse resp =
        (TrainingResponse)output[“Response”];

// პასუხის ეკრანზე გამოტანა
Console.WriteLine(“Response received from the {0} branch”,
               resp.Provider.BranchName);
Console.WriteLine();
Console.WriteLine(“Press ENTER to exit”);
Console.ReadLine();
```

```
// WorkflowServiceHost-ის დახურვა
wsh.Close();
```

ეს კოდი იცდის, სანამ მომხმარებელი არ დააჭერს Enter ღილაკს, რომელიც მოგვცემს დროს, რათა მივიღოთ რამდენიმე მუშა კოპიო და თვალყური ვადევნოთ შემოსულ შეტყობინებებს.

ჯერ იქმნება ლექსიკონი შემავალი არგუმენტებისთვის. შემდეგ ის იყენებს WorkflowInvoker კლასის Invoke() მეთოდს, რათა დაწყებულ იქნას SendRequest სამუშაო პროცესის ახალი ეგზემპლარი.

Response პასუხის გამომავალი არგუმენტები ამოიღება ლექსიკონიდან, რომელიც დაბრუნდება უკან როცა მუშა პროცესი დასრულდება. დასასრულ, WorkflowServiceHost დაიხურა მუშა პროცესის დასრულებამდე.

ქვემოთ მოცემულია მთლიანი Program.cs კოდის ლისტინგი:

```
// – – – ლისტინგი_1.15 – – – Program.cs  კოდი – – –
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.ServiceModel;
    using System.ServiceModel.Activities;
    using System.ServiceModel.Activities.Description;
    using System.ServiceModel.Description;
    using System.Activities;
    using System.Xml.Linq;
   using System.Configuration;
//  config ფაილის გახსნა და  ფილიალის სახელის (name) და
// ქსელის მისამართის (adress) მიღება
Configuration config = ConfigurationManager
      .OpenExeConfiguration(ConfigurationUserLevel.None);
AppSettingsSection app =
       (AppSettingsSection)config.GetSection(“appSettings”);
string adr = app.Settings[“Address”].Value;
Console.WriteLine(app.Settings[“Branch Name”].Value);
// სერვისის შექმნა შემოსული მოთხოვნების დასამუშავებლად
  WorkflowService service = new WorkflowService
    {
      Name = „TrainingProcess“,
      Body = new ProcessRequest(),
      Endpoints =
       {
       new Endpoint
        {
         ServiceContractName=”ItrainingProcess”,
           AddressUri = new Uri(“http://localhost:” + adr +
                          “/TrainingProcess”),
            Binding = new BasicHttpBinding(),
        }
       }
```

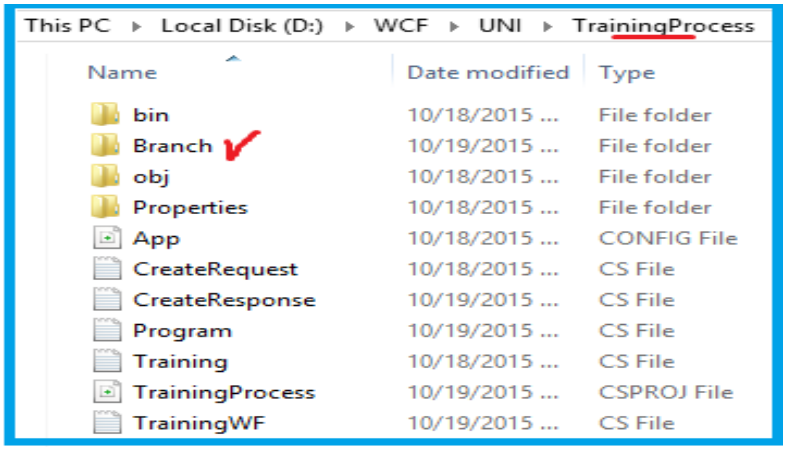
```
        };
// WorkflowServiceHost –ის შექმნა შემოსული მოთხოვნების მისაღებად
System.ServiceModel.Activities.WorkflowServiceHost wsh =
      new System.ServiceModel.Activities.WorkflowServiceHost(service);

wsh.Open();

Console.WriteLine
        ("Waiting for requests, press ENTER to send a request.");
Console.ReadLine();
// ლექსიკონის შექმნა შემავალი არგუმენტებით სამუშაო პროცესისთვის
 Idictionary<string, object> input = new Dictionary<string, object>
 {
    { "Sagani", "Distributed DBS Management" },
    { "Lector", "Gia Surguladze" },
    { "Jgupi", "151100108350" }
 };
 // SendRequest სამუშაო პროცესის გამოძახება
Idictionary<string, object> output =
           WorkflowInvoker.Invoke(new SendRequest(), input);
TrainingResponse resp= (TrainingResponse)output["Response"];

 // პასუხის ეკრანზე გამოტანა
 Console.WriteLine("Response received from the {0} branch",
                resp.Provider.BranchName);
 Console.WriteLine();
 Console.WriteLine("Press ENTER to exit");
 Console.ReadLine();
   // WorkflowServiceHost დახურვა
  wsh.Close();
 }
 }
}
```

5. აპლიკაციათა ამუშავება

F6-ით ავამუშავოთ აპლიკაცია და გავმართოთ კოდი. ჩვენ შგვიძლია აპლიკაციის ორი ეგზემპლარის ამუშავება და მათ დასჭირდება განსხვავებული კონფიგურაციის ფაილები. ამიტომაც შეიძლება მათთვის შერჩეულ იქნას სხვადასხვა პორტის ნომრები.

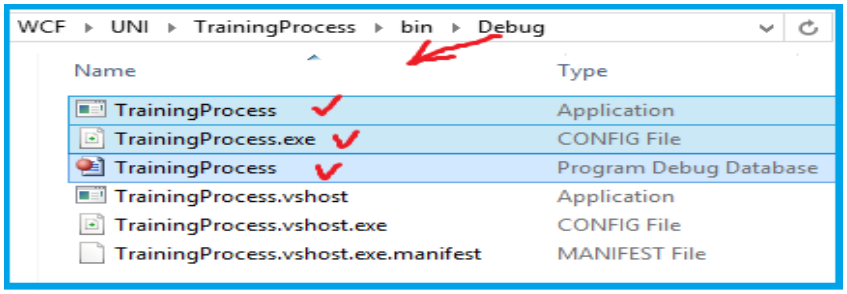
6. ტრეინინგ-ცენტრის ფილიალის კონფიგურირება

გავხსნათ Windows Explorer და ჩვენი პროექტის TrainingProcess ფოლდერში ჩავამატოთ ახალი ქვეკატალოგი Branch (ფილიალი) – (იხ. ნახ.20).



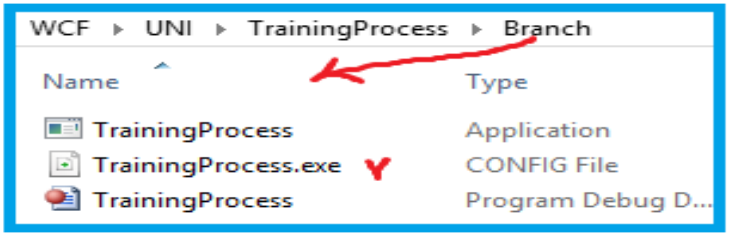
ნახ.20

Debag-კატალოგში მოვნიშნოთ სამი ფაილი (ნახ.21) და კოპირებით გადავიტანოთ Branch ფოლდერში.



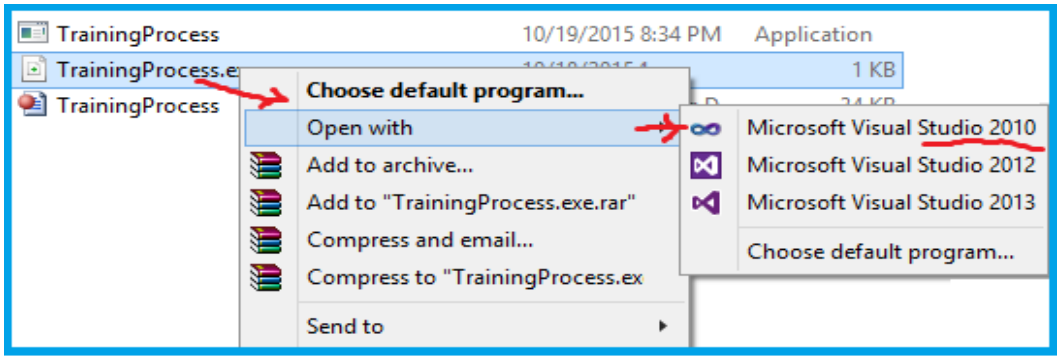
ნახ.21

კოპირების შემდეგ Branch–ში მივიღებთ:



ნახ.22

გავხსნათ TrainingProcess.exe.config ფაილი რომელიმე ტექსტურ რედაქტორში, მაუსის მარჯვენა ღილაკით Open with… და მაგალითად, Visual Studio .NET – ში (ნახ.23).



ნახ.23

```
<!– ლისტინგი_1.16 ——— –>
<?xml version=”1.0” encoding=”utf-8” ?>
<configuration>
 <appSettings>
   <add key=”Branch Name” value=”TrainingCenter”/>
   <add key=”ID” value=”{43E6DADD-4751-4056-8BB7-
                                          7459B5C361AB}”/>
    <add key=”Address” value=”8000”/>
    <add key=”Request Address” value=”8730”/>
  </appSettings>

</configuration>
```

შევცვალოთ კონფიგურაციის ფაილის ტექსტი შემდეგი კოდის 1.17_ლისტინგით.

```
<!– ლისტინგი_1.17 ——— –>
<?xml version=”1.0” encoding=”utf-8” ?>
<configuration>
  <appSettings>
    <add key=”Branch Name” value=”TrainingBranch_GTU”/>
    <add key=”ID” value=”{43E6DADD-4751-4056-8BB7-
                                          7459B5C361AB}”/>
    <add key=”Address” value=”8730”/>
    <add key=”Request Address” value=”8000”/>
  </appSettings>

</configuration>
```

დავაკვირდეთ, რომ პორტის ნომრები მისამართისა და მოთხოვნისთვის წესრიგშია (ტრაინინგ-ცენტრის და ფილიალის პორტის ნომრები განსხვავებულია). ავამუშავოთ .exe ფაილი. კონსოლზე მივიღებთ ასეთ ტექსტს:

```
TrainingBranch_GTU
Waiting for requests, press ENTER to send a request.
```

7. აპლიკაციათა მუშაობის მოსალოდნელი შედეგები

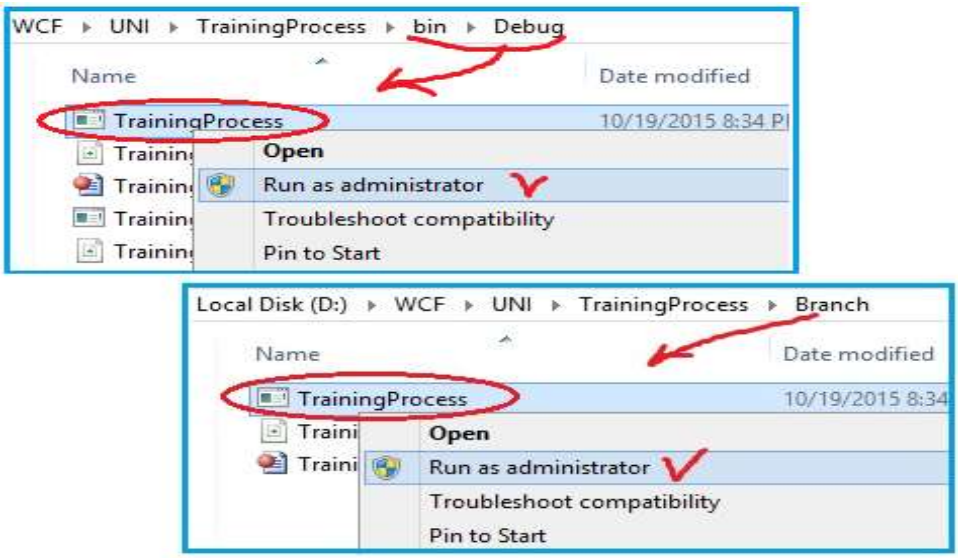
Visual Studio-დან F5–ით ავამუშავოთ აპლიკაცია. ამ დროს უნდა გაიხსნას მეორე კონსოლის ფანჯარა ასეთი ტექსტით:

```
TrainingCenter
Waiting for requests, press ENTER to send a request.
```

შესაძლებელია ფანჯრების გადაადგილება ისე, რომ ორივე ჩანდეს. ავირჩიოთ ერთ-ერთი და დავაჭიროთ Enter-ს. რამდენიმე წამის შემდეგ უნდა გამოჩნდეს შედეგები.

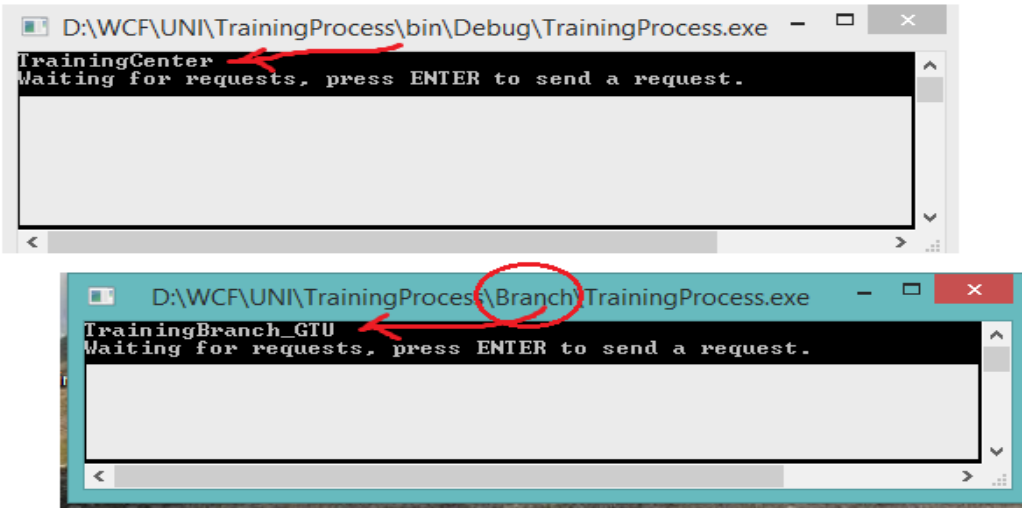
თუ შედეგები არ მიიღება და პროგრამა ავარიულად ჩერდება, შესაძლებელია მოისინჯოს ორივე აპლიკაციის (TrainingCenter და Branch) ამუშავება ადმინისტრატორის უფლებით.

მაგალითად, 24-ე ნახაზზე ნაჩვენებია ტრეინინგ-ცენტრის და მისი ფილიალის მხრიდან, შესაბამისად, bin->Debag და Branch-კატალოგებში ადმინისტრატორის უფლებით ორივე აპლიკაციის ამოქმედება.



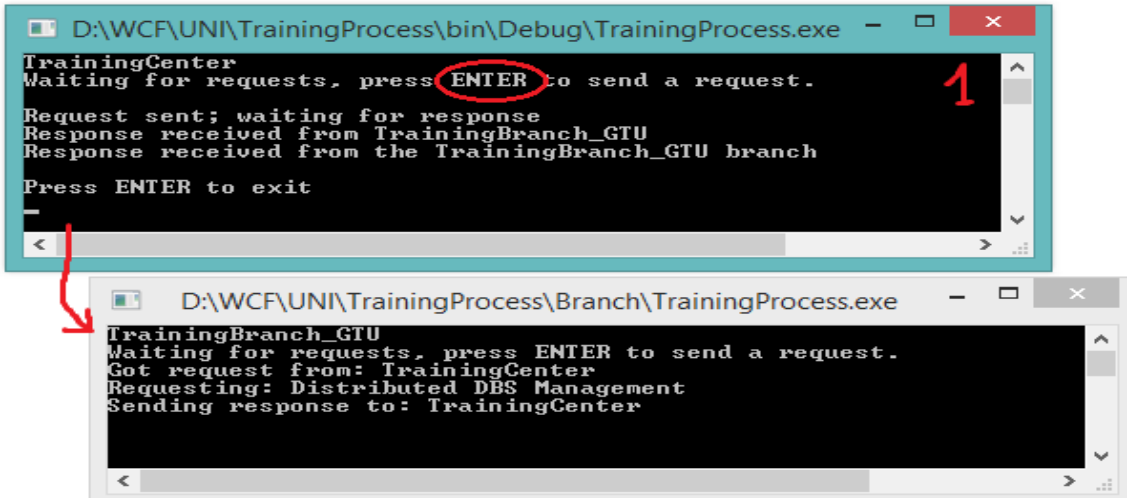
ნახ.24. ადმინისტრატორის უფლების გამოყენება

25-ა ნახაზზე ნაჩვენებია ორივე აპლიკაციის საწყისი მდგომარეობა. პირველ სტრიქონში ჩანს ფანჯრის მიკუთვნება ცენტრზე ან ფილიალზე.



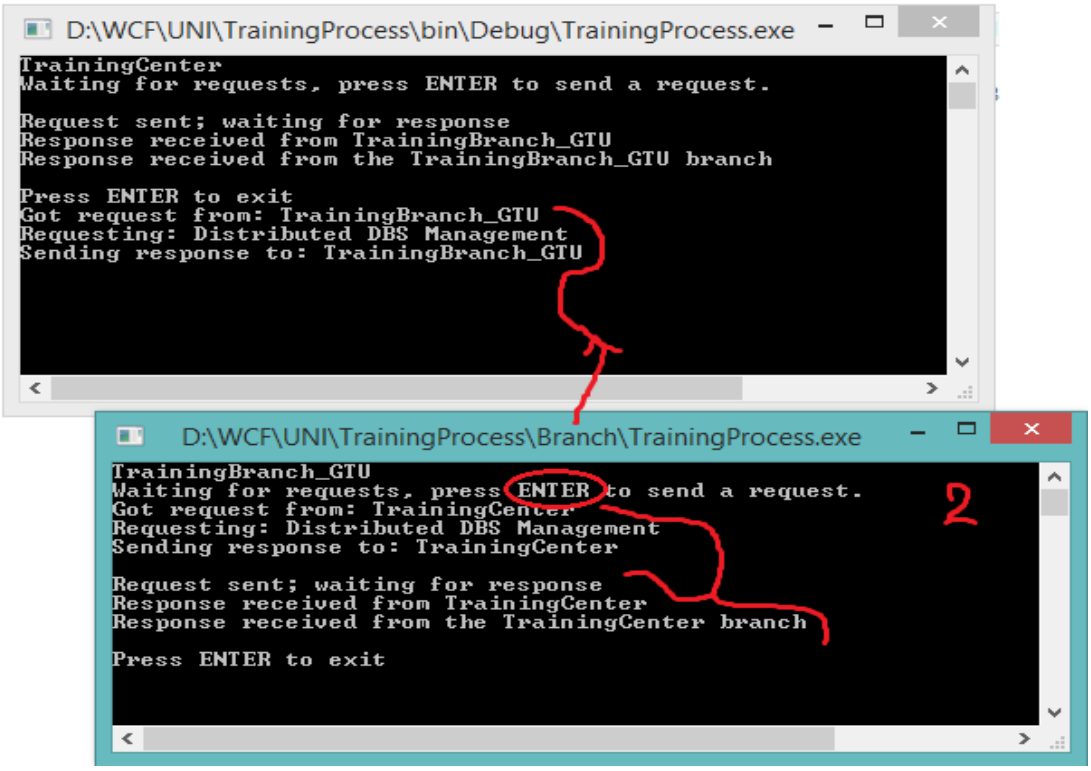
ნახ.25-ა

პირველ ფანჯარაში Enter–ღილაკის დაჭერით მივიღებთ ასეთ შედეგს ტექსტებით:



ნახ.25-ბ. აქტიურია 1-ფანჯარა (TrainingCenter)

ახლა მეორე ფანჯარაში ვაჭერთ Enter–ღილაკს, რომლის შედეგები ასეთია:



ნახ.25-გ. აქტიურია 2-ფანჯარა (TrainingBranch_GTU)

როგორც ვხედავთ, განხორციელდა ინფორმაციის გაცვლა ცენტრსა და მის ფილიალს შორის.

Enter–ღილაკით შეიძლება დაიხუროს აპლიკაციები.

7. საკურსო პროექტი

7.1. სამუშაოს ეტაპები სილაბუსის შესაბამისად

1. VisualStudio.NET 2019 სამუშაო გარემოში საილუსტრაციო საპრობლემო სფეროს პროგრამული აპლიკაციის სადემონსტრაციო მაგალითების განხილვა. პროგრამული უზრუნველყოფის შექმნის პრობლემების და ამოცანების ანალიზი. პროგრამული აპლიკაციების აგება WCF/WPF და ASP.NET MVC პლატფორმაზე.

2. პროექტების (ან ჯგუფური პროექტების) თემების განხილვა და განაწილება სტუდენტებზე.

თანამედროვე პროგრამული პლატფორმებისა და ენების შერჩევა;

3. გამოყენებითი სფეროს ფუნქციონალური მოთხოვნები: ფორმირება და შესაბამისი პროგრამული სისტემის აგების ტექნიკური დავალების შედგენა ობიექტ-ორიენტირებული მიდგომის საფუძველზე

4. გამოყენებითი სფეროს პროგრამული აპლიკაცია: დაპროექტების და რეალიზაციის ეტაპების ამოცანების განსაზღვრა და სამუშაოების დაგეგმვა;

5. საპრობლემო სფეროს ბიზნეს-პროცესები და ბიზნეს-წესები: მათ საფუძველზე პროცესების სცენარების შედგენა (აქტიურობათა და მიმდევრობითობის დიაგრამების აგება)

6. ბიზნეს-პროცესების სცენარების პროგრამული კოდის აგება CASE ინსტრუმენტით: მიმდევრო¬ბითობის დიაგრამის შესაბამისი C#კოდის გენერირება და გამოკვლევა მუშაობისუნარიანობაზე

7. მომხმარებელთა ინტერფეისები: საპრობლემო სფეროს მომხმარებელთა ინტერფეისების შემუშავება;

8. ობიექტები და კლასები: საპრობლემო სფეროს ობიექტებისა და კლასების დაპროექტება. კლასების ატრიბუტების (ცვლადების), მეთოდების (ფუნქციების), მოვლენების განსაზღვრა

9. კლასთა-ასოციაციების დიაგრამები: გამოყენებითი სფეროს კლასთა-ასოციაციების დიაგრამე-ბის დაპროექტება უნიფიცირებული, ობიექტ-ორიენტირებული მიდგომის საფუძველზე.

მიღებული კლასთა ასოციაციის მოდელის შესაბამისი C#კოდის გენერირება და პროგრამის ტექსტის ანალიზი

10. სტანდარტული ბიბლიოთეკები: საპრობლემო სფეროს პროგრამული აპლიკაციის ასაგებად სტანდარტულ ბიბლიოთეკათა გამოყენება

11. მონაცემთა ბაზები ან/და ფაილური სისტემი: საპრობლემო სფეროს პროგრამული აპლიკაციის¬თვის მონაცემთა ბაზის ან/და ფაილური სისტემის მომზადება (SQL Server ბაზისთვის)

12. ა) Windows აპლიკაციის (Desktop) დაპროექტება: სტრუქტურული და ობიექტ-ორიენტირებული მეთოდებით ან

ბ) WEB-აპლიკაციის დაპროექტება და პროგრამული რეალიზაცია: სტრუქტურული და ობიექტ-ორიენტირებული მეთოდებით;

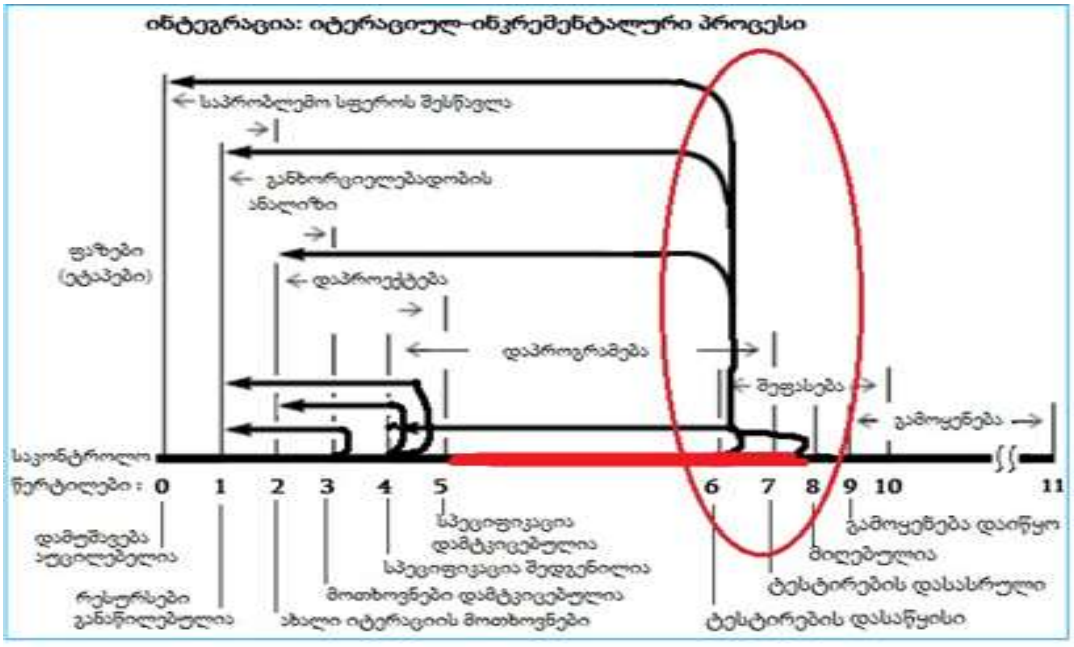
13. ა) Windows აპლიკაციის (Desktop) პროგრამული რეალიზაცია WPF-გარემოში: დიალოგური პროცედურების პროგრამირება ან

ბ) WEB-აპლიკაციისთვის დიალოგური პროცედურების დაპროგრამება WPF-გარემოში: მომხმარებლის ინტერფეისის (UI) დაპროგრამება ვებ-გვერდისათვის.

14. საპრობლემო სფეროს პროგრამული აპლიკაციის გამართვა და ტესტირება. პროგრამული სისტემის დემოვერსიის გამართვა, ტესტირება, ექსპერიმენტული მონაცემების საფუძველზე პროგრამული შედეგების მიღება და ანალიზი

15. საკურსო სამუშაოს პრეზენტაცია. საპრეზენტაციო ფაილის და საფინალო რეპორტის მომზადება

7.2. სამუშაოს ეტაპები პროგრამული სისტემის სასიცოცხლო ციკლის შესაბამისად



ნახ.26. პროგრამის დეველოპმენტის და ტესტირების პროცესი

7.3. საკურსო პროექტის სარჩევი

1. საპრობლემო სფეროს შერჩევა და კვლევის ობიექტისთვის ამოცანის დასმა (კლიენტ-სერვერული ან სერვის-ორიენტირებული არქიტექტურით);

2. გადასაწყვეტი ამოცანის „როლების და ფუნქციების“ განსაზღვრა (მაგალითად, UML დიაგრამები: UseCase, Activity და სცენარი - SequenceD;

3. პროექტისთვის მონაცემთა ბაზის სტრუქტურის შემუშავება (ER მოდელი) და SQL Server ბაზის შექმნა რამდენიმე ცხრილით (დასაშვებია სხვა მბ-ის გამოყენება);

4. მომხმარებელთა ინტერფეისების აგება (სამუშაო ფორმები), MsVisual Studio.NET ინტეგრირებულ გარემოში (WPF/WCF-ის ბაზაზე), C# და XAML საფუძველზე;

7. მომხმარებელთა ინტერფეისების დაკავშირება მონაცემთა ბაზასთან და შედეგების ასახვა Windows-ან Web- ფორმებზე (ბრაუზერში);

8. დამუშავებული პროგრამის ტესტირება და საბოლოო პროდუქტის მომზადება;

9. სისტემის დემოვერსია და საპრეზენტაციო სლაიდები.

8. საკურსო პროექტის სავარაუდო კვლევის ობიექტები და თემები:

1. საპრობლემო სფერო: **ბიბლიოთეკა**
   - 1.1. ობიექტი: მკითხველთა რეგისტრაცია
   - 1.2. ობიექტი: წიგნების ანბანური კატალოგები
   - 1.3. ობიექტი: წიგნების თემატური კატალოგები
   - 1.4. ობიექტი: წიგნების გაცემა/დაბრუნება
   - 1.5. ობიექტი: კადრები/ხელფასები
2. საპრობლემო სფერო: **ფაკულტეტი**
   - 2.1. ობიექტი: სტუდენტები, ჯგუფები, კურსები
   - 2.2. ობიექტი: ფაკულტეტის კათედრები, სპეციალობები
   - 2.3. ობიექტი: ჯგუფები, საგნები, კრედიტები
   - 2.4. ობიექტი: სტუდენტები, საგნები, გამოცდები, შედეგები
   - 2.5. ობიექტი: ლექტორები, კათედრები, ჯგუფები
3. საპრობლემო სფერო: **სუპერმარკეტი**
   - 3.1. ობიექტი: პროდუქტი, ფირმა, ფასი, კატეგორია
   - 3.2. ობიექტი: კლიენტთა მომსახურება, სალარო/ჩეკი
   - 3.3. ობიექტი: ელ–შეკვეთა ბინაზე მიტანით
   - 3.4. ობიექტი: დღიური/თვიური/წლიური ვაჭრობა
   - 3.5. ობიექტი: საწყობში პროდუქციის აღრიცხვა
4. საპრობლემო სფერო: **აფთიაქი**
   - 4.1. ობიექტი: მედიკამენტი, ქვეყანა, ფასი, კატეგორია
   - 4.2. ობიექტი: კლიენტთა მომსახურება, სალარო/ჩეკი
   - 4.3. ობიექტი: ელ–შეკვეთა ბინაზე მიტანით
   - 4.4. ობიექტი: დღიური/თვიური/წლიური ეკონომუკური მაჩვენებლები
   - 4.5. ობიექტი: აფთიაქის საწყობი
   - 4.6. ან სხვ.
5. საპრობლემო სფერო: **საწარმოო ფირმა**
   - 5.1. ობიექტი: პროდუქტი, ფასი, კატეგორია
   - 5.2. ობიექტი: პროდუქტი, თვითღირებულება, ფასი, მოგება, რენტაბელობა
   - 5.3. ობიექტი: პროდუქტი, ნედლეული, მიმწოდებელი, ნედლეულის_ფასი
   - 5.4. ობიექტი: თვიური/წლიური შემოსავალი, ხარჯი, მოგება
   - 5.5. ობიექტი: ნედლეულის და მზა პროდუქციის საწყობები
6. საპრობლემო სფერო: **კლინიკა**
   - 6.1. ობიექტი: პაციენტი, დაავადება, მკურნალი_ექიმი
   - 6.2. ობიექტი: ექიმი, ნოზოლოგიური_განყოფილება, ოთახი, ტელეფონი

6.3. ობიექტი: პაციენტი, დაავადება, მკურნალობის_ფასი
6.4. ობიექტი: თვიური/წლიური შემოსავალი, ხარჯი, მოგება
6.5. ობიექტი: ნოზოლოგიური_განყოფილება, საწოლების რაოდენობა, მკურნალობის_ვადა, მდგომარეობა

7. საპრობლემო სფერო: **მარკეტინგი**
3.1. ობიექტი: ავტომანქანების ბაზარი (ფირმა, მოდელი, სხვ.)
3.2. ობიექტი: ავტოპროფილაქტიკა, მომსახურების აღრიცხვა
3.3. ობიექტი: კომპიუტერების მაღაზია
3.4. ობიექტი: წიგნების მაღაზია სექციების მიხედვით
3.5. ობიექტი: პარფიუმერიის მაღაზია

8. საპრობლემო სფერო: **რესტორანი**
8.1. ობიექტი: მენიუ, კერძები, ფასები
8.2. ობიექტი: წინასწარი დაჯავშნის ამოცანა
8.3. ობიექტი: კლიენტთა მაგიდის მომსახურება
8.4. ობიექტი: შეკვეთების მიღება კერძების ბინაზე მიტანით
8.5. ობიექტი: თვიური/წლიური შემოსავალი, ხარჯი, მოგება

9. საპრობლემო სფერო: **კომერციული ბანკი**
9.1. ობიექტი: კრედიტების მენეჯმენტის სისტემა
9.2. ობიექტი: რისკების მართვა საკრედიტო სისტემისთვის
9.3. ობიექტი: საკრედიტო სისტემის მონიტორინგი
9.4. ობიექტი: ანაბრების მართვის და კონტროლის სისტემა
9.5. ობიექტი: ბანკთაშორისი გადარიცხვების ავტომატიზებული სისტემა

10. საკურსო თემების დაზუსტება ან სხვა სფეროებიდან შერჩევა ხდება პროფესორის და სტუდენტის კონსულტაციების პროცესში.

შენიშვნა: დასაშვებია ინდივიდუალური და ჯგუფური (2-3 მაგისტრანტი) პროექტის შესრულება. ჯგუფურ პროექტში ამოცანების რაოდენობა პროპორციულად იზრდება.

9. რეკომენდებული ლიტერატურა:

1. სურგულაძე გ., კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი (WCF ტექნოლოგია). ISBN 978-9941-0-7878-1. სტუ. „IT-კონსალტინგ ცენტრი“. თბ.,2015. -154 გვ.
2. 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.
3. 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
4. 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. CD-6016. https://gtu.ge/book/Surgu_SoftwareQuality.pdf
5. სურგულაძე გ., ურუშაძე ბ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). ISBN 978-9941-20-458-6. სტუ, „ტექნიკ.უნივ.“. თბ., 2014. -320 გვ. ბიბლ.ინდ. 004/23
6. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., ISBN 978-9941-20-165-1. სტუ, „ტექნიკ.უნივ.“. თბ., 2012. -324 გვ. ბიბლ.ინდ. 681.327/18
7. სურგულაძე გ., კივილაძე გ. (2017). შესავალი NoSQL მონაცემთა ბაზებში. ISBN 978-9941-0-9642-6. სტუ. „ტექნიკ.უნივ.“. თბ., – 152 გვ.: ბიბლ.ინდ. 004.65(02) /2
8. 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. ბიბლ.ინდ. 004.42(02)/8

დანართი_1

პროექტი: საპრობლემო სფერო „საფინანსო ბანკი“

1. ინფორმაციის გაცვლის პროგრამული რეალიზაციის ამოცანა SOA-ისთვის

ბიზნესპროცესების შესრულებისას ერთ-ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე სამუშაო პროცესებსა და ჰოსტდანართებს შორის [6].

წინამდებარე პარაგრფში აღწერილი გვაქვს, თუ როგორ შეიძლება ბიზნესპროცესების გამოყენებით გამარტივდეს და კოორდინაცია გაეწიოს კომუნიკაციათა სხვადასხვა სცენარს. აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საწარმოო ფირმებსა და ბანკებს შორის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, იურიდიული პირის მიერ საკრედიტო განაცხადის წარდგენა) გადაეცემა ბანკს. იმავე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა (სხვა ბანკში) და ასევე პასუხის გაცემა სხვა ორგანიზაციების მოთხოვნაზე.

მთავარი ქმედებები, რომლებიც კომუნიკაციისთვის გამოიყენება, არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ [1,3].

ჩვენ ავაგებთ მარტივ WPF-აპლიკაციას (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციას ბიზნესპროცესებს შორის. ბიზნესპროცესები შეიძლება განთავსდეს ვებსერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას სამუშაო პროცესის გადაწყვეტილების მისაწოდებლად არამუშა პროცესის კლიენტებისათვის, როგორიცაა ვებ-აპლიკაციები.

ვებსერვისი იღებს მოთხოვნას, ასრულებს მის სათანადო გადამუშავებას და აბრუნებს პასუხს. ეს, ბუნებრივია, სრულდება Receive და Send ქმედებებით. ვინაიდან ეს აქტიურობები ინტეგრირებულია Windows Communication Foundation (WCF) -თან, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები [1].

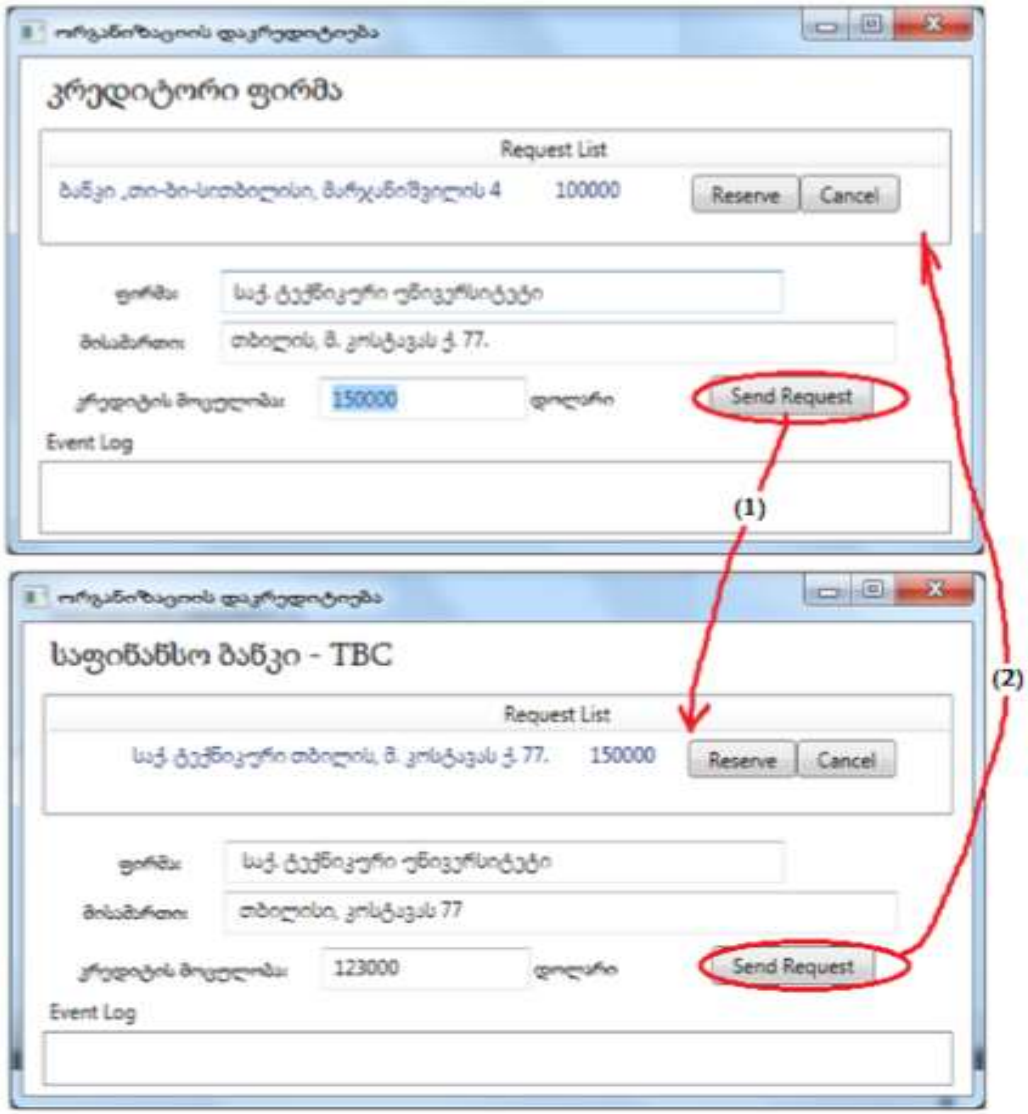
2. Window Form –ის განსაზღვრა

ავაგოთ მომხმარებელთა ინტერფეისის ფორმა, რომელიც გამოდგება, მაგალითად ბანკებისათვის. მისი მაკეტი მოცემულია დ1 ნახაზზე. შესაბამისი ფორმა აგებულია XAML ფაილის სახით დიზაინის რეჟიმში, როგორც ზემოთ აღვნიშნეთ, WPF ტექნოლოგიის გამოყენებით.

ინტერფეისის დაპროეტება და პროგრამული რეალიზაცია ხდება Visual Studio.NET Framework 4.5 გარემოში, C#.NET ენის საფუძველზე.

გამოიყენება ინტერფეისის დიზაინისთვის XAML და პროგრამის ლოგიკისათვის C# ენები. 1_ლისტინგში მოცემულია XAML ფაილის ტექსტი. ფორმის ზედა ნაწილში „მოთხოვნების სია“ (Request List) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია.

მოთხოვნის გასაგზავნად, მაგალითად, ბანკში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება: - „ფირმა“, - „მისამართი“ და - მოთხოვნილი „კრედიტის მოცულობა“.



დ1. მომხმარებელთა ინტერფეისები

შემდეგ ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). ქვედა მარცხენა კუთხეში Event Log ასახავს ბიზნესპროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.

Crediting.xaml ფაილის ლისტინგი ასეთია:

```
<!– – ლისტინგი_1 – – – –>
<Window x:Class="FirmsCrediting.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ორგანიზაციის დაკრედიტება" Height="480" Width="650"
    Loaded="Window_Loaded" Unloaded="Window_Unloaded">
  <Grid>
    <Label Height="40" HorizontalAlignment="Left" Margin="12,0,0,0"
      Name="lblBranch" FontSize="20" VerticalAlignment="Top"
        Width="462" FontStretch="Expanded"
         FontFamily="Sylfaen">კრედიტორი ფირმა</Label>
    <ListView x:Name="requestList" Margin="12,42,12,5" Height="150"
```

```
          VerticalAlignment="Top" ItemsSource="{Binding}">
      <ListView.View>
        <GridView>
          <GridViewColumn Header="Request List" Width="610">
            <GridViewColumn.CellTemplate>
              <DataTemplate>
                <StackPanel Orientation="Horizontal">
    <TextBlock Text="{Binding Requester.BranchName}" Width="100"/>
    <TextBlock Text="{Binding FirmName}" Width="95"/>
    <TextBlock Text="{Binding Adress}" Width="180"/>
    <TextBlock Text="{Binding CreditQ}" Width="90"/>
    <Button Content="Reserve" Tag="{Binding InstanceID}"
                    Click="Reserve" Width="65"/>
    <Button Content="Cancel"  Tag="{Binding InstanceID}"
                    Click="Cancel" Width="60"/>
                </StackPanel>
              </DataTemplate>
            </GridViewColumn.CellTemplate>
          </GridViewColumn>
        </GridView>
      </ListView.View>
  </ListView>
 <Label Height="30" Margin="27,0,0,205" Name="label5"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="73" HorizontalContentAlignment="Right"
          Content="ფირმა:" FontFamily="Sylfaen"></Label>
  <Label Height="30" Margin="27,0,0,176" Name="label2"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="77" HorizontalContentAlignment="Right"
          Content="მისამართი:" FontFamily="Sylfaen"></Label>
  <Label Height="30" Margin="13,0,0,142" Name="label3"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="151" HorizontalContentAlignment="Right"
          Content="კრედიტის მოცულობა:" FontFamily="Sylfaen">
  </Label>
  <TextBox Height="25" Margin="121,0,0,210"  Name="txtFirmName"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="400" /> <TextBox Height="25" Margin="121,0,0,180"
          Name="txtAdress" VerticalAlignment="Bottom"
          HorizontalAlignment="Left" Width="468" />
  <TextBox Height="25" Margin="0,0,329,147" Name="txtCreditQ"
          VerticalAlignment="Bottom" HorizontalAlignment="Right"
          Width="125" />
  <Button Height="23" Margin="477,0,0,150" Name="btnRequest"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="98" Click="btnRequest_Click">Send Request</Button>
```

```
    <Label Height="27" HorizontalAlignment="Left" Margin="11,0,0,121"
           Name="label4" VerticalAlignment="Bottom" Width="76">
           Event Log</Label>
    <ListBox Margin="12,0,12,12" Name="lstEvents" Height="111"
           VerticalAlignment="Bottom" FontStretch="Condensed"
           FontSize="10" />
     <Label Content="დოლარი" Height="28" HorizontalAlignment="Left"
           Margin="302,269,0,0"  Name="label1" VerticalAlignment="Top"
           Width="67" FontFamily="Sylfaen" />
  </Grid>
</Window>
```

ფორმის ზედა ნაწილში „მოთხოვნების სია“ (Request List) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად მაგალითად, ბანკში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება „ფირმა“, „მისამართი“ და მოთხოვნილი „კრედიტის მოცულობა“, შემდეგ გაგზავნის ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). ქვედა მარცხენა კუთხეში Event Log ასახავს ბიზნესპროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.

3. სერვისის პროგრამული რეალიზაცია

ჩვენი სისტემის ClientService.cs კოდის რეალიზაცია ნაჩვენებია 2_ლისტინგში.

```
// ----- ლისტინგი_2 --- ClientService.cs ---------------
using System;
using System.ServiceModel;   // !!!
namespace FirmsCrediting
{
   public class ClientService : ICreditReservation
   {
      public void RequestCredit(CreditingRequest request)
      {
         ApplicationInterface.RequestCredit(request);
      }
      public void RespondToRequest(CreditingResponse response)
      {
         ApplicationInterface.RespondToRequest(response);
      }
   }
}
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენ მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს.

ApplicationInterface.cs ფაილი მოცემულია 3_ლისტინგში.

```
//  ---- ლისტინგი 26.3 --- ApplicationInterface.cs ფაილისთვის ----
using System;
using System.Windows.Controls;
using System.Activities;
namespace FirmsCrediting
{
```

```
public static class ApplicationInterface
 {
   public static MainWindow _app { get; set; }
   public static void AddEvent(String status)
    {
      if (_app != null)
       {
         new ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
        }
    }
 public static void RequestCredit(CreditingRequest request)
  {
     if (_app != null)
        _app.RequestCredit(request);
  }
 public static void RespondToRequest(CreditingResponse response)
  {
     if (_app != null)
         _app.RespondToRequest(response);
     }
   public static void NewRequest(CreditingRequest request)
    {
      if (_app != null)
        _app.AddNewRequest(request);
    }
  }
}
```

ეს მეთოდები, თავის მხრივ, იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო იქნება ამ მეთოდების რეალიზება Crediting.xaml.cs ფაილში.

4. ServiceHost -ის რეალიზაცია

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად. იგი პროექტის Crediting.xaml.cs ფაილში თავსდება კონსტრუქტორის წინ კლასის წევრის სახით (ლისტინგი_4).

```
//-- ლისტინგი_4 ------
public partial class MainWindow : Window
  {
     private ServiceHost _sh;   // !!!
     public MainWindow()
     {    InitializeComponent();
        ApplicationInterface._app = this;
     }
  ...
```

ServiceHost იწყება მაშინ, როცა ფანჯარა ჩატვირთულია და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 5_ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
 // --- ლისტინგი_5 ------------------
private void Window_Loaded(object sender, RoutedEventArgs e)
 {
  // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და მისი
  // ქსელური მისამართი
  Configuration config = ConfigurationManager
                  .OpenExeConfiguration(ConfigurationUserLevel.None);
  AppSettingsSection app = (AppSettingsSection)config
                     .GetSection("appSettings");
  string adr = app.Settings["BranchAddress"].Value;
      // ფილიალის სახელის გამოტანა ფორმაზე
      lblBranch.Content = app.Settings["Branch Name"].Value;

      // ServiceHost-ის შექმნა
      _sh = new ServiceHost(typeof(ClientService));

     // დასასრულის წერტილის (Endpoint) დამატება
     string szAddress = "http://localhost:" + adr + "/ClientService";
     System.ServiceModel.Channels.Binding bBinding =
           new BasicHttpBinding();
    _sh.AddServiceEndpoint(typeof(ICreditReservation),bBinding,
                  szAddress);
    // ServiceHost-ის გახსნა შეტყობინების მისაღებად (listen)
    _sh.Open();
 }
private void Window_Unloaded(object sender, RoutedEventArgs e)
 {
   // service host-ის დატოვება
     _sh.Close();
 }
```

მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს ფილიალის სახელს lblBranch -მართვის ელემენტში, ამიტომაც ფორმა ასახავს ფირმის სახელს. შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასის. შემდეგ იგი აკონფიგურირებს დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიბმისა და კონტრაქტის სამეულს.

Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი აღარ მოხდება შეტყობინებების მიღება.

აპლიკაციის ამუშავება: საჭიროა აპლიკაციის რამდენიმე კოპიოს ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება Solution-ის (გადაწყვეტის) აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად.

შევქმნათ ახალი ფოლდერი BankRequest -ფოლდერის ქვეშ, რომელიც იძახებს ბანკებს. შემდეგ დავაკოპიროთ ბანკის ფოლდერში ფაილები, რომლებიც ზემოთ შევქმენით;

FirmsCrediting.exe // Application
FirmsCrediting.exe.config // XML Configuration file
FirmsCrediting.pdb // program debug database

სისტემის ამუშავების შემდეგ გვექნება ორი სამუშაო ფანჯარა, მაგალითად, ერთი კრედიტორი ფირმის, მეორე საფინანსო ბანკის. მათ შორის შესაძლებელი იქნება ინფორმაციისა და შეტყობინებების გაცვლა, რისი მიღწევაც გვინდოდა (ნახ. დ1).

გადაეცა წარმოებას 10..03.2021. ხელმოწერილია დასაბეჭდად 20.03.2021. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 2,5. ტირაჟი 50 ეგზ.

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

ISBN 978-9941-8-2725-9