**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** | RU.17701729.04.03-01 12 |

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Руководитель проекта,  Профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_ /В.В. Шилов/  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  проф, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /В.В. Шилов/  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. |

**НАСТОЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ СОЗДАНИЯ ДОКУМЕНТАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.03-01 12 01-1-ЛУ**

Исполнитель:

студент группы БПИ181(1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Е.В. Степанов /

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019г.

**Москва 2019**

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** | RU.17701729.04.03-01 12 |

УТВЕРЖДЕН

RU.17701729.04.03-01 34 01-1-ЛУ

**НАСТОЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ СОЗДАНИЯ ДОКУМЕНТАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**Текст программы**

**RU.17701729.04.03-01 12 01-1**

**Листов 539**

**Москва 2019**

**СОДЕРЖАНИЕ**

1. Текст программы4

1.1. SDWP.exe4

1.2. ApplicationLib.dll315

1.3. SDWPApi.dll458

1.4. AeroORMFramework.dll515

1. **ТЕКСТ ПРОГРАММЫ**
   1. SDWP.exe

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using SDWP.Interfaces;

namespace SDWP

{

public class DocumentationController : IDocController

{

#region Propeties

public Documentation Documentation { get; private set; }

public List<Document> Documents { get; private set; }

public Document CurrentDocument { get; private set; }

public Item CurrentItem { get; private set; }

public Item CurrentContentItem { get; private set; }

public List<Item> CurrentItemsList { get; private set; }

public List<Paragraph> CurrentParagraphsList { get; private set; }

#endregion

public DocumentationController() { }

public void UploadLocalDocumentation(LocalDocumentation localDocumentation)

{

Documentation = localDocumentation.Documentation;

Documents = localDocumentation.Documents;

}

public void UploadCloudDocumentation(Documentation documentation, List<Document> documents)

{

Documentation = documentation;

Documents = documents;

}

public void UploadItem(Item selectedItem)

{

if (selectedItem.Items == null)

{

CurrentParagraphsList = selectedItem.Paragraphs;

CurrentContentItem = selectedItem;

}

else

{

CurrentItemsList = selectedItem.Items;

CurrentItem = selectedItem;

}

}

public void UploadParagraphs(Item selectedItem)

{

CurrentContentItem = selectedItem;

CurrentParagraphsList = selectedItem.Paragraphs;

}

public void UploadDocument(Document document)

{

CurrentItemsList = document.Items;

CurrentDocument = document;

}

public void GoToPreviousItem(Item currentItem)

{

CurrentItemsList = currentItem.ParentList;

CurrentItem = currentItem.ParentItem;

}

public bool CanGoToPrevItem()

{

return CurrentItem != null && CurrentItem.ParentList != null;

}

/// <summary>

/// Sets all references of properties ti null

/// </summary>

public void Clear()

{

Documentation = null;

Documents = null;

CurrentContentItem = null;

CurrentDocument = null;

CurrentItem = null;

CurrentItemsList = null;

CurrentParagraphsList = null;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Threading;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Navigation;

using System.Windows.Shapes;

using SDWP.Interfaces;

namespace SDWP.Exceptions

{

public class ExceptionHandler : IExceptionHandler

{

#region Propperties/Variables

public Dispatcher Dispatcher { get; set; }

private string MessageCaption { get; } = "SDWP Soft's message";

#endregion

public ExceptionHandler(Dispatcher dispatcher)

{

Dispatcher = dispatcher;

}

#region Methods

public void HandleWithMessageBox(Exception ex)

{

Dispatcher.Invoke(() => SDWPMessageBox.ShowSDWPMessageBox(

"SDWP's message", ex.Message, MessageBoxButton.OK));

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Threading;

using ApplicationLib.Models;

using FileLib.Interfaces;

using FileLib.FileParsers;

using SDWP.Exceptions;

using SDWP.Interfaces;

namespace SDWP.Factories

{

class SdwpAbstractFactory : ISdwpAbstractFactory

{

public IDocController GetDocController()

{

return new DocumentationController();

}

public IExceptionHandler GetExceptionHandler(Dispatcher dispatcher)

{

return new ExceptionHandler(dispatcher);

}

public IFileParser GetFileParser()

{

return new CSFileParser();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SDWP.Interfaces

{

interface IAccountPage

{

Action CloseAccGrid { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

namespace SDWP.Interfaces

{

public interface IDocController

{

/// <summary>

/// Current documentation

/// </summary>

Documentation Documentation { get; }

/// <summary>

/// List of documetns of this documentation

/// </summary>

List<Document> Documents { get; }

Document CurrentDocument { get; }

/// <summary>

/// The item, which items are now uploaded

/// </summary>

Item CurrentItem { get; }

/// <summary>

/// An item, which content os now uploaded

/// </summary>

Item CurrentContentItem { get; }

/// <summary>

/// The list where the CurrentItem is

/// </summary>

List<Item> CurrentItemsList { get; }

/// <summary>

/// The list of paragraphs which is now uploaded to UI

/// </summary>

List<Paragraph> CurrentParagraphsList { get; }

void UploadLocalDocumentation(LocalDocumentation documentation);

void UploadCloudDocumentation(Documentation documentation, List<Document> documents);

/// <summary>

/// Changes the state of this controller by uploading the selected doc

/// </summary>

void UploadDocument(Document document);

/// <summary>

/// Changes the state of this controller by uploading the content or list of items of

/// selected item

/// </summary>

void UploadItem(Item selectedItem);

void UploadParagraphs(Item selectedItem);

/// <summary>

/// Decides whether it is possible to go to the previous item

/// </summary>

bool CanGoToPrevItem();

/// <summary>

/// Changes the state of this controller when the user wants to get to previous item

/// </summary>

/// <param name="currentItem">CURRENT (the item with which user is working now) item</param>

void GoToPreviousItem(Item currentItem);

void Clear();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Threading;

namespace SDWP.Interfaces

{

public interface IExceptionHandler

{

Dispatcher Dispatcher { get; set; }

void HandleWithMessageBox(Exception ex);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Threading;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using FileLib.Interfaces;

using FileLib.FileParsers;

using SDWP.Exceptions;

using SDWP.Interfaces;

namespace SDWP.Interfaces

{

interface ISdwpAbstractFactory

{

IDocController GetDocController();

IExceptionHandler GetExceptionHandler(Dispatcher dispatcher);

IFileParser GetFileParser();

}

}

using System;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media.Animation;

namespace SDWP

{

public class MainPageAnimations

{

public static void HideLeftDocumentationGrid(Grid leftDocGrid)

{

leftDocGrid.Children.OfType<Grid>().ToList()[2].Visibility = Visibility.Collapsed;

DoubleAnimation hideAnimation = new DoubleAnimation()

{

To = 60,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

hideAnimation.Completed += (sender, e) =>

{

leftDocGrid.Width = 60;

};

leftDocGrid.BeginAnimation(FrameworkElement.WidthProperty, hideAnimation);

}

public static void ShowLeftDocumentationGrid(Grid leftDocGrid)

{

DoubleAnimation hideAnimation = new DoubleAnimation()

{

To = 250,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

hideAnimation.Completed += (sender, e) =>

{

leftDocGrid.Width = 250;

leftDocGrid.Children.OfType<Grid>().ToList()[2].Visibility = Visibility.Visible;

};

leftDocGrid.BeginAnimation(FrameworkElement.WidthProperty, hideAnimation);

}

public static void AnimateMargin(Grid grid, Thickness newMargin)

{

ThicknessAnimation marginAnimation = new ThicknessAnimation()

{

To = newMargin,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

marginAnimation.Completed += (sender, e) =>

{

grid.Margin = newMargin;

};

grid.BeginAnimation(FrameworkElement.MarginProperty, marginAnimation);

}

public static void AnimateWidth(FrameworkElement frameworkElement, double newWidth) =>

AnimateWidth(frameworkElement, newWidth, new Action(() => { }));

public static void AnimateWidth(FrameworkElement frameworkElement, double newWidth,

Action afterAnimationAction)

{

DoubleAnimation widthAnimation = new DoubleAnimation()

{

To = newWidth,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

widthAnimation.Completed += (sender, e) =>

{

frameworkElement.Width = newWidth;

afterAnimationAction();

};

frameworkElement.BeginAnimation(FrameworkElement.WidthProperty, widthAnimation);

}

public static void AnimateHeight(FrameworkElement frameworkElement, double newHeight) =>

AnimateHeight(frameworkElement, newHeight, new Action(() => { }));

public static void AnimateHeight(FrameworkElement frameworkElement, double newHeight,

Action afterAnimationAction)

{

DoubleAnimation heightAnimation = new DoubleAnimation()

{

To = newHeight,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

heightAnimation.Completed += (sender, e) =>

{

frameworkElement.Height = newHeight;

afterAnimationAction();

};

frameworkElement.BeginAnimation(FrameworkElement.HeightProperty, heightAnimation);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SDWP.Models

{

public class DocumentationListBoxItem

{

public string DocumentationTitle { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Text;

using System.Data.SqlClient;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

using System.Data.SqlTypes;

using System.Net.Mail;

namespace SDWP.Models

{

public class UserAccLeftMenuGrids

{

public Rectangle HeaderBottomLineRect { get; set; }

public Grid OptionsGrid { get; set; }

}

}

using System.Windows.Controls;

namespace SDWP.Models

{

public class UserAccountGrids

{

public Grid UserAccGrid { get; set; }

public Grid LeftMenuGrid { get; set; }

public Frame ContentFrame { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using ApplicationLib.Models;

namespace SDWP.Models

{

public class DocTreeViewItem : TreeViewItem, INotifyPropertyChanged

{

public Item Item { get; set; }

private string headerText;

public string HeaderText

{

get

{

return headerText;

}

set

{

headerText = value;

OnPropertyChanged("HeaderText");

}

}

private bool isEnabledForEdditing;

public bool IsEnabledForEdditing

{

get

{

return isEnabledForEdditing;

}

set

{

isEnabledForEdditing = value;

OnPropertyChanged("IsEnabledForEdditing");

}

}

public DocTreeViewItem(Item item) : base()

{

Item = item;

HeaderText = Item.Name;

this.DataContext = this;

}

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName]string prop = "")

{

if (prop == "HeaderText")

{

Item.Name = HeaderText;

}

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

namespace SDWP.Models

{

public class TemplateTreeViewItem : TreeViewItem, INotifyPropertyChanged

{

private string headerText;

public string HeaderText

{

get

{

return headerText;

}

set

{

headerText = value;

OnPropertyChanged("HeaderText");

}

}

private bool isEnabledForEdditing;

public bool IsEnabledForEdditing

{

get

{

return isEnabledForEdditing;

}

set

{

isEnabledForEdditing = value;

OnPropertyChanged("IsEnabledForEdditing");

}

}

public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged([CallerMemberName]string prop = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Media;

using ApplicationLib.Models;

namespace SDWP.Models

{

public class TemplateTreeViewItemItem : TemplateTreeViewItem

{

public Item Item { get; set; }

public TemplateTreeViewItemItem(Item item) : base()

{

Item = item;

HeaderText = Item.Name;

DataContext = this;

}

public override void OnPropertyChanged([CallerMemberName] string prop = "")

{

if (prop == "HeaderText")

Item.Name = HeaderText;

base.OnPropertyChanged(prop);

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using ApplicationLib.Models;

namespace SDWP.Models

{

public class TemplateTreeViewParagraphItem : TemplateTreeViewItem

{

private ImageSource imageSource;

public Paragraph Paragraph { get; set; }

public ImageSource ImageSource

{

get

{

return imageSource;

}

set

{

imageSource = value;

OnPropertyChanged("ImageSource");

}

}

public TemplateTreeViewParagraphItem(Paragraph paragraph) : base()

{

Paragraph = paragraph;

HeaderText = Paragraph.ParagraphElement.Title;

switch (paragraph.Type)

{

case "Subparagraph":

ImageSource = new BitmapImage(new Uri("pack://application:,,,/Resources/templateTreeViewSubparagraphIcon.png"));

break;

case "Table":

ImageSource = new BitmapImage(new Uri("pack://application:,,,/Resources/templateTreeViewTableIcon.png"));

break;

case "NumberedList":

ImageSource = new BitmapImage(new Uri("pack://application:,,,/Resources/templateTreeViewNumberedListIcon.png"));

break;

case "ParagraphImage":

ImageSource = new BitmapImage(new Uri("pack://application:,,,/Resources/templateTreeViewImageIcon.png"));

break;

}

DataContext = this;

}

public override void OnPropertyChanged([CallerMemberName] string prop = "")

{

if (prop == "HeaderText")

Paragraph.ParagraphElement.Title = HeaderText;

base.OnPropertyChanged(prop);

}

}

}

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media.Animation;

namespace SDWP

{

public static class WelcomePageBottomGridAnimation

{

public static void ShowTheHintGrid(Grid grid)

{

SetIntialParameters(grid);

grid.BeginAnimation(FrameworkElement.MarginProperty, CreateShowAnimation(grid));

}

public static void CloseTheHintGrid(Grid grid)

{

grid.BeginAnimation(FrameworkElement.MarginProperty, CreateHideAnimation(grid));

}

private static void SetIntialParameters(Grid grid)

{

grid.Margin = new Thickness(0, 0, 0, -100);

grid.Visibility = Visibility.Visible;

}

private static ThicknessAnimation CreateHideAnimation(Grid grid)

{

ThicknessAnimation hideAnimation = new ThicknessAnimation

{

From = grid.Margin,

To = new Thickness(0, 0, 0, -125),

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop,

};

hideAnimation.Completed += (sender, e) =>

{

grid.Visibility = Visibility.Collapsed;

};

return hideAnimation;

}

private static ThicknessAnimation CreateShowAnimation(Grid grid)

{

ThicknessAnimation showAnimation = new ThicknessAnimation

{

From = new Thickness(0, 0, 0, -125),

To = new Thickness(0, 0, 0, 0),

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop,

};

showAnimation.Completed += (sender, e) => SetFinalParameters(grid);

return showAnimation;

}

private static void SetFinalParameters(Grid grid)

{

grid.Margin = new Thickness(0, 0, 0, 0);

grid.Visibility = Visibility.Visible;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

namespace SDWP

{

public static class WelcomePageRightGridAnimations

{

public static List<Grid> RightGridsList { get; set; }

#region Show the right grids

public static void ShowTheGrid(Grid grid)

{

MakeAllRightGridsCollapsed();

LowerTheGrid(grid);

}

public static void HideTheGrid(Grid grid)

{

UpperTheGrid(grid);

}

private static void UpperTheGrid(Grid grid)

{

ThicknessAnimation upperAnimation = new ThicknessAnimation

{

From = grid.Margin,

To = new Thickness(0, -500, 0, 0),

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop,

};

upperAnimation.Completed += (sender, eArgs) =>

{

grid.Visibility = Visibility.Collapsed;

};

grid.BeginAnimation(FrameworkElement.MarginProperty, upperAnimation);

//this is the application description grid, we make it visible in the end of the animation

RightGridsList[0].Visibility = Visibility.Visible;

}

private static void LowerTheGrid(Grid grid)

{

SetInitalParamsOfAnimatedGrid(grid);

ThicknessAnimation lowerAnimation = new ThicknessAnimation

{

From = grid.Margin,

To = new Thickness(0, 0, 0, 0),

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

lowerAnimation.Completed += (sender, eArgs) =>

{

SetFinalParamsOfAnimatedGrid(grid);

};

grid.BeginAnimation(FrameworkElement.MarginProperty, lowerAnimation);

}

private static void SetFinalParamsOfAnimatedGrid(Grid grid)

{

grid.Margin = new Thickness(0, 0, 0, 0);

grid.Visibility = Visibility.Visible;

}

private static void SetInitalParamsOfAnimatedGrid(Grid grid)

{

grid.Margin = new Thickness(0, -500, 0, 0);

grid.Visibility = Visibility.Visible;

}

private static void MakeAllRightGridsCollapsed()

{

foreach (Grid grid in RightGridsList)

{

grid.Visibility = Visibility.Collapsed;

}

}

private static void MakeAllGridCollapsedInsteadOfOne(Grid grid)

{

foreach (Grid gridFromList in RightGridsList)

{

if (gridFromList != grid)

grid.Visibility = Visibility.Collapsed;

}

}

#endregion

}

}

<Application x:Class="SDWP.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:SDWP">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="AppResources.xaml"/>

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using SDWP.Factories;

using SDWP.Exceptions;

using SDWP.Interfaces;

using System.Threading;

using System.Runtime.InteropServices;

using System.Reflection;

namespace SDWP

{

public partial class App : Application

{

private static ISdwpAbstractFactory AbstractFactory { get; set; }

private static IExceptionHandler ExceptionHandler { get; set; }

App()

{

InitializeComponent();

AbstractFactory = new SdwpAbstractFactory();

ExceptionHandler = AbstractFactory.GetExceptionHandler(Dispatcher);

}

[STAThread]

static void Main()

{

string guid = Marshal.GetTypeLibGuidForAssembly(Assembly.GetExecutingAssembly()).ToString();

using (var mutex = new Mutex(true, guid))

{

try

{

if (!mutex.WaitOne(TimeSpan.FromMilliseconds(5), false))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Приложение уже запущено",

MessageBoxButton.OK);

return;

}

App app = new App();

MainWindow mainWindow = new MainWindow();

app.Run(mainWindow);

}

catch (Exception ex)

{

SDWPMessageBox.ShowSDWPMessageBox("Фатальная ошибка", "Произошла фатальная ошибка," +

"приложение будет закрыто, попробуйте перезапустить или перустановить его. " +

"Сейчас будет показано сообщение об ошибке.",

MessageBoxButton.OK);

ExceptionHandler.HandleWithMessageBox(ex);

}

finally

{

mutex.ReleaseMutex();

}

}

}

}

}

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:SDWP">

<!--Fonts for text-->

<FontFamily x:Key="mainFontFamily">Arial</FontFamily>

<SolidColorBrush x:Key="mainThemeBrush" Color="OrangeRed"/>

<Color x:Key="mainThemeColor">OrangeRed</Color>

<SolidColorBrush x:Key="subparagraphBorderBrush" Color="LightGray"/>

<SolidColorBrush x:Key="defaultTextColorBrush" Color="Black"/>

<SolidColorBrush x:Key="paragraphsTopOptionBrush" Color="#808080"/>

<Color x:Key="paragraphsTopOptionColor">#808080</Color>

<!--Menu option background brushes and colors-->

<SolidColorBrush x:Key="documentsLeftGridBackgroundBrush" Color="#E6E6E6"/>

<Color x:Key="documentsLeftGridBackgroundColor">#E6E6E6</Color>

<SolidColorBrush x:Key="mainPageItemsGridBackgroundBrush" Color="#F0F0F0"/>

<Color x:Key="mainPageItemsGridBackgroundColor">#F0F0F0</Color>

<SolidColorBrush x:Key="mainPageParagraphsGridBackgroundBrush" Color="White"/>

<Color x:Key="mainPageParagraphsGridBackgroundColor">#FFFFFF</Color>

<SolidColorBrush x:Key="mainWindowTopOptionsGridBavkgroundBrush" Color="White"/>

<SolidColorBrush x:Key="propertyNameForeground" Color="Gray"/>

<SolidColorBrush x:Key="devisionRectangleFill" Color="Gray"/>

<!--Create new objects (Documents, items etc) windows styles-->

<Style x:Key="createNewObjectWindowsBtnStyle" TargetType="Button">

<Setter Property="OverridesDefaultStyle" Value="True"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Width" Value="90"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="13"/>

<Setter Property="Foreground" Value="OrangeRed"/>

<Setter Property="BorderThickness" Value="2"/>

<Setter Property="BorderBrush" Value="OrangeRed"/>

<Setter Property="Background" Value="White"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="Margin" Value="0, 0, 20, 0"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border Name="border"

CornerRadius="5"

BorderBrush="{StaticResource mainThemeBrush}"

BorderThickness="1"

SnapsToDevicePixels="True"

Background="{TemplateBinding Background}">

<TextBlock.Foreground>

<SolidColorBrush x:Name="tbForeground" Color="White"/>

</TextBlock.Foreground>

<ContentPresenter HorizontalAlignment="Center"

VerticalAlignment="Center"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup Name="CommonStates">

<VisualState Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetProperty="Background.Color"

Storyboard.TargetName="border"

To="White"

SpeedRatio="100"/>

<ColorAnimation Storyboard.TargetName="tbForeground"

Storyboard.TargetProperty="Color"

To="{StaticResource mainThemeColor}"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

<VisualState Name="Normal">

<Storyboard>

<ColorAnimation Storyboard.TargetProperty="Background.Color"

Storyboard.TargetName="border"

To="{StaticResource mainThemeColor}"

SpeedRatio="100"/>

<ColorAnimation Storyboard.TargetName="tbForeground"

Storyboard.TargetProperty="Color"

To="White"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="createNewObjectWindowsPropertyHeadStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="FontWeight" Value="SemiBold"/>

</Style>

<Style x:Key="createNewObjectWindowsPropertyValueStyle" TargetType="TextBox">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="{StaticResource defaultTextColorBrush}"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="FontSize" Value="16"/>

<Setter Property="Padding" Value="5, 3, 5, 3"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="Background" Value="Transparent"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="1"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="10"

Background="{TemplateBinding Background}">

<ScrollViewer x:Name="PART\_ContentHost"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

VerticalScrollBarVisibility="Disabled"

Margin="{TemplateBinding Padding}"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup Name="CommonStates">

<VisualState Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="BorderBrush.Color"

SpeedRatio="100"

To="#FF8C00"/>

</Storyboard>

</VisualState>

<VisualState Name="Normal">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="BorderBrush.Color"

SpeedRatio="100"

To="{StaticResource mainThemeColor}"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="createNewObjectWindowsRadioBtnStyle" TargetType="RadioButton">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="{StaticResource defaultTextColorBrush}"/>

<Setter Property="FontSize" Value="14"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="BorderBrush" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Background" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="RadioButton">

<BulletDecorator Background="Transparent">

<BulletDecorator.Bullet>

<Grid Width="15"

Height="15">

<Ellipse x:Name="Border"

StrokeThickness="1"

Stroke="{StaticResource mainThemeBrush}"

Fill="Transparent"/>

<Ellipse x:Name="CheckMark"

Margin="4"

Fill="{StaticResource mainThemeBrush}"

Visibility="Collapsed"/>

</Grid>

</BulletDecorator.Bullet>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup Name="CommonStates">

<VisualState Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetName="Border"

Storyboard.TargetProperty="Stroke.Color"

SpeedRatio="100"

To="#FF8C00"/>

</Storyboard>

</VisualState>

<VisualState Name="Normal"/>

</VisualStateGroup>

<VisualStateGroup Name="CheckStates">

<VisualState Name="Checked">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="CheckMark"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0"

Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState Name="Unchecked">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="CheckMark"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0"

Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Indeterminate" />

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<ContentPresenter Margin="4,0,0,0"

VerticalAlignment="Center"

HorizontalAlignment="Left"

RecognizesAccessKey="True" />

</BulletDecorator>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</ResourceDictionary>

<Window x:Class="SDWP.CreateNewDocumentationWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новую документацию"

Height="200"

Width="400"

MinHeight="200"

MinWidth="400"

MaxHeight="200"

MaxWidth="400">

<Window.Content>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="400"

Height="200">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите имя документации: "

Margin="20, 10, 0, 0"/>

<TextBox x:Name="documentationNameTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Margin="20, 10, 0, 0"/>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewDocumentBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewDocumentation"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Window.Content>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.IO;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using SDWP.Factories;

using SDWP.Interfaces;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для CreateNewDocumentationWindow.xaml

/// </summary>

public partial class CreateNewDocumentationWindow : Window

{

#region Services and factories

private ISdwpAbstractFactory SdwpAbstractFactory { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

private ILocalDocumentationService LocalDocumentationService { get; }

private ICloudDocumentationService CloudDocumentationService { get; }

#endregion

#region Properties

private TextBox DocumentationNameTextBox { get; set; }

#endregion

#region Constructors and initialize methods

public CreateNewDocumentationWindow()

{

InitializeComponent();

InitializeProperties();

InitializeServices();

}

public CreateNewDocumentationWindow(ILocalDocumentationService localDocumentationService)

: this()

{

LocalDocumentationService = localDocumentationService;

}

public CreateNewDocumentationWindow(ICloudDocumentationService cloudDocumentationService)

: this()

{

CloudDocumentationService = cloudDocumentationService;

}

private void InitializeProperties()

{

DocumentationNameTextBox = documentationNameTextBox;

}

private void InitializeServices()

{

SdwpAbstractFactory = new SdwpAbstractFactory();

ExceptionHandler = SdwpAbstractFactory.GetExceptionHandler(Dispatcher);

}

#endregion

#region Event handlers

private void CancelCreation(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

private async void CreateNewDocumentation(object sender, RoutedEventArgs e)

{

try

{

string documentationName = DocumentationNameTextBox.Text;

if (string.IsNullOrEmpty(documentationName) || string.IsNullOrWhiteSpace(documentationName))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Введите имя документации", MessageBoxButton.OK);

return;

}

Documentation documentation = GetNewDocumentation(documentationName);

if (CloudDocumentationService == null)

{

documentation.StorageType = StorageType.Local;

LocalDocumentation localDocumentation = new LocalDocumentation(documentation, new List<Document>());

localDocumentation.DocumentationPath = System.IO.Path.Combine(LocalDocumentationService.StoragePath,

localDocumentation.Documentation.Name + ".sdwp");

await LocalDocumentationService.CreateLocalDocumentationFile(localDocumentation);

}

else

{

documentation.StorageType = StorageType.Cloud;

await CloudDocumentationService.InsertDocumentation(documentation);

}

DialogResult = true;

Close();

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private Documentation GetNewDocumentation(string name)

{

return new Documentation()

{

Access = Access.Private,

AuthorID = UserInfo.CurrentUser.ID,

AuthorName = UserInfo.CurrentUser.Name,

CreationDate = DateTime.Now,

Name = name,

UpdatedAt = DateTime.Now,

ManagerName = string.Empty,

ProjectCode = string.Empty,

ProjectName = string.Empty,

SoftwareEngineerName = string.Empty,

TeamLeadName = string.Empty

};

}

#endregion

}

}

<Window x:Class="SDWP.CreateNewDocumentWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новый документ"

Height="400"

Width="400"

MinHeight="300"

MinWidth="400"

MaxHeight="300"

MaxWidth="400">

<Window.Resources>

<Style x:Key="templatesTypesTextBlockStyle" TargetType="TextBlock">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontWeight" Value="Light"/>

<EventSetter Event="MouseEnter" Handler="TemplateTypesTextBlockMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="TemplateTypesTextBlockMouseLeave"/>

</Style>

<Style x:Key="templateComboBoxStyle" TargetType="ComboBox">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Width" Value="190"/>

<Setter Property="Padding" Value="0, 5, 0, 5"/>

</Style>

<DataTemplate x:Key="localTemplatesComboBoxItemTemplate">

<StackPanel Orientation="Vertical"

VerticalAlignment="Center"

HorizontalAlignment="Left">

<TextBlock VerticalAlignment="Center"

HorizontalAlignment="Left"

Margin="10, 0, 0, 0"

FontFamily="Arial, Verdana"

Foreground="Black"

FontSize="14"

Text="{Binding FileName}"/>

</StackPanel>

</DataTemplate>

<DataTemplate x:Key="cloudTemplatesComboBoxItemTemplate">

<StackPanel Orientation="Vertical"

VerticalAlignment="Center"

HorizontalAlignment="Left">

<TextBlock VerticalAlignment="Center"

HorizontalAlignment="Left"

Margin="10, 0, 0, 0"

FontFamily="Arial, Verdana"

Foreground="Black"

FontSize="14"

Text="{Binding TemplateName}"/>

</StackPanel>

</DataTemplate>

</Window.Resources>

<Window.Content>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="400"

Height="300">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите имя документа: "

Margin="20, 10, 0, 0"/>

<TextBox x:Name="documentNameTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Margin="20, 10, 0, 0"/>

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Выберете шаблон для документа: "

Margin="20, 20, 0, 0"/>

<StackPanel x:Name="templateTypesStackPanel"

Orientation="Horizontal"

Margin="20, 10, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Left">

<TextBlock x:Name="offlineTemplatesTextBlock"

Text="Локальные шаблоны"

Style="{StaticResource templatesTypesTextBlockStyle}"

MouseDown="LocalTemplatesTextBlockMouseDown"/>

<TextBlock x:Name="onlineTemplatesTextBlock"

Style="{StaticResource templatesTypesTextBlockStyle}"

Margin="20, 0, 0, 0"

Text="Облачные шаблоны"

MouseDown="CloudTemplatesTextBlockMouseDown"/>

</StackPanel>

<Grid x:Name="templatesComboBoxesGrid">

<ComboBox x:Name="localTemplatesComboBox"

Margin="20, 10, 0, 0"

Style="{StaticResource templateComboBoxStyle}"

ItemTemplate="{StaticResource localTemplatesComboBoxItemTemplate}"/>

<ComboBox x:Name="cloudTemplatesComboBox"

Margin="20, 10, 0, 0"

Visibility="Collapsed"

Style="{StaticResource templateComboBoxStyle}"

ItemTemplate="{StaticResource cloudTemplatesComboBoxItemTemplate}"/>

</Grid>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewDocumentBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewDocument"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Window.Content>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.IO;

using System.Data.SqlClient;

using ApplicationLib.Models;

using ApplicationLib.Services;

using ApplicationLib.Interfaces;

using ApplicationLib.Database;

using ApplicationLib.Factories;

using SDWP.Factories;

using SDWP.Interfaces;

using SDWP.Exceptions;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для CreateNewDocumentWindow.xaml

/// </summary>

public partial class CreateNewDocumentWindow : Window

{

#region Properties

private List<Document> Documents { get; }

private Documentation Documentation { get; }

private List<LocalTemplate> LocalTemplates { get; set; }

private List<Template> CloudTemplates { get; set; }

private string DefaultStoragePath { get; }

private TextBlock LocalTemplatesTextBlock { get; set; }

private TextBlock CloudTemplatesTextBlock { get; set; }

private ComboBox LocalTemplatesComboBox { get; set; }

private ComboBox CloudTemplatesComboBox { get; set; }

private TextBox DocumentNameTextBox { get; set; }

#endregion

#region Services and factories

private IServiceAbstractFactory ServiceAbstractFactory { get; set; }

private ISdwpAbstractFactory SdwpAbstractFactory { get; set; }

private ICloudTemplateService CloudTemplateService { get; set; }

private ILocalTemplateService LocalTemplateService { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

#endregion

#region Constrcutors and initialize methods

public CreateNewDocumentWindow(List<Document> documents, Documentation documentation)

{

InitializeComponent();

InitializeServices();

InitializeProperties();

DefaultStoragePath = System.IO.Path.Combine(Directory.GetCurrentDirectory(), "Templates");

Documentation = documentation;

Documents = documents;

}

private void InitializeProperties()

{

LocalTemplatesTextBlock = offlineTemplatesTextBlock;

CloudTemplatesTextBlock = onlineTemplatesTextBlock;

LocalTemplatesComboBox = localTemplatesComboBox;

CloudTemplatesComboBox = cloudTemplatesComboBox;

DocumentNameTextBox = documentNameTextBox;

}

private void InitializeServices()

{

ServiceAbstractFactory = new ServiceAbstractFactory();

SdwpAbstractFactory = new SdwpAbstractFactory();

CloudTemplateService = ServiceAbstractFactory.GetCloudTemplateService();

LocalTemplateService = ServiceAbstractFactory.GetLocalTemplateService();

ExceptionHandler = SdwpAbstractFactory.GetExceptionHandler(Dispatcher);

}

#endregion

#region Event handlers

private void CancelCreation(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

/// <summary>

/// Creates a new document with items from a selected template

/// </summary>

private void CreateNewDocument(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(DocumentNameTextBox.Text) ||

string.IsNullOrWhiteSpace(DocumentNameTextBox.Text))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Введите имя документа", MessageBoxButton.OK);

return;

}

Document document = GetDocumentObject();

if (LocalTemplatesComboBox.Visibility == Visibility.Visible)

{

LocalTemplate selectedTemplate = LocalTemplatesComboBox.SelectedItem as LocalTemplate;

document.Items = selectedTemplate.Template.Items;

Documents.Add(document);

}

else

{

if (CloudTemplatesComboBox.SelectedItem is Template selectedTemplate)

{

document.Items = selectedTemplate.Items;

Documents.Add(document);

}

}

DialogResult = true;

Close();

}

/// <summary>

/// Creates and initializes the document object with necessary values

/// </summary>

private Document GetDocumentObject()

{

return new Document()

{

Name = DocumentNameTextBox.Text,

CreationDate = DateTime.Now,

AuthorID = UserInfo.CurrentUser.ID,

UpdatedAt = DateTime.Now,

AuthorName = UserInfo.CurrentUser.Name,

Access = Access.Private,

DocumentationID = Documentation.ID,

};

}

#region Templates selection and uploading

private async void LocalTemplatesTextBlockMouseDown(object sender, MouseButtonEventArgs e)

{

LocalTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.OrangeRed);

CloudTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.Black);

LocalTemplatesComboBox.Visibility = Visibility.Visible;

CloudTemplatesComboBox.Visibility = Visibility.Collapsed;

await GetLocalTemplates();

AddEmptyTemplateToLocalTemplates();

LocalTemplatesComboBox.ItemsSource = null;

LocalTemplatesComboBox.ItemsSource = LocalTemplates;

if (LocalTemplates.Count > 0)

LocalTemplatesComboBox.SelectedIndex = 0;

}

private async Task GetLocalTemplates()

{

LocalTemplates = new List<LocalTemplate>();

try

{

LocalTemplateService.StoragePath = DefaultStoragePath;

LocalTemplates = (await LocalTemplateService.GetLocalTemplates()).ToList();

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async void CloudTemplatesTextBlockMouseDown(object sender, MouseButtonEventArgs e)

{

LocalTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.Black);

CloudTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.OrangeRed);

LocalTemplatesComboBox.Visibility = Visibility.Collapsed;

CloudTemplatesComboBox.Visibility = Visibility.Visible;

await GetCloudTemplates();

CloudTemplatesComboBox.ItemsSource = null;

CloudTemplatesComboBox.ItemsSource = CloudTemplates;

if (CloudTemplates.Count > 0)

{

CloudTemplatesComboBox.SelectedIndex = 0;

}

}

private async Task GetCloudTemplates()

{

CloudTemplates = new List<Template>();

try

{

CloudTemplates = (await CloudTemplateService.GetUserTemplates(UserInfo.CurrentUser.ID)).ToList();

}

catch (SqlException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

#endregion

/// <summary>

/// Adds an local template with an empty Items so that user can create an empty document

/// </summary>

private void AddEmptyTemplateToLocalTemplates()

{

Template emptyTemplate = new Template()

{

Items = new List<Item>()

};

LocalTemplate emptyLocalTemplate = new LocalTemplate(emptyTemplate)

{

FileName = "Пустой документ"

};

List<LocalTemplate> localTemplates = new List<LocalTemplate>

{

emptyLocalTemplate

};

localTemplates.AddRange(LocalTemplates);

LocalTemplates = localTemplates;

}

private void TemplateTypesTextBlockMouseEnter(object sender, RoutedEventArgs e)

{

(sender as TextBlock).TextDecorations.Add(TextDecorations.Underline);

}

private void TemplateTypesTextBlockMouseLeave(object sender, RoutedEventArgs e)

{

(sender as TextBlock).TextDecorations.Clear();

}

#endregion

}

}

<Window x:Class="SDWP.CreateNewImageWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новый рисунок"

Height="400"

Width="400"

MinHeight="300"

MinWidth="400"

MaxHeight="300"

MaxWidth="400">

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="400"

Height="300">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Orientation="Vertical"

Background="Transparent">

<TextBlock Style="{DynamicResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите название рисунка: "

Margin="20, 10, 0, 0"/>

<TextBox x:Name="imageTitleTextBox"

Style="{DynamicResource createNewObjectWindowsPropertyValueStyle}"

GotFocus="ImageTitleTextBoxGotFocus"

LostFocus="ImageTitleTextBoxLostFocus"

Margin="20, 10, 0, 0"/>

<TextBlock Style="{DynamicResource createNewObjectWindowsPropertyHeadStyle}"

Text="Выберете рисунок"

Margin="20, 20, 0, 0"/>

<StackPanel Orientation="Horizontal"

Margin="0, 10, 0, 0">

<TextBox x:Name="imagePathTextBox"

Style="{DynamicResource createNewObjectWindowsPropertyValueStyle}"

Text="Путь до рисунка..."

Width="200"

Margin="20, 0, 0, 0"

GotFocus="ImagePathTextBoxGotFocus"

LostFocus="ImagePathTextBoxLostFocus"/>

<Button Style="{DynamicResource createNewObjectWindowsBtnStyle}"

Content="Обзор"

Margin="10, 0, 0, 0"

VerticalAlignment="Center"

Click="SelectImage"/>

</StackPanel>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewItemBtn"

Style="{DynamicResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewImage"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.IO;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using SDWP.Exceptions;

using SDWP.Factories;

using SDWP.Interfaces;

using Microsoft.Win32;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для CreateNewImageWindow.xaml

/// </summary>

public partial class CreateNewImageWindow : Window

{

#region Properties

private Item ContentItem { get; }

private string ImagePathTextBoxHint { get; } = "Путь до рисунка...";

private string ImageTitleTextBoxHint { get; } = "Введите название рисунка: ";

private TextBox ImagePathTextBox { get; }

private TextBox ImageTitleTextBox { get; }

#endregion

#region Services and factories

private ISdwpAbstractFactory AbstractFactory { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

#endregion

public CreateNewImageWindow(Item currentContentItem)

{

InitializeComponent();

ContentItem = currentContentItem;

ImagePathTextBox = imagePathTextBox;

ImageTitleTextBox = imageTitleTextBox;

InitializeServices();

}

private void InitializeServices()

{

AbstractFactory = new SdwpAbstractFactory();

ExceptionHandler = AbstractFactory.GetExceptionHandler(Dispatcher);

}

#region Event handlers

private void CancelCreation(object sender, RoutedEventArgs e)

{

Close();

}

private void ImagePathTextBoxGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(ImagePathTextBoxHint, sender as TextBox);

}

private void ImagePathTextBoxLostFocus(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(ImagePathTextBoxHint, sender as TextBox);

}

private void ImageTitleTextBoxLostFocus(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(ImagePathTextBoxHint, sender as TextBox);

}

private void ImageTitleTextBoxGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(ImagePathTextBoxHint, sender as TextBox);

}

private void OnTextBoxGotFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == textBoxHint)

textBox.Text = string.Empty;

}

private void OnTextBoxLostFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == string.Empty)

textBox.Text = textBoxHint;

}

public void SelectImage(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog()

{

Title = "Выберете рисунок",

CheckFileExists = true,

Filter = "JPG files (\*jpg)|\*jpg|PNG files (\*png)|\*png",

Multiselect = false

};

if (openFileDialog.ShowDialog() == true)

{

ImagePathTextBox.Text = openFileDialog.FileName;

}

}

private void CreateNewImage(object sender, RoutedEventArgs e)

{

try

{

string filePath = ImagePathTextBox.Text;

string imageTitle = ImageTitleTextBox.Text;

if (string.IsNullOrEmpty(imageTitle) || string.IsNullOrWhiteSpace(imageTitle))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Введите имя рисунка", MessageBoxButton.OK);

return;

}

byte[] imageByteArr = GetByteRepresentationOfImage(filePath);

ParagraphImage paragraphImage = new ParagraphImage(imageByteArr)

{

Title = imageTitle

};

Paragraph paragraph = new Paragraph(typeof(ParagraphImage).Name, paragraphImage);

(paragraph as IParentableParagraph).SetParents(ContentItem, ContentItem.Paragraphs);

ContentItem.Paragraphs.Add(paragraph);

DialogResult = true;

Close();

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private byte[] GetByteRepresentationOfImage(string filePath)

{

byte[] imageByteArr;

using (var fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))

{

fs.Seek(0, SeekOrigin.Begin);

imageByteArr = new byte[fs.Length];

fs.Read(imageByteArr, 0, (int)fs.Length);

}

return imageByteArr;

}

#endregion

}

}

<Window x:Class="SDWP.CreateNewItemWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новый пункт"

Height="300"

Width="400"

MinHeight="300"

MinWidth="400"

MaxHeight="300"

MaxWidth="400">

<Grid>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="400"

Height="300">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите имя пункта: "

Margin="20, 10, 0, 0"/>

<TextBox x:Name="itemNameTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Margin="20, 10, 0, 0"/>

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Выберете режим создания: "

Margin="20, 20, 0, 0"/>

<RadioButton x:Name="contentCreationModeRadioBtn"

Style="{StaticResource createNewObjectWindowsRadioBtnStyle}"

Content="Создать пункт с контентом"

IsChecked="True"

Margin="20, 10, 0, 0"/>

<RadioButton x:Name="itemsCreationModeRadioBtn"

Style="{StaticResource createNewObjectWindowsRadioBtnStyle}"

Content="Создать пункт со списком пунктов"

Margin="20, 5, 0, 0"/>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewItemBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewItem"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Grid>

</Window>

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using ApplicationLib.Interfaces;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для CreateNewItemWindow.xaml

/// </summary>

public partial class CreateNewItemWindow : Window

{

private List<Item> CurrentItemsList { get; set; }

private Item CurrentItem { get; set; }

#region Constructors

public CreateNewItemWindow() { InitializeComponent(); }

public CreateNewItemWindow(List<Item> items, Item currentItem)

{

InitializeComponent();

CurrentItemsList = items;

CurrentItem = currentItem;

}

#endregion

/// <summary>

/// Creates a new item. If the content creation mode is selected then item object

/// is initialized with the null in Items and empty list in Paragraphs, if the items creation

/// mode is selected then everything is conversed.

/// </summary>

private void CreateNewItem(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(itemNameTextBox.Text))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Введите имя нового пункта", MessageBoxButton.OK);

return;

}

if (contentCreationModeRadioBtn.IsChecked == true)

{

Item newItem = new Item()

{

Items = null,

Paragraphs = new List<Paragraph>(),

Name = itemNameTextBox.Text

};

(newItem as IParentableItem).SetParents(CurrentItem, CurrentItemsList);

CurrentItemsList.Add(newItem);

}

else

{

Item newItem = new Item()

{

Items = new List<Item>(),

Paragraphs = null,

Name = itemNameTextBox.Text

};

(newItem as IParentableItem).SetParents(CurrentItem, CurrentItemsList);

CurrentItemsList.Add(newItem);

}

DialogResult = true;

Close();

}

private void CancelCreation(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

}

}

<Window x:Class="SDWP.CreateNewNumberedListWindow"

x:Name="createNewNumberedListWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Создать новый пункт"

Height="200"

Width="400"

MinHeight="200"

MinWidth="400"

MaxHeight="200"

MaxWidth="400">

<Grid>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="{Binding ElementName=createNewNumberedListWindow, Path=Width}"

Height="{Binding ElementName=createNewNumberedListWindow, Path=Height}">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите название списка: "

Margin="20, 10, 0, 0"/>

<TextBox x:Name="numberedListTitleTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Width="200"

Margin="20, 10, 0, 0"

GotFocus="NumberedListTitleTextBoxGotFocus"

LostFocus="NumberedListTitleTextBoxGotFocus"/>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewItemBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewNumberedList"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

namespace SDWP

{

public partial class CreateNewNumberedListWindow : Window

{

private string NumberedListTitleHintText { get; } = "Название списка...";

private TextBox NumberedListTitleTextBox { get; }

private Item ContentItem { get; }

public CreateNewNumberedListWindow(Item contentItem)

{

InitializeComponent();

NumberedListTitleTextBox = numberedListTitleTextBox;

ContentItem = contentItem;

}

private void NumberedListTitleTextBoxGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(NumberedListTitleHintText, sender as TextBox);

}

private void NumberedListTitleTextBoxLostFocus(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(NumberedListTitleHintText, sender as TextBox);

}

private void OnTextBoxGotFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == textBoxHint)

textBox.Text = string.Empty;

}

private void OnTextBoxLostFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == string.Empty)

textBox.Text = textBoxHint;

}

private void CancelCreation(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

private void CreateNewNumberedList(object sender, RoutedEventArgs e)

{

string listTitle = NumberedListTitleTextBox.Text;

if (!string.IsNullOrEmpty(listTitle) && !(listTitle == NumberedListTitleHintText))

{

NumberedList numberedList = new NumberedList(new List<NumberedListElement>()

{

new NumberedListElement(string.Empty)

})

{

Title = listTitle,

};

Paragraph paragraph = new Paragraph(typeof(NumberedList).Name, numberedList);

(paragraph as IParentableParagraph).SetParents(ContentItem, ContentItem.Paragraphs);

ContentItem.Paragraphs.Add(paragraph);

DialogResult = true;

Close();

}

}

}

}

<Window x:Class="SDWP.CreateNewSubparagraphWindow"

x:Name="createNewSubparagraphWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новый пункт"

Height="200"

Width="400"

MinHeight="200"

MinWidth="400"

MaxHeight="200"

MaxWidth="400">

<Grid>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="{Binding ElementName=createNewSubparagraphWindow, Path=Width}"

Height="{Binding ElementName=createNewSubparagraphWindow, Path=Height}">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите навание параграфа: "

Margin="20, 10, 0, 0"/>

<TextBox x:Name="subparagraphTitleTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Width="200"

Margin="20, 10, 0, 0"

GotFocus="SubparagraphTitleTextBoxGotFocus"

LostFocus="SubparagraphTitleTextBoxGotFocus"/>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewItemBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewSubparagraph"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

namespace SDWP

{

public partial class CreateNewSubparagraphWindow : Window

{

#region Properties

private string SubparagraphTitleTextBoxHint { get; } = "Имя абзаца...";

private TextBox SubparagraphTitleTextBox { get; }

private Item CurrentItem { get; }

#endregion

public CreateNewSubparagraphWindow(Item currentItem)

{

InitializeComponent();

CurrentItem = currentItem;

SubparagraphTitleTextBox = subparagraphTitleTextBox;

}

#region Event handlers

private void SubparagraphTitleTextBoxGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(SubparagraphTitleTextBoxHint, sender as TextBox);

}

private void SubparagraphTitleTextBoxGotFocu(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(SubparagraphTitleTextBoxHint, sender as TextBox);

}

private void OnTextBoxGotFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == textBoxHint)

textBox.Text = string.Empty;

}

private void OnTextBoxLostFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == string.Empty)

textBox.Text = textBoxHint;

}

private void CancelCreation(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

private void CreateNewSubparagraph(object sender, RoutedEventArgs e)

{

string subparagraphTitle = SubparagraphTitleTextBox.Text;

if (CheckSubparagraphTitle(subparagraphTitle))

{

Subparagraph subparagraph = new Subparagraph(string.Empty)

{

Title = subparagraphTitle

};

Paragraph paragraph= new Paragraph(typeof(Subparagraph).Name, subparagraph);

(paragraph as IParentableParagraph).SetParents(CurrentItem, CurrentItem.Paragraphs);

CurrentItem.Paragraphs.Add(paragraph);

DialogResult = true;

Close();

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка при создании абзаца", "Непрвильно введены данные",

MessageBoxButton.OK);

}

}

private bool CheckSubparagraphTitle(string title)

{

return !string.IsNullOrEmpty(title) && !(title == SubparagraphTitleTextBoxHint);

}

#endregion

}

}

<Window x:Class="SDWP.CreateNewTableWindow"

x:Name="createNewTableWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новую таблицу"

Height="400"

Width="400"

MinHeight="400"

MinWidth="400"

MaxHeight="400"

MaxWidth="400">

<Grid>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="{Binding ElementName=createNewTableWindow, Path=Width}"

Height="{Binding ElementName=createNewTableWindow, Path=Height}">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите имя таблицы"

Margin="20, 20, 0, 0"/>

<TextBox x:Name="tableTitleTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Width="200"

Margin="20, 10, 0, 0"/>

<Rectangle Width="{Binding ElementName=createNewTableWindow, Path=Width}"

Height="1"

Margin="20, 10, 20, 0"

Fill="LightGray"/>

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите размеры таблицы: "

Margin="20, 10, 0, 0"/>

<StackPanel Orientation="Horizontal">

<TextBox x:Name="tableWidthTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Text="Строки..."

Width="100"

Margin="20, 10, 0, 0"

GotFocus="TableWidthTextBoxTextGotFocus"

LostFocus="TableWidthTextBoxTextLostFocus"/>

<TextBox x:Name="tableHeightTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Text="Столбцы..."

Width="100"

Margin="20, 10, 0, 0"

GotFocus="TableHeightTextBoxGotFocus"

LostFocus="TableHeightTextBoxLostFocus"/>

</StackPanel>

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Или выберете файл C#"

Margin="20, 20, 0, 0"/>

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="для сканирования кода: "

Margin="20, 0, 0, 0"/>

<StackPanel Orientation="Horizontal"

Margin="0, 10, 0, 0">

<TextBox x:Name="filePathTexxBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Text="Путь до файла..."

Width="200"

Margin="20, 0, 0, 0"

GotFocus="FilePathTextBoxGotFocus"

LostFocus="FilePathTextBoxLostFocus"/>

<Button Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Обзор"

Margin="10, 0, 0, 0"

VerticalAlignment="Center"

Click="SelectAssembly"/>

</StackPanel>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewItemBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewTable"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.IO;

using Microsoft.Win32;

using FileLib.Interfaces;

using FileLib.FileParsers;

using SDWP.Factories;

using SDWP.Exceptions;

using SDWP.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

namespace SDWP

{

public partial class CreateNewTableWindow : Window

{

#region Propeties

private string TableWidthTextBoxHint { get; } = "Строки...";

private string TableHeightTextBoxHint { get; } = "Столбцы...";

private string FilePathTextBoxHint { get; } = "Путь до файла...";

private Item CurrentItem { get; }

private TextBox TableWidthTextBox { get; }

private TextBox TableHeightTextBox { get; }

private TextBox FilePathTextBox { get; }

private TextBox TableTitleTextBox { get; }

private ISdwpAbstractFactory AbstractFactory { get; set; }

private IFileParser FileParser { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

#endregion

public CreateNewTableWindow(Item currentItem)

{

InitializeComponent();

CurrentItem = currentItem;

TableWidthTextBox = tableWidthTextBox;

TableHeightTextBox = tableHeightTextBox;

FilePathTextBox = filePathTexxBox;

TableTitleTextBox = tableTitleTextBox;

InitializeServices();

}

private void InitializeServices()

{

AbstractFactory = new SdwpAbstractFactory();

FileParser = AbstractFactory.GetFileParser();

ExceptionHandler = AbstractFactory.GetExceptionHandler(Dispatcher);

}

#region Event handlers

private void TableWidthTextBoxTextGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(TableWidthTextBoxHint, sender as TextBox);

}

private void TableWidthTextBoxTextLostFocus(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(TableWidthTextBoxHint, sender as TextBox);

}

private void TableHeightTextBoxGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(TableHeightTextBoxHint, sender as TextBox);

}

private void TableHeightTextBoxLostFocus(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(TableHeightTextBoxHint, sender as TextBox);

}

private void FilePathTextBoxGotFocus(object sender, RoutedEventArgs e)

{

OnTextBoxGotFocus(FilePathTextBoxHint, sender as TextBox);

}

private void FilePathTextBoxLostFocus(object sender, RoutedEventArgs e)

{

OnTextBoxLostFocus(FilePathTextBoxHint, sender as TextBox);

}

private void OnTextBoxGotFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == textBoxHint)

textBox.Text = string.Empty;

}

private void OnTextBoxLostFocus(string textBoxHint, TextBox textBox)

{

if (textBox.Text == string.Empty)

textBox.Text = textBoxHint;

}

private void CreateNewTable(object sender, RoutedEventArgs e)

{

if (FilePathTextBox.Text != FilePathTextBoxHint)

CreateTableFromCSFile(FilePathTextBox.Text);

else

CreateEmptyTable();

}

private void CancelCreation(object sender, RoutedEventArgs e)

{

Close();

}

private void SelectAssembly(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog()

{

Filter = "(\*dll)|\*dll|(\*exe)|\*exe",

CheckFileExists = true,

CheckPathExists = true,

Title = "Выберете DLL или EXE файл для сканирования",

Multiselect = false,

};

if (openFileDialog.ShowDialog() == true)

{

FilePathTextBox.Text = openFileDialog.FileName;

}

}

#endregion

#region Table creation

private void CreateTableFromCSFile(string filePath)

{

try

{

Table[] tables = FileParser.GetAssemblyTables(filePath);

for (int i = 0; i<tables.Length; i++)

{

AddNewTable(tables[i]);

}

DialogResult = true;

Close();

}

catch (IOException ex)

{

DialogResult = false;

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

DialogResult = false;

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private void AddNewTable(Table table)

{

Paragraph paragraph = new Paragraph(typeof(Table).Name, table);

(paragraph as IParentableParagraph).SetParents(CurrentItem, CurrentItem.Paragraphs);

CurrentItem.Paragraphs.Add(paragraph);

}

private void CreateEmptyTable()

{

int tableRowCount = 0, tableColCount = 0;

string tableTitle = TableTitleTextBox.Text;

if (CheckTableWidthAndHeight(ref tableRowCount, ref tableColCount) &&

CheckTableTitle(tableTitle))

{

string[][] tableCells = GetEmptyTableCells(tableRowCount, tableColCount);

Table table = new Table(tableCells)

{

Title = tableTitle

};

AddNewTable(table);

DialogResult = true;

Close();

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка при создании таблицы", "Неверно введены данные",

MessageBoxButton.OK);

}

}

private string[][] GetEmptyTableCells(int rowNum, int colNum)

{

string[][] tableCells = new string[rowNum][];

for (int i = 0; i < rowNum; i++)

{

tableCells[i] = new string[colNum];

for (int j = 0; j < tableCells[i].Length; j++)

tableCells[i][j] = j.ToString();

}

return tableCells;

}

private bool CheckTableTitle(string title)

{

return !string.IsNullOrEmpty(title);

}

/// <summary>

/// Checks the correctness of the table width and height and id everything is OK sets the w and h ref variables

/// the values of table width and height

/// </summary>

private bool CheckTableWidthAndHeight(ref int rows, ref int cols)

{

bool result = int.TryParse(TableWidthTextBox.Text, out rows) | int.TryParse(TableHeightTextBox.Text, out cols);

result = result && !(rows <= 0 || rows >= 50) && !(cols <= 0 || cols >= 50);

return result;

}

#endregion

}

}

<Window x:Class="SDWP.CreateTemplateTreeViewParagraphWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

Title="Создать новый параграф"

Height="300"

Width="400"

MinHeight="300"

MinWidth="400"

MaxHeight="300"

MaxWidth="400">

<Window.Content>

<Grid>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="400"

Height="300">

<StackPanel x:Name="mainContentStackPanel"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 44"

Background="Transparent">

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Введите имя параграфа:"

Margin="20, 10, 0, 0"/>

<TextBox x:Name="paragraphNameTextBox"

Style="{StaticResource createNewObjectWindowsPropertyValueStyle}"

Text=""

Margin="20, 10, 0, 0"/>

<TextBlock Style="{StaticResource createNewObjectWindowsPropertyHeadStyle}"

Text="Выберете режим создания: "

Margin="20, 20, 0, 0"/>

<RadioButton x:Name="subparagraphCreationModeRadioBtn"

IsChecked="True"

Style="{StaticResource createNewObjectWindowsRadioBtnStyle}"

Content="Создать абзац"

Margin="20, 10, 0, 0"/>

<RadioButton x:Name="imageCreationModeRadioBtn"

Style="{StaticResource createNewObjectWindowsRadioBtnStyle}"

Content="Создать рисунок"

Margin="20, 5, 0, 0"/>

<RadioButton x:Name="tableCreationModeRadioBtn"

Style="{StaticResource createNewObjectWindowsRadioBtnStyle}"

Content="Создать таблицу"

Margin="20, 5, 0, 0"/>

<RadioButton x:Name="numberedListCreationModeRadioBtn"

Style="{StaticResource createNewObjectWindowsRadioBtnStyle}"

Content="Создать список"

Margin="20, 5, 0, 0"/>

</StackPanel>

<Grid VerticalAlignment="Bottom"

HorizontalAlignment="Stretch"

Height="44"

Margin="0, 0, 0, 38"

Background="#E1E1E1">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1">

<Button x:Name="createNewParagraphBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Создать"

Click="CreateNewParagraph"/>

<Button x:Name="cancelBtn"

Style="{StaticResource createNewObjectWindowsBtnStyle}"

Content="Отмена"

Click="CancelCreation"/>

</StackPanel>

</Grid>

</Grid>

</Grid>

</Window.Content>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using System.IO;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для CreateTemplateTreeViewParagraphWindow.xaml

/// </summary>

public partial class CreateTemplateTreeViewParagraphWindow : Window

{

private Item CurrentItem { get; set; }

#region Constructors

public CreateTemplateTreeViewParagraphWindow() { InitializeComponent(); }

public CreateTemplateTreeViewParagraphWindow(Item currentItem)

{

InitializeComponent();

CurrentItem = currentItem;

}

#endregion

private void CreateNewParagraph(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(paragraphNameTextBox.Text))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Введите имя нового параграфа", MessageBoxButton.OK);

return;

}

if (tableCreationModeRadioBtn.IsChecked == true)

{

Paragraph paragraph = new Paragraph()

{

ParagraphElement = new Table()

{

Title = paragraphNameTextBox.Text,

TableCells = new string[2][]

{

new string[2] {string.Empty, string.Empty},

new string[2] {string.Empty, string.Empty},

}

},

Type = "Table"

};

(paragraph as IParentableParagraph).SetParents(CurrentItem, CurrentItem.Paragraphs);

CurrentItem.Paragraphs.Add(paragraph);

}

else if (numberedListCreationModeRadioBtn.IsChecked == true)

{

Paragraph paragraph = new Paragraph()

{

ParagraphElement = new NumberedList()

{

Title = paragraphNameTextBox.Text,

ListElements = new List<NumberedListElement>()

},

Type = "NumberedList"

};

(paragraph as IParentableParagraph).SetParents(CurrentItem, CurrentItem.Paragraphs);

CurrentItem.Paragraphs.Add(paragraph);

}

else if (imageCreationModeRadioBtn.IsChecked == true)

{

Paragraph paragraph = new Paragraph()

{

ParagraphElement = new ParagraphImage()

{

Title = paragraphNameTextBox.Text,

ImageSource = GetResourceImageBytes("pack://application:,,,/Resources/defaultParagraphImageImage.png")

},

Type = "ParagraphImage"

};

(paragraph as IParentableParagraph).SetParents(CurrentItem, CurrentItem.Paragraphs);

CurrentItem.Paragraphs.Add(paragraph);

}

else if (subparagraphCreationModeRadioBtn.IsChecked == true)

{

Paragraph paragraph = new Paragraph()

{

ParagraphElement = new Subparagraph()

{

Title = paragraphNameTextBox.Text,

Text = string.Empty

},

Type = "Subparagraph"

};

(paragraph as IParentableParagraph).SetParents(CurrentItem, CurrentItem.Paragraphs);

CurrentItem.Paragraphs.Add(paragraph);

}

DialogResult = true;

Close();

}

private byte[] GetResourceImageBytes(string uri)

{

byte[] imageBytes;

Uri imageUri = new Uri(uri);

BitmapImage bitmapImage = new BitmapImage(imageUri);

using (var ms = new MemoryStream())

{

JpegBitmapEncoder encoder = new JpegBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(bitmapImage));

encoder.Save(ms);

imageBytes = ms.ToArray();

}

return imageBytes;

}

private void CancelCreation(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

}

}

<Page x:Class="SDWP.DocumentTemplatesPage"

x:Name="documentTemplatesPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:SDWP"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Page.Resources>

<Style x:Key="templateOptionsTextBlockStyle" TargetType="TextBlock">

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="FontSize" Value="13"/>

</Style>

<Style TargetType="ListBoxItem">

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="Margin" Value="0, 10, 0, 0"/>

<Setter Property="Background" Value="LightGray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ListBoxItem">

<Border x:Name="border"

SnapsToDevicePixels="true"

Background="{TemplateBinding Background}"

CornerRadius="5"

BorderThickness="1"

BorderBrush="Gray">

<ContentPresenter/>

</Border>

<ControlTemplate.Triggers>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="True" />

<Condition Property="IsSelected" Value="True" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="#f0f0f0"/>

</MultiTrigger.Setters>

</MultiTrigger>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="False" />

<Condition Property="IsSelected" Value="True" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="#f0f0f0"/>

</MultiTrigger.Setters>

</MultiTrigger>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="False" />

<Condition Property="IsSelected" Value="False" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="LightGray"/>

</MultiTrigger.Setters>

</MultiTrigger>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="True" />

<Condition Property="IsSelected" Value="False" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="#f0f0f0"/>

</MultiTrigger.Setters>

</MultiTrigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="ListBoxItemMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="ListBoxItemMouseLeave"/>

<EventSetter Event="PreviewMouseDown" Handler="ListBoxItemMouseDown"/>

</Style>

<DataTemplate x:Key="localTemplateListBoxDataTemplate">

<Grid VerticalAlignment="Center"

HorizontalAlignment="Center"

Margin="0, 0, 0, 0"

Width="280">

<Border x:Name="border"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="280"

Height="50"

BorderThickness="0"

Background="Transparent"

Padding="5"

BorderBrush="Gray"

Margin="0, 0, 0, 0">

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Height="50"

Width="280"

Orientation="Vertical">

<StackPanel VerticalAlignment="Center"

HorizontalAlignment="Left"

Orientation="Horizontal"

Height="50"

Width="280">

<Image Source="../Resources/documentationIcon.png"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="20"

Width="20"

Margin="5, -12, 0, 0"/>

<TextBlock Text="{Binding Path = FileName, UpdateSourceTrigger=PropertyChanged}"

Style="{StaticResource templateOptionsTextBlockStyle}"

Margin="10, -10, 0, 0"/>

</StackPanel>

</StackPanel>

</Border>

</Grid>

</DataTemplate>

<DataTemplate x:Key="cloudTemplateListBoxDataTemplate">

<Grid VerticalAlignment="Center"

HorizontalAlignment="Center"

Margin="0, 0, 0, 0"

Width="280">

<Border x:Name="border"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="280"

Height="50"

BorderThickness="0"

Background="Transparent"

Padding="5"

BorderBrush="Gray"

Margin="0, 0, 0, 0">

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Height="50"

Width="280"

Orientation="Vertical">

<StackPanel VerticalAlignment="Center"

HorizontalAlignment="Left"

Orientation="Horizontal"

Height="50"

Width="280">

<Image Source="../Resources/documentationIcon.png"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="20"

Width="20"

Margin="5, -12, 0, 0"/>

<TextBlock Text="{Binding Path = TemplateName, UpdateSourceTrigger=PropertyChanged}"

Style="{StaticResource templateOptionsTextBlockStyle}"

Margin="10, -10, 0, 0"/>

</StackPanel>

</StackPanel>

</Border>

</Grid>

</DataTemplate>

<Style x:Key="templatesListBoxStyle" TargetType="ListBox">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Background" Value="#E1E1E1"/>

<Setter Property="Width" Value="300"/>

<Setter Property="Height" Value="400"/>

<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Hidden"/>

<Setter Property="ScrollViewer.HorizontalScrollBarVisibility" Value="Hidden"/>

<Setter Property="BorderThickness" Value="0, 0, 1, 0"/>

<Setter Property="BorderBrush" Value="Gray"/>

<Setter Property="Padding" Value="0, 0, 10, 0"/>

<Setter Property="SelectionMode" Value="Single"/>

</Style>

<Style x:Key="filePathTextBoxStyle" TargetType="TextBox">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Background" Value="Transparent"/>

<Setter Property="Padding" Value="5, 5, 5, 5"/>

<Setter Property="Height" Value="40"/>

<Setter Property="Width" Value="300"/>

<Setter Property="IsReadOnly" Value="True"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="1"

BorderBrush="LightGray"

CornerRadius="10"

Background="White"

Padding="{TemplateBinding Padding}">

<ScrollViewer x:Name="PART\_ContentHost"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="templatesProperteisBtnStyle" TargetType="Button">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Right"/>

<Setter Property="Width" Value="110"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border x:Name="border"

BorderThickness="1"

CornerRadius="10"

Background="#F0F0F0"

BorderBrush="LightGray"

Padding="{TemplateBinding Padding}">

<ContentPresenter x:Name="contentPresenter"

VerticalAlignment="Center"

HorizontalAlignment="Center"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetProperty="BorderBrush.Color"

Storyboard.TargetName="border"

To="{StaticResource mainThemeColor}"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Normal">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="BorderBrush.Color"

To="#F0F0F0"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="templateTypesTextBlockStyle" TargetType="TextBlock">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="20"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontWeight" Value="Light"/>

<EventSetter Event="MouseEnter" Handler="TemplateTypesTextBlockMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="TemplateTypesTextBlockMouseLeave"/>

</Style>

<ContextMenu x:Key="treeViewItemItemContextMenu">

<MenuItem Header="Добавить новый пункт"

Click="AddNewItem">

<MenuItem.Icon>

<Image Source="../Resources/addNewItemIconStatic.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переименовать пункт"

Click="OnTreeViewRenameItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuRename.png"></Image>

</MenuItem.Icon>

</MenuItem>

<Separator/>

<MenuItem Header="Удалить"

Click="OnTreeViewDeleteItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<ContextMenu x:Key="treeViewContentItemContextMenu">

<MenuItem Header="Добавить новый параграф"

Click="AddNewParagraph">

<MenuItem.Icon>

<Image Source="../Resources/addNewItemIconStatic.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переименовать пункт"

Click="OnTreeViewRenameItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuRename.png"></Image>

</MenuItem.Icon>

</MenuItem>

<Separator/>

<MenuItem Header="Удалить"

Click="OnTreeViewDeleteItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<ContextMenu x:Key="treeViewParagraphItemContextMenu">

<MenuItem Header="Переименовать пункт"

Click="OnTreeViewRenameItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuRename.png"></Image>

</MenuItem.Icon>

</MenuItem>

<Separator/>

<MenuItem Header="Удалить"

Click="OnTreeViewDeleteItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<ContextMenu x:Key="treeViewContextMenu">

<MenuItem Header="Добавить новый пункт"

Click="AddNewItemToRoot">

<MenuItem.Icon>

<Image Source="../Resources/addNewItemIconStatic.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<Style x:Key="treeViewItemToggleBtnStyle" TargetType="ToggleButton">

<Setter Property="Width" Value="15"/>

<Setter Property="Height" Value="15"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Background" Value="Transparent"/>

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="ClickMode" Value="Press"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ToggleButton">

<Grid>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CheckStates">

<VisualState x:Name="Checked">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetProperty="Background.Color"

Storyboard.TargetName="Border">

<EasingColorKeyFrame KeyTime="0" Value="Transparent" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unchecked" />

<VisualState x:Name="Indeterminate" />

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<Border x:Name="Border"

Background="Transparent">

<ContentPresenter/>

</Border>

</Grid>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="treeViewInnerTextBoxStyle" TargetType="TextBox">

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="0">

<ScrollViewer x:Name="PART\_ContentHost"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"

VerticalScrollBarVisibility="Disabled"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="treeViewItemsItemStyle" TargetType="TreeViewItem">

<Setter Property="Background" Value="Transparent"/>

<Setter Property="HorizontalContentAlignment"

Value="{Binding Path=HorizontalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="VerticalContentAlignment"

Value="{Binding Path=VerticalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="Padding" Value="1, 0, 0, 0"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="IsExpanded" Value="True"/>

<Setter Property="ContextMenu" Value="{StaticResource treeViewItemItemContextMenu}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TreeViewItem">

<StackPanel Orientation="Vertical"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<StackPanel Orientation="Horizontal"

Height="25"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<Grid>

<ToggleButton x:Name="Expander"

Visibility="Visible"

Style="{StaticResource treeViewItemToggleBtnStyle}"

IsChecked="{Binding IsExpanded,

RelativeSource={RelativeSource TemplatedParent}}">

<ToggleButton.Content>

<Image x:Name="expanderImage"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="15"

Height="15"

Source="../Resources/showTheTreeItemItemsIcon.png"/>

</ToggleButton.Content>

</ToggleButton>

<ToggleButton x:Name="Hider"

Visibility="Collapsed"

Style="{StaticResource treeViewItemToggleBtnStyle}"

IsChecked="{Binding IsExpanded,

RelativeSource={RelativeSource TemplatedParent}}">

<ToggleButton.Content>

<Image x:Name="hiderImage"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="15"

Height="15"

Source="../Resources/hideTheTreeItemItemsIcon.png"/>

</ToggleButton.Content>

</ToggleButton>

</Grid>

<Border x:Name="border"

VerticalAlignment="Center"

Margin="5, 0, 0, 0"

Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}"

Padding="{TemplateBinding Padding}">

<TextBox Text="{Binding HeaderText, UpdateSourceTrigger=PropertyChanged}"

Style="{StaticResource treeViewInnerTextBoxStyle}"

IsEnabled="{Binding IsEnabledForEdditing}"

ContextMenu="{StaticResource treeViewItemItemContextMenu}"

LostFocus="TreeViewItemTextBoxLostFocus"/>

</Border>

</StackPanel>

<ItemsPresenter x:Name="ItemsHost"

Margin="15, 0, 0, 0"

Visibility="Collapsed"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="SelectionStates">

<VisualState x:Name="Selected">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unselected" />

<VisualState x:Name="SelectedInactive">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

</VisualStateGroup>

<VisualStateGroup x:Name="ExpansionStates">

<VisualState x:Name="Expanded">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"

Storyboard.TargetName="ItemsHost">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}" />

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Expander"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Hider"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Collapsed">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Expander"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Hider"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="PreviewMouseDown" Handler="TreeViewItemMouseDown"/>

</Style>

<Style x:Key="treeViewContentItemStyle" TargetType="TreeViewItem">

<Setter Property="Background" Value="Transparent"/>

<Setter Property="HorizontalContentAlignment"

Value="{Binding Path=HorizontalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="VerticalContentAlignment"

Value="{Binding Path=VerticalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="Padding" Value="1, 0, 0, 0"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="IsExpanded" Value="True"/>

<Setter Property="ContextMenu" Value="{StaticResource treeViewContentItemContextMenu}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TreeViewItem">

<StackPanel Orientation="Vertical"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<StackPanel Orientation="Horizontal"

Height="25"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<ToggleButton x:Name="Contenter"

Visibility="Visible"

Style="{StaticResource treeViewItemToggleBtnStyle}"

IsChecked="{Binding IsExpanded,

RelativeSource={RelativeSource TemplatedParent}}">

<ToggleButton.Content>

<Image x:Name="expanderImage"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="15"

Height="15"

Source="{Binding ImageSource, UpdateSourceTrigger=PropertyChanged}"/>

</ToggleButton.Content>

</ToggleButton>

<Border x:Name="border"

VerticalAlignment="Center"

Margin="5, 0, 0, 0"

Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}"

Padding="{TemplateBinding Padding}">

<TextBox x:Name="contentTextBox"

Text="{Binding HeaderText, UpdateSourceTrigger=PropertyChanged}"

Style="{StaticResource treeViewInnerTextBoxStyle}"

IsEnabled="{Binding IsEnabledForEdditing}"

ContextMenu="{StaticResource treeViewContentItemContextMenu}"

LostFocus="TreeViewItemTextBoxLostFocus"/>

</Border>

</StackPanel>

<ItemsPresenter x:Name="ItemsHost"

Margin="15, 0, 0, 0"

Visibility="Collapsed"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="SelectionStates">

<VisualState x:Name="Selected">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unselected" />

<VisualState x:Name="SelectedInactive">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

</VisualStateGroup>

<VisualStateGroup x:Name="ExpansionStates">

<VisualState x:Name="Expanded">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"

Storyboard.TargetName="ItemsHost">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}" />

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Collapsed"/>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="PreviewMouseDown" Handler="TreeViewItemMouseDown"/>

</Style>

<Style x:Key="templateTreeViewStyle" TargetType="TreeView">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Width" Value="300"/>

<Setter Property="Height" Value="350"/>

<Setter Property="Background" Value="{StaticResource mainPageItemsGridBackgroundBrush}"/>

<Setter Property="BorderBrush" Value="Transparent"/>

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="ContextMenu" Value="{StaticResource treeViewContextMenu}"/>

<Setter Property="Padding" Value="10, 10, 0, 0"/>

</Style>

<Style x:Key="hintTextBoxStyle" TargetType="TextBox">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="35"/>

<Setter Property="Padding" Value="5, 2, 5, 2"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="1"

BorderBrush="LightGray"

CornerRadius="10"

VerticalAlignment="Top"

Width="{TemplateBinding Width}"

Height="{TemplateBinding Height}"

HorizontalAlignment="Left"

Background="{TemplateBinding Background}"

Padding="{TemplateBinding Padding}">

<ScrollViewer x:Name="PART\_ContentHost"

VerticalAlignment="Center"

HorizontalAlignment="Left"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Page.Resources>

<Page.Content>

<ScrollViewer VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="{Binding ElementName=documentTemplatesPage, Path=ActualWidth}"

Height="{Binding ElementName=documentTemplatesPage, Path=ActualHeight}">

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical"

Margin="0, 0, 0, 20">

<local:PageHeader x:Name="pageHeader"

Header="Шаблоны"

IsRefreshEnabled="False"

VerticalAlignment="Top"

Width="{Binding ElementName=documentTemplatesPage, Path=ActualWidth}"

Height="86"

HorizontalAlignment="Left"/>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 25, 0, 0"

Orientation="Horizontal">

<TextBlock x:Name="goToLocalDocumentation"

Text="Локальные шаблоны"

Style="{StaticResource templateTypesTextBlockStyle}"

PreviewMouseDown="GoToLocalTemplatesMode"/>

<TextBlock x:Name="goToCloudDocumentation"

Text="Облачные шаблоны"

Margin="20, 0, 0, 0"

Style="{StaticResource templateTypesTextBlockStyle}"

PreviewMouseDown="GoToCloudTemplatesMode"/>

</StackPanel>

<StackPanel x:Name="optionBtnsStackPanel"

Orientation="Horizontal"

Margin="20, 10, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Left">

<Button x:Name="createNewTemplateBtn"

Style="{StaticResource templatesProperteisBtnStyle}"

Margin="0, 0, 0, 0"

Click="CreateNewTemplate">

<TextBlock FontFamily="Arial, Verdana"

FontSize="14"

Foreground="{StaticResource mainThemeBrush}"

Text="Создать"

VerticalAlignment="Top"

HorizontalAlignment="Left"/>

</Button>

<Button x:Name="deleteSelectedTemplateBtn"

Style="{StaticResource templatesProperteisBtnStyle}"

Margin="10, 0, 0, 0"

Click="DeleteTemplate">

<TextBlock FontFamily="Arial, Verdana"

FontSize="14"

Foreground="{StaticResource mainThemeBrush}"

Text="Удалить"

VerticalAlignment="Top"

HorizontalAlignment="Left"/>

</Button>

<Button x:Name="saveAllTemplatesBtn"

Style="{StaticResource templatesProperteisBtnStyle}"

Margin="10, 0, 0, 0"

Width="150"

Click="SaveAllTemplatesBtn">

<TextBlock FontFamily="Arial, Verdana"

FontSize="14"

Foreground="{StaticResource mainThemeBrush}"

Text="Сохранить всё"

VerticalAlignment="Top"

HorizontalAlignment="Left"/>

</Button>

<Button x:Name="saveCurrentTemplateBtn"

Style="{StaticResource templatesProperteisBtnStyle}"

Margin="10, 0, 0, 0"

Width="200"

Click="SaveCurrentTemplate">

<TextBlock FontFamily="Arial, Verdana"

FontSize="14"

Foreground="{StaticResource mainThemeBrush}"

Text="Сохранить текущий шаблон"

VerticalAlignment="Top"

HorizontalAlignment="Left"/>

</Button>

</StackPanel>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 20, 0, 0"

Orientation="Horizontal">

<Grid Margin="0, 00, 0, 0">

<StackPanel x:Name="localTemplatesStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Horizontal"

Margin="0, 0, 0, 0">

<ListBox x:Name="localTemplatesListBox"

Margin="0, 0, 0, 0"

ItemTemplate="{StaticResource localTemplateListBoxDataTemplate}"

Style="{StaticResource templatesListBoxStyle}"/>

</StackPanel>

<StackPanel x:Name="cloudTemplatesStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Visibility="Collapsed"

Orientation="Horizontal"

Margin="0, 0, 0, 0">

<ListBox x:Name="cloudTemplatesListBox"

Margin="0, 0, 0, 0"

Style="{StaticResource templatesListBoxStyle}"

ItemTemplate="{StaticResource cloudTemplateListBoxDataTemplate}"/>

</StackPanel>

</Grid>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Vertical">

<TextBlock Text="Имя шаблона"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="10, 0, 0, 0"

FontFamily="Arial, Verdana"

Foreground="Gray"

FontSize="13"/>

<TextBox x:Name="templateNameTextBox"

Style="{StaticResource hintTextBoxStyle}"

Margin="10, 0, 0, 10"

TextChanged="TemplateTextBoxTextChanged"/>

<TextBlock Text="Подсказка"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="10, 0, 0, 0"

FontFamily="Arial, Verdana"

Foreground="Gray"

FontSize="13"/>

<TextBox x:Name="hintTextBox"

Style="{StaticResource hintTextBoxStyle}"

Margin="10, 0, 0, 10"

TextChanged="HintTextBoxTextChanged"/>

<TreeView x:Name="templateTreeView"

Margin="10, 0, 0, 0"

Style="{StaticResource templateTreeViewStyle}"/>

</StackPanel>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Page.Content>

</Page>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using ApplicationLib.Services;

using ApplicationLib.Database;

using ApplicationLib.Factories;

using SDWP.Interfaces;

using SDWP.Exceptions;

using SDWP.Factories;

using SDWP.Models;

using System.IO;

using System.Data.SqlClient;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для DocumentTemplatesPage.xaml

/// </summary>

public partial class DocumentTemplatesPage : Page, IAccountPage

{

#region IAccountPage

public Action CloseAccGrid { get; set; }

#endregion

#region Services and factories

private IServiceAbstractFactory ServiceAbstractFactory { get; set; }

private ISdwpAbstractFactory SdwpAbstractFactory { get; set; }

private ILocalTemplateService LocalTemplateService { get; set; }

private ICloudTemplateService CloudTemplateService { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

#endregion

#region Propeties

#warning delete from the table CurrentUser property

private string DefaultStoragePath { get; }

//status: either local templates or cloud templates

private bool LocalTemplatesMode { get; set; }

//Framework elements

private TextBlock SelectedLocalTemplateTextBlock { get; set; }

private StackPanel LocalTemplatesStackPanel { get; set; }

private StackPanel CloudTemplatesStackPanel { get; set; }

private ListBox LocalTemplatesListBox { get; set; }

private ListBox CloudTemplatesListBox { get; set; }

private TreeView TemplateTreeView { get; set; }

private PageHeader PageHeader { get; set; }

private TextBox TemplateNameTextBox { get; set; }

private TextBox HintTextBox { get; set; }

private TextBlock LocalTemplatesTextBlock { get; set; }

private TextBlock CloudTemplatesTextBlock { get; set; }

#endregion

#region Constructors and initializing methods

public DocumentTemplatesPage(UserInfo currentUser)

{

InitializeComponent();

InitializeProperties();

InitializeServices();

DefaultStoragePath = Path.Combine(Directory.GetCurrentDirectory(), "Templates");

GoToLocalTemplatesMode(null, null);

}

private void InitializeServices()

{

ServiceAbstractFactory = new ServiceAbstractFactory();

SdwpAbstractFactory = new SdwpAbstractFactory();

ExceptionHandler = SdwpAbstractFactory.GetExceptionHandler(Dispatcher);

LocalTemplateService = ServiceAbstractFactory.GetLocalTemplateService();

LocalTemplateService.StoragePath = DefaultStoragePath;

CloudTemplateService = ServiceAbstractFactory.GetCloudTemplateService();

}

private void InitializeProperties()

{

LocalTemplatesStackPanel = localTemplatesStackPanel;

CloudTemplatesStackPanel = cloudTemplatesStackPanel;

LocalTemplatesListBox = localTemplatesListBox;

CloudTemplatesListBox = cloudTemplatesListBox;

TemplateTreeView = templateTreeView;

PageHeader = pageHeader;

TemplateNameTextBox = templateNameTextBox;

HintTextBox = hintTextBox;

LocalTemplatesTextBlock = goToLocalDocumentation;

CloudTemplatesTextBlock = goToCloudDocumentation;

}

#endregion

#region Templates uploading

private async Task UploadTemplatesFromLocalStorage()

{

PageHeader.SwitchOnTopLoader();

try

{

List<LocalTemplate> localTemplates = (await LocalTemplateService.GetLocalTemplates()).ToList();

LocalTemplatesListBox.ItemsSource = localTemplates;

PageHeader.SwitchOffTheLoader();

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async Task UploadTemplatesFromColoudStorage()

{

try

{

PageHeader.SwitchOnTopLoader();

List<Template> templates = (await CloudTemplateService.GetUserTemplates(UserInfo.CurrentUser.ID)).ToList();

cloudTemplatesListBox.ItemsSource = templates;

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

#endregion

#region Parent setting

private void SetTemplateItemsParents(Template template)

{

if (template.Items != null)

{

foreach (Item item in template.Items)

{

SetItemParents(item);

}

}

}

private void SetItemParents(Item item)

{

if (item.Items != null)

{

foreach (Item i in item.Items)

{

(i as IParentableItem).SetParents(item, item.Items);

SetItemParents(i);

}

}

if (item.Paragraphs != null)

{

foreach (Paragraph paragraph in item.Paragraphs)

{

(paragraph as IParentableParagraph).SetParents(item, item.Paragraphs);

paragraph.ParagraphElement.ParentParagraph = paragraph;

}

}

}

#endregion

#region Templates list boxes events

private void ListBoxItemMouseEnter(object sender, MouseEventArgs e)

{

(sender as ListBoxItem).Background = new SolidColorBrush(Color.FromRgb(240, 240, 240));

}

private void ListBoxItemMouseLeave(object sender, MouseEventArgs e)

{

ListBoxItem listBoxItem = sender as ListBoxItem;

if (!listBoxItem.IsSelected)

{

listBoxItem.Background = new SolidColorBrush(Colors.LightGray);

}

}

private void TemplateTypesTextBlockMouseEnter(object sender, MouseEventArgs e)

{

(sender as TextBlock).TextDecorations.Add(TextDecorations.Underline);

}

private void TemplateTypesTextBlockMouseLeave(object sender, MouseEventArgs e)

{

(sender as TextBlock).TextDecorations.Clear();

}

private void ListBoxItemMouseDown(object sender, MouseButtonEventArgs e)

{

ListBoxItem selectedItem = sender as ListBoxItem;

selectedItem.IsSelected = true;

Template selectedTemplate = GetSelectedTemplate();

TemplateNameTextBox.Text = selectedTemplate.TemplateName;

CreateTemplateTreeView(selectedTemplate);

}

#endregion

#region Template tree view

/// <summary>

/// Buids a tree view with a given template file, also sets parents to all tree nodes (items or paragraphs)

/// </summary>

/// <param name="localTemplate"></param>

private void CreateTemplateTreeView(Template template)

{

TemplateTreeView.Items.Clear();

SetTemplateItemsParents(template);

foreach (Item item in template.Items)

{

TemplateTreeViewItem treeViewItem = CreateTemplateTreeViewItem(item);

UploadItemsToTreeView(item, treeViewItem);

if (item.Paragraphs != null)

{

foreach (Paragraph paragraph in item.Paragraphs)

{

TemplateTreeViewItem treeItem = CreateTemplateTreeViewItem(paragraph);

treeViewItem.Items.Add(treeItem);

}

}

TemplateTreeView.Items.Add(treeViewItem);

}

}

/// <summary>

/// Recursive algorithm which upload items and paragraphs to the tree view root item

/// </summary>

/// <param name="item"></param>

/// <param name="rootItem"></param>

private void UploadItemsToTreeView(Item item, TemplateTreeViewItem rootItem)

{

if (item.Items != null)

foreach (Item i in item.Items)

{

TemplateTreeViewItem treeViewItem = CreateTemplateTreeViewItem(i);

rootItem.Items.Add(treeViewItem);

if (i.Items != null)

{

UploadItemsToTreeView(i, treeViewItem);

}

if (i.Paragraphs != null)

{

foreach (Paragraph paragraph in i.Paragraphs)

{

TemplateTreeViewItem treeItem = CreateTemplateTreeViewItem(paragraph);

treeViewItem.Items.Add(treeItem);

}

}

}

}

private TemplateTreeViewItem CreateTemplateTreeViewItem(Paragraph paragraph)

{

TemplateTreeViewItem templateTreeViewItem = new TemplateTreeViewParagraphItem(paragraph)

{

Style = Resources["treeViewContentItemStyle"] as Style,

ContextMenu = Resources["treeViewParagraphItemContextMenu"] as ContextMenu,

Margin = new Thickness(10, 0, 0, 0),

};

return templateTreeViewItem;

}

private TemplateTreeViewItem CreateTemplateTreeViewItem(Item item)

{

TemplateTreeViewItem templateTreeViewItem = new TemplateTreeViewItemItem(item);

if (item.Paragraphs != null)

{

templateTreeViewItem.Style = Resources["treeViewItemsItemStyle"] as Style;

templateTreeViewItem.ContextMenu = Resources["treeViewContentItemContextMenu"] as ContextMenu;

}

else

templateTreeViewItem.Style = Resources["treeViewItemsItemStyle"] as Style;

templateTreeView.Margin = new Thickness(10, 0, 0, 0);

return templateTreeViewItem;

}

private void TreeViewItemMouseDown(object sender, RoutedEventArgs e)

{

TreeViewItem selectedItem = sender as TreeViewItem;

selectedItem.IsSelected = true;

if (selectedItem is TemplateTreeViewParagraphItem)

{

HintTextBox.Text = (selectedItem as TemplateTreeViewParagraphItem).Paragraph.ParagraphElement.Hint;

}

else

{

HintTextBox.Text = string.Empty;

}

}

/// <summary>

/// Enables the edditing of a name of a selected tree view item

/// </summary>

private void OnTreeViewRenameItem(object sender, RoutedEventArgs e)

{

(TemplateTreeView.SelectedItem as TemplateTreeViewItem).IsEnabledForEdditing = true;

}

private void AddNewItem(object sender, RoutedEventArgs e)

{

if (TemplateTreeView.SelectedItem is TemplateTreeViewItem templateTreeViewItem)

{

Item selectedItem = (templateTreeViewItem as TemplateTreeViewItemItem).Item;

CreateNewItemWindow createNewItemWindow = new CreateNewItemWindow(selectedItem.Items, selectedItem);

if (createNewItemWindow.ShowDialog() == true)

{

RefreshTemplateTreeView();

}

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете шаблон",

MessageBoxButton.OK);

}

}

private void RefreshTemplateTreeView()

{

if (LocalTemplatesMode)

CreateTemplateTreeView((LocalTemplatesListBox.SelectedItem as LocalTemplate).Template);

else

CreateTemplateTreeView(CloudTemplatesListBox.SelectedItem as Template);

}

private void AddNewParagraph(object sender, RoutedEventArgs e)

{

Item selectedItem = (TemplateTreeView.SelectedItem as TemplateTreeViewItemItem).Item;

CreateTemplateTreeViewParagraphWindow createWindow = new CreateTemplateTreeViewParagraphWindow(selectedItem);

if (createWindow.ShowDialog() == true)

{

RefreshTemplateTreeView();

}

}

private void TreeViewItemTextBoxLostFocus(object sender, RoutedEventArgs e)

{

(TemplateTreeView.SelectedItem as TemplateTreeViewItem).IsEnabledForEdditing = false;

}

private void OnTreeViewDeleteItem(object sender, RoutedEventArgs e)

{

TemplateTreeViewItem selectedItem = TemplateTreeView.SelectedItem as TemplateTreeViewItem;

if (selectedItem is TemplateTreeViewItemItem)

{

Item item = (selectedItem as TemplateTreeViewItemItem).Item;

item.ParentList.Remove(item);

}

else

{

Paragraph paragraph = (selectedItem as TemplateTreeViewParagraphItem).Paragraph;

paragraph.ParentList.Remove(paragraph);

}

RefreshTemplateTreeView();

}

private void AddNewItemToRoot(object sender, RoutedEventArgs e)

{

Template selectedTemplate = GetSelectedTemplate();

CreateNewItemWindow createNewItemWindow = new CreateNewItemWindow(selectedTemplate.Items, null);

if (createNewItemWindow.ShowDialog() == true)

{

RefreshTemplateTreeView();

}

}

/// <summary>

/// Gets a selected template from a LocalTempaltesListBox if the current mode is local mode

/// or from a CloudTemplatesListBox if the current mode is cloud mode

/// </summary>

private Template GetSelectedTemplate()

{

if (LocalTemplatesMode)

return (LocalTemplatesListBox.SelectedItem as LocalTemplate).Template;

else

return CloudTemplatesListBox.SelectedItem as Template;

}

#endregion

#region Template buttons (4 top buttons) methods

/// <summary>

/// Saves all templates to the local templates folder

/// </summary>

private async void SaveAllTemplatesBtn(object sender, RoutedEventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

if (LocalTemplatesMode)

{

if (LocalTemplatesListBox.ItemsSource is IEnumerable<LocalTemplate> templates)

{

foreach (LocalTemplate template in templates)

await LocalTemplateService.RewriteTemplateFile(template);

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала загрузите шаблоны",

MessageBoxButton.OK);

return;

}

}

else

{

IEnumerable<Template> templates = CloudTemplatesListBox.ItemsSource as IEnumerable<Template>;

if (CloudTemplatesListBox.ItemsSource is IEnumerable<Template>)

foreach (Template template in templates)

await CloudTemplateService.UpdateTemplate(template);

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала загрузите шаблоны",

MessageBoxButton.OK);

return;

}

}

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех", "Все шаблоны выбранного раздела сохранены",

MessageBoxButton.OK);

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Saves current template to a file in a local templates folder

/// </summary>

private async void SaveCurrentTemplate(object sender, RoutedEventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

if (LocalTemplatesMode)

{

if (LocalTemplatesListBox.SelectedItem is LocalTemplate localTemplate)

await LocalTemplateService.RewriteTemplateFile(localTemplate);

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Выберете шаблон для сохранения",

MessageBoxButton.OK);

return;

}

}

else

{

if (CloudTemplatesListBox.SelectedItem is Template template)

await CloudTemplateService.UpdateTemplate(template);

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Выберете шаблон для сохранения",

MessageBoxButton.OK);

return;

}

}

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех", "Шаблон успешно сохранен", MessageBoxButton.OK);

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

finally

{

PageHeader.SwitchOffTheLoader();

}

}

/// <summary>

/// Deletes the template from the list box, deletes the template file from

/// the template directory and also clears the tree view

/// </summary>

private void DeleteTemplate(object sender, RoutedEventArgs e)

{

if (SDWPMessageBox.ConfirmAction() == MessageBoxResult.Cancel)

return;

try

{

if (LocalTemplatesMode)

{

if (LocalTemplatesListBox.SelectedItem is LocalTemplate selectedTemplate)

{

LocalTemplateService.DeleteTemplateFile(selectedTemplate);

List<LocalTemplate> listBoxItemsSource = LocalTemplatesListBox.ItemsSource as List<LocalTemplate>;

listBoxItemsSource.Remove(selectedTemplate);

LocalTemplatesListBox.ItemsSource = null;

LocalTemplatesListBox.ItemsSource = listBoxItemsSource;

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете шаблон для удаления",

MessageBoxButton.OK);

return;

}

}

else

{

if (CloudTemplatesListBox.SelectedItem is Template selectedTemplate)

{

CloudTemplateService.DeleteTemplate(selectedTemplate);

List<Template> templates = CloudTemplatesListBox.ItemsSource as List<Template>;

templates.Remove(selectedTemplate);

CloudTemplatesListBox.ItemsSource = null;

CloudTemplatesListBox.ItemsSource = templates;

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете шаблон для удаления",

MessageBoxButton.OK);

return;

}

}

TemplateTreeView.Items.Clear();

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех", "Шаблон успешно удален", MessageBoxButton.OK);

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Creates a new template in a list box and also creates a file of this template in a template folder

/// </summary>

private async void CreateNewTemplate(object sender, RoutedEventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

Template template = CreateNewTemplate();

if (LocalTemplatesMode)

{

LocalTemplate localTemplate = new LocalTemplate(template);

await LocalTemplateService.CreateTemplateFile(localTemplate);

List<LocalTemplate> templates = (LocalTemplatesListBox.ItemsSource as

IEnumerable<LocalTemplate>).ToList();

templates.Add(localTemplate);

LocalTemplatesListBox.ItemsSource = null;

LocalTemplatesListBox.ItemsSource = templates;

}

else

{

await CloudTemplateService.InsertTemplate(template);

List<Template> templates = CloudTemplatesListBox.ItemsSource as List<Template>;

templates.Add(template);

CloudTemplatesListBox.ItemsSource = null;

CloudTemplatesListBox.ItemsSource = templates;

}

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех", "Шаблон успешно создан", MessageBoxButton.OK);

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

#warning insert this into tables

private Template CreateNewTemplate()

{

return new Template()

{

Items = new List<Item>(),

TemplateName = "Новый шаблон",

CreatedAt = DateTime.Now,

UpdatedAt = DateTime.Now,

UserID = UserInfo.CurrentUser.ID

};

}

#endregion

#region Mode switching methods

private async void GoToLocalTemplatesMode(object sender, MouseButtonEventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

LocalTemplatesStackPanel.Visibility = Visibility.Visible;

CloudTemplatesStackPanel.Visibility = Visibility.Collapsed;

LocalTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.OrangeRed);

CloudTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.Black);

if (LocalTemplateService.StoragePath != null)

await UploadTemplatesFromLocalStorage();

LocalTemplatesMode = true;

PageHeader.SwitchOffTheLoader();

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async void GoToCloudTemplatesMode(object sender, MouseButtonEventArgs e)

{

PageHeader.SwitchOnTopLoader();

try

{

LocalTemplatesStackPanel.Visibility = Visibility.Collapsed;

CloudTemplatesStackPanel.Visibility = Visibility.Visible;

LocalTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.Black);

CloudTemplatesTextBlock.Foreground = new SolidColorBrush(Colors.OrangeRed);

await UploadTemplatesFromColoudStorage();

LocalTemplatesMode = false;

PageHeader.SwitchOffTheLoader();

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

#endregion

/// <summary>

/// If paragraph element is selected the hint is displayed in the Hint text box

/// If the hint text box's text is changed then the hint of the selected paragraph is changed

/// to the new value

/// </summary>

private void HintTextBoxTextChanged(object sender, TextChangedEventArgs e)

{

TreeViewItem selectedTreeViewItem = TemplateTreeView.SelectedItem as TreeViewItem;

if (selectedTreeViewItem is TemplateTreeViewParagraphItem)

{

Paragraph paragraph = (selectedTreeViewItem as TemplateTreeViewParagraphItem).Paragraph;

paragraph.ParagraphElement.Hint = (sender as TextBox).Text;

}

}

/// <summary>

/// When the template is selected it's name is displayed in the Template text box.

/// If the text in the template text box is changed the name of the template is changed

/// </summary>

private void TemplateTextBoxTextChanged(object sender, TextChangedEventArgs e)

{

if (LocalTemplatesMode)

{

if (LocalTemplatesListBox.SelectedItem is LocalTemplate selectedLocalTemplate)

{

selectedLocalTemplate.Template.TemplateName = TemplateNameTextBox.Text;

}

}

else

{

if (CloudTemplatesListBox.SelectedItem is Template selectedTemplate)

{

selectedTemplate.TemplateName = TemplateNameTextBox.Text;

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.IO;

using System.Data.SqlClient;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using ApplicationLib.Services;

using ApplicationLib.Database;

using ApplicationLib.Word;

using SDWP.Interfaces;

using SDWP.Exceptions;

using SDWP.Factories;

using Microsoft.Win32;

using ApplicationLib.Factories;

using ApplicationLib.Word.Interfaces;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для ExportDocumentationPage.xaml

/// </summary>

public partial class ExportDocumentationPage : Page, IAccountPage

{

#region IAccountPage properties

public Action CloseAccGrid { get; set; }

#endregion

#region Services and factories

private IServiceAbstractFactory ServiceAbstractFactory { get; set; }

private ISdwpAbstractFactory SdwpAbstractFactory { get; set; }

private IRenderersAbstractFactory RenderersAbstractFactory { get; set; }

private ICloudDocumentationService CloudDocumentationService { get; set; }

private ICloudDocumentsService CloudDocumentsService { get; set; }

private ILocalDocumentationService LocalDocumentationService { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

private IWordRenderer WordDocumentRenderer { get; set; }

#endregion

#region Properties

private MainPage MainPage { get; }

private RenderSettings RenderSettings { get; set; }

private TextBox WordFilePathTextBox { get; set; }

private PageHeader PageHeader { get; set; }

#endregion

public ExportDocumentationPage(MainPage mainPage)

{

InitializeComponent();

InitializeServices();

SetUpInitialWordRenderSettings();

SetDataContext();

InitializeProperties();

MainPage = mainPage;

}

private void SetDataContext()

{

DataContext = RenderSettings;

}

private void SetUpInitialWordRenderSettings()

{

RenderSettings = new RenderSettings()

{

AddFooter = true,

AddHeader = true,

AddLeftTable = true,

AddSecondPage = true,

AddTitlePage = true,

DefaultColor = "#000000",

DefaultTextSize = "12",

FontFamily = "Times New Roman"

};

}

private void InitializeServices()

{

ServiceAbstractFactory = new ServiceAbstractFactory();

SdwpAbstractFactory = new SdwpAbstractFactory();

RenderersAbstractFactory = new RenderersAbstractFactory();

WordDocumentRenderer = RenderersAbstractFactory.GetWordDocumentRender();

CloudDocumentationService = ServiceAbstractFactory.GetCloudDocumentationService();

CloudDocumentsService = ServiceAbstractFactory.GetCloudDocumentsService();

LocalDocumentationService = ServiceAbstractFactory.GetLocalDocumentationService();

ExceptionHandler = SdwpAbstractFactory.GetExceptionHandler(Dispatcher);

}

private void InitializeProperties()

{

WordFilePathTextBox = wordExportFilePathTextBox;

PageHeader = pageHeader;

}

#region Saving processes

/// <summary>

/// This method is called when the save btn is pressed. We get a local doc from main page, then check if the path is determined in the

/// local doc and if everything is OK we save the documentation

/// </summary>

private async void SaveDocumentation(object sender, RoutedEventArgs e)

{

LocalDocumentation localDocumentation = GetMainPageLocalDocumentation();

if (localDocumentation == null)

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Нет документации для сохранения",

MessageBoxButton.OK);

return;

}

if (localDocumentation.DocumentationPath == null)

{

localDocumentation.DocumentationPath = SelectFilePathForSaving();

}

await SaveDocumentation(localDocumentation);

}

/// <summary>

/// Gets the main page documentation and documents which are related to this documentation and forms the

/// local documentation object, even if the cloud documentation is uploaded.

/// </summary>

private LocalDocumentation GetMainPageLocalDocumentation()

{

if (MainPage.LocalDocumentation == null)

return null;

Documentation documentation = MainPage.DocController.Documentation;

List<Document> documents = MainPage.DocController.Documents;

LocalDocumentation localDocumentation = new LocalDocumentation(documentation, documents)

{

DocumentationPath = MainPage.LocalDocumentation.DocumentationPath

};

return localDocumentation;

}

/// <summary>

/// The method which actualy saves documentation

///</summary>

private async Task SaveDocumentation(LocalDocumentation localDocumentation)

{

try

{

await LocalDocumentationService.CreateLocalDocumentationFile(localDocumentation);

SDWPMessageBox.ShowSDWPMessageBox("Статус сохранения", "Документация успешно сохранена",

MessageBoxButton.OK);

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// This methods is called when save documentation as btn is pressed.

/// </summary>

private async void SaveDocumentationAs(object sender, RoutedEventArgs e)

{

string savingFilePath = SelectFilePathForSaving();

if (savingFilePath == null)

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Ошибка выбора файла для сохранения",

MessageBoxButton.OK);

return;

}

LocalDocumentation localDocumentation = GetMainPageLocalDocumentation();

if (localDocumentation == null)

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Нет документации для сохранения",

MessageBoxButton.OK);

return;

}

localDocumentation.DocumentationPath = savingFilePath;

await SaveDocumentation(localDocumentation);

}

/// <summary>

/// Select a file where to save the documentation

/// </summary>

private string SelectFilePathForSaving()

{

SaveFileDialog saveFileDialog = new SaveFileDialog()

{

Filter = "(\*sdwp)|\*sdwp",

Title = "Выберете файл для сохранения",

CreatePrompt = true

};

if (saveFileDialog.ShowDialog() == true)

{

string path = saveFileDialog.FileName;

if (path.Substring(path.Length - 5) != ".sdwp")

path += ".sdwp";

return path;

}

return null;

}

#endregion

private async void SaveCloudDocumentation(object sender, RoutedEventArgs e)

{

try

{

if (MainPage.DocController.Documentation != null &&

MainPage.DocController.Documentation.StorageType == StorageType.Cloud)

{

Documentation documentation = MainPage.DocController.Documentation;

List<Document> documents = MainPage.DocController.Documents;

await CloudDocumentationService.UpdateDocumentation(documentation);

await SynchronizeDocuments(documentation, documents);

SDWPMessageBox.ShowSDWPMessageBox("Успешно", "Докуемнтация успешно сохранена",

MessageBoxButton.OK);

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Нет облачной документации для сохранения",

MessageBoxButton.OK);

}

}

catch (SqlException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Synchronizes the new document list after it was changed locally with the database

/// </summary>

private async Task SynchronizeDocuments(Documentation documentation, List<Document> documents)

{

List<Document> databaseDocuments = (await CloudDocumentsService.

GetDocumentationDocuments(documentation.ID)).ToList();

//update the documents which were updated and add new ones

foreach (Document document in documents)

{

if (databaseDocuments.FindIndex(d => d.ID == document.ID) > -1)

{

await CloudDocumentsService.UpdateDocument(document);

}

else

{

await CloudDocumentsService.InsertDocument(document);

}

}

//delete the documents from the database which were deleted locally

foreach (Document databaseDocument in databaseDocuments)

{

if (documents.FindIndex(d => d.ID == databaseDocument.ID) == -1)

{

await CloudDocumentsService.DeleteDocument(databaseDocument);

}

}

}

#region Word export

private void SelectWordFileForSaving(object sender, RoutedEventArgs e)

{

string filePath = FolderDialog.ShowDialog();

if (filePath != null)

{

WordFilePathTextBox.Text = filePath;

}

}

private async void ExportDocumentationToWord(object sender, RoutedEventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

Documentation documentation = MainPage.DocController.Documentation;

List<Document> documents = MainPage.DocController.Documents;

CheckIfDocumentationReadyToExport(documentation, documents);

foreach (Document document in documents)

{

WordDocumentRenderer.SetRenderParams(RenderSettings, document,

documentation);

await WordDocumentRenderer.Render();

}

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех",

"Документация успешно экспортирована в Word", MessageBoxButton.OK);

}

catch (ArgumentException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Checks if the render parameters are set correctly and if the

/// documentation and documents are ready to be rendered

/// </summary>

/// <exception cref="ArgumentException">

/// The argument exception is thrown when some of the render params are wrong

/// </exception>

private void CheckIfDocumentationReadyToExport(Documentation documentation,

List<Document> documents)

{

if (documents == null || documentation == null)

throw new ArgumentException("Нет документации или документов для экспорта");

if (!int.TryParse(RenderSettings.DefaultTextSize, out int size) || size <= 0)

throw new ArgumentException("Неправильно введен размер шрифта");

if (string.IsNullOrEmpty(RenderSettings.FolderPath) ||

string.IsNullOrWhiteSpace(RenderSettings.FolderPath) ||

!Directory.Exists(RenderSettings.FolderPath))

{

throw new ArgumentException("Неправильно введен путь для сохранения");

}

}

#endregion

}

}

<Page x:Class="SDWP.MainPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:SDWP"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch">

<Page.Resources>

<Style x:Key="propertyNameStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="11"/>

<Setter Property="Foreground" Value="{StaticResource propertyNameForeground}"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="TextAlignment" Value="Left"/>

<Setter Property="FontWeight" Value="UltraLight"/>

<Setter Property="Padding" Value="0, 0, 0, 0"/>

</Style>

<Style x:Key="documentationNameGridStyle" TargetType="TextBox">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Stretch"/>

<Setter Property="Width" Value="220"/>

<Setter Property="Height" Value="30"/>

<Setter Property="TextAlignment" Value="Left"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="FontWeight" Value="Bold"/>

<Setter Property="Padding" Value="0, 0, 0, 0"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 0"

BorderBrush="{StaticResource devisionRectangleFill}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="{TemplateBinding Width}"

Height="{TemplateBinding Height}"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="propertyValueStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="{StaticResource defaultTextColorBrush}"/>

<Setter Property="FontWeight" Value="Light"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Stretch"/>

<Setter Property="Padding" Value="0, 0, 0, 0"/>

<Setter Property="TextAlignment" Value="Left"/>

</Style>

<Style x:Key="horizontalDevisionRectangleStyle" TargetType="Rectangle">

<Setter Property="Fill" Value="{StaticResource devisionRectangleFill}"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Stretch"/>

<Setter Property="Height" Value="1"/>

</Style>

<Style x:Key="listScrollViewerStyle" TargetType="ScrollViewer">

<Setter Property="VerticalAlignment" Value="Stretch"/>

<Setter Property="HorizontalAlignment" Value="Stretch"/>

<Setter Property="Background" Value="Transparent"/>

<Setter Property="VerticalScrollBarVisibility" Value="Hidden"/>

</Style>

<Style x:Key="paragraphsSearchTextBoxStyle" TargetType="TextBox">

<Setter Property="Height" Value="30"/>

<Setter Property="Width" Value="190"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="{StaticResource mainFontFamily}"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Padding" Value="5, 4, 0, 0"/>

<Setter Property="CaretBrush" Value="{StaticResource paragraphsTopOptionBrush}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="1"

Background="Transparent"

BorderBrush="{StaticResource paragraphsTopOptionBrush}"

CornerRadius="5">

<ScrollViewer Name="PART\_ContentHost"

Margin="{TemplateBinding Padding}"

Width="{TemplateBinding Width}"

Height="{TemplateBinding Height}"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="Background.Color"

To="#F8F8F8"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Normal">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="Background.Color"

To="Transparent"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="paragraphsLineGridStyle" TargetType="Grid">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Stretch"/>

<Setter Property="Height" Value="1"/>

<Setter Property="Background" Value="{StaticResource paragraphsTopOptionBrush}"/>

</Style>

<Style x:Key="paragraphsTopOptionImageStyle" TargetType="Image">

<Setter Property="Width" Value="35"/>

<Setter Property="Height" Value="35"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Right"/>

</Style>

<Style x:Key="hintTextBlockHeaderStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="Width" Value="150"/>

<Setter Property="Height" Value="20"/>

</Style>

<Style x:Key="treeViewOptionTextBoxStyle" TargetType="TextBox">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="10"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="TextAlignment" Value="Left"/>

<Setter Property="FontWeight" Value="UltraLight"/>

<Setter Property="Padding" Value="0, 0, 0, 0"/>

<Setter Property="Cursor" Value="Arrow"/>

<Setter Property="IsReadOnly" Value="True"/>

<Setter Property="SelectionBrush" Value="Transparent"/>

<Setter Property="Foreground">

<Setter.Value>

<SolidColorBrush x:Name="foregroundBrush"

Color="{StaticResource paragraphsTopOptionColor}">

</SolidColorBrush>

</Setter.Value>

</Setter>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="0, 0, 0, 0"

BorderBrush="{StaticResource paragraphsTopOptionBrush}">

<ScrollViewer Name="PART\_ContentHost"

Margin="{TemplateBinding Padding}"

Width="{TemplateBinding Width}"

Height="{TemplateBinding Height}"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="TreeViewOptionTextBoxMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="TreeViewOptionTextBoxMouseLeave"/>

</Style>

<Style x:Key="treeViewItemToggleBtnStyle" TargetType="ToggleButton">

<Setter Property="Width" Value="15"/>

<Setter Property="Height" Value="15"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Background" Value="Transparent"/>

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="ClickMode" Value="Press"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ToggleButton">

<Grid>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CheckStates">

<VisualState x:Name="Checked">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetProperty="Background.Color"

Storyboard.TargetName="Border">

<EasingColorKeyFrame KeyTime="0" Value="Transparent" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unchecked" />

<VisualState x:Name="Indeterminate" />

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<Border x:Name="Border"

Background="Transparent">

<ContentPresenter/>

</Border>

</Grid>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="treeViewInnerTextBoxStyle" TargetType="TextBox">

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="0">

<ScrollViewer x:Name="PART\_ContentHost"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"

VerticalScrollBarVisibility="Disabled"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<ContextMenu x:Key="treeViewItemItemContextMenu">

<MenuItem Header="Добавить новый пункт"

Click="OnTreeContextMenuAddNewItem">

<MenuItem.Icon>

<Image Source="../Resources/addNewItemIconStatic.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переименовать пункт"

Click="OnTreeViewRenameItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuRename.png"></Image>

</MenuItem.Icon>

</MenuItem>

<Separator/>

<MenuItem Header="Удалить"

Click="OnTreeViewDeleteItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переместить вверх"

Click="MoveItemUp">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переместить вниз"

Click="MoveItemDown">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<ContextMenu x:Key="treeViewContentItemContextMenu">

<MenuItem Header="Переименовать пункт"

Click="OnTreeViewRenameItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuRename.png"></Image>

</MenuItem.Icon>

</MenuItem>

<Separator/>

<MenuItem Header="Удалить"

Click="OnTreeViewDeleteItem">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переместить вверх"

Click="MoveItemUp">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="Переместить вниз"

Click="MoveItemDown">

<MenuItem.Icon>

<Image Source="../Resources/contextMenuDeleteItem.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<ContextMenu x:Key="treeViewContextMenu">

<MenuItem Header="Добавить новый пункт"

Click="OnTreeContextMenuAddNewItemToRoot">

<MenuItem.Icon>

<Image Source="../Resources/addNewItemIconStatic.png"></Image>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

<Style x:Key="documentTreeViewStyle" TargetType="TreeView">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Background" Value="{StaticResource mainPageItemsGridBackgroundBrush}"/>

<Setter Property="BorderBrush" Value="Transparent"/>

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="Padding" Value="5, 10, 0, 0"/>

<Setter Property="ContextMenu" Value="{StaticResource treeViewContextMenu}"/>

</Style>

<Style x:Key="treeViewItemsItemStyle" TargetType="TreeViewItem">

<Setter Property="Background" Value="Transparent"/>

<Setter Property="HorizontalContentAlignment"

Value="{Binding Path=HorizontalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="VerticalContentAlignment"

Value="{Binding Path=VerticalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="Padding" Value="1, 0, 0, 0"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="IsExpanded" Value="True"/>

<Setter Property="ContextMenu" Value="{StaticResource treeViewItemItemContextMenu}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TreeViewItem">

<StackPanel Orientation="Vertical"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<StackPanel Orientation="Horizontal"

Height="25"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<Grid>

<ToggleButton x:Name="Expander"

Visibility="Visible"

Style="{StaticResource treeViewItemToggleBtnStyle}"

IsChecked="{Binding IsExpanded,

RelativeSource={RelativeSource TemplatedParent}}">

<ToggleButton.Content>

<Image x:Name="expanderImage"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="15"

Height="15"

Source="../Resources/showTheTreeItemItemsIcon.png"/>

</ToggleButton.Content>

</ToggleButton>

<ToggleButton x:Name="Hider"

Visibility="Collapsed"

Style="{StaticResource treeViewItemToggleBtnStyle}"

IsChecked="{Binding IsExpanded,

RelativeSource={RelativeSource TemplatedParent}}">

<ToggleButton.Content>

<Image x:Name="hiderImage"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="15"

Height="15"

Source="../Resources/hideTheTreeItemItemsIcon.png"/>

</ToggleButton.Content>

</ToggleButton>

</Grid>

<Border x:Name="border"

VerticalAlignment="Center"

Margin="5, 0, 0, 0"

Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}"

Padding="{TemplateBinding Padding}">

<TextBox Text="{Binding HeaderText, UpdateSourceTrigger=PropertyChanged}"

Style="{StaticResource treeViewInnerTextBoxStyle}"

IsEnabled="{Binding IsEnabledForEdditing}"

ContextMenu="{StaticResource treeViewItemItemContextMenu}"

LostFocus="TreeViewItemTextBoxLostFocus"/>

</Border>

</StackPanel>

<ItemsPresenter x:Name="ItemsHost"

Margin="15, 0, 0, 0"

Visibility="Collapsed"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="SelectionStates">

<VisualState x:Name="Selected">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unselected" />

<VisualState x:Name="SelectedInactive">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

</VisualStateGroup>

<VisualStateGroup x:Name="ExpansionStates">

<VisualState x:Name="Expanded">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"

Storyboard.TargetName="ItemsHost">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}" />

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Expander"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Hider"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Collapsed">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Expander"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="Hider"

Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="PreviewMouseDown" Handler="TreeViewItemMouseDown"/>

</Style>

<Style x:Key="treeViewContentItemStyle" TargetType="TreeViewItem">

<Setter Property="Background" Value="Transparent"/>

<Setter Property="HorizontalContentAlignment"

Value="{Binding Path=HorizontalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="VerticalContentAlignment"

Value="{Binding Path=VerticalContentAlignment,

RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}"/>

<Setter Property="Padding" Value="1, 0, 0, 0"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="IsExpanded" Value="True"/>

<Setter Property="ContextMenu" Value="{StaticResource treeViewContentItemContextMenu}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TreeViewItem">

<StackPanel Orientation="Vertical"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<StackPanel Orientation="Horizontal"

Height="25"

HorizontalAlignment="Stretch"

VerticalAlignment="Top">

<ToggleButton x:Name="Contenter"

Visibility="Visible"

Style="{StaticResource treeViewItemToggleBtnStyle}"

IsChecked="{Binding IsExpanded,

RelativeSource={RelativeSource TemplatedParent}}">

<ToggleButton.Content>

<Image x:Name="expanderImage"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="15"

Height="15"

Source="../Resources/treeViewContentItemIcon.png"/>

</ToggleButton.Content>

</ToggleButton>

<Border x:Name="border"

VerticalAlignment="Center"

Margin="5, 0, 0, 0"

Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}"

Padding="{TemplateBinding Padding}">

<TextBox x:Name="contentTextBox"

Text="{Binding HeaderText, UpdateSourceTrigger=PropertyChanged}"

Style="{StaticResource treeViewInnerTextBoxStyle}"

IsEnabled="{Binding IsEnabledForEdditing}"

ContextMenu="{StaticResource treeViewContentItemContextMenu}"

LostFocus="TreeViewItemTextBoxLostFocus"/>

</Border>

</StackPanel>

<ItemsPresenter x:Name="ItemsHost"

Margin="15, 0, 0, 0"

Visibility="Collapsed"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="SelectionStates">

<VisualState x:Name="Selected">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unselected" />

<VisualState x:Name="SelectedInactive">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"

Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">

<EasingColorKeyFrame KeyTime="0" Value="#E5E5E5" />

</ColorAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

</VisualStateGroup>

<VisualStateGroup x:Name="ExpansionStates">

<VisualState x:Name="Expanded">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"

Storyboard.TargetName="ItemsHost">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}" />

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Collapsed"/>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="PreviewMouseDown" Handler="TreeViewItemMouseDown"/>

</Style>

<Style x:Key="leftDocGridImgStyle" TargetType="Image">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Width" Value="30"/>

<Setter Property="Height" Value="30"/>

</Style>

</Page.Resources>

<Grid VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Background="Wheat">

<Grid VerticalAlignment="Stretch"

HorizontalAlignment="Stretch">

<!--Left documentation grid-->

<Grid x:Name="leftDocumentsGrid"

VerticalAlignment="Stretch"

HorizontalAlignment="Left"

Width="250"

Background="{StaticResource documentsLeftGridBackgroundBrush}">

<Grid x:Name="showLeftGridImagesGrid"

Visibility="Collapsed">

<Image x:Name="showLeftGridImageStatic"

Visibility="Visible"

Style="{StaticResource leftDocGridImgStyle}"

Margin="15, 15, 0, 0"

Source="../Resources/forwardArrowIconStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="showLeftGridImageActive"

Visibility="Collapsed"

Margin="15, 15, 0, 0"

Style="{StaticResource leftDocGridImgStyle}"

Source="../Resources/forwardArrowIconActive.png"

MouseLeave="IconMouseLeave"

MouseDown="ShowLeftDocumentsGrid"/>

</Grid>

<Grid x:Name="hideLeftGridImagesGrid"

Visibility="Visible">

<Image x:Name="hideLeftGridImageStatic"

Style="{StaticResource leftDocGridImgStyle}"

Visibility="Visible"

Margin="15, 15, 0, 0"

Source="../Resources/closeAccGridStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="hideLeftGridImageActive"

Visibility="Collapsed"

Style="{StaticResource leftDocGridImgStyle}"

Margin="15, 15, 0, 0"

Source="../Resources/closeAccGridActive.png"

MouseLeave="IconMouseLeave"

MouseDown="HideLeftDocumentsGrid"/>

</Grid>

<Grid x:Name="addNewDocumentGrid"

Margin="210, 15, 0, 0"

Visibility="Visible">

<Image x:Name="createNewDocStatic"

Style="{StaticResource leftDocGridImgStyle}"

Source="../Resources/addNewItemIconStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="createNewDocActive"

Style="{StaticResource leftDocGridImgStyle}"

Source="../Resources/addNewItemIconActive.png"

MouseLeave="IconMouseLeave"

MouseDown="CreateNewDocument"/>

</Grid>

<Grid x:Name="leftDocumentsGridContent"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Margin="0, 40, 0, 0">

<StackPanel x:Name="documentationPropertiesStackPannel"

Margin="0, 30, 0, 0">

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 0, 0, 0"

Text="Назавние проекта:"/>

<TextBox x:Name="documentationNameTextBox"

Margin="10, 0, 15, 0"

IsEnabled="False"

Style="{StaticResource documentationNameGridStyle}"

Text="{Binding ProjectName, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 0, 0, 0"

Text="Team Lead:"/>

<TextBox x:Name="documentationCodeTextBox"

Margin="10, 0, 15, 0"

IsEnabled="False"

Style="{StaticResource documentationNameGridStyle}"

Text="{Binding TeamLeadName, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 0, 0, 0"

Text="Имя менеджера:"/>

<TextBox x:Name="documentationTeamLeadTextBox"

Margin="10, 0, 15, 0"

IsEnabled="False"

Style="{StaticResource documentationNameGridStyle}"

Text="{Binding ManagerName, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 0, 0, 0"

Text="Код проекта:"/>

<TextBox x:Name="documentationProjectCodeTextBox"

Margin="10, 0, 15, 0"

IsEnabled="False"

Style="{StaticResource documentationNameGridStyle}"

Text="{Binding ProjectCode, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 0, 0, 0"

Text="Имя разработчика:"/>

<TextBox x:Name="documentationSoftwareEngineerNameTextBox"

Margin="10, 0, 15, 0"

IsEnabled="False"

Style="{StaticResource documentationNameGridStyle}"

Text="{Binding SoftwareEngineerName, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Text="Дата создания: "

Margin="15, 0, 0, 0"/>

<TextBlock x:Name="documentationCreationDateTextBlock"

Style="{StaticResource propertyValueStyle}"

IsEnabled="False"

Text="{Binding CreationDate}"

Margin="15, 0, 15, 0"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 10, 0, 0"

FontSize="15"

Text="Документы:"/>

<ScrollViewer VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="0, 0, 0, 0"

VerticalScrollBarVisibility="Hidden">

<StackPanel x:Name="documentsListStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"/>

</ScrollViewer>

</StackPanel>

</Grid>

</Grid>

<Grid x:Name="itemsGrid"

VerticalAlignment="Stretch"

HorizontalAlignment="Left"

Width="250"

Margin="250, 0, 0, 0"

Background="{StaticResource mainPageItemsGridBackgroundBrush}">

<Grid x:Name="topItemsGridNavBarGrid"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Height="40"

Margin="15, 10, 15, 0"

Background="Transparent">

<Grid x:Name="backToPreviousItemGrid">

<Image x:Name="backToPreviousItemStaticImage"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Visibility="Visible"

Width="30"

Height="30"

Margin="0, 0, 0, 0"

IsEnabled="False"

Source="../Resources/backToPreviousItemIconStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="backToPreviousItemActiveImage"

VerticalAlignment="Center"

Visibility="Collapsed"

HorizontalAlignment="Left"

Width="30"

Height="30"

Margin="0, 0, 0, 0"

Source="../Resources/backToPreviousItemIconActive.png"

MouseLeave="IconMouseLeave"

MouseDown="GoToPreviousItem"/>

<TextBlock x:Name="backToPreviousItemTextBlock"

FontFamily="Arial, Verdana"

Foreground="Black"

Visibility="Visible"

Width="150"

TextTrimming="CharacterEllipsis"

HorizontalAlignment="Left"

VerticalAlignment="Center"

Margin="35, 0, 0, 0"/>

</Grid>

<Image x:Name="addNewItemStaticImage"

VerticalAlignment="Center"

HorizontalAlignment="Right"

Visibility="Visible"

Width="30"

Height="30"

Margin="0, 0, 0, 0"

Source="../Resources/addNewItemIconStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="addNewItemActiveImage"

VerticalAlignment="Center"

Visibility="Collapsed"

HorizontalAlignment="Right"

Width="30"

Height="30"

Margin="0, 0, 0, 0"

Source="../Resources/addNewItemIconActive.png"

MouseLeave="IconMouseLeave"

MouseDown="CreateNewItem"/>

</Grid>

<Grid Style="{StaticResource paragraphsLineGridStyle}"

Margin="10, 55, 10, 0"/>

<TextBox x:Name="goToTreeViewModeTextBox"

Style="{StaticResource treeViewOptionTextBoxStyle}"

Margin="12.5, 70, 0, 0"

FontSize="11"

Text="Перейти к дереву документа:"

PreviewMouseDown="GoToTreeViewMode"/>

<TextBox x:Name="goToListModeTextBox"

Style="{StaticResource treeViewOptionTextBoxStyle}"

Margin="12.5, 70, 0, 0"

Visibility="Collapsed"

FontSize="11"

Text="Перейти к листу документа:"

PreviewMouseDown="GoToListMode"/>

<TextBlock Style="{StaticResource propertyNameStyle}"

Margin="15, 90, 0, 0"

FontSize="15"

Text="Пункты:"/>

<ScrollViewer Style="{StaticResource listScrollViewerStyle}"

Margin="0, 110, 0, 60"

x:Name="listModelScroll"

Visibility="Visible">

<StackPanel x:Name="itemsListStackPanel"

Width="0"

VerticalAlignment="Stretch"

HorizontalAlignment="Left"

Orientation="Vertical"/>

</ScrollViewer>

<TreeView x:Name="documentTreeView"

Margin="0, 110, 0, 60"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Visibility="Collapsed"

Style="{StaticResource documentTreeViewStyle}"/>

</Grid>

<Grid x:Name="paragraphsGrid"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Margin="500, 0, 0, 0"

Background="{StaticResource mainPageParagraphsGridBackgroundBrush}">

<Grid x:Name="paragraphsGridTopOptionsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Margin="25, 10, 25, 0"

Height="40"

Background="Transparent">

<TextBox x:Name="paragraphsSearchTextBox"

Text="Введите запрос..."

GotFocus="ParagraphSearchTextBoxGotFocus"

LostFocus="ParagraphSearchTextBoxLostFocus"

PreviewKeyDown="ParagraphSearchTextBoxKeyDown"

Margin="0, 0, 0, 0"

Style="{StaticResource paragraphsSearchTextBoxStyle}"/>

<Grid x:Name="addNewParagraphElementGrid"

VerticalAlignment="Center"

HorizontalAlignment="Right"

Height="35"

Width="35">

<Image x:Name="addNewParagraphElementImageStatic"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Visible"

Source="../Resources/addNewParagraphElementStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="addNewParagraphElementImageActive"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Collapsed"

Source="../Resources/addNewParagraphElementActive.png"

MouseLeave="IconMouseLeave"

MouseDown="ShowParagraphElementsAddOptions"/>

</Grid>

<Grid x:Name="paragraphElementsGrid"

VerticalAlignment="Center"

HorizontalAlignment="Right"

Visibility="Collapsed"

Width="0"

Height="35"

MouseLeave="ParagraphElementsGridMouseLeave"

Background="Transparent">

<Image x:Name="addNewTableImageStatic"

Width="30"

Height="30"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Visible"

Source="../Resources/addNewTableIconStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="addNewTableImageActive"

Width="30"

Height="30"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Collapsed"

Source="../Resources/addNewTableActive.png"

MouseLeave="IconMouseLeave"

PreviewMouseDown="AddNewTable"/>

<Image x:Name="addNewNumberedListImageStatic"

Margin="0, 0, 35, 0"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Visible"

Source="../Resources/addNewNumberedListStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="addNewNumberedListImageActive"

Margin="0, 0, 35, 0"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Collapsed"

Source="../Resources/addNewNumberedListActive.png"

MouseLeave="IconMouseLeave"

PreviewMouseDown="AddNewNumberedList"/>

<Image x:Name="addNewImageImageStatic"

Margin="0, 0, 75, 0"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Visible"

Source="../Resources/addNewImageStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="addNewImageImageActive"

Margin="0, 0, 75, 0"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Collapsed"

Source="../Resources/addNewImageActive.png"

MouseLeave="IconMouseLeave"

PreviewMouseDown="AddNewParagraphImage"/>

<Image x:Name="addNewSubparagraphImageStatic"

Margin="0, 0, 115, 0"

Width="25"

Height="25"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Visible"

Source="../Resources/addNewSubparagraphStatic.png"

MouseEnter="IconMouseEnter"/>

<Image x:Name="addNewSubparagraphImageActive"

Margin="0, 0, 115, 0"

Width="25"

Height="25"

Style="{StaticResource paragraphsTopOptionImageStyle}"

Visibility="Collapsed"

Source="../Resources/addNewSubparagraphActive.png"

MouseLeave="IconMouseLeave"

PreviewMouseLeftButtonDown="AddNewSubparagraph"/>

</Grid>

</Grid>

<Grid Style="{StaticResource paragraphsLineGridStyle}"

Margin="25, 55, 25, 0"/>

<ScrollViewer Style="{StaticResource listScrollViewerStyle}"

Margin="40, 60, 40, 0"

VerticalAlignment="Top"

HorizontalAlignment="Stretch"

VerticalScrollBarVisibility="Visible">

<Grid>

<StackPanel x:Name="paragraphsListPanel"

Orientation="Vertical"

CanVerticallyScroll="True"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"/>

<StackPanel x:Name="paragraphsSearchResultsPanel"

Orientation="Vertical"

CanVerticallyScroll="True"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"/>

</Grid>

</ScrollViewer>

</Grid>

</Grid>

</Grid>

</Page>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.IO;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using ApplicationLib.Views;

using SDWP.Factories;

using SDWP.Models;

using SDWP.Interfaces;

using System.Threading;

namespace SDWP

{

public partial class MainPage : Page

{

#region Properties

private Grid LeftDocumentationGrid { get; set; }

private Grid ItemsGrid { get; set; }

private Grid ParagraphsGrid { get; set; }

private Grid HideLeftGridImagesGrid { get; set; }

private Grid ShowLeftGridImagesGrid { get; set; }

private Image BackToPreviousItemStaticImage { get; set; }

private Image AddNewItemStaticImage { get; set; }

private StackPanel DocumentsListStackPanel { get; set; }

private StackPanel ParagraphsListPanel { get; set; }

#warning add this to the tables

private StackPanel ParagraphsSearchResultPanel { get; set; }

private StackPanel ItemsListStackPanel { get; set; }

private Grid ParagraphElementsGrid { get; set; }

private Grid AddNewParagraphElementGrid { get; set; }

private TextBox GoToTreeViewModeTextBox { get; set; }

private TextBox GoToListModeTextBox { get; set; }

private TreeView DocumentTreeView { get; set; }

private ScrollViewer ListModeScroll { get; set; }

private TextBox ParagraphSearchTextBox { get; set; }

#warning add this property to tables

private StackPanel DocumentationPropertiesStackPannel { get; set; }

#warning add this property to tables

private CancellationTokenSource TokenSource { get; set; } = new CancellationTokenSource();

#warning add this property to tables

private CancellationToken Token { get; set; }

#warning add this property to tables

private bool DoWeHaveToCancelParagraphUploading { get; set; }

#warning add this to the tables

private int NumOfParagraphsPerPanel { get; } = 3;

private string ParagraphSearchTextBoxDefaultText { get; } = "Введите запрос...";

private ISdwpAbstractFactory AbstractFactory { get; set; }

public IDocController DocController { get; set; }

public LocalDocumentation LocalDocumentation { get; set; }

#endregion

public MainPage()

{

InitializeComponent();

InitializeProperties();

InitializeServices();

}

#region Initialize page methods

private void InitializeServices()

{

AbstractFactory = new SdwpAbstractFactory();

DocController = AbstractFactory.GetDocController();

}

private void InitializeProperties()

{

LeftDocumentationGrid = leftDocumentsGrid;

ItemsGrid = itemsGrid;

ParagraphsGrid = paragraphsGrid;

HideLeftGridImagesGrid = hideLeftGridImagesGrid;

ShowLeftGridImagesGrid = showLeftGridImagesGrid;

BackToPreviousItemStaticImage = backToPreviousItemStaticImage;

AddNewItemStaticImage = addNewItemStaticImage;

DocumentsListStackPanel = documentsListStackPanel;

ParagraphsListPanel = paragraphsListPanel;

ParagraphsSearchResultPanel = paragraphsSearchResultsPanel;

ItemsListStackPanel = itemsListStackPanel;

ParagraphElementsGrid = paragraphElementsGrid;

AddNewParagraphElementGrid = addNewParagraphElementGrid;

ParagraphSearchTextBox = paragraphsSearchTextBox;

GoToTreeViewModeTextBox = goToTreeViewModeTextBox;

GoToListModeTextBox = goToListModeTextBox;

DocumentTreeView = documentTreeView;

ListModeScroll = listModelScroll;

DocumentationPropertiesStackPannel = documentationPropertiesStackPannel;

Token = TokenSource.Token;

}

#endregion

#region Upload new documentation

public void UploadLocalDocumentation(LocalDocumentation localDocumentation)

{

ClearMainPage();

LocalDocumentation = localDocumentation;

DocController.UploadLocalDocumentation(LocalDocumentation);

UploadDocumentation();

}

public void UploadCloudDocumentation(Documentation documentation, List<Document> documents)

{

ClearMainPage();

LocalDocumentation = null;

DocController.UploadCloudDocumentation(documentation, documents);

UploadDocumentation();

}

/// <summary>

/// Clears all documentations' stack panels and DocController

/// </summary>

private void ClearMainPage()

{

DocController.Clear();

DocumentsListStackPanel.Children.Clear();

ItemsListStackPanel.Children.Clear();

ParagraphsListPanel.Children.Clear();

}

/// <summary>

/// Uploads documentation to the main page and makes all textboxes with

/// documentation properties active

/// </summary>

private void UploadDocumentation()

{

Array.ForEach(DocumentationPropertiesStackPannel.Children.

OfType<TextBox>().Where(t => t.Name != "documentationCreationDateTextBlock")

.ToArray(), tx => tx.IsEnabled = true);

UploadDocumentationDataToUI(DocController.Documentation);

UploadDocumentsDataToUI(DocController.Documents);

}

private void UploadDocumentationDataToUI(Documentation documentation)

{

DataContext = documentation;

}

/// <summary>

/// Uploads all documents which are in the current documentation to the left

/// document grid, and creates a PreviewMouseDown Event for every document

/// visual object

/// </summary>

private void UploadDocumentsDataToUI(List<Document> documents)

{

DocumentsListStackPanel.Children.Clear();

SetDocumentItemParents(documents);

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

{

DocumentMenuOption documentMenuOption = new DocumentMenuOption(documents[i],

documents);

//set events

documentMenuOption.OnDocumentItemClick += UploadDocumentItems;

documentMenuOption.UpdateList += RefreshDocumentUI;

DocumentsListStackPanel.Children.Add(documentMenuOption);

}

}

private void SetDocumentItemParents(List<Document> documents)

{

foreach (Document document in documents)

{

foreach (Item item in document.Items)

{

item.SetParents(null, document.Items);

SetItemParents(item);

}

}

}

private void SetItemParents(Item item)

{

if (item.Items != null)

{

foreach (Item i in item.Items)

{

i.SetParents(item, item.Items);

SetItemParents(i);

}

}

if (item.Paragraphs != null)

{

foreach (Paragraph paragraph in item.Paragraphs)

{

(paragraph as IParentableParagraph).SetParents(item, item.Paragraphs);

paragraph.ParagraphElement.ParentParagraph = paragraph;

}

}

}

#endregion

#region Document Tree View

private void TreeViewItemMouseDown(object sender, RoutedEventArgs e)

{

(sender as DocTreeViewItem).IsSelected = true;

}

private void OnTreeViewRenameItem(object sender, RoutedEventArgs e)

{

DocTreeViewItem docTreeViewItem = DocumentTreeView.SelectedItem as DocTreeViewItem;

docTreeViewItem.IsEnabledForEdditing = true;

}

private void OnTreeContextMenuAddNewItem(object sender, RoutedEventArgs e)

{

DocTreeViewItem docTreeViewItem = DocumentTreeView.SelectedItem as DocTreeViewItem;

Item item = docTreeViewItem.Item;

CreateNewItemWindow createNewItemWindow = new CreateNewItemWindow(item.Items, item);

if (createNewItemWindow.ShowDialog() == true)

RefreshItemsUI();

}

private void OnTreeContextMenuAddNewItemToRoot(object sender, RoutedEventArgs e)

{

try

{

List<Item> currentItemsList = DocController.CurrentDocument.Items;

CreateNewItemWindow createNewItemWindow = new CreateNewItemWindow(

currentItemsList, null);

if (createNewItemWindow.ShowDialog() == true)

RefreshItemsUI();

}

catch (NullReferenceException)

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала загрузите документ",

MessageBoxButton.OK);

}

}

private void TreeViewItemTextBoxLostFocus(object sender, RoutedEventArgs e)

{

(sender as TextBox).IsEnabled = false;

}

private void OnTreeViewDeleteItem(object sender, RoutedEventArgs e)

{

DocTreeViewItem docTreeViewItem = DocumentTreeView.SelectedItem as DocTreeViewItem;

Item selectedItem = docTreeViewItem.Item;

selectedItem.ParentList.Remove(selectedItem);

RefreshItemsUI();

}

/// <summary>

/// Creates the document tree view (the second way to observe the document)

/// </summary>

private void CreateDocumentTreeView(Document document)

{

if (document != null)

{

DocumentTreeView.Items.Clear();

foreach (Item item in document.Items)

{

DocTreeViewItem treeViewItem = CreateNewListItem(item);

UploadItemsToTreeView(item, treeViewItem);

DocumentTreeView.Items.Add(treeViewItem);

}

}

}

private DocTreeViewItem CreateNewListItem(Item item)

{

DocTreeViewItem treeItem = new DocTreeViewItem(item);

if (item.Paragraphs != null)

treeItem.Style = Resources["treeViewContentItemStyle"] as Style;

else

treeItem.Style = Resources["treeViewItemsItemStyle"] as Style;

treeItem.Margin = new Thickness(10, 0, 0, 0);

treeItem.MouseDoubleClick += OnTreeViewItemDoubleClick;

return treeItem;

}

private void OnTreeViewItemDoubleClick(object sender, MouseButtonEventArgs e)

{

DocTreeViewItem treeItem = sender as DocTreeViewItem;

if (treeItem.Item.Paragraphs != null)

{

DocController.UploadParagraphs(treeItem.Item);

CancelParagraphsUploading();

UploadParagraphsToParagraphsListPanel(treeItem.Item.Paragraphs,

ParagraphsListPanel);

}

}

private async void CancelParagraphsUploading()

{

if (DoWeHaveToCancelParagraphUploading)

{

TokenSource.Cancel();

ParagraphsListPanel.Children.Clear();

await Task.Delay(TimeSpan.FromMilliseconds(5));

}

}

#warning add this to the tables

private void MoveItemUp(object sender, RoutedEventArgs e)

{

if (DocumentTreeView.SelectedItem is DocTreeViewItem treeViewItem)

{

Item item = treeViewItem.Item;

if (item.ParentList.FindIndex(i => i.Equals(item)) == 0)

{

item.ParentList.Remove(item);

item.ParentList.Add(item);

}

else

{

int itemIndex = item.ParentList.FindIndex(i => i.Equals(item));

Item temp = item.ParentList[itemIndex - 1];

item.ParentList[itemIndex - 1] = item;

item.ParentList[itemIndex] = temp;

}

CreateDocumentTreeView(DocController.CurrentDocument);

}

}

#warning add this to the tables

private void MoveItemDown(object sender, RoutedEventArgs e)

{

if (DocumentTreeView.SelectedItem is DocTreeViewItem treeViewItem)

{

Item item = treeViewItem.Item;

if (item.ParentList.FindIndex(i => i.Equals(item)) == item.ParentList.Count - 1)

{

for (int i = item.ParentList.Count - 1; i > 0; i--)

{

item.ParentList[i] = item.ParentList[i - 1];

}

item.ParentList[0] = item;

}

else

{

int itemIndex = item.ParentList.FindIndex(i => i.Equals(item));

Item temp = item.ParentList[itemIndex + 1];

item.ParentList[itemIndex + 1] = item;

item.ParentList[itemIndex] = temp;

}

CreateDocumentTreeView(DocController.CurrentDocument);

}

}

#endregion

#region Refresh UI methods

#warning add this to the table

/// <summary>

/// Refreshes the paragraph UI after a pargraph was moved up or down.

/// IN this method we asume that the total number

/// of paragraphs havent changed

/// </summary>

private void RefreshParagraphsUIAfterSwap()

{

List<Paragraph> paragraphs = DocController.CurrentParagraphsList;

List<IParagraphEditView> paragraphEditViews = new List<IParagraphEditView>();

List<FrameworkElement> paragraphViews = new List<FrameworkElement>();

foreach (FrameworkElement fe in ParagraphsListPanel.Children)

{

StackPanel stackPanel = fe as StackPanel;

foreach (FrameworkElement stackPanelChild in stackPanel.Children)

{

paragraphEditViews.Add(stackPanelChild as IParagraphEditView);

paragraphViews.Add(stackPanelChild);

}

}

if (paragraphEditViews.Count > 1)

{

//if we added the last paragraph to the top

if (paragraphs[1].Equals(paragraphEditViews[0].Paragraph) &&

paragraphs[0].Equals(paragraphEditViews[paragraphEditViews.Count - 1].Paragraph))

{

StackPanel firstPanel = ParagraphsListPanel.Children[0] as StackPanel;

if (firstPanel.Children.Count == NumOfParagraphsPerPanel)

{

ParagraphsListPanel.Children.Insert(0, new StackPanel());

firstPanel = ParagraphsListPanel.Children[0] as StackPanel;

}

//delete the last paragraph

(ParagraphsListPanel.Children[ParagraphsListPanel.Children.Count - 1] as StackPanel)

.Children.Remove(paragraphViews.Last());

(ParagraphsListPanel.Children[ParagraphsListPanel.Children.Count - 1] as StackPanel).UpdateLayout();

//add the last paragraph to the top

firstPanel.Children.Insert(0, paragraphViews.Last());

firstPanel.UpdateLayout();

}

else if (paragraphs.Last().Equals(paragraphEditViews[0].Paragraph) &&

paragraphs[0].Equals(paragraphEditViews[1].Paragraph))

{

StackPanel lastPanel = ParagraphsListPanel.Children[

ParagraphsListPanel.Children.Count - 1] as StackPanel;

if (lastPanel.Children.Count == NumOfParagraphsPerPanel)

{

lastPanel = new StackPanel();

ParagraphsListPanel.Children.Add(lastPanel);

}

(ParagraphsListPanel.Children[0] as StackPanel).Children.Remove(paragraphViews[0]);

(ParagraphsListPanel.Children[0] as StackPanel).UpdateLayout();

lastPanel.Children.Add(paragraphViews[0]);

lastPanel.UpdateLayout();

}

else

{

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

{

if (!paragraphs[i].Equals(paragraphEditViews[i].Paragraph))

{

int firstIndex = (paragraphViews[i].Parent as StackPanel).Children.

OfType<IParagraphEditView>().ToList().FindIndex(p => p.Equals(paragraphViews[i]));

int secondIndex = (paragraphViews[i + 1].Parent as StackPanel).Children.

OfType<IParagraphEditView>().ToList().FindIndex(p => p.Equals(paragraphViews[i + 1]));

StackPanel firstPanel = (paragraphViews[i] as UserControl).Parent as StackPanel;

StackPanel secondPanel = (paragraphViews[i + 1] as UserControl).Parent as StackPanel;

firstPanel.Children.Remove(paragraphViews[i]);

secondPanel.Children.Remove(paragraphViews[i + 1]);

firstPanel.Children.Insert(firstIndex, paragraphViews[i + 1]);

secondPanel.Children.Insert(secondIndex, paragraphViews[i]);

break;

}

}

}

}

}

/// <summary>

/// Updates the current item's list ui (uploads the items of a CurrentList again),

/// if the current item's list is not null. Usualy this methods are used in events

/// </summary>

private void RefreshItemsUI()

{

if (DocumentTreeView.Visibility == Visibility.Collapsed)

{

if (DocController.CurrentItemsList != null)

UploadItemsToPanel(DocController.CurrentItem, DocController.CurrentItemsList);

}

else

{

CreateDocumentTreeView(DocController.CurrentDocument);

}

}

private async void RefreshParagraphsUI()

{

List<Paragraph> paragraphs = DocController.CurrentParagraphsList;

//try to delete the pragraphEditView if the action was to delete. Only one paragraph can be deleted

//so when wwe found the deleted paragraph, we can execute the return statement

if (paragraphs != null)

{

CancelParagraphsUploading();

foreach (FrameworkElement fe in ParagraphsListPanel.Children)

{

StackPanel stackPanel = fe as StackPanel;

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

{

IParagraphEditView paragraphEditView = stackPanel.Children[i] as

IParagraphEditView;

SetParagraphEditViewEvents(paragraphEditView);

if (paragraphs.FindIndex(p => p.Equals(paragraphEditView.Paragraph)) == -1)

{

stackPanel.Children.Remove(stackPanel.Children[i]);

stackPanel.UpdateLayout();

return;

}

}

}

}

//try to add an pragraphEditView if the action was to add. If we reached that part of code

//that means that the new paragraph was added, but the adding of the paragraph always adds the paragraph

//to the end of the list, so we must just add a new paragraph view to the end of our main stack panel

if (ParagraphsListPanel.Children.Count == 0)

ParagraphsListPanel.Children.Add(new StackPanel());

List<IParagraphEditView> paragraphEditViews = new List<IParagraphEditView>();

foreach (FrameworkElement fe in ParagraphsListPanel.Children)

{

StackPanel stackPanel = fe as StackPanel;

foreach (FrameworkElement frameworkElement in stackPanel.Children)

{

paragraphEditViews.Add(frameworkElement as IParagraphEditView);

}

}

for (int i = paragraphEditViews.Count; i < paragraphs.Count; i++)

{

StackPanel stackPanel = ParagraphsListPanel.Children[ParagraphsListPanel.Children.Count - 1]

as StackPanel;

if (stackPanel.Children.Count == NumOfParagraphsPerPanel)

{

stackPanel = new StackPanel();

ParagraphsListPanel.Children.Add(stackPanel);

}

(paragraphs[i] as IParentableParagraph).SetParents(DocController.CurrentContentItem,

DocController.CurrentContentItem.Paragraphs);

await UploadSingleParagraphToPanel(stackPanel, paragraphs[i]);

}

}

#warning add this to the table

private async Task UploadSingleParagraphToPanel(StackPanel stackPanel, Paragraph paragraph)

{

UserControl paragraphView = null;

Dispatcher.Invoke(new Action(() => paragraphView = paragraph.ParagraphElement.GetEditView()));

await Dispatcher.BeginInvoke(new Action(() =>

paragraphView.Margin = new Thickness(0, 10, 0, 0)));

SetParagraphEditViewEvents(paragraphView as IParagraphEditView);

await Dispatcher.BeginInvoke(new Action(() => stackPanel.Children.Add(paragraphView)));

await Dispatcher.BeginInvoke(new Action(() => stackPanel.UpdateLayout()));

await Task.Delay(TimeSpan.FromMilliseconds(1));

}

#warning add this to the table

private void SetParagraphEditViewEvents(IParagraphEditView paragraphEditView)

{

Dispatcher.BeginInvoke(new Action(() => (paragraphEditView).

RefreshParagraphsUI = RefreshParagraphsUI));

Dispatcher.BeginInvoke(new Action(() => (paragraphEditView).

RefreshParagraphsUIAfterSwap = RefreshParagraphsUIAfterSwap));

}

private void RefreshDocumentUI()

{

if (DocController.Documents != null)

UploadDocumentsDataToUI(DocController.Documents);

}

#endregion

#region Option click events

/// <summary>

/// Uploads all visualizations of Items to the ItemsListStackPanel, creates a PreviewMouseDown event

/// (to upload the content of item when clicked) and then

/// begins the animation to show this grid

/// </summary>

private void UploadDocumentItems(object sender, EventArgs e)

{

DocumentMenuOption clickedDocumentMenuOption = sender as DocumentMenuOption;

ChangeBackgroundsOfDocumentBtns(clickedDocumentMenuOption);

Document document = clickedDocumentMenuOption.Document;

DocController.UploadDocument(document);

UploadItemsToPanel(null, document.Items);

CreateDocumentTreeView(document);

}

/// <summary>

/// The recursive algorithm which creates the tree view for a selected doc.

/// Firstly we upload all "item" items and then upload the content of the item

/// </summary>

/// <param name="item"></param>

/// <param name="rootItem"></param>

private void UploadItemsToTreeView(Item item, DocTreeViewItem rootItem)

{

if (item.Items != null)

foreach (Item i in item.Items)

{

DocTreeViewItem treeViewItem = CreateNewListItem(i);

rootItem.Items.Add(treeViewItem);

if (i.Items != null)

{

UploadItemsToTreeView(i, treeViewItem);

}

}

}

/// <summary>

/// Uploads all items to the itmes list panel,

/// for each item the parent list is defined

/// </summary>

/// <param name="items"></param>

private void UploadItemsToPanel(Item parentItem, List<Item> items)

{

ItemsListStackPanel.Visibility = Visibility.Collapsed;

ItemsListStackPanel.Width = 0;

ItemsListStackPanel.Children.Clear();

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

{

UploadSingleItemToPanel(items[i]);

(items[i] as IParentableItem).SetParents(parentItem, items);

}

ItemsListStackPanel.Visibility = Visibility.Visible;

MainPageAnimations.AnimateWidth(ItemsListStackPanel, 250);

}

private void UploadSingleItemToPanel(Item item)

{

ItemMenuOption itemMenuOption = new ItemMenuOption(item);

itemMenuOption.UpdateList += RefreshItemsUI;

//set events

itemMenuOption.OnItemClick += UploadItemData;

ItemsListStackPanel.Children.Add(itemMenuOption);

}

/// <summary>

/// After the click on an document element changes the background of this document item to

/// the color of the ItemsGrid

/// </summary>

private void ChangeBackgroundsOfDocumentBtns(DocumentMenuOption clickedOption)

{

foreach (FrameworkElement el in DocumentsListStackPanel.Children)

{

if (el is DocumentMenuOption)

(el as DocumentMenuOption).DocumentBtn.Background = new SolidColorBrush(Colors.Transparent);

}

Color clickedBtnBackgroundColor = (Color)Application.Current.Resources["mainPageItemsGridBackgroundColor"];

clickedOption.DocumentBtn.Background = new SolidColorBrush(clickedBtnBackgroundColor);

}

/// <summary>

/// After the click on an item element uploads the data of this item to either the paragraphs grid

/// if this Item object contains paragraphs or to the ItemsGrid, if this item contains Items

/// If we upload items then we must set a backToPreviousItemTextBlock text (so we can get to the

/// previous item list)

/// </summary>

private void UploadItemData(object sender, EventArgs e)

{

ItemMenuOption clickedItemMenuOption = sender as ItemMenuOption;

Item item = clickedItemMenuOption.Item;

item.SetParents(DocController.CurrentItem,

DocController.CurrentItemsList);

ChangeBackgroundOfItemBtns(clickedItemMenuOption);

DocController.UploadItem(item);

if (item.Items != null)

{

UploadItemsToPanel(DocController.CurrentItem, DocController.CurrentItem.Items);

backToPreviousItemTextBlock.Text = "к " + item.Name;

if (DocController.CurrentItem.ParentList != null)

backToPreviousItemStaticImage.IsEnabled = true;

}

else

{

CancelParagraphsUploading();

UploadParagraphsToParagraphsListPanel(item.Paragraphs,

ParagraphsListPanel);

}

}

/// <summary>

/// Uploads all paragraphs of a current item to the paragraphs stack panel

/// </summary>

private async void UploadParagraphsToParagraphsListPanel(List<Paragraph> paragraphs,

StackPanel stackPanel)

{

DoWeHaveToCancelParagraphUploading = true;

await Dispatcher.BeginInvoke(new Action(() => stackPanel.Children.Clear()));

StackPanel currentPanel = new StackPanel();

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

{

if (i % NumOfParagraphsPerPanel == 0)

{

currentPanel = new StackPanel();

stackPanel.Children.Add(currentPanel);

}

if (Token.IsCancellationRequested)

{

DoWeHaveToCancelParagraphUploading = false;

TokenSource = new CancellationTokenSource();

Token = TokenSource.Token;

return;

}

(paragraphs[i] as IParentableParagraph).SetParents(DocController.CurrentContentItem,

DocController.CurrentContentItem.Paragraphs);

await UploadSingleParagraphToPanel(currentPanel, paragraphs[i]);

}

DoWeHaveToCancelParagraphUploading = false;

}

/// <summary>

/// After the click on an items element changes the background of this item to

/// the color of the ParagraphsGrid

/// </summary>

private void ChangeBackgroundOfItemBtns(ItemMenuOption clickedItem)

{

foreach (FrameworkElement el in ItemsListStackPanel.Children)

{

if (el is ItemMenuOption)

(el as ItemMenuOption).ItemBtn.Background = new SolidColorBrush(Colors.Transparent);

}

Color clickedBtnBackgroundColor = (Color)Application.Current.Resources["mainPageParagraphsGridBackgroundColor"];

clickedItem.ItemBtn.Background = new SolidColorBrush(clickedBtnBackgroundColor);

}

#endregion

#region Event handlers

private void ParagraphElementsGridMouseLeave(object sender, MouseEventArgs e)

{

MainPageAnimations.AnimateWidth(ParagraphElementsGrid, 0,

new Action(() =>

{

AddNewParagraphElementGrid.Visibility = Visibility.Visible;

ParagraphElementsGrid.Visibility = Visibility.Collapsed;

}));

}

/// <summary>

/// Starts the process of creation a new item in the current item

/// </summary>

private void CreateNewItem(object sender, MouseButtonEventArgs e)

{

if (DocController.CurrentItemsList != null)

{

CreateNewItemWindow createNewItemWindow = new CreateNewItemWindow(

DocController.CurrentItemsList, DocController.CurrentItem);

createNewItemWindow.ShowDialog();

UploadItemsToPanel(DocController.CurrentItem, DocController.CurrentItemsList);

}

else

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала загрузите документ либо пункт",

MessageBoxButton.OK);

}

}

private void GoToListMode(object sender, MouseButtonEventArgs e)

{

RefreshItemsUI();

BackToPreviousItemStaticImage.IsEnabled = true;

AddNewItemStaticImage.IsEnabled = true;

DocumentTreeView.Visibility = Visibility.Collapsed;

ListModeScroll.Visibility = Visibility.Visible;

GoToListModeTextBox.Visibility = Visibility.Collapsed;

GoToTreeViewModeTextBox.Visibility = Visibility.Visible;

}

private void GoToTreeViewMode(object sender, MouseButtonEventArgs e)

{

CreateDocumentTreeView(DocController.CurrentDocument);

BackToPreviousItemStaticImage.IsEnabled = false;

AddNewItemStaticImage.IsEnabled = false;

DocumentTreeView.Visibility = Visibility.Visible;

ListModeScroll.Visibility = Visibility.Collapsed;

GoToListModeTextBox.Visibility = Visibility.Visible;

GoToTreeViewModeTextBox.Visibility = Visibility.Collapsed;

}

private void IconMouseEnter(object sender, MouseEventArgs e)

{

Image thisImage = sender as Image;

thisImage.Visibility = Visibility.Collapsed;

List<Image> images = (thisImage.Parent as Grid).Children.OfType<Image>().ToList();

int thisImageIndex = images.FindIndex(img => img.Name == thisImage.Name);

images[thisImageIndex + 1].Visibility = Visibility.Visible;

}

private void IconMouseLeave(object sender, MouseEventArgs e)

{

Image thisImage = sender as Image;

thisImage.Visibility = Visibility.Collapsed;

List<Image> images = (thisImage.Parent as Grid).Children.OfType<Image>().ToList();

int thisImageIndex = images.FindIndex(img => img.Name == thisImage.Name);

images[thisImageIndex - 1].Visibility = Visibility.Visible;

}

private void HideLeftDocumentsGrid(object sender, MouseButtonEventArgs e)

{

MainPageAnimations.HideLeftDocumentationGrid(LeftDocumentationGrid);

MainPageAnimations.AnimateMargin(ItemsGrid, new Thickness(60, 0, 0, 0));

MainPageAnimations.AnimateMargin(ParagraphsGrid, new Thickness(310, 0, 0, 0));

HideLeftGridImagesGrid.Visibility = Visibility.Collapsed;

ShowLeftGridImagesGrid.Visibility = Visibility.Visible;

}

private void ShowLeftDocumentsGrid(object sender, MouseButtonEventArgs e)

{

MainPageAnimations.AnimateMargin(ItemsGrid, new Thickness(250, 0, 0, 0));

MainPageAnimations.AnimateMargin(ParagraphsGrid, new Thickness(500, 0, 0, 0));

MainPageAnimations.ShowLeftDocumentationGrid(LeftDocumentationGrid);

HideLeftGridImagesGrid.Visibility = Visibility.Visible;

ShowLeftGridImagesGrid.Visibility = Visibility.Collapsed;

}

private void ShowParagraphElementsAddOptions(object sender, MouseButtonEventArgs e)

{

AddNewParagraphElementGrid.Visibility = Visibility.Collapsed;

ParagraphElementsGrid.Visibility = Visibility.Visible;

MainPageAnimations.AnimateWidth(ParagraphElementsGrid, 200);

}

private void TreeViewOptionTextBoxMouseEnter(object sender, MouseEventArgs e)

{

TextBox textBox = sender as TextBox;

textBox.TextDecorations.Add(TextDecorations.Underline);

}

private void TreeViewOptionTextBoxMouseLeave(object sender, MouseEventArgs e)

{

TextBox textBox = sender as TextBox;

textBox.TextDecorations.Clear();

}

/// <summary>

/// If the text of a search text box is a default text (like a placeholder)

/// we must clear the textbox for a user's purposes

/// </summary>

private void ParagraphSearchTextBoxGotFocus(object sender, RoutedEventArgs e)

{

TextBox textBox = sender as TextBox;

if (textBox.Text == ParagraphSearchTextBoxDefaultText)

textBox.Text = string.Empty;

}

/// <summary>

/// If the textbox's text is an empty string then we must place a placeholder in that

/// textbox

/// </summary>

private void ParagraphSearchTextBoxLostFocus(object sender, RoutedEventArgs e)

{

TextBox textBox = sender as TextBox;

if (textBox.Text == string.Empty)

textBox.Text = ParagraphSearchTextBoxDefaultText;

}

/// <summary>

/// On every text changed we must find all the elements which sutisfiy the search query

/// </summary>

private void ParagraphSearchTextBoxKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Enter)

{

if (ParagraphSearchTextBox == null || DocController.CurrentParagraphsList == null)

return;

string searchText = ParagraphSearchTextBox.Text;

if (string.IsNullOrEmpty(searchText) || searchText == ParagraphSearchTextBoxDefaultText)

{

CancelParagraphsUploading();

ParagraphsSearchResultPanel.Visibility = Visibility.Collapsed;

ParagraphsListPanel.Visibility = Visibility.Visible;

return;

}

List<Paragraph> suitableParagraphs = new List<Paragraph>();

List<Paragraph> currentParagraphs = DocController.CurrentParagraphsList;

foreach (Paragraph paragraph in currentParagraphs)

{

if (paragraph.ParagraphElement.Title.IndexOf(searchText) > -1)

suitableParagraphs.Add(paragraph);

}

CancelParagraphsUploading();

ParagraphsSearchResultPanel.Visibility = Visibility.Visible;

ParagraphsListPanel.Visibility = Visibility.Collapsed;

UploadParagraphsToParagraphsListPanel(suitableParagraphs,

ParagraphsSearchResultPanel);

}

}

/// <summary>

/// If the current item has a parent and parent has item's list,

/// then we must upload get back to the parent item

/// </summary>

private void GoToPreviousItem(object sender, MouseButtonEventArgs e)

{

if (DocController.CanGoToPrevItem())

{

UploadItemsToPanel(DocController.CurrentItem.ParentItem,

DocController.CurrentItem.ParentList);

//update text of a backToPreviousItemTextBlock and if there is no parent disable the back img

if (DocController.CurrentItem.ParentItem != null)

{

backToPreviousItemTextBlock.Text = "к " + DocController.CurrentItem.ParentItem.Name;

}

else

{

backToPreviousItemTextBlock.Text = string.Empty;

backToPreviousItemStaticImage.IsEnabled = false;

}

DocController.GoToPreviousItem(DocController.CurrentItem);

}

}

#endregion

#region Add new paragraphs (IParagraphElement) methods

private void AddNewSubparagraph(object sender, MouseEventArgs e)

{

if (DocController.CurrentContentItem != null)

{

Item contentItem = DocController.CurrentContentItem;

CreateNewSubparagraphWindow createWindow = new CreateNewSubparagraphWindow(contentItem);

createWindow.ShowDialog();

if (createWindow.DialogResult == true)

RefreshParagraphsUI();

}

}

private void AddNewTable(object sender, MouseEventArgs e)

{

if (DocController.CurrentContentItem != null)

{

Item contentItem = DocController.CurrentContentItem;

CreateNewTableWindow createWindow = new CreateNewTableWindow(contentItem);

createWindow.ShowDialog();

if (createWindow.DialogResult == true)

RefreshParagraphsUI();

}

}

private void AddNewNumberedList(object sender, MouseEventArgs e)

{

if (DocController.CurrentContentItem != null)

{

Item contentItem = DocController.CurrentContentItem;

CreateNewNumberedListWindow createWindow = new CreateNewNumberedListWindow(contentItem);

createWindow.ShowDialog();

if (createWindow.DialogResult == true)

RefreshParagraphsUI();

}

}

private void AddNewParagraphImage(object sender, MouseButtonEventArgs e)

{

if (DocController.CurrentContentItem != null)

{

Item contentItem = DocController.CurrentContentItem;

CreateNewImageWindow createNewImageWindow = new CreateNewImageWindow(contentItem);

createNewImageWindow.ShowDialog();

if (createNewImageWindow.DialogResult == true)

RefreshParagraphsUI();

}

}

#endregion

private void CreateNewDocument(object sender, MouseButtonEventArgs e)

{

List<Document> currentDocumentList = DocController.Documents;

Documentation documentation = DocController.Documentation;

if (currentDocumentList == null || documentation == null)

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала откройти документацию", MessageBoxButton.OK);

return;

}

CreateNewDocumentWindow createNewDocumentWindow = new CreateNewDocumentWindow(currentDocumentList,

documentation);

if (createNewDocumentWindow.ShowDialog() == true)

{

RefreshDocumentUI();

}

}

}

}

<UserControl x:Class="SDWP.PageHeader"

x:Name="pageHeader"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:SDWP"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>

<ThicknessAnimationUsingKeyFrames x:Key="topLoaderThirdEllipseAnimation"

AccelerationRatio="0"

RepeatBehavior="Forever"

SpeedRatio="1.1">

<EasingThicknessKeyFrame KeyTime="0:0:0"

Value="50, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.5"

Value="225, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1"

Value="225, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.2"

Value="225, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.7"

Value="50, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:2.2"

Value="50, 0, 0, 0"/>

</ThicknessAnimationUsingKeyFrames>

<ThicknessAnimationUsingKeyFrames x:Key="topLoaderSecondEllipseAnimation"

AccelerationRatio="0"

RepeatBehavior="Forever"

SpeedRatio="1.1">

<EasingThicknessKeyFrame KeyTime="0:0:0.0"

Value="25, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.1"

Value="25, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.6"

Value="200, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1"

Value="200, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.1"

Value="200, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.6"

Value="25, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:2.2"

Value="25, 0, 0, 0"/>

</ThicknessAnimationUsingKeyFrames>

<ThicknessAnimationUsingKeyFrames x:Key="topLoaderFirstEllipseAnimation"

AccelerationRatio="0"

RepeatBehavior="Forever"

SpeedRatio="1.1">

<EasingThicknessKeyFrame KeyTime="0:0:0.0"

Value="0, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.2"

Value="0, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.7"

Value="175, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1"

Value="175, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.5"

Value="0, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:2.2"

Value="0, 0, 0, 0"/>

</ThicknessAnimationUsingKeyFrames>

</UserControl.Resources>

<UserControl.Content>

<Grid>

<Grid x:Name="topHeaderGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="0, 0, 0, 0"

Height="86"

Width="{Binding ElementName=pageHeader, Path=ActualWidth}"

Background="#f3f3f3">

<TextBlock x:Name="userGridMainGridHeader"

FontFamily="Arial, Verdana"

FontSize="40"

Margin="20, 17, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Foreground="OrangeRed"

Text="{Binding Header, UpdateSourceTrigger=PropertyChanged}"/>

<Grid x:Name="topLoaderGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="250"

Height="50"

Visibility="Visible"

Margin="20, 45, 0, 0">

<Ellipse x:Name="loaderEllipse1"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Margin="0, 0, 0, 0"

Height="10"

Width="10"

Fill="OrangeRed"/>

<Ellipse x:Name="loaderEllipse2"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Margin="25, 0, 0, 0"

Height="10"

Width="10"

Fill="OrangeRed"/>

<Ellipse x:Name="loaderEllipse3"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Margin="50, 0, 0, 0"

Height="10"

Width="10"

Fill="OrangeRed"/>

</Grid>

<Image x:Name="refreshIconActive"

Source="../Resources/refreshIconActive.png"

VerticalAlignment="Top"

HorizontalAlignment="Right"

MouseLeave="RefreshIconMouseLeave"

Height="35"

Width="35"

MouseDown="Refresh"

Margin="0, 20, 20, 0"/>

<Image x:Name="refreshIconStatic"

IsEnabled="{Binding IsRefreshEnabled}"

Source="../Resources/refreshIconStatic.png"

VerticalAlignment="Top"

HorizontalAlignment="Right"

MouseEnter="RefreshIconMouseEnter"

MouseDown="Refresh"

Height="35"

Width="35"

Margin="0, 20, 20, 0"/>

<Rectangle x:Name="headerRect"

Height="1"

Fill="Gray"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 85, 20, 0"/>

</Grid>

</Grid>

</UserControl.Content>

</UserControl>

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace SDWP

{

public partial class PageHeader : UserControl

{

#region Properties

public Action OnRefresh { get; set; }

public string Header { get; set; }

public bool IsRefreshEnabled { get; set; }

#endregion

public PageHeader()

{

InitializeComponent();

DataContext = this;

}

#region INotifyPropertyChanged

public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged([CallerMemberName]string prop = "")

{

PropertyChanged?.Invoke(prop, new PropertyChangedEventArgs(prop));

}

#endregion

#region Event handlers

private void RefreshIconMouseEnter(object sender, EventArgs e)

{

refreshIconActive.Visibility = Visibility.Visible;

refreshIconStatic.Visibility = Visibility.Collapsed;

}

private void RefreshIconMouseLeave(object sender, EventArgs e)

{

refreshIconActive.Visibility = Visibility.Collapsed;

refreshIconStatic.Visibility = Visibility.Visible;

}

private void TempaltePageSizeChanged(object sender, SizeChangedEventArgs e)

{

headerRect.Width = this.Width;

}

private void Refresh(object sender, MouseButtonEventArgs e)

{

try

{

SwitchOnTopLoader();

OnRefresh();

}

finally

{

SwitchOffTheLoader();

}

}

#endregion

#region Loader animations

public void SwitchOnTopLoader()

{

topLoaderGrid.Visibility = Visibility.Visible;

List<Ellipse> ellipsesList = topLoaderGrid.Children.Cast<Ellipse>().

ToList();

ellipsesList[2].BeginAnimation(FrameworkElement.MarginProperty,

this.Resources["topLoaderThirdEllipseAnimation"] as

ThicknessAnimationUsingKeyFrames);

ellipsesList[1].BeginAnimation(FrameworkElement.MarginProperty,

this.Resources["topLoaderSecondEllipseAnimation"] as

ThicknessAnimationUsingKeyFrames);

ellipsesList[0].BeginAnimation(FrameworkElement.MarginProperty,

this.Resources["topLoaderFirstEllipseAnimation"] as

ThicknessAnimationUsingKeyFrames);

}

public void SwitchOffTheLoader()

{

List<Ellipse> ellipsesList = topLoaderGrid.Children.Cast<Ellipse>().

ToList();

ellipsesList[2].BeginAnimation(FrameworkElement.MarginProperty, null);

ellipsesList[1].BeginAnimation(FrameworkElement.MarginProperty, null);

ellipsesList[0].BeginAnimation(FrameworkElement.MarginProperty, null);

}

#endregion

}

}

<Window x:Class="SDWP.SDWPMainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

mc:Ignorable="d"

x:Name="sdwpMainWindow"

Title="SDWPMainWindow"

SizeChanged="SDWPMainWindowSizeChanged"

Height="450"

MouseMove="SDWPMainWindowMouseMove"

MinWidth="1000"

MinHeight="500"

Width="800"

Closed="SdwpMainWindowClosed">

<Window.Resources>

<Style x:Key="topOptionsGridTextBlock" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="10"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="TextWrapping" Value="Wrap"/>

</Style>

<Style x:Key="leftMenuOptionGridStyle" TargetType="Grid">

<Setter Property="Width" Value="100"/>

<Setter Property="Height" Value="50"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Background" Value="LightGray"/>

<EventSetter Event="MouseEnter" Handler="LeftMenuOptionGridMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="LeftMenuOptionGridMouseLeave"/>

<EventSetter Event="MouseDown" Handler="LeftMenuOptionGridMouseDown"/>

</Style>

<Style x:Key="leftMenuOptionTextBlockStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="OrangeRed"/>

<Setter Property="FontSize" Value="16"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Margin" Value="10, 0, 0, 0"/>

</Style>

</Window.Resources>

<Grid>

<!--Main grid-->

<Grid x:Name="documentationGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="Red"

Margin="0, 0, 0, 0">

<Frame x:Name="mainPageFrame"

Margin="0, 0, 0, 0"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Background="Green"/>

</Grid>

<!--Top options grid -->

<Grid x:Name="topOptionsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="800"

Height="0"

Background="{StaticResource mainWindowTopOptionsGridBavkgroundBrush}">

<!--User acc grid-->

<Grid x:Name="userAccOptionsGrid"

Height="100"

Width="100"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="OptionGridMouseEnter"

MouseLeave="OptionGridMouseLeave"

MouseDown="ShowTheUserMainGrid">

<Image x:Name="userAccIconStatic"

Width="48"

Height="48"

Margin="0, 10, 0, 0"

Source="../Resources/userAccountStatic.png"

VerticalAlignment="Top"

HorizontalAlignment="Center"/>

<Image x:Name="userAccIconActive"

Visibility="Collapsed"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="48"

Height="48"

Margin="0, 10, 0, 0"

Source="../Resources/userAccountActive.png"/>

<TextBlock Style="{StaticResource topOptionsGridTextBlock}"

Text="Личный кабинет"

Width="60"

TextAlignment="Center"

HorizontalAlignment="Center"

VerticalAlignment="Bottom"

Margin="0, 0, 0, 15"/>

</Grid>

<!--Line grid-->

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="Gray"

Margin="100, 0, 0, 0"

Width="1"

Height="100"/>

<!--Save changes grid-->

<Grid x:Name="saveChangesGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="101, 0, 0, 0"

Height="100"

Width="100"

MouseEnter="OptionGridMouseEnter"

MouseLeave="OptionGridMouseLeave"

MouseDown="SaveDocumentation">

<Image x:Name="saveChangesIconStatic"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="48"

Height="48"

Margin="0, 10, 0, 0"

Source="../Resources/saveIconStatic.png"/>

<Image x:Name="saveChangesIconActive"

Visibility="Collapsed"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="48"

Height="48"

Margin="0, 10, 0, 0"

Source="../Resources/saveIconActive.png"/>

<TextBlock Style="{StaticResource topOptionsGridTextBlock}"

Text="Сохранить изменения"

Width="60"

TextAlignment="Center"

HorizontalAlignment="Center"

VerticalAlignment="Bottom"

Margin="0, 0, 0, 15"/>

</Grid>

<!--Line grid-->

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="201, 0, 0, 0"

Height="100"

Width="1"

Background="Gray"/>

<!--Delete documentation grid-->

<Grid x:Name="deleteDocumentationGrid"

Width="155"

Height="32"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="OptionGridMouseEnter"

MouseLeave="OptionGridMouseLeave"

MouseDown="OpenUserDocsPage"

Margin="202, 0, 0, 0">

<Image x:Name="deleteIconStatic"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="25"

Width="25"

Margin="5, 0, 0, 0"

Source="../Resources/deleteIconStatic.png"/>

<Image x:Name="deleteIconActive"

Visibility="Collapsed"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="25"

Width="25"

Margin="5, 0, 0, 0"

Source="../Resources/deleteIconActive.png"/>

<TextBlock Style="{StaticResource topOptionsGridTextBlock}"

Text="Удалить документацию"

Margin="35, 0, 0, 0"

Width="125"

VerticalAlignment="Center"

HorizontalAlignment="Left"/>

</Grid>

<!--Line grid-->

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="Gray"

Margin="202, 32, 0, 0"

Width="155"

Height="1"/>

<!--Access control grid-->

<Grid x:Name="accessControlGrid"

Width="155"

Height="32"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="OptionGridMouseEnter"

MouseLeave="OptionGridMouseLeave"

Margin="202, 33, 0, 0">

<Image x:Name="accessControlIconStatic"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="25"

Width="25"

Margin="5, 0, 0, 0"

Source="../Resources/accessControlIconStatic.png"/>

<Image x:Name="accessControlIconActive"

Visibility="Collapsed"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="25"

Width="25"

Margin="5, 0, 0, 0"

Source="../Resources/accessControlIconActive.png"/>

<TextBlock Style="{StaticResource topOptionsGridTextBlock}"

Text="Настройки доступа"

Margin="35, 0, 0, 0"

Width="125"

VerticalAlignment="Center"

HorizontalAlignment="Left"/>

</Grid>

<!--Line grid-->

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="Gray"

Margin="202, 65, 0, 0"

Width="155"

Height="1"/>

<!--publish documentation grid-->

<Grid x:Name="publishDocumentationGrid"

Width="155"

Height="32"

VerticalAlignment="Top"

MouseEnter="OptionGridMouseEnter"

MouseLeave="OptionGridMouseLeave"

HorizontalAlignment="Left"

Margin="202, 66, 0, 0">

<Image x:Name="publishIconStatic"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="25"

Width="25"

Margin="5, 0, 0, 0"

Source="../Resources/shareDocIconStatic.png"/>

<Image x:Name="publishIconActive"

Visibility="Collapsed"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="25"

Width="25"

Margin="5, 0, 0, 0"

Source="../Resources/shareDocIconActive.png"/>

<TextBlock Style="{StaticResource topOptionsGridTextBlock}"

Text="Опубликовать"

Margin="35, 0, 0, 0"

Width="125"

VerticalAlignment="Center"

HorizontalAlignment="Left"/>

</Grid>

</Grid>

<!--User account-->

<Grid x:Name="userAccMainGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="LightGray"

Width="70"

Height="100"

Visibility="Hidden">

<Grid x:Name="userGridLeftMenuGrid"

Visibility="Visible"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="0, 0, 0, 0"

Width="70"

Height="100"

Background="LightGray">

<!--#region Left menu header -->

<Image x:Name="closeAccGridStaticIcon"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="CloseAccGridStaticIconMouseEnter"

Margin="10, 20, 0, 0"

Source="../Resources/closeAccGridStatic.png"

Width="48"

Height="48"

Visibility="Visible"/>

<Image x:Name="closeAccGridActiveIcon"

VerticalAlignment="Top"

MouseLeave="CloseAccGridActiveIconMouseLeave"

MouseDown="HideTheUserAccGrid"

HorizontalAlignment="Left"

Margin="10, 20, 0, 0"

Source="../Resources/closeAccGridActive.png"

Width="48"

Height="48"

Visibility="Collapsed"/>

<TextBlock Text="Опции"

FontFamily="Arial, Verdana"

FontSize="25"

Foreground="OrangeRed"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="78, 30, 0, 0"/>

<Rectangle x:Name="headerBottomLineRect"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="10, 85, 10, 0"

Width="100"

Height="1"

Fill="Gray"/>

<!--#endregion-->

<!--#region Left menu options-->

<Grid x:Name="leftMenuOptionsGridsGrid">

<!--Profile option grid-->

<Grid x:Name="leftMenuUserProfileOptionGrid"

x:Uid="0"

Style="{StaticResource leftMenuOptionGridStyle}"

Margin="10, 100, 10, 0">

<TextBlock Style="{StaticResource leftMenuOptionTextBlockStyle}"

Text="Профиль"/>

</Grid>

<!--User's documentsoption-->

<Grid x:Name="leftMenuUserUserDocsOptionGrid"

x:Uid="1"

Style="{StaticResource leftMenuOptionGridStyle}"

Margin="10, 150, 10, 0">

<TextBlock Style="{StaticResource leftMenuOptionTextBlockStyle}"

Text="Документация"/>

</Grid>

<!--User's shared documents option-->

<Grid x:Name="leftMenuUserSharedDocsOptionGrid"

x:Uid="2"

Style="{StaticResource leftMenuOptionGridStyle}"

Margin="10, 200, 10, 0">

<TextBlock Style="{StaticResource leftMenuOptionTextBlockStyle}"

Text="Шаблоны"/>

</Grid>

<!--Export documentation option-->

<Grid x:Name="leftMenuUserExportOptionGrid"

x:Uid="3"

Style="{StaticResource leftMenuOptionGridStyle}"

Margin="10, 250, 10, 0">

<TextBlock Style="{StaticResource leftMenuOptionTextBlockStyle}"

Text="Экспорт"/>

</Grid>

<!--Quit the programm option-->

<Grid x:Name="leftMenuUserQuitOptionGrid"

x:Uid="4"

Style="{StaticResource leftMenuOptionGridStyle}"

Margin="10, 300, 10, 0">

<TextBlock Style="{StaticResource leftMenuOptionTextBlockStyle}"

Text="Выйти"/>

</Grid>

</Grid>

<!--#endregion-->

</Grid>

<Frame x:Name="userGridFrame"

Visibility="Visible"

VerticalAlignment="Top"

HorizontalAlignment="Left"

NavigationUIVisibility="Hidden"

Background="#E1E1E1">

</Frame>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Text;

using System.Data.SqlClient;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Threading;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

using System.Data.SqlTypes;

using System.Net.Mail;

using ApplicationLib.Models;

using Newtonsoft.Json;

using SDWP.Models;

using System.Threading;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для SDWPMainWindow.xaml

/// </summary>

public partial class SDWPMainWindow : Window

{

#region Properties

private MainGrids MainGrids { get; set; }

private UserAccountGrids UserAccountGrids { get; set; }

private UserAccLeftMenuGrids UserAccLeftMenuGrids { get; set; }

private MainPage MainPage { get; }

private UserDocsPage UserDocsPage { get; set; }

private ExportDocumentationPage ExportDocumentationPage { get; set; }

private UserProfilePage UserProfilePage { get; set; }

private DocumentTemplatesPage DocumentTemplatesPage { get; set; }

#endregion

#region Constructors

public SDWPMainWindow(UserInfo user)

{

InitializeComponent();

InitializePositionObjects();

InitializeInitialPosition();

Position.PositionObj.UpdateMainWindow(this);

mainPageFrame.Content = MainPage = new MainPage();

}

#endregion

#region Initializing methods

/// <summary>

/// Initialize the initial position of this window (top and left margin)

/// </summary>

private void InitializeInitialPosition()

{

Top = 0;

Left = 0;

}

/// <summary>

/// Initializes the objects which are then used in Position class to resize elements when the

/// window is being resized

/// </summary>

private void InitializePositionObjects()

{

MainGrids = new MainGrids

{

TopOptionsGrid = topOptionsGrid,

MainGrid = documentationGrid,

UserAccountGrid = userAccMainGrid

};

UserAccountGrids = new UserAccountGrids

{

UserAccGrid = userAccMainGrid,

LeftMenuGrid = userGridLeftMenuGrid,

ContentFrame = userGridFrame,

};

UserAccLeftMenuGrids = new UserAccLeftMenuGrids

{

HeaderBottomLineRect = headerBottomLineRect,

OptionsGrid = leftMenuOptionsGridsGrid,

};

}

#endregion

#region Size changed events

/// <summary>

/// Resizes the elements when user changes the size of the window

/// </summary>

private void SDWPMainWindowSizeChanged(object sender, SizeChangedEventArgs e)

{

if (WindowState == WindowState.Maximized)

Position.PositionObj.UpdateMainWindow(this);

if (userAccMainGrid.Visibility == Visibility.Visible)

{

Position.PositionObj.InitializeUserAccountGrids(UserAccountGrids);

Position.PositionObj.InitializeUserAccLeftMenuGrids(UserAccLeftMenuGrids);

Position.PositionObj.InitializeUserAccGridPagesParams(userGridFrame);

}

else

Position.PositionObj.InitializeMainGrids(MainGrids);

}

#endregion

#region Top Grids operations

/// <summary>

/// When the mouse in the suitable area of this window this method shows the top options grid,

/// when the mouse leaves the area where top options grid is this method hides the top option grid

/// </summary>

private void SDWPMainWindowMouseMove(object sender, MouseEventArgs e)

{

Point cursorCoordinates = e.GetPosition(this);

if (cursorCoordinates.Y < 5 && cursorCoordinates.X < 500 && topOptionsGrid.Height == 0)

{

ShowTopGrids();

}

else if (cursorCoordinates.Y > 100 && topOptionsGrid.Height == 100)

{

HideTopGrids();

}

}

/// <summary>

/// Changes the colors and font weight of the elements in one of

/// the options element

/// </summary>

private void OptionGridMouseEnter(object sender, MouseEventArgs e)

{

Grid grid = sender as Grid;

List<TextBlock> gridTextBlocksList = grid.Children.OfType<TextBlock>().ToList();

gridTextBlocksList[0].FontWeight = FontWeights.Bold;

grid.Background = new SolidColorBrush(Color.FromRgb(250, 250, 250));

List<Image> iconsList = grid.Children.OfType<Image>().ToList();

iconsList[0].Visibility = Visibility.Collapsed;

iconsList[1].Visibility = Visibility.Visible;

}

/// <summary>

/// Changes everything which was changed in the

/// previous method to its original values

/// </summary>

private void OptionGridMouseLeave(object sender, MouseEventArgs e)

{

Grid grid = sender as Grid;

//make the text usual

List<TextBlock> gridTextBlocksList = grid.Children.OfType<TextBlock>().ToList();

gridTextBlocksList[0].FontWeight = FontWeights.Normal;

//change the background back

grid.Background = new SolidColorBrush(Color.FromRgb(255, 255, 255));

//set the static image visible

List<Image> iconsList = grid.Children.OfType<Image>().ToList();

iconsList[0].Visibility = Visibility.Visible;

iconsList[1].Visibility = Visibility.Collapsed;

}

/// <summary>

/// Hides the top option grid when the mouse leaves the top option grid

/// </summary>

private void HideTopGrids()

{

topOptionsGrid.BeginAnimation(HeightProperty,

CreateUpTopGridAnimation());

mainPageFrame.BeginAnimation(MarginProperty,

CreateDocumentFrameUpAnimation());

}

/// <summary>

/// Shows top option grids when the mouse is in a suitable position (under the documents panel)

/// </summary>

private void ShowTopGrids()

{

topOptionsGrid.BeginAnimation(HeightProperty,

CreateDownTopGridAnimation());

mainPageFrame.BeginAnimation(MarginProperty,

CreateDocumentFrameDownAnimation());

}

/// <summary>

/// Creates ans returns the animation which is used to hide the top options grid

/// This animation animates the height of the grid

/// </summary>

private DoubleAnimation CreateUpTopGridAnimation()

{

DoubleAnimation optionsGridHeightAnimation = new DoubleAnimation

{

From = 100,

To = 0,

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

optionsGridHeightAnimation.Completed += (send, events) =>

{

topOptionsGrid.Height = 0;

};

return optionsGridHeightAnimation;

}

/// <summary>

/// Creates and returns an animation which is used to show the top options grid

/// This animation animates the height of the grid

/// </summary>

private DoubleAnimation CreateDownTopGridAnimation()

{

DoubleAnimation optionsGridHeightAnimation = new DoubleAnimation

{

From = 0,

To = 100,

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

optionsGridHeightAnimation.Completed += (send, events) =>

{

topOptionsGrid.Height = 100;

};

return optionsGridHeightAnimation;

}

/// <summary>

/// Creates and returns an animation which is used to lower the document frame when the

/// top option grid is being shown

/// </summary>

private ThicknessAnimation CreateDocumentFrameDownAnimation()

{

ThicknessAnimation lowerTheDocumentFrameAnimation = new ThicknessAnimation

{

To = new Thickness(0, 100, 0, 0),

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

lowerTheDocumentFrameAnimation.Completed += (send, events) =>

{

mainPageFrame.Margin = new Thickness(0, 100, 0, 0);

};

return lowerTheDocumentFrameAnimation;

}

/// <summary>

/// Creates and returns an animation which is used to up rise the document frame when

/// the top options grid is being hidden

/// </summary>

private ThicknessAnimation CreateDocumentFrameUpAnimation()

{

ThicknessAnimation upperTheDocumentFrameAnimation = new ThicknessAnimation

{

To = new Thickness(0, 0, 0, 0),

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

upperTheDocumentFrameAnimation.Completed += (send, events) =>

{

mainPageFrame.Margin = new Thickness(0, 0, 0, 0);

};

return upperTheDocumentFrameAnimation;

}

#endregion

#region User main grid operations

/// <summary>

/// After a click on an option "Account" in a top options grid shows the User main grid

/// where all account functionality is

/// </summary>

private void ShowTheUserMainGrid(object sender, MouseButtonEventArgs e)

{

userAccMainGrid.Width = 0;

Position.PositionObj.InitializeUserAccountGrids(UserAccountGrids);

Position.PositionObj.InitializeUserAccLeftMenuGrids(UserAccLeftMenuGrids);

userAccMainGrid.Visibility = Visibility.Visible;

userAccMainGrid.BeginAnimation(WidthProperty,

CreateShowMainUserGridAnimation());

}

/// <summary>

/// Clears the history of a frame, where all pages which in User main grid are displayed

/// </summary>

private void ClearFrameHistory(Frame frame)

{

frame.Navigate(new Page());

if (!frame.CanGoBack && !frame.CanGoForward)

return;

var entry = frame.RemoveBackEntry();

while (frame.CanGoBack)

entry = frame.RemoveBackEntry();

frame.Navigate(new PageFunction<string>() { RemoveFromJournal = true });

}

/// <summary>

/// Creates and returns an animation which is used to show user main grid

/// </summary>

private DoubleAnimation CreateShowMainUserGridAnimation()

{

DoubleAnimation showTheGridAnimation = new DoubleAnimation

{

From = 0,

To = 3 \* this.Width / 4,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

showTheGridAnimation.Completed += (sender, e) =>

{

userAccMainGrid.Width = 3 \* this.Width / 4;

};

return showTheGridAnimation;

}

/// <summary>

/// Creates and returns an animation which is used to show user main grid

/// </summary>

private DoubleAnimation CreateHideMainGridAnimation()

{

DoubleAnimation hideTheGridAnimation = new DoubleAnimation

{

From = 3 \* this.Width / 4,

To = 0,

Duration = TimeSpan.FromMilliseconds(200),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

hideTheGridAnimation.Completed += (sender, e) =>

{

userAccMainGrid.Width = 0;

userAccMainGrid.Visibility = Visibility.Collapsed;

};

return hideTheGridAnimation;

}

private void CloseAccGridActiveIconMouseLeave(object sender, MouseEventArgs e)

{

closeAccGridStaticIcon.Visibility = Visibility.Visible;

closeAccGridActiveIcon.Visibility = Visibility.Collapsed;

}

private void HideTheUserAccGrid(object sender, EventArgs e)

{

userAccMainGrid.BeginAnimation(WidthProperty,

CreateHideMainGridAnimation());

}

private void CloseAccGridStaticIconMouseEnter(object sender, EventArgs e)

{

closeAccGridStaticIcon.Visibility = Visibility.Collapsed;

closeAccGridActiveIcon.Visibility = Visibility.Visible;

}

#endregion

#region User grid event handlers

private void LeftMenuOptionGridMouseEnter(object sender, EventArgs e)

{

Grid optionGrid = sender as Grid;

optionGrid.Background = new SolidColorBrush(Color.FromRgb(219, 219, 219));

TextBlock optionGridText = optionGrid.Children[0] as TextBlock;

optionGridText.FontWeight = FontWeights.Bold;

}

private void LeftMenuOptionGridMouseLeave(object sender, EventArgs e)

{

Grid optionGrid = sender as Grid;

optionGrid.Background = new SolidColorBrush(Colors.LightGray);

TextBlock optionGridText = optionGrid.Children[0] as TextBlock;

optionGridText.FontWeight = FontWeights.Normal;

}

private void LeftMenuOptionGridMouseDown(object sender, EventArgs e)

{

Grid clickedOptionGrid = sender as Grid;

SwitchTheMainGrid(clickedOptionGrid.Uid);

}

/// <summary>

/// Uploads the page which realtes to the clicked header in user main grid to

/// userGridFrame

/// </summary>

private void SwitchTheMainGrid(string clickedGridUid)

{

ClearFrameHistory(userGridFrame);

switch (clickedGridUid)

{

case "0":

if (UserProfilePage == null)

{

UserProfilePage = new UserProfilePage(UserInfo.CurrentUser)

{

Width = userGridFrame.Width,

CloseAccGrid = new Action(() => HideTheUserAccGrid(null, null))

};

}

userGridFrame.Navigate(UserProfilePage);

break;

case "1":

if (UserDocsPage == null)

{

UserDocsPage = new UserDocsPage(MainPage)

{

Width = userGridFrame.Width,

CloseAccGrid = new Action(() => HideTheUserAccGrid(null, null))

};

}

userGridFrame.Navigate(UserDocsPage);

break;

case "2":

if (DocumentTemplatesPage == null)

{

DocumentTemplatesPage = new DocumentTemplatesPage(UserInfo.CurrentUser)

{

Width = userGridFrame.Width,

Height = userGridFrame.Height,

CloseAccGrid = new Action(() => HideTheUserAccGrid(null, null))

};

}

userGridFrame.Navigate(DocumentTemplatesPage);

break;

case "3":

if (ExportDocumentationPage == null)

{

ExportDocumentationPage = new ExportDocumentationPage(MainPage)

{

Width = userGridFrame.Width,

CloseAccGrid = new Action(() => HideTheUserAccGrid(null, null))

};

}

userGridFrame.Navigate(ExportDocumentationPage);

break;

case "4":

QuitTheProgram();

break;

}

}

#endregion

#region Top option grid options methods

/// <summary>

/// When the save option is pressed this method opens Export documentation page

/// in a user main grid

/// </summary>

private void SaveDocumentation(object sender, EventArgs e)

{

ShowTheUserMainGrid(null, null);

//3 = export documentation page

SwitchTheMainGrid("3");

}

/// <summary>

/// Opens the user doc page in the user main grid when the "delete documentation" option is pressed

/// </summary>

private void OpenUserDocsPage(object sender, RoutedEventArgs e)

{

ShowTheUserMainGrid(null, null);

//1 = user's documentations page

SwitchTheMainGrid("1");

}

#endregion

private void QuitTheProgram()

{

if (SDWPMessageBox.ShowSDWPMessageBox("Подтверждение", "Вы уверены, что хотите выйти?", MessageBoxButton.OKCancel)

== MessageBoxResult.OK)

{

Close();

(new MainWindow()).Show();

}

}

#region Close window methods

#warning add this ti the table

private void SdwpMainWindowClosed(object sender, EventArgs e)

{

App.Current.Shutdown();

}

#endregion

}

}

<Window x:Class="SDWP.SDWPMessageBox"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

mc:Ignorable="d"

Title="SDWP Software message"

Height="210"

Width="400"

MinHeight="210"

MinWidth="400"

MaxHeight="210"

MaxWidth="400"

WindowStartupLocation="CenterScreen">

<Window.Resources>

<Style TargetType="Button">

<Setter Property="OverridesDefaultStyle" Value="True"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Width" Value="90"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="13"/>

<Setter Property="Foreground" Value="OrangeRed"/>

<Setter Property="BorderThickness" Value="2"/>

<Setter Property="BorderBrush" Value="OrangeRed"/>

<Setter Property="Background" Value="White"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="Margin" Value="0, 0, 20, 0"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border CornerRadius="5"

BorderBrush="{StaticResource mainThemeBrush}"

BorderThickness="1"

SnapsToDevicePixels="True"

Background="{TemplateBinding Background}">

<ContentPresenter HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="SDWPMessageBoxBtnMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="SDWPMessageBoxBtnMouseLeave"/>

</Style>

</Window.Resources>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="400"

Height="210">

<Grid x:Name="warningIconGrid"

Width="120"

Height="130"

Background="White"

VerticalAlignment="Top"

HorizontalAlignment="Left">

<Image x:Name="warningIcon"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="100"

Height="100"

Source="../Resources/SDWPmessageBoxWarningIcon.png"/>

<Rectangle Width="1"

Height="100"

VerticalAlignment="Center"

HorizontalAlignment="Right"

Fill="#E1E1E1"/>

</Grid>

<Grid x:Name="messageGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="White"

Width="270"

Height="130"

Margin="120, 0, 0, 0">

<TextBlock x:Name="messageTextBlock"

FontFamily="Arial, Verdana"

FontSize="15"

TextWrapping="WrapWithOverflow"

Foreground="OrangeRed"

VerticalAlignment="Center"

HorizontalAlignment="Center"/>

</Grid>

<Grid VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="400"

Height="44"

Background="#E1E1E1"

Margin="0,130,0,0">

<StackPanel x:Name="buttonsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Height="44"

Orientation="Horizontal"

Background="#E1E1E1"/>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace SDWP

{

public partial class SDWPMessageBox : Window

{

#region Properties

private MessageBoxResult MessageBoxResult { get; set; } = MessageBoxResult.None;

#endregion

public SDWPMessageBox()

{

InitializeComponent();

}

#warning add this method to the table

public static MessageBoxResult ConfirmAction()

{

SDWPMessageBox messageBox = new SDWPMessageBox { Title = "Подтверждение действия" };

messageBox.messageTextBlock.Text = "Вы действително хотите сделать это?";

messageBox.AddMessageBoxButtons(MessageBoxButton.OKCancel);

messageBox.ShowDialog();

return messageBox.MessageBoxResult;

}

public static MessageBoxResult ShowSDWPMessageBox(string title, string message,

MessageBoxButton buttons)

{

SDWPMessageBox messageBox = new SDWPMessageBox { Title = title };

messageBox.messageTextBlock.Text = message;

messageBox.AddMessageBoxButtons(buttons);

messageBox.ShowDialog();

return messageBox.MessageBoxResult;

}

#region Add buttons methods

public void AddMessageBoxButtons(MessageBoxButton buttons)

{

switch (buttons)

{

case MessageBoxButton.OK:

AddMessageBoxButton("OK", MessageBoxResult.OK, isDefault: true);

break;

case MessageBoxButton.OKCancel:

AddMessageBoxButton("Cancel", MessageBoxResult.Cancel,

isCancel: true);

AddMessageBoxButton("OK", MessageBoxResult.OK, isDefault: true);

break;

case MessageBoxButton.YesNo:

AddMessageBoxButton("Yes", MessageBoxResult.Yes, isDefault: true);

AddMessageBoxButton("No", MessageBoxResult.No);

break;

case MessageBoxButton.YesNoCancel:

AddMessageBoxButton("No", MessageBoxResult.No);

AddMessageBoxButton("Yes", MessageBoxResult.Yes, isDefault: true);

AddMessageBoxButton("Cancel", MessageBoxResult.Cancel,

isCancel: true);

break;

}

}

private void AddMessageBoxButton(string text, MessageBoxResult messageBoxResult,

bool isCancel = false, bool isDefault = false)

{

Button button = new Button()

{

Content = text,

IsCancel = isCancel,

IsDefault = isDefault

};

button.Click += (sender, e) =>

{

MessageBoxResult = messageBoxResult;

DialogResult = true;

};

buttonsGrid.Children.Add(button);

}

#endregion

#region Event handlers

private void SDWPMessageBoxBtnMouseEnter(object sender, MouseEventArgs e)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.OrangeRed);

button.Foreground = new SolidColorBrush(Colors.White);

}

private void SDWPMessageBoxBtnMouseLeave(object sender, MouseEventArgs e)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.White);

button.Foreground = new SolidColorBrush(Colors.OrangeRed);

}

#endregion

}

}

<Page x:Class="SDWP.UserDocsPage"

x:Name="userDocsPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:SDWP"

Title="UserDocsPage">

<Page.Resources>

<Style x:Key="userDocsPropertiesName" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="10"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

</Style>

<Style x:Key="userDocsTextBox" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="userDocsDataHeader" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="Foreground" Value="OrangeRed"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontWeight" Value="Bold"/>

</Style>

<Style x:Key="userDocsButton" TargetType="Button">

<Setter Property="OverridesDefaultStyle" Value="True"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="12"/>

<Setter Property="Padding" Value="0"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="150"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="Background" Value="{StaticResource mainThemeColor}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border CornerRadius="5"

BorderBrush="{StaticResource mainThemeBrush}"

BorderThickness="1"

SnapsToDevicePixels="True"

Background="{TemplateBinding Background}">

<ContentPresenter HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="UserDocsBtnMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="UserDocsBtnMouseLeave"/>

</Style>

<Style x:Key="userProfileTextBlockStyle" TargetType="TextBlock">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="{StaticResource mainThemeColor}"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

</Style>

<Style x:Key="documentationListBoxHeaderTextBlockStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

</Style>

<LinearGradientBrush x:Key="documentationOptionBorderBrush"

StartPoint="0, 0"

EndPoint="0, 1">

<GradientStop Color="OrangeRed" Offset="0.37"/>

<GradientStop Color="White" Offset="0.37"/>

</LinearGradientBrush>

<LinearGradientBrush x:Key="listboxSelectedItemGradient">

<GradientStop Color="Orange" Offset="0.3"/>

<GradientStop Color="Orange" Offset="0.7"/>

</LinearGradientBrush>

<Style TargetType="ListBoxItem">

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="Margin" Value="0, 10, 0, 0"/>

<Setter Property="Background" Value="LightGray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ListBoxItem">

<Border x:Name="border"

SnapsToDevicePixels="true"

Background="{TemplateBinding Background}"

CornerRadius="5"

BorderThickness="1"

BorderBrush="Gray">

<ContentPresenter/>

</Border>

<ControlTemplate.Triggers>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="True" />

<Condition Property="IsSelected" Value="True" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="#f0f0f0"/>

</MultiTrigger.Setters>

</MultiTrigger>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="False" />

<Condition Property="IsSelected" Value="True" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="#f0f0f0"/>

</MultiTrigger.Setters>

</MultiTrigger>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="False" />

<Condition Property="IsSelected" Value="False" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="LightGray"/>

</MultiTrigger.Setters>

</MultiTrigger>

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="True" />

<Condition Property="IsSelected" Value="False" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter TargetName="border"

Property="Background"

Value="#f0f0f0"/>

</MultiTrigger.Setters>

</MultiTrigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="ListBoxItemMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="ListBoxItemMouseLeave"/>

<EventSetter Event="PreviewMouseDown" Handler="ListBoxItemMouseDown"/>

</Style>

<Style x:Key="documentationOptionsTextBlockStyle" TargetType="TextBlock">

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="FontSize" Value="13"/>

</Style>

<DataTemplate x:Key="documentationListBoxDataTemplate">

<Grid VerticalAlignment="Center"

HorizontalAlignment="Center"

Margin="0, 0, 0, 0"

Width="280">

<Border x:Name="border"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Width="280"

Height="50"

BorderThickness="0"

Background="Transparent"

Padding="5"

BorderBrush="Gray"

Margin="0, 0, 0, 0">

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Height="50"

Width="280"

Orientation="Vertical">

<StackPanel VerticalAlignment="Center"

HorizontalAlignment="Left"

Orientation="Horizontal"

Height="50"

Width="280">

<Image Source="../Resources/documentationIcon.png"

VerticalAlignment="Center"

HorizontalAlignment="Left"

Height="20"

Width="20"

Margin="5, -12, 0, 0"/>

<TextBlock Text="{Binding Path = Name}"

Style="{StaticResource documentationListBoxHeaderTextBlockStyle}"

Margin="10, -10, 0, 0"/>

</StackPanel>

</StackPanel>

</Border>

</Grid>

</DataTemplate>

<Style x:Key="documentationListBoxStyle" TargetType="ListBox">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Background" Value="#E1E1E1"/>

<Setter Property="Width" Value="300"/>

<Setter Property="Height" Value="400"/>

<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Hidden"/>

<Setter Property="ScrollViewer.HorizontalScrollBarVisibility" Value="Hidden"/>

<Setter Property="BorderThickness" Value="0, 0, 1, 0"/>

<Setter Property="BorderBrush" Value="Gray"/>

<Setter Property="Padding" Value="0, 0, 10, 0"/>

<Setter Property="ItemTemplate" Value="{StaticResource documentationListBoxDataTemplate}"/>

</Style>

<Style x:Key="doucmentationTypesTextBlockStyle" TargetType="TextBlock">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="20"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontWeight" Value="Light"/>

<EventSetter Event="MouseEnter" Handler="DocTypesTextBlockMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="DocTypesTextBlockMouseLeave"/>

</Style>

<Style x:Key="filePathTextBoxStyle" TargetType="TextBox">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Background" Value="Transparent"/>

<Setter Property="Padding" Value="5, 5, 5, 5"/>

<Setter Property="Height" Value="40"/>

<Setter Property="Width" Value="300"/>

<Setter Property="IsReadOnly" Value="True"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border x:Name="border"

BorderThickness="1"

BorderBrush="LightGray"

CornerRadius="10"

Background="White"

Padding="{TemplateBinding Padding}">

<ScrollViewer x:Name="PART\_ContentHost"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="documentationProperteisBtnStyle" TargetType="Button">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Right"/>

<Setter Property="Width" Value="110"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border x:Name="border"

BorderThickness="1"

CornerRadius="10"

Background="#F0F0F0"

BorderBrush="LightGray"

Padding="{TemplateBinding Padding}">

<ContentPresenter x:Name="contentPresenter"

VerticalAlignment="Center"

HorizontalAlignment="Center"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetProperty="BorderBrush.Color"

Storyboard.TargetName="border"

To="{StaticResource mainThemeColor}"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Normal">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="BorderBrush.Color"

To="#F0F0F0"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="localDocOptionBtnStyle" TargetType="Button">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Width" Value="100"/>

<Setter Property="Height" Value="100"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border x:Name="border"

BorderThickness="1"

CornerRadius="10"

Background="#F0F0F0"

BorderBrush="LightGray"

Padding="{TemplateBinding Padding}">

<ContentPresenter x:Name="contentPresenter"

VerticalAlignment="Center"

HorizontalAlignment="Center"/>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Storyboard.TargetProperty="BorderBrush.Color"

Storyboard.TargetName="border"

To="{StaticResource mainThemeColor}"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Normal">

<Storyboard>

<ColorAnimation Storyboard.TargetName="border"

Storyboard.TargetProperty="BorderBrush.Color"

To="#F0F0F0"

SpeedRatio="100"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="localDocOptionBtnImageStyle" TargetType="Image">

<Setter Property="Width" Value="40"/>

<Setter Property="Height" Value="40"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

</Style>

<Style x:Key="localDocOptionBtnTextBlockStyle" TargetType="TextBlock">

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="{StaticResource defaultTextColorBrush}"/>

<Setter Property="FontWeight" Value="Light"/>

</Style>

</Page.Resources>

<Page.Content>

<Grid x:Name="userDocsOutterGrid">

<ScrollViewer VerticalAlignment="Top"

VerticalScrollBarVisibility="Hidden"

HorizontalScrollBarVisibility="Auto"

Width="{Binding ElementName=userDocsPage, Path=ActualWidth}"

Height="{Binding ElementName=userDocsPage, Path=ActualHeight}">

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<local:PageHeader x:Name="pageHeader"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="{Binding ElementName=userDocsPage, Path=ActualWidth}"

Height="86"

Header="Документация"/>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 25, 0, 0"

Orientation="Horizontal">

<TextBlock x:Name="goToLocalDocumentation"

Text="Локальная документация"

Style="{StaticResource doucmentationTypesTextBlockStyle}"

PreviewMouseDown="GoToLocalDocumentationPanel"/>

<TextBlock x:Name="goToCloudDocumentation"

Text="Облачная документация"

Margin="20, 0, 0, 0"

Style="{StaticResource doucmentationTypesTextBlockStyle}"

PreviewMouseDown="GoToCloudDocumentationPanel"/>

</StackPanel>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Horizontal"

Margin="0, 0, 20, 0">

<Grid x:Name="userDocumentationGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="#E1E1E1"

Margin="0, 20, 0, 0">

<StackPanel x:Name="cloudDocStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Vertical"

Visibility="Collapsed">

<ListBox x:Name="onlineDocumentationListBox"

Style="{StaticResource documentationListBoxStyle}"

Margin="20, 5, 0, 0"/>

</StackPanel>

<StackPanel x:Name="localDocStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Vertical"

Margin="0, 0, 0, 0"

Visibility="Visible">

<ListBox x:Name="offlineDocumentationListBox"

Style="{StaticResource documentationListBoxStyle}"

Margin="20, 5, 0, 0"/>

</StackPanel>

</Grid>

<Grid x:Name="documentationPropertiesGrid">

<!--Local documentation grid-->

<StackPanel x:Name="localDocumentationPropertiesStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Visibility="Visible"

Orientation="Vertical">

<TextBlock FontFamily="Arial, Verdana"

FontSize="15"

Foreground="OrangeRed"

Margin="20, 30, 0, 0"

Text="Путь до папки с документациями"/>

<TextBox x:Name="filePathTextBox"

Style="{StaticResource filePathTextBoxStyle}"

Text=""

Margin="20, 5, 0, 0"/>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Horizontal"

Margin="20, 0, 0, 0">

<Button x:Name="createDocumentationBtn"

Style="{StaticResource documentationProperteisBtnStyle}"

Margin="0, 10, 0, 0"

Click="CreateLocalDocumentation">

<TextBlock FontFamily="Arial, Verdana"

FontSize="14"

Foreground="{StaticResource mainThemeBrush}"

Text="Создать"

VerticalAlignment="Top"

HorizontalAlignment="Left"/>

</Button>

<Button x:Name="selectDocumentationFolderBtn"

Style="{StaticResource documentationProperteisBtnStyle}"

Margin="100, 10, 0, 0"

Click="SelectLocalDocumentationFolder">

<TextBlock FontFamily="Arial, Verdana"

FontSize="14"

Foreground="{StaticResource mainThemeBrush}"

Text="Обзор"

VerticalAlignment="Top"

HorizontalAlignment="Left"/>

</Button>

</StackPanel>

<Grid HorizontalAlignment="Stretch"

VerticalAlignment="Top"

Background="Gray"

Height="1"

Margin="20, 10, 0, 0"/>

<StackPanel x:Name="localDocsOptionsPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Vertical"

Margin="20, 10, 0, 0">

<StackPanel Orientation="Horizontal">

<TextBlock Style="{StaticResource localDocOptionBtnTextBlockStyle}"

Text="Документация: "

Foreground="Gray"

HorizontalAlignment="Left"/>

<TextBlock x:Name="selectedLocalDocumentationNameTextBlock"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"

HorizontalAlignment="Left"/>

</StackPanel>

<StackPanel Orientation="Horizontal"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="0, 10, 0, 0">

<Button x:Name="openLocalDocumentationBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Click="UploadLocalDocumentationToMainPage">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/openLocalDocumentationIcon.png"/>

<TextBlock Text="Открыть"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

<Button x:Name="deleteLocalDocumentationBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Margin="5, 0, 0, 0"

Click="DeleteLocalDocumentation">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/deleteLocalDocumentationIcon.png"/>

<TextBlock Text="Удалить"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

<Button x:Name="publishDocumentationBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Margin="5, 0, 0, 0"

Click="PublishLocalDocumentation">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/shareLocalDocumentationIcon.png"/>

<TextBlock Text="Публикация"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

</StackPanel>

</StackPanel>

</StackPanel>

<!--Cloud documentation stack pannel -->

<StackPanel x:Name="cloudDocumentationPropertiesStackPanel"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Vertical"

Visibility="Collapsed"

Margin="20, 15, 0, 0">

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Horizontal"

Margin="0, 20, 0, 0">

<Button x:Name="createCloudDocumentationBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Click="CreateCloudDocumentation">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/createNewCloudDocIcon.png"/>

<TextBlock Text="Создать"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

<Button x:Name="openCloudDocumentationBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Margin="5, 0, 0, 0"

Click="UploadCloudDocumentationToMainPage">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/openLocalDocumentationIcon.png"/>

<TextBlock Text="Открыть"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

<Button x:Name="deleteCloudDocumentationBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Margin="5, 0, 0, 0"

Click="DeleteCloudDocumentation">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/deleteLocalDocumentationIcon.png"/>

<TextBlock Text="Удалить"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

</StackPanel>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Left"

Orientation="Horizontal"

Margin="0, 10, 0, 0">

<Button x:Name="createLocalCopyBtn"

Style="{StaticResource localDocOptionBtnStyle}"

Width="200"

Margin="0, 0, 0, 0"

Click="CreateLocalCopy">

<Button.Content>

<StackPanel VerticalAlignment="Top"

HorizontalAlignment="Stretch"

Orientation="Vertical">

<Image Style="{StaticResource localDocOptionBtnImageStyle}"

Source="../Resources/createLocalCopyIcon.png"/>

<TextBlock Text="Создать локальную копию"

Margin="0, 10, 0, 0"

Style="{StaticResource localDocOptionBtnTextBlockStyle}"/>

</StackPanel>

</Button.Content>

</Button>

</StackPanel>

</StackPanel>

</Grid>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Grid>

</Page.Content>

</Page>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.IO;

using System.Data.SqlClient;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Factories;

using SDWP.Factories;

using SDWP.Interfaces;

using Microsoft.Win32;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для UserDocsPage.xaml

/// </summary>

public partial class UserDocsPage : Page, IAccountPage

{

#region IAccountPage properties

public Action CloseAccGrid { get; set; }

#endregion

#region Services and factories

private IServiceAbstractFactory ServiceAbstractFactory { get; set; }

private ISdwpAbstractFactory SdwpAbstractFactory { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

private ICloudDocumentationService CloudDocumentationService { get; set; }

private ICloudDocumentsService CloudDocumentsService { get; set; }

private ILocalDocumentationService LocalDocumentationService { get; set; }

#endregion

#region Propeties

private PageHeader PageHeader { get; }

private string CurrentFilePath { get; set; }

private List<LocalDocumentation> LocalDocumentations { get; set; }

private List<Documentation> CloudDocumentation { get; set; }

private MainPage MainPage { get; }

private StackPanel LocalDocsPanel { get; set; }

private StackPanel CloudDocsPanel { get; set; }

private TextBlock GoToLocalDocsTextBlock { get; set; }

private TextBlock GoToCloudDocsTextBlock { get; set; }

private TextBlock SelectedLocalDocumentationTextBlock { get; set; }

private StackPanel LocalDocumentationPropertiesStackPanel { get; set; }

private StackPanel CloudDocumentationPropertiesStackPanel { get; set; }

private TextBox FilePathTextBox { get; set; }

private ListBox LocalDocumentationListBox { get; set; }

private ListBox CloudDocumentationListBox { get; set; }

#endregion

#region Constructors and initializing methods

public UserDocsPage(MainPage mainPage)

{

InitializeComponent();

InitializeServices();

InitializeProperties();

MainPage = mainPage;

GoToLocalDocumentationPanel(null, null);

PageHeader = pageHeader;

}

private void InitializeProperties()

{

LocalDocsPanel = localDocStackPanel;

CloudDocsPanel = cloudDocStackPanel;

GoToCloudDocsTextBlock = goToCloudDocumentation;

GoToLocalDocsTextBlock = goToLocalDocumentation;

SelectedLocalDocumentationTextBlock = selectedLocalDocumentationNameTextBlock;

LocalDocumentationPropertiesStackPanel = localDocumentationPropertiesStackPanel;

CloudDocumentationPropertiesStackPanel = cloudDocumentationPropertiesStackPanel;

FilePathTextBox = filePathTextBox;

LocalDocumentationListBox = offlineDocumentationListBox;

CloudDocumentationListBox = onlineDocumentationListBox;

}

private void InitializeServices()

{

SdwpAbstractFactory = new SdwpAbstractFactory();

ServiceAbstractFactory = new ServiceAbstractFactory();

ExceptionHandler = SdwpAbstractFactory.GetExceptionHandler(Dispatcher);

CloudDocumentationService = ServiceAbstractFactory.GetCloudDocumentationService();

LocalDocumentationService = ServiceAbstractFactory.GetLocalDocumentationService();

CloudDocumentsService = ServiceAbstractFactory.GetCloudDocumentsService();

}

#endregion

#region Upload documentation methods

private async Task UploadDocumentationsFromLocalSotrage()

{

try

{

LocalDocumentations = (await LocalDocumentationService.GetLocalDocumentations()).ToList();

LocalDocumentationListBox.ItemsSource = LocalDocumentations.Select(lc => lc.Documentation).ToList();

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async Task UploadDocumentationsFromCloudSotrage()

{

PageHeader.SwitchOnTopLoader();

try

{

CloudDocumentation = (await CloudDocumentationService.GetUserDocumentations(UserInfo.CurrentUser.ID))

.ToList();

CloudDocumentationListBox.ItemsSource = CloudDocumentation;

PageHeader.SwitchOffTheLoader();

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

#endregion

#region Event handlers

private void UserDocsBtnMouseEnter(object sender, MouseEventArgs e)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.White);

button.Foreground = new SolidColorBrush(Colors.OrangeRed);

}

private void UserDocsBtnMouseLeave(object sender, MouseEventArgs e)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.OrangeRed);

button.Foreground = new SolidColorBrush(Colors.White);

}

private void ListBoxItemMouseEnter(object sender, MouseEventArgs e)

{

(sender as ListBoxItem).Background = new SolidColorBrush(Color.FromRgb(240, 240, 240));

}

private void ListBoxItemMouseLeave(object sender, MouseEventArgs e)

{

ListBoxItem listBoxItem = sender as ListBoxItem;

if (!listBoxItem.IsSelected)

{

listBoxItem.Background = new SolidColorBrush(Colors.LightGray);

}

}

private void ListBoxItemMouseDown(object sender, MouseButtonEventArgs e)

{

ListBoxItem selectedItem = sender as ListBoxItem;

selectedItem.IsSelected = true;

SelectedLocalDocumentationTextBlock.Text = (selectedItem.DataContext as Documentation).Name;

}

private void DocTypesTextBlockMouseEnter(object sender, MouseEventArgs e)

{

(sender as TextBlock).TextDecorations.Add(TextDecorations.Underline);

}

private void DocTypesTextBlockMouseLeave(object sender, MouseEventArgs e)

{

(sender as TextBlock).TextDecorations.Clear();

}

private async void GoToLocalDocumentationPanel(object sender, RoutedEventArgs e)

{

GoToLocalDocsTextBlock.Foreground = new SolidColorBrush(Colors.OrangeRed);

GoToCloudDocsTextBlock.Foreground = new SolidColorBrush(Colors.Black);

LocalDocsPanel.Visibility = Visibility.Visible;

CloudDocsPanel.Visibility = Visibility.Collapsed;

CloudDocumentationPropertiesStackPanel.Visibility = Visibility.Collapsed;

LocalDocumentationPropertiesStackPanel.Visibility = Visibility.Visible;

//if the user selected the folder with templates then we have to upload local docs

if (LocalDocumentationService.StoragePath != null)

await UploadDocumentationsFromLocalSotrage();

}

private async void GoToCloudDocumentationPanel(object sender, RoutedEventArgs e)

{

GoToLocalDocsTextBlock.Foreground = new SolidColorBrush(Colors.Black);

GoToCloudDocsTextBlock.Foreground = new SolidColorBrush(Colors.OrangeRed);

LocalDocsPanel.Visibility = Visibility.Collapsed;

CloudDocsPanel.Visibility = Visibility.Visible;

CloudDocumentationPropertiesStackPanel.Visibility = Visibility.Visible;

LocalDocumentationPropertiesStackPanel.Visibility = Visibility.Collapsed;

await UploadDocumentationsFromCloudSotrage();

}

#endregion

#region Cloud documentation methods

/// <summary>

/// Deletes the documentation from the database and also deletes all the docuemnts which were in

/// that docuemntation

/// </summary>

private async void DeleteCloudDocumentation(object sender, RoutedEventArgs e)

{

PageHeader.SwitchOnTopLoader();

if (SDWPMessageBox.ConfirmAction() == MessageBoxResult.Cancel)

return;

try

{

if (CloudDocumentationListBox.SelectedItem is Documentation selectedDocumentation)

{

await CloudDocumentationService.DeleteDocumentation(selectedDocumentation);

await DeleteAllDocumentationDocuments(selectedDocumentation.ID);

await UploadDocumentationsFromCloudSotrage();

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех", "Документация и все документы связанные с ней успешно удалены",

MessageBoxButton.OK);

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете документацию",

MessageBoxButton.OK);

}

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async Task DeleteAllDocumentationDocuments(int documentationID)

{

IEnumerable<Document> documents = await CloudDocumentsService.

GetDocumentationDocuments(documentationID);

foreach (Document document in documents)

{

await CloudDocumentsService.DeleteDocument(document);

}

}

private async void CreateCloudDocumentation(object sender, RoutedEventArgs e)

{

PageHeader.SwitchOnTopLoader();

try

{

CreateNewDocumentationWindow createNewDocumentationWindow =

new CreateNewDocumentationWindow(CloudDocumentationService);

if (createNewDocumentationWindow.ShowDialog() == true)

{

await UploadDocumentationsFromCloudSotrage();

}

PageHeader.SwitchOffTheLoader();

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async void UploadCloudDocumentationToMainPage(object sender, RoutedEventArgs e)

{

PageHeader.SwitchOnTopLoader();

try

{

if (CloudDocumentationListBox.SelectedItem is Documentation selectedDocumentation)

{

List<Document> documents = (await CloudDocumentsService.

GetDocumentationDocuments(selectedDocumentation.ID)).ToList();

MainPage.UploadCloudDocumentation(selectedDocumentation, documents);

CloseAccGrid();

PageHeader.SwitchOffTheLoader();

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете докуемнтацию для открытия",

MessageBoxButton.OK);

}

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Creates a local copy of a selected cloud documentation

/// </summary>

private async void CreateLocalCopy(object sender, RoutedEventArgs e)

{

PageHeader.SwitchOnTopLoader();

try

{

if (CloudDocumentationListBox.SelectedItem is Documentation selectedDocumentation)

{

List<Document> documents = (await CloudDocumentsService.

GetDocumentationDocuments(selectedDocumentation.ID)).ToList();

selectedDocumentation.StorageType = StorageType.Local;

string filePath = GetFilePathToLocalCopy();

if (filePath != null)

{

LocalDocumentation localDocumentation = new LocalDocumentation(

selectedDocumentation, documents)

{

DocumentationPath = filePath

};

await LocalDocumentationService.CreateLocalDocumentationFile(localDocumentation);

PageHeader.SwitchOffTheLoader();

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Вы не выбрали путь для сохранения",

MessageBoxButton.OK);

}

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете документацию из облака",

MessageBoxButton.OK);

}

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (IOException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Gets a file path via save file dialog for a copy of a cloud documentation

/// </summary>

private string GetFilePathToLocalCopy()

{

SaveFileDialog saveFileDialog = new SaveFileDialog()

{

Title = "Выберете место для создание копии",

FileName = "Новый документ",

AddExtension = true,

Filter = "(\*sdwp)|\*sdwp",

OverwritePrompt = true,

};

if (saveFileDialog.ShowDialog() == true)

{

string filePath = saveFileDialog.FileName;

if (filePath.Substring(filePath.Length - LocalDocumentationService.Extension.Length) !=

LocalDocumentationService.Extension)

{

filePath += LocalDocumentationService.Extension;

}

return filePath;

}

return null;

}

#endregion

#region Local documentation methods

private async void CreateLocalDocumentation(object sender, RoutedEventArgs e)

{

try

{

if (LocalDocumentationService.StoragePath == null)

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала откройте папку с документациями",

MessageBoxButton.OK);

return;

}

CreateNewDocumentationWindow createNewDocumentationWindow =

new CreateNewDocumentationWindow(LocalDocumentationService);

if (createNewDocumentationWindow.ShowDialog() == true)

{

await UploadDocumentationsFromLocalSotrage();

}

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

/// <summary>

/// Publishes selected documentation to the database with a private access

/// </summary>

private async void PublishLocalDocumentation(object sender, RoutedEventArgs e)

{

if (LocalDocumentationListBox.SelectedItem is Documentation selectedDocumentation)

{

try

{

PageHeader.SwitchOnTopLoader();

SetDocumentationPropertiesForPublishing(selectedDocumentation);

List<Document> documents = LocalDocumentations.Find(ld => ld.Documentation

== selectedDocumentation).Documents;

await CloudDocumentationService.InsertDocumentation(selectedDocumentation);

await PublishDocuments(await GetLastDocumentationID(), documents);

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех", "Документация успешно опубликована",

MessageBoxButton.OK);

}

catch (SqlException ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Сначала выберете докуемнтацию для публикации",

MessageBoxButton.OK);

}

}

private async Task PublishDocuments(int documentationID, List<Document> documents)

{

try

{

foreach (Document document in documents)

{

document.DocumentationID = documentationID;

await CloudDocumentsService.InsertDocument(document);

}

}

catch (SqlException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private void SetDocumentationPropertiesForPublishing(Documentation documentation)

{

documentation.Access = Access.Private;

documentation.StorageType = StorageType.Cloud;

}

/// <summary>

/// Gets the ID of a last user's (which is logged in the system) documentation.

/// </summary>

private async Task<int> GetLastDocumentationID()

{

List<Documentation> documentations = (await CloudDocumentationService.GetUserDocumentations(

UserInfo.CurrentUser.ID)).ToList();

return documentations[documentations.Count - 1].ID;

}

private void UploadLocalDocumentationToMainPage(object sender, RoutedEventArgs e)

{

if (!(LocalDocumentationListBox.SelectedItem is Documentation selectedDocumentation))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Вы не выбрали документацию",

MessageBoxButton.OK);

return;

}

LocalDocumentation localDocumentation = LocalDocumentations.Find(

ld => ld.Documentation == selectedDocumentation);

localDocumentation.DocumentationPath = Path.Combine(CurrentFilePath,

localDocumentation.Documentation.Name + ".sdwp");

MainPage.UploadLocalDocumentation(localDocumentation);

CloseAccGrid();

}

private async void SelectLocalDocumentationFolder(object sender, RoutedEventArgs e)

{

string folderPath = FolderDialog.ShowDialog();

FilePathTextBox.Text = CurrentFilePath = folderPath;

if (folderPath != null)

{

LocalDocumentationService.StoragePath = folderPath;

await UploadDocumentationsFromLocalSotrage();

}

}

private async void DeleteLocalDocumentation(object sender, RoutedEventArgs e)

{

if (SDWPMessageBox.ConfirmAction() == MessageBoxResult.Cancel)

return;

try

{

if (!(LocalDocumentationListBox.SelectedItem is Documentation selectedDocumentation))

{

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Вы не выбрали документацию", MessageBoxButton.OK);

return;

}

LocalDocumentationService.DeleteLocalDocumentationFile(LocalDocumentations.Find((

ld => ld.Documentation == selectedDocumentation)));

await UploadDocumentationsFromLocalSotrage();

}

catch (SqlException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

#endregion

}

}

<Page x:Class="SDWP.UserProfilePage"

x:Name="userProfilePage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:SDWP"

VerticalAlignment="Stretch"

HorizontalAlignment="Stretch"

Background="#E1E1E1"

SizeChanged="RepositionElements"

Title="UserProfilePage">

<Page.Resources>

<Style x:Key="userProfilePropertiesName" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="10"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

</Style>

<Style x:Key="userProfileTextBoxEnabled" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="IsEnabled" Value="True"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="userProfileTextBoxDisabled" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="IsEnabled" Value="False"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="#ffbba2"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="userProfilePasswordBoxEnabled" TargetType="PasswordBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="IsEnabled" Value="False"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="PasswordBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="userProfilePasswordBoxDisabled" TargetType="PasswordBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="IsEnabled" Value="False"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="PasswordBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="#ffbba2"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="profileDataHeader" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="17"/>

<Setter Property="Foreground" Value="OrangeRed"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="FontWeight" Value="Bold"/>

</Style>

<Style x:Key="userProfileButton" TargetType="Button">

<Setter Property="OverridesDefaultStyle" Value="True"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="12"/>

<Setter Property="Padding" Value="0"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="150"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="Background" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border CornerRadius="5"

BorderBrush="{StaticResource mainThemeBrush}"

BorderThickness="1"

SnapsToDevicePixels="True"

Background="{TemplateBinding Background}">

<ContentPresenter HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="UserProfileBtnMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="UserProfileBtnMouseLeave"/>

</Style>

<Style x:Key="userProfileTextBlockStyle" TargetType="TextBlock">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

</Style>

</Page.Resources>

<Page.Content>

<Grid x:Name="userProfileGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Visibility="Visible"

Background="#E1E1E1">

<ScrollViewer x:Name="pageScrollViewer"

VerticalScrollBarVisibility="Visible"

Width="{Binding ElementName =userProfilePage, Path=ActualWidth}"

Height="{Binding ElementName=userProfilePage, Path=ActualHeight}">

<Grid x:Name="contentOutterGrid">

<TextBlock x:Name="userGridMainGridHeader"

FontFamily="Arial, Verdana"

FontSize="40"

Margin="20, 30, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Foreground="OrangeRed"

Text="Профиль"/>

<local:PageHeader x:Name="pageHeader"

VerticalAlignment="Top"

Header="Мой профиль"

Width="{Binding ElementName=userProfileGrid, Path=ActualWidth}"/>

<!-- Photo and name grid -->

<Border x:Name="userPhotoBorder"

Margin="30, 140, 0, 0"

Width="210"

Height="210"

VerticalAlignment="Top"

HorizontalAlignment="Left"

CornerRadius="210"

BorderBrush="#ffbba2"

BorderThickness="3">

<Ellipse x:Name="userPhotoEllipse"

Width="200"

Height="200"

MouseEnter="UserPhotoMouseEnter"

MouseLeave="UserPhotoMouseLeave"

MouseDown="UserPhotoMouseDown"

VerticalAlignment="Center"

HorizontalAlignment="Center">

<Ellipse.Fill>

<ImageBrush ImageSource="../Resources/emptyUserPhotoImage.png"

Stretch="UniformToFill"/>

</Ellipse.Fill>

</Ellipse>

</Border>

<Button x:Name="editProfileDataBtn"

Content="Редактировать профиль"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="60, 370, 0, 0"

Click="EditProfileData"

Style="{StaticResource userProfileButton}"/>

<Button x:Name="updateProfileDataBtn"

Content="Обновить профиль"

IsEnabled="False"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Click="StartUpdatingProcessAsync"

Margin="60, 410, 0, 0"

Style="{StaticResource userProfileButton}"

Background="#ffbba2"/>

<Button x:Name="updatePasswordBtn"

Content="Обновить пароль"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Click="UpdatePassword"

Margin="60, 480, 0, 0"

Style="{StaticResource userProfileButton}"/>

<Grid x:Name="userProfileDataGrid"

VerticalAlignment="Top"

Margin="270, 140, 0, 0"

Visibility="Visible"

HorizontalAlignment="Left">

<TextBlock Text="ФИО"

Style="{StaticResource profileDataHeader}"

Margin="22, 0, 0, 0"/>

<TextBlock Text="Имя"

Style="{StaticResource userProfilePropertiesName}"

Margin="22, 25, 0, 0"/>

<TextBox x:Name="nameTextBox"

Text="Name"

Style="{StaticResource userProfileTextBoxDisabled}"

Margin="20, 35, 0, 0"/>

<TextBlock Text="Фамилия"

Style="{StaticResource userProfilePropertiesName}"

Margin="22, 85, 0, 0"/>

<TextBox x:Name="surnameTextBox"

Text="Surname"

Style="{StaticResource userProfileTextBoxDisabled}"

Margin="20, 95, 0, 0"/>

<TextBlock Text="Детали авторизации"

Style="{StaticResource profileDataHeader}"

Margin="22, 150, 0, 0"/>

<TextBlock Text="Логин"

Style="{StaticResource userProfilePropertiesName}"

Margin="22, 175, 0, 0"/>

<TextBox x:Name="loginTextBox"

Text="Login"

Style="{StaticResource userProfileTextBoxDisabled}"

Margin="20, 185, 0, 0"/>

<TextBlock Text="Другая информация"

Style="{StaticResource profileDataHeader}"

Margin="22, 240, 0, 0"/>

<TextBlock Text="Почта"

Style="{StaticResource userProfilePropertiesName}"

Margin="22, 265, 0, 0"/>

<TextBox x:Name="emailTextBox"

Text="Email"

Style="{StaticResource userProfileTextBoxDisabled}"

Margin="20, 275, 0, 0"/>

<TextBlock Text="Дата рождения"

Style="{StaticResource userProfilePropertiesName}"

Margin="22, 330, 0, 0"/>

<TextBox x:Name="birthDateTextBox"

Text="30.05.2000"

Style="{StaticResource userProfileTextBoxDisabled}"

Margin="20, 340, 0, 0"/>

</Grid>

<Grid x:Name="enterCodeGrid"

Height="500"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Background="#E1E1E1"

Visibility="Collapsed"

Margin="0, 140, 0, 0">

<TextBlock Text="Введите код подтверждения"

Style="{StaticResource profileDataHeader}"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="22, 0, 0, 0"/>

<TextBlock x:Name="timeTillCodeExpireTextBlock"

Style="{StaticResource userProfileTextBlockStyle}"

Margin="22, 20, 0, 0"/>

<TextBox x:Name="emailCodeTextBox"

Style="{StaticResource userProfileTextBoxEnabled}"

TextAlignment="Center"

MaxLength="6"

Margin="20, 50, 0, 0"/>

<Button Style="{StaticResource userProfileButton}"

Content="Обновить данные"

Margin="20, 100, 0, 0"

VerticalAlignment="Top"

Click="UpdateRecordAfterEmailConfirmationAsync"

HorizontalAlignment="Center"/>

<TextBlock Text="Отмена"

Style="{StaticResource userProfileTextBlockStyle}"

Margin="20, 140, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Center"

MouseEnter="CancelTextBlockMouseEnter"

MouseLeave="CancelTextBlockMouseLeave"

MouseDown="CloseEnterCodeGrid"/>

</Grid>

</Grid>

</ScrollViewer>

</Grid>

</Page.Content>

</Page>

using System;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Media.Animation;

using Microsoft.Win32;

using System.IO;

using System.Windows.Threading;

using ApplicationLib.Models;

using ApplicationLib.Exceptions;

using ApplicationLib.Interfaces;

using ApplicationLib.Factories;

using SDWP.Factories;

using SDWP.Interfaces;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для UserProfilePage.xaml

/// </summary

public partial class UserProfilePage : Page, IAccountPage

{

#region Services

private IEmailService<UserInfo> EmailService { get; set; }

private IUserService<UserInfo> UserService { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

#endregion

#region Properties

private int CodeID { get; set; } = -1;

public Action CloseAccGrid { get; set; }

private byte[] NewUserPhoto { get; set; } = null;

private bool IsProfileDataEdititing { get; set; } = false;

private PageHeader PageHeader { get; set; }

#endregion

#region Constructors and Utility methods

public UserProfilePage(UserInfo user)

{

InitializeComponent();

InitializeInterfaces();

UploadUserDataToUI();

PageHeader = pageHeader;

PageHeader.OnRefresh = UpdateCommonUserAndRefreshUI;

}

private void InitializeInterfaces()

{

IServiceAbstractFactory serviceFactory = new ServiceAbstractFactory();

ISdwpAbstractFactory sdwpAbstractFactory = new SdwpAbstractFactory();

EmailService = serviceFactory.GetEmailService();

UserService = serviceFactory.GetUserService();

ExceptionHandler = sdwpAbstractFactory.GetExceptionHandler(Dispatcher);

}

private void UploadUserDataToUI()

{

nameTextBox.Text = UserInfo.CurrentUser.Name;

surnameTextBox.Text = UserInfo.CurrentUser.Surname;

loginTextBox.Text = UserInfo.CurrentUser.Login;

birthDateTextBox.Text = UserInfo.CurrentUser.BirthDate.

ToShortDateString();

emailTextBox.Text = UserInfo.CurrentUser.Email;

if (UserInfo.CurrentUser.UserPhoto != null && UserInfo.CurrentUser.UserPhoto.Length > 1)

SetUserPhotoImageImageBrush(UserInfo.CurrentUser.UserPhoto);

else

{

SetUserPhotoImageImageBrush(GetResourceImageBytes(

"pack://application:,,,/Resources/emptyUserPhotoImage.png"));

}

}

#warning add to tables

private byte[] GetResourceImageBytes(string imagePath)

{

byte[] imageBytes;

Uri imageUri = new Uri(imagePath);

BitmapImage bitmapImage = new BitmapImage(imageUri);

using (var ms = new MemoryStream())

{

JpegBitmapEncoder encoder = new JpegBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(bitmapImage));

encoder.Save(ms);

imageBytes = ms.ToArray();

}

return imageBytes;

}

private void SetUserPhotoImageImageBrush(byte[] imageByteArray)

{

try

{

MemoryStream memoryStream = new MemoryStream();

memoryStream.Write(imageByteArray, 0, (int)imageByteArray.Length);

memoryStream.Seek(0, SeekOrigin.Begin);

BitmapImage userPhotoSource = new BitmapImage();

userPhotoSource.BeginInit();

userPhotoSource.StreamSource = memoryStream;

userPhotoSource.EndInit();

userPhotoEllipse.Fill = new ImageBrush(userPhotoSource);

}

catch (ArgumentNullException ex) { ExceptionHandler.HandleWithMessageBox(ex); }

catch (IOException ex) { ExceptionHandler.HandleWithMessageBox(ex); }

catch (Exception ex) { ExceptionHandler.HandleWithMessageBox(ex); }

}

#endregion

#region Grids animations

private void ShowEnterCodeGrid()

{

ChangeEnablePropertyOfUserDataElemenets(false);

enterCodeGrid.Visibility = Visibility.Visible;

enterCodeGrid.Margin = new Thickness(0, -500, 0, 0);

ThicknessAnimation lowerTheGrid = new ThicknessAnimation

{

From = enterCodeGrid.Margin,

To = new Thickness(0, 140, 0, 0),

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

lowerTheGrid.Completed += (sender, e) =>

{

enterCodeGrid.Margin = new Thickness(0, 140, 0, 0);

};

enterCodeGrid.BeginAnimation(MarginProperty, lowerTheGrid);

}

private void HideEnterCodeGrid()

{

ChangeEnablePropertyOfUserDataElemenets(true);

ThicknessAnimation lowerTheGrid = new ThicknessAnimation

{

From = enterCodeGrid.Margin,

To = new Thickness(0, -500, 0, 0),

Duration = TimeSpan.FromMilliseconds(300),

DecelerationRatio = 1,

SpeedRatio = 0.5,

FillBehavior = FillBehavior.Stop

};

lowerTheGrid.Completed += (sender, e) =>

{

enterCodeGrid.Margin = new Thickness(250, -500, 0, 0);

enterCodeGrid.Visibility = Visibility.Collapsed;

};

enterCodeGrid.BeginAnimation(MarginProperty, lowerTheGrid);

}

#endregion

#region Edit profile methods

private void EditProfileData(object sender, RoutedEventArgs e)

{

if (!IsProfileDataEdititing)

{

ChangeEnablePropertyOfUserDataElemenets(true);

ChangeButtonsProperties(true);

IsProfileDataEdititing = true;

}

else

{

UploadUserDataToUI();

ChangeEnablePropertyOfUserDataElemenets(false);

ChangeButtonsProperties(false);

IsProfileDataEdititing = false;

}

}

private void ChangeButtonsProperties(bool status)

{

if (status)

{

editProfileDataBtn.Content = "Отмена";

updateProfileDataBtn.Background = new SolidColorBrush(Color.FromRgb(

255, 69, 0));

}

else

{

editProfileDataBtn.Content = "Редактировать профиль";

updateProfileDataBtn.Background = new SolidColorBrush(Color.FromRgb(

255, 187, 162));

}

}

private void ChangeEnablePropertyOfUserDataElemenets(bool newEnableValue)

{

TextBox[] propertiesTextBoxes = userProfileDataGrid.Children.

OfType<TextBox>().ToArray();

Array.ForEach(propertiesTextBoxes, tb =>

{

tb.IsEnabled = newEnableValue;

if (newEnableValue)

tb.Style = this.Resources["userProfileTextBoxEnabled"] as Style;

else

tb.Style = this.Resources["userProfileTextBoxDisabled"] as Style;

});

if (newEnableValue)

userPhotoBorder.BorderBrush = new SolidColorBrush(Color.FromRgb(

255, 69, 0));

else

userPhotoBorder.BorderBrush = new SolidColorBrush(Color.FromRgb(

255, 187, 162));

updateProfileDataBtn.IsEnabled = newEnableValue;

}

#endregion

#region Update profile data methods

private async void UpdatePassword(object sender, RoutedEventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

await EmailService.SendChangePassLink(UserInfo.CurrentUser);

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Успех",

"Письмо со ссылкой для смены пароля выслано Вам на почту", MessageBoxButton.OK);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async void StartUpdatingProcessAsync(object sender, EventArgs e)

{

PageHeader.SwitchOnTopLoader();

if (CheckIfDataChanged())

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Статус обновления профиля",

"Данные не были изменены", MessageBoxButton.OK);

}

else

{

try

{

UserInfo newUserInfo = CreateNewUserObject();

UserInfo.CheckUserProperties(newUserInfo);

if (UserInfo.CurrentUser.Login != newUserInfo.Login)

await UserService.CheckLogin(newUserInfo.Login);

if (newUserInfo.Email != UserInfo.CurrentUser.Email)

{

await UserService.CheckEmail(newUserInfo.Email);

CodeID = await EmailService.SendCodeEmail(newUserInfo);

ShowEnterCodeGrid();

StartCodeTimer();

}

else

{

await UserService.UpdateRecord(newUserInfo);

OnSuccesfullUpdate();

}

}

catch (NotAppropriateUserParam ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

PageHeader.SwitchOffTheLoader();

}

private void OnSuccesfullUpdate()

{

UpdateCommonUserAndRefreshUI();

Dispatcher.Invoke(() => PageHeader.SwitchOffTheLoader());

Dispatcher.Invoke(() => SDWPMessageBox.ShowSDWPMessageBox(

"Статус обновления профиля", "Профиль успешно обновлен",

MessageBoxButton.OK));

Dispatcher.Invoke(() => HideEnterCodeGrid());

}

private async void UpdateCommonUserAndRefreshUI()

{

UserInfo.CurrentUser = await UserService.GetUserByID(UserInfo.CurrentUser.ID);

Dispatcher.Invoke(() => UploadUserDataToUI());

}

private async void UpdateRecordAfterEmailConfirmationAsync(object sender,

EventArgs e)

{

try

{

PageHeader.SwitchOnTopLoader();

UserInfo newUserInfo = CreateNewUserObject();

if (CodeID != -1 && await EmailService.CheckCode(CodeID, emailCodeTextBox.Text))

{

await UserService.UpdateRecord(newUserInfo);

OnSuccesfullUpdate();

}

else

{

PageHeader.SwitchOffTheLoader();

SDWPMessageBox.ShowSDWPMessageBox("Ошибка подтверждения e-mail",

"Вы ввели неверный код", MessageBoxButton.OK);

}

}

catch (Exception ex)

{

PageHeader.SwitchOffTheLoader();

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private void StartCodeTimer()

{

DispatcherTimer timer = new DispatcherTimer();

TimeSpan currentTimerTime = new TimeSpan(0);

timer.Interval = new TimeSpan(0, 0, 1);

timer.Tick += async (sender, e) =>

{

timeTillCodeExpireTextBlock.Text = "Время до исчезновения кода " +

$"{new TimeSpan(0, 2, 0) - currentTimerTime}";

currentTimerTime += new TimeSpan(0, 0, 1);

if (currentTimerTime.Seconds == 0 && currentTimerTime.Minutes == 2)

{

await DeleteCode();

timeTillCodeExpireTextBlock.Text = "Код недействителен";

timer.Stop();

}

};

timer.Start();

}

private async Task DeleteCode()

{

try

{

await EmailService.DeleteCode(CodeID);

CodeID = -1;

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

}

}

private async void CloseEnterCodeGrid(object sender, MouseButtonEventArgs e)

{

emailCodeTextBox.Text = string.Empty;

await DeleteCode();

PageHeader.SwitchOffTheLoader();

HideEnterCodeGrid();

}

private bool CheckIfDataChanged()

{

return nameTextBox.Text == UserInfo.CurrentUser.Name &&

surnameTextBox.Text == UserInfo.CurrentUser.Surname &&

loginTextBox.Text == UserInfo.CurrentUser.Login &&

emailTextBox.Text == UserInfo.CurrentUser.Email &&

birthDateTextBox.Text == UserInfo.CurrentUser.BirthDate.ToShortDateString()

&& NewUserPhoto == null;

}

private UserInfo CreateNewUserObject()

{

return new UserInfo

{

ID = UserInfo.CurrentUser.ID,

Login = loginTextBox.Text,

Password = UserInfo.CurrentUser.Password,

Name = nameTextBox.Text,

Surname = surnameTextBox.Text,

BirthDate = GetNewUserDateTime(),

Email = emailTextBox.Text,

UserPhoto = NewUserPhoto ?? UserInfo.CurrentUser.UserPhoto

};

}

private DateTime GetNewUserDateTime()

{

string[] dateElements = birthDateTextBox.Text.Split('.');

if (dateElements.Length != 3 || !int.TryParse(dateElements[0], out int day) ||

!int.TryParse(dateElements[1], out int month) ||

!int.TryParse(dateElements[2], out int year) || day < 0 || day > 31 ||

month < 0 || month > 12 || year < 1900)

throw new NotAppropriateUserParam("Дата рождения введена неверно");

return new DateTime(year, month, day);

}

#endregion

#region Event handlers

private void RepositionElements(object sender, SizeChangedEventArgs e)

{

userProfileGrid.Width = this.Width;

pageScrollViewer.Height = this.Height;

contentOutterGrid.Height = SystemParameters.MaximizedPrimaryScreenHeight;

}

private void UserPhotoMouseEnter(object sender, MouseEventArgs e)

{

if (IsProfileDataEdititing)

userPhotoBorder.BorderBrush = new SolidColorBrush(Colors.Orange);

}

private void UserPhotoMouseLeave(object sender, MouseEventArgs e)

{

if (IsProfileDataEdititing)

userPhotoBorder.BorderBrush = new SolidColorBrush(Colors.OrangeRed);

}

/// <summary>

/// Shows a file open dialog to choose a new picture

/// </summary>

private void UserPhotoMouseDown(object sender, MouseButtonEventArgs e)

{

if (IsProfileDataEdititing)

{

string imagePath = string.Empty;

OpenFileDialog chooseImageDialog = new OpenFileDialog

{

Filter = "Фотографии(\*.JPG;\*.PNG)|\*.JPG;\*.PNG",

Multiselect = false,

CheckFileExists = true

};

if (chooseImageDialog.ShowDialog() == true)

imagePath = chooseImageDialog.FileName;

if (imagePath != string.Empty)

{

NewUserPhoto = CreateByteRepresentationOfAnImage(imagePath);

SetUserPhotoImageImageBrush(NewUserPhoto);

}

}

}

private byte[] CreateByteRepresentationOfAnImage(string imagePath)

{

try

{

FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);

byte[] imageByteArr = new byte[fileStream.Length];

fileStream.Read(imageByteArr, 0, (int)fileStream.Length);

fileStream.Close();

return imageByteArr;

}

catch (ArgumentNullException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

return new byte[0];

}

catch (IOException ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

return new byte[0];

}

catch (Exception ex)

{

ExceptionHandler.HandleWithMessageBox(ex);

return new byte[0];

}

}

private void UserProfileBtnMouseEnter(object sender, MouseEventArgs e)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.White);

button.Foreground = new SolidColorBrush(Colors.OrangeRed);

}

private void UserProfileBtnMouseLeave(object sender, MouseEventArgs e)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.OrangeRed);

button.Foreground = new SolidColorBrush(Colors.White);

}

private void CancelTextBlockMouseEnter(object sender, MouseEventArgs e)

{

TextBlock cancelTextBlock = sender as TextBlock;

cancelTextBlock.TextDecorations = TextDecorations.Underline;

}

private void CancelTextBlockMouseLeave(object sender, MouseEventArgs e)

{

TextBlock cancelTextBlock = sender as TextBlock;

cancelTextBlock.TextDecorations = null;

}

#endregion

}

}

<Window x:Class="SDWP.MainWindow"

x:Name="authWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SDWP"

mc:Ignorable="d"

Title="SDWP.Authorization"

Height="500"

Width="800"

MinHeight="500"

MinWidth="800"

MaxHeight="500"

MaxWidth="800">

<Window.Resources>

<Style x:Key="authorizationPasswordBoxDarkStyle" TargetType="PasswordBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="White"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="PasswordBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Background="#323232"

Width="200"

Height="30"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="authorizationPasswordBoxLightStyle" TargetType="PasswordBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="PasswordBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="authorizationTextBoxLightStyle" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="200"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="dayMonthTextBoxLightStyle" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="40"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="40"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="yearTextBoxLightStyle" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="80"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="Gray"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Width="80"

Height="30"

VerticalScrollBarVisibility="Disabled"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="authorizationTextBoxDarkStyle" TargetType="TextBox">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="200"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Padding" Value="0, 7, 0, 0"/>

<Setter Property="CaretBrush" Value="White"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="TextBox">

<Border BorderThickness="0, 0, 0, 1.5"

BorderBrush="{StaticResource mainThemeBrush}"

CornerRadius="0"

SnapsToDevicePixels="True">

<ScrollViewer Name="PART\_ContentHost"

Background="#323232"

Width="200"

Height="30"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="authorizationTextBlockStyle" TargetType="TextBlock">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="Foreground" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

</Style>

<Style x:Key="headerTextBlockStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Garamond"/>

<Setter Property="FontSize" Value="55"/>

<Setter Property="Language" Value="English"/>

<Setter Property="Foreground" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="FontWeight" Value="500"/>

</Style>

<Style x:Key="headerDescripitonTextBlockStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Garamond"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Language" Value="English"/>

<Setter Property="Foreground" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

</Style>

<Style x:Key="welcomeWindowBtnStyle" TargetType="Button">

<Setter Property="OverridesDefaultStyle" Value="True"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="12"/>

<Setter Property="Padding" Value="0"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="Width" Value="150"/>

<Setter Property="Height" Value="30"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="Background" Value="{StaticResource mainThemeBrush}"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border CornerRadius="5"

BorderBrush="{StaticResource mainThemeBrush}"

BorderThickness="1"

SnapsToDevicePixels="True"

Background="{TemplateBinding Background}">

<ContentPresenter HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<EventSetter Event="MouseEnter" Handler="WelcomeBtnMouseEnter"/>

<EventSetter Event="MouseLeave" Handler="WelcomeBtnMouseLeave"/>

</Style>

<Style x:Key="appDescriptionTextBlockStyle" TargetType="TextBlock">

<Setter Property="OverridesDefaultStyle" Value="False"/>

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Top"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="#AAAAAA"/>

<Setter Property="Width" Value="300"/>

<Setter Property="Height" Value="200"/>

<Setter Property="Background" Value="White"/>

<Setter Property="TextWrapping" Value="Wrap"/>

</Style>

<Style x:Key="hintTextBlockHeaderStyle" TargetType="TextBlock">

<Setter Property="FontFamily" Value="Arial, Verdana"/>

<Setter Property="FontSize" Value="15"/>

<Setter Property="Foreground" Value="Gray"/>

<Setter Property="Width" Value="150"/>

<Setter Property="Height" Value="20"/>

</Style>

<ThicknessAnimationUsingKeyFrames x:Key="leftLoaderThirdRectAnimation"

AccelerationRatio="0"

RepeatBehavior="Forever"

SpeedRatio="1.1">

<EasingThicknessKeyFrame KeyTime="0:0:0"

Value="30, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.5"

Value="190, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1"

Value="190, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.2"

Value="190, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.7"

Value="30, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:2.2"

Value="30, 0, 0, 0"/>

</ThicknessAnimationUsingKeyFrames>

<ThicknessAnimationUsingKeyFrames x:Key="leftLoaderSecondRectAnimation"

AccelerationRatio="0"

RepeatBehavior="Forever"

SpeedRatio="1.1">

<EasingThicknessKeyFrame KeyTime="0:0:0.0"

Value="15, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.1"

Value="15, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.6"

Value="175, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1"

Value="175, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.1"

Value="175, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.6"

Value="15, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:2.2"

Value="15, 0, 0, 0"/>

</ThicknessAnimationUsingKeyFrames>

<ThicknessAnimationUsingKeyFrames x:Key="leftLoaderFirstRectAnimation"

AccelerationRatio="0"

RepeatBehavior="Forever"

SpeedRatio="1.1">

<EasingThicknessKeyFrame KeyTime="0:0:0.0"

Value="0, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.2"

Value="0, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:0.7"

Value="160, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1"

Value="160, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:1.5"

Value="0, 0, 0, 0"/>

<EasingThicknessKeyFrame KeyTime="0:0:2.2"

Value="0, 0, 0, 0"/>

</ThicknessAnimationUsingKeyFrames>

</Window.Resources>

<Grid x:Name="backgroundGrid"

Background="White">

<!--grid where all authorization functuionality is placed-->

<Grid x:Name="authorizationGrid"

Background="#323232"

HorizontalAlignment="Left"

VerticalAlignment="Top"

Width="400"

Height="500">

<TextBlock x:Name="headerTextBlock"

Text="SDWP"

Margin="0, 20, 0, 0"

Style="{StaticResource headerTextBlockStyle}"/>

<TextBlock x:Name="headerDescriptionTextBlock"

Text="software developer work place"

Margin="0, 70, 0, 0"

Style="{StaticResource headerDescripitonTextBlockStyle}"/>

<!--Left loader-->

<Grid x:Name="leftLoaderGrid"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="200"

Visibility="Collapsed"

Margin="0, 100, 0, 0"

Height="10">

<Rectangle x:Name="loaderRect1"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="0, 0, 0, 0"/>

<Rectangle x:Name="loaderRect2"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="15, 0, 0, 0"/>

<Rectangle x:Name="loaderRect3"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="30, 0, 0, 0"/>

</Grid>

<Grid x:Name="backgroundLoginGrid"

Width="300"

Height="300"

Margin="0, 100, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Center">

<TextBlock x:Name="loginTextBlock"

Text="Введите логин: "

Foreground="White"

HorizontalAlignment="Left"

Style="{StaticResource authorizationTextBlockStyle}"

Margin="20, 30, 0, 0"/>

<TextBox x:Name="loginTextBox"

Margin="20, 50, 0, 0"

HorizontalAlignment="Left"

Style="{StaticResource authorizationTextBoxDarkStyle}"/>

<TextBlock x:Name="passwordTextBlock"

Margin="20, 90, 0, 0"

HorizontalAlignment="Left"

Foreground="White"

Text="Введите пароль: "

Style="{StaticResource authorizationTextBlockStyle}"/>

<PasswordBox x:Name="passwordTextBox"

Margin="20, 110, 0, 0"

HorizontalAlignment="Left"

Style="{StaticResource authorizationPasswordBoxDarkStyle}"/>

<TextBlock x:Name="forgotPassTextBlock"

Text="Забыли пароль?"

Margin="20, 160, 0, 0"

MouseEnter="ForgotPassTextBlockMouseEnter"

MouseLeave="ForgotPassTextBlockMouseLeave"

HorizontalAlignment="Left"

Foreground="White"

MouseDown="ForgotPassTextBlockClick"

Style="{StaticResource authorizationTextBlockStyle}"/>

<!--Login button-->

<Button x:Name="loginBtn"

Content="Войти"

Margin="20, 180, 0, 0"

Width="100"

Height="40"

IsDefault="True"

MouseEnter="WelcomeBtnMouseEnter"

MouseLeave="WelcomeBtnMouseLeave"

Click="LoginBtnMouseClick"

Style="{StaticResource welcomeWindowBtnStyle}"/>

<TextBlock x:Name="noAccTextBlock"

Text="У вас не аккаунта?"

HorizontalAlignment="Left"

Margin="20, 230, 0, 0"

Foreground="White"

Style="{StaticResource authorizationTextBlockStyle}"/>

<!--Create account button-->

<Button x:Name="registrationBtn"

Content="Создать аккаунт"

Margin="20, 250, 0, 0"

MouseLeave="WelcomeBtnMouseLeave"

MouseEnter="WelcomeBtnMouseEnter"

Click="OpenRegistrationGrid"

Style="{StaticResource welcomeWindowBtnStyle}"/>

</Grid>

</Grid>

<Rectangle VerticalAlignment="Top"

HorizontalAlignment="Right"

Fill="White"

Width="400"

Height="500"/>

<Grid x:Name="applicationDescriptionGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Width="400"

Height="500"

Visibility="Visible"

Background="White">

<TextBlock x:Name="appDescriptionHeaderTextBlock"

Text="Об этом"

Margin="0, 20, 0, 0"

Style="{StaticResource headerTextBlockStyle}"/>

<TextBlock x:Name="appDescriptionSubbHeaderTextBlock"

Text="приложении"

Margin="0, 70, 0, 0"

Style="{StaticResource headerDescripitonTextBlockStyle}"/>

<TextBlock x:Name="applicationDescriptionTextBlock"

Text="SDWP - это программа для упрощения создания документации к программам. Я надеюсь, что эта программа поможет

Вам в написании документации. Если у Вас есть вопросы, предложения или замечания просьба отправлять их мне на почту: aerooneQ@yandex.ru"

Margin="0, 130, 0, 0"

Style="{StaticResource appDescriptionTextBlockStyle}"/>

<TextBlock x:Name="aeroTextBlock"

Text="©Aero"

FontSize="20"

Margin="0, 260, 0, 0"

Style="{StaticResource appDescriptionTextBlockStyle}"/>

</Grid>

<!--Remind pass grid-->

<Grid x:Name="remindPassGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Width="400"

Height="500"

Visibility="Hidden"

Background="White">

<TextBlock x:Name="remindPassHeader"

Text="Восстановить"

Margin="0, 20, 0, 0"

Style="{StaticResource headerTextBlockStyle}"/>

<TextBlock Text="пароль "

Margin="0, 70, 0, 0"

Style="{StaticResource headerDescripitonTextBlockStyle}"/>

<Grid x:Name="remindPassFormGrid"

Background="White"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="300"

Height="300"

Margin="0, 100, 0, 0">

<Grid x:Name="rightRemindPassLoaderGrid"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="200"

Visibility="Collapsed"

Margin="0, 0, 0, 0"

Height="10">

<Rectangle x:Name="rightRemindPassLoaderRect1"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="0, 0, 0, 0"/>

<Rectangle x:Name="rightRemindPassLoaderRect2"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="15, 0, 0, 0"/>

<Rectangle x:Name="rightRemindPassLoaderRect3"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="30, 0, 0, 0"/>

</Grid>

<TextBlock x:Name="remindPassEnterLoginTextBlock"

Text="Введите логин: "

Margin="0, 40, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="remindPassEnterLoginTextBox"

Margin="0, 60, 0, 0"

Width="200"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<TextBlock x:Name="remindPassEnterEmailTextBlock"

Text="Введите email: "

Margin="0, 100, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="remindPassEnterEmailTextBox"

Margin="0, 120, 0, 0"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<Button x:Name="remindPassBtn"

Content="Отправить email"

HorizontalAlignment="Center"

MouseEnter="WelcomeBtnMouseEnter"

MouseLeave="WelcomeBtnMouseLeave"

Width="100"

Height="40"

Margin="0, 180, 0, 0"

Click="SendNewPassToUser"

Style="{StaticResource welcomeWindowBtnStyle}"/>

<TextBlock x:Name="closeRemindPassGridTextBlock"

Text="Закрыть"

MouseEnter="ForgotPassTextBlockMouseEnter"

MouseLeave="ForgotPassTextBlockMouseLeave"

VerticalAlignment="Bottom"

HorizontalAlignment="Center"

Margin="0, 0, 0, 20"

MouseDown="CloseTheRemindPassGrid"

Style="{StaticResource authorizationTextBlockStyle}"/>

</Grid>

</Grid>

<!--Registration grid-->

<Grid x:Name="createAnAccountGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Width="400"

Height="500"

Visibility="Collapsed"

Background="White">

<TextBlock x:Name="createAccHeader"

Text="Создать"

Margin="0, 20, 0, 0"

Style="{StaticResource headerTextBlockStyle}"/>

<Grid x:Name="createAnAccountForm"

Background="White"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Height="300"

Width="300"

Margin="0, 100, 0, 0">

<!--Right loader-->

<Grid x:Name="rightCreateAccLoaderGrid"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Width="200"

Visibility="Collapsed"

Margin="0, 0, 0, 0"

Height="10">

<Rectangle x:Name="rightCreateAccLoaderRect1"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="0, 0, 0, 0"/>

<Rectangle x:Name="rightCreateAccLoaderRect2"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="15, 0, 0, 0"/>

<Rectangle x:Name="rightCreateAccLoaderRect3"

Fill="{StaticResource mainThemeBrush}"

Width="10"

Height="10"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="30, 0, 0, 0"/>

</Grid>

<ScrollViewer x:Name="createAccScroll"

Width="300"

Height="270"

Margin="0, 30, 0, 0"

VerticalAlignment="Top"

HorizontalAlignment="Left"

HorizontalScrollBarVisibility="Hidden"

VerticalScrollBarVisibility="Visible"

Background="#DDDDDD">

<Grid x:Name="registrationElementsGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="300"

Background="White">

<!--Login-->

<TextBlock x:Name="regLoginTextBlock"

Text="Введите логин: "

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 0, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="regLoginTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 20, 0, 0"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<Image x:Name="loginQMarkPng"

Source="../Resources/questionMarkStatic.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="QuestionMarkMouseEnter"

Height="15"

Width="15"

Margin="200, 0, 0, 0"/>

<Image x:Name="loginQMarkActivePng"

Source="../Resources/questionMarkActive.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseLeave="QuestionMarkMouseLeave"

Height="15"

Width="15"

Visibility="Collapsed"

Margin="200, 0, 0, 0"/>

<!--Password-->

<TextBlock x:Name="regPassTextBlock"

Text="Введите пароль:"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 60, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<PasswordBox x:Name="regPassTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 80, 0, 0"

Style="{StaticResource authorizationPasswordBoxLightStyle}"/>

<Image x:Name="passwordQMarkPng"

Source="../Resources/questionMarkStatic.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="QuestionMarkMouseEnter"

Height="15"

Width="15"

Margin="200, 60, 0, 0"/>

<Image x:Name="passwordQMarkActivePng"

Source="../Resources/questionMarkActive.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

Visibility="Collapsed"

HorizontalAlignment="Left"

MouseLeave="QuestionMarkMouseLeave"

Height="15"

Width="15"

Margin="200, 60, 0, 0"/>

<!--Name-->

<TextBlock x:Name="regNameTextBlock"

Text="Введите имя:"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 120, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="regNameTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 140, 0, 0"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<Image x:Name="nameQMarkPng"

Source="../Resources/questionMarkStatic.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="QuestionMarkMouseEnter"

Height="15"

Width="15"

Margin="200, 120, 0, 0"/>

<Image x:Name="nameQMarkActivePng"

Source="../Resources/questionMarkActive.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

Visibility="Collapsed"

HorizontalAlignment="Left"

MouseLeave="QuestionMarkMouseLeave"

Height="15"

Width="15"

Margin="200, 120, 0, 0"/>

<!--Surname-->

<TextBlock x:Name="regSurnameTextBlock"

Text="Введите фамилию: "

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 180, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="regSurnameTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 200, 0, 0"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<Image x:Name="surNQMarkPng"

Source="../Resources/questionMarkStatic.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="QuestionMarkMouseEnter"

Height="15"

Width="15"

Margin="200, 180, 0, 0"/>

<Image x:Name="surNQMarkActivePng"

Source="../Resources/questionMarkActive.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

Visibility="Collapsed"

HorizontalAlignment="Left"

MouseLeave="QuestionMarkMouseLeave"

Height="15"

Width="15"

Margin="200, 180, 0, 0"/>

<!--Birth date-->

<TextBlock x:Name="regBirthDateTextBlock"

Text="Введите дату рождения:"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 240, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="regDayOfBirthTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 260, 0, 0"

MaxLength="2"

Width="40"

Style="{StaticResource dayMonthTextBoxLightStyle}"/>

<TextBox x:Name="regMonthOfBirthTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="80, 260, 0, 0"

MaxLength="2"

Width="40"

Style="{StaticResource dayMonthTextBoxLightStyle}"/>

<TextBox x:Name="regYearOfBirthTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="140, 260, 0, 0"

Width="80"

MaxLength="4"

Style="{StaticResource yearTextBoxLightStyle}"/>

<Image x:Name="bDayQMarkPng"

Source="../Resources/questionMarkStatic.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="QuestionMarkMouseEnter"

Height="15"

Width="15"

Margin="200, 240, 0, 0"/>

<Image x:Name="bDayQMarkActivePng"

Source="../Resources/questionMarkActive.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseLeave="QuestionMarkMouseLeave"

Visibility="Collapsed"

Height="15"

Width="15"

Margin="200, 240, 0, 0"/>

<!--Email-->

<TextBlock x:Name="regEmailTextBlock"

Text="Введите email:"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 300, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="regEmailTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="20, 320, 0, 0"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<Image x:Name="emailQMarkPng"

Source="../Resources/questionMarkStatic.png"

MouseDown="ShowHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="QuestionMarkMouseEnter"

Height="15"

Width="15"

Margin="200, 300, 0, 0"/>

<Image x:Name="emailQMarkActivePng"

Source="../Resources/questionMarkActive.png"

MouseDown="ShowHintGrid"

Visibility="Collapsed"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseLeave="QuestionMarkMouseLeave"

Height="15"

Width="15"

Margin="200, 300, 0, 0"/>

<Button x:Name="closeRegistrationGridTextBlock"

Content="Отмена"

VerticalAlignment="Top"

HorizontalAlignment="Left"

MouseEnter="WelcomeBtnMouseEnter"

MouseLeave="WelcomeBtnMouseLeave"

Width="80"

Margin="180, 380, 0, 0"

Click="CloseTheRegistrationGrid"

Style="{StaticResource welcomeWindowBtnStyle}"/>

<Button x:Name="createAccBtn"

VerticalAlignment="Top"

Content="Далее"

HorizontalAlignment="Left"

MouseEnter="WelcomeBtnMouseEnter"

MouseLeave="WelcomeBtnMouseLeave"

Margin="20, 380, 0, 20"

Click="GoToEnterTheCodeGrid"

Style="{StaticResource welcomeWindowBtnStyle}"/>

</Grid>

</ScrollViewer>

<Grid x:Name="enterTheEmailCodeGrid"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="300"

Height="270"

Margin="0, 30, 0, 0"

Visibility="Collapsed"

Background="White">

<TextBlock x:Name="enterTheCodeTextBlock"

Text="Введите код: "

VerticalAlignment="Top"

HorizontalAlignment="Center"

Margin="0, 0, 0, 0"

FontSize="16"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBlock x:Name="explanationOfTheEmailTextBlock"

Text="Мы отправили Вам письмо, в котором содержиться код, который нужно ввести ниже"

TextWrapping="Wrap"

Width="200"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Margin="0, 40, 0, 0"

Style="{StaticResource authorizationTextBlockStyle}"/>

<TextBox x:Name="codeTextBox"

VerticalAlignment="Top"

HorizontalAlignment="Center"

Margin="0, 100, 0, 0"

Style="{StaticResource authorizationTextBoxLightStyle}"/>

<Button x:Name="createNewAcc"

Content="Создать аккаунт"

VerticalAlignment="Top"

Width="140"

HorizontalAlignment="Left"

Margin="20, 160, 0, 0"

Click="CreateNewAccount"

Style="{StaticResource welcomeWindowBtnStyle}"/>

<Button x:Name="cancelCreationOperation"

Content="Отмена"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Width="90"

Margin="0, 160, 20, 0"

Click="CloseTheRegistrationGrid"

Style="{StaticResource welcomeWindowBtnStyle}"/>

</Grid>

</Grid>

<Grid x:Name="registrationHintGrid"

VerticalAlignment="Bottom"

HorizontalAlignment="Left"

Height="125"

Width="400"

Visibility="Hidden"

Background="Wheat">

<TextBlock x:Name="inputDataNameTextBlock"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="30, 10, 0, 0"

Style="{StaticResource hintTextBlockHeaderStyle}"/>

<TextBlock x:Name="inputDataDescriptionTextBlock"

TextWrapping="Wrap"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Width="300"

Height="45"

Foreground="Black"

FontSize="13"

Style="{StaticResource hintTextBlockHeaderStyle}"

Margin="30, 35, 0, 0"/>

<Image x:Name="closeHintIcon"

Source="../Resources/closeHintIcon.png"

MouseDown="CloseTheHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

MouseEnter="CloseHintMouseEnter"

Width="20"

Height="20"

Margin="0, 10, 10, 0"/>

<Image x:Name="closeHintActiveIcon"

Source="../Resources/closeHintIconActive.png"

MouseLeave="CloseHintMouseLeave"

MouseDown="CloseTheHintGrid"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Visibility="Collapsed"

Width="20"

Height="20"

Margin="0, 10, 10, 0"/>

</Grid>

</Grid>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data.SqlClient;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

using System.Data.SqlTypes;

using System.Net.Mail;

using ApplicationLib.Models;

using ApplicationLib.Exceptions;

using ApplicationLib.Services;

using ApplicationLib.Interfaces;

using ApplicationLib.Factories;

using SDWP.Factories;

using SDWP.Exceptions;

using SDWP.Interfaces;

namespace SDWP

{

/// <summary>

/// Логика взаимодействия для WelcomeWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

#region Services

private IEmailService<UserInfo> EmailService { get; set; }

private IUserService<UserInfo> UserService { get; set; }

private IExceptionHandler ExceptionHandler { get; set; }

#endregion

#region Properties

private UserInfo NewUser { get; set; }

private int CodeID { get; set; } = -1;

#endregion

public MainWindow()

{

InitializeComponent();

InitializeInterfaces();

loginTextBox.Focus();

//initialize the list of all grids

WelcomePageRightGridAnimations.RightGridsList = new List<Grid>

{

applicationDescriptionGrid,

remindPassGrid,

createAnAccountGrid

};

}

private void InitializeInterfaces()

{

IServiceAbstractFactory serviceFactory = new ServiceAbstractFactory();

ISdwpAbstractFactory sdwpAbstractFactory = new SdwpAbstractFactory();

EmailService = serviceFactory.GetEmailService();

UserService = serviceFactory.GetUserService();

ExceptionHandler = sdwpAbstractFactory.GetExceptionHandler(Dispatcher);

}

#region Registration process

/// <summary>

/// Checks the input data in the registration form, and if everything is alright

/// redirects user to the enter code grid

/// </summary>

private async void GoToEnterTheCodeGrid(object sender, RoutedEventArgs e)

{

try

{

SwitchOnTheLoader(rightCreateAccLoaderGrid);

UserInfo newUser = CreateNewUser();

bool checkingResult = await CheckUserInputDataAsync(newUser);

if (checkingResult)

SendEmailAndGoToCodeGridAsync(newUser);

}

catch (InvalidCastException ex)

{

HandleAccCreationException(ex);

}

catch (Exception ex)

{

HandleAccCreationException(ex);

}

}

/// <summary>

/// Checks all properties of user object and

/// availability of login and email

/// </summary>

private async Task<bool> CheckUserInputDataAsync(UserInfo newUser)

{

try

{

UserInfo.CheckUserProperties(newUser);

await UserService.CheckLogin(newUser.Login);

await UserService.CheckEmail(newUser.Email);

return true;

}

catch (NotAppropriateUserParam ex)

{

HandleAccCreationException(ex);

return false;

}

catch (SqlException ex)

{

HandleAccCreationException(ex);

return false;

}

catch (Exception ex)

{

HandleAccCreationException(ex);

return false;

}

}

/// <summary>

/// Sends an email with a code to confirm this email and then redirects

/// user to the 'enter code grid'

/// </summary>

private async void SendEmailAndGoToCodeGridAsync(UserInfo newUser)

{

try

{

NewUser = newUser;

CodeID = await EmailService.SendCodeEmail(NewUser);

}

catch (InvalidOperationException ex)

{

HandleAccCreationException(ex);

}

catch (ArgumentNullException ex)

{

HandleAccCreationException(ex);

}

catch (Exception ex)

{

HandleAccCreationException(ex);

}

finally

{

SwitchOffTheLoader(rightCreateAccLoaderGrid);

enterTheEmailCodeGrid.Visibility = Visibility.Visible;

}

}

/// <summary>

/// Checks the code entered by user and if it is correct

/// creates new account into the database

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private async void CreateNewAccount(object sender, RoutedEventArgs e)

{

try

{

(sender as Button).IsEnabled = false;

string code = codeTextBox.Text;

await CheckCodeAndCreateAccAsync(code);

}

catch (NotAppropriateUserParam ex)

{

HandleAccCreationException(ex);

}

catch (InvalidCastException ex)

{

HandleAccCreationException(ex);

}

catch (Exception ex)

{

HandleAccCreationException(ex);

}

finally

{

(sender as Button).IsEnabled = true;

}

}

private async Task CheckCodeAndCreateAccAsync(string code)

{

SwitchOnTheLoader(rightCreateAccLoaderGrid);

if (await EmailService.CheckCode(CodeID, code))

{

await UserService.CreateNewAccountAsync(NewUser);

SwitchOffTheLoader(rightCreateAccLoaderGrid);

SDWPMessageBox.ShowSDWPMessageBox("Статус создание аккаунта",

"Аккаунт был успешно создан", MessageBoxButton.OK);

enterTheEmailCodeGrid.Visibility = Visibility.Collapsed;

WelcomePageRightGridAnimations.HideTheGrid(createAnAccountGrid);

}

else

{

SwitchOffTheLoader(rightCreateAccLoaderGrid);

SDWPMessageBox.ShowSDWPMessageBox("Ошибка", "Вы ввели неверный код", MessageBoxButton.OK);

}

}

private async Task DeleteCode()

{

await EmailService.DeleteCode(CodeID);

CodeID = -1;

}

/// <summary>

/// Tries to create a new UserInfo object based on the data given by user

/// </summary>

private UserInfo CreateNewUser()

{

return new UserInfo

{

Login = regLoginTextBox.Text,

Password = regPassTextBox.Password.GetHashCode().ToString(),

Name = regNameTextBox.Text,

Surname = regSurnameTextBox.Text,

BirthDate = GetUserBirthDate(),

Email = regEmailTextBox.Text

};

}

/// <summary>

/// Creates a DateTime object with the input year, month and day,

/// which are entered by user

/// </summary>

private DateTime GetUserBirthDate()

{

try

{

int year = int.Parse(regYearOfBirthTextBox.Text);

int month = int.Parse(regMonthOfBirthTextBox.Text);

int day = int.Parse(regDayOfBirthTextBox.Text);

if (year >= DateTime.Now.Year || year <= 1900)

throw new NotAppropriateUserParam("Неверно введен год рождения. ");

if (month <= 0 || month > 12)

throw new NotAppropriateUserParam("Неверно введен месяц рождения");

if (day <= 0 || day > DateTime.DaysInMonth(year, month))

throw new NotAppropriateUserParam("Неверно введен день рождения");

return new DateTime(year, month, day);

}

catch

{

throw new NotAppropriateUserParam("Неверно введенна дата рождения");

}

}

/// <summary>

/// Handles any exception which occures in the process of account creation

/// </summary>

/// <param name="ex"></param>

/// <returns></returns>

private bool HandleAccCreationException(Exception ex)

{

Dispatcher.Invoke(() => SwitchOffTheLoader(rightCreateAccLoaderGrid));

ExceptionHandler.HandleWithMessageBox(ex);

return false;

}

#endregion

#region Autharization process

/// <summary>

/// Creates the LoginData object basing on the input data and

/// initializes the process of authorization

/// </summary>

private void LoginBtnMouseClick(object sender, EventArgs eArgs)

{

try

{

(sender as Button).IsEnabled = false;

LoginData loginData = new LoginData

{

Login = loginTextBox.Text,

Password = passwordTextBox.Password

};

TryToLoginAsync(loginData);

}

finally

{

(sender as Button).IsEnabled = true;

}

}

/// <summary>

/// Handles every exception which can arise in the login process

/// </summary>

/// <returns>

/// An empty UserInfo object

/// </returns>

private UserInfo HandleExceptionAndReturnEmptyUser(Exception ex)

{

Dispatcher.Invoke(() => SwitchOffTheLoader(leftLoaderGrid));

ExceptionHandler.HandleWithMessageBox(ex);

return null;

}

/// <summary>

/// Tries to login with the given login data

/// </summary>

/// <returns>

/// The user object if the input data is correct,

/// the empty user object otherwise

/// </returns>

private async Task<UserInfo> LoginAndReturnUserObj(LoginData loginData)

{

try

{

return await UserService.AuthorizeUserAsync(loginData);

}

catch (UserNotFoundException ex)

{

return HandleExceptionAndReturnEmptyUser(ex);

}

catch (SqlException ex)

{

return HandleExceptionAndReturnEmptyUser(ex);

}

catch (Exception ex)

{

return HandleExceptionAndReturnEmptyUser(ex);

}

}

/// <summary>

/// Tries to login into the system with the given input login data

/// </summary>

private async void TryToLoginAsync(LoginData loginData)

{

SwitchOnTheLoader(leftLoaderGrid);

UserInfo user = await LoginAndReturnUserObj(loginData);

if (user != null)

{

SwitchOffTheLoader(leftLoaderGrid);

Hide();

UserInfo.CurrentUser = user;

SDWPMainWindow mainWindow = new SDWPMainWindow(UserInfo.CurrentUser);

mainWindow.Show();

}

}

#endregion

#region Hint grid processes

/// <summary>

/// Shows the hint grid when the question mark is tapped

/// </summary>

private void ShowHintGrid(object sender, EventArgs eArgs)

{

Image pressedImage = sender as Image;

WelcomePageBottomGridAnimation.ShowTheHintGrid(registrationHintGrid);

inputDataNameTextBlock.Text = GetInputDataName(pressedImage.Name);

inputDataDescriptionTextBlock.Text = GetInputDataDescription(pressedImage.Name);

}

private void CloseTheHintGrid(object sender, EventArgs eArgs)

{

WelcomePageBottomGridAnimation.CloseTheHintGrid(registrationHintGrid);

}

private string GetInputDataName(string imageName)

{

if (imageName.IndexOf("login") > -1)

return "Логин";

if (imageName.IndexOf("password") > -1)

return "Пароль";

if (imageName.IndexOf("name") > -1)

return "Имя";

if (imageName.IndexOf("surN") > -1)

return "Фамилия";

if (imageName.IndexOf("bDay") > -1)

return "Дата рождения";

if (imageName.IndexOf("email") > -1)

return "E-mail";

return string.Empty;

}

private string GetInputDataDescription(string imageName)

{

if (imageName.IndexOf("login") > -1)

return "Длина логина должна быть больше 6 и меньше 200 символов, логин должен содержать только " +

"буквы английского алфавита или цифры.";

if (imageName.IndexOf("password") > -1)

return "Пароль состоит из любых символов, пароль должен быть состоять минимум из 8 символов.";

if (imageName.IndexOf("name") > -1)

return "Длина имени должна быть больше одного.";

if (imageName.IndexOf("surN") > -1)

return "Длина фамилии должна быть больше двух.";

if (imageName.IndexOf("bDay") > -1)

return "В первое поле для ввода введите день рождения, во второе - месяц, в третье - год.";

if (imageName.IndexOf("email") > -1)

return "Используйте стандартные правила написания адреса электронной почты";

return string.Empty;

}

#endregion

#region Welcome button mouse enter/leave events

/// <summary>

/// Makes the background white and the text the color of main theme

/// </summary>

private void WelcomeBtnMouseEnter(object sender, EventArgs eArgs)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.White);

button.Foreground = new SolidColorBrush(Colors.OrangeRed);

}

/// <summary>

/// Makes the background color the color of main theme and text white

/// </summary>

private void WelcomeBtnMouseLeave(object sender, EventArgs eArgs)

{

Button button = sender as Button;

button.Background = new SolidColorBrush(Colors.OrangeRed);

button.Foreground = new SolidColorBrush(Colors.White);

}

#endregion

#region Loader opertaions

private void SwitchOnTheLoader(Grid loaderGrid)

{

loaderGrid.Visibility = Visibility.Visible;

Rectangle rect3 = loaderGrid.Children[2] as Rectangle;

Rectangle rect2 = loaderGrid.Children[1] as Rectangle;

Rectangle rect1 = loaderGrid.Children[0] as Rectangle;

rect3.BeginAnimation(MarginProperty,

(ThicknessAnimationUsingKeyFrames)authWindow.Resources["leftLoaderThirdRectAnimation"]);

rect2.BeginAnimation(MarginProperty,

(ThicknessAnimationUsingKeyFrames)authWindow.Resources["leftLoaderSecondRectAnimation"]);

rect1.BeginAnimation(MarginProperty,

(ThicknessAnimationUsingKeyFrames)authWindow.Resources["leftLoaderFirstRectAnimation"]);

}

private void SwitchOffTheLoader(Grid loaderGrid)

{

loaderGrid.Visibility = Visibility.Collapsed;

Rectangle rect3 = loaderGrid.Children[2] as Rectangle;

Rectangle rect2 = loaderGrid.Children[1] as Rectangle;

Rectangle rect1 = loaderGrid.Children[0] as Rectangle;

rect3.BeginAnimation(MarginProperty, null);

rect2.BeginAnimation(MarginProperty, null);

rect1.BeginAnimation(MarginProperty, null);

}

#endregion

#region Active text boxes mouse enter/leave events

/// <summary>

/// Underlines the text of a textblock

/// </summary>

private void ForgotPassTextBlockMouseEnter(object sender, MouseEventArgs eArgs)

{

TextBlock textBlock = sender as TextBlock;

textBlock.TextDecorations.Add(TextDecorations.Underline);

}

/// <summary>

/// Makes the text of a textblock non-underlined

/// </summary>

private void ForgotPassTextBlockMouseLeave(object sender, MouseEventArgs eArgs)

{

TextBlock textBlock = sender as TextBlock;

textBlock.TextDecorations.Clear();

}

#endregion

#region Remind pass process

/// <summary>

/// Handles all exceptions which can arise during remind pass process

/// </summary>

private void HandleRemindPassExceptions(Exception ex)

{

Dispatcher.Invoke(() => SwitchOffTheLoader(rightRemindPassLoaderGrid));

ExceptionHandler.HandleWithMessageBox(ex);

}

private async void SendNewPassToUser(object sender, RoutedEventArgs e)

{

try

{

SwitchOnTheLoader(rightRemindPassLoaderGrid);

string login = remindPassEnterLoginTextBox.Text;

string email = remindPassEnterEmailTextBox.Text;

bool remindPassResult = await RemindPassAsync(login, email);

SwitchOffTheLoader(rightRemindPassLoaderGrid);

if (remindPassResult)

{

SDWPMessageBox.ShowSDWPMessageBox("Статус обновления пароля",

"Письмо было выслано Вам на почту", MessageBoxButton.OK);

}

}

catch (InvalidCastException ex)

{

HandleRemindPassExceptions(ex);

}

catch (Exception ex)

{

HandleRemindPassExceptions(ex);

}

}

/// <summary>

/// Initializes the process of pass restoration

/// </summary>

/// <returns>

/// True if everything is OK, false otherwise

/// </returns>

private async Task<bool> RemindPassAsync(string login, string email)

{

try

{

await UserService.RemindPassAsync(login, email);

return true;

}

catch (UserNotFoundException ex)

{

HandleRemindPassExceptions(ex);

return false;

}

catch (SqlException ex)

{

HandleRemindPassExceptions(ex);

return false;

}

catch (Exception ex)

{

HandleRemindPassExceptions(ex);

return false;

}

}

#endregion

private void ForgotPassTextBlockClick(object sender, EventArgs eArgs)

{

WelcomePageRightGridAnimations.ShowTheGrid(remindPassGrid);

}

private void CloseTheRemindPassGrid(object sender, MouseButtonEventArgs e)

{

WelcomePageRightGridAnimations.HideTheGrid(remindPassGrid);

}

private void CloseTheRegistrationGrid(object sender, RoutedEventArgs eArgs)

{

WelcomePageRightGridAnimations.HideTheGrid(createAnAccountGrid);

}

private void OpenRegistrationGrid(object sender, RoutedEventArgs e)

{

enterTheEmailCodeGrid.Visibility = Visibility.Collapsed;

WelcomePageRightGridAnimations.ShowTheGrid(createAnAccountGrid);

}

private void QuestionMarkMouseEnter(object sender, MouseEventArgs e)

{

Image image = sender as Image;

List<Image> imageList = ((Grid)image.Parent).Children.OfType<Image>().ToList();

int imageIndex = imageList.FindIndex(x => x.Name == image.Name);

image.Visibility = Visibility.Hidden;

imageList[imageIndex + 1].Visibility = Visibility.Visible;

}

private void QuestionMarkMouseLeave(object sender, MouseEventArgs e)

{

Image image = sender as Image;

List<Image> imageList = ((Grid)image.Parent).Children.OfType<Image>().ToList();

int imageIndex = imageList.FindIndex(x => x.Name == image.Name);

image.Visibility = Visibility.Hidden;

imageList[imageIndex - 1].Visibility = Visibility.Visible;

}

private void CloseHintMouseEnter(object sender, MouseEventArgs e)

{

Image image = sender as Image;

List<Image> imageList = ((Grid)image.Parent).Children.OfType<Image>().ToList();

int imageIndex = imageList.FindIndex(x => x.Name == image.Name);

image.Visibility = Visibility.Hidden;

imageList[imageIndex + 1].Visibility = Visibility.Visible;

}

private void CloseHintMouseLeave(object sender, MouseEventArgs e)

{

Image image = sender as Image;

List<Image> imageList = ((Grid)image.Parent).Children.OfType<Image>().ToList();

int imageIndex = imageList.FindIndex(x => x.Name == image.Name);

image.Visibility = Visibility.Hidden;

imageList[imageIndex - 1].Visibility = Visibility.Visible;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data.SqlClient;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

using System.Data.SqlTypes;

using System.Net.Mail;

namespace SDWP

{

public class MainGrids

{

public Grid TopOptionsGrid { get; set; }

public Grid MainGrid { get; set; }

public Grid UserAccountGrid { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data.SqlClient;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

using System.Data.SqlTypes;

using System.Net.Mail;

using SDWP.Models;

namespace SDWP

{

/// <summary>

/// Class where all operations connected with size and position of

/// the elements are

/// </summary>

public class Position

{

#region Constants (sizes)

private const double LeftMenuOptionGridWidth = 230;

#endregion

#region Properties

public SDWPMainWindow MainWindow { get; private set; }

#endregion

#region Singleton

private static Position position;

public static Position PositionObj

{

get

{

if (position == null)

position = new Position();

return position;

}

}

#endregion

#region Constrcutors

private Position() { }

#endregion

public void UpdateMainWindow(SDWPMainWindow mainWindow)

{

MainWindow = mainWindow;

MainWindow.Width = SystemParameters.FullPrimaryScreenWidth;

MainWindow.Height = SystemParameters.FullPrimaryScreenHeight;

}

#region Position methods

/// <summary>

/// Sets the position and size of the top and main grid

/// </summary>

public void InitializeMainGrids(MainGrids mainGrids)

{

mainGrids.TopOptionsGrid.Width = MainWindow.Width;

mainGrids.MainGrid.Width = MainWindow.Width;

mainGrids.MainGrid.Height = MainWindow.Height;

}

public void InitializeUserAccountGrids(UserAccountGrids userAccountGrids)

{

userAccountGrids.UserAccGrid.Width = (double)3 / 4 \* MainWindow.Width;

userAccountGrids.UserAccGrid.Height = MainWindow.Height;

userAccountGrids.LeftMenuGrid.Width = LeftMenuOptionGridWidth;

userAccountGrids.LeftMenuGrid.Height = MainWindow.Height;

userAccountGrids.LeftMenuGrid.Margin = new Thickness(0, 0, 0, 0);

userAccountGrids.ContentFrame.Width = (double)3 / 4 \* MainWindow.Width

- LeftMenuOptionGridWidth;

userAccountGrids.ContentFrame.Height = MainWindow.Height;

userAccountGrids.ContentFrame.Margin = new Thickness(

userAccountGrids.LeftMenuGrid.Width, 0, 0, 0);

}

public void InitializeUserAccLeftMenuGrids(UserAccLeftMenuGrids leftMenuGrids)

{

leftMenuGrids.HeaderBottomLineRect.Width = (double)3 / 4 \* MainWindow.Width - 78;

leftMenuGrids.OptionsGrid.Width = LeftMenuOptionGridWidth;

for (int i = 0; i < leftMenuGrids.OptionsGrid.Children.Count; i++)

{

Grid optionGrid = leftMenuGrids.OptionsGrid.Children[i] as Grid;

optionGrid.Width = leftMenuGrids.OptionsGrid.Width;

}

}

#region User Grid pages resize methods

public void InitializeUserAccGridPagesParams(Frame parentFrame)

{

if (parentFrame != null)

(parentFrame.Content as Page).Width = parentFrame.Width;

}

#endregion

#endregion

}

}

* 1. ApplicationLib.dll

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.IO;

using Newtonsoft.Json;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Exceptions;

using System.Runtime.InteropServices;

namespace ApplicationLib.Database

{

public class DocumentationDB : ICloudDocumentationDB<Documentation>

{

#region Properties

private string ApiURL { get; } = "https://aerothedeveloper.ru/sdwpapi/v1.0.0/documentations";

#endregion

public async Task<IEnumerable<Documentation>> GetAllDocumentations()

{

return await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

throw new ServerException();

string responseContent = HTTP.GetResponseContent(httpWebResponse);

IEnumerable<Documentation> documentations =

JsonConvert.DeserializeObject<IEnumerable<Documentation>>(responseContent);

return documentations;

});

}

public async Task<IEnumerable<Documentation>> GetUserDocumentations(int userID)

{

return await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?userID={userID}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

throw new ServerException();

string responseContent = HTTP.GetResponseContent(httpWebResponse);

IEnumerable<Documentation> documentations = JsonConvert.

DeserializeObject<IEnumerable<Documentation>>(responseContent);

return documentations;

});

}

public async Task UpdateDocumentation(Documentation documentation)

{

await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "PUT");

using (var rqStream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(rqStream))

{

sw.Write(JsonConvert.SerializeObject(documentation));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError ||

httpWebResponse.StatusCode == HttpStatusCode.BadRequest)

{

throw new ServerException();

}

});

}

public async Task DeleteDocumentation(int documentationID)

{

await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?documentationID={documentationID}", "DELETE");

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

});

}

public async Task InsertDocumentation(Documentation documentation)

{

await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "POST");

using (var rqStream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(rqStream))

{

sw.Write(JsonConvert.SerializeObject(documentation));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError ||

httpWebResponse.StatusCode == HttpStatusCode.BadRequest)

{

throw new ServerException();

}

});

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Exceptions;

using Newtonsoft.Json;

using System.IO;

namespace ApplicationLib.Database

{

public class DocumentsDB : ICloudDocumentDB<Document>

{

private string ApiURL { get; } = "https://aerothedeveloper.ru/sdwpapi/v1.0.0/documents";

public async Task DeleteDocument(Document document)

{

await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?documentID={document.ID}", "DELETE");

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

throw new ServerException();

});

}

public async Task<IEnumerable<Document>> GetDocumentationDocuments(int documentationID)

{

return await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?documentationID={documentationID}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

throw new ServerException();

string responseContent = HTTP.GetResponseContent(httpWebResponse);

List<Document> documents = JsonConvert.DeserializeObject

<List<Document>>(responseContent);

return documents;

});

}

public async Task<IEnumerable<Document>> GetAllDocuments()

{

return await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

throw new ServerException();

string responseContent = HTTP.GetResponseContent(httpWebResponse);

return JsonConvert.DeserializeObject<IEnumerable<Document>>(responseContent);

});

}

public async Task InsertDocument(Document document)

{

await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "POST");

using (var stream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(stream))

{

sw.Write(JsonConvert.SerializeObject(document));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.BadRequest ||

httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

});

}

public async Task UpdateDocument(Document document)

{

await Task.Run(async () =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "PUT");

using (var stream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(stream))

{

sw.Write(JsonConvert.SerializeObject(document));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)(await httpWebRequest.GetResponseAsync());

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError ||

httpWebResponse.StatusCode == HttpStatusCode.BadRequest)

{

throw new ServerException();

}

});

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Mail;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using ApplicationLib.Exceptions;

namespace ApplicationLib.Database

{

public class EmailDB : IEmailDatabase<UserInfo>

{

private string ApiURL { get; } = "https://aerothedeveloper.ru/sdwpapi/v1.0.0/emailcodes";

public async Task<bool> CheckCode(int codeID, string code)

{

return await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?codeID={codeID}&code={code}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.NoContent)

{

return false;

}

else if (httpWebResponse.StatusCode == HttpStatusCode.OK)

{

return true;

}

else

{

throw new ServerException();

}

});

}

public async Task DeleteCode(int codeID)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?codeID={codeID}", "DELETE");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

});

}

public async Task SendChangePassLink(UserInfo user)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?userID={user.ID}", "PUT");

httpWebRequest.ContentLength = 0;

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

});

}

public async Task<int> SendCodeEmail(UserInfo user)

{

return await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?userID={user.ID}&email={user.Email}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

return int.Parse(HTTP.GetResponseContent(httpWebResponse));

});

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Database

{

public static class HTTP

{

const string Login = "sdwpapimainuser";

const string Password = "dihfsodgoias;pdlvknkdslnvasoifjklfnsldafjsdlfa";

public static string GetResponseContent(HttpWebResponse httpWebResponse)

{

string responseContent;

using (var stream = httpWebResponse.GetResponseStream())

{

using (var sr = new StreamReader(stream))

{

responseContent = sr.ReadToEnd();

};

};

return responseContent;

}

/// <summary>

/// Creates a request with a given api url and sets the authorization headers

/// </summary>

public static HttpWebRequest GetRequest(string apiURL, string method)

{

HttpWebRequest httpWebRequest = WebRequest.CreateHttp(apiURL);

string credentials = Convert.ToBase64String(Encoding.ASCII.GetBytes(Login + ":" + Password));

httpWebRequest.Headers.Add("Authorization", "Basic " + credentials);

httpWebRequest.Method = method;

return httpWebRequest;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using System.Net;

using ApplicationLib.Exceptions;

using Newtonsoft.Json;

using System.IO;

namespace ApplicationLib.Database

{

public class TemplatesDB : ICloudTemplatesDB<Template>

{

private string ApiURL { get; } = "https://aerothedeveloper.ru/sdwpapi/v1.0.0/templates";

public async Task DeleteTemplate(Template template)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?templateID={template.ID}", "DELETE");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError ||

httpWebResponse.StatusCode == HttpStatusCode.BadRequest)

{

throw new ServerException();

}

});

}

public async Task<IEnumerable<Template>> GetAllTemplates()

{

return await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

string responseContent = HTTP.GetResponseContent(httpWebResponse);

IEnumerable<Template> templates = JsonConvert.DeserializeObject<

IEnumerable<Template>>(responseContent);

return templates;

});

}

public async Task<IEnumerable<Template>> GetUserTemplates(int userID)

{

return await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?userID={userID}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

string responseContent = HTTP.GetResponseContent(httpWebResponse);

IEnumerable<Template> templates = JsonConvert.DeserializeObject<

IEnumerable<Template>>(responseContent);

return templates;

});

}

public async Task InsertTemplate(Template template)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "POST");

using (var stream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(stream))

{

sw.Write(JsonConvert.SerializeObject(template));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError ||

httpWebResponse.StatusCode == HttpStatusCode.BadRequest)

{

throw new ServerException();

}

});

}

public async Task UpdateTemplate(Template template)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "PUT");

using (var stream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(stream))

{

sw.Write(JsonConvert.SerializeObject(template));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError ||

httpWebResponse.StatusCode == HttpStatusCode.BadRequest)

{

throw new ServerException();

}

});

}

}

}

using ApplicationLib.Exceptions;

using ApplicationLib.Models;

using ApplicationLib.Services;

using System;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using System.Net;

using Newtonsoft.Json;

using System.IO;

namespace ApplicationLib.Database

{

public class UsersDB : IUserDatabase<UserInfo>

{

#region Properties

private string ApiURL { get; } = "https://aerothedeveloper.ru/sdwpapi/v1.0.0/users";

#endregion

#region Authorization methods

public async Task<UserInfo> TryToLoginAsync(LoginData loginData)

{

return await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL +

$"?login={loginData.Login}&pass={loginData.Password.GetHashCode()}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.NoContent)

{

throw new UserNotFoundException("Неправильный пароль или логин");

}

string responseContent = HTTP.GetResponseContent(httpWebResponse);

return JsonConvert.DeserializeObject<UserInfo>(responseContent);

});

}

#endregion

#region Create new account methods

public async Task CreateNewAccountAsync(UserInfo newUser)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "POST");

using (var stream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(stream))

{

sw.Write(JsonConvert.SerializeObject(newUser));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

});

}

#endregion

#region Check user object methods

public async Task CheckLogin(string login)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?login={login}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

else if (httpWebResponse.StatusCode == HttpStatusCode.NoContent)

{

throw new NotAppropriateUserParam("Логин уже занят");

}

});

}

public async Task CheckEmail(string email)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?email={email}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

else if (httpWebResponse.StatusCode == HttpStatusCode.NoContent)

{

throw new NotAppropriateUserParam("E-mail уже занят");

}

});

}

#endregion

#region Remind pass methods

public async Task RemindPassAsync(string login, string email)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL +

$"/remindpass?login={login}&email={email}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.NoContent)

{

throw new UserNotFoundException("Пользователь с такими параметрами не существует");

}

else if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

});

}

#endregion

#region Update methods

public async Task UpdateUserRecord(UserInfo user)

{

await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL, "PUT");

using (var stream = httpWebRequest.GetRequestStream())

{

using (var sw = new StreamWriter(stream))

{

sw.Write(JsonConvert.SerializeObject(user));

}

}

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

});

}

#endregion

public async Task<UserInfo> GetUserByID(int id)

{

return await Task.Run(() =>

{

HttpWebRequest httpWebRequest = HTTP.GetRequest(ApiURL + $"?id={id}", "GET");

HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)

{

throw new ServerException();

}

string responseContent = HTTP.GetResponseContent(httpWebResponse);

return JsonConvert.DeserializeObject<UserInfo>(responseContent);

});

}

}

}

using System;

using System.Runtime.Serialization;

namespace ApplicationLib.Exceptions

{

/// <summary>

/// Exception which is thrown when the pararmeter of user class is incorrect

/// </summary>

public class NotAppropriateUserParam : SystemException

{

public NotAppropriateUserParam() { }

public NotAppropriateUserParam(string message) : base(message) { }

public NotAppropriateUserParam(string message, Exception innerException)

: base(message, innerException) { }

protected NotAppropriateUserParam(SerializationInfo info, StreamingContext context)

: base(info, context) { }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Exceptions

{

public class ServerException : Exception

{

}

}

using System;

using System.Runtime.Serialization;

namespace ApplicationLib.Exceptions

{

/// <summary>

/// Exception which is thrown when there is en error in finding an user

/// record in the database

/// </summary>

public class UserNotFoundException : SystemException

{

public UserNotFoundException() { }

public UserNotFoundException(string message)

:base(message) { }

public UserNotFoundException(string message, Exception innerException)

: base(message, innerException) { }

protected UserNotFoundException(SerializationInfo info, StreamingContext context)

: base(info, context) { }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Word;

using ApplicationLib.Models;

using ApplicationLib.Word.Interfaces;

namespace ApplicationLib.Factories

{

public class RenderersAbstractFactory : IRenderersAbstractFactory

{

public IWordRenderer GetWordDocumentRender()

{

return new WordRenderer();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Services;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

namespace ApplicationLib.Factories

{

public class ServiceAbstractFactory : IServiceAbstractFactory

{

public IEmailService<UserInfo> GetEmailService()

{

return new EmailService();

}

public IUserService<UserInfo> GetUserService()

{

return new UserService();

}

public ILocalDocumentationService GetLocalDocumentationService()

{

return new LocalDocumentationService();

}

public ILocalTemplateService GetLocalTemplateService()

{

return new LocalTemplatesService();

}

public ICloudDocumentationService GetCloudDocumentationService()

{

return new CloudDocumentationService();

}

public ICloudDocumentsService GetCloudDocumentsService()

{

return new CloudDocumentsService();

}

public ICloudTemplateService GetCloudTemplateService()

{

return new CloudTemplateService();

}

}

}

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ICloudDocumentationService

{

Task DeleteDocumentation(Documentation entity);

Task<IEnumerable<Documentation>> GetAllDocumentations();

Task<IEnumerable<Documentation>> GetUserDocumentations(int id);

Task InsertDocumentation(Documentation entity);

Task UpdateDocumentation(Documentation entity);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface ICloudDocumentsService

{

Task DeleteDocument(Document entity);

Task<IEnumerable<Document>> GetAllDocuments();

Task<IEnumerable<Document>> GetDocumentationDocuments(int documentationID);

Task InsertDocument(Document entity);

Task UpdateDocument(Document entity);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface ICloudTemplateService

{

Task DeleteTemplate(Template entity);

Task<IEnumerable<Template>> GetAllTemplates();

Task<IEnumerable<Template>> GetUserTemplates(int userID);

Task InsertTemplate(Template entity);

Task UpdateTemplate(Template entity);

}

}

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IEmailService<UserType>

{

Task<int> SendCodeEmail(UserType user);

Task<bool> CheckCode(int codeID, string code);

Task DeleteCode(int codeID);

Task SendChangePassLink(UserType user);

}

}

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IUserService<UserType>

{

Task<UserType> AuthorizeUserAsync(LoginData loginData);

Task CreateNewAccountAsync(UserType user);

Task RemindPassAsync(string login, string email);

Task CheckLogin(string login);

Task CheckEmail(string email);

Task UpdateRecord(UserType user);

Task<UserType> GetUserByID(int id);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ICloudDatabase<Type>

{

Task<IEnumerable<Type>> GetAllRecords();

Task<IEnumerable<Type>> GetRecords(string columnName, object value);

Task UpdateRecord(Type documentation);

Task DeleteRecord(Type documentation);

Task InsertRecord(Type documentation);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ICloudDocumentationDB<Type>

{

Task<IEnumerable<Type>> GetAllDocumentations();

Task<IEnumerable<Type>> GetUserDocumentations(int userID);

Task UpdateDocumentation(Type documentation);

Task DeleteDocumentation(int documentationID);

Task InsertDocumentation(Type documentation);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ICloudDocumentDB<DocumentType>

{

Task<IEnumerable<DocumentType>> GetAllDocuments();

Task<IEnumerable<DocumentType>> GetDocumentationDocuments(int documentationID);

Task UpdateDocument(DocumentType document);

Task DeleteDocument(DocumentType document);

Task InsertDocument(DocumentType document);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ICloudTemplatesDB<TemplateType>

{

Task<IEnumerable<TemplateType>> GetAllTemplates();

Task<IEnumerable<TemplateType>> GetUserTemplates(int userID);

Task UpdateTemplate(TemplateType template);

Task DeleteTemplate(TemplateType template);

Task InsertTemplate(TemplateType template);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IEmailDatabase<UserType>

{

Task<int> SendCodeEmail(UserType user);

Task<bool> CheckCode(int codeID, string code);

Task DeleteCode(int codeID);

Task SendChangePassLink(UserType user);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface IUserDatabase<UserType>

{

Task<UserType> TryToLoginAsync(LoginData loginData);

Task CreateNewAccountAsync(UserType newUser);

Task CheckLogin(string login);

Task CheckEmail(string email);

Task RemindPassAsync(string login, string email);

Task UpdateUserRecord(UserType user);

Task<UserType> GetUserByID(int id);

}

}

using ApplicationLib.Models;

using ApplicationLib.Word;

using ApplicationLib.Word.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IRenderersAbstractFactory

{

IWordRenderer GetWordDocumentRender();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface IServiceAbstractFactory

{

IUserService<UserInfo> GetUserService();

IEmailService<UserInfo> GetEmailService();

ILocalDocumentationService GetLocalDocumentationService();

ILocalTemplateService GetLocalTemplateService();

ICloudDocumentationService GetCloudDocumentationService();

ICloudDocumentsService GetCloudDocumentsService();

ICloudTemplateService GetCloudTemplateService();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface ILocalDocumentationService : ILocalStorage

{

string Extension { get; }

Task<IEnumerable<LocalDocumentation>> GetLocalDocumentations();

Task CreateLocalDocumentationFile(LocalDocumentation localDocumentation);

void DeleteLocalDocumentationFile(LocalDocumentation localDocumentation);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ILocalStorage

{

string StoragePath { get; set; }

int ErrorsCount { get; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface ILocalTemplateService: ILocalStorage

{

Task<IEnumerable<LocalTemplate>> GetLocalTemplates();

Task CreateTemplateFile(LocalTemplate template);

void DeleteTemplateFile(LocalTemplate template);

Task RewriteTemplateFile(LocalTemplate template);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using ApplicationLib.Models;

using ApplicationLib.Views;

namespace ApplicationLib.Interfaces

{

public interface IParagraphElement

{

string Hint { get; set; }

string Title { get; set; }

UserControl GetEditView();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

namespace ApplicationLib.Interfaces

{

public interface IParentableItem

{

Item ParentItem { get; }

List<Item> ParentList { get; }

void SetParents(Item parentItem, List<Item> parentList);

}

}

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IParentableParagraph

{

List<Paragraph> ParentList { get; }

Item ParentItem { get; }

void SetParents(Item parentItem, List<Paragraph> parentList);

void RemoveParagraphFromParentList();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface ISerializable

{

string GetJsonString();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IImageSettings : IParagraphSettings

{

Action OnUploadNewImage { get; set; }

}

}

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IParagraphEditView

{

Paragraph Paragraph { get; }

Action RefreshParagraphsUI { get; set; }

Action RefreshParagraphsUIAfterSwap { get; set; }

void DeleteParagraph();

void ShowOrHideHint();

void MoveParagraphUp();

void MoveParagraphDown();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Interfaces

{

public interface IParagraphSettings

{

Action OnParagraphDelete { get; set; }

Action OnParagraphShowOrHideHint { get; set; }

Action MoveParagraphUp { get; set; }

Action MoveParagraphDown { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

using ApplicationLib.Views;

namespace ApplicationLib.Models

{

public class NumberedList : ParagraphElement

{

#region Properties

[JsonProperty("elements")]

public List<NumberedListElement> ListElements { get; set; }

#endregion

public NumberedList() { }

public NumberedList(List<NumberedListElement> listElements)

{

ListElements = listElements;

SetIndexes();

}

private void SetIndexes()

{

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

{

ListElements[i].Index = i.ToString();

}

}

public void MoveItemUp(int itemIndex)

{

if (itemIndex >= 0 && itemIndex < ListElements.Count)

{

if (itemIndex == 0)

{

NumberedListElement firstItem = ListElements[0];

for (int i = 0; i < ListElements.Count - 1; i++)

{

ListElements[i] = ListElements[i + 1];

}

ListElements[ListElements.Count - 1] = firstItem;

}

else

{

NumberedListElement temp = ListElements[itemIndex];

ListElements[itemIndex] = ListElements[itemIndex - 1];

ListElements[itemIndex - 1] = temp;

}

SetIndexes();

}

}

public void MoveItemDown(int itemIndex)

{

if (itemIndex >= 0 && itemIndex < ListElements.Count)

{

if (itemIndex == ListElements.Count - 1)

{

NumberedListElement lastItem = ListElements[ListElements.Count - 1];

for (int i = ListElements.Count - 1; i > 0; i--)

{

ListElements[i] = ListElements[i - 1];

}

ListElements[0] = lastItem;

}

else

{

NumberedListElement temp = ListElements[itemIndex];

ListElements[itemIndex] = ListElements[itemIndex + 1];

ListElements[itemIndex + 1] = temp;

}

SetIndexes();

}

}

public void AddItem(int clickedItemIndex)

{

ListElements.Insert(clickedItemIndex + 1, new NumberedListElement(string.Empty));

SetIndexes();

}

public void DeleteItem(int itemIndex)

{

ListElements.RemoveAt(itemIndex);

SetIndexes();

}

public override UserControl GetEditView()

{

return new NumberedListEditView(ParentParagraph);

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Models

{

public class NumberedListElement

{

[JsonProperty("text")]

public string Text { get; set; }

[JsonProperty("index")]

public string Index { get; set; }

public NumberedListElement() { }

public NumberedListElement(string text)

{

Text = text;

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

namespace ApplicationLib.Models

{

[JsonConverter(typeof(ParagraphJsonConverter))]

public class Paragraph : IParentableParagraph

{

#region Properties

[JsonProperty("type")]

public string Type { get; set; }

[JsonProperty("element")]

public ParagraphElement ParagraphElement { get; set; }

[JsonIgnore]

public Item ParentItem { get; private set; }

[JsonIgnore]

public List<Paragraph> ParentList { get; private set; }

#endregion

#region Constructors

public Paragraph() { }

public Paragraph(string type, ParagraphElement paragraphElement)

{

Type = type;

ParagraphElement = paragraphElement;

ParagraphElement.ParentParagraph = this;

}

#endregion

public void SetParents(Item parentItem, List<Paragraph> parentList)

{

ParentItem = parentItem;

ParentList = parentList;

}

public void RemoveParagraphFromParentList()

{

ParentList.Remove(this);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using ApplicationLib.Interfaces;

using ApplicationLib.Views;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class ParagraphElement : IParagraphElement

{

[JsonProperty("hint")]

public string Hint { get; set; }

[JsonProperty("title")]

public string Title { get; set; }

[JsonIgnore]

public Paragraph ParentParagraph { get; set; }

public virtual UserControl GetEditView()

{

return null;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

using ApplicationLib.Views;

namespace ApplicationLib.Models

{

public class ParagraphImage : ParagraphElement

{

#region Properties

[JsonProperty("source")]

public byte[] ImageSource { get; set; }

#endregion

public ParagraphImage() { }

public ParagraphImage(byte[] imageSource)

{

ImageSource = imageSource;

}

public override UserControl GetEditView()

{

return new ImageEditView(ParentParagraph);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using ApplicationLib.Interfaces;

using ApplicationLib.Views;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class Subparagraph : ParagraphElement

{

#region Properties

[JsonProperty("text")]

public string Text { get; set; }

#endregion

public Subparagraph() { }

public Subparagraph(string text)

{

Text = text;

}

public override UserControl GetEditView() => new SubparagraphEditView(ParentParagraph);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using ApplicationLib.Interfaces;

using ApplicationLib.Views;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class Table : ParagraphElement

{

#region Properties

[JsonProperty("tableCells")]

public string[][] TableCells { get; set; }

#endregion

public Table() { }

public Table(string[][] tableCells)

{

TableCells = tableCells;

}

public void DeleteCol(int colNum)

{

if (colNum >= 0 && TableCells.Length > 0 && colNum < TableCells[0].Length)

for (int i = 0; i < TableCells.Length; i++)

{

string[] newRow = new string[TableCells[i].Length - 1];

int currIndex = 0;

for (int j = 0; j < TableCells[i].Length; j++)

{

if (j != colNum)

{

newRow[currIndex] = TableCells[i][j];

currIndex++;

}

}

TableCells[i] = newRow;

}

}

public void DeleteRow(int rowNum)

{

if (rowNum >= 0 && rowNum < TableCells.Length)

{

string[][] newCells = new string[TableCells.Length - 1][];

int currIndex = 0;

for (int i = 0; i < TableCells.Length; i++)

{

if (i != rowNum)

{

newCells[currIndex] = TableCells[i];

currIndex++;

}

}

TableCells = newCells;

}

}

public void AddNewDownRow(int rowNum)

{

if (rowNum >= 0 && TableCells.Length > 0 && rowNum < TableCells.Length)

{

string[][] newCells = new string[TableCells.Length + 1][];

int currIndex = 0;

for (int i = 0; i < newCells.Length; i++)

{

newCells[i] = new string[TableCells[0].Length];

if (i != rowNum + 1)

{

for (int j = 0; j < newCells[i].Length; j++)

{

newCells[i][j] = TableCells[currIndex][j];

}

currIndex++;

}

}

TableCells = newCells;

}

}

public void AddNewUpRow(int rowNum)

{

if (rowNum >= 0 && TableCells.Length > 0 && rowNum < TableCells.Length)

{

string[][] newCells = new string[TableCells.Length + 1][];

int currIndex = 0;

for (int i = 0; i < newCells.Length; i++)

{

newCells[i] = new string[TableCells[0].Length];

if (i != rowNum)

{

for (int j = 0; j < newCells[i].Length; j++)

{

newCells[i][j] = TableCells[currIndex][j];

}

currIndex++;

}

}

TableCells = newCells;

}

}

public void AddNewLeftCol(int colNum)

{

if (colNum >= 0 && TableCells.Length > 0 && colNum < TableCells[0].Length)

{

string[][] newCells = new string[TableCells.Length][];

for (int i = 0; i < newCells.Length; i++)

{

newCells[i] = new string[TableCells[i].Length + 1];

int currIndex = 0;

for (int j = 0; j < newCells[i].Length; j++)

{

if (j != colNum)

{

newCells[i][j] = TableCells[i][currIndex];

currIndex++;

}

}

}

TableCells = newCells;

}

}

public void AddNewRightCol(int colNum)

{

if (colNum >= 0 && TableCells.Length > 0 && colNum < TableCells[0].Length)

{

string[][] newCells = new string[TableCells.Length][];

for (int i = 0; i < newCells.Length; i++)

{

newCells[i] = new string[TableCells[i].Length + 1];

int currIndex = 0;

for (int j = 0; j < newCells[i].Length; j++)

{

if (j != colNum + 1)

{

newCells[i][j] = TableCells[i][currIndex];

currIndex++;

}

}

}

TableCells = newCells;

}

}

public override UserControl GetEditView()

{

return new TableEditView(ParentParagraph);

}

}

}

namespace ApplicationLib.Models

{

public enum Access

{

Public = 1,

Private = 2

}

}

using System;

using System.Collections.Generic;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class Document

{

[JsonProperty("id")]

public int ID { get; set; }

[JsonProperty("documentationID")]

public int DocumentationID { get; set; }

[JsonProperty("name")]

public string Name { get; set; }

[JsonProperty("authorID")]

public int AuthorID { get; set; }

[JsonProperty("authorName")]

public string AuthorName { get; set; }

[JsonProperty("creationDate")]

public DateTime CreationDate { get; set; }

[JsonProperty("updatedAt")]

public DateTime UpdatedAt { get; set; }

[JsonProperty("access")]

public Access Access { get; set; }

[JsonProperty("items")]

public List<Item> Items { get; set; }

}

}

using System;

using System.Collections.Generic;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class Documentation

{

[JsonProperty("id")]

public int ID { get; set; }

[JsonProperty("name")]

public string Name { get; set; }

[JsonProperty("authorID")]

public int AuthorID { get; set; }

[JsonProperty("authorName")]

public string AuthorName { get; set; }

[JsonProperty("creationDate")]

public DateTime CreationDate { get; set; }

[JsonProperty("updatedAt")]

public DateTime UpdatedAt { get; set; }

[JsonProperty("access")]

public Access Access { get; set; }

[JsonProperty("storageType")]

public StorageType StorageType { get; set; }

[JsonProperty("projectName")]

public string ProjectName { get; set; }

[JsonProperty("teamLeadName")]

public string TeamLeadName { get; set; }

[JsonProperty("managerName")]

public string ManagerName { get; set; }

[JsonProperty("projectCode")]

public string ProjectCode { get; set; }

[JsonProperty("softwareEngineerName")]

public string SoftwareEngineerName { get; set; }

}

}

using System.Collections.Generic;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class Item : IParentableItem

{

[JsonProperty("name")]

public string Name { get; set; }

[JsonProperty("items")]

public List<Item> Items { get; set; }

[JsonProperty("paragraphs")]

public List<Paragraph> Paragraphs { get; set; }

[JsonIgnore]

public Item ParentItem { get; private set; }

[JsonIgnore]

public List<Item> ParentList { get; private set; }

public void SetParents(Item parentItem, List<Item> parentList)

{

ParentItem = parentItem;

ParentList = parentList;

}

}

}

using System.Collections.Generic;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

/// <summary>

/// Class for storing data in the offline storages

/// </summary>

public class LocalDocumentation : ISerializable

{

public Documentation Documentation { get; set; }

public List<Document> Documents { get; set; }

public string DocumentationPath { get; set; }

public LocalDocumentation(Documentation documentation, List<Document> documents)

{

Documentation = documentation;

Documents = documents;

}

public static LocalDocumentation GetObjectFromJsonString(string jsonString)

{

return JsonConvert.DeserializeObject<LocalDocumentation>(jsonString);

}

public string GetJsonString()

{

return JsonConvert.SerializeObject(this);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class LocalTemplate : ISerializable

{

public string FileName { get; set; }

public Template Template { get; set; }

public LocalTemplate(Template template)

{

Template = template;

}

public string GetJsonString()

{

return JsonConvert.SerializeObject(this);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Models

{

public enum StorageType

{

Local = 0,

Cloud = 1,

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using Newtonsoft.Json;

namespace ApplicationLib.Models

{

public class Template

{

[JsonProperty("id")]

public int ID { get; set; }

[JsonProperty("userID")]

public int UserID { get; set; }

[JsonProperty("templateName")]

public string TemplateName { get; set; }

[JsonProperty("createdAt")]

public DateTime CreatedAt { get; set; }

[JsonProperty("updatedAt")]

public DateTime UpdatedAt { get; set; }

[JsonProperty("items")]

public List<Item> Items { get; set; }

public string GetJsonString()

{

return JsonConvert.SerializeObject(this);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

using Newtonsoft.Json.Converters;

namespace ApplicationLib.Models

{

public class ParagraphJsonConverter : CustomCreationConverter<Paragraph>

{

public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)

{

var paragraph = new Paragraph();

while (reader.Read())

{

if (reader.TokenType == JsonToken.EndObject)

return paragraph;

if (reader.TokenType != JsonToken.PropertyName)

continue;

string propertyName = reader.Value.ToString();

if (!reader.Read())

return paragraph;

if ("type".Equals(propertyName))

{

paragraph.Type = reader.Value?.ToString();

}

else if ("element".Equals(propertyName))

{

switch (paragraph.Type)

{

case "Subparagraph":

paragraph.ParagraphElement = serializer.Deserialize<Subparagraph>(reader);

break;

case "Table":

paragraph.ParagraphElement = serializer.Deserialize<Table>(reader);

break;

case "NumberedList":

paragraph.ParagraphElement = serializer.Deserialize<NumberedList>(reader);

break;

case "ParagraphImage":

paragraph.ParagraphElement = serializer.Deserialize<ParagraphImage>(reader);

break;

}

}

}

return paragraph;

}

public override Paragraph Create(Type objectType)

{

return new Paragraph();

}

}

}

namespace ApplicationLib.Models

{

public class LoginData

{

public string Login { get; set; }

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using ApplicationLib.Exceptions;

namespace ApplicationLib.Models

{

public class UserInfo

{

#region Constants

private const string loginAllowedSymbols = "qwertyuiopasdfghklzxcvbnmQWERTYUIOPASDFG" +

"HJKLZXCVBNM1234567890";

private const string emailAllowedSymbols = "qwertyuiopasdfghklzxcvbnmQWERTYUIOPASDFG" +

"HJKLZXCVBNM1234567890!#$%&'\*+-/=?^\_`{|}~@\"(),:;<>@[\\]";

#endregion

/// <summary>

/// The user who is now working with a system

/// </summary>

public static UserInfo CurrentUser { get; set; }

#region Public database properties

public int ID { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public string Name { get; set; }

public string Surname { get; set; }

public DateTime BirthDate { get; set; }

public string Email { get; set; }

public byte[] UserPhoto { get; set; }

#endregion

#region Constructors

public UserInfo() { }

public UserInfo(UserInfo user)

{

ID = user.ID;

Login = user.Login;

Password = user.Password;

Name = user.Name;

Surname = user.Surname;

BirthDate = user.BirthDate;

Email = user.Email;

UserPhoto = user.UserPhoto;

}

#endregion

/// <exception cref="NotAppropriateUserParam">

/// When some of the param is wrong

/// </exception>

public static void CheckUserProperties(UserInfo user)

{

//Login check

if (user.Login.Length < 6 || user.Login.Length > 200)

throw new NotAppropriateUserParam("Длина логина должна быть больше 6 и " +

"меньше 200 символов, логин состоит только из букв латинского алфавита и цифр");

for (int i = 0; i < user.Login.Length; i++)

{

if (loginAllowedSymbols.IndexOf(user.Login[i]) < 0)

throw new NotAppropriateUserParam("Длина логина должна быть больше 6 и " +

"меньше 200 символов, логин состоит только из букв латинского алфавита и цифр");

}

//Name check

if (user.Name.Length < 1)

throw new NotAppropriateUserParam("Имя должно состоять минимум из одной буквы");

//Surname check

if (user.Surname.Length < 1)

throw new NotAppropriateUserParam("Фамилия должна состоять минимум из одного " +

"символа");

//Email check

string email = user.Email;

if (email.Length < 3)

throw new NotAppropriateUserParam("Длина email должна быть больше трех");

if (email[0] == '.')

throw new NotAppropriateUserParam("Email не может начинаться с точки");

if (email.IndexOf("@") < 0)

throw new NotAppropriateUserParam("Email должен содержать символ '@'");

for (int i = 0; i < email.Length; i++)

{

if (emailAllowedSymbols.IndexOf(email[0]) < 0)

throw new NotAppropriateUserParam("Один мз символов введенного email недопустим");

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Database;

namespace ApplicationLib.Services

{

public class CloudDocumentationService : ICloudDocumentationService

{

public ICloudDocumentationDB<Documentation> Database { get; }

public CloudDocumentationService()

{

Database = new DocumentationDB();

}

public async Task DeleteDocumentation(Documentation entity) =>

await Database.DeleteDocumentation(entity.ID);

public async Task<IEnumerable<Documentation>> GetAllDocumentations() =>

await Database.GetAllDocumentations();

public async Task<IEnumerable<Documentation>> GetUserDocumentations(int userID) =>

await Database.GetUserDocumentations(userID);

public async Task InsertDocumentation(Documentation entity) =>

await Database.InsertDocumentation(entity);

public async Task UpdateDocumentation(Documentation entity) =>

await Database.UpdateDocumentation(entity);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Database;

namespace ApplicationLib.Services

{

public class CloudDocumentsService : ICloudDocumentsService

{

public ICloudDocumentDB<Document> Database { get; }

public CloudDocumentsService()

{

Database = new DocumentsDB();

}

public async Task DeleteDocument(Document document)

{

await Database.DeleteDocument(document);

}

public async Task<IEnumerable<Document>> GetAllDocuments()

{

return await Database.GetAllDocuments();

}

public async Task<IEnumerable<Document>> GetDocumentationDocuments(

int documentationID)

{

return await Database.GetDocumentationDocuments(documentationID);

}

public async Task InsertDocument(Document document)

{

await Database.InsertDocument(document);

}

public async Task UpdateDocument(Document document)

{

await Database.UpdateDocument(document);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using ApplicationLib.Database;

namespace ApplicationLib.Services

{

public class CloudTemplateService : ICloudTemplateService

{

public ICloudTemplatesDB<Template> Database { get; }

public CloudTemplateService()

{

Database = new TemplatesDB();

}

public async Task DeleteTemplate(Template template)

{

await Database.DeleteTemplate(template);

}

public async Task<IEnumerable<Template>> GetAllTemplates()

{

return await Database.GetAllTemplates();

}

public async Task<IEnumerable<Template>> GetUserTemplates(int userID)

{

return await Database.GetUserTemplates(userID);

}

public async Task InsertTemplate(Template template)

{

await Database.InsertTemplate(template);

}

public async Task UpdateTemplate(Template template)

{

await Database.UpdateTemplate(template);

}

}

}

using ApplicationLib.Database;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Services

{

public class EmailService : IEmailService<UserInfo>

{

private IEmailDatabase<UserInfo> Email { get; } = new EmailDB();

public async Task<bool> CheckCode(int codeID, string code)

{

return await Email.CheckCode(codeID, code);

}

public async Task DeleteCode(int codeID)

{

await Email.DeleteCode(codeID);

}

public async Task SendChangePassLink(UserInfo user)

{

await Email.SendChangePassLink(user);

}

public async Task<int> SendCodeEmail(UserInfo user) =>

await Email.SendCodeEmail(user);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Database;

using ApplicationLib.Interfaces;

namespace ApplicationLib.Services

{

public class UserService : IUserService<UserInfo>

{

private IUserDatabase<UserInfo> Database { get; } = new UsersDB();

public async Task<UserInfo> AuthorizeUserAsync(LoginData loginData) =>

await Database.TryToLoginAsync(loginData);

public async Task CreateNewAccountAsync(UserInfo newUser) =>

await Database.CreateNewAccountAsync(newUser);

public async Task RemindPassAsync(string login, string email) =>

await Database.RemindPassAsync(login, email);

public async Task UpdateRecord(UserInfo user) =>

await Database.UpdateUserRecord(user);

public async Task<UserInfo> GetUserByID(int id) =>

await Database.GetUserByID(id);

public async Task CheckLogin(string login) =>

await Database.CheckLogin(login);

public async Task CheckEmail(string email) =>

await Database.CheckEmail(email);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using Newtonsoft.Json;

namespace ApplicationLib.Services

{

public class LocalDocumentationService : ILocalDocumentationService

{

public string StoragePath { get; set; }

public int ErrorsCount { get; private set; }

public string Extension => ".sdwp";

public LocalDocumentationService() { }

public async Task CreateLocalDocumentationFile(LocalDocumentation localDocumentation)

{

byte[] localDocumentationBytes = GetByteArrayFromString(localDocumentation.GetJsonString());

using (var fs = new FileStream(localDocumentation.DocumentationPath, FileMode.Create, FileAccess.Write))

{

fs.Seek(0, SeekOrigin.Begin);

await fs.WriteAsync(localDocumentationBytes, 0, localDocumentationBytes.Length);

}

}

private byte[] GetByteArrayFromString(string str)

{

Encoding encoding = Encoding.UTF8;

return encoding.GetBytes(str);

}

/// <summary>

/// Deletes the documentation file is this file exists

/// </summary>

/// <exception cref="FileNotFoundException"/>

public void DeleteLocalDocumentationFile(LocalDocumentation localDocumentation)

{

string path = Path.Combine(localDocumentation.DocumentationPath);

if (File.Exists(path))

{

File.Delete(path);

}

else

{

throw new FileNotFoundException("Файл документации не найден");

}

}

public async Task<IEnumerable<LocalDocumentation>> GetLocalDocumentations()

{

return await Task.Run(() =>

{

string[] filePaths = Directory.GetFiles(StoragePath, "\*.sdwp");

List<LocalDocumentation> localDocumentations = new List<LocalDocumentation>();

ErrorsCount = 0;

foreach (string filePath in filePaths)

{

try

{

byte[] fileBytes;

using (var fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))

{

fs.Seek(0, SeekOrigin.Begin);

fileBytes = new byte[fs.Length];

fs.Read(fileBytes, 0, (int)fs.Length);

}

string localDocJsonString = GetStringFromByteArray(fileBytes);

LocalDocumentation localDocumentation = JsonConvert.DeserializeObject

<LocalDocumentation>(localDocJsonString);

localDocumentation.DocumentationPath = filePath;

localDocumentations.Add(localDocumentation);

}

catch (Exception)

{

ErrorsCount++;

}

}

return localDocumentations;

});

}

private string GetStringFromByteArray(byte[] arr)

{

Encoding encoding = Encoding.UTF8;

return encoding.GetString(arr);

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Interfaces;

using ApplicationLib.Models;

using Newtonsoft.Json;

namespace ApplicationLib.Services

{

public class LocalTemplatesService : ILocalTemplateService

{

public string StoragePath { get; set; }

public int ErrorsCount { get; private set; }

public static string TemplateExtension { get; } = ".tsdwp";

public async Task RewriteTemplateFile(LocalTemplate localTemplate)

{

byte[] templateBytes = GetStringBytes(localTemplate.GetJsonString());

string templatePath = StoragePath + "\\" + localTemplate.FileName;

using (var fs = new FileStream(templatePath, FileMode.Truncate, FileAccess.Write))

{

fs.Seek(0, SeekOrigin.Begin);

await fs.WriteAsync(templateBytes, 0, templateBytes.Length);

}

}

public async Task CreateTemplateFile(LocalTemplate template)

{

byte[] templateBytes = GetStringBytes(template.GetJsonString());

string templatePath = GetTemplatePath(template);

using (var fs = new FileStream(templatePath, FileMode.Create, FileAccess.Write))

{

fs.Seek(0, SeekOrigin.Begin);

await fs.WriteAsync(templateBytes, 0, templateBytes.Length);

}

}

private string GetTemplatePath(LocalTemplate localTemplate)

{

string templatePath = Path.Combine(StoragePath, localTemplate.Template.TemplateName);

if (File.Exists(templatePath + TemplateExtension))

{

int addNum = 1;

while (File.Exists(templatePath + " " + addNum.ToString() + TemplateExtension))

{

addNum++;

}

localTemplate.FileName = localTemplate.Template.TemplateName + " " + addNum.ToString() + TemplateExtension;

templatePath += " " + addNum.ToString() + TemplateExtension;

}

else

{

localTemplate.FileName = localTemplate.Template.TemplateName + TemplateExtension;

templatePath += TemplateExtension;

}

return templatePath;

}

private byte[] GetStringBytes(string str)

{

Encoding encoding = Encoding.UTF8;

return encoding.GetBytes(str);

}

public void DeleteTemplateFile(LocalTemplate template)

{

string templatePath = Path.Combine(StoragePath, template.FileName);

if (File.Exists(templatePath))

{

File.Delete(templatePath);

return;

}

throw new FileNotFoundException("Файл шаблона не найден");

}

public async Task<IEnumerable<LocalTemplate>> GetLocalTemplates()

{

return await Task.Run(() =>

{

ErrorsCount = 0;

string[] filePaths = Directory.GetFiles(StoragePath);

List<LocalTemplate> templates = new List<LocalTemplate>();

foreach (string filePath in filePaths)

{

try

{

byte[] fileBytes;

using (var fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))

{

fs.Seek(0, SeekOrigin.Begin);

fileBytes = new byte[fs.Length];

fs.Read(fileBytes, 0, (int)fs.Length);

}

string templateJsonString = GetStringFromBytes(fileBytes);

LocalTemplate template = JsonConvert.DeserializeObject<LocalTemplate>(templateJsonString);

template.FileName = filePath.Substring(filePath.LastIndexOf('\\') + 1);

templates.Add(template);

}

catch (Exception)

{

ErrorsCount++;

}

}

return templates;

});

}

private string GetStringFromBytes(byte[] bytes)

{

Encoding encoding = Encoding.UTF8;

return encoding.GetString(bytes);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using ApplicationLib.Models;

namespace ApplicationLib.Tests

{

[TestFixture]

public class NumberedListTests

{

private NumberedListElement fixedElement;

private NumberedList numberedList;

[SetUp]

public void SetUpTest()

{

numberedList = new NumberedList(new List<NumberedListElement>()

{

new NumberedListElement("first"),

new NumberedListElement("second"),

new NumberedListElement("third"),

new NumberedListElement("fourth"),

new NumberedListElement("fifth")

});

}

[TestCase(0, ExpectedResult = 4)]

[TestCase(1, ExpectedResult = 0)]

[TestCase(2, ExpectedResult = 1)]

[TestCase(3, ExpectedResult = 2)]

[TestCase(4, ExpectedResult = 3)]

public int TestMoveItemUp(int itemIndex)

{

fixedElement = numberedList.ListElements[itemIndex];

numberedList.MoveItemUp(itemIndex);

return numberedList.ListElements.FindIndex(item => item.Equals(fixedElement));

}

[TestCase(0, ExpectedResult = 1)]

[TestCase(1, ExpectedResult = 2)]

[TestCase(2, ExpectedResult = 3)]

[TestCase(3, ExpectedResult = 4)]

[TestCase(4, ExpectedResult = 0)]

public int TestMoveDownItem(int itemIndex)

{

fixedElement = numberedList.ListElements[itemIndex];

numberedList.MoveItemDown(itemIndex);

return numberedList.ListElements.FindIndex(item => item.Equals(fixedElement));

}

[TestCase(0, ExpectedResult = 1)]

[TestCase(1, ExpectedResult = 2)]

[TestCase(2, ExpectedResult = 3)]

[TestCase(3, ExpectedResult = 4)]

[TestCase(4, ExpectedResult = 5)]

public int TestAddNewItem(int selectedItemIndex)

{

fixedElement = new NumberedListElement(string.Empty);

numberedList.AddItem(selectedItemIndex);

return numberedList.ListElements.FindIndex(item => item.Text == fixedElement.Text);

}

[TearDown]

public void TearDownTest()

{

fixedElement = null;

numberedList = null;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using ApplicationLib.Views;

using ApplicationLib.Models;

namespace ApplicationLib.Tests

{

[TestFixture]

public class TableTest

{

Table table;

[SetUp]

public void SetUpTests()

{

table = new Table(GetCells());

}

private string[][] GetCells()

{

return new string[5][]

{

new string[4] {"1", "2", "3", "4",},

new string[4] {"a", "b", "c", "d",},

new string[4] {"1", "2", "3", "4",},

new string[4] {"1", "2", "3", "4",},

new string[4] {"aa", "bb", "cc", "dd",}

};

}

[TestCase(-1)]

[TestCase(0)]

[TestCase(5)]

[TestCase(4)]

[TestCase(3)]

public void TestDeleteCol(int value)

{

int initialColCount = table.TableCells[0].Length;

table.DeleteCol(value);

if ((value < 0 || value >= initialColCount))

{

for (int i = 0; i < table.TableCells.Length; i++)

{

if (table.TableCells[i].Length != 4)

Assert.Fail($"Out of range test failed {table.TableCells[i].Length}");

}

}

else

{

for (int i = 0; i < table.TableCells.Length; i++)

{

if (table.TableCells[i].Length != 3)

Assert.Fail(table.TableCells[i].Length.ToString());

}

}

Assert.Pass();

}

[TestCase(-1)]

[TestCase(0)]

[TestCase(5)]

[TestCase(4)]

[TestCase(3)]

public void TestDeleteRow(int value)

{

int initialRowCount = table.TableCells.Length;

table.DeleteRow(value);

if (value < 0 || value >= initialRowCount)

{

if (table.TableCells.Length != 5)

Assert.Fail("Out of range test failed");

}

else

{

if (table.TableCells.Length != 4)

Assert.Fail("Range test failed");

}

Assert.Pass();

}

[TestCase(-1)]

[TestCase(0)]

[TestCase(5)]

[TestCase(4)]

[TestCase(3)]

public void TestAddDownRow(int rowIndex)

{

table.AddNewDownRow(rowIndex);

if (rowIndex < 0 || rowIndex >= table.TableCells.Length)

{

if (table.TableCells.Length != 5)

Assert.Fail("Out of ranged test failed");

}

else

{

if (table.TableCells.Length != 6)

Assert.Fail("Range test failed");

}

Assert.Pass();

}

[TestCase(-1)]

[TestCase(0)]

[TestCase(5)]

[TestCase(4)]

[TestCase(3)]

public void TestAddUpRow(int rowIndex)

{

table.AddNewUpRow(rowIndex);

if (rowIndex < 0 || rowIndex >= table.TableCells.Length)

{

if (table.TableCells.Length != 5)

Assert.Fail("Out of ranged test failed");

}

else

{

if (table.TableCells.Length != 6)

Assert.Fail("Range test failed");

}

Assert.Pass();

}

[TestCase(-1)]

[TestCase(0)]

[TestCase(5)]

[TestCase(4)]

[TestCase(3)]

public void TestAddLeftCol(int colNum)

{

int intialColCount = table.TableCells[0].Length;

table.AddNewLeftCol(colNum);

if (colNum < 0 || colNum >= intialColCount)

{

for (int i = 0; i < table.TableCells.Length; i++)

{

if (table.TableCells[i].Length != 4)

Assert.Fail("Ranged test failed");

}

}

else

{

for (int i = 0; i < table.TableCells.Length; i++)

{

if (table.TableCells[i].Length != 5)

Assert.Fail("Ranged test failed");

}

}

Assert.Pass();

}

[TestCase(-1)]

[TestCase(0)]

[TestCase(5)]

[TestCase(4)]

[TestCase(3)]

public void TestAddRightCol(int colNum)

{

int intialColCount = table.TableCells[0].Length;

table.AddNewRightCol(colNum);

if (colNum < 0 || colNum >= intialColCount)

{

for (int i = 0; i < table.TableCells.Length; i++)

{

if (table.TableCells[i].Length != 4)

Assert.Fail("Ranged test failed");

}

}

else

{

for (int i = 0; i < table.TableCells.Length; i++)

{

if (table.TableCells[i].Length != 5)

Assert.Fail("Ranged test failed");

}

}

Assert.Pass();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using ApplicationLib.Word.Interfaces;

namespace ApplicationLib.Word.Commands

{

public class FooterCommand : IWordCommand

{

#region Properties

public WordprocessingDocument WordDocument { get; }

public ICommandsContainer CommandsContainer { get; }

#endregion

public FooterCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

var MainPart = WordDocument.MainDocumentPart;

MainPart.DeleteParts(MainPart.FooterParts);

FooterPart footerPart = MainPart.AddNewPart<FooterPart>();

string footerID = MainPart.GetIdOfPart(footerPart);

CreateFooterContent(footerPart);

var sectionPropsList = MainPart.Document.Body.Descendants<SectionProperties>().ToList();

var sectionProps = sectionPropsList[sectionPropsList.Count - 1];

sectionProps.Append(new FooterReference() { Type = HeaderFooterValues.Default, Id = footerID });

}

public void CreateFooterContent(FooterPart footerPart)

{

Footer footer = new Footer() { MCAttributes = new MarkupCompatibilityAttributes() { Ignorable = "w14 wp14" } };

footer.AddNamespaceDeclaration("wpc", "http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas");

footer.AddNamespaceDeclaration("mc", "http://schemas.openxmlformats.org/markup-compatibility/2006");

footer.AddNamespaceDeclaration("o", "urn:schemas-microsoft-com:office:office");

footer.AddNamespaceDeclaration("r", "http://schemas.openxmlformats.org/officeDocument/2006/relationships");

footer.AddNamespaceDeclaration("m", "http://schemas.openxmlformats.org/officeDocument/2006/math");

footer.AddNamespaceDeclaration("v", "urn:schemas-microsoft-com:vml");

footer.AddNamespaceDeclaration("wp14", "http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing");

footer.AddNamespaceDeclaration("wp", "http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing");

footer.AddNamespaceDeclaration("w10", "urn:schemas-microsoft-com:office:word");

footer.AddNamespaceDeclaration("w", "http://schemas.openxmlformats.org/wordprocessingml/2006/main");

footer.AddNamespaceDeclaration("w14", "http://schemas.microsoft.com/office/word/2010/wordml");

footer.AddNamespaceDeclaration("wpg", "http://schemas.microsoft.com/office/word/2010/wordprocessingGroup");

footer.AddNamespaceDeclaration("wpi", "http://schemas.microsoft.com/office/word/2010/wordprocessingInk");

footer.AddNamespaceDeclaration("wne", "http://schemas.microsoft.com/office/word/2006/wordml");

footer.AddNamespaceDeclaration("wps", "http://schemas.microsoft.com/office/word/2010/wordprocessingShape");

WordTable wordTable = new WordTable();

TableProperties tableProperties = new TableProperties(new TableBorders(

new TopBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 12

},

new BottomBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 12

},

new LeftBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 12

},

new RightBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 12

},

new InsideHorizontalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 12

},

new InsideVerticalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 12

}), new Justification() { Val = JustificationValues.Center });

wordTable.Append(tableProperties);

for (int i = 0; i < RenderData.Obj.RenderSettings.FooterTable.Length; i++)

{

TableRow tableRow = new TableRow();

for (int j = 0; j < RenderData.Obj.RenderSettings.FooterTable[i].Length; j++)

{

TableCell tableCell = new TableCell();

WordParagraph paragraph = new WordParagraph();

ParagraphProperties pproperties = new ParagraphProperties(

new Justification() { Val = JustificationValues.Center }, new SpacingBetweenLines()

{

After = "0",

Before = "0",

Line = "200"

});

paragraph.Append(pproperties);

Run run = new Run();

RunProperties runProperties = new RunProperties(new RunFonts()

{

Ascii = RenderData.Obj.RenderSettings.FontFamily,

HighAnsi = RenderData.Obj.RenderSettings.FontFamily

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize }

};

Text text = new Text(RenderData.Obj.RenderSettings.FooterTable[i][j]);

run.Append(runProperties);

run.Append(text);

paragraph.Append(run);

tableCell.Append(paragraph);

tableRow.Append(tableCell);

}

wordTable.Append(tableRow);

}

footer.Append(wordTable);

footerPart.Footer = footer;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

namespace ApplicationLib.Word.Commands

{

public class HeaderCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ICommandsContainer CommandsContainer { get; }

public HeaderCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

var MainPart = WordDocument.MainDocumentPart;

MainPart.DeleteParts(MainPart.HeaderParts);

HeaderPart headerPart = MainPart.AddNewPart<HeaderPart>();

string headerID = MainPart.GetIdOfPart(headerPart);

CreateHeaderContent(headerPart);

var sectionPropsList = MainPart.Document.Body.Descendants<SectionProperties>().ToList();

var sectionProps = sectionPropsList[sectionPropsList.Count - 1];

sectionProps.Append(new HeaderReference() { Type = HeaderFooterValues.Default, Id = headerID });

}

private void CreateHeaderContent(HeaderPart headerPart)

{

Header header = new Header() { MCAttributes = new MarkupCompatibilityAttributes() { Ignorable = "w14 wp14" } };

header.AddNamespaceDeclaration("wpc", "http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas");

header.AddNamespaceDeclaration("mc", "http://schemas.openxmlformats.org/markup-compatibility/2006");

header.AddNamespaceDeclaration("o", "urn:schemas-microsoft-com:office:office");

header.AddNamespaceDeclaration("r", "http://schemas.openxmlformats.org/officeDocument/2006/relationships");

header.AddNamespaceDeclaration("m", "http://schemas.openxmlformats.org/officeDocument/2006/math");

header.AddNamespaceDeclaration("v", "urn:schemas-microsoft-com:vml");

header.AddNamespaceDeclaration("wp14", "http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing");

header.AddNamespaceDeclaration("wp", "http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing");

header.AddNamespaceDeclaration("w10", "urn:schemas-microsoft-com:office:word");

header.AddNamespaceDeclaration("w", "http://schemas.openxmlformats.org/wordprocessingml/2006/main");

header.AddNamespaceDeclaration("w14", "http://schemas.microsoft.com/office/word/2010/wordml");

header.AddNamespaceDeclaration("wpg", "http://schemas.microsoft.com/office/word/2010/wordprocessingGroup");

header.AddNamespaceDeclaration("wpi", "http://schemas.microsoft.com/office/word/2010/wordprocessingInk");

header.AddNamespaceDeclaration("wne", "http://schemas.microsoft.com/office/word/2006/wordml");

header.AddNamespaceDeclaration("wps", "http://schemas.microsoft.com/office/word/2010/wordprocessingShape");

header.Append(GetHeaderContent());

headerPart.Header = header;

}

private SdtBlock GetHeaderContent()

{

return new SdtBlock(

new SdtProperties(

new SdtId() { Val = 317275692 },

new SdtContentDocPartObject(

new DocPartGallery() { Val = "Page Numbers (Top of Page)" },

new DocPartUnique())),

new SdtContentBlock(

new WordParagraph(

new ParagraphProperties(

new ParagraphStyleId() { Val = "Header" },

new Justification() { Val = JustificationValues.Center }),

new SimpleField(

new Run(

new RunProperties(

new NoProof(),

new SpacingBetweenLines()

{

After = "100",

Before = "100",

Line = "200"

}),

new Text("1"))

{ RsidRunAddition = "001F06F5" })

{ Instruction = " PAGE \\\* MERGEFORMAT " })

{ RsidParagraphAddition = "00F1559F", RsidRunAdditionDefault = "00BB29E4" },

new WordParagraph(

new ParagraphProperties(

new Caps(),

new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize },

new Justification() { Val = JustificationValues.Center },

new SpacingBetweenLines()

{

After = "100",

Before = "0",

Line = "200"

}),

new Run(

new RunProperties(new RunFonts()

{

Ascii = RenderData.Obj.RenderSettings.FontFamily,

HighAnsi = RenderData.Obj.RenderSettings.FontFamily

}),

new Text("RU.17701729.04.03-01 ТЗ")))));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class ItemHeaderCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public string Name { get; }

public ItemHeaderCommand(WordprocessingDocument wordDocument, string name)

{

WordDocument = wordDocument;

Name = name;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetSectionHeadParagraph());

}

private WordParagraph GetSectionHeadParagraph()

{

WordParagraph p = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties(new Justification() { Val = JustificationValues.Center });

p.Append(pp);

Run run = new Run();

RunProperties runProperties = new RunProperties(new RunFonts()

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize },

Caps = new Caps(),

Bold = new Bold()

};

run.Append(runProperties);

Text text = new Text(Name);

run.Append(text);

p.Append(run);

return p;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Word.Containers;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class ItemsCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ICommandsContainer CommandsContainer { get; private set; }

public ItemsCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

CommandsContainer = new CommandsContainer();

}

public void Render()

{

CreateCommandsList();

CommandsContainer.Render();

}

private void CreateCommandsList()

{

CommandsContainer.Refresh();

string index = "0";

foreach (Item item in RenderData.Obj.Document.Items)

{

index = (int.Parse(index) + 1).ToString();

AddCommandsToList(item, 0, index);

CommandsContainer.Add(new EndOfPageCommand(WordDocument));

}

CommandsContainer.Add(new SectionPtrCommand(WordDocument));

}

private void AddCommandsToList(Item item, int depth, string index)

{

if (depth == 0)

CommandsContainer.Add(new ItemHeaderCommand(WordDocument, index + ". " + item.Name));

else

CommandsContainer.Add(new ParagraphHeaderCommand(WordDocument, index + " " + item.Name, depth));

if (item.Items != null)

{

string dopIndex = "0";

foreach (Item i in item.Items)

{

dopIndex = (int.Parse(dopIndex) + 1).ToString();

AddCommandsToList(i, depth + 1, index + "." + dopIndex);

}

}

if (item.Paragraphs != null)

{

foreach (Models.Paragraph paragraph in item.Paragraphs)

{

switch (paragraph.Type)

{

case "Table":

CommandsContainer.Add(new TableCommand(WordDocument,

paragraph.ParagraphElement as Models.Table, depth));

break;

case "Subparagraph":

CommandsContainer.Add(new SubparagraphCommand(WordDocument,

paragraph.ParagraphElement as Subparagraph, depth));

break;

case "NumberedList":

CommandsContainer.Add(new NumberedListCommand(WordDocument,

paragraph.ParagraphElement as NumberedList, depth));

break;

case "ParagraphImage":

CommandsContainer.Add(new ParagraphImageCommand(WordDocument,

paragraph.ParagraphElement as ParagraphImage, depth));

break;

}

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 1));

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class ParagraphHeaderCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public string Name { get; }

public int Depth { get; }

public ParagraphHeaderCommand(WordprocessingDocument document, string name, int depth)

{

WordDocument = document;

Name = name;

Depth = depth;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetParagraphHeader());

}

private WordParagraph GetParagraphHeader()

{

WordParagraph p = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties(

new Justification() { Val = JustificationValues.Left },

new Indentation() { Left = (RenderData.Obj.RenderSettings.TabValue \* Depth).ToString() });

p.Append(pp);

Run run = new Run();

RunProperties runProperties = new RunProperties(new RunFonts()

{ HighAnsi = RenderData.Obj.RenderSettings.FontFamily, Ascii = RenderData.Obj.RenderSettings.FontFamily })

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize },

Bold = new Bold()

};

Text text = new Text(Name);

run.Append(runProperties);

run.Append(text);

p.Append(run);

return p;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class NumberedListCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

private int Depth { get; }

private NumberedList NumberedList { get; }

public NumberedListCommand(WordprocessingDocument wordDocument, NumberedList numberedList,

int depth)

{

Depth = depth;

WordDocument = wordDocument;

NumberedList = numberedList;

}

public void Render()

{

var paragraphs = RenderNumberedList();

foreach (var p in paragraphs)

{

WordDocument.MainDocumentPart.Document.Body.Append(p);

}

RenderData.Obj.CurrentNumID++;

}

private IEnumerable<WordParagraph> RenderNumberedList()

{

List<WordParagraph> wordParagraphs = new List<WordParagraph>();

int index = 1;

foreach (NumberedListElement element in NumberedList.ListElements)

{

WordParagraph p = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties()

{

NumberingProperties = new NumberingProperties(

new NumberingId() { Val = RenderData.Obj.CurrentNumID},

new NumberingLevelReference() {Val = 0 }),

Indentation = new Indentation() { Left = (500 \* Depth).ToString() },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "250",

LineRule = LineSpacingRuleValues.Exact

},

};

NumberingProperties numberingProperties = new NumberingProperties()

{

NumberingLevelReference = new NumberingLevelReference() { Val = 0 },

NumberingId = new NumberingId() { Val = 0 }

};

pp.Append(numberingProperties);

p.Append(pp);

Run run = new Run();

RunProperties runProperties = new RunProperties(new RunFonts() { HighAnsi = "Times New Roman" })

{

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize },

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor }

};

Text text = new Text(index + ") " + element.Text);

run.Append(runProperties);

run.Append(text);

p.Append(run);

wordParagraphs.Add(p);

index++;

}

return wordParagraphs;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using A = DocumentFormat.OpenXml.Drawing;

using DW = DocumentFormat.OpenXml.Drawing.Wordprocessing;

using PIC = DocumentFormat.OpenXml.Drawing.Pictures;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

namespace ApplicationLib.Word.Commands

{

public class ParagraphImageCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ParagraphImage ParagraphImage { get; }

public int Depth { get; }

public ParagraphImageCommand(WordprocessingDocument document, ParagraphImage paragraphImage,

int depth)

{

WordDocument = document;

ParagraphImage = paragraphImage;

Depth = depth;

}

public void Render()

{

ImagePart imagePart = WordDocument.MainDocumentPart.AddImagePart(ImagePartType.Png);

using (var ms = new MemoryStream(ParagraphImage.ImageSource))

{

imagePart.FeedData(ms);

}

WordParagraph p = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties()

{

Indentation = new Indentation() { Left = (500 \* Depth).ToString() }

};

p.Append(pp);

Run run = new Run(GetDrawing(WordDocument.MainDocumentPart.GetIdOfPart(imagePart)));

p.Append(run);

WordDocument.MainDocumentPart.Document.Body.AppendChild(p);

WordDocument.MainDocumentPart.Document.Body.AppendChild(GetParagraphImageTitle());

}

private Drawing GetDrawing(string relationshipId)

{

var element =

new Drawing(

new DW.Inline(

new DW.Extent() { Cx = 990000L, Cy = 792000L },

new DW.EffectExtent()

{

LeftEdge = 0L,

TopEdge = 0L,

RightEdge = 0L,

BottomEdge = 0L

},

new DW.DocProperties()

{

Id = (UInt32Value)1U,

Name = "Picture 1"

},

new DW.NonVisualGraphicFrameDrawingProperties(

new A.GraphicFrameLocks() { NoChangeAspect = true }),

new A.Graphic(

new A.GraphicData(

new PIC.Picture(

new PIC.NonVisualPictureProperties(

new PIC.NonVisualDrawingProperties()

{

Id = (UInt32Value)0U,

Name = "New Bitmap Image.jpg"

},

new PIC.NonVisualPictureDrawingProperties()),

new PIC.BlipFill(

new A.Blip(

new A.BlipExtensionList(

new A.BlipExtension()

{

Uri = "{28A0092B-C50C-407E-A947-70E740481C1C}"

})

)

{

Embed = relationshipId,

CompressionState =

A.BlipCompressionValues.Print

},

new A.Stretch(

new A.FillRectangle())),

new PIC.ShapeProperties(

new A.Transform2D(

new A.Offset() { X = 0L, Y = 0L },

new A.Extents() { Cx = 990000L, Cy = 792000L }),

new A.PresetGeometry(

new A.AdjustValueList()

)

{ Preset = A.ShapeTypeValues.Rectangle }))

)

{ Uri = "http://schemas.openxmlformats.org/drawingml/2006/picture" })

)

{

DistanceFromTop = (UInt32Value)0U,

DistanceFromBottom = (UInt32Value)0U,

DistanceFromLeft = (UInt32Value)0U,

DistanceFromRight = (UInt32Value)0U,

EditId = "50D07946"

});

return element;

}

private WordParagraph GetParagraphImageTitle()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var runNumber = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = (int.Parse(RenderData.Obj.

RenderSettings.DefaultTextSize) - 2).ToString() },

Italic = new Italic()

};

runNumber.PrependChild(runProperties);

var text = new Text("Рисунок (номер). ");

runNumber.Append(text);

var runName = new Run();

var runNameProperties = new RunProperties(new RunFonts()

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize()

{

Val = (int.Parse(RenderData.Obj.

RenderSettings.DefaultTextSize) - 2).ToString()

},

};

var textName = new Text(ParagraphImage.Title);

runName.Append(runNameProperties);

runName.Append(textName);

paragraph.Append(runNumber);

paragraph.Append(runName);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class SubparagraphCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public Subparagraph Subparagraph { get; }

public int Depth { get; }

public SubparagraphCommand(WordprocessingDocument wordDocument, Subparagraph subparagraph,

int depth)

{

WordDocument = wordDocument;

Subparagraph = subparagraph;

Depth = depth;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(RenderSubparagraph());

}

private WordParagraph RenderSubparagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Both },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

},

Indentation = new Indentation() { Left = (500 \* Depth).ToString() }

};

paragraph.Append(pp);

for (int i = 0; i < Depth; i++)

{

paragraph.Append(new TabCommand().GetElement());

}

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue("Times New Roman"),

Ascii = "Times New Roman"

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize },

};

run.PrependChild(runProperties);

var text = new Text(Subparagraph.Text);

run.Append(text);

paragraph.Append(new TabCommand().GetElement());

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using ApplicationLib.Word.Interfaces;

using ApplicationLib.Models;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class TableCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

private Models.Table Table { get; }

private int Depth { get; }

public TableCommand(WordprocessingDocument wordDocument, Models.Table table,

int depth)

{

Depth = depth;

WordDocument = wordDocument;

Table = table;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetTableNumberParagraph());

WordDocument.MainDocumentPart.Document.Body.Append(GetTableNameParagraph());

WordTable wordTable = RenderTable();

WordDocument.MainDocumentPart.Document.Body.Append(wordTable);

}

#warning correct classes tables

private WordParagraph GetTableNumberParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Right },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue("Times New Roman"),

Ascii = "Times New Roman"

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize()

{

Val = RenderData.Obj.RenderSettings.DefaultTextSize

},

};

run.PrependChild(runProperties);

var text = new Text("Таблица (номер)");

run.Append(text);

paragraph.Append(run);

return paragraph;

}

private WordParagraph GetTableNameParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily),

Ascii = RenderData.Obj.RenderSettings.FontFamily

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize()

{

Val = RenderData.Obj.RenderSettings.DefaultTextSize

},

};

run.PrependChild(runProperties);

var text = new Text(Table.Title);

run.Append(text);

paragraph.Append(run);

return paragraph;

}

private WordTable RenderTable()

{

WordTable wordTable = new WordTable();

TableProperties tableProperties = new TableProperties(new TableBorders(

new TopBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 6

},

new BottomBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 6

},

new LeftBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 6

},

new RightBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 6

},

new InsideHorizontalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 6

},

new InsideVerticalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 6

}));

wordTable.Append(tableProperties);

string[][] data = Table.TableCells;

bool didMethodsStart = false;

for (int i = 0; i < data.GetLength(0); i++)

{

TableRow tableRow = new TableRow();

tableRow.Append(new TableRowProperties(new TableRowHeight() { Val = 100, HeightType = HeightRuleValues.Auto }));

for (int j = 0; j < data[i].Length; j++)

{

TableCell tableCell = new TableCell();

if (CheckIfHeaderRow(data, i))

{

if (j == 0)

tableCell.Append(new TableCellProperties(

new HorizontalMerge() { Val = MergedCellValues.Restart },

new Shading()

{

Val = ShadingPatternValues.Clear,

Fill = "f0f0f0",

Color = "Auto"

}));

else

tableCell.Append(new TableCellProperties(

new HorizontalMerge() { Val = MergedCellValues.Continue },

new Shading()

{

Val = ShadingPatternValues.Clear,

Fill = "f0f0f0",

Color = "Auto"

}));

if (data[i][0] == "Методы")

didMethodsStart = true;

}

if (!didMethodsStart && j == 3)

{

tableCell.Append(new TableCellProperties(new HorizontalMerge() { Val = MergedCellValues.Restart }));

}

else if (!didMethodsStart && j == 4)

{

tableCell.Append(new TableCellProperties(new HorizontalMerge() { Val = MergedCellValues.Continue }));

}

WordParagraph p = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties(new Justification()

{ Val = CheckIfHeaderRow(data, i) ? JustificationValues.Left : JustificationValues.Center },

new SpacingBetweenLines()

{

After = "0"

});

p.Append(pp);

Run run = new Run();

RunProperties rp = new RunProperties(new RunFonts()

{

HighAnsi = "Times New Roman",

Ascii = "Times New Roman"

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize }

};

run.Append(rp);

Text text = new Text(data[i][j]);

run.Append(text);

p.Append(run);

tableCell.Append(p);

tableRow.Append(tableCell);

}

wordTable.Append(tableRow);

}

return wordTable;

}

private bool CheckIfHeaderRow(string[][] data, int i)

{

return data[i][0] == "Поля" || data[i][0] == "Свойства" || data[i][0] == "Методы";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

using ApplicationLib.Word.Containers;

namespace ApplicationLib.Word.Commands

{

public class SecondPageCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ICommandsContainer CommandsContainer { get; private set; }

public SecondPageCommand(WordprocessingDocument wordDocument)

{

CommandsContainer = new CommandsContainer();

WordDocument = wordDocument;

}

public void Render()

{

CreateCommandsList();

CommandsContainer.Render();

}

private void CreateCommandsList()

{

CommandsContainer.Refresh();

CommandsContainer.Add(new TitleTableCommand(WordDocument));

CommandsContainer.Add(new ApprovedParagraphCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 5));

CommandsContainer.Add(new DocumentTitleCommand(WordDocument));

CommandsContainer.Add(new DocumentNameCommand(WordDocument));

CommandsContainer.Add(new PagesCountCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 1));

CommandsContainer.Add(new EndOfPageCommand(WordDocument));

CommandsContainer.Add(new SectionPtrCommand(WordDocument));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class CenterParagraphCommand : IWordSecondaryCommand

{

public string Text { get; }

public CenterParagraphCommand(string text)

{

Text = text;

}

public OpenXmlCompositeElement GetElement()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "250",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "27" },

};

run.PrependChild(runProperties);

var wordText = new Text(Text);

run.Append(wordText);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

namespace ApplicationLib.Word.Commands

{

public class EmptyParagraphCommand : IWordSecondaryCommand

{

public OpenXmlCompositeElement GetElement()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification()

{

Val = JustificationValues.Center

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties();

run.PrependChild(runProperties);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

namespace ApplicationLib.Word.Commands

{

public class RightParagraphCommand : IWordSecondaryCommand

{

public string Text { get; }

public RightParagraphCommand(string text)

{

Text = text;

}

public OpenXmlCompositeElement GetElement()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Right },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "250",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "27" },

};

run.PrependChild(runProperties);

var wordText = new Text(Text);

run.Append(wordText);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

namespace ApplicationLib.Word.Commands

{

public class TabCommand : IWordSecondaryCommand

{

public OpenXmlCompositeElement GetElement()

{

Run run = new Run();

RunProperties runProperties = new RunProperties(

new RunFonts()

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

});

run.Append(runProperties);

run.Append(new TabChar());

return run;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

using ApplicationLib.Models;

namespace ApplicationLib.Word.Commands

{

public class ContentsTableCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ContentsTableCommand(WordprocessingDocument document)

{

WordDocument = document;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetTableOfContents());

}

private SdtBlock GetTableOfContents()

{

SdtBlock tableOfContents = new SdtBlock();

RunProperties tocRpr = new RunProperties(new RunFonts()

{ HighAnsi = RenderData.Obj.RenderSettings.FontFamily },

new Color() { Val = "auto" }, new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize });

SdtContentDocPartObject sdtContentDocPartObject = new SdtContentDocPartObject(

new DocPartGallery() { Val = "Table of Contents" }, new DocPartUnique());

SdtProperties sdtProperties = new SdtProperties(tocRpr, sdtContentDocPartObject);

tableOfContents.Append(sdtProperties);

tableOfContents.Append(new SdtEndCharProperties());

SdtContentBlock sdtContentBlock = new SdtContentBlock();

WordParagraph p = new WordParagraph();

ParagraphProperties ppr = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center }

};

p.Append(ppr);

RunProperties rpr = new RunProperties(new RunFonts()

{

Ascii = RenderData.Obj.RenderSettings.FontFamily,

HighAnsi = RenderData.Obj.RenderSettings.FontFamily

})

{

Bold = new Bold(),

Caps = new Caps(),

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize }

};

Run run = new Run();

run.Append(rpr);

Text text = new Text("Содержание");

run.Append(text);

p.Append(run);

sdtContentBlock.Append(p);

string index = "0";

foreach (Item item in RenderData.Obj.Document.Items)

{

index = (int.Parse(index) + 1).ToString();

UploadItemsToTableOfContents(item, sdtContentBlock, 0, index);

}

tableOfContents.Append(sdtContentBlock);

return tableOfContents;

}

private void UploadItemsToTableOfContents(Item item, SdtContentBlock sdtContentBlock,

int depth, string index)

{

AddTableOfContentsElement(sdtContentBlock, depth, index + ". " + item.Name);

string dopIndex = "0";

if (item.Items != null)

foreach (Item i in item.Items)

{

dopIndex = (int.Parse(dopIndex) + 1).ToString();

UploadItemsToTableOfContents(i, sdtContentBlock, depth + 1, index + "." + dopIndex);

}

}

private void AddTableOfContentsElement(SdtContentBlock sdtContentBlock, int depth,

string name)

{

WordParagraph p = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties(new ParagraphStyleId() { Val = "31" })

{

Indentation = new Indentation() { FirstLine = "262", Left = "0" },

Justification = new Justification() { Val = JustificationValues.Both }

};

RunFonts runFonts = new RunFonts()

{

Ascii = RenderData.Obj.RenderSettings.FontFamily,

HighAnsi = RenderData.Obj.RenderSettings.FontFamily

};

pp.Append(runFonts);

p.Append(pp);

for (int i = 0; i < depth; i++)

{

p.Append(new TabCommand().GetElement());

}

Run run = new Run();

RunProperties runProperties = new RunProperties(new RunFonts()

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

})

{

FontSize = new FontSize() { Val = RenderData.Obj.RenderSettings.DefaultTextSize }

};

if (depth == 0)

{

runProperties.Bold = new Bold();

}

Text text = new Text(name);

run.Append(runProperties);

run.Append(text);

p.Append(run);

run = new Run();

runProperties = new RunProperties(new RunFonts()

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

});

run.Append(runProperties);

run.Append(new PositionalTab()

{

Leader = AbsolutePositionTabLeaderCharValues.Dot,

Alignment = AbsolutePositionTabAlignmentValues.Right,

RelativeTo = AbsolutePositionTabPositioningBaseValues.Margin

});

p.Append(run);

run = new Run();

runProperties = new RunProperties(new RunFonts()

{

HighAnsi = RenderData.Obj.RenderSettings.FontFamily,

Ascii = RenderData.Obj.RenderSettings.FontFamily

});

run.Append(runProperties);

text = new Text("0");

run.Append(text);

p.Append(run);

sdtContentBlock.Append(p);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

using ApplicationLib.Word.Containers;

namespace ApplicationLib.Word.Commands

{

public class TableOfContentsPageCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ICommandsContainer CommandsContainer { get; private set; }

public TableOfContentsPageCommand(WordprocessingDocument document)

{

CommandsContainer = new CommandsContainer();

WordDocument = document;

}

public void Render()

{

CreateCommandsList();

CommandsContainer.Render();

}

private void CreateCommandsList()

{

CommandsContainer.Refresh();

CommandsContainer.Add(new ContentsTableCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 1));

CommandsContainer.Add(new EndOfPageCommand(WordDocument));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class ApprovalTableCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; set; }

public ApprovalTableCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetApprovalTable());

}

private WordTable GetApprovalTable()

{

WordTable table = new WordTable();

TableProperties tableProperties = new TableProperties()

{

TableJustification = new TableJustification() { Val = TableRowAlignmentValues.Center },

};

table.AppendChild(tableProperties);

TableRow tableRow = new TableRow();

TableCell firstCell = new TableCell();

firstCell.Append(new CenterParagraphCommand(" СОГЛАСОВАННО ").GetElement());

firstCell.Append(new CenterParagraphCommand(" Руководитель проекта ").GetElement());

firstCell.Append(new CenterParagraphCommand(" Должность должность должность ").GetElement());

firstCell.Append(new CenterParagraphCommand(" Должность должность должность ").GetElement());

firstCell.Append(new CenterParagraphCommand(" ").GetElement());

firstCell.Append(new CenterParagraphCommand($" /{RenderData.Obj.Documentation.TeamLeadName}/")

.GetElement());

firstCell.Append(new CenterParagraphCommand(" \"\_\_\_\"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_").GetElement());

tableRow.Append(firstCell);

TableCell middleCell = new TableCell(new TableCellProperties()

{

TableCellWidth = new TableCellWidth() { Width = "1000" }

});

middleCell.Append(new EmptyParagraphCommand().GetElement());

tableRow.Append(middleCell);

TableCell secondCell = new TableCell();

secondCell.Append(new CenterParagraphCommand(" УТВЕРЖДАЮ ").GetElement());

secondCell.Append(new CenterParagraphCommand(" Самый главный руководитель проекта ").GetElement());

secondCell.Append(new CenterParagraphCommand(" Должность должность должность ").GetElement());

secondCell.Append(new CenterParagraphCommand(" Должность должность должность ").GetElement());

secondCell.Append(new CenterParagraphCommand(" ").GetElement());

secondCell.Append(new CenterParagraphCommand($" /{RenderData.Obj.Documentation.ManagerName}/")

.GetElement());

secondCell.Append(new CenterParagraphCommand(" \"\_\_\_\"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_").GetElement());

tableRow.Append(secondCell);

table.Append(tableRow);

return table;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class ApprovedParagraphCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ApprovedParagraphCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetApprovedParagraph());

WordDocument.MainDocumentPart.Document.Body.Append(GetNumberParagraph());

}

private WordParagraph GetApprovedParagraph()

{

WordParagraph paragraph = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Left },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "250",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

Run run = new Run();

RunProperties runProperties = new RunProperties(

new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Caps = new Caps(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "27" }

};

run.PrependChild(runProperties);

Text text = new Text("Утвержден") { Space = SpaceProcessingModeValues.Preserve };

run.Append(text);

paragraph.Append(run);

return paragraph;

}

private WordParagraph GetNumberParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Left },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "250",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(

new RunFonts

{ HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily) })

{

Caps = new Caps(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "27" }

};

run.PrependChild(runProperties);

var text = new Text(RenderData.Obj.Documentation.ProjectCode);

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class DocumentNameCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public DocumentNameCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetDocumentNameParagraph());

}

private WordParagraph GetDocumentNameParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily),

Ascii = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Bold = new Bold(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "30" }

};

run.Append(runProperties);

var text = new Text(RenderData.Obj.Document.Name);

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class DocumentNumberCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public DocumentNumberCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetDocumentNumberParagraph());

}

private WordParagraph GetDocumentNumberParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(

new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Bold = new Bold(),

Caps = new Caps(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "30" }

};

run.Append(runProperties);

var text = new Text(RenderData.Obj.Documentation.ProjectCode);

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class DocumentTitleCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public DocumentTitleCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetDocumentTitleParagraph());

}

private WordParagraph GetDocumentTitleParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily),

Ascii = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Bold = new Bold(),

Caps = new Caps(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "35" },

};

run.PrependChild(runProperties);

var text = new Text(RenderData.Obj.Documentation.ProjectName);

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class OrganizationNameCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public OrganizationNameCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetOrganizationHeaderParagraph());

}

private WordParagraph GetOrganizationHeaderParagraph()

{

WordParagraph paragraph = new WordParagraph();

ParagraphProperties pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

};

paragraph.Append(pp);

Run run = new Run();

RunProperties runProperties = new RunProperties(

new RunFonts { HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily) })

{

Bold = new Bold(),

Caps = new Caps(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "27" }

};

run.PrependChild(runProperties);

Text text = new Text("ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ") { Space = SpaceProcessingModeValues.Preserve };

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class PagesCountCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public PagesCountCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetListCountParagraph());

}

private WordParagraph GetListCountParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center }

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(

new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Bold = new Bold(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "25" }

};

run.Append(runProperties);

var text = new Text("Листов <количесво листов>");

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class SoftwareEngineerSignatureCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public SoftwareEngineerSignatureCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

Body body = WordDocument.MainDocumentPart.Document.Body;

body.Append(new RightParagraphCommand(" Исполнитель").GetElement());

body.Append(new RightParagraphCommand(" <Введите свою должность>").GetElement());

body.Append(new RightParagraphCommand(" /<Введите своё имя>/").GetElement());

body.Append(new EmptyParagraphCommand().GetElement());

body.Append(new RightParagraphCommand(" \"\_\_\_\"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019").GetElement());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class TitlePageTitleCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public TitlePageTitleCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetTitlePageTitleParagraph());

}

private WordParagraph GetTitlePageTitleParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

SpacingBetweenLines = new SpacingBetweenLines()

{

Before = "100",

After = "100",

Line = "300",

LineRule = LineSpacingRuleValues.Exact

}

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(

new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Bold = new Bold(),

Caps = new Caps(),

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "30" }

};

run.Append(runProperties);

var text = new Text("Лист утверждения");

run.Append(text);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class TitleTableCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public TitleTableCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetTitlePageTable());

}

private WordTable GetTitlePageTable()

{

WordTable table = new WordTable();

TableProperties tableProps = new TableProperties(

new TableBorders(

new TopBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 5

},

new BottomBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 5

},

new LeftBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 5

},

new RightBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 5

},

new InsideHorizontalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 5

},

new InsideVerticalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 5

}))

{

TablePositionProperties = new TablePositionProperties()

{

TablePositionX = 256,

TablePositionY = 1097,

HorizontalAnchor = HorizontalAnchorValues.Page,

VerticalAnchor = VerticalAnchorValues.Page,

RightFromText = 180,

LeftFromText = 180

}

};

table.AppendChild(tableProps);

for (int i = 0; i < 5; i++)

{

TableRow tableRow = new TableRow(new TableRowProperties(

new TableRowHeight() { Val = (uint)(i == 4 ? 2600 : 2000) }));

for (int j = 0; j < 2; j++)

{

TableCell tableCell = new TableCell();

var p = new WordParagraph();

var pprops = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center }

};

p.Append(pprops);

var r = new Run();

var rp = new RunProperties()

{

Italic = new Italic(),

};

var t = new Text(RenderData.Obj.RenderSettings.TitlePageTable[j][i]);

r.Append(rp);

r.Append(t);

p.Append(r);

tableCell.Append(new TableCellProperties(

new TableCellWidth { Width = "3000" })

{

TextDirection = new TextDirection() { Val = TextDirectionValues.BottomToTopLeftToRight },

TableCellVerticalAlignment = new TableCellVerticalAlignment()

{ Val = TableVerticalAlignmentValues.Center },

TableCellWidth = new TableCellWidth() { Width = "529", Type = TableWidthUnitValues.Dxa },

});

tableCell.Append(p);

tableRow.Append(tableCell);

}

table.Append(tableRow);

}

return table;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class TownAndYearCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public TownAndYearCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetTownAndYearParagraph());

}

private WordParagraph GetTownAndYearParagraph()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

};

paragraph.Append(pp);

var run = new Run();

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(RenderData.Obj.RenderSettings.FontFamily)

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "30" }

};

run.PrependChild(runProperties);

var wordText = new Text("Москва, 2019");

run.Append(wordText);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

using ApplicationLib.Word.Containers;

namespace ApplicationLib.Word.Commands

{

public class TitlePageCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public ICommandsContainer CommandsContainer { get; private set; }

public TitlePageCommand(WordprocessingDocument document)

{

WordDocument = document;

CommandsContainer = new CommandsContainer();

}

public void Render()

{

CreateCommandsList();

CommandsContainer.Render();

}

private void CreateCommandsList()

{

CommandsContainer.Refresh();

if (RenderData.Obj.RenderSettings.AddTitlePage)

CommandsContainer.Add(new TitleTableCommand(WordDocument));

CommandsContainer.Add(new OrganizationNameCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 1));

CommandsContainer.Add(new ApprovalTableCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 2));

CommandsContainer.Add(new DocumentTitleCommand(WordDocument));

CommandsContainer.Add(new DocumentNameCommand(WordDocument));

CommandsContainer.Add(new TitlePageTitleCommand(WordDocument));

CommandsContainer.Add(new DocumentNumberCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 3));

CommandsContainer.Add(new SoftwareEngineerSignatureCommand(WordDocument));

CommandsContainer.Add(new EmptyParagraphsCommand(WordDocument, 5));

CommandsContainer.Add(new TownAndYearCommand(WordDocument));

CommandsContainer.Add(new EndOfPageCommand(WordDocument));

CommandsContainer.Add(new SectionPtrCommand(WordDocument));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

namespace ApplicationLib.Word.Commands

{

public class EmptyParagraphsCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

private int NumberOfParagraphs { get; }

public EmptyParagraphsCommand(WordprocessingDocument wordDocument, int numberOfParagraphs)

{

WordDocument = wordDocument;

NumberOfParagraphs = numberOfParagraphs;

}

public void Render()

{

for (int i = 0; i<NumberOfParagraphs; i++)

{

WordDocument.MainDocumentPart.Document.Body.Append(new

EmptyParagraphCommand().GetElement());

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class EndOfPageCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public EndOfPageCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetLastParagraphOfThePage());

}

public WordParagraph GetLastParagraphOfThePage ()

{

var paragraph = new WordParagraph();

var pp = new ParagraphProperties()

{

Justification = new Justification() { Val = JustificationValues.Center },

};

paragraph.Append(pp);

var run = new Run(new LastRenderedPageBreak(), new Break() { Type = BreakValues.Page });

var runProperties = new RunProperties(new RunFonts

{

HighAnsi = new StringValue(

RenderData.Obj.RenderSettings.FontFamily)

})

{

Color = new Color() { Val = RenderData.Obj.RenderSettings.DefaultColor },

FontSize = new FontSize() { Val = "30" }

};

run.PrependChild(runProperties);

var wordText = new Text();

run.Append(wordText);

paragraph.Append(run);

return paragraph;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Commands

{

public class SectionPtrCommand : IWordCommand

{

public WordprocessingDocument WordDocument { get; }

public SectionPtrCommand(WordprocessingDocument wordDocument)

{

WordDocument = wordDocument;

}

public void Render()

{

WordDocument.MainDocumentPart.Document.Body.Append(GetSectionPtr());

}

public OpenXmlCompositeElement GetSectionPtr() => new WordParagraph(

new ParagraphProperties(

new SectionProperties()));

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Word.Interfaces;

using DocumentFormat.OpenXml.Packaging;

namespace ApplicationLib.Word.Containers

{

public class CommandsContainer : ICommandsContainer

{

public IList<IWordCommand> Commands { get; set; } = new List<IWordCommand>();

public void Add(IWordCommand command)

{

Commands.Add(command);

}

public void Refresh()

{

Commands = new List<IWordCommand>();

}

public void Remove(IWordCommand command)

{

Commands.Remove(command);

}

public void Render()

{

foreach (IWordCommand command in Commands)

{

command.Render();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Word.Interfaces

{

public interface ICommandsContainer : IWordCommand

{

IList<IWordCommand> Commands { get; set; }

void Add(IWordCommand command);

void Remove(IWordCommand command);

void Refresh();

}

}

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Word.Interfaces

{

public interface IDocumentRenderer

{

Task Render();

void SetRenderParams(RenderSettings renderSettings, Document document,

Documentation documentation);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

namespace ApplicationLib.Word.Interfaces

{

/// <summary>

/// IWordCommand is used to create a part of a document like tables, paragraphs and so on.

/// IWordCommand can be used singly or it can be used in a IWordMainCommand

/// </summary>

public interface IWordCommand

{

void Render();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ApplicationLib.Models;

using ApplicationLib.Word;

namespace ApplicationLib.Word.Interfaces

{

public interface IWordRenderer : IDocumentRenderer

{

ICommandsContainer CommandsContainer { get; set; }

}

}

using DocumentFormat.OpenXml;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Word.Interfaces

{

/// <summary>

/// The secondary command is used in IWordCommand to get small elements, like a Tab run and so on

/// </summary>

public interface IWordSecondaryCommand

{

OpenXmlCompositeElement GetElement();

}

}

using ApplicationLib.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Word

{

public class RenderData

{

public RenderSettings RenderSettings { get; private set; }

public Document Document { get; private set; }

public Documentation Documentation { get; private set; }

#warning add to tables

//id for numbered lists in the document

public int CurrentNumID { get; set; } = 0;

#region Singleton

private static RenderData renderData;

public static RenderData Obj

{

get

{

if (renderData == null)

renderData = new RenderData();

return renderData;

}

}

#endregion

#region Constructors

private RenderData() { }

#endregion

public static void UpdateData(RenderSettings renderSettings, Document document,

Documentation documentation)

{

Obj.RenderSettings = new RenderSettings(renderSettings);

Obj.Document = document;

Obj.Documentation = documentation;

Obj.CurrentNumID = 0;

Obj.RenderSettings.TitlePageTable[1][4] = Obj.Documentation.ProjectCode;

Obj.RenderSettings.FooterTable[2][0] = Obj.Documentation.ProjectCode;

Obj.RenderSettings.DefaultTextSize = (int.Parse(Obj.RenderSettings.DefaultTextSize)

\* 2).ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ApplicationLib.Word

{

public class RenderSettings

{

#region Properties

public string FontFamily { get; set; }

public bool AddFooter { get; set; }

public bool AddHeader { get; set; }

public string DefaultTextSize { get; set; }

public string DefaultColor { get; set; }

public bool AddLeftTable { get; set; }

public bool AddTitlePage { get; set; }

public bool AddSecondPage { get; set; }

public string FolderPath { get; set; }

#endregion

#region Word render constants

public double TabValue { get; } = 400;

public string[][] FooterTable { get; } = new string[4][]

{

new string[5]{ "", "", "", "", "" },

new string[5]{ "Изм", "Лист.", "№ Документа", "Подпись", "Дата"},

new string[5]{ "RU.17701729.04.03-01 ТЗ", "", "", "", "" },

new string[5]{ "Инв. № подл.", "Подп. и дата", "Взам. Инв №", "Инв. № дубл", "Подп и дата"}

};

public string[][] TitlePageTable { get; } = new string[2][]

{

new string[5]{"Инв. № подл", "Подп и дата", "Взаим инв №", "Инв № дубл", "Подп и дата"}.Reverse().ToArray(),

new string[5]{ "RU.17701729.04.03-01 ТЗ", "", "", "", "" }.Reverse().ToArray()

};

#endregion

#region Constructors

public RenderSettings() { }

public RenderSettings(RenderSettings settings)

{

FontFamily = settings.FontFamily;

AddFooter = settings.AddFooter;

AddHeader = settings.AddHeader;

DefaultTextSize = settings.DefaultTextSize;

DefaultColor = settings.DefaultColor;

AddLeftTable = settings.AddLeftTable;

AddTitlePage = settings.AddTitlePage;

AddSecondPage = settings.AddSecondPage;

FolderPath = settings.FolderPath;

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

using System.IO;

using ApplicationLib.Models;

using ApplicationLib.Interfaces;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using WordParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using WordTable = DocumentFormat.OpenXml.Wordprocessing.Table;

using WordDocument = DocumentFormat.OpenXml.Wordprocessing.Document;

using ApplicationLib.Word.Interfaces;

using ApplicationLib.Word.Containers;

using ApplicationLib.Word.Commands;

namespace ApplicationLib.Word

{

public class WordRenderer : IWordRenderer

{

public ICommandsContainer CommandsContainer { get; set; }

private WordprocessingDocument WordDocument { get; set; }

public WordRenderer()

{

CommandsContainer = new CommandsContainer();

}

public void SetRenderParams(RenderSettings renderSettings, Models.Document document,

Documentation documentation)

{

RenderData.UpdateData(renderSettings, document, documentation);

}

#region IDocumentRenderer

public async Task Render()

{

await Task.Run(() =>

{

string filePath = Path.Combine(RenderData.Obj.RenderSettings.FolderPath,

RenderData.Obj.Document.Name + ".docx");

if (File.Exists(filePath))

File.Delete(filePath);

using (WordDocument = WordprocessingDocument.Create(

filePath, WordprocessingDocumentType.Document, true))

{

WordDocument.AddMainDocumentPart();

WordDocument.MainDocumentPart.Document = new WordDocument

{

Body = new Body()

};

SetDocumentCommands();

CommandsContainer.Render();

}

using (WordDocument = WordprocessingDocument.Open(filePath, true))

{

SetFooterAndHeaderCommands();

CommandsContainer.Render();

}

});

}

#endregion

private void SetFooterAndHeaderCommands()

{

CommandsContainer.Refresh();

if (RenderData.Obj.RenderSettings.AddFooter)

CommandsContainer.Add(new FooterCommand(WordDocument));

if (RenderData.Obj.RenderSettings.AddHeader)

CommandsContainer.Add(new HeaderCommand(WordDocument));

}

private void SetDocumentCommands()

{

CommandsContainer.Refresh();

if (RenderData.Obj.RenderSettings.AddTitlePage)

CommandsContainer.Add(new TitlePageCommand(WordDocument));

if (RenderData.Obj.RenderSettings.AddSecondPage)

CommandsContainer.Add(new SecondPageCommand(WordDocument));

CommandsContainer.Add(new TableOfContentsPageCommand(WordDocument));

CommandsContainer.Add(new ItemsCommand(WordDocument));

}

}

}

* 1. SDWPApi.dll

using System.Web;

using System.Web.Optimization;

namespace SDWPApi

{

public class BundleConfig

{

// Дополнительные сведения об объединении см. на странице https://go.microsoft.com/fwlink/?LinkId=301862

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

// Используйте версию Modernizr для разработчиков, чтобы учиться работать. Когда вы будете готовы перейти к работе,

// готово к выпуску, используйте средство сборки по адресу https://modernizr.com, чтобы выбрать только необходимые тесты.

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"));

}

}

}

using System.Web;

using System.Web.Mvc;

namespace SDWPApi

{

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

namespace SDWPApi

{

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

using SDWPApi.Attributes;

namespace SDWPApi

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "sdwpapi/v1.0.0/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

//config.Filters.Add(new LoggerAttribute());

config.Filters.Add(new BasicAuthorizationAttribute());

}

}

}

using System;

using System.Net;

using System.Net.Http;

using System.Security.Principal;

using System.Text;

using System.Threading;

using System.Web.Http;

using System.Web.Http.Controllers;

using System.Web.Http.Filters;

namespace SDWPApi.Attributes

{

public class BasicAuthorizationAttribute : AuthorizationFilterAttribute

{

private string Login { get; } = "sdwpapimainuser";

private string Password { get; } = "dihfsodgoias;pdlvknkdslnvasoifjklfnsldafjsdlfa";

public override void OnAuthorization(HttpActionContext actionContext)

{

var authHeader = actionContext.Request.Headers.Authorization;

if (authHeader != null)

{

var authToken = actionContext.Request.Headers.Authorization.Parameter;

var decodedAuthToken = System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(authToken));

var authData = decodedAuthToken.Split(':');

var login = authData[0];

var password = authData[1];

if (login == Login && password == Password)

{

var principal = new GenericPrincipal(new GenericIdentity(login), null);

Thread.CurrentPrincipal = principal;

return;

}

}

actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);

}

}

}

using System;

using System.Net;

using System.Net.Http;

using System.Security.Principal;

using System.Text;

using System.Threading;

using System.Web.Http;

using System.Web.Http.Controllers;

using System.Web.Http.Filters;

using SDWPApi.Models;

using SDWPApi.Databases;

using AeroORMFramework;

namespace SDWPApi.Attributes

{

public class LoggerAttribute : ActionFilterAttribute

{

public override void OnActionExecuted(HttpActionExecutedContext actionExecutedContext)

{

RequestLog requestLog = GetRequestLog(actionExecutedContext);

Connector connector = new Connector(DatabaseProperties.ConnectionString);

connector.Insert(requestLog);

}

private RequestLog GetRequestLog(HttpActionExecutedContext context)

{

return new RequestLog(context);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using System.Web.Http;

using AeroORMFramework;

using Newtonsoft.Json;

using SDWPApi.Interfaces;

using SDWPApi.Models;

using SDWPApi.Databases;

using SDWPApi.Factories;

using SDWPApi.Attributes;

namespace SDWPApi.Controllers

{

[Logger]

[BasicAuthorization]

public class DocumentationsController : ApiController

{

#region Databases and factories

private IAbstractFactory DatabaseAbstractFactory { get; }

private IDatabase Database { get; }

private IHttpHelper HttpHelper { get; }

#endregion

public DocumentationsController()

{

DatabaseAbstractFactory = new AbstractFactory();

HttpHelper = DatabaseAbstractFactory.GetHttpHelper();

Database = DatabaseAbstractFactory.GetDatabase(DatabaseProperties.ConnectionString);

}

[HttpGet]

public async Task<HttpResponseMessage> GetAllDocumentations()

{

return await Task.Run(async () =>

{

try

{

IEnumerable<Documentation> documentations = await Database.GetAllRecords

<Documentation>();

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(documentations),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

});

}

[HttpGet]

public async Task<HttpResponseMessage> GetUserDocumentation(int userID)

{

return await Task.Run(async () =>

{

try

{

IEnumerable<Documentation> documentations = await Database.GetRecords

<Documentation>("AuthorID", userID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(documentations),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

});

}

[HttpPut]

public async Task<HttpResponseMessage> UpdateDocumentationRecord()

{

try

{

string body = await HttpHelper.GetRequestBody(Request);

Documentation documentation = JsonConvert.DeserializeObject<Documentation>(body);

if ((await Database.GetRecords<Documentation>("ID", documentation.ID)).ToList().Count != 1 ||

documentation == null)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.UpdateRecord(documentation);

documentation = await Database.GetRecord<Documentation>("ID", documentation.ID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(documentation),

System.Text.Encoding.UTF8, "application/json")

};

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPost]

public async Task<HttpResponseMessage> InsertNewDocumentation()

{

try

{

string requestBody = await HttpHelper.GetRequestBody(Request);

Documentation documentation = JsonConvert.DeserializeObject

<Documentation>(requestBody);

if (documentation == null)

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

await Database.InsertRecord(documentation);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpDelete]

public async Task<HttpResponseMessage> Delete(int documentationID)

{

try

{

if ((await Database.GetRecords<Documentation>("ID", documentationID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.DeleteRecord(new Documentation() { ID = documentationID });

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using System.Web.Http;

using AeroORMFramework;

using Newtonsoft.Json;

using SDWPApi.Models;

using SDWPApi.Interfaces;

using SDWPApi.Factories;

using SDWPApi.Databases;

using SDWPApi.Attributes;

namespace SDWPApi.Controllers

{

[Logger]

[BasicAuthorization]

public class DocumentsController : ApiController

{

#region Databases and factories

private IAbstractFactory DatabaseAbstractFactory { get; }

private IDatabase Database { get; }

private IHttpHelper HttpHelper { get; }

#endregion

public DocumentsController()

{

DatabaseAbstractFactory = new AbstractFactory();

HttpHelper = DatabaseAbstractFactory.GetHttpHelper();

Database = DatabaseAbstractFactory.GetDatabase(DatabaseProperties.ConnectionString);

}

[HttpGet]

public async Task<HttpResponseMessage> GetAllDocuments()

{

try

{

IEnumerable<Document> documents = await Database.GetAllRecords<Document>();

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(documents), System.Text.Encoding.UTF8,

"application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpGet]

public async Task<HttpResponseMessage> GetDocumentationDocuments(int documentationID)

{

try

{

IEnumerable<Document> documents = await Database.GetRecords<Document>("DocumentationID", documentationID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(documents), System.Text.Encoding.UTF8,

"application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPost]

public async Task<HttpResponseMessage> InsertDocument()

{

try

{

string requestBody = await HttpHelper.GetRequestBody(Request);

Document document = JsonConvert.DeserializeObject<Document>(requestBody);

if (document == null)

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

await Database.InsertRecord(document);

return new HttpResponseMessage(System.Net.HttpStatusCode.Created);

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPut]

public async Task<HttpResponseMessage> UpdateDocument()

{

try

{

string requestBody = await HttpHelper.GetRequestBody(Request);

Document document = JsonConvert.DeserializeObject<Document>(requestBody);

if (document == null ||

(await Database.GetRecords<Document>("ID", document.ID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.UpdateRecord(document);

document = await Database.GetRecord<Document>("ID", document.ID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(document), System.Text.Encoding.UTF8,

"application/json")

};

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpDelete]

public async Task<HttpResponseMessage> DeleteDocument(int documentID)

{

try

{

if ((await Database.GetRecords<Document>("ID", documentID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.DeleteRecord(new Document { ID = documentID });

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using System.Web.Http;

using AeroORMFramework;

using Newtonsoft.Json;

using SDWPApi.Models;

using SDWPApi.Interfaces;

using SDWPApi.Factories;

using SDWPApi.Databases;

using SDWPApi.Attributes;

namespace SDWPApi.Controllers

{

[Logger]

[BasicAuthorization]

public class EmailCodesController : ApiController

{

private static Random Random { get; } = new Random();

#region Databases and factories

private IAbstractFactory DatabaseAbstractFactory { get; }

private IDatabase Database { get; }

private IEmailDatabase EmailDatabase { get; }

#endregion

public EmailCodesController()

{

DatabaseAbstractFactory = new AbstractFactory();

Database = new SDWPApi.Databases.Database(DatabaseProperties.ConnectionString);

EmailDatabase = DatabaseAbstractFactory.GetEmailDatabase();

}

[HttpGet]

public async Task<HttpResponseMessage> SendCode(int userID, string email)

{

try

{

IList<UserInfo> users = (await Database.GetRecords<UserInfo>("ID", userID)).ToList();

string code = EmailDatabase.GetNewCode();

await EmailDatabase.SendCodeEmail(email, code);

EmailCode emailCode = GetEmailCodeObject(code, userID);

int codeID = await Database.InsertRecordAndGetID(emailCode);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(codeID.ToString(), System.Text.Encoding.UTF8,

"plain/text")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

private EmailCode GetEmailCodeObject(string code, int userID)

{

return new EmailCode()

{

Code = code,

CreationDate = DateTime.Now,

Used = false,

UserID = userID

};

}

[HttpGet]

public async Task<HttpResponseMessage> CheckCode(int codeID, string code)

{

try

{

List<EmailCode> emailCodes = (await Database.GetRecords<EmailCode>("ID", codeID)).ToList();

if (emailCodes.Count == 0 || (emailCodes.Count == 1 && emailCodes[0].Used == true ||

emailCodes[0].Code != code))

{

return new HttpResponseMessage(System.Net.HttpStatusCode.NoContent);

}

else if (emailCodes.Count == 1 && emailCodes[0].Code == code && emailCodes[0].Used == false)

{

emailCodes[0].Used = true;

await Database.UpdateRecord(emailCodes[0]);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

else

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpDelete]

public async Task<HttpResponseMessage> DeleteCode(int codeID)

{

try

{

if ((await Database.GetRecords<EmailCode>("ID", codeID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.DeleteRecord<EmailCode>(new EmailCode() { ID = codeID });

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPut]

public async Task<HttpResponseMessage> SendChangePassReference(int userID)

{

try

{

if ((await Database.GetRecords<UserInfo>("ID", userID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

var passChange = GetPassChangeObject(userID);

await Database.InsertRecord(passChange);

string link = $"https://aerothedeveloper.ru/Password/ResetPassword?userID={userID}&hash=" +

$"{passChange.AccessHash}";

List<UserInfo> users = (await Database.GetRecords<UserInfo>("ID", userID)).ToList();

await EmailDatabase.SendChangePassLink(users[0].Email, link);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

private PasswordChange GetPassChangeObject(int userID)

{

return new PasswordChange()

{

UserID = userID,

AccessHash = GetPassChangeHash(),

Closed = false,

CreationDate = DateTime.Now

};

}

private string GetPassChangeHash()

{

string str = string.Empty;

for (int i = 0; i < 100; i++)

{

str += (char)Random.Next('A', 'Z' + 1);

}

return str.GetHashCode().ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using System.Web.Http;

using AeroORMFramework;

using Newtonsoft.Json;

using SDWPApi.Models;

using SDWPApi.Interfaces;

using SDWPApi.Factories;

using SDWPApi.Databases;

using SDWPApi.Attributes;

using System.Web.Mvc;

namespace SDWPApi.Controllers

{

public class PassChangeContext

{

public string OldPass { get; set; }

public string NewPass { get; set; }

public string PassConfirmation { get; set; }

}

public class PasswordController : Controller

{

#region Databases and factories

private IAbstractFactory DatabaseAbstractFactory { get; }

private IEmailDatabase EmailDatabase { get; }

private IDatabase Database { get; }

#endregion

public PasswordController()

{

DatabaseAbstractFactory = new AbstractFactory();

EmailDatabase = DatabaseAbstractFactory.GetEmailDatabase();

Database = DatabaseAbstractFactory.GetDatabase(DatabaseProperties.ConnectionString);

}

#region Reset password

public async Task<ActionResult> ResetPassword(int userID, string hashCode)

{

List<PasswordChange> passChanges = (await Database.GetRecords<PasswordChange>

("UserID", userID)).ToList();

List<PasswordChange> suitablePassChanges = (passChanges.

Where(pc => pc.Closed == false)).ToList();

await DeleteOldPassChanges(suitablePassChanges, hashCode);

if (suitablePassChanges.Count == 0)

return null;

SetTempData(userID, suitablePassChanges.First().ID);

return View();

}

private async Task DeleteOldPassChanges(List<PasswordChange> suitablePassChanges,

string hashCode)

{

if (suitablePassChanges.Count != 1)

{

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

{

if (suitablePassChanges[i].AccessHash != hashCode)

{

suitablePassChanges[i].Closed = true;

await Database.UpdateRecord(suitablePassChanges[i]);

suitablePassChanges.Remove(suitablePassChanges[i]);

}

}

}

}

private void SetTempData(int userID, int passChangeID)

{

TempData["UserID"] = userID;

TempData["PassChangeID"] = passChangeID;

}

#endregion

#region Update password

[System.Web.Http.HttpPut]

public async Task<ActionResult> UpdatePassword(PassChangeContext context)

{

try

{

if (!CheckSessionParams())

return null;

List<UserInfo> users = (await Database.GetRecords<UserInfo>("ID",

(int)TempData["UserID"])).ToList();

if (users.Count != 1 || IsPassChangeContextWrong(context, users[0]))

{

ViewBag.Result = "Непраивльно введены данные";

return View();

}

await UpdatePasswordChangeStatus();

await UpdateUserPass(users[0], context);

ViewBag.Result = "Пароль успешно изменен";

return View();

}

catch (Exception)

{

ViewBag.Result = "Ошибка на сервере";

return View();

}

}

private async Task UpdateUserPass(UserInfo user, PassChangeContext context)

{

user.Password = context.NewPass.GetHashCode().ToString();

await Database.UpdateRecord(user);

}

private async Task UpdatePasswordChangeStatus()

{

List<PasswordChange> passwordChanges = (await Database.GetRecords<PasswordChange>("ID",

(int)TempData["PassChangeID"])).ToList();

passwordChanges[0].Closed = true;

await Database.UpdateRecord(passwordChanges[0]);

}

private bool CheckSessionParams()

{

return (TempData.ContainsKey("UserID") && TempData.ContainsKey("PassChangeID"));

}

private bool IsPassChangeContextWrong(PassChangeContext context, UserInfo user)

{

string oldPass = context.OldPass;

string newPass = context.NewPass;

string passConfirmation = context.PassConfirmation;

return user.Password != oldPass.GetHashCode().ToString()

|| newPass != passConfirmation || string.IsNullOrEmpty(oldPass) ||

string.IsNullOrEmpty(newPass) || string.IsNullOrEmpty(passConfirmation) ||

string.IsNullOrWhiteSpace(newPass) || string.IsNullOrWhiteSpace(oldPass)

|| string.IsNullOrWhiteSpace(passConfirmation);

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using System.Web.Http;

using AeroORMFramework;

using Newtonsoft.Json;

using SDWPApi.Models;

using SDWPApi.Interfaces;

using SDWPApi.Factories;

using SDWPApi.Databases;

using SDWPApi.Attributes;

namespace SDWPApi.Controllers

{

[Logger]

[BasicAuthorization]

public class TemplatesController : ApiController

{

#region Databases and factories

private IAbstractFactory DatabaseAbstractFactory { get; }

private IDatabase Database { get; }

private IHttpHelper HttpHelper { get; }

#endregion

public TemplatesController()

{

DatabaseAbstractFactory = new AbstractFactory();

HttpHelper = DatabaseAbstractFactory.GetHttpHelper();

Database = DatabaseAbstractFactory.GetDatabase(DatabaseProperties.ConnectionString);

}

[HttpGet]

public async Task<HttpResponseMessage> GetAllTemplates()

{

try

{

IEnumerable<Template> templates = await Database.GetAllRecords<Template>();

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(templates),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

public async Task<HttpResponseMessage> GetUserTemplate(int userID)

{

try

{

IEnumerable<Template> templates = await Database.GetRecords<Template>

("UserID", userID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(templates),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPost]

public async Task<HttpResponseMessage> InsertNewTemplate()

{

try

{

string requestBody = await HttpHelper.GetRequestBody(Request);

Template template = JsonConvert.DeserializeObject<Template>(requestBody);

if (template == null)

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

await Database.InsertRecord(template);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPut]

public async Task<HttpResponseMessage> UpdateTemplate()

{

try

{

string requesBody = await HttpHelper.GetRequestBody(Request);

Template template = JsonConvert.DeserializeObject<Template>(requesBody);

if (template == null ||

(await Database.GetRecords<Template>("ID", template.ID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.UpdateRecord(template);

template = await Database.GetRecord<Template>("ID", template.ID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(template),

System.Text.Encoding.UTF8, "application/json")

};

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpDelete]

public async Task<HttpResponseMessage> DeleteTemplate(int templateID)

{

try

{

if ((await Database.GetRecords<Template>("ID", templateID)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.DeleteRecord<Template>(new Template { ID = templateID });

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using System.Web.Http;

using AeroORMFramework;

using Newtonsoft.Json;

using SDWPApi.Models;

using SDWPApi.Interfaces;

using SDWPApi.Factories;

using SDWPApi.Databases;

using SDWPApi.Attributes;

using System.Threading;

namespace SDWPApi.Controllers

{

[Logger]

[BasicAuthorization]

public class UsersController : ApiController

{

#region Databases and factories

private IAbstractFactory DatabaseAbstractFactory { get; }

private IDatabase Database { get; }

private IEmailDatabase EmailDatabase { get; }

private IHttpHelper HttpHelper { get; }

#endregion

public UsersController()

{

DatabaseAbstractFactory = new AbstractFactory();

HttpHelper = DatabaseAbstractFactory.GetHttpHelper();

EmailDatabase = DatabaseAbstractFactory.GetEmailDatabase();

Database = DatabaseAbstractFactory.GetDatabase(DatabaseProperties.ConnectionString);

}

[HttpGet]

public async Task<HttpResponseMessage> GetAllUsers()

{

try

{

IEnumerable<UserInfo> userInfos = await Database.GetAllRecords<UserInfo>();

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(userInfos),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpGet]

public async Task<HttpResponseMessage> GetUserByID(int id)

{

try

{

UserInfo userInfo = await Database.GetRecord<UserInfo>("ID", id);

if (userInfo.ID == 0)

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(userInfo),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpGet]

public async Task<HttpResponseMessage> GetUserByLoginAndPass(string login,

string pass)

{

try

{

UserInfo userInfo = await Database.GetRecord<UserInfo>("Login", login);

if (userInfo.Password == pass)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(userInfo),

System.Text.Encoding.UTF8, "application/json")

};

}

else

{

return new HttpResponseMessage(System.Net.HttpStatusCode.NoContent);

}

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPost]

public async Task<HttpResponseMessage> InsertUserRecord()

{

try

{

string requestBody = await HttpHelper.GetRequestBody(Request);

UserInfo userInfo = JsonConvert.DeserializeObject<UserInfo>(requestBody);

if (userInfo == null)

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

await Database.InsertRecord(userInfo);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch (JsonSerializationException)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpPut]

public async Task<HttpResponseMessage> UpdateUserRecord()

{

try

{

string bodyContent = await HttpHelper.GetRequestBody(Request);

UserInfo userInfo = await Task.Run(() => JsonConvert.DeserializeObject

<UserInfo>(bodyContent));

if (userInfo == null)

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

await Database.UpdateRecord(userInfo);

userInfo = await Database.GetRecord<UserInfo>("ID", userInfo.ID);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK)

{

Content = new StringContent(JsonConvert.SerializeObject(userInfo),

System.Text.Encoding.UTF8, "application/json")

};

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpDelete]

public async Task<HttpResponseMessage> DeleteUserRecord(int id)

{

try

{

if ((await Database.GetRecords<UserInfo>("ID", id)).ToList().Count != 1)

{

return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);

}

await Database.DeleteRecord(new UserInfo { ID = id });

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

#region Check methods

[HttpGet]

public async Task<HttpResponseMessage> CheckNewLogin(string login)

{

try

{

List<UserInfo> userInfos = (await Database.GetRecords<UserInfo>

("Login", login)).ToList();

if (userInfos.Count == 0)

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

else

{

return new HttpResponseMessage(System.Net.HttpStatusCode.NoContent);

}

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpGet]

public async Task<HttpResponseMessage> CheckNewEmail(string email)

{

try

{

List<UserInfo> userInfos = (await Database.GetRecords<UserInfo>

("Email", email)).ToList();

if (userInfos.Count == 0)

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

else

{

return new HttpResponseMessage(System.Net.HttpStatusCode.NoContent);

}

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

}

[HttpGet]

public async Task<HttpResponseMessage> RemindPass(string login, string email)

{

return await Task.Run(async () =>

{

try

{

if (string.IsNullOrWhiteSpace(login) || string.IsNullOrEmpty(login) ||

string.IsNullOrEmpty(email) || string.IsNullOrWhiteSpace(email))

{

return new HttpResponseMessage(System.Net.HttpStatusCode.NoContent);

}

List<UserInfo> users = (await Database.GetRecords<UserInfo>("Login", login)).

ToList();

if (users.Count != 1 || users[0].Email != email)

return new HttpResponseMessage(System.Net.HttpStatusCode.NoContent);

string newPass = EmailDatabase.GetNewPassword();

string passwordHash = newPass.GetHashCode().ToString();

users[0].Password = passwordHash;

await Database.UpdateRecord(users[0]);

await EmailDatabase.SendNewPassword(users[0].Email, newPass);

return new HttpResponseMessage(System.Net.HttpStatusCode.OK);

}

catch

{

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

});

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using SDWPApi.Interfaces;

using AeroORMFramework;

using SDWPApi.Models;

namespace SDWPApi.Databases

{

public class Database : IDatabase

{

public Connector Connector { get; }

public Database(string connectionString)

{

Connector = new Connector(connectionString);

}

public async Task DeleteRecord<T>(T entity) where T : new()

{

await Task.Run(() =>

{

Connector.DeleteRecord<T>(entity);

});

}

public async Task<IEnumerable<T>> GetAllRecords<T>() where T : new()

{

return await Task.Run(() =>

{

return Connector.GetAllRecords<T>();

});

}

public async Task<T> GetRecord<T>(string columnName, object value) where T : new()

{

return await Task.Run(() =>

{

return Connector.GetRecord<T>(columnName, value);

});

}

public async Task<IEnumerable<T>> GetRecords<T>(string columnName, object value)

where T : new()

{

return await Task.Run(() =>

{

return Connector.GetRecords<T>(columnName, value);

});

}

public async Task UpdateRecord<T>(T entity) where T : new()

{

await Task.Run(() =>

{

Connector.UpdateRecord(entity);

});

}

public async Task InsertRecord<T>(T entity) where T : new()

{

await Task.Run(() =>

{

Connector.Insert(entity);

});

}

public async Task<int> InsertRecordAndGetID<T>(T entity) where T : new()

{

return await Task.Run(() =>

{

return Connector.InsertAndGetID(entity);

});

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Databases

{

public static class DatabaseProperties

{

#region Connection Parameters

private static string ServerName { get; } = "31.31.196.199";

private static string InitialCatalog { get; } = "u0661866\_sdwpdb";

private static string IntegratedSecurity { get; } = "False";

private static string UserID { get; } = "u0661866\_AeroOne";

private static string Password { get; } = "fx6-Fym-fHC-vjf";

private static string TrustedConnection { get; } = "False";

private static string Encrypt { get; } = "False";

#endregion

public static string ConnectionString { get; } =

$"Data Source={ServerName}; Initial Catalog={InitialCatalog}; " +

$"Integrated Security={IntegratedSecurity}; User ID={UserID}; " +

$"Password={Password}; TrustServerCertificate={TrustedConnection}; Encrypt={Encrypt};";

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Mail;

using System.Threading.Tasks;

using System.Web;

using SDWPApi.Models;

using SDWPApi.Interfaces;

namespace SDWPApi.Databases

{

public class EmailDB : IEmailDatabase

{

private Random Random { get; set; } = new Random();

private string ServerMail { get; } = "sdwpemailservice@aerothedeveloper.ru";

private string ServerMailPass { get; } = "QRJ-6Cf-hDV-cqG";

private string Host { get; } = "mail.aerothedeveloper.ru";

public async Task SendCodeEmail(string email, string code)

{

await Task.Run(() =>

{

MailAddress senderAdress = new MailAddress(ServerMail);

MailAddress addresseeAdress = new MailAddress(email);

MailMessage mailMessage = new MailMessage(senderAdress, addresseeAdress)

{

Subject = "Code for SDWP",

Body = $"This is automatically-generated message, do not reply.\n\n" +

$"Your code is <h2>{code}</h2>\n\n Best wishes,\n SDWP team",

IsBodyHtml = true

};

SmtpClient smtpClient = new SmtpClient(Host, 25)

{

Credentials = new NetworkCredential(ServerMail, ServerMailPass),

EnableSsl = false

};

smtpClient.Send(mailMessage);

});

}

public string GetNewCode()

{

string code = string.Empty;

for (int i = 0; i < 6; i++)

{

code += (char)Random.Next('A', 'Z' + 1);

}

return code;

}

public async Task SendChangePassLink(string email, string link)

{

await Task.Run(() =>

{

MailAddress senderAdress = new MailAddress(ServerMail);

MailAddress addresseeAdress = new MailAddress(email);

MailMessage mailMessage = new MailMessage(senderAdress, addresseeAdress)

{

Subject = "Change pass link",

Body = $"This is automatically-generated message, do not reply\n\n" +

$"Your link is:\n <h2>{link}</h2>\n\n Best wishes,\n SDWP team",

IsBodyHtml = true

};

SmtpClient smtpClient = new SmtpClient(Host, 25)

{

Credentials = new NetworkCredential(ServerMail, ServerMailPass),

EnableSsl = false,

};

smtpClient.Send(mailMessage);

});

}

public async Task SendNewPassword(string email, string newPass)

{

await Task.Run(() =>

{

MailAddress senderAdress = new MailAddress(ServerMail);

MailAddress addresseeAdress = new MailAddress(email);

MailMessage mailMessage = new MailMessage(senderAdress, addresseeAdress)

{

Subject = "New password",

Body = $"This is automatically-generated message, do not reply\n\n" +

$"Your new password is:\n <h2>{newPass}<h2>\n\n Best wishes,\n SDWP team",

IsBodyHtml = true

};

SmtpClient smtpClient = new SmtpClient(Host, 25)

{

Credentials = new NetworkCredential(ServerMail, ServerMailPass),

EnableSsl = false

};

smtpClient.Send(mailMessage);

});

}

public string GetNewPassword()

{

string password = string.Empty;

for (int i = 0; i < 10; i++)

{

password += (char)Random.Next('A', 'Z' + 1);

}

return password;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using SDWPApi.Interfaces;

using SDWPApi.Databases;

using SDWPApi.Models;

using SDWPApi.Requests;

namespace SDWPApi.Factories

{

public class AbstractFactory : IAbstractFactory

{

public IDatabase GetDatabase(string connectionString)

{

return new Database(connectionString);

}

public IEmailDatabase GetEmailDatabase()

{

return new EmailDB();

}

public IHttpHelper GetHttpHelper()

{

return new HttpHelper();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using SDWPApi.Models;

using SDWPApi.Interfaces;

namespace SDWPApi.Factories

{

public interface IAbstractFactory

{

IDatabase GetDatabase(string connectionString);

IEmailDatabase GetEmailDatabase();

IHttpHelper GetHttpHelper();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AeroORMFramework;

using SDWPApi.Models;

namespace SDWPApi.Interfaces

{

public interface IDatabase

{

Connector Connector { get; }

Task<IEnumerable<T>> GetAllRecords<T>()

where T : new();

Task<IEnumerable<T>> GetRecords<T>(string columnName, object value)

where T : new();

Task<T> GetRecord<T>(string columName, object value)

where T : new();

Task UpdateRecord<T>(T entity) where T : new();

Task DeleteRecord<T>(T entity) where T : new();

Task InsertRecord<T>(T entity) where T : new();

Task<int> InsertRecordAndGetID<T>(T entity) where T : new();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SDWPApi.Interfaces

{

#warning change this in the tables

public interface IEmailDatabase

{

Task SendCodeEmail(string email, string code);

Task SendNewPassword(string email, string password);

string GetNewPassword();

string GetNewCode();

Task SendChangePassLink(string email, string link);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

namespace SDWPApi.Interfaces

{

public interface IHttpHelper

{

Task<string> GetRequestBody(HttpRequestMessage requestMsg);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using Newtonsoft.Json;

namespace SDWPApi.Models

{

public class NumberedList : ParagraphElement

{

[JsonProperty("elements")]

public List<NumberedListElement> ListElements { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class NumberedListElement

{

[JsonProperty("text")]

public string Text { get; set; }

[JsonProperty("index")]

public string Index { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

[JsonConverter(typeof(ParagraphJsonConverter))]

public class Paragraph

{

[JsonProperty("type")]

public string Type { get; set; }

[JsonProperty("element")]

public ParagraphElement ParagraphElement { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class ParagraphElement

{

[JsonProperty("hint")]

public string Hint { get; set; }

[JsonProperty("title")]

public string Title { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class ParagraphImage : ParagraphElement

{

[JsonProperty("source")]

public byte[] ImageSource { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class Subparagraph : ParagraphElement

{

[JsonProperty("text")]

public string Text { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class Table : ParagraphElement

{

[JsonProperty("tableCells")]

public string[][] TableCells { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public enum Access

{

Public = 1,

Private = 2

}

}

using AeroORMFramework;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class Document

{

[PrimaryKey, CanBeNull(false), AutoincrementID, JsonProperty("id")]

public int ID { get; set; }

[CanBeNull(false), JsonProperty("documentationID")]

public int DocumentationID { get; set; }

[CanBeNull(false), JsonProperty("name")]

public string Name { get; set; }

[CanBeNull(false), JsonProperty("authorID")]

public int AuthorID { get; set; }

[CanBeNull(false), JsonProperty("authorName")]

public string AuthorName { get; set; }

[CanBeNull(false), JsonProperty("creationDate")]

public DateTime CreationDate { get; set; }

[CanBeNull(false), JsonProperty("updatedAt")]

public DateTime UpdatedAt { get; set; }

[CanBeNull(false), SetAzureSQLDataType("int"), JsonProperty("access")]

public Access Access { get; set; }

[CanBeNull(false), Json, JsonProperty("items")]

public List<Item> Items { get; set; }

}

}

using AeroORMFramework;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class Documentation

{

[AutoincrementID, PrimaryKey, CanBeNull(false), JsonProperty("id")]

public int ID { get; set; }

[CanBeNull(false), JsonProperty("name")]

public string Name { get; set; }

[CanBeNull(false), JsonProperty("authorID")]

public int AuthorID { get; set; }

[CanBeNull(false), JsonProperty("authorName")]

public string AuthorName { get; set; }

[CanBeNull(false), JsonProperty("creationDate")]

public DateTime CreationDate { get; set; }

[CanBeNull(false), JsonProperty("updatedAt")]

public DateTime UpdatedAt { get; set; }

[CanBeNull(false), SetAzureSQLDataType("int"), JsonProperty("access")]

public Access Access { get; set; }

[CanBeNull(false), SetAzureSQLDataType("int"), JsonProperty("storageType")]

public StorageType StorageType { get; set; }

[CanBeNull(true), JsonProperty("projectName")]

public string ProjectName { get; set; }

[CanBeNull(true), JsonProperty("teamLeadName")]

public string TeamLeadName { get; set; }

[CanBeNull(true), JsonProperty("managerName")]

public string ManagerName { get; set; }

[CanBeNull(true), JsonProperty("projectCode")]

public string ProjectCode { get; set; }

[CanBeNull(true), JsonProperty("softwareEngineerName")]

public string SoftwareEngineerName { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class Item

{

[JsonProperty("name")]

public string Name { get; set; }

[JsonProperty("items")]

public List<Item> Items { get; set; }

[JsonProperty("paragraphs")]

public List<Paragraph> Paragraphs { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public enum StorageType

{

Local = 0,

Cloud = 1,

}

}

using AeroORMFramework;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class Template

{

[PrimaryKey, AutoincrementID, CanBeNull(false), JsonProperty("id")]

public int ID { get; set; }

[CanBeNull(false), JsonProperty("userID")]

public int UserID { get; set; }

[CanBeNull(false), JsonProperty("templateName")]

public string TemplateName { get; set; }

[CanBeNull(false), JsonProperty("createdAt")]

public DateTime CreatedAt { get; set; }

[CanBeNull(false), JsonProperty("updatedAt")]

public DateTime UpdatedAt { get; set; }

[CanBeNull(false), Json, JsonProperty("items")]

public List<Item> Items { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

using Newtonsoft.Json.Converters;

namespace SDWPApi.Models

{

class ParagraphJsonConverter : CustomCreationConverter<Paragraph>

{

public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)

{

var paragraph = new Paragraph();

while (reader.Read())

{

if (reader.TokenType == JsonToken.EndObject)

return paragraph;

if (reader.TokenType != JsonToken.PropertyName)

continue;

string propertyName = reader.Value.ToString();

if (!reader.Read())

return paragraph;

if ("type".Equals(propertyName))

{

paragraph.Type = reader.Value?.ToString();

}

else if ("element".Equals(propertyName))

{

switch (paragraph.Type)

{

case "Subparagraph":

paragraph.ParagraphElement = serializer.Deserialize<Subparagraph>(reader);

break;

case "Table":

paragraph.ParagraphElement = serializer.Deserialize<Table>(reader);

break;

case "NumberedList":

paragraph.ParagraphElement = serializer.Deserialize<NumberedList>(reader);

break;

case "ParagraphImage":

paragraph.ParagraphElement = serializer.Deserialize<ParagraphImage>(reader);

break;

}

}

}

return paragraph;

}

public override Paragraph Create(Type objectType)

{

return new Paragraph();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using AeroORMFramework;

namespace SDWPApi.Models

{

public class EmailCode

{

[PrimaryKey, AutoincrementID, CanBeNull(false)]

public int ID { get; set; }

[CanBeNull(false)]

public int UserID { get; set; }

[CanBeNull(false)]

public DateTime CreationDate { get; set; }

[CanBeNull(false)]

public string Code { get; set; }

[CanBeNull(false)]

public bool Used { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using AeroORMFramework;

namespace SDWPApi.Models

{

public class PasswordChange

{

[AutoincrementID, PrimaryKey, CanBeNull(false)]

public int ID { get; set; }

[CanBeNull(false)]

public int UserID { get; set; }

[CanBeNull(false)]

public string AccessHash { get; set; }

[CanBeNull(false)]

public bool Closed { get; set; }

[CanBeNull(false)]

public DateTime CreationDate { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Web;

using System.Web.Http.Filters;

using AeroORMFramework;

namespace SDWPApi.Models

{

public class RequestLog

{

[AutoincrementID, PrimaryKey, CanBeNull(false)]

public int ID { get; set; }

[CanBeNull(false)]

public DateTime RequestDate { get; set; }

[CanBeNull(false)]

public string ControllerName { get; set; }

[CanBeNull(false), Json]

public Uri Uri { get; set; }

[CanBeNull(false), Json]

public HttpContent Content { get; set; }

[CanBeNull(false)]

public string Method { get; set; }

[CanBeNull(false), Json]

public Version Version { get; set; }

[CanBeNull(false), Json]

public HttpHeaders Headers { get; set; }

public RequestLog() { }

public RequestLog(HttpActionExecutedContext context)

{

RequestDate = DateTime.Now;

ControllerName = context.ActionContext.ControllerContext.ControllerDescriptor.ControllerName;

Uri = context.Request.RequestUri;

Content = context.Request.Content;

Method = context.Request.Method.Method;

Version = context.Request.Version;

Headers = context.Request.Headers;

}

}

}

using AeroORMFramework;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace SDWPApi.Models

{

public class UserInfo

{

[CanBeNull(false)]

[PrimaryKey]

[AutoincrementID]

public int ID { get; set; }

[CanBeNull(false)]

public string Login { get; set; }

[CanBeNull(false)]

public string Password { get; set; }

[CanBeNull(false)]

public string Name { get; set; }

[CanBeNull(false)]

public string Surname { get; set; }

[CanBeNull(false)]

public DateTime BirthDate { get; set; }

[CanBeNull(false)]

public string Email { get; set; }

[CanBeNull(true), Json]

public byte[] UserPhoto { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web;

using SDWPApi.Interfaces;

namespace SDWPApi.Requests

{

public class HttpHelper : IHttpHelper

{

public async Task<string> GetRequestBody(HttpRequestMessage requestMsg)

{

string body;

using (var stream = await requestMsg.Content.ReadAsStreamAsync())

{

stream.Seek(0, SeekOrigin.Begin);

using (var sr = new StreamReader(stream))

{

body = await sr.ReadToEndAsync();

}

}

return body;

}

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Смена пароля</title>

<style>

body {

background-color: #ffffff

}

#bodyCont{

height: 100%;

width: 100%;

}

#outterBorder{

margin:0 auto;

border-style: solid;

border-color: orangered;

border-width: 0.5px;

border-radius: 15px;

width: 400px;

text-align: center;

}

#outterBorder:hover{

background-color: #f8f8f8;

}

#logoText{

font-family: 'Times New Roman';

font-size: 25px;

margin-top: 20px;

color: orangered;

margin-bottom: 15px;

}

.input{

border-style: solid;

border-color: firebrick;

border-width: 0.5px;

border-radius: 10px;

padding-left: 10px;

padding-right: 10px;

height: 25px;

font-size: 14px;

color: #2d2828

}

.input:focus{

outline: none;

border-color: orangered;

}

.input:hover{

border-color: orangered;

}

.inputHeader{

font-family: 'Times New Roman';

font-size: 13px;

color: gray;

margin-top: 5px;

}

#sendBtn{

width: 150px;

height: 27px;

background-color: #7d0e0e;

margin-top: 15px;

margin-bottom: 20px;

border-radius: 5px;

border-style: solid;

border-color: #4c0808;

color: white;

border-width: 0.5px;

}

#sendBtn:hover{

background-color: firebrick;

outline: none;

}

#sendBtn:focus{

outline: none;

}

</style>

</head>

<body>

<div>

<div id="bodyCont">

<div id="outterBorder">

<div id="logoContainer">

<div id="logoText">

SDWP. Восстановление пароля

</div>

</div>

<form method="PUT" action="~/Password/UpdatePassword">

<div id="inputContainer">

<div id="oldPassCont">

<div class="inputHeader">Введите старый пароль</div>

<input class="input" name="oldPass" type="password" />

</div>

<div id="newPassCont">

<div class="inputHeader">Введите новый пароль</div>

<input class="input" name="newPass" type="password" />

</div>

<div id="passConfirmationCont">

<div class="inputHeader">Повторите новый пароль</div>

<input class="input" name="passConfirmation" type="password" />

</div>

<div id="sendBtnCont">

<input id="sendBtn" type="submit" value="Поменять пароль">

</div>

</div>

</form>

</div>

</div>

</div>

</body>

</html>

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Статус смены пароля</title>

<style>

body {

background-color: #ffffff

}

#bodyCont{

height: 100%;

width: 100%;

}

#outterBorder{

margin:0 auto;

border-style: solid;

border-color: orangered;

border-width: 0.5px;

border-radius: 15px;

width: 400px;

text-align: center;

}

#outterBorder:hover{

background-color: #f8f8f8;

}

#logoText{

font-family: 'Times New Roman';

font-size: 25px;

margin-top: 20px;

color: orangered;

margin-bottom: 15px;

}

</style>

</head>

<body>

<div>

<div id="outterBorder">

<div id="logoContainer">

<div id="logoText">

@ViewBag.Result

</div>

</div>

</div>

</div>

</body>

</html>

* 1. AeroORMFramework.dll

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AeroORMFramework

{

/// <summary>

/// This attribute is set to a property which must be autoincremented

/// </summary>

public class AutoincrementIDAttribute : Attribute

{

}

}

using System;

namespace AeroORMFramework

{

[AttributeUsage(AttributeTargets.Property)]

public class CanBeNullAttribute : Attribute

{

public bool Value { get; }

public CanBeNullAttribute(bool value)

{

Value = value;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AeroORMFramework

{

public class JsonAttribute : Attribute

{

public JsonAttribute()

{

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AeroORMFramework

{

public class PrimaryKeyAttribute : Attribute

{

public PrimaryKeyAttribute() { }

}

}

using System;

namespace AeroORMFramework

{

/// <summary>

/// Attribute which is used to set the azure sql data type,

/// if the programmer isnt sutisfied with the default choice

/// </summary>

[AttributeUsage(AttributeTargets.Property)]

public class SetAzureSQLDataTypeAttribute : Attribute

{

public string SqlDataType { get; }

public SetAzureSQLDataTypeAttribute(string sqlDataType)

{

SqlDataType = sqlDataType;

}

}

}

using System;

using System.Runtime.Serialization;

namespace AeroORMFramework

{

/// <summary>

/// Exception which is thrown when an input parameter is wrong

/// </summary>

public class NotAppropriateParamException : SystemException

{

public NotAppropriateParamException() { }

public NotAppropriateParamException(string message) : base(message) { }

public NotAppropriateParamException(string message, Exception innerException)

: base(message, innerException) { }

protected NotAppropriateParamException(SerializationInfo info, StreamingContext context)

: base(info, context) { }

}

}

using System;

using System.Runtime.Serialization;

namespace AeroORMFramework

{

/// <summary>

/// Exception which is thrown when the input type when creation

/// of a new table is proceeding is wrong

/// </summary>

public class NotAppropriateTypeException : SystemException

{

public NotAppropriateTypeException() { }

public NotAppropriateTypeException(string message) : base(message) { }

public NotAppropriateTypeException(string message, Exception innerException) : base(message, innerException) { }

protected NotAppropriateTypeException(SerializationInfo info, StreamingContext context) : base(info, context) { }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

using System.Data.SqlClient;

namespace AeroORMFramework

{

/// <summary>

/// Main class, the object of which connects the entities of the programm

/// with database

/// </summary>

public class Connector

{

#region Connection settings

/// <summary>

/// Typical azure connection string with server and database parameters

/// </summary>

public string ConnectionString { get; }

private Database Database { get; }

#endregion

/// Initializes the connection to database with the given connection string

/// </summary>

public Connector(string connectionString)

{

ConnectionString = connectionString;

Database = new Database(ConnectionString);

}

/// <summary>

/// Initializes the connection to database with the given connection

/// </summary>

/// <param name="connection"></param>

public Connector(SqlConnection connection)

{

ConnectionString = connection.ConnectionString;

Database = new Database(ConnectionString);

}

#region Add Table

/// <summary>

/// Creates a table which represents the given entity

/// </summary>

/// <typeparam name="Entity">

/// The typw which will be represented in the table

/// </typeparam>

public void AddTable<Entity>()

{

Database.CreateNewTable<Entity>(GetAllProperties(typeof(Entity)));

}

/// <summary>

/// Gets all properties of a given type (if it has at least one)

/// and returns the list of them

/// </summary>

/// <exception cref="NotAppropriateTypeException">

/// Throws that exception when the type has no public instance property

/// </exception>

private List<PropertyInfo> GetAllProperties(Type type)

{

List<PropertyInfo> propertiesList = type.GetProperties(BindingFlags.Public |

BindingFlags.Instance).ToList();

CheckProperties(propertiesList);

return propertiesList;

}

private void CheckProperties(List<PropertyInfo> propertiesList)

{

if (propertiesList.Count == 0)

{

throw new NotAppropriateTypeException("In the type there must" +

" be at least one property");

}

}

#endregion

#region Get all Records

/// <summary>

/// Gets all records from the table which represents the given type

/// </summary>

/// <returns>The list of objects</returns>

public List<Entity> GetAllRecords<Entity>()

where Entity : new()

{

return Database.GetAllRecords<Entity>();

}

#endregion

#region Get records

/// <summary>

/// Gets all records where the value of a given column is equal to value

/// </summary>

/// <returns>The List of found entities</returns>

public List<Entity> GetRecords<Entity>(string columnName, object value)

where Entity : new()

{

return Database.GetRecords<Entity>(columnName, value);

}

#endregion

#region Insert new Record

/// <summary>

/// Inserts the object recrod in the table which represents the type

/// </summary>

public void Insert<Entity>(Entity entity)

where Entity : new()

{

Database.InsertRecord(entity);

}

public int InsertAndGetID<Entity>(Entity entity)

where Entity : new()

{

return Database.InsertRecordAndReturnID(entity);

}

#endregion

#region Delete table

/// <summary>

/// Delete the table which represents the given type

/// </summary>

public void DeleteTable<Entity>()

where Entity : new()

{

Database.DeleteTable<Entity>();

}

#endregion

#region Delete record

/// <summary>

/// Deletes the record which represents the given object

/// </summary>

/// <typeparam name="Entity">The type of the object</typeparam>

/// <param name="entity">The object which will be deleted</param>

public void DeleteRecord<Entity>(Entity entity)

where Entity : new()

{

Database.DeleteRecord(entity);

}

#endregion

#region Find record

/// <summary>

/// Finds the record which satisfies the given parameters

/// </summary>

/// <typeparam name="Entity">

/// The type of the object that will be returned

/// </typeparam>

/// <param name="columnName">

/// The name of the property, on which the search wil be based

/// </param>

/// <param name="value">

/// The value which you want to find

/// </param>

/// <returns>

/// The object of the type given

/// </returns>

public Entity GetRecord<Entity>(string columnName, object value)

where Entity : new()

{

return Database.FindRecord<Entity>(columnName, value);

}

#endregion

#region Update record

/// <summary>

/// Updates the record (basing on the primary key property) with the

/// given new record

/// </summary>

public void UpdateRecord<Entity>(Entity entity)

{

Database.UpdateRecord<Entity>(entity);

}

#endregion

#region Update table

/// <summary>

/// Updates the table when the class was changed (the public properties)

/// </summary>

public void UpdateTable<Entity>()

{

Database.UpdateTable<Entity>();

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

using System.Data.SqlClient;

using System.Data.SqlTypes;

using Newtonsoft.Json;

namespace AeroORMFramework

{

public class Database

{

private string ConnectionString { get; }

public Database(string connectionString) => ConnectionString = connectionString;

private SqlConnection CreateNewConnection()

{

return new SqlConnection

{

ConnectionString = ConnectionString

};

}

#region Create new table

/// <summary>

/// Creates new table in the database which represents the given type entity

/// </summary>

/// <param name="propertiesList">

/// List of properties which are defined in the type

/// </param>

public void CreateNewTable<Entity>(List<PropertyInfo> propertiesList)

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

SqlCommand createTableCommand = SqlCommandBuilder.BuildCreateTableCommand

<Entity>(propertiesList, connection);

createTableCommand.ExecuteNonQuery();

}

finally

{

connection.Close();

}

}

#endregion

#region Delete table

public void DeleteTable<Entity>()

{

SqlConnection connection = CreateNewConnection();

connection.Open();

SqlCommand deleteTableCommand = SqlCommandBuilder.BuildDeleteTableCommand<Entity>(connection);

deleteTableCommand.ExecuteNonQuery();

}

#endregion

#region Get all records

public List<Entity> GetAllRecords<Entity>()

where Entity : new()

{

SqlConnection connection = CreateNewConnection();

connection.Open();

SqlCommand getAllRecordsCommand = SqlCommandBuilder.BuildSelectAllCommand<Entity>(connection);

SqlDataReader sqlDataReader = getAllRecordsCommand.ExecuteReader();

return CreateEntitiesList<Entity>(sqlDataReader);

}

/// <summary>

/// Creates the list of entities (with help of already executed sqlDataReader)

/// of a given type and sets the properties of every instance

/// </summary>

private List<Entity> CreateEntitiesList<Entity>(SqlDataReader sqlDataReader)

where Entity : new()

{

List<Entity> entitiesList = new List<Entity>();

PropertyInfo[] propertiesArray = typeof(Entity).GetProperties(BindingFlags.Public

| BindingFlags.Instance);

while (sqlDataReader.Read())

{

Entity entity = new Entity();

for (int i = 0; i < propertiesArray.Length; i++)

{

if (sqlDataReader.IsDBNull(i))

{

propertiesArray[i].SetValue(entity, null);

}

else

{

object columnData = sqlDataReader.GetValue(i);

if (propertiesArray[i].GetCustomAttribute<JsonAttribute>() != null)

{

columnData = JsonConvert.DeserializeObject((string)columnData,

propertiesArray[i].PropertyType);

propertiesArray[i].SetValue(entity, columnData, null);

}

else

propertiesArray[i].SetValue(entity, columnData);

}

}

entitiesList.Add(entity);

}

return entitiesList;

}

#endregion

#region Insert record

public void InsertRecord<Entity>(Entity entity)

where Entity : new()

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

SqlCommand insertionCommand = SqlCommandBuilder.BuildInsertCommand(entity,

connection);

insertionCommand.ExecuteNonQuery();

}

finally

{

connection.Close();

}

}

#endregion

#region Insert record and get ID

public int InsertRecordAndReturnID<Entity>(Entity entity)

where Entity : new()

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

SqlCommand insertionCommand = SqlCommandBuilder.BuildInsertAndGetIDCommand(entity,

connection);

insertionCommand.ExecuteNonQuery();

#warning FIX THIS

insertionCommand.CommandText = $"select max(ID) from {typeof(Entity).Name}";

return (int)insertionCommand.ExecuteScalar();

}

finally

{

connection.Close();

}

}

#endregion

#region Delete record

public void DeleteRecord<Entity>(Entity entity)

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

SqlCommand deleteRecordCommand = SqlCommandBuilder.BuildDeleteRecordCommand(entity, connection);

deleteRecordCommand.ExecuteNonQuery();

}

finally

{

connection.Close();

}

}

#endregion

#region Find record

/// <summary>

/// Finds the record in the table which satisfies the given parameters

/// </summary>

/// <param name="columnName">The name of the property, based on which the search is going</param>

/// <param name="value">the value of the property</param>

/// <returns>The instance of the Entity if the object was found, the empty object otherwise</returns>

public Entity FindRecord<Entity>(string columnName, object value)

where Entity : new()

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

PropertyInfo findingProperty = GetFindingProperty<Entity>(columnName);

List<Entity> allRecordsList = GetAllRecords<Entity>();

return GetTheRecordInTheListOfEntities(allRecordsList, findingProperty, value);

}

finally

{

connection.Close();

}

}

/// <summary>

/// Returns the Property Info object which indicates which

/// property will be used in the search

/// </summary>

/// <param name="columnName"> The string name of the property</param>

private PropertyInfo GetFindingProperty<Entity>(string columnName)

{

PropertyInfo[] propertyInfosArray = typeof(Entity).GetProperties(BindingFlags.Public

| BindingFlags.Instance);

foreach (PropertyInfo property in propertyInfosArray)

{

if (property.Name == columnName)

{

return property;

}

}

return null;

}

/// <summary>

/// Finds and returns the entity in the List of entities

/// </summary>

private Entity GetTheRecordInTheListOfEntities<Entity>(List<Entity> allRecordsList,

PropertyInfo findingProperty, object value) where Entity : new()

{

foreach (Entity entity in allRecordsList)

{

if (findingProperty.GetValue(entity).Equals(value))

{

return entity;

}

}

return new Entity();

}

#endregion

#region Get records

public List<Entity> GetRecords<Entity>(string columnName, object value)

where Entity : new()

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

PropertyInfo findingProperty = GetFindingProperty<Entity>(columnName);

SqlCommand getRecordsCommand = CreateGetRecordsCommand<Entity>(findingProperty,

value, connection);

SqlDataReader sqlDataReader = getRecordsCommand.ExecuteReader();

return CreateEntitiesList<Entity>(sqlDataReader);

}

finally

{

connection.Close();

}

}

private SqlCommand CreateGetRecordsCommand<Entity>(PropertyInfo property, object value,

SqlConnection connection)

{

SqlCommand getRecordsCommand = new SqlCommand

{

Connection = connection,

CommandText = $"SELECT \* FROM {typeof(Entity).Name} WHERE {property.Name} like @value"

};

getRecordsCommand.Parameters.AddWithValue("@value", value);

return getRecordsCommand;

}

#endregion

#region Update record

public void UpdateRecord<Entity>(Entity entity)

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

SqlCommand updateRecordCommand = SqlCommandBuilder.

BuildUpdateRecordComand(entity, connection);

updateRecordCommand.ExecuteNonQuery();

}

finally

{

connection.Close();

}

}

#endregion

#region Update table

public void UpdateTable<Entity>()

{

SqlConnection connection = CreateNewConnection();

try

{

connection.Open();

SqlCommand sqlCommand = SqlCommandBuilder.BuildUpdateTableCommand<Entity>(connection);

SqlDataReader reader = sqlCommand.ExecuteReader();

List<PropertyInfo> properties = typeof(Entity).GetProperties().ToList();

List<string> columns = new List<string>();

List<string> typeProperties = properties.Select(p => p.Name).ToList();

while (reader.Read())

{

columns.Add((string)reader.GetValue(0));

}

reader.Close();

foreach (string pName in typeProperties)

{

if (columns.FindIndex(s => s == pName) == -1)

{

SqlCommand command = SqlCommandBuilder.BuildCreateNewColumnCommand<Entity>(connection,

properties.Find(p => p.Name == pName));

command.ExecuteNonQuery();

}

}

foreach (string column in columns)

{

if (typeProperties.FindIndex(s => s == column) == -1)

{

SqlCommand command = SqlCommandBuilder.BuildDeleteColumnCommand<Entity>(connection,

column);

command.ExecuteNonQuery();

}

}

}

finally

{

connection.Close();

}

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

using System.Reflection;

using System.Data.SqlTypes;

using Newtonsoft.Json;

namespace AeroORMFramework

{

public static class SqlCommandBuilder

{

private static TypeConverter TypeConverter { get; } = new TypeConverter();

#region CREATE TABLE COMMAND

public static SqlCommand BuildCreateTableCommand<Entity>(List<PropertyInfo> propertiesList,

SqlConnection connection)

{

return new SqlCommand

{

Connection = connection,

CommandText = CreateTableCommandString<Entity>(propertiesList)

};

}

/// <summary>

/// Determines wether the column in the database can be null or not

/// </summary>

/// <returns

/// Returns the "null" if the property can be null,

/// "not null" otherwise

/// </returns>

private static string SetCanBeNullString(PropertyInfo property)

{

if (property.GetCustomAttribute<CanBeNullAttribute>() != null)

{

bool canBeNull = property.GetCustomAttribute<CanBeNullAttribute>().Value;

if (canBeNull)

return "null";

}

return "not null";

}

/// <summary>

/// If the programmer decided to set the data type himself, then we set this type,

/// otherwise we take the default

/// </summary>

/// <returns>

/// The data-type

/// </returns>

private static string SetAzureSqlDataType(PropertyInfo property)

{

SetAzureSQLDataTypeAttribute setAzureSQLDataType =

property.GetCustomAttribute<SetAzureSQLDataTypeAttribute>();

JsonAttribute jsonAttribute = property.GetCustomAttribute<JsonAttribute>();

//if the property is json-coded then it has a type "nvarchar(max)"

if (jsonAttribute != null)

return "nvarchar(max)";

//if propgrammer decided to use default types, then the method returns the default types,

//else the data-type selected by developer is implemented

if (setAzureSQLDataType == null)

return TypeConverter.GetAzureSQLType(property.PropertyType);

return setAzureSQLDataType.SqlDataType;

}

/// <summary>

/// Inserts the names of the columns, their data types and additional parameters

/// into the string

/// </summary>

/// <returns>

/// The string which descries the tables, and then put into the command text

/// </returns>

private static string InsertInitialColsInTheCommandText(List<PropertyInfo> propertiesList)

{

string insertionString = string.Empty;

foreach (PropertyInfo property in propertiesList)

{

//get the null-state and data type

string canBeNullString = SetCanBeNullString(property);

string primaryKeyString = SetPrimaryKeyString(property);

string azureSqlDataTypeString = SetAzureSqlDataType(property);

insertionString += $"{property.Name} {azureSqlDataTypeString} " +

$"{canBeNullString}{primaryKeyString}, ";

}

return insertionString;

}

private static string SetPrimaryKeyString(PropertyInfo property)

{

string primaryKeyString = string.Empty;

if (property.GetCustomAttribute<PrimaryKeyAttribute>() != null)

{

if (property.GetCustomAttribute<AutoincrementIDAttribute>() != null)

primaryKeyString += " IDENTITY (1,1)";

primaryKeyString += " PRIMARY KEY";

}

return primaryKeyString;

}

private static string CreateTableCommandString<Type>(List<PropertyInfo> propertiesList)

{

string commandString = $"CREATE TABLE {typeof(Type).Name} (";

commandString += InsertInitialColsInTheCommandText(propertiesList);

//delete extra comma in the end of the command

commandString.Remove(commandString.Length - 2, 2);

commandString += ")";

return commandString;

}

#endregion

#region GET ALL RECORDS

public static SqlCommand BuildSelectAllCommand<Entity>(SqlConnection connection)

{

return new SqlCommand

{

CommandText = $"SELECT \* FROM {typeof(Entity).Name}",

Connection = connection

};

}

#endregion

#region INSERT RECORD COMAND

public static SqlCommand BuildInsertCommand<Entity>(Entity entity,

SqlConnection connection)

{

SqlCommand insertCommand = new SqlCommand

{

Connection = connection,

CommandText = CreateInsertionCommandText(entity)

};

PropertyInfo[] propertiesArray = typeof(Entity).GetProperties();

foreach (PropertyInfo property in propertiesArray)

{

if (property.GetCustomAttribute<PrimaryKeyAttribute>() == null)

{

insertCommand.Parameters.AddWithValue($"@{property.Name}",

GetPropertyValueObjectRepresentation(entity, property));

}

}

return insertCommand;

}

private static string CreateInsertionCommandText<Entity>(Entity entity)

{

PropertyInfo[] propertiesArray = typeof(Entity).GetProperties();

string parametersNames = CreateParametersString(propertiesArray);

string commandString = $"INSERT INTO {typeof(Entity).Name} {parametersNames} VALUES(";

foreach (PropertyInfo propertyInfo in propertiesArray)

{

if (propertyInfo.GetCustomAttribute<AutoincrementIDAttribute>() == null)

commandString += $"@{propertyInfo.Name}, ";

}

//delete last space

commandString = commandString.Remove(commandString.Length - 2, 2);

commandString += ")";

return commandString;

}

private static string CreateParametersString(PropertyInfo[] propertiesArray)

{

string parametersNames = "(";

foreach (PropertyInfo property in propertiesArray)

{

if (property.GetCustomAttribute<AutoincrementIDAttribute>() == null)

parametersNames += $"{property.Name}, ";

}

//delete last space and coma

parametersNames = parametersNames.Remove(parametersNames.Length - 2, 2);

return parametersNames + ")";

}

private static object GetPropertyValueObjectRepresentation<Entity>(Entity entity,

PropertyInfo property)

{

string valueString = string.Empty;

if (property.GetCustomAttribute<JsonAttribute>() != null)

{

string jsonString = JsonConvert.SerializeObject(property.GetValue(entity));

valueString += $"{jsonString}";

}

else

{

return property.GetValue(entity);

}

return valueString;

}

#endregion

#region INSERT AND GET ID COMMAND

public static SqlCommand BuildInsertAndGetIDCommand<Entity>(Entity entity,

SqlConnection connection)

{

SqlCommand insertCommand = new SqlCommand

{

Connection = connection,

CommandText = CreateInsertionCommandText(entity)

};

PropertyInfo[] propertiesArray = typeof(Entity).GetProperties();

foreach (PropertyInfo property in propertiesArray)

{

if (property.GetCustomAttribute<PrimaryKeyAttribute>() == null)

{

insertCommand.Parameters.AddWithValue($"@{property.Name}",

GetPropertyValueObjectRepresentation(entity, property));

}

}

return insertCommand;

}

#endregion

#region DELETE RECORD COMMAND

public static SqlCommand BuildDeleteRecordCommand<Entity>(Entity entity,

SqlConnection connection)

{

//get the primary property

PropertyInfo primaryProperty = GetPrimaryProperty<Entity>();

return new SqlCommand

{

Connection = connection,

CommandText = $"DELETE FROM {typeof(Entity).Name} WHERE {primaryProperty.Name} " +

$"like '{primaryProperty.GetValue(entity)}'"

};

}

/// <summary>

/// Returns a property which is marked with a primary key attribute

/// </summary>

private static PropertyInfo GetPrimaryProperty<Entity>()

{

PropertyInfo[] propertiesArray = typeof(Entity).GetProperties(BindingFlags.Public

| BindingFlags.Instance);

foreach (PropertyInfo property in propertiesArray)

{

if (property.GetCustomAttribute<PrimaryKeyAttribute>() != null)

{

return property;

}

}

return null;

}

#endregion

#region DELETE TABLE COMMAND

public static SqlCommand BuildDeleteTableCommand<Entity>(SqlConnection connection)

{

return new SqlCommand

{

Connection = connection,

CommandText = $"DROP TABLE {typeof(Entity).Name}"

};

}

#endregion

#region UPDATE RECORD COMMAND

public static SqlCommand BuildUpdateRecordComand<Entity>(Entity entity,

SqlConnection connection)

{

List<PropertyInfo> propertiesList = typeof(Entity).GetProperties(BindingFlags.Public

| BindingFlags.Instance).ToList();

SqlCommand updateCommand = new SqlCommand

{

Connection = connection,

CommandText = CreateUpdateRecordCommandText(propertiesList, entity)

};

foreach (PropertyInfo property in propertiesList)

{

if (property.GetCustomAttribute<PrimaryKeyAttribute>() == null)

{

updateCommand.Parameters.AddWithValue($"@{property.Name}",

GetPropertyValueObjectRepresentation(entity, property));

}

}

return updateCommand;

}

private static string CreateUpdateRecordCommandText<Entity>(List<PropertyInfo> propertiesList,

Entity entity)

{

string updateRecordCommandText = $"UPDATE {typeof(Entity).Name} SET ";

string commandEndingPredicate = string.Empty;

foreach (PropertyInfo property in propertiesList)

{

if (property.GetCustomAttribute<PrimaryKeyAttribute>() != null)

{

commandEndingPredicate += $"WHERE {property.Name} like " +

$"{GetPropertyValueObjectRepresentation(entity, property)}";

}

else

{

updateRecordCommandText += $"{property.Name} = " +

$"@{property.Name}, ";

}

}

//delete the last coma

updateRecordCommandText = updateRecordCommandText.Remove(

updateRecordCommandText.LastIndexOf(","), 1);

updateRecordCommandText += commandEndingPredicate;

return updateRecordCommandText;

}

#endregion

#region UPDATE TABLE COMMAND

public static SqlCommand BuildUpdateTableCommand<Entity>(SqlConnection connection)

{

return new SqlCommand()

{

Connection = connection,

CommandText = $"SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS WHERE " +

$"table\_name='{typeof(Entity).Name}'"

};

}

public static SqlCommand BuildCreateNewColumnCommand<Entity>(SqlConnection connection, PropertyInfo property)

{

string canBeNullString = SetCanBeNullString(property);

string azureSqlDataTypeString = SetAzureSqlDataType(property);

return new SqlCommand()

{

Connection = connection,

CommandText = $"ALTER TABLE {typeof(Entity).Name} ADD {property.Name}" +

$" {azureSqlDataTypeString} null"

};

}

public static SqlCommand BuildDeleteColumnCommand<Entity>(SqlConnection connection, string columnName)

{

return new SqlCommand()

{

Connection = connection,

CommandText = $"ALTER TABLE {typeof(Entity).Name} DROP COLUMN {columnName}"

};

}

#endregion

}

}

using System;

using System.Collections.Generic;

using Newtonsoft.Json;

namespace AeroORMFramework

{

public class TypeConverter

{

/// <summary>

/// Dictionary which matches the types from C# to types of Azure SQL

/// </summary>

private Dictionary<Type, string> TypesDictionary { get; set; } = new Dictionary<Type, string>

{

{typeof(string), "nvarchar(max)" },

//DateTime

{typeof(DateTime), "datetime" },

//Bool

{typeof(bool), "bit" },

//Image types

{typeof(byte[]), "varbinary(max)" },

//Value types

{typeof(byte), "tinyint" },

{typeof(sbyte), "smallint" },

{typeof(short), "smallint" },

{typeof(ushort), "int" },

{typeof(int), "int" },

{typeof(uint), "bigint" },

{typeof(long), "bigint" },

{typeof(ulong), "bigint" },

{typeof(double), "float(53)" },

{typeof(float), "float(53)" },

{typeof(decimal), "decimal(38, 38)" }

};

/// <summary>

/// With the help of TypesDictionary matches the type in cSharp to type in AzureSQL

/// </summary>

/// String representation of a SQL type which mathces the CsharpType

/// Empty string in case of exception

/// </returns>

public string GetAzureSQLType(Type cSharpType)

{

try

{

return TypesDictionary[cSharpType];

}

catch (KeyNotFoundException)

{

return "varchar(max)";

}

catch (ArgumentNullException)

{

return string.Empty;

}

}

}

}

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм | Номера листов | | | | Всего страниц в документе | № документа | Входящий номер сопроводительного документа и дата | Подпись | Дата |
|  | Измененных | Замененных | Новых | Аннулированных |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |