תרשים תוכנית ניהול ערים C# :

צד C#:

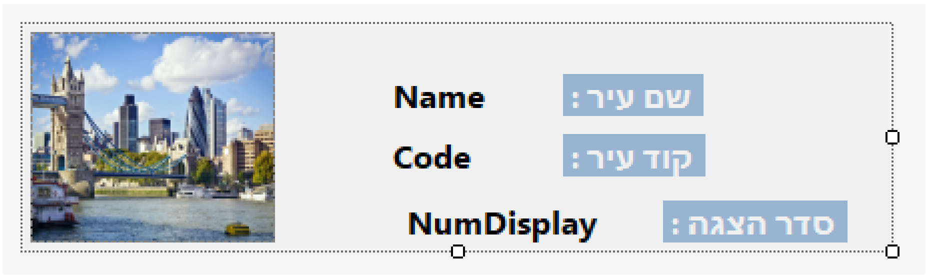
Entitles:

```
public class Street
    {
        private string Name;
        private static int streets = 0;
        private int StreetCodeNow;
        private int Order;

        private int CityCode;

        public Street(string Name, int Order, int CityCode)
        {
            this.Name = Name;
            this.StreetCodeNow = streets;
            streets++;
            this.Order = Order;
            this.CityCode = CityCode;
        }

        public string StreetName
        {
            get { return Name; }
            set { Name = value; }
        }

        public int StreetOrder
        {
            get { return Order; }
            set { Order = value; }
        }

        public int CityCodeNow
        {
            get { return CityCode; }
            set { CityCode = value; }
        }

        public int GetStreetCodeNow()
        { return StreetCodeNow; }

    }
```

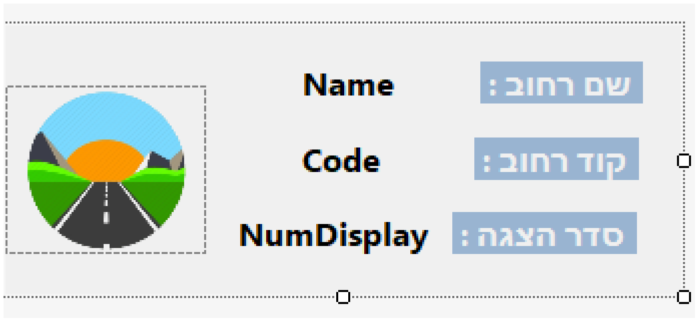
```
public class City
    {

        #region Properties
        private static int citiyCount = 0;
        private int cityCodeNow = 0;

        private string cityName;
        private int cityOrder;

        #endregion
        public string CityName
        {
            get { return cityName; }
            set { cityName = value; }
        }

        public int CityOrder
        {
            get { return cityOrder; }
            set { cityOrder = value; }
        }
        public City(string cityName, int cityOrder)
        {
            this.cityName = cityName;
            citiyCount++;
            this.cityOrder = cityOrder;
            this.cityCodeNow = citiyCount;
        }

        public int getCityCodeNow()
        {
            return cityCodeNow;
        }

    }
```

```
using System.Windows.Forms;

public class HelpFuncs
    {
        public static void createOrderList(Form form, ComboBox NumDisplay,
string kindList)
        {
            Add parent = form as Add;
            NumDisplay.Items.Clear();
            NumDisplay.Items.Add(1);
            if (kindList == "city")
            {
                if (parent.CityList.Count > 0)
                {
                    for (int i = 0; i < parent.CityList.Count; i++)
                    {
                        NumDisplay.Items.Add(i + 2);
                    }
                }
            }
            else
            {
                if (parent.CityStreet.Count > 0)
                {
                    for (int i = 0; i < parent.CityStreet.Count; i++)
                    {
                        NumDisplay.Items.Add(i + 2);
                    }
                }
            }


            NumDisplay.SelectedIndex = 0;
        }

        public static void Create_FlowLayoutPanel_FromItems(object[]
uItems, string kindOfShow, FlowLayoutPanel flowLayoutPanel)
        {
            if (uItems != null && uItems.Length > 0)
            {
                if (flowLayoutPanel.Controls.Count > 0)
                    flowLayoutPanel.Controls.Clear();
                if (uItems is UCity[] && kindOfShow == "cities")
                {
                    foreach (UCity u in uItems)
                    {
                        flowLayoutPanel.Controls.Add(u);
                    }
                }

                else if (uItems is UStreet[] && kindOfShow == "streets")
                {
                    foreach (UStreet u in uItems)
                    {
                        flowLayoutPanel.Controls.Add(u);
                    }
                }
            }
        }
    }
```

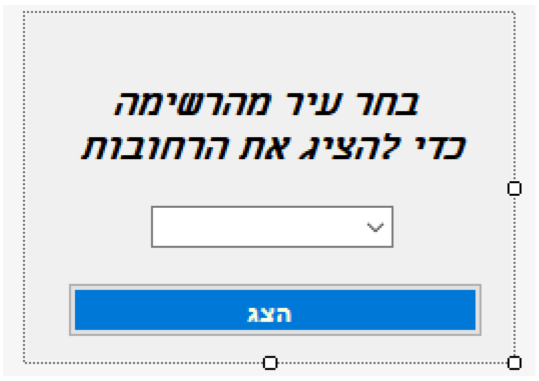
UC:



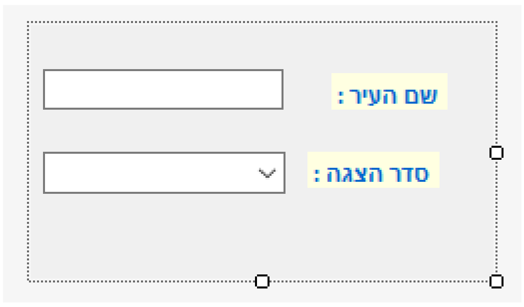
```
public partial class UCity : UserControl
{

    public UCity()
    {
        InitializeComponent();
    }

    #region Properties

    private string cityName;
    private int cityOrder;
    private int cityCode;

    #endregion

    [Category("Custom Props")]
    public string CityName
    {
        get { return cityName; }
        set { cityName = value; Name.Text = value; }
    }

    [Category("Custom Props")]
    public int CityOrder
    {
        get { return cityOrder; }
        set { cityOrder = value; NumDisplay.Text = value.ToString(); }
    }
    [Category("Custom Props")]
    public int CityCode
    {
        get { return cityCode; }
        set { cityCode = value; Code.Text = value.ToString(); }
    }
}
```



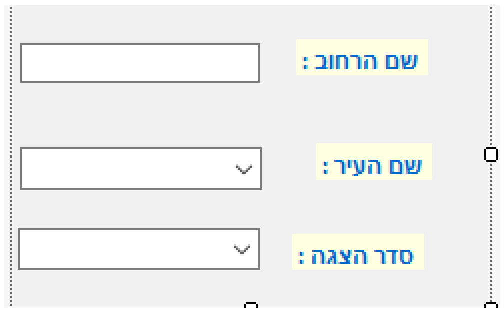
```
public UStreet()
        {
            InitializeComponent();
        }

        #region Properties

        private string streetName;
        private int streetCode;
        private int streetOrder;
        private int cityCode;

        #endregion

        [Category("Custom Props")]
        public string StreetName
        {
            get { return streetName; }
            set { streetName = value; Name.Text = value; }
        }

        [Category("Custom Props")]
        public int StreetCode
        {
            get { return streetCode; }
            set { streetCode = value; Code.Text = value.ToString(); }
        }
        [Category("Custom Props")]
        public int StreetOrder
        {
            get { return streetOrder; }
            set { streetOrder = value; NumDisplay.Text = value.ToString();
}
        }

        [Category("Custom Props")]
        public int CityCode
        {
            get { return cityCode; }
            set { cityCode = value; }
        }
    }
```



```
public ChooseCityShow()
        {
            InitializeComponent();
        }

        Show parent;

        private void ChooseCityShow_Load(object sender, EventArgs e)
        {
            parent = this.Parent as Show;
            List<City> cities = parent.CityList;
            if (cities.Count == 0)
            {
                MessageBox.Show("Oh no, there are no more cities !");
                return;
            }
            Cities.DataSource = cities;
            Cities.ValueMember = "cityName";
            Cities.DisplayMember = "cityName";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            City city = Cities.SelectedItem as City;
            if (city != null)
            {
                int cityCode = city.getCityCodeNow();
                parent.SetStreetsShow(cityCode);
            }
        }
```

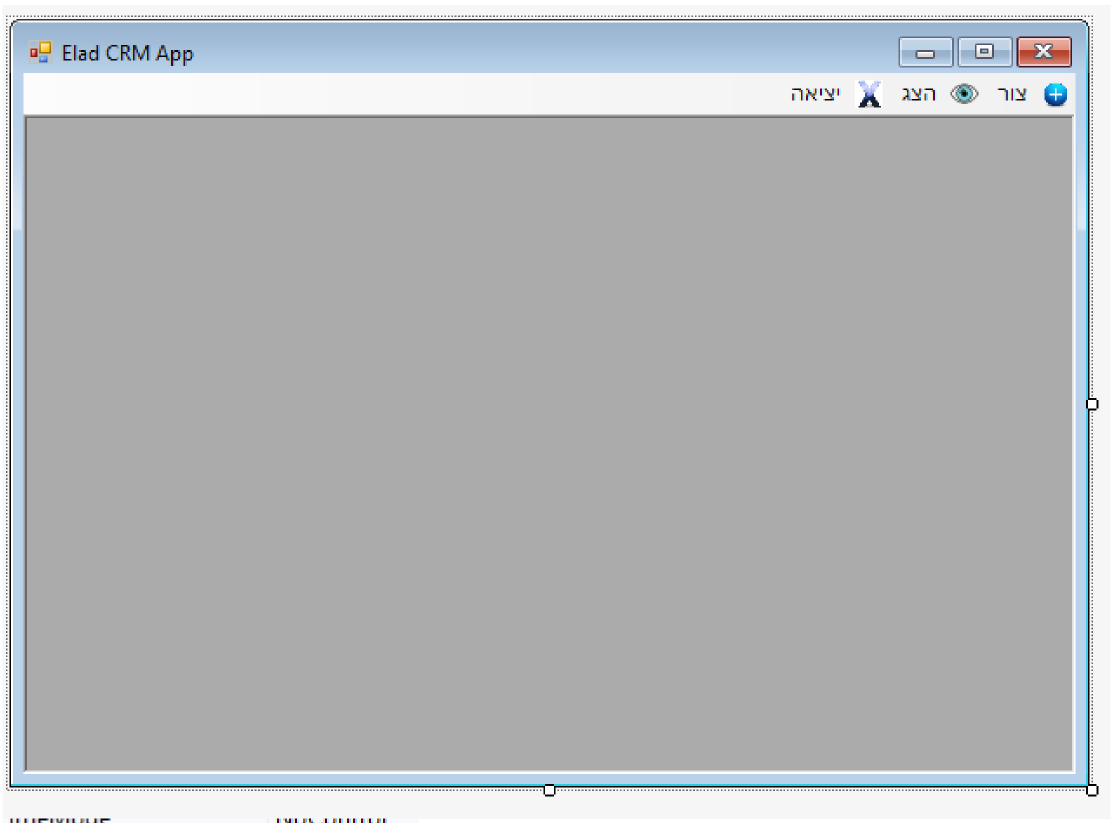


```
public partial class AddCity : UserControl
    {
        public AddCity()
        {
            InitializeComponent();
        }

        private void NameCity_KeyPress(object sender, KeyPressEventArgs e)
        {
            // Verify that the pressed key isn't letter or control like 
del key
            if (!char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar))
            {
                e.Handled = true;
            }
        }

        Add parent;
        private void AddCity_Load(object sender, EventArgs e)
        {
            HelpFuncs.createOrderList(this.Parent as Add, NumDisplay,
"city");
            parent = this.Parent as Add;
        }

        public void AddNewCityFromUC()
        {
            string cityName = NameCity.Text;
            int cityOrder = NumDisplay.SelectedIndex;
            City city = new City(cityName, cityOrder);
            parent.addToList("city", city);
            HelpFuncs.createOrderList(this.Parent as Add, NumDisplay,
"city");
        }
    }
```



```
    public partial class AddStreet : UserControl
    {
        public AddStreet()
        {
            InitializeComponent();
        }

        private void NameStreet_KeyPress(object sender, KeyPressEventArgs
e)
        {
            // Verify that the pressed key isn't letter or control like
del key
            if (!char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar))
            {
                e.Handled = true;
            }

        }

        Add parent;

        private void AddStreet_Load(object sender, EventArgs e)
        {
            HelpFuncs.createOrderList(this.Parent as Add, NumDisplay,
"street");
            parent = this.Parent as Add;
            List<City> cities = parent.CityList;
            if (cities.Count == 0)
            {
                MessageBox.Show("Oh no, there are no more cities !");
                return;
            }
            Cities.DataSource = cities;
            Cities.ValueMember = "cityName";
            Cities.DisplayMember = "cityName";
        }

        public void AddNewStreetFromUC()
        {

            City city = Cities.SelectedItem as City;
            if (city != null)
            {
                string streetName = NameStreet.Text;
                int streetOrder = NumDisplay.SelectedIndex;
                int cityCode = city.getCityCodeNow();
                Street street = new Street(streetName, streetOrder,
cityCode);
                parent.addToList("street", street);
                HelpFuncs.createOrderList(this.Parent as Add, NumDisplay,
"street");
            }}}
```

Form:



IsMdiContainer	True

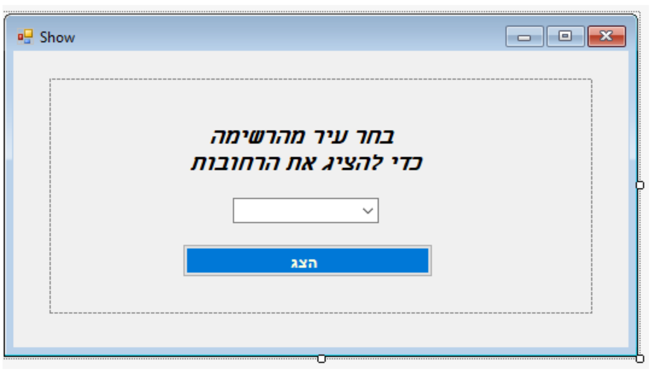
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
        private MenuStrip menuStrip2;
        private ToolStripMenuItem צורToolStripMenuItem;
        private ToolStripMenuItem עירToolStripMenuItem;
        private ToolStripMenuItem רחובToolStripMenuItem;
        private ToolStripMenuItem הצגToolStripMenuItem;
        private ToolStripMenuItem עריםToolStripMenuItem;
        private ToolStripMenuItem רחובותלפיעירToolStripMenuItem;
        private ToolStripMenuItem יציאהToolStripMenuItem;

        private List<City> cityList;
        private List<Street> streetList;


        public Program()
        {
            InitializeComponent();
            cityList = new List<City>();
            streetList = new List<Street>();
        }
        static void Main(string[] args)
        {
```

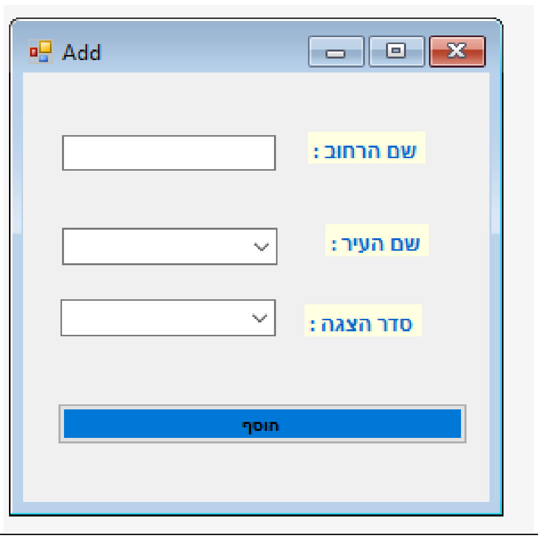
```
        private void עירToolStripMenuItem_Click(object sender, EventArgs e)
        {
            RemoveFormsAndShow("add", "addCity");
        }

        private void רחובToolStripMenuItem_Click(object sender, EventArgs e)
        {
            RemoveFormsAndShow("add", "addStreet");
        }


        private void עריםToolStripMenuItem_Click(object sender, EventArgs e)
        {
```

```
        public void addCityToList(City city)
        {
            cityList.Add(city);
        }

        public List<Street> GetStreetList()
        {
            return streetList;
        }
        public void addStreeToList(Street street)
```

```
        public void addCityToList(City city)
        {
            cityList.Add(city);
        }

        public List<Street> GetStreetList()
        {
            return streetList;
        }
        public void addStreeToList(Street street)
```



```
public partial class Show : Form
    {
        List<City> cityList;
        List<Street> streetList;

        public Show(string kindOfShow, List<City> cityList, List<Street> streetList)
        {
            InitializeComponent();
            this.cityList = cityList;
            this.streetList = streetList;
            uCity1.Hide();
            uStreet1.Hide();
            flowLayoutPanel1.Hide();
            chooseCityShow1.Hide();
            switch (kindOfShow)
            {
                case "showCities":
                    int itemsCount = cityList.Count;
                    UCity[] cities = new UCity[itemsCount];
                    int iCity = 0;
                    foreach (City city in cityList)
                    {
                        cities[iCity] = new UCity();
                        cities[iCity].CityName = city.CityName;
                        cities[iCity].CityCode = city.getCityCodeNow();
                        cities[iCity].CityOrder = city.CityOrder;
                        iCity++;

                    }
                    flowLayoutPanel1.Show();
                    HelpFuncs.Create_FlowLayoutPanel_FromItems(cities, "cities", flowLayoutPanel1);
                    break;

                case "showStreets":
                    chooseCityShow1.Show();
                    break;
            }
        }
        public void SetStreetsShow(int codeCityShow)
        {
            chooseCityShow1.Hide();
            int streetCount = streetList.Count;
            UStreet[] streets = new UStreet[streetCount];
            int iStreet = 0;
            foreach (Street street in streetList)
            {
                if (street.CityCodeNow == codeCityShow)
                {
                    streets[iStreet] = new UStreet();
                    streets[iStreet].StreetName = street.StreetName;
                    streets[iStreet].StreetCode = street.GetStreetCodeNow();
                    streets[iStreet].StreetOrder = street.StreetOrder;
                }

                iStreet++;

            }

            HelpFuncs.Create_FlowLayoutPanel_FromItems(streets, "streets", flowLayoutPanel1);
            flowLayoutPanel1.Show();
        }

        public List<City> CityList
        {
            get { return cityList; }
        }

        public List<Street> CityStreet
        {
            get { return streetList; }
        }

    }
```

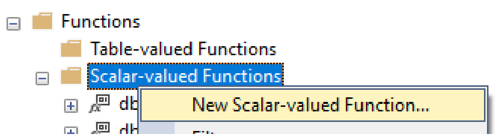


```
  //point to function
    public delegate void add();
    public partial class Add : Form
    {
        // create function to delegate
        public add addItem;

        // list of the current city and street
        List<City> cityList;
        List<Street> streetList;

        public Add(string kindOfAdd, List<City> cityList, List<Street> streetList)
        {
            InitializeComponent();

            this.cityList = cityList;
            this.streetList = streetList;

            addCity1.Hide();
            addStreet1.Hide();
            switch (kindOfAdd)
            {
                case "addCity":
                    addCity1.Show();
                    break;

                case "addStreet":
                    addStreet1.Show();
                    break;
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Check if have functions => We'll only want one function to run
            if (this.addItem != null)
            {
                this.addItem = null;
            }

            // Check which kind of UC add open
            if (addCity1.IsHandleCreated)
            {
                this.addItem += new add(addCity1.AddNewCityFromUC);
            }
            else
            {
                this.addItem += new add(addStreet1.AddNewStreetFromUC);
            }
            this.addItem();
        }
        public List<City> CityList
        {
            get { return cityList; }
        }

        public List<Street> CityStreet
        {
            get { return streetList; }
        }

        public void addToList(string kindList, object item)
        {
            if (kindList == "city")
            {
                foreach (City city in cityList)
                {
                    if (city.CityName == (item as City).CityName)
                    {
                        MessageBox.Show($"You cannot create a city with a name that already exists - 
{city.CityName}");
                        return;
                    }

                    else if (city.CityOrder == (item as City).CityOrder)
                    {
                        MessageBox.Show($"You cannot create a city with a city order that already 
exists - {city.CityOrder + 1}");
                        return;
                    }
                }
                this.cityList.Add(item as City);
            }
            else
            {
                foreach (Street street in streetList)
                {
                    if (street.StreetName == (item as Street).StreetName)
                    {
                        //check not have the same city code like the another
                        if (street.CityCodeNow == (item as Street).CityCodeNow)
                        {
                            MessageBox.Show($"You cannot create a street with the same name in the 
same city - {street.StreetName} in {street.CityCodeNow}");
                            return;
                        }
                    }

                    else if (street.StreetOrder == (item as Street).StreetOrder)
                    {
                        MessageBox.Show($"You cannot create a street with a street order that already 
exists - {street.StreetOrder + 1}");
                        return;
                    }
                }
                this.streetList.Add(item as Street);
            }
            MessageBox.Show("Add successfully!");
        }
                                                                                                    }
```

תרשים תוכנית ניהול ספרייה C# :

צד SQL:

Functions:



```
USE [Library]
GO
/****** Object:  UserDefinedFunction [dbo].[funName]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:          <Daniel Artzi>
-- Create date: <07/09/2022>
-- Description:     <fun desc >
-- =============================================
```

```sql
ALTER FUNCTION [dbo].[Validation_CheckBook]
(
   @Book_Code nchar(13),
   @Book_Title nvarchar(20),
   @Book_FirstName_Author nvarchar(20) ,
   @Book_LastName_Author nvarchar(20),
   @Book_PublicationDate date,
   @Book_Category nvarchar(25),
   @Book_SecondaryCategory nvarchar(35) = null
   --There are situations in which we would like to test only on
)
RETURNS nvarchar(500)

AS
BEGIN


       declare @Error nvarchar(500)
       --                                         -- *** Check for values:
       --                                                       --- Date greater than current date
       --                                                       --- 13 digits code
       --                                                       --- only digits code
       --                                                       --- only letters name
       --                                                       --- only letters category
       --                                                       --- checking if a category exists ***

       if (@Book_PublicationDate > GETDATE())
       begin

              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')


                     -- linebreaks ->
                     SET @ERROR =  'Date cannot be greater than current date !'
              else
                     SET @ERROR += CHAR(13)+CHAR(10)+ 'Date cannot be greater than current date !'
       end

              --                                         -- *** Check for 13 digits code  ***

       if (LEN(@Book_Code) != 13)
       begin

              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')


                     -- linebreaks ->
                     SET @ERROR =  'Barcode must contain 13 digits !'
              else
                     SET @ERROR += CHAR(13)+CHAR(10)+ 'Barcode must contain 13 digits !'
       end
       --                                         -- *** Check for only digits ***

       if (@Book_Code LIKE '%[^0-9]%' or @Book_Code is null)
       begin

              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')


                     -- linebreaks ->
                     SET @ERROR =  'Barcode must contain only digits !'
              else
                     SET @ERROR += CHAR(13)+CHAR(10)+'Barcode must contain only digits !'
       end
```

```
	--						-- *** Check for only letters ***

	if (( @Book_FirstName_Author  LIKE '%[^A-Za-zת-א]%' or @Book_FirstName_Author  Is Null or 
@Book_FirstName_Author  = '' )
		or ( @Book_LastName_Author  LIKE '%[^A-Za-zת-א]%' or @Book_LastName_Author  Is Null or 
@Book_LastName_Author  = ''))
	begin

		-- *** Checking if errors are already written ***
		if (@ERROR IS NULL or @ERROR = '')


			-- linebreaks ->
			SET @ERROR =  'Name author must be only letters. !'
		else
			SET @ERROR += CHAR(13)+CHAR(10)+ 'Name author must be only letters !'
	end

	if (( @Book_Category  LIKE '%[^A-Za-zת-א]%' or @Book_Category IS NULl or @Book_Category = '' )
	    or (@Book_SecondaryCategory  LIKE '%[^A-Za-zת-א]%' or @Book_SecondaryCategory = '' ) )
	begin

		-- *** Checking if errors are already written ***
		if (@ERROR IS NULL or @ERROR = '')


			-- linebreaks ->
			SET @ERROR =  'Category must be only letters. !'
		else
			SET @ERROR += CHAR(13)+CHAR(10)+ 'Category must be only letters !'
	end

	--						-- *** Check if a category exists and enter secondary 
category  ***

	if  (@Book_SecondaryCategory is not null)
	begin
		if not EXISTS (select top 1 * from ShowAllCategories with(nolock) where Category = 
@Book_Category and SecondaryCategory = @Book_SecondaryCategory)
		begin
		-- *** Checking if errors are already written ***
				if (@ERROR IS NULL or @ERROR = '')


			-- linebreaks ->
					SET @ERROR =  'The category or secondary - category does not exist !'
				else
					SET @ERROR += CHAR(13)+CHAR(10)+ 'The category or secondary - category does 
not exist !'
		end
	end

	else
	begin
		if not EXISTS (select top 1 * from ShowAllCategories with(nolock) where Category = 
@Book_Category)
		begin
		-- *** Checking if errors are already written ***
				if (@ERROR IS NULL or @ERROR = '')


			-- linebreaks ->
					SET @ERROR =  'The category or secondary - category does not exist !'
				else
					SET @ERROR += CHAR(13)+CHAR(10)+ 'The category or secondary - category does 
not exist !'
		end
	end

	RETURN @ERROR

END
```

```
ALTER FUNCTION [dbo].[Validation_CheckBorrow]
(
   @Code nchar(13),
   @Id nchar(9)
)

RETURNS nvarchar(500)

AS
BEGIN


       declare @Error nvarchar(500)
       --                                          -- *** Check for values:
       --                                                        --- already exists
       --                                                        --- only letters category ***

                                                          -- *** Checking if the category with subcategory
exists ***

       -- *** Checking if there is a book code or id  ***

       if not EXISTS(select top 1 * from Book with(nolock) where Code = @Code)
       begin
              SET @ERROR =  'An error occurred, such a book code does not exist !'
       end

       if not EXISTS(select top 1 * from Users with(nolock) where id = @Id)
       begin
              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')



                     SET @ERROR =  'What a shame, there is no user with such an ID card :K !'
              else                        -- linebreaks ->
                     SET @ERROR +=  CHAR(13)+CHAR(10)+'What a shame, there is no user with such an ID card :K
!'
       end

       RETURN @ERROR

END
```

```sql
ALTER FUNCTION [dbo].[Validation_CheckExistingCategories]
(
   @Category nvarchar(25),
   @SecondaryCategory nvarchar(35)
)

RETURNS nvarchar(500)

AS
BEGIN


       declare @Error nvarchar(500)
       --                                          -- *** Check for values:
       --                                                        --- already exists
       --                                                        --- only letters category ***

                                                          -- *** Checking if the category with subcategory
exists ***

       if EXISTS(select top 1 * from ExistingCategories with(nolock) where Category = @Category and
SecondaryCategory = @SecondaryCategory)
       begin
              SET @ERROR =  'This classification category already exists !'
       end

       if (( @Category  LIKE '%[^A-Za-zא-ת]%' or @Category IS NULl or @Category = '' )
           or (@SecondaryCategory  LIKE '%[^A-Za-zא-ת]%' or @SecondaryCategory IS NULl  or @SecondaryCategory
= '' ) )
       begin

              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')



                     SET @ERROR =  'Category must be only letters. !'
              else                         -- linebreaks ->
                     SET @ERROR +=  CHAR(13)+CHAR(10)+'Category must be only letters !'
       end

       RETURN @ERROR

END
```

```
ALTER FUNCTION [dbo].[Validation_CheckIsraelID]
(
       @id nchar(9)
)
RETURNS bit

AS
BEGIN

       -- need 9 digit
       if len(@id)<>9  return 0;

       --The right digit is the check digit
       declare @numberPass TinyInt = Right(@id,1)

       -- All but the rightmost digits are the body of the number
       declare @numbersID nvarchar(10) = left(@id,8)

       declare @numbeCheck TinyInt = 0;
       declare @strNum nvarchar(20) = '';
       declare @i int = 1;

       --Accumulates the digits by multiplying them by weights

       WHILE @i <= 8
       begin
              --The test coefficient is in the form of
                     --1 2 1 2 1 2 1 2 1
                     -- SUBSTRING(string, start, length)
                     -- get the next number
              set @strNum += cast(Cast(SUBSTRING(@numbersID,@i,1) As TinyInt) * (case when @i%2 = 0 then 2
else 1 end) as nvarchar);
              set @i+=1;
       end

       set @i = 1;

       --connect the generated digits

       WHILE @i <= len(@strNum)
       begin
              set @numbeCheck += Cast(SUBSTRING(@strNum,@i,1) As TinyInt)
              set @i+=1;
       end

       -- Updates to the number of complements to an exact multiple of ten

       set @numbeCheck = (10 - (@numbeCheck%10))

       -- Returns a value verified by checking whether the check digit matches
       --If the number is divisible by 10 without a remainder, then the id is correct

       RETURN (case when @numbeCheck=@numberPass then 1 else 0 end)

END
```

```
ALTER FUNCTION [dbo].[Validation_CheckUser]
(
  @User_id nchar(9),
  @User_FirstName nvarchar(20),
  @User_LastName nvarchar(20) ,
  @User_Type bit,
  @User_Email nvarchar(20),
  @User_Password nchar(10)
)
RETURNS nvarchar(500)

AS
BEGIN


       declare @Error nvarchar(500)

       --                                       -- *** Check for values:
       --                                                     --- id format ( 9 digit )
       --                                                     --- type value
       --                                                     --- only letters name
       --                                                     --- Email is written correctly
       --                                                     --- A password must be 10 characters ,
contains a number, an uppercase letter, a lowercase letter, and a special character ***

       declare @resCheckId bit;
       set @resCheckId = [dbo].[Validation_CheckIsraelID](@User_id)
       if(@resCheckId = 0 )
       begin



                     SET @ERROR =  'Incorrect ID ! '
       end

       if(@User_Type > 1 or @User_Type < 0)
       begin
       -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')



                     SET @ERROR =  'The value of type must be either 0 or 1 ... !'
              else                    -- linebreaks ->
                     SET @ERROR += CHAR(13)+CHAR(10)+ 'The value of type must be either 0 or 1 ...  !'
       end

       if (( @User_FirstName  LIKE '%[^A-Za-zא-ת]%' or @User_FirstName IS NULL or @User_FirstName = '' )
              or ( @User_LastName  LIKE '%[^A-Za-zא-ת]%' or @User_LastName IS NULL or @User_LastName = '' ))
       begin

              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')



                     SET @ERROR =  'Name user must be only letters. !'
              else                    -- linebreaks ->
                     SET @ERROR += CHAR(13)+CHAR(10)+ 'Name user must be only letters !'
       end
```

```
              --                                        -- *** Checking if the email is written correctly ***

         if ( @User_Email like '%[^a-z,0-9,@,.,!,#,$,%%,&,'',*,+,--,/,=,?,^,_,`,{,|,},~]%' --First Carat ^
means Not these characters in the LIKE clause. The list is the valid email characters.

               --an email format _@__.__
               or @User_Email not like '%_@_%_.[a-z0-9][a-z]%'

              --an email does not start / end at .
              --an email does not contain a sequence of @ / .
            Or @User_Email like '%@%@%'
         Or @User_Email like '%..%'
         Or @User_Email like '.%'
         Or @User_Email like '%.'
        )
       begin
              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')



                     SET @ERROR =  'The email is not written correctly !'

              else                        -- linebreaks ->
                     SET @ERROR += CHAR(13)+CHAR(10)+ 'The email is not written correctly !'
       end

       --                                                              --- Check - A password must be 10 characters ,
contains a number, an uppercase letter, a lowercase letter, and a special character ***

       if(LEN(@User_Password) <> 10 )
       begin
              -- *** Checking if errors are already written ***
                     if (@ERROR IS NULL or @ERROR = '')



                            SET @ERROR =  'Password must be 10 characters in length !'
                     else                 -- linebreaks ->
                            SET @ERROR += CHAR(13)+CHAR(10)+ 'Password must be 10 characters in length !'
       end



                                          --We will use a function PATINDEX
                                          -- to check if there are values (the index is returned if there
is)
                                          -- COLLATE Latin1_General_100_BIN2 : binary collation
(Latin1_General_100_BIN2).
                                                                                                        --
binary collations sort each case separately (like this: AB....YZ...ab...yz).
                                                                                                        --
Other collations tend to intermingle the uppercase and lowercase letters (like this: AaBb...YyZz),
                                                                                                        --
which would therefore match both uppercase and lowercase characters.

       if (PATINDEX('%[A-Z]%',@User_Password COLLATE Latin1_General_100_BIN2) = 0)
       begin

              -- *** Checking if errors are already written ***
              if (@ERROR IS NULL or @ERROR = '')



                     SET @ERROR =  'Password must contain  an uppercase letter !'
              else                        -- linebreaks ->
                     SET @ERROR +=  CHAR(13)+CHAR(10)+'Password must contain  an uppercase letter !'
       end
```

```
	if(PATINDEX('%[a-z]%',@User_Password COLLATE Latin1_General_100_BIN2) = 0)
	begin

		-- *** Checking if errors are already written ***
		if (@ERROR IS NULL or @ERROR = '')



			SET @ERROR =  'Password must contain a lowercase letter !'
		else                          -- linebreaks ->
			SET @ERROR += CHAR(13)+CHAR(10)+ 'Password must contain a lowercase letter !'
	end

	if(@User_Password not like '%[-+_!@#$%^&*.,?~^(){}=]%')
	begin

		-- *** Checking if errors are already written ***
		if (@ERROR IS NULL or @ERROR = '')



			SET @ERROR =  'Password must contain a special character !'
		else                   -- linebreaks ->
			SET @ERROR +=  CHAR(13)+CHAR(10)+'Password must contain a special character !'
	end
	RETURN @ERROR

END
```

Stored Procurers:

```
USE [Library]
GO
/****** Object:  StoredProcedure [dbo].[name]    10:25:42 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:           <Daniel Artzi>
-- Create date: <07/09/2022>
-- Description:      <Add a stored >
-- =============================================
```

```sql
ALTER PROCEDURE [dbo].[addNewBook]
       -- parameters for the new book
       (@newBook_Code nchar(13),
        @newBook_Title nvarchar(20),
        @newBook_FirstName_Author nvarchar(20) ,
        @newBook_LastName_Author nvarchar(20),
        @newBook_PublicationDate date,
        @newBook_Category nvarchar(25),
        @newBook_SecondaryCategory nvarchar(35) = null,
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                            -- *** Checking if the code is already saved ***

              if EXISTS(select top 1 * from Book with(nolock) where Code = @newBook_Code)
              begin
                     SET @ERROR =  'Book with this code already exists !'  + CHAR(13)+CHAR(10)
              end

              else
              begin
                                                   --~~   We will go into in-depth tests ~~

                     set @ERROR =
[dbo].[Validation_CheckBook](@newBook_Code,@newBook_Title,@newBook_FirstName_Author,@newBook_LastName_Auth
or,@newBook_PublicationDate,@newBook_Category,@newBook_SecondaryCategory)

                     if(@ERROR IS NULL or @ERROR = '')
                     begin

                            set rowcount 1
                            INSERT INTO Book(Code, Title, FirstName_Author, LastName_Author,
PublicationDate, Category, SecondaryCategory)
                            VALUES (@newBook_Code, @newBook_Title, @newBook_FirstName_Author,
@newBook_LastName_Author, @newBook_PublicationDate, @newBook_Category, @newBook_SecondaryCategory);
                            set rowcount 0
                     end
              end
END
```

```sql
ALTER PROCEDURE [dbo].[addNewExistingCategory]
       -- parameters for the new book
       (@newCategory nvarchar(25),
        @newSecondaryCategory nvarchar(35),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;
                                                                              --~~   We will go into in-depth
tests ~~
       set @ERROR = [dbo].[Validation_CheckExistingCategories](@newCategory,@newSecondaryCategory)

       if(@ERROR IS NULL or @ERROR = '')
              begin

                     set rowcount 1
                     INSERT INTO ExistingCategories(Category,SecondaryCategory)
                               VALUES (@newCategory, @newSecondaryCategory);
                     set rowcount 0
              end
END
```

```
ALTER PROCEDURE [dbo].[addNewBorrow]
       -- parameters for the new book
       (@newBorrow_Code nchar(13),
        @newBorrow_Id nchar(9),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;


                        -- *** Checking if the borrow with the same values already exists ***

       if EXISTS(select top 1 * from Borrows with(nolock) where Code = @newBorrow_Code and id = @newBorrow_Id
)
       begin
              SET @ERROR =  'Berry mice some questions! There is no choice, the loan already exists in the
system ...'  + CHAR(13)+CHAR(10)
       end

       else
       begin
                                                                               --~~   We will go into in-depth
tests ~~
              set @ERROR = [dbo].[Validation_CheckBorrow](@newBorrow_Code, @newBorrow_Id)

              if(@ERROR IS NULL or @ERROR = '')
              begin

                     set rowcount 1
                     INSERT INTO Borrows(Code,id)
                               VALUES (@newBorrow_Code,@newBorrow_Id);
                     set rowcount 0
              end

       end
```

```
ALTER PROCEDURE [dbo].[addNewUser]
       -- parameters for the new book
       (@newUser_id nchar(9),
        @newUser_FirstName nvarchar(20),
        @newUser_LastName nvarchar(20) ,
        @newUser_Type bit,
        @newUser_Email nvarchar(20),
        @newUser_Password nchar(10),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;


                                               -- *** Checking if the id is already saved ***

       if EXISTS(select top 1 * from Users with(nolock) where id = @newUser_id)
       begin
              SET @ERROR =  'User with this id already exists !'  + CHAR(13)+CHAR(10)
       end

              else
              begin
                                               --~~   We will go into in-depth tests ~~

                     set @ERROR =
[dbo].[Validation_CheckUser](@newUser_id,@newUser_FirstName,@newUser_LastName,@newUser_Type,@newUser_Email,@
newUser_Password)

                     if(@ERROR IS NULL or @ERROR = '')
                     begin

                            set rowcount 1
                            INSERT INTO Users(id,FirstName,LastName,[Type],Email,[Password])
                                   VALUES (@newUser_id, @newUser_FirstName, @newUser_LastName, @newUser_Type,
@newUser_Email, @newUser_Password);
                            set rowcount 0
                     end
              end
End
```

```sql
ALTER PROCEDURE [dbo].[deleteSelectedBook]
       (@selectedBook_Code nchar (13),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                          -- *** Checking if books exist ***

       if NOT EXISTS (select top 1 * from book with(nolock) where Code = @selectedBook_Code )
       begin
              SET @ERROR =  'Sorry, no book with this code was found :< !'
       end

       else
       begin

              set rowcount 1
              DELETE From Book Where Code = @selectedBook_Code
              set rowcount 0

       end
END
```

```sql
ALTER PROCEDURE [dbo].[deleteSelectedBorrow]
       -- parameters for the new day
       (@selectedCode nchar (13),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

   -- *** Checking if the borrow with the same values already exists ***

       if Not EXISTS(select top 1 * from Borrows with(nolock) where Code = @selectedCode )
       begin
              SET @ERROR =  'I am the number 1 producer of the loans, and unfortunately there is no such 
borrow :O'
       End

       else
       begin

              set rowcount 1
              DELETE From Borrows
              Where Code = @selectedCode
              set rowcount 0

       end
END
```

```sql
ALTER PROCEDURE [dbo].[deleteSelectedExistingCategory]
       -- parameters for the new day
       (@selectedCategory nvarchar (25),
        @selectedSecondaryCategory nvarchar (35),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                        -- *** Checking if Existing Category exist ***

       if NOT EXISTS (select top 1 * from ExistingCategories with(nolock) where Category = @selectedCategory
and SecondaryCategory = @selectedSecondaryCategory )
       begin
              SET @ERROR =  'Sorry, No such categories were found :L !'
       end

       else
       begin

              set rowcount 1
              DELETE From ExistingCategories
              Where Category = @selectedCategory and SecondaryCategory = @selectedSecondaryCategory
              set rowcount 0

       end
END
```

```sql
ALTER PROCEDURE [dbo].[deleteSelectedUser]
       -- parameters for the new day
       (@selectedUser_Id nchar (9),
        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                  -- *** Checking if the id does not exist ***

       if Not EXISTS(select top 1 * from Users with(nolock) where Id = @selectedUser_Id)
       begin
              SET @ERROR =  'Sorry, no user with this id was found :<'  + CHAR(13)+CHAR(10)
       end

       else
       begin

              set rowcount 1
              DELETE From Users Where id = @selectedUser_Id
              set rowcount 0

       end
END
```

```
ALTER PROCEDURE [dbo].[getBooks]
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                           -- *** Checking if books exist ***

       if NOT EXISTS (select * from Book with(nolock))
       begin
              SET @ERROR =  'It is not possible! You didn`t keep a single book @#@'
       end

       else
       begin
               select * from Book with(nolock)
                                               -- *** A representative according to the publication
date of the books from day to the past ***
               ORDER BY PublicationDate DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @booksCount int

       SELECT @booksCount = count(*) FROM Book

       RETURN @booksCount

END
```

```sql
ALTER PROCEDURE [dbo].[getBorrows]
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if borrows exist ***

       if NOT EXISTS (select * from Borrows with(nolock))
       begin
              SET @ERROR =  'It is not possible! You didn`t keep a single borrow @#@'
       end

       else
       begin
               select Book.*, us.* from Borrows borrow with(nolock)
               INNER JOIN Users us on borrow.Id=us.Id
               INNER JOIN Book book on book.Code = borrow.Code
                                                     -- *** We would like a display according to the id ***
               ORDER BY borrow.Id;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @borrowsCount int

       SELECT @borrowsCount = count(*) FROM Book

       RETURN @borrowsCount

END
```

```sql
ALTER PROCEDURE [dbo].[getExistingCategories]
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if category exist ***

       if NOT EXISTS (select * from ExistingCategories with(nolock))
       begin
              SET @ERROR =  'It is not possible! You didn`t keep a single Category @#@'
       end

       else
       begin
               select Category from ExistingCategories with(nolock)
                                                     -- *** We would like a display of categories by main
category  ***
               ORDER BY Category;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @categorieCount int

       SELECT @categorieCount = count(*) FROM Book

       RETURN @categorieCount

END
```

```
ALTER PROCEDURE [dbo].[getUsers]
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if users exist ***

       if NOT EXISTS (select * from Users with(nolock))
       begin
              SET @ERROR =  'It is not possible! You didn`t keep a single user @#@'
       end

       else
       begin
               select * from Users with(nolock)
                                                     -- *** Representative according to the order of A and B
***
               ORDER BY FirstName,LastName DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @usersCount int

       SELECT @usersCount = count(*) FROM Book

       RETURN @usersCount

END
```

```
ALTER PROCEDURE [dbo].[ShowFromBook_BookFromSpecificCode]
        @code nchar(13),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if code exist ***

       if NOT EXISTS (select * from Book with(nolock) where Code = @code)
       begin
              SET @ERROR =  'God of all shifra! code not found @#@'
       end


       else
       begin
               select top 1 * from Book with(nolock)
               where Code = @code
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int

       SELECT @BooksCount = count(*) FROM Book

       RETURN @BooksCount

END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromBook_BooksFromCategory]
        @category nvarchar(25),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                          -- *** Checking if category exist ***

       if NOT EXISTS (select * from Book with(nolock) where Category like '%'+@Category+'%')
       begin
              SET @ERROR =  'God of all shifra! category not found @#@'
       end

       else
       begin
               select * from Book with(nolock)
               where Category like '%'+@Category+'%'
               -- *** A representative according to the publication date of the books from day to the past ***
               ORDER BY PublicationDate DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int

       SELECT @BooksCount = count(*) FROM Book

       RETURN @BooksCount

END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromBook_BooksFromFirstName_Author]
        @firstName_Author nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                          -- *** Checking if firstName- author exist ***

       if NOT EXISTS (select * from Book with(nolock) where FirstName_Author like '%'+@FirstName_Author+'%')
       begin
              SET @ERROR =  'God of all shifra! firstName - Author not found @#@'
       end

       else
       begin
               select * from Book with(nolock)
               where FirstName_Author like '%'+@FirstName_Author+'%'
               -- *** A representative according to the publication date of the books from day to the past ***
               ORDER BY PublicationDate DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int

       SELECT @BooksCount = count(*) FROM Book

       RETURN @BooksCount

END
```

```
ALTER PROCEDURE [dbo].[ShowFromBook_BooksFromLastName_Author]
        @lastName_Author nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                           -- *** Checking if lastName - Author exist ***

       if NOT EXISTS (select * from Book with(nolock) where LastName_Author like '%'+@LastName_Author+'%')
       begin
              SET @ERROR =  'God of all shifra! title not found @#@'
       end

       else
       begin
               select * from Book with(nolock)
               where LastName_Author like '%'+@LastName_Author+'%'
               -- *** A representative according to the publication date of the books from day to the past
***
               ORDER BY PublicationDate DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int

       SELECT @BooksCount = count(*) FROM Book

       RETURN @BooksCount
END
```

```
ALTER PROCEDURE [dbo].[ShowFromBook_BooksFromName_Author]
        @firstName_Author nvarchar(20),
        @lastName_Author nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN
       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;
                                           -- *** Checking if name exist ***

       if NOT EXISTS (select * from Book with(nolock) where FirstName_Author like '%'+@FirstName_Author+'%'
and LastName_Author like '%'+@LastName_Author+'%' )
       begin
              SET @ERROR =  'God of all shifra! name - Author not found @#@'
       end

       else
       begin
               select * from Book with(nolock)
               where FirstName_Author like '%'+@FirstName_Author+'%' and LastName_Author like
'%'+@LastName_Author+'%'
               -- *** A representative according to the publication date of the books from day to the past
***
               ORDER BY PublicationDate DESC;
       end
          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int

       SELECT @BooksCount = count(*) FROM Book

       RETURN @BooksCount

END
```

```
ALTER PROCEDURE [dbo].[ShowFromBook_BooksFromPublicationYear]
        @publicationYear int,
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

       if(@publicationYear < 0 or @publicationYear > 9999)
       begin
              SET @ERROR =  'Do you want to travel in time? This year makes no sense @#@'
       end
                                                -- *** Checking if year exist ***

       else if NOT EXISTS (select * from Book with(nolock) where year(PublicationDate) = @publicationYear)
       begin
              SET @ERROR =  'God of all shifra! publication year not found @#@'
       end

       else
       begin
               select * from Book with(nolock)
               where year(PublicationDate) = @publicationYear
               -- *** A representative according to the publication date of the books from day to the past
***
               ORDER BY PublicationDate DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int
       SELECT @BooksCount = count(*) FROM Book
       RETURN @BooksCount
END
```

```
ALTER PROCEDURE [dbo].[ShowFromBook_BooksFromTitle]
        @title nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN
       --message that indicates the number of rows --that are affected by the T-SQL statement --is not
returned as part of the results.
       SET NOCOUNT ON;
                                                -- *** Checking if title exist ***

       if NOT EXISTS (select * from Book with(nolock) where Title like '%'+@title+'%')
       begin
              SET @ERROR =  'God of all shifra! title not found @#@'
       end
                                         -- *** Checking if all the values are correct and no error message
was generated ***

       else
       begin
               select * from Book with(nolock)
               where Title like '%'+@title+'%'
               -- *** A representative according to the publication date of the books from day to the past
***
               ORDER BY PublicationDate DESC;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @BooksCount int
       SELECT @BooksCount = count(*) FROM Book
       RETURN @BooksCount

END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromBorrow_byUserCheckType]
        @id nchar(13),
        @typeUser bit,
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;
                                                  -- *** Checking if library worker ***
       if(@typeUser <> 0)
       begin
              if not exists(select top 1 * from Users with(nolock) where id = @id)
              begin
                     set @ERROR = 'OOO This is not your ID ! alarm alarm !'
              end

              else if not exists(select * from Borrows where id = @id)
              begin
                     set @ERROR = 'The user did not lend books sorrry ..'
              end
       end

                             -- *** Checking if id exists  ***
       else if not exists(select * from Borrows where id = @id)
       begin
              set @ERROR = 'The user did not lend books sorrry ..'
       end

       if(@ERROR IS NULL or @ERROR = '')
       begin
               select Book.* from Borrows borrow with(nolock)
               INNER JOIN Book book on book.Code = borrow.Code
               where borrow.Id = @id
                                                      -- *** We would like a display according to the id ***
               ORDER BY borrow.Id;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @borrowsCount int

       SELECT @borrowsCount = count(*) FROM Book

       RETURN @borrowsCount

END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromBorrow_SpecificBook]
        @codeBook nchar(13),
        @ERROR nvarchar(500) OUT
AS
BEGIN
       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if borrows exist ***

       if NOT EXISTS (select * from Borrows with(nolock) where Code = @codeBook)
       begin
              SET @ERROR =  'God of all shifra! No one borrowed the book @#@'
       end

       else
       begin
               select Book.* from Borrows borrow with(nolock)
               INNER JOIN Book book on book.Code = borrow.Code
               where borrow.Code = @codeBook
                                                     -- *** We would like a display according to the id ***
               ORDER BY borrow.Id;
       end

          ----can return only integer values -> return the number of books we saved
       DECLARE @borrowsCount int
       SELECT @borrowsCount = count(*) FROM Book
       RETURN @borrowsCount
END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromBorrow_User'sBorrows]
        @idUser nchar(9),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if id - borrows exist ***

       if NOT EXISTS (select * from Borrows with(nolock) where Id = @idUser)
       begin
              SET @ERROR =  'God of all shifra! The user has not borrowed any books @#@'
       end

       else
       begin
               select Book.*, us.* from Borrows borrow with(nolock)
               INNER JOIN Users us on borrow.Id=us.Id
               INNER JOIN Book book on book.Code = borrow.Code
               where borrow.Id = @idUser
                                                     -- *** We would like a display according to the id ***
               ORDER BY borrow.Id;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @borrowsCount int
       SELECT @borrowsCount = count(*) FROM Book
       RETURN @borrowsCount
END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromExistingCategories_SubcategoryFromCategory]
        @category nvarchar(25),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                          -- *** Checking if Category exist ***

       if NOT EXISTS (select * from ExistingCategories with(nolock) where Category like '%'+@category+'%')
       begin
              SET @ERROR =  'God of all shifra! Category not found @#@'
       end

       else
       begin
               select SecondaryCategory from ExistingCategories with(nolock)
               where Category = @category
                                              -- *** We would like a display according to the
secondary category ***
               ORDER BY SecondaryCategory;
       end

          ----can return only integer values -> return the number of books we saved
       DECLARE @SecondaryCategoriesCount int
       SELECT @SecondaryCategoriesCount = count(*) FROM Book
       RETURN @SecondaryCategoriesCount
END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromExistingCategories_SubcategoryFromSpecificCategory]
        @category nvarchar(25),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                          -- *** Checking if Category exist ***

       if NOT EXISTS (select * from ExistingCategories with(nolock) where Category = @category)
       begin
              SET @ERROR =  'God of all shifra! Category not found @#@'
       end

       else
       begin
               select SecondaryCategory from ExistingCategories with(nolock)
               where Category = @category
                                              -- *** We would like a display according to the
secondary category ***
               ORDER BY SecondaryCategory;
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @SecondaryCategoriesCount int

       SELECT @SecondaryCategoriesCount = count(*) FROM Book

       RETURN @SecondaryCategoriesCount

END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromUser_UserFromSpecific_Id_Email_Password]
        @id nchar(9),
        @email nvarchar(50),
        @password nchar(10),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                  -- *** Checking if id exist ***
       if NOT EXISTS (select * from Users with(nolock) where Id = @id)
       begin
              SET @ERROR =  'God of all shifra! id not found @#@'
       end

                                                  -- *** Checking if email or password correct ***
       else
       begin
              if NOT EXISTS (select * from Users with(nolock) where Id = @id and Email = @email)
              begin
                     SET @ERROR =  'God of all shifra! email not correct @#@'
              end
              if NOT EXISTS (select * from Users with(nolock) where Id = @id and Password = @password)
              begin
                     if (@ERROR IS NULL or @ERROR = '')


                     -- linebreaks ->
                            SET @ERROR =  'God of all shifra! password not correct @#@'
                     else
                            SET @ERROR +=   CHAR(13)+CHAR(10) + 'God of all shifra! password not correct
@#@'
              end
       end

       if (@ERROR IS NULL or @ERROR = '')
       begin
               select top 1 * from Users with(nolock)
               where id = @id
                     and Email = @email
                     and Password = @password
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @UsersCount int

       SELECT @UsersCount = count(*) FROM Book

       RETURN @UsersCount

END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromUser_UserFromSpecificEmail]
        @email nvarchar(50),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;
                                          -- *** Checking if Email exist ***

       if NOT EXISTS (select * from Users with(nolock) where Email = @email)
       begin
              SET @ERROR =  'God of all shifra! email not found @#@'
       end

       else
       begin
               select top 1 * from Users with(nolock)
               where Email = @email
       end

          ----can return only integer values -> return the number of books we saved
       DECLARE @UsersCount int
       SELECT @UsersCount = count(*) FROM Book
       RETURN @UsersCount
END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromUser_UserFromSpecificId]
        @id nchar(13),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;
                                          -- *** Checking if borrows exist ***

       if NOT EXISTS (select * from Users with(nolock) where Id = @id)
       begin
              SET @ERROR =  'God of all shifra! id not found @#@'
       end

       else
       begin
               select top 1 * from Users with(nolock)
               where Id = @id
       end

          ----can return only integer values -> return the number of books we saved
       DECLARE @UsersCount int
       SELECT @UsersCount = count(*) FROM Book
       RETURN @UsersCount
END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromUser_UserFromSpecificPassword]
        @password nchar(10),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;
                                                 -- *** Checking if borrows exist ***

       if NOT EXISTS (select * from Users with(nolock) where Password = @password)
       begin
              SET @ERROR =  'God of all shifra! password not found @#@'
       end

       else
       begin
               select top 1 * from Users with(nolock)
               where Password = @password
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @UsersCount int
       SELECT @UsersCount = count(*) FROM Book
       RETURN @UsersCount
END
```

```sql
ALTER PROCEDURE [dbo].[ShowFromUser_UsersFromFirstName]
        @firstName nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                 -- *** Checking if First Name exist ***

       if NOT EXISTS (select * from Users with(nolock) where FirstName like '%'+@firstName+'%')
       begin
              SET @ERROR =  'God of all shifra! name not found @#@'
       end


       else
       begin
               select * from Users with(nolock)
               where FirstName like '%'+@firstName+'%'
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @UsersCount int

       SELECT @UsersCount = count(*) FROM Book

       RETURN @UsersCount

END
```

```
ALTER PROCEDURE [dbo].[ShowFromUser_UsersFromLastName]
        @lastName nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                           -- *** Checking if Last Name exist ***

       if NOT EXISTS (select * from Users with(nolock) where LastName like '%'+@lastName+'%')
       begin
              SET @ERROR =  'God of all shifra! name not found @#@'
       end

       else
       begin
               select * from Users with(nolock)
               where LastName like '%'+@lastName+'%'
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @UsersCount int
       SELECT @UsersCount = count(*) FROM Book
       RETURN @UsersCount
END
```

```
ALTER PROCEDURE [dbo].[ShowFromUser_UsersFromName]
        @firstName nvarchar(20),
        @lastName nvarchar(20),
        @ERROR nvarchar(500) OUT
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                           -- *** Checking if name exist ***

       if NOT EXISTS (select * from Users with(nolock) where FirstName like  '%'+@lastName+'%' and
LastName like '%'+@lastName+'%')
       begin
              SET @ERROR =  'God of all shifra! name not found @#@'
       end


       else
       begin
               select * from Users with(nolock)
               where FirstName like  '%'+@lastName+'%' and LastName like '%'+@lastName+'%'
       end

          ----can return only integer values -> return the number of books we saved

       DECLARE @UsersCount int

       SELECT @UsersCount = count(*) FROM Book

       RETURN @UsersCount

END
```

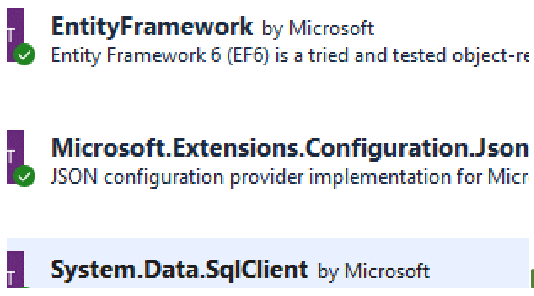
```sql
ALTER PROCEDURE [dbo].[updateSelectedBook]
       -- parameters for the new book
       (@updateBook_Code nchar(13),
        @updateBook_Title nvarchar(20),
        @updateBook_FirstName_Author nvarchar(20) ,
        @updateBook_LastName_Author nvarchar(20),
        @updateBook_PublicationDate date,
        @updateBook_Category nvarchar(25),
        @updateBook_SecondaryCategory nvarchar(35) = null,
        @ERROR nvarchar(500) OUT
        )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;


                                           -- *** Checking if the code does not exist ***

       if Not EXISTS(select top 1 * from Book with(nolock) where Code = @updateBook_Code)
       begin
              SET @ERROR =  'Sorry, no book with this code was found :<'
       end

       else
       begin
                                                                                 --~~   We will go into in-depth
tests ~~

              set @ERROR =
[dbo].[Validation_CheckBook](@updateBook_Code,@updateBook_Title,@updateBook_FirstName_Author,@updateBook_La
stName_Author,@updateBook_PublicationDate,@updateBook_Category,@updateBook_SecondaryCategory)

              if(@ERROR IS NULL or @ERROR = '')
              begin

                     set rowcount 1
                     UPDATE Book
                     Set Title = @updateBook_Title,
                         FirstName_Author = @updateBook_FirstName_Author,
                         LastName_Author = @updateBook_LastName_Author,
                         PublicationDate = @updateBook_PublicationDate,
                         Category = @updateBook_Category,
                         SecondaryCategory = @updateBook_SecondaryCategory
                     Where Code = @updateBook_Code
                     set rowcount 0
              end
       end

END
```

```sql
ALTER PROCEDURE [dbo].[updateSelectedBorrow]
       -- parameters for the new book
       (@updateCode nchar(13),
        @updateId nchar(9),

        @ERROR nvarchar(500) OUT )
AS
BEGIN
       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

              -- *** Checking if the borrow with the same values already exists ***

       if Not EXISTS(select top 1 * from Borrows with(nolock) where Code = @updateCode)
       begin
              SET @ERROR =  'I am the number 1 producer of the loans, and unfortunately there is no such
borrow :O'  + CHAR(13)+CHAR(10)
       end

       else
       begin
                                                                  --~~   We will go into in-depth tests ~~
              set @ERROR = [dbo].[Validation_CheckBorrow](@updateCode, @updateId)

              if(@ERROR IS NULL or @ERROR = '')
              begin
                     set rowcount 1
                     UPDATE Borrows
                     Set id = @updateId
                     Where Code = @updateCode
                     set rowcount 0
              end
       end
END
```

```sql
ALTER PROCEDURE [dbo].[updateSelectedExistingCategory]
       -- parameters for the new book
       (@currentCategory nvarchar(25),
        @currentSecondaryCategory nvarchar(35),
        @updateCategory nvarchar(25),
        @updateSecondaryCategory nvarchar(35),

        @ERROR nvarchar(500) OUT )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;


                                                  -- *** Checking if the Category does not exist ***

       if Not EXISTS(select top 1 * from ExistingCategories with(nolock) where Category =
@currentCategory and SecondaryCategory = @currentSecondaryCategory)
       begin
              SET @ERROR =  'No such categories were found :L'  + CHAR(13)+CHAR(10)
       end

       --                                         -- *** Check for values:
       --                                                       --- only letters category ***

       else
       begin
                                                                            --~~   We will go into in-
depth tests ~~

              set @ERROR =
[dbo].[Validation_CheckExistingCategories](@updateCategory,@updateSecondaryCategory)

              if(@ERROR IS NULL or @ERROR = '')
              begin

                                                  --     Since all values are keys
                                                  --To know which values to change
                                                  --We change all the key fields that currently exist to
new ones

                     set rowcount 1
                     UPDATE ExistingCategories
                     Set Category = @updateCategory,
                         SecondaryCategory = @updateSecondaryCategory
                     Where Category = @currentCategory and SecondaryCategory = @currentSecondaryCategory
                     set rowcount 0

              end
       end

END
```

```sql
ALTER PROCEDURE [dbo].[updateSelectedUser]
       -- parameters for the new book
       (@updateUser_id nchar(9),
        @updateUser_FirstName nvarchar(20),
        @updateUser_LastName nvarchar(20) ,
        @updateUser_Type bit,
        @updateUser_Email nvarchar(20),
        @updateUser_Password nchar(10),
        @ERROR nvarchar(500) OUT
       )
AS
BEGIN

       --message that indicates the number of rows
       --that are affected by the T-SQL statement
       --is not returned as part of the results.
       SET NOCOUNT ON;

                                                   -- *** Checking if the id does not exist ***

       if Not EXISTS(select top 1 * from Users with(nolock) where Id = @updateUser_id)
       begin
              SET @ERROR =  'Sorry, no user with this id was found :<'
       end

       else
       begin
                                                                                      --~~   We will go into in-depth
tests ~~

              set @ERROR =
[dbo].[Validation_CheckUser](@updateUser_id,@updateUser_FirstName,@updateUser_LastName,@updateUser_Type,@up
dateUser_Email,@updateUser_Password)

              if(@ERROR IS NULL or @ERROR = '')
              begin

                     set rowcount 1
                     UPDATE Users
                     Set FirstName = @updateUser_FirstName,
                         LastName = @updateUser_LastName,
                         Type = @updateUser_Type,
                         Email = @updateUser_Email,
                         Password = @updateUser_Password
                     Where Id = @updateUser_id
                     set rowcount 0
              end
       end
END
```

צד C# - data access layer :



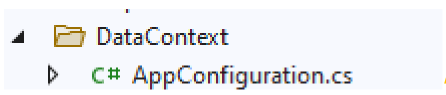
EntityFramework by Microsoft
Entity Framework 6 (EF6) is a tried and tested object-re

Microsoft.Extensions.Configuration.Json
JSON configuration provider implementation for Micr

System.Data.SqlClient by Microsoft

EntityFramework **Microsoft.Extensions.Configuration.Json**

Syste.Data.SqlClient

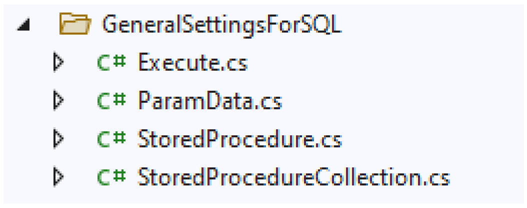


DataContext
C# AppConfiguration.cs

DataContext:

`AppConfiguration.cs:`

```
using Microsoft.Extensions.Configuration;

public class AppConfiguration
    {

        // We will create a class that receives a connection path to the database dynamically,
        //according to the main folder path
        public AppConfiguration()
        {
            // There can be a situation where the library will be used and the path will not be to
windows app, i.e. build
            try
            {
                ConfigurationBuilder configBuildr = new ConfigurationBuilder();
                //the url end at bin\Debug\net6.0-windows\ -> want parent
                DirectoryInfo pathToApp =
Directory.GetParent(Directory.GetCurrentDirectory())!.Parent!.Parent!;

                string path = Path.Combine(pathToApp!.FullName, "appsettings.json");
                configBuildr.AddJsonFile(path, false); // Not Optional Mast Be There
                IConfigurationRoot root = configBuildr.Build();
                IConfigurationSection appSeting =
root.GetSection("ConnectionStrings:DefaultConnestion");
                sqlConectionString = appSeting.Value;
            }
            catch (Exception ex)
            {
                sqlConectionString = "";
                Console.WriteLine(ex.Message);
            }
        }

        public string sqlConectionString { get; set; }

    }
```


GeneralSettingsForSQL
Execute.cs
ParamData.cs
StoredProcedure.cs
StoredProcedureCollection.cs

GeneralSettingsForSQL:

ParamData.cs:

```
using System.Data;
using System.Runtime.InteropServices;



// We create structure to display the structure of a parameter ->
    // { parameter name, parameter value , parameter direction, size and data type }

    public struct ParamData
    {
        public string pName;
        public SqlDbType pDataType;
        public object? pValue; // can be number , string , date
        public ParameterDirection pDirection;
        public int? pSize;

        public ParamData(string pName, SqlDbType pDataType, object? pValue, ParameterDirection
pDirection, [OptionalAttribute] int? size)
        {
            this.pName = pName;
            this.pDataType = pDataType;
            this.pValue = pValue;
            this.pDirection = pDirection;
            this.pSize = size;

        }
    }
```

StoredProcedure.cs:

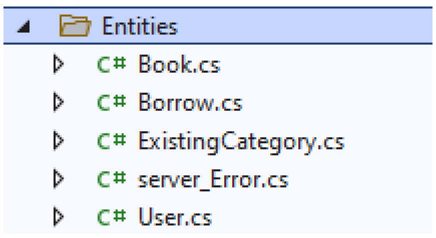
```
using System.Data;
using System.Runtime.InteropServices;



//A class that will represent a procedure,
    //with a list of parameters and the name of the procedure
    public class StoredProcedure
    {
        List<ParamData> sParams;
        public string ProcName;

        public StoredProcedure()
        {
            sParams = new List<ParamData>();
            ProcName = "";
        }
        public void SetParam(string pName, SqlDbType pDataType, object? pValue, ParameterDirection
pDirection, [OptionalAttribute] int? pSize)
        {

            ParamData pData = new ParamData(pName, pDataType, pValue, pDirection, pSize);
            sParams.Add(pData);
        }


        //We will add a function to get the
        //list of parameters, parameter by name

        public List<ParamData>? GetParams()
        {
            if (sParams.Count != 0)
            {
                return sParams;
            }
            else
            {
                return null;
            }
        }


        public ParamData? GetParamByName(string pNameGet)
        {
            if (sParams.Count != 0)
            {
                foreach (ParamData pData in sParams)
                {
                    if (pData.pName == pNameGet)
                    {
                        return pData;
                    }
                }
                return null;

            }
            else
            {
                return null;
            }
        }
    }
```

StoredProcedureCollection.cs:

```
//We will create a class that will be a collection
    //of procedures that we will define,
    //with add and remove functions as needed

    public class StoredProcedureCollection
    {

        public List<StoredProcedure> listStoredProcedures;
        public StoredProcedureCollection()
        {
            listStoredProcedures = new List<StoredProcedure>();
        }

        public void add(StoredProcedure value)
        {
            listStoredProcedures.Add(value);
        }

        public void remove(int index)
        {
            if (index > listStoredProcedures.Count - 1 || index < 0)
            {
                Console.WriteLine("No data to remove");
            }
            else
            {
                listStoredProcedures.RemoveAt(index);
            }
        }

        //In addition there will be a function to receive a specific procedure

        public StoredProcedure getProcedureByIndex(int Index)
        {
            //return (StoredProcedure)listStoredProcedures[Index];
            return listStoredProcedures[Index];
        }

    }
```

StoredProcedureCollection.cs:

```
using System.Collections;using System.Data;using System.Data.SqlClient;
     public class Execute
    {
        // return -> error message / boolean ( true )
        public static object ExecuteSps(StoredProcedureCollection spCollection, SqlConnection 
Connection)
        {
            try
            {
                // Go over the procedures to be performed
                foreach (StoredProcedure spData in spCollection.listStoredProcedures)
                {
                    SqlCommand cmd = new SqlCommand();
                    if (Connection.State != ConnectionState.Open)
                        Connection.Open();
                    cmd.Connection = Connection;
                    cmd.CommandType = CommandType.StoredProcedure;
                    cmd.CommandText = spData.ProcName;

                    //Go over the parameters of the procedure
                    IEnumerator myEnumerator = spData.GetParams()!.GetEnumerator();
                    int i = 0;
                    while (myEnumerator.MoveNext())
                    {
                        ParamData pData = (ParamData)myEnumerator.Current;
                        cmd.Parameters.Add(pData.pName, pData.pDataType);
                        cmd.Parameters[i].Value = pData.pValue;
                        cmd.Parameters[i].Direction = pData.pDirection;
                        if (pData.pSize.HasValue)
                            cmd.Parameters[i].Size = (int)pData.pSize;
                        i++;
                    }
                    //Carrying out the procedure and checking
                    //whether there was an error during the execution

                    SqlDataReader dr = cmd.ExecuteReader();
                    if (cmd.Parameters["@ERROR"].Value != null &&
cmd.Parameters["@ERROR"].Value.ToString()!.Length > 0)
                    {
                        string message = (string)cmd.Parameters["@ERROR"].Value;

                        // We'll close the connection path so you can read more procedures

                        Connection.Close();
                        return message;
                    }

                    //Checking if there is data
                    else if (dr.HasRows)
                    {
                        SqlDataReader sqlDataReader = (SqlDataReader)dr;
                        var dataTable = new DataTable();
                        dataTable.Load(sqlDataReader);
                        Connection.Close();
                        return dataTable;
                    }
                }
                //Closing the database connection
                Connection.Close();
                return true;
            }
            catch (Exception exc)
            {
                return exc.Message;
            }
        }
    }
```


```
◢ Entities
    ▷ C# Book.cs
    ▷ C# Borrow.cs
    ▷ C# ExistingCategory.cs
    ▷ C# server_Error.cs
    ▷ C# User.cs
```

Entities:

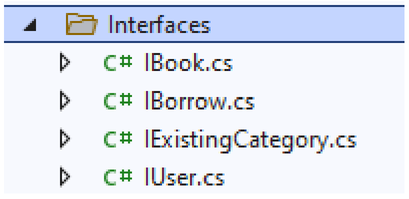
```csharp
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

public class Book
    {
        [Key]
        [StringLength(13)]
        public string Code { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(20)")]
        [MaxLength(20)]
        public string Title { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(20)")]
        [MaxLength(20)]
        public string FirstName_Author { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(20)")]
        [MaxLength(20)]
        public string LastName_Author { get; set; }

        [Required]
        public DateTime PublicationDate { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(25)")]
        [MaxLength(25)]
        public string Category { get; set; }

        [Column(TypeName = "nvarchar(35)")]
        [MaxLength(35)]
        public string SecondaryCategory { get; set; }

        public ICollection<Borrow> borrows { get; set; }

    }
```

```csharp
    public class Borrow
     {
         [Key]
         [Column(Order = 1)]
         [ForeignKey("Book")]
         [StringLength(13)]
         public string Code { get; set; }
         public Book Book { get; set; }


         [Required]
         [Column(Order = 2)]
         [ForeignKey("User")]
         [StringLength(9)]
         public string Id { get; set; }
         public User User { get; set; }


     }
```

```csharp
    public class ExistingCategory
     {
         [Index("CI_Category", IsClustered = true)]
         [Key]
         [Column(Order = 1, TypeName = "nvarchar(25)")]
         [MaxLength(25)]
         public string Category { get; set; }

         [Key]
         [Column(Order = 1 ,TypeName = "nvarchar(35)")]
         [MaxLength(35)]
         public string SecondaryCategory { get; set; }
     }
```

```csharp
public struct Server_Error
    {
        public string? typeRequest { get; set; }
        public string? description { get; set; }

        public string? errorTime { get; set; }
    }
```

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

public class User
    {

        [Key]
        [Column(TypeName = "nchar(9)")]
        [StringLength(9)]
        public string Id { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(20)")]
        [MaxLength(20)]
        public string FirstName { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(20)")]
        [MaxLength(20)]
        public string LastName { get; set; }

        [Required]
        // SQL Server (Type) -> data type = bit
        public bool Type { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(50)")]
        [MaxLength(50)]
        public string Email { get; set; }

        [Required]
        [StringLength(10)]
        public string Password { get; set; }

        public ICollection<Borrow> borrows { get; set; }
    }
```


Interfaces
IBook.cs
IBorrow.cs
IExistingCategory.cs
IUser.cs

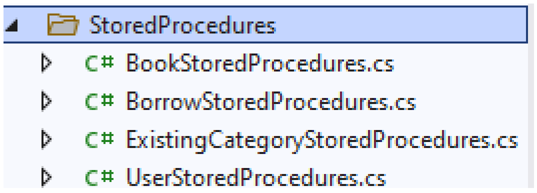
Interfaces:

```
public interface IBook
    {
        // which functions must be used at sql
        public object addNewBook(Book newBook);
        public object deleteSelectedBook(string selectedCode);
        public object getBooks();
        public object ShowFromBook_BookFromSpecificCode(string code);
        public object ShowFromBook_BooksFromCategory(string category);
        public object ShowFromBook_BooksFromFirstName_Author(string firstName_Author);
        public object ShowFromBook_BooksFromLastName_Author(string lastName_Author);
        public object ShowFromBook_BooksFromName_Author(string firstName_Author, string
lastName_Author);
        public object ShowFromBook_BooksFromPublicationYear(int publicationYear);
        public object ShowFromBook_BooksFromTitle(string title);
        public object updateSelectedBook(Book updateBook);
    }
```

```
public interface IBorrow
    {
        // which functions must be used at sql
        public object addNewBorrow(Borrow newBorrow);
        public object deleteSelectedBorrow(string selectedCode);
        public object getBorrows();
        public object ShowFromBorrow_byUserCheckType(bool type, string id);
        public object ShowFromBorrow_SpecificBook(string codeBook);
        public object ShowFromBorrow_UserBorrows(string idUser);
        public object updateSelectedBorrow(Borrow updateBorrow);
    }
```

```
public interface IExistingCategory
    {
        // which functions must be used at sql
        public object addNewExistingCategory(ExistingCategory newExistingCategory);
        public object deleteSelectedExistingCategory(ExistingCategory selectedExistingCategory);
        public object getExistingCategories();
        public object ShowFromExistingCategories_SubcategoryFromSpecificCategory(string category);
        public object ShowFromExistingCategories_SubcategoryFromCategory(string category);
        public object updateSelectedExistingCategory(ExistingCategory currentExistingCategory,
ExistingCategory updateExistingCategory);
    }
```

```
public interface IUser
    {
        // which functions must be used at sql
        public object addNewUser(User newUser);
        public object deleteSelectedUser(string selectedId);
        public object getUsers();
        public object ShowFromUser_UserFromSpecific_Id_Email_Password(string id, string email, string
password);
        public object ShowFromUser_UserFromSpecificEmail(string email);
        public object ShowFromUser_UserFromSpecificId(string id);
        public object ShowFromUser_UserFromSpecificPassword (string password);
        public object ShowFromUser_UsersFromFirstName(string firstName);
        public object ShowFromUser_UsersFromLastName(string lastName);
        public object ShowFromUser_UsersFromName(string firstName, string lastName);

        public object updateSelectedUser(User updateUser);
    }
```



StoredProcedures:

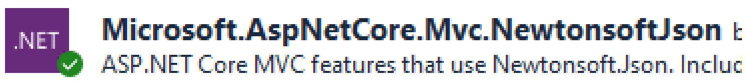
BookStoredProcedures.cs:

```
using DataAccessLayer.DataContext;
using DataAccessLayer.Interfaces;
using System.Data.SqlClient;
using DataAccessLayer.Entities;
using System.Data;
using DataAccessLayer.GeneralSettingsForSQL;

public class BookStoredProcedures : IBook
    {

        //We will connect to the database and run procedures
        private AppConfiguration settings { get; set; }
        private SqlConnection connection { get; set; }

        public BookStoredProcedures()
        {
            settings = new AppConfiguration();
            connection = new SqlConnection();
            connection.ConnectionString = settings.sqlConectionString;
        }

        public object addNewBook(Book newBook)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "addNewBook";
            spData.SetParam("@newBook_Code", SqlDbType.NChar, newBook.Code, ParameterDirection.Input);
            spData.SetParam("@newBook_Title", SqlDbType.NVarChar, newBook.Title,
ParameterDirection.Input);
            spData.SetParam("@newBook_FirstName_Author", SqlDbType.NVarChar, newBook.FirstName_Author,
ParameterDirection.Input);
            spData.SetParam("@newBook_LastName_Author", SqlDbType.NVarChar, newBook.LastName_Author,
ParameterDirection.Input);
            spData.SetParam("@newBook_PublicationDate", SqlDbType.DateTime, newBook.PublicationDate,
ParameterDirection.Input);
            spData.SetParam("@newBook_Category", SqlDbType.NVarChar, newBook.Category,
ParameterDirection.Input);
            spData.SetParam("@newBook_SecondaryCategory", SqlDbType.NVarChar,
newBook.SecondaryCategory, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);

        }

        public object deleteSelectedBook(string selectedCode)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "deleteSelectedBook";
            spData.SetParam("@selectedBook_Code", SqlDbType.NChar, selectedCode,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
        public object getBooks()
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "getBooks";
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBook_BookFromSpecificCode(string code)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BookFromSpecificCode";
            spData.SetParam("@code", SqlDbType.NChar, code, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBook_BooksFromCategory(string category)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BooksFromCategory";
            spData.SetParam("@category", SqlDbType.NVarChar, category, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBook_BooksFromFirstName_Author(string firstName_Author)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BooksFromFirstName_Author";
            spData.SetParam("@firstName_Author", SqlDbType.NVarChar, firstName_Author,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBook_BooksFromLastName_Author(string lastName_Author)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BooksFromLastName_Author";
            spData.SetParam("@lastName_Author", SqlDbType.NVarChar, lastName_Author,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
        public object ShowFromBook_BooksFromName_Author(string firstName_Author, string
lastName_Author)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BooksFromName_Author";
            spData.SetParam("@firstName_Author", SqlDbType.NVarChar, firstName_Author,
ParameterDirection.Input);
            spData.SetParam("@lastName_Author", SqlDbType.NVarChar, lastName_Author,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBook_BooksFromPublicationYear(int publicationYear)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BooksFromPublicationYear";
            spData.SetParam("@publicationYear", SqlDbType.Int, publicationYear,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBook_BooksFromTitle(string title)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBook_BooksFromTitle";
            spData.SetParam("@title", SqlDbType.NVarChar, title, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
        public object updateSelectedBook(Book updateBook)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "updateSelectedBook";
            spData.SetParam("@updateBook_Code", SqlDbType.NChar, updateBook.Code,
ParameterDirection.Input);
            spData.SetParam("@updateBook_Title", SqlDbType.NVarChar, updateBook.Title,
ParameterDirection.Input);
            spData.SetParam("@updateBook_FirstName_Author", SqlDbType.NVarChar,
updateBook.FirstName_Author, ParameterDirection.Input);
            spData.SetParam("@updateBook_LastName_Author", SqlDbType.NVarChar,
updateBook.LastName_Author, ParameterDirection.Input);
            spData.SetParam("@updateBook_PublicationDate", SqlDbType.DateTime,
updateBook.PublicationDate, ParameterDirection.Input);
            spData.SetParam("@updateBook_Category", SqlDbType.NVarChar, updateBook.Category,
ParameterDirection.Input);
            spData.SetParam("@updateBook_SecondaryCategory", SqlDbType.NVarChar,
updateBook.SecondaryCategory, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
    }
```

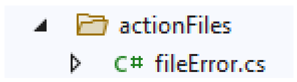
BorrowStoredProcedures.cs

```
using DataAccessLayer.Interfaces;
using DataAccessLayer.DataContext;
using System.Data;
using System.Data.SqlClient;
using DataAccessLayer.Entities;
using DataAccessLayer.GeneralSettingsForSQL;


public class BorrowStoredProcedures : IBorrow
    {


        //We will connect to the database and run procedures
        private AppConfiguration settings { get; set; }
        private SqlConnection connection { get; set; }

        public BorrowStoredProcedures()
        {
            settings = new AppConfiguration();
            connection = new SqlConnection();
            connection.ConnectionString = settings.sqlConectionString;
        }

        public object addNewBorrow(Borrow newBorrow)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "addNewBorrow";
            spData.SetParam("@newBorrow_Code", SqlDbType.NChar, newBorrow.Code,
ParameterDirection.Input);
            spData.SetParam("@newBorrow_Id", SqlDbType.NChar, newBorrow.Id,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);

        }

        public object deleteSelectedBorrow(string selectedCode)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "deleteSelectedBorrow";
            spData.SetParam("@selectedCode", SqlDbType.NVarChar, selectedCode,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object getBorrows()
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "getBorrows";
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
        public object ShowFromBorrow_byUserCheckType(bool type, string id)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBorrow_byUserCheckType";
            spData.SetParam("@type", SqlDbType.Bit, type, ParameterDirection.Input);
            spData.SetParam("@id", SqlDbType.NChar, id, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
        public object ShowFromBorrow_SpecificBook(string codeBook)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBorrow_SpecificBook";
            spData.SetParam("@codeBook", SqlDbType.NChar, codeBook, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object ShowFromBorrow_UserBorrows(string idUser)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "ShowFromBorrow_User'sBorrows";
            spData.SetParam("@idUser", SqlDbType.NChar, idUser, ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }

        public object updateSelectedBorrow(Borrow updateBorrow)
        {
            if (connection.State != ConnectionState.Open)
                connection.Open();
            StoredProcedureCollection spCollection = new StoredProcedureCollection();
            StoredProcedure spData = new StoredProcedure();
            spData.ProcName = "updateSelectedBorrow";
            spData.SetParam("@updateCode", SqlDbType.NChar, updateBorrow.Code,
ParameterDirection.Input);
            spData.SetParam("@updateId", SqlDbType.NChar, updateBorrow.Id,
ParameterDirection.Input);
            spData.SetParam("@ERROR", SqlDbType.NVarChar, "", ParameterDirection.InputOutput, 500);
            spCollection.add(spData);
            return Execute.ExecuteSps(spCollection, connection);
        }
    }
```

צד C# - BusinessLogicLayer :


Microsoft.AspNetCore.Mvc.NewtonsoftJson
ASP.NET Core MVC features that use Newtonsoft.Json. Incluc

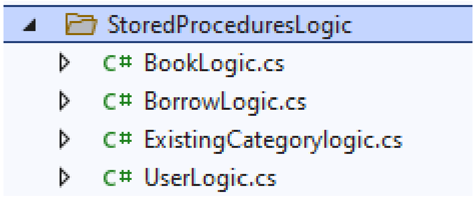
Microsoft.aspnetcore.mvc.newtonsoftjson



ActionFiles:

FileError.cs

```
using DataAccessLayer.Entities;
using Newtonsoft.Json;

public class FileError
    {

        string directoryPath { get; set; }
        string filePath { get; set; }

        public FileError()
        {
            //PresentionLayer
            DirectoryInfo pathToApp =
Directory.GetParent(Directory.GetCurrentDirectory())!.Parent!.Parent!;
            directoryPath = Path.Combine(pathToApp.FullName, "Files");
            filePath = Path.Combine(directoryPath, "Errors.txt");

        }

        //// Create textWriter to add and read errors to file

        public void addError(string type, string desc)
        {

            TextWriter txt = new StreamWriter(filePath, append: true);

            Server_Error newServerErrorObj = new Server_Error()
            {
                typeRequest = type,
                description = desc,
                errorTime = DateTime.Now.ToString("dd'-'MM'-'yyyy' 'HH':'mm':'ss")
            };

            string newServerErrorStr = JsonConvert.SerializeObject(newServerErrorObj);
            txt.WriteLine(newServerErrorStr);
            txt.Close();
        }

        public void getErrors()
        {
            string[] lines = { };
            lines = System.IO.File.ReadAllLines(filePath);
            List<Server_Error> errors = new List<Server_Error>();
            foreach (string line in lines)
            {
                errors.Add(JsonConvert.DeserializeObject<Server_Error>(line)!);
            }
        }
```

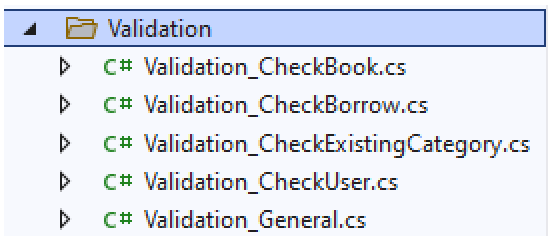

StoredProceduresLogic
BookLogic.cs
BorrowLogic.cs
ExistingCategorylogic.cs
UserLogic.cs

StoredProceduresLogic:

BookLogic.cs

```
using DataAccessLayer.Entities;
using DataAccessLayer.Interfaces;
using BusinessLogicLayer.actionFiles;
using BusinessLogicLayer.Validation;
using System.Data;

public class BookLogic
    {
        // We will implement the functions we defined
        // in the "DataAccessLayer" and check the results
        private IBook _IBook = new DataAccessLayer.StoredProcedures.BookStoredProcedures();
        private FileError _fileError = new FileError();
        public object addNewBook(Book newBook)
        {
            #region Checking the correct input
            string? checkBook = Validation_CheckBook.checkBook(newBook);
            if (checkBook != null)
            {
                return checkBook;
            }
            #endregion

            #region Query execution
            try
            {
                object result = _IBook.addNewBook(newBook);
                // check if return true / message
                if (result.GetType() != typeof(Boolean))
                {
                    return Validation_General.insertErr("", result.ToString()!, "SQL Exception");
                }
                else
                    return "Book successfully added !";
            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }
            #endregion
        }
        public object deleteSelectedBook(string selectedCode)
        {
            #region Checking the correct input

            string? checkValid = Validation_CheckBook.checkCode(selectedCode);
            if (checkValid != null)
            {
                return Validation_General.insertErr("", checkValid, "Client Exception");
            }

            #endregion

            #region Query execution
            try
            {
                object result = _IBook.deleteSelectedBook(selectedCode);
                // check if return true / message
                if (result.GetType() != typeof(Boolean))
                {
                    return Validation_General.insertErr("", result.ToString()!, "SQL Exception");

                }
                else
                    return "Book successfully deleted !";

            }
            catch (Exception ex)
            {
                _fileError.addError("Server Exception", ex.Message);
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }
            #endregion

        }

        public object getBooks()
        {
            try
            {
                object result = _IBook.getBooks()!;
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }
        }
```

```
    public object ShowFromBook_BookFromSpecificCode(string code)
        {
            #region Checking the correct input

            string? checkValid = Validation_CheckBook.checkCode(code);
            if (checkValid != null)
            {
                return Validation_General.insertErr("", checkValid, "Client Exception");
            }

            #endregion

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BookFromSpecificCode(code);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }

        public object ShowFromBook_BooksFromCategory(string category)
        {
            #region Checking the correct input

            string? checkValid = Validation_General.checkOnlyLetter(category);
            if (checkValid != null)
            {
                return Validation_General.insertErr("", checkValid, "Client Exception");
            }

            #endregion

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BooksFromCategory(category);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }
        public object ShowFromBook_BooksFromFirstName_Author(string firstName_Author)
        {
            #region Checking the correct input

            string? checkValid = Validation_General.checkOnlyLetter(firstName_Author);
            if (checkValid != null)
            {
                return Validation_General.insertErr("", checkValid, "Client Exception");
            }

            #endregion

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BooksFromFirstName_Author(firstName_Author);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }

      public object ShowFromBook_BooksFromLastName_Author(string lastName_Author)
        {
            #region Checking the correct input

            string? checkValid = Validation_General.checkOnlyLetter(lastName_Author);
            if (checkValid != null)
            {
                return Validation_General.insertErr("", checkValid, "Client Exception");
            }

            #endregion

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BooksFromLastName_Author(lastName_Author);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }
        public object ShowFromBook_BooksFromName_Author(string firstName_Author, string
lastName_Author)
        {
            #region Checking the correct input
            string errors = "";
            string? checkValid;
            checkValid = Validation_General.checkOnlyLetter(firstName_Author);
            if (checkValid != null)
            {
                errors = Validation_General.insertErr("", checkValid, "Client Exception");
            }
            checkValid = Validation_General.checkOnlyLetter(lastName_Author);
            if (checkValid != null)
            {
                errors = Validation_General.insertErr(errors, checkValid, "Client Exception");
            }

            if (errors != "")
            { return errors; }
            #endregion

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BooksFromName_Author(firstName_Author,
lastName_Author);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }
        public object ShowFromBook_BooksFromPublicationYear(int publicationYear)
        {
            #region Checking the correct input


            if (publicationYear < 0 || publicationYear > 9999)
            {
                return Validation_General.insertErr("", "Do you want to travel in time? This year
makes no sense", "Client Exception");
            }

            #endregion

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BooksFromPublicationYear(publicationYear);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }

        public object ShowFromBook_BooksFromTitle(string title)
        {

            #region Query execution
            try
            {
                object result = _IBook.ShowFromBook_BooksFromTitle(title);
                // check if type of DataTable -> data to show
                if (result.GetType() != typeof(DataTable))
                {
                    _fileError.addError("SQL Exception", result + "");
                }
                return result;

            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }

            #endregion
        }
        public object updateSelectedBook(Book updateBook)
        {
            #region Checking the correct input
            string? checkBook = Validation_CheckBook.checkBook(updateBook);
            if (checkBook != null)
            {
                return checkBook;
            }
            #endregion

            #region Query execution
            try
            {
                object result = _IBook.updateSelectedBook(updateBook)!;
                // check if return true / message
                if (result.GetType() != typeof(Boolean))
                {
                    return Validation_General.insertErr("", result.ToString()!, "SQL Exception");
                }
                else
                    return "Book successfully updated !";
            }
            catch (Exception ex)
            {
                return Validation_General.insertErr("", ex.Message, "Server Exception");
            }
            #endregion
        }


    }
```


Validation
Validation_CheckBook.cs
Validation_CheckBorrow.cs
Validation_CheckExistingCategory.cs
Validation_CheckUser.cs
Validation_General.cs

Validation:

Validation_CheckBook.cs

```
using BusinessLogicLayer.actionFiles;
using DataAccessLayer.Entities;

public static class Validation_CheckBook
    {
        private static FileError _fileError = new FileError();

        public static string? checkDate(DateTime dateBook)
        {
            if (dateBook > DateTime.Now)
            {
                return "Date cannot be greater than current date !";
            }
            else
            {
                return null;
            }
        }

        public static string? checkCode(string codeBook)
        {
            if (codeBook.Length != 13)
            {
                return "Barcode must contain 13 digits !";
            }

            //Note that the IsDigit() method does not strictly check for a character in the range
0 through 9.
            //It allows a few characters such as Thai numerals ๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙.
            //We can use the following code to strictly check for ASCII digits:

            else if (!codeBook.All(c => (c >= 48 && c <= 57)))
            {
                return "Barcode must contain only digits ! -> " + codeBook;
            }
            else
            {
                return null;
            }
        }
        public static string? checkBook(Book checkBook)
        {
            #region Checking the correct input

            // Validation for null
            string errsValidBook = "";
            if (checkBook == null)
            {
                errsValidBook = Validation_General.insertErr("", "Values cannot be null", "Client
Exception");
                return errsValidBook;
            }

            // Validation for values
            else
            {
                string? checkValid;
                checkValid = checkDate(checkBook.PublicationDate);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr("", checkValid, "Client
Exception");
                }

                checkValid = checkCode(checkBook.Code);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr(errsValidBook, checkValid,
"Client Exception");
                }


                checkValid = Validation_General.checkOnlyLetter(checkBook.FirstName_Author);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr(errsValidBook, checkValid,
"Client Exception");
                }

                checkValid = Validation_General.checkOnlyLetter(checkBook.LastName_Author);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr(errsValidBook, checkValid,
"Client Exception");
                }

                checkValid = Validation_General.checkOnlyLetter(checkBook.Category);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr(errsValidBook, checkValid,
"Client Exception");
                }

                if(checkBook.SecondaryCategory != null)
                {
                    checkValid = Validation_General.checkOnlyLetter(checkBook.SecondaryCategory);
                    if (checkValid != null)
                    {
                        errsValidBook = Validation_General.insertErr(errsValidBook, checkValid,
"Client Exception");
                    }
                }

            }

            if (errsValidBook != "")
            { return errsValidBook; }

            else
            { return null; }
            #endregion}
```

Validation_CheckBorrow.cs

```
using BusinessLogicLayer.actionFiles;
using DataAccessLayer.Entities;

public static class Validation_CheckBorrow
    {
        private static FileError _fileError = new FileError();

        public static string? checkBorrow(Borrow checkBorrow)
        {
            #region Checking the correct input

            // Validation for null
            string errsValidBorrow = "";
            if (checkBorrow == null)
            {
                errsValidBorrow = Validation_General.insertErr("", "Values cannot be null",
"Client Exception");
                return errsValidBorrow;
            }

            // Validation for values
            else
            {
                string? checkValid;
                checkValid = Validation_CheckBook.checkCode(checkBorrow.Code);
                if (checkValid != null)
                {
                    errsValidBorrow = Validation_General.insertErr("", checkValid, "Client
Exception");
                }

                checkValid = Validation_CheckUser.checkId(checkBorrow.Id);
                if (checkValid != null)
                {
                    errsValidBorrow = Validation_General.insertErr(errsValidBorrow, checkValid,
"Client Exception");
                }

            }

            if (errsValidBorrow != null)
            { return errsValidBorrow; }

            else
            { return null; }
            #endregion

        }
    }
```

Validation_CheckExistingCategory.cs

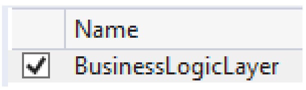
```
using BusinessLogicLayer.actionFiles;
using DataAccessLayer.Entities;

public static class Validation_CheckExistingCategory
    {
        private static FileError _fileError = new FileError();

        public static string? checkExistingCategory(ExistingCategory checkExistingCategory)
        {
            #region Checking the correct input

            // Validation for null
            string errsValidBook = "";
            if (checkExistingCategory == null)
            {
                errsValidBook = Validation_General.insertErr("", "Values cannot be null", "Client
Exception");
                return errsValidBook;
            }

            // Validation for values
            else
            {
                string? checkValid;
                checkValid = Validation_General.checkOnlyLetter(checkExistingCategory.Category);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr("", checkValid, "Client
Exception");
                }

                checkValid =
Validation_General.checkOnlyLetter(checkExistingCategory.SecondaryCategory);
                if (checkValid != null)
                {
                    errsValidBook = Validation_General.insertErr(errsValidBook, checkValid,
"Client Exception");
                }

            }

            if (errsValidBook != null)
            { return errsValidBook; }

            else
            { return null; }
            #endregion

        }
```

Validation_CheckExistingCategory.cs

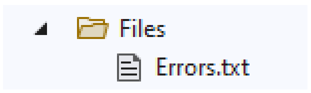
```
using System.Text.RegularExpressions;
using BusinessLogicLayer.actionFiles;
using DataAccessLayer.Entities;

public static class Validation_CheckUser
    {
        private static FileError _fileError = new FileError();

        public static string? checkEmail(string email)
        {
            var trimmedEmail = email.Trim();
            //            ^     Begin the match at the start of the string.
            //          [^@\s] +  Match one or more occurrences of any character other than the @
character or whitespace.
            //          @ Match the @ character.
            //          \.      Match a single period character.
            //          $ End the match at the end of the string.
            //          a@a.a
            string emailReg = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";
            if (!Regex.Match(trimmedEmail, emailReg).Success ||
               trimmedEmail.EndsWith(".") || trimmedEmail.StartsWith(".") ||
trimmedEmail.Contains("..") || trimmedEmail.Contains("..") ||
                // compare IndexOf to LastIndexOf to check
                // if there is more than one @
                trimmedEmail.IndexOf("@") != trimmedEmail.LastIndexOf("@"))
            {
                return "The email is not written correctly !";
            }
            try
            {
                var addr = new System.Net.Mail.MailAddress(email);
                if (addr.Address != trimmedEmail)
                {
                    return "The email is not written correctly !";
                }
            }
            catch
            {
                return "The email is not written correctly !";
            }

            return null;
        }
        public static string? checkPassword(string password)
        {
            if (password.Length != 10)
            {
                return "Password must be 10 characters in length !";
            }

            else
            {
                string resCheck = "";
                if (!password.Any(char.IsUpper))
                {
                    resCheck = "Password must contain  an uppercase letter !";
                }
                if (!password.Any(char.IsLower))
                {
                    if (resCheck != "")
                        resCheck += "\n" + "Password must contain  an uppercase letter !";
                }
                if (!password.Any(char.IsLower))
                {
                    if (resCheck != "")
                        resCheck += "\n" + "Password must contain  an uppercase letter !";
                }
                Regex rgx = new Regex("[^A-Za-z0-9]");
                if (!rgx.IsMatch(password))
                {
                    if (resCheck != "")
                        resCheck += "\n" + "Password must contain a special character !";
                }

                if (resCheck != "")
                { return resCheck; }

                else
                { return null; }
            }
        }
        public static string? checkId(string id)
        {
            if (id.Length != 9)
            {
                return "Id must contain 9 digits ! ";
            }

            else
            {

                  //    The test coefficient is in the form of
                 //     1 2 1 2 1 2 1 2 1

                int[] id_12_digits = { 1, 2, 1, 2, 1, 2, 1, 2, 1 };
                int count = 0;

                // The right digit is the check digit
                id = id.PadLeft(9, '0');

                for (int i = 0; i < 9; i++)
                {
                    //Multiply a digit by a check factor and add decimal digits
                    int num = Int32.Parse(id.Substring(i, 1)) * id_12_digits[i];

                    if (num > 9)
                        num = (num / 10) + (num % 10);

                    count += num;
                }

                //Checking if divisible by 10
                if (count % 10 != 0)
                {
                    return "The id format is incorrect ";
                }
                else
                {
                    return null;
                }
            }
        }
        public static string? checkUser(User checkUser)
        {
            #region Checking the correct input

            // Validation for null
            string errsValidUser = "";
            if (checkUser == null)
            {
                errsValidUser = Validation_General.insertErr("", "Values cannot be null", "Client 
Exception");
                return errsValidUser;
            }

            // Validation for values
            else
            {
                string? checkValid;
                checkValid = checkId(checkUser.Id);
                if (checkValid != null)
                {
                    errsValidUser = Validation_General.insertErr("", checkValid, "Client 
Exception");
                }

                checkValid = Validation_General.checkOnlyLetter(checkUser.FirstName);
                if (checkValid != null)
                {
                    errsValidUser = Validation_General.insertErr(errsValidUser, checkValid, 
"Client Exception");
                }

                checkValid = Validation_General.checkOnlyLetter(checkUser.LastName);
                if (checkValid != null)
                {
                    errsValidUser = Validation_General.insertErr(errsValidUser, checkValid, 
"Client Exception");
                }

                checkValid = checkEmail(checkUser.Email);
                if (checkValid != null)
                {
                    errsValidUser = Validation_General.insertErr(errsValidUser, checkValid, 
"Client Exception");
                }

                checkValid = checkPassword(checkUser.Password);
                if (checkValid != null)
                {
                    errsValidUser = Validation_General.insertErr(errsValidUser, checkValid, 
"Client Exception");
                }

            }

            if (errsValidUser != null)
            { return errsValidUser; }

            else
            { return null; }
            #endregion

        }
    }
```

Validation_General.cs

```
using System.Text.RegularExpressions;
using BusinessLogicLayer.actionFiles;


public  static class Validation_General
    {
        private static FileError _fileError = new FileError();

        public static string? checkOnlyLetter(string word)
        {
            if (!Regex.IsMatch(word, @"^[a-zA-Zא-ת]+$"))
            {
                return "Must write only letters ! -> " + word;
            }
            else
            {
                return null;
            }
        }

        public static string insertErr(string err, string newErr, string kindErr)
        {
            _fileError.addError(kindErr, newErr);
            if (err != "")
                err += "\n" + newErr;
            else
                err = newErr;

            return err;


        }
    }
```

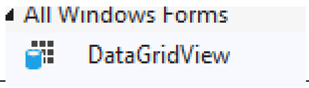


צד AppLayer - C# :



```
{
    "ConnectionStrings": {
        "DefaultConnestion": "Data Source=.;Initial
Catalog=Library;Integrated Security=True"
    }
```

Original get data:



ה

```
   public partial class getData : Form
    {
        string conStrin = @"Data Source=.;Initial Catalog=Library;Integrated Security=True";
        public getData()
        {
            InitializeComponent();
            SqlCommand cmd = new SqlCommand();
            using (SqlConnection sqlConnection = new SqlConnection(conStrin))
            {
                if (sqlConnection.State != ConnectionState.Open)
                    sqlConnection.Open();
                cmd.Connection = sqlConnection;
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.CommandText = "getBooks";
                cmd.Parameters.Add("@ERROR");
                cmd.Parameters[0].Direction = ParameterDirection.Output;
                SqlDataReader dr = cmd.ExecuteReader();

                if (cmd.Parameters["@ERROR"].Value != null &&
cmd.Parameters["@ERROR"].Value.ToString()!.Length > 0)
                {
                    string message = (string)cmd.Parameters["@ERROR"].Value;

                    // We'll close the connection path so you can read more procedures

                    sqlConnection.Close();
                }
                else if (dr.HasRows)
                {
                    SqlDataReader sqlDataReader = (SqlDataReader)dr;
                    DataTable dataTable = new DataTable();
                    dataTable.Load(sqlDataReader);
                    sqlConnection.Close();

                    DataAccessLayer.Entities.User user = new DataAccessLayer.Entities.User();
                    foreach (DataRow row in dataTable.Rows)
                    {
                        user = new DataAccessLayer.Entities.User()
                        {
                            Id = row["id"].ToString()!,
                            Email = row["email"].ToString()!,
                            Password = row["password"].ToString()!,
                            FirstName = row["FirstName"].ToString()!,
                            LastName = row["LastName"].ToString()!,
                            Type = (bool)row["type"]
                        };
                        MessageBox.Show(user.ToString());

                    }
                }

            }
        }
```

        }

Good get data:

good.cs

```
  private void specialButton1_Click(object sender, EventArgs e)
        {
            string? checkValues;
            checkValues = Validation_CheckUser.checkId(id.Text);
            checkAndSetError(id,checkValues);

            checkValues = Validation_CheckUser.checkEmail(email.Text);
            checkAndSetError(email, checkValues);

            checkValues = Validation_CheckUser.checkPassword(password.Text);
            checkAndSetError(password, checkValues);

            if (checkValues == null)
            {
                object resFun =
userLogic.ShowFromUser_UserFromSpecific_Id_Email_Password(id.Text, email.Text, password.Text);
                if (resFun.GetType() != typeof(DataTable))
                {
                    MessageBox.Show(resFun.ToString());
                }
                else
                {
                    DataTable dt = (DataTable)resFun;
                    User user = new User();
                    foreach (DataRow row in dt.Rows)
                    {
                        user = new User()
                        {
                            Id = row["id"].ToString()!,
                            Email = row["email"].ToString()!,
                            Password = row["password"].ToString()!,
                            FirstName = row["FirstName"].ToString()!,
                            LastName = row["LastName"].ToString()!,
                            Type = (bool)row["type"]
                        };

                    }
                    MainApp mainApp = new MainApp();
                    mainApp.TopLevel = false;
                    mainApp.Parent = this.MdiParent;
                    mainApp.Activate();
                    mainApp.Location = new Point((this.MdiParent.Width - mainApp.Width) / 2,
(this.MdiParent.Height - mainApp.Height) / 2);
                    mainApp.Show();
                    this.Close();

                    //MainApp mainApp = new MainApp();
                    //mainApp.Show();
                    //this.Hide();
                }

            }

        }
```

Help func:

help.cs

```
public static void createCategories(List<string> categories, ComboBox category)
        {
            ExistingCategorylogic existingCategorylogic = new ExistingCategorylogic();
            object resFun = existingCategorylogic.getExistingCategories();
            if (resFun.GetType() != typeof(DataTable))
            {
                MessageBox.Show(resFun.ToString());
            }
            else
            {
                DataTable dt = (DataTable)resFun;
                foreach (DataRow row in dt.Rows)
                {
                    categories.Add((string)row["Category"]);

                }
            }
            category.DataSource = categories;

        }

        public static void category_SelectedIndexChanged(List<string> secondaryCategorySelect,
ComboBox secondaryCategory, string choose)
        {
            ExistingCategorylogic existingCategorylogic = new ExistingCategorylogic();
            secondaryCategory.DataSource = null;
            secondaryCategorySelect.Clear();
            object resFun =
existingCategorylogic.ShowFromExistingCategories_SubcategoryFromCategory(choose);
            if (resFun.GetType() != typeof(DataTable))
            {
                MessageBox.Show(resFun.ToString());
            }
            else
            {
                secondaryCategorySelect.Add("No secondary category");
                DataTable dt = (DataTable)resFun;
                foreach (DataRow row in dt.Rows)
                {
                    secondaryCategorySelect.Add((string)row["secondaryCategory"]);
                }

            }

            secondaryCategory.DataSource = secondaryCategorySelect;
        }
    }
```

```
private static void resizeControl(Rectangle r, Control c, Rectangle originalFormSize, object
thisObj)
        {
            float xRatio;
            float yRatio;
            if(thisObj == null)
            {
                return;
            }
            else if (thisObj.GetType().BaseType.Name == "Form")
            {
                Form thisForm = (Form)thisObj;
                xRatio  = (float)(thisForm.Width) / (float)(originalFormSize.Width);
                yRatio = (float)(thisForm.Height) / (float)(originalFormSize.Height);
            }
            else if (thisObj.GetType().BaseType.Name == "UserControl")
            {
                UserControl thisUC = (UserControl)thisObj;
                xRatio = (float)(thisUC.Width) / (float)(originalFormSize.Width);
                yRatio = (float)(thisUC.Height) / (float)(originalFormSize.Height);
            }
            else
            {
                return;
            }


            int newX = (int)(r.Location.X * xRatio);
            int newY = (int)(r.Location.Y * yRatio);

            int newWidth = (int)(r.Width * xRatio);
            int newHeight = (int)(r.Height * yRatio);

            c.Location = new Point(newX, newY);
            c.Size = new Size(newWidth, newHeight);
        }

        public  static void Form_Resize(Control[] controls , Rectangle []
controlerOriginalRectangle,Rectangle originalFormSize,object thisObj)
        {
            // loop over controls and updates values
            foreach (var (control, index) in controls.Select((value, i) => (value, i)))
            {
                resizeControl(controlerOriginalRectangle[index], control, originalFormSize,
thisObj);
            }

        }
        public static void addImgCursor(string url, Size size, Control control)
        {
            Bitmap bitmap = new Bitmap(new Bitmap(url), size);
            control.Cursor = new Cursor(bitmap.GetHicon());
        }

        public static void Form_LoadCreateRectangles(ref Rectangle originalFormSize, ref
Control[] controls, ref Rectangle[] controlerOriginalRectangle, object thisObj)
        {


            if (thisObj.GetType().BaseType.Name == "Form")
            {
                Form  thisForm = (Form)thisObj;
                originalFormSize = new Rectangle(thisForm.Location.X, thisForm.Location.Y,
thisForm.Size.Width, thisForm.Size.Height);

                controlerOriginalRectangle = new Rectangle[thisForm.Controls.Count];
                controls = new Control[thisForm.Controls.Count];
                // copy all collection to array from 0
                thisForm.Controls.CopyTo(controls, 0);
            }

            else if(thisObj.GetType().BaseType.Name == "UserControl")
            {
                UserControl thisForm = (UserControl)thisObj;
                originalFormSize = new Rectangle(thisForm.Location.X, thisForm.Location.Y,
thisForm.Size.Width, thisForm.Size.Height);

                controlerOriginalRectangle = new Rectangle[thisForm.Controls.Count];
                controls = new Control[thisForm.Controls.Count];
                // copy all collection to array from 0
                thisForm.Controls.CopyTo(controls, 0);

            }

            else
            {
                return ;
            }


            //// Loop over tuples with the item and its index
            foreach (var (control, index) in controls.Select((value, i) => (value, i)))
            {
                controlerOriginalRectangle[index] = new Rectangle(control.Location.X,
control.Location.Y, control.Width, control.Height);
            }

        }
        // Allow Combo Box to center aligned
        public static void cbxDesign_DrawItem(ref object sender, ref DrawItemEventArgs e)
        {
            // By using Sender, one method could handle multiple ComboBoxes
            ComboBox cbx = sender as ComboBox;
            if (cbx != null)
            {
                // Always draw the background
                e.DrawBackground();

                // Drawing one of the items?
                if (e.Index >= 0)
                {
                    // Set the string alignment.  Choices are Center, Near and Far
                    StringFormat sf = new StringFormat();
                    sf.LineAlignment = StringAlignment.Center;
                    sf.Alignment = StringAlignment.Center;

                    // Set the Brush to ComboBox ForeColor to maintain any ComboBox color 
settings
                    // Assumes Brush is solid
                    Brush brush = new SolidBrush(cbx.ForeColor);

                    // If drawing highlighted selection, change brush
                    if ((e.State & DrawItemState.Selected) == DrawItemState.Selected)
                        brush = SystemBrushes.HighlightText;

                    // Draw the string
                    e.Graphics.DrawString(cbx.Items[e.Index].ToString(), cbx.Font, brush, 
e.Bounds, sf);
                }
            }
        }

        public static void hideAndShowUC(UserControl[] ucs, string kindAction,Form form)
        {
            if(ucs.Length != 4)
            {
                MessageBox.Show("The array must contain 4 UC (add, delete, show, update)");
                return;
            }
            foreach (UserControl uc in ucs)
            {
                uc.Size = new Size(uc.Parent.Width - 50, uc.Height);
                uc.Location = new Point((form.Width - uc.Width) / 2 - 10, (form.Height - 
uc.Height) / 2 - 30);
                uc.Hide();
            }

            switch (kindAction)
            {
                case "Add":
                    ucs[0].Show();
                    break;

                case "Delete":
                    ucs[1].Show();
                    break;

                case "Show":
                    ucs[2].Show();
                    break;

                case "Update":
                    ucs[3].Show();
                    break;
            }
        }
```

Tool Strip

help.cs

```
private void MenuItem_Click(object sender, EventArgs e)
        {

            FormCollection FormsOpen = Application.OpenForms;
            for (int i = 0; i < FormsOpen.Count; i++)
            {
                if (FormsOpen[i].Name != "Main")
                    FormsOpen[i].Close();
            }

            ToolStripMenuItem menuStrip = (ToolStripMenuItem)sender;
            ToolStripItem parent = menuStrip.OwnerItem;

            // We will check what type of form we would like to show / add
            // And then what kind of add / show -> city / street
            switch (parent.Text)
            {
                case "Books":
                    AreaBook book = new AreaBook(menuStrip.Text);
                    book.MdiParent = this;
                    book.Activate();
                    book.Show();
                    book.Size = new Size(this.Width - 100, this.Height - 150);
                    //book.Location = new Point((this.Width - book.Width) / 2, (this.Height -
book.Height) / 2);
                    book.Location = new Point((this.Width - book.Width) / 2 - 10, (this.Height -
book.Height) / 2 - 30);
                    break;

                case "Borrow":
                    AreaBorrow borrow = new AreaBorrow(menuStrip.Text);
                    borrow.MdiParent = this;
                    borrow.Activate();
                    borrow.Show();
                    borrow.Size = new Size(this.Width - 100, this.Height - 150);
                    borrow.Location = new Point((this.Width - borrow.Width) / 2 - 10,
(this.Height - borrow.Height) / 2 - 30);
                    break;

                case "Categories":
                    AreaExistingCategories existingCategories = new
AreaExistingCategories(menuStrip.Text);
                    existingCategories.MdiParent = this;
                    existingCategories.Activate();
                    existingCategories.Show();
                    existingCategories.Size = new Size(this.Width - 100, this.Height - 150);
                    existingCategories.Location = new Point((this.Width -
existingCategories.Width) / 2 - 10, (this.Height - existingCategories.Height) / 2 - 30);
                    break;

                case "Users":
                    AreaUser user = new AreaUser(menuStrip.Text);
                    user.MdiParent = this;
                    user.Activate();
                    user.Show();
                    user.Size = new Size(this.Width - 100, this.Height - 150);
                    user.Location = new Point((this.Width - user.Width) / 2 - 10, (this.Height -
user.Height) / 2 - 30);
                    break;
            }
```

Tool Strip

help.cs

```
private void MenuItem_Click(object sender, EventArgs e)
        {

            FormCollection FormsOpen = Application.OpenForms;
            for (int i = 0; i < FormsOpen.Count; i++)
            {
                if (FormsOpen[i].Name != "Main")
                    FormsOpen[i].Close();
            }

            ToolStripMenuItem menuStrip = (ToolStripMenuItem)sender;
            ToolStripItem parent = menuStrip.OwnerItem;

            // We will check what type of form we would like to show / add
            // And then what kind of add / show -> city / street
            switch (parent.Text)
            {
                case "Books":
                    AreaBook book = new AreaBook(menuStrip.Text);
                    book.MdiParent = this;
                    book.Activate();
                    book.Show();
                    book.Size = new Size(this.Width - 100, this.Height - 150);
                    //book.Location = new Point((this.Width - book.Width) / 2, (this.Height - 
book.Height) / 2);
                    book.Location = new Point((this.Width - book.Width) / 2 - 10, (this.Height - 
book.Height) / 2 - 30);
                    break;

                case "Borrow":
                    AreaBorrow borrow = new AreaBorrow(menuStrip.Text);
                    borrow.MdiParent = this;
                    borrow.Activate();
                    borrow.Show();
                    borrow.Size = new Size(this.Width - 100, this.Height - 150);
                    borrow.Location = new Point((this.Width - borrow.Width) / 2 - 10, 
(this.Height - borrow.Height) / 2 - 30);
                    break;

                case "Categories":
                    AreaExistingCategories existingCategories = new 
AreaExistingCategories(menuStrip.Text);
                    existingCategories.MdiParent = this;
                    existingCategories.Activate();
                    existingCategories.Show();
                    existingCategories.Size = new Size(this.Width - 100, this.Height - 150);
                    existingCategories.Location = new Point((this.Width - 
existingCategories.Width) / 2 - 10, (this.Height - existingCategories.Height) / 2 - 30);
                    break;

                case "Users":
                    AreaUser user = new AreaUser(menuStrip.Text);
                    user.MdiParent = this;
                    user.Activate();
                    user.Show();
                    user.Size = new Size(this.Width - 100, this.Height - 150);
                    user.Location = new Point((this.Width - user.Width) / 2 - 10, (this.Height - 
user.Height) / 2 - 30);
                    break;
            }
```