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);

}

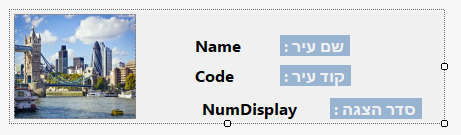
}

}

}

}

## **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

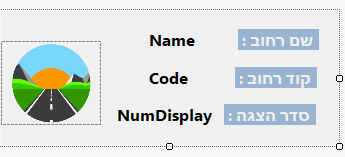
{

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

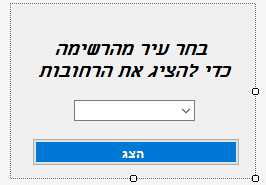
{

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)

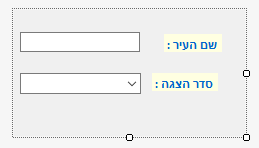
{

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;

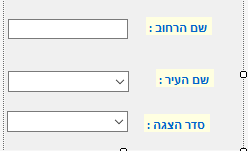
City city = new City(cityName, cityOrder);

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();

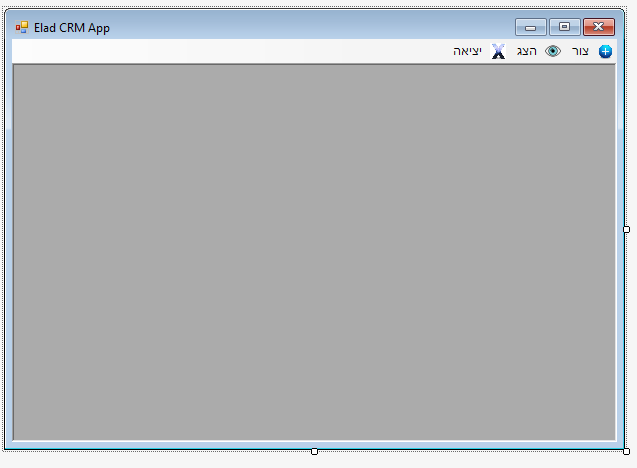
Street street = new Street(streetName, streetOrder, cityCode);

parent.addToList("street", street);

HelpFuncs.createOrderList(this.Parent as Add, NumDisplay, "street");

}}}

## **Form:**





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)

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Program());

}

private void InitializeComponent()

{

this.menuStrip2 = new System.Windows.Forms.MenuStrip();

this.צורToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.עירToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.רחובToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.הצגToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.עריםToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.רחובותלפיעירToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.יציאהToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.menuStrip2.SuspendLayout();

this.SuspendLayout();

//

// menuStrip2

//

this.menuStrip2.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.צורToolStripMenuItem,

this.הצגToolStripMenuItem,

this.יציאהToolStripMenuItem});

this.menuStrip2.Location = new System.Drawing.Point(0, 0);

this.menuStrip2.Name = "menuStrip2";

this.menuStrip2.RightToLeft = System.Windows.Forms.RightToLeft.Yes;

this.menuStrip2.Size = new System.Drawing.Size(605, 24);

this.menuStrip2.TabIndex = 2;

this.menuStrip2.Text = "menuStrip2";

//

// צורToolStripMenuItem

//

this.צורToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.עירToolStripMenuItem,

this.רחובToolStripMenuItem});

this.צורToolStripMenuItem.Image = global::CV\_Daniel\_Artzi.Properties.Resources.add;

this.צורToolStripMenuItem.Name = "צורToolStripMenuItem";

this.צורToolStripMenuItem.Size = new System.Drawing.Size(52, 20);

this.צורToolStripMenuItem.Text = "צור";

//

// עירToolStripMenuItem

//

this.עירToolStripMenuItem.Name = "עירToolStripMenuItem";

this.עירToolStripMenuItem.Size = new System.Drawing.Size(180, 22);

this.עירToolStripMenuItem.Text = "עיר";

this.עירToolStripMenuItem.Click += new System.EventHandler(this.עירToolStripMenuItem\_Click);

//

// רחובToolStripMenuItem

//

this.רחובToolStripMenuItem.Name = "רחובToolStripMenuItem";

this.רחובToolStripMenuItem.Size = new System.Drawing.Size(180, 22);

this.רחובToolStripMenuItem.Text = "רחוב";

this.רחובToolStripMenuItem.Click += new System.EventHandler(this.רחובToolStripMenuItem\_Click);

//

// הצגToolStripMenuItem

//

this.הצגToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.עריםToolStripMenuItem,

this.רחובותלפיעירToolStripMenuItem});

this.הצגToolStripMenuItem.Image = global::CV\_Daniel\_Artzi.Properties.Resources.show;

this.הצגToolStripMenuItem.Name = "הצגToolStripMenuItem";

this.הצגToolStripMenuItem.Size = new System.Drawing.Size(55, 20);

this.הצגToolStripMenuItem.Text = "הצג";

//

// עריםToolStripMenuItem

//

this.עריםToolStripMenuItem.Name = "עריםToolStripMenuItem";

this.עריםToolStripMenuItem.Size = new System.Drawing.Size(152, 22);

this.עריםToolStripMenuItem.Text = "ערים";

this.עריםToolStripMenuItem.Click += new System.EventHandler(this.עריםToolStripMenuItem\_Click);

//

// רחובותלפיעירToolStripMenuItem

//

this.רחובותלפיעירToolStripMenuItem.Name = "רחובותלפיעירToolStripMenuItem";

this.רחובותלפיעירToolStripMenuItem.Size = new System.Drawing.Size(152, 22);

this.רחובותלפיעירToolStripMenuItem.Text = "רחובות לפי עיר";

this.רחובותלפיעירToolStripMenuItem.Click += new System.EventHandler(this.רחובותלפיעירToolStripMenuItem\_Click);

//

// יציאהToolStripMenuItem

//

this.יציאהToolStripMenuItem.Image = global::CV\_Daniel\_Artzi.Properties.Resources.close;

this.יציאהToolStripMenuItem.Name = "יציאהToolStripMenuItem";

this.יציאהToolStripMenuItem.Size = new System.Drawing.Size(64, 20);

this.יציאהToolStripMenuItem.Text = "יציאה";

this.יציאהToolStripMenuItem.Click += new System.EventHandler(this.יציאהToolStripMenuItem\_Click);

//

// Program

//

this.ClientSize = new System.Drawing.Size(605, 403);

this.Controls.Add(this.menuStrip2);

this.IsMdiContainer = true;

this.Name = "Program";

this.RightToLeft = System.Windows.Forms.RightToLeft.No;

this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;

this.Text = "Elad CRM App";

this.Load += new System.EventHandler(this.Program\_Load);

this.menuStrip2.ResumeLayout(false);

this.menuStrip2.PerformLayout();

this.ResumeLayout(false);

this.PerformLayout();

}

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)

{

RemoveFormsAndShow("show", "showCities");

}

private void רחובותלפיעירToolStripMenuItem\_Click(object sender, EventArgs e)

{

RemoveFormsAndShow("show", "showStreets");

}

private void RemoveFormsAndShow(string AddOrShowForm, string kindAddOrShow)

{

// We will go through the open forms and close any form that is not the main form

// We can't do this in a loop for each

//Since the transition will be destroyed at the first closing of Open Form

//Go through the actual amount of Open Forms

FormCollection FormsOpen = Application.OpenForms;

for (int i = 0; i < FormsOpen.Count; i++)

{

if (FormsOpen[i].Name != "Program")

FormsOpen[i].Close();

}

// We will check what type of form we would like to show / add

// And then what kind of add / show -> city / street

switch (AddOrShowForm)

{

case "add":

Add add = new Add(kindAddOrShow, this.cityList, this.streetList);

add.MdiParent = this;

add.Show();

add.Activate();

add.Location = new Point((this.Width - add.Width) / 2, 0);

break;

case "show":

Show show = new Show(kindAddOrShow, this.cityList, this.streetList);

show.MdiParent = this;

show.Show();

show.Activate();

show.Location = new Point((this.Width - show.Width) / 2, 0);

break;

}

}

private void יציאהToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

public List<City> GetCityList()

{

return cityList;

}

public void addCityToList(City city)

{

cityList.Add(city);

}

public List<Street> GetStreetList()

{

return streetList;

}

public void addStreeToList(Street street)

{

streetList.Add(street);

}

private void Program\_Load(object sender, EventArgs e)

{

MdiClient mdiClient;

foreach (Control control in this.Controls)

{

if (control is MdiClient)

{

mdiClient = (MdiClient)control;

mdiClient.BackColor = Color.LightSeaGreen;

mdiClient.BackgroundImage = Properties.Resources.back;

mdiClient.BackgroundImageLayout = ImageLayout.Center;

}

}

}

public void addCityToList(City city)

{

cityList.Add(city);

}

public List<Street> GetStreetList()

{

return streetList;

}

public void addStreeToList(Street street)

{

streetList.Add(street);

}

private void Program\_Load(object sender, EventArgs e)

{

MdiClient mdiClient;

foreach (Control control in this.Controls)

{

if (control is MdiClient)

{

mdiClient = (MdiClient)control;

mdiClient.BackColor = Color.LightSeaGreen;

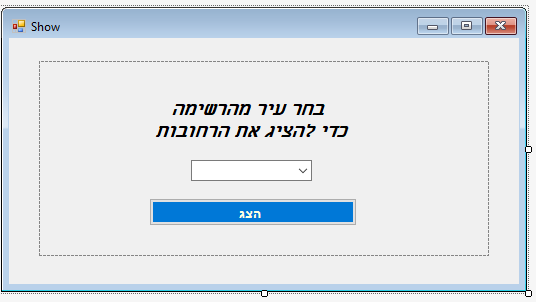
mdiClient.BackgroundImage = Properties.Resources.back;

mdiClient.BackgroundImageLayout = ImageLayout.Center;

}

}

}



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; }

}

}

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; }

}

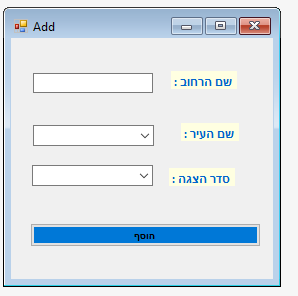
public List<Street> CityStreet

{

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!");

}

}

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!");

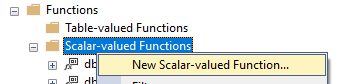
}

}

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 >

-- =============================================

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-zא-ת]%' or @Book\_FirstName\_Author Is Null or @Book\_FirstName\_Author = '' )

or ( @Book\_LastName\_Author LIKE '%[^A-Za-zא-ת]%' 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-zא-ת]%' or @Book\_Category IS NULl or @Book\_Category = '' )

or (@Book\_SecondaryCategory LIKE '%[^A-Za-zא-ת]%' 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

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

-- -- \*\*\* 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

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 >

-- =============================================

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\_Author,@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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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\_LastName\_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

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

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

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,@updateUser\_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

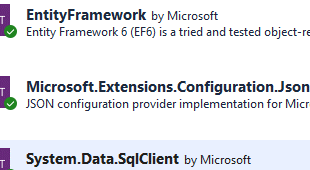
set rowcount 0

end

end

END

# צד C# - data access layer:

**EntityFramework Microsoft.Extensions.Configuration.Json Syste.Data.SqlClient**

## **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:**

**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:**

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; }

}

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; }

}

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; }

}

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; }

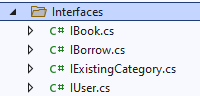
[Required]

[StringLength(10)]

public string Password { get; set; }

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

}



## **Interfaces:**

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);

}

}

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);

}

}

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);

}

}

C

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



****

## **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)

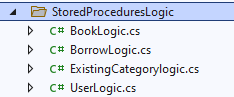
{

errors.Add(JsonConvert.DeserializeObject<Server\_Error>(line)!);

}

}

}



## **StoredProceduresLogic:**

**BookLogic.cs**

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 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\_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

}

}

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

}

}

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

}

}

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)

{

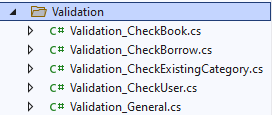
return Validation\_General.insertErr("", ex.Message, "Server Exception");

}

#endregion

}

}



## **Validation:**

**Validation\_CheckBook.cs**

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

}alid != 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 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}

}alid != 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");

}

}

**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

}

}

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

}alid != 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");

}

}

**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

}

}

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

}

}

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

}

}

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 C# - AppLayer:

{

"ConnectionStrings": {

"DefaultConnestion": "Data Source=.;Initial Catalog=Library;Integrated Security=True"

}

}

****

## **Original get data:**

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());

}

}

}

}

ה

## **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;

}

}

}

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;

}

}

}

// 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;

}