Главное окно (MainWindowVM)

```
using GongSolutions.Wpf.DragDrop;

using System.Windows.Navigation;

using MahApps.Metro.Controls;

using Microsoft.Toolkit.Mvvm.ComponentModel;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using TrainingSystem.Messages;

using TrainingSystem.Model;

using TrainingSystem.Resources;

using TrainingSystem.Services;

using TrainingSystem.View;

using MenuItem = TrainingSystem.ViewModel.MenuItem;

using System.Runtime.CompilerServices;
```

```
using TrainingSystem.Model.StudyObjectFactory;
using Microsoft.Toolkit;
using System.IO;
using System.Windows.Markup;
using System.IO.Packaging;
using System.Xml.Linq;
using Microsoft.Win32;

namespace TrainingSystem.ViewModel
{
    internal class MainWindowVM : BaseVM
    {

        private readonly IMessenger _Messenger;
        public MainWindowVM(IMessenger messenger)
        {
            _Messenger = messenger;
            _Messenger.Subscribe<CurrentUserChangedMessage>(this,
CurrentUserChanged);
            _Messenger.Subscribe<CurrentThemeChangedMessage>(this,
CurrentThemeChanged);
            _Messenger.Subscribe<CurrentMainWindowMessage>(this,
CurrentMainWindowChanged);
```

```
        _Messenger.Subscribe<ChosenTestMessage>(this, TestIdChosen);
    }


    private MainWindow currentMainWindow;
    public MainWindow CurrentMainWindow
    {
        get { return currentMainWindow; }
        set { currentMainWindow = value;
NotifyPropertyChanged(nameof(CurrentMainWindow)); }
    }
    private Theme currentTheme;
    public Theme CurrentTheme
    {
        get { return currentTheme; }
        set { currentTheme = value;
NotifyPropertyChanged(nameof(CurrentTheme)); }
    }

    private int chosenTestId;
    public int ChosenTestId
    {
        get { return chosenTestId; }
```

```
        protected set { chosenTestId = value;
NotifyPropertyChanged(nameof(ChosenTestId)); }
    }

    protected override void NotifyPropertyChanged([CallerMemberName] string
propertyName = "")
    {
        base.NotifyPropertyChanged(propertyName);
        if (propertyName == nameof(CurrentUser))
        {
            Properties.Settings.Default.Save();
            _Messenger.Send(new CurrentUserChangedMessage(CurrentUser));
        }
        if (propertyName == nameof(CurrentTheme))
        {
            Properties.Settings.Default.Save();
            _Messenger.Send(new CurrentThemeChangedMessage(CurrentTheme));
        }
        if (propertyName == nameof(CurrentMainWindow))
        {
            Properties.Settings.Default.Save();
            _Messenger.Send(new
CurrentMainWindowMessage(CurrentMainWindow));
```

```
    }
    if (propertyName == nameof(ChosenTestId))
    {
        Properties.Settings.Default.Save();
        _Messenger.Send(new ChosenTestMessage(ChosenTestId));
    }
}

private RelayCommand showUserInfo;
public RelayCommand ShowUserInfo
{
    get
    {
        return showUserInfo ?? new RelayCommand(obj =>
        {
            var wnd = obj as Window;
            using (Model.AppContext db = new())
            {
                var user = db.Employees.First(el => el.Login == UserLogin);

                if (user != null)
                {
                    //UserFirstName = user.FirstName;
```

```
                    //UserSecondName = user.LastName;
                    //UserLastName = user.LastName;
                    //UserLogin = user.Login;
                }
                else
                {

ShowMessageToUser(LocalizedStrings.Instance["SomethingWrong"]);
                }
            }
        });
    }
}

private RelayCommand showElement;
public RelayCommand ShowElement
{
    get
    {
        return showElement ?? new RelayCommand(obj =>
        {
            var element = obj as UIElement;
```

```
            if (element != null)
            {
                element.Visibility = Visibility.Visible;
            }
        });
    }
}
private RelayCommand crt;
public RelayCommand Crt
{
    get
    {
        return crt ?? new RelayCommand(obj =>
        {
            var element = obj as FlowDocumentReader;
            element.Document = null;
            FlowDocument document = new FlowDocument();

            var openFileDialog = new OpenFileDialog()
            {
                DefaultExt = ".docx",
                Filter = "Word documents (.docx)|*.docx"
            };
```

```
            if (openFileDialog.ShowDialog() == true)
            {
                using (var stream = File.Open(openFileDialog.FileName,
FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
                {
                    var flowDocumentConverter = new
DocxToFlowDocumentConverter(stream);
                    if (!flowDocumentConverter.PackageIsEmpty)
                    {
                        flowDocumentConverter.Read();
                        element.Document = flowDocumentConverter.Document;
                    }

ShowMessageToUser(Path.GetFileName(openFileDialog.FileName));

                }
            }

            //using (Package package =
Package.Open("C:/Users/KazernoyTys/Desktop/Repa/ISB_CorpTrainingSystem/src
/TrainingSystem/TrainingSystem/y.docx", FileMode.Open))
            //{
```

```
            //  Uri documentUri = new Uri("/word/document.xml",
UriKind.Relative);

            //  PackagePart documentPart = package.GetPart(documentUri);

            //  XElement wordDoc = XElement.Load(new
StreamReader(documentPart.GetStream()));


            //  XNamespace w =
"http://schemas.openxmlformats.org/wordprocessingml/2006/main";

            //  var paragraphs = from p in wordDoc.Descendants(w + "p") select p;


            //  foreach (var p in paragraphs)

            //  {

            //      var style = from s in p.Descendants(w + "pPr") select s;


            //      var font = (from f in style.Descendants(w + "rFonts") select
f.FirstAttribute).FirstOrDefault();

            //      var size = (from s in style.Descendants(w + "sz") select
s.FirstAttribute).FirstOrDefault();


            //      Paragraph par = new Paragraph();

            //      Run r = new Run(p.Value);

            //      if (font != null)

            //      {

            //          FontFamilyConverter converter = new FontFamilyConverter();
```

```
                //          r.FontFamily =
(FontFamily)converter.ConvertFrom(font.Value);
                //      }
                //      if (size != null)
                //      {
                //          r.FontSize = double.Parse(size.Value);
                //      }
                //      par.Inlines.Add(r);
                //      document.Blocks.Add(par);
                //  }

                //  element.Document = document;
                //}
            });
        }
    }
    private RelayCommand treeViewSelectedChanged;
    public RelayCommand TreeViewSelectedChanged
    {
        get
        {
            return treeViewSelectedChanged ?? new RelayCommand(obj =>
            {
```

```
            var element = obj as Label;


            if(treeView.SelectedItem != null)
            {
                element.Content = (treeView.SelectedItem as
TreeViewItem).Header;
                element.BorderBrush = Brushes.Black;
            }
            if (treeViewCourses.Any(x => x == treeView.SelectedItem as
TreeViewItem))
            {
                grigForContentObjectStudy.Children.Clear();
                grigForContentObjectStudy.Children.Add(new Label
                {
                    Content = (treeView.SelectedItem as TreeViewItem).Tag as string,
                    BorderBrush = Brushes.Gray,
                    BorderThickness = new Thickness(1, 1, 1, 1),
                    FontSize = 16
                });
            }
            else if (treeViewTests.Any(x => x == treeView.SelectedItem as
TreeViewItem))
            {
```

```
                grigForContentObjectStudy.Children.Clear();
                grigForContentObjectStudy.Children.Add(new Label
                {
                    Content = (treeView.SelectedItem as TreeViewItem).Tag as string,
                    BorderBrush = Brushes.Gray,
                    BorderThickness = new Thickness(1, 1, 1, 1),
                    FontSize = 16
                });

                grigForContentObjectStudy.Children.Add(new Button
                {
                    Content = "Открыть тест",
                    FontSize = 18,
                    FontWeight = FontWeights.Medium,
                    BorderBrush = Brushes.Black,
                    BorderThickness = new Thickness(1, 2, 2, 1),
                    HorizontalAlignment = HorizontalAlignment.Center,
                    VerticalAlignment = VerticalAlignment.Top,
                    Margin = new Thickness(5),
                    Command = new RelayCommand(obj =>
                    {
                        ChosenTestId = Convert.ToInt32((treeView.SelectedItem as
TreeViewItem).Tag);
```

```
                CurrentMainWindow.Hide();
                OpenTestPassingWindowMethod();
            })
        });
    }
    else
    {
        for (var i = 0; i < treeViewThemes.Count; ++i)
        {
            if (treeViewThemes[i].Any(x => x == treeView.SelectedItem as
TreeViewItem))
            {
                grigForContentObjectStudy.Children.Clear();
                grigForContentObjectStudy.Children.Add(new Label
                {
                    Content = (treeView.SelectedItem as TreeViewItem).Tag as
string,
                    BorderBrush = Brushes.Gray,
                    BorderThickness = new Thickness(1, 1, 1, 1),
                    FontSize = 16
                });
                break;
            }
```

```
                else if (i == treeViewThemes.Count - 1)
                {
                    for (var j = 0; j < treeViewLessons.Count; ++j)
                    {
                        if (treeViewLessons[j].Any(x => x == treeView.SelectedItem
as TreeViewItem))
                        {
                            grigForContentObjectStudy.Children.Clear();
                            grigForContentObjectStudy.Children.Add(new Label
                            {
                                Content = (treeView.SelectedItem as
TreeViewItem).Tag as string,
                                BorderBrush = Brushes.Gray,
                                BorderThickness = new Thickness(1, 1, 1, 1),
                                FontSize = 16
                            });
                            grigForContentObjectStudy.Children.Add(new Button
                            {
                                Content = "Открыть урок",
                                FontSize = 18,
                                FontWeight = FontWeights.Medium,
                                BorderBrush = Brushes.Black,
                                BorderThickness = new Thickness(1, 2, 2, 1),
```

```
HorizontalAlignment = HorizontalAlignment.Center,
VerticalAlignment = VerticalAlignment.Top,
Margin = new Thickness(5),
Command = new RelayCommand (obj =>
{
    grigForContentObjectStudy.Children.Clear();
    int lessonId = (new
List<Lesson>(DataWorker.GetAllLessons().Where(x => x.LessonName ==
(treeView.SelectedItem as TreeViewItem).Header.ToString())))[0].IdLesson;
    ObjectStudyFactory factory = new
LessonObjectStudyFactory((new
List<LessonMaterial>(DataWorker.GetAllLessonMaterials().Where(x =>
x.IdLesson == lessonId)))[0].Filling,
                                                    (new
List<LessonMaterial>(DataWorker.GetAllLessonMaterials().Where(x =>
x.IdLesson == lessonId)))[0].Description);
    IObjectStudy newLesson =
factory.CreateObjectStudy();

    //grigForContentObjectStudy.Children.Add(new
FlowDocumentReader
    //{
    //    Document = newLesson.FlDocReader.Document
    //});
```

```
grigForContentObjectStudy.Children.Add(newLesson.FlDocReader);


                                //(grigForContentObjectStudy.Children[0] as
FlowDocumentReader).Document = newLesson.FlDocReader.Document;



                                //foreach (var les in newLesson.LessonContent)
                                //{
                                //    (grigForContentObjectStudy.Children[0] as
FlowDocumentReader)
                                //    grigForContentObjectStudy.Children.Add(les);
                                //
grigForContentObjectStudy.RowDefinitions.Add(new RowDefinition { Height =
GridLength.Auto });
                                //    les.SetValue(Grid.RowProperty,
grigForContentObjectStudy.Children.Count - 1);
                                //}



//grigForContentObjectStudy.RowDefinitions.Add(new RowDefinition { Height =
new GridLength(1, GridUnitType.Star) });
```

```
                        })
                    });
                }
            }
        }
    }
});
}
}

private RelayCommand hideElement;
public RelayCommand HideElement
{
    get
    {
        return hideElement ?? new RelayCommand(obj =>
        {
            var element = obj as UIElement;

            if (element != null)
```

```
            {
                element.Visibility = Visibility.Collapsed;
            }
        });
    }
}



#region TREEVIEW WORKING
private DropDownButton btnCreateNew = new DropDownButton();
private Grid gridForDescription = new Grid();

private Grid gridForSwitch = new Grid();
private Grid mainGrid = new Grid();
private TreeView treeView = new TreeView();
private Grid grigForContentObjectStudy = new Grid();

private RelayCommand loadMainGrid;
public RelayCommand LoadMainGrid
{
    get
    {
```

```
        return loadMainGrid ?? new RelayCommand(obj =>
        {
            mainGrid = obj as Grid;
        });
    }
}
private RelayCommand loadGrigForContentObjectStudy;
public RelayCommand LoadGrigForContentObjectStudy
{
    get
    {
        return loadGrigForContentObjectStudy ?? new RelayCommand(obj =>
        {
            grigForContentObjectStudy = obj as Grid;
        });
    }
}



private RelayCommand loadGridForDescription;
public RelayCommand LoadGridForDescription
{
    get
```

```
    {
        return loadGridForDescription ?? new RelayCommand(obj =>
        {
            gridForDescription = obj as Grid;
        });
    }
}

private RelayCommand loadStackPanel;
public RelayCommand LoadStackPanel
{
    get
    {
        return loadStackPanel ?? new RelayCommand(obj =>
        {
            StackPanel = obj as StackPanel;
        });
    }
}

private void SetItemsInTreeView(ref TreeView treeView)
{
    TreeViewObjects = new ObservableCollection<object>();
```

```
        TreeViewCourses = new ObservableCollection<object>();

        treeViewCoursesForListBox = new ObservableCollection<object>();

        TreeViewThemes = new
ObservableCollection<ObservableCollection<object>>();

        TreeViewLessons = new
ObservableCollection<ObservableCollection<object>>();

        TreeViewTests= new ObservableCollection<object>();


        List<Course> courses = DataWorker.GetAllCourses();

        List<Model.Theme> themes = DataWorker.GetAllThemes();

        List<Lesson> lessons = DataWorker.GetAllLessons();

        List<Test> tests = DataWorker.GetAllTests();


        int indexOfCourse = 0,

           indexOfLesson = 0,

           indexOfTest = 0;



        if (courses.Count != 0)

        {

           foreach (var course in courses)

           {
```

```
var treeViewItem = new TreeViewItem
{
    FontWeight = FontWeights.Bold,
    Header = course.CourseName,
    Tag = course.Specification
};
var treeViewItemForListBox = new TreeViewItem
{
    FontWeight = FontWeights.Bold,
    Header = course.CourseName,
    Tag = course.Specification
};
TreeViewCourses.Add(treeViewItem);
TreeViewThemes.Add(new ObservableCollection<object>());
indexOfCourse = TreeViewCourses.IndexOf(treeViewItem);
treeViewItem.ItemsSource = TreeViewThemes[indexOfCourse];
TreeViewCourses.Remove(TreeViewCourses[indexOfCourse]);
TreeViewCourses.Add(treeViewItem);
TreeViewCoursesForListBox.Add(treeViewItemForListBox);

if (themes.Count != 0)
{
    foreach (var theme in themes)
```

```
                {
                    if (course.IdCourse == theme.IdCourse)
                    {
                        var treeViewItemTheme = new TreeViewItem
                        {
                            FontWeight = FontWeights.Bold,
                            Header = theme.ThemeName,
                            Tag = theme.Specification
                        };
                        treeViewLessons.Add(new ObservableCollection<object>());
                        indexOfLesson = treeViewLessons.Count() - 1;
                        treeViewItemTheme.ItemsSource =
treeViewLessons[treeViewLessons.Count - 1];
                        treeViewThemes[indexOfCourse].Add(treeViewItemTheme);

                        if (lessons.Count != 0)
                        {
                            foreach (var lesson in lessons)
                            {
                                if (theme.IdTheme == lesson.IdTheme)
                                {
                                    var treeViewItemLesson = new TreeViewItem
                                    {
```

```
                    FontWeight = FontWeights.Bold,
                    Header = lesson.LessonName,
                    Tag = lesson.Specification
                };

TreeViewLessons[indexOfLesson].Add(treeViewItemLesson);
                }
            }
        }
    }
}

if (tests.Count != 0)
{
    foreach (var test in tests)
    {
        var treeViewItemTest = new TreeViewItem
```

```
        {
            FontWeight = FontWeights.Bold,
            Header = test.TestName,
            Tag = test.IdTest
        };
        indexOfTest = TreeViewTests.Count() - 1;
        TreeViewTests.Add(treeViewItemTest);
    }
}

TreeViewObjects.Add(new TreeViewItem
{
    FontWeight = FontWeights.ExtraBold,
    Header = "Курсы",
    Tag = "Раздел с курсами"
});
if(courses.Count != 0)
(TreeViewObjects[0] as TreeViewItem).ItemsSource = TreeViewCourses;
TreeViewObjects.Add(new TreeViewItem
{
    FontWeight = FontWeights.ExtraBold,
    Header = "Тесты",
    Tag = "Раздел с тестами"
```

```
            });
            if (tests.Count != 0)
            (TreeViewObjects[1] as TreeViewItem).ItemsSource = TreeViewTests;
        }

        private RelayCommand loadTreeView;
        public RelayCommand LoadTreeView
        {
            get
            {
                return loadTreeView ?? new RelayCommand(obj =>
                {
                    treeView = obj as TreeView;

                    if(TreeViewCourses.Count == 0)
                    SetItemsInTreeView(ref treeView);
                    treeView.RequestBringIntoView +=
trVwThemes_RequestBringIntoView;



                });
            }
        }
```

```
        private void trVwThemes_RequestBringIntoView(object sender,
RequestBringIntoViewEventArgs e)
        {
            e.Handled = true;
        }



        private RelayCommand loadBtnCreateNew;
        public RelayCommand LoadBtnCreateNew
        {
            get
            {
                return loadBtnCreateNew ?? new RelayCommand(obj =>
                {
                    btnCreateNew = obj as DropDownButton;
                });
            }
        }
        private RelayCommand refreshTreeView;
        public RelayCommand RefreshTreeView
        {
            get
```

```
        {
            return refreshTreeView ?? new RelayCommand(obj =>
            {
                SetItemsInTreeView(ref treeView);
            });
        }
    }
    private ObservableCollection<object> treeViewObjects = new
ObservableCollection<object>();
    public ObservableCollection<object> TreeViewObjects
    {
        get { return treeViewObjects; }
        set { treeViewObjects = value;
NotifyPropertyChanged(nameof(TreeViewObjects)); }
    }

    private ObservableCollection<object> treeViewCourses = new
ObservableCollection<object>();
    public ObservableCollection<object> TreeViewCourses
    {
        get { return treeViewCourses; }
        set { treeViewCourses = value;
NotifyPropertyChanged(nameof(TreeViewCourses)); }
    }
```

```
        private ObservableCollection<object> treeViewCoursesForListBox = new
ObservableCollection<object>();
        public ObservableCollection<object> TreeViewCoursesForListBox
        {
            get { return treeViewCoursesForListBox; }
            set { treeViewCoursesForListBox = value;
NotifyPropertyChanged(nameof(TreeViewCoursesForListBox)); }
        }

        private ObservableCollection<ObservableCollection<object>>
treeViewThemes = new ObservableCollection<ObservableCollection<object>>();
        private ObservableCollection<ObservableCollection<object>>
TreeViewThemes
        {
            get { return treeViewThemes; }
            set { treeViewThemes = value;
NotifyPropertyChanged(nameof(TreeViewThemes)); }
        }

        private ObservableCollection<ObservableCollection<object>>
treeViewLessons = new ObservableCollection<ObservableCollection<object>>();
        private ObservableCollection<ObservableCollection<object>>
TreeViewLessons
```

```
        {
            get { return treeViewLessons; }
            set { treeViewLessons = value;
NotifyPropertyChanged(nameof(TreeViewLessons)); }
        }

        private ObservableCollection<object> treeViewTests = new
ObservableCollection<object>();
        private ObservableCollection<object> TreeViewTests
        {
            get { return treeViewTests; }
            set { treeViewTests = value;
NotifyPropertyChanged(nameof(TreeViewTests)); }
        }

        private void HndleEnterPressCourse(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Enter)
            {
                if (CourseName.Text != "" || CourseName.Text.Replace(" ", "") != "")
                {
```

```
            if
(DataWorker.CreateNewCourse(CourseName.Text.ToString()[0].ToString().ToUpp
er() + CourseName.Text.ToString().Remove(0, 1), 1, null) == "Уже существует")
            {
                ShowMessageToUser("Такой курс уже есть");
            }
            else
            {
                var index = 0;
                var treeViewItem = new TreeViewItem
                {
                    FontWeight = FontWeights.Bold,
                    Header = CourseName.Text.ToString()[0].ToString().ToUpper() +
CourseName.Text.ToString().Remove(0, 1),
                };
                var treeViewItemForListBox = new TreeViewItem
                {
                    FontWeight = FontWeights.Bold,
                    Header = CourseName.Text.ToString()[0].ToString().ToUpper() +
CourseName.Text.ToString().Remove(0, 1),
                };
                TreeViewCourses.Add(treeViewItem);
                TreeViewThemes.Add(new ObservableCollection<object>());
                index = TreeViewCourses.IndexOf(treeViewItem);
```

```
                treeViewItem.ItemsSource = TreeViewThemes[index];
                TreeViewCourses.Remove(TreeViewCourses[index]);
                TreeViewCourses.Add(treeViewItem);
                TreeViewCoursesForListBox.Add(treeViewItemForListBox);
                CourseName.Visibility = Visibility.Collapsed;

                StackPanel.MaxHeight = 100002;
                gridForDescription.Visibility = Visibility.Visible;
                gridForDescription.Tag = "course";
            }

        }
    }
    else if (e.Key == Key.Escape)
    {
        CourseName.Visibility = Visibility.Collapsed;
        btnCreateNew.Visibility = Visibility.Visible;
        stackPanel.Opacity = 1.0;
    }

}

private RelayCommand selectCourseDiscription;
```

```
public RelayCommand SelectCourseDiscription
{
    get
    {
        return selectCourseDiscription ?? new RelayCommand(obj =>
        {
            var discription = obj as TextBox;

            if (discription.Text != "" && discription.Text.Replace(" ", "") != "")
            {
                if (gridForDescription.Tag.ToString() == "course")
                {
                    List<Course> courses = DataWorker.GetAllCourses();

                    DataWorker.EditCourse(courses.First(el => el.CourseName ==
CourseName.Text.ToString()[0].ToString().ToUpper() +

CourseName.Text.ToString().Remove(0, 1)),
CourseName.Text.ToString()[0].ToString().ToUpper() +

CourseName.Text.ToString().Remove(0, 1), 1, discription.Text);

                    StackPanel.MaxHeight = 100000;
                    gridForDescription.Visibility = Visibility.Collapsed;
```

```
                stackPanel.Opacity = 1.0;
                btnCreateNew.Visibility = Visibility.Visible;

                SetItemsInTreeView(ref treeView);
            }
            else if (gridForDescription.Tag.ToString() == "theme")
            {
                List<Model.Theme> themes = DataWorker.GetAllThemes();

                var someTh = themes.First(el => el.ThemeName ==
ThemeName.Text.ToString()[0].ToString().ToUpper() +
ThemeName.Text.ToString().Remove(0, 1));

                DataWorker.EditTheme(someTh,
ThemeName.Text.ToString()[0].ToString().ToUpper() +
                                     ThemeName.Text.ToString().Remove(0, 1),
someTh.IdCourse, 1, discription.Text);

                StackPanel.MaxHeight = 100000;
                gridForDescription.Visibility = Visibility.Collapsed;

                stackPanel.Opacity = 1.0;
```

```
                btnCreateNew.Visibility = Visibility.Visible;


                SetItemsInTreeView(ref treeView);
            }


            discription.Text = "";


        }
    });
  }
}

private int themeIndex = 0;

private void HndleEnterPressTheme(object sender, KeyEventArgs e)
{
  if (e.Key == Key.Enter)
  {
    if (ThemeName.Text != "" || ThemeName.Text.Replace(" ", "") != "")
    {
      List<Course> courses = DataWorker.GetAllCourses();
```

```
            var course = courses.First(el => el.CourseName ==
selectCurrentCourse);


            if
(DataWorker.CreateNewTheme(ThemeName.Text.ToString()[0].ToString().ToUpp
er() + ThemeName.Text.ToString().Remove(0, 1), course.IdCourse, 1, null) ==
"Уже существует")
            {
                ShowMessageToUser($"Такая тема в курсе {course.CourseName}
уже есть");
            }
            else
            {
                var index = themeIndex;
                var treeViewItem = new TreeViewItem
                {
                    FontWeight = FontWeights.Bold,
                    Header = ThemeName.Text.ToString()[0].ToString().ToUpper() +
ThemeName.Text.ToString().Remove(0, 1),
                };


                treeViewLessons.Add(new ObservableCollection<object>());
                //index = treeViewLessons[index].IndexOf(treeViewItem);
```

```
                treeViewItem.ItemsSource =
treeViewLessons[treeViewLessons.Count - 1];
                treeViewThemes[index].Remove(sender as TextBox);
                treeViewThemes[index].Add(treeViewItem);
                ThemeName.Visibility = Visibility.Collapsed;
                stackPanel.Opacity = 1.0;

                StackPanel.MaxHeight = 100002;
                gridForDescription.Visibility = Visibility.Visible;
                gridForDescription.Tag = "theme";
            }

        }
    }
    else if (e.Key == Key.Escape)
    {
        ThemeName.Visibility = Visibility.Collapsed;

        stackPanel.Opacity = 1.0;
    }

}
```

```
private RelayCommand expandedFalse;

public RelayCommand ExpandedFalse
{
    get
    {
        return expandedFalse ?? new RelayCommand(obj =>
        {
            var btn = obj as DropDownButton;


            if (btn != null)
            {
                btn.IsExpanded = false;
            }
        });
    }
}


private Visibility visibilityOrCollapse;

public Visibility VisibilityOrCollapse
{
    get
    {
        if (ThemeName == null)
```

```
            visibilityOrCollapse = Visibility.Visible;
        else if (ThemeName.Visibility == Visibility.Visible)
            visibilityOrCollapse = Visibility.Collapsed;
        else if (ThemeName.Visibility == Visibility.Collapsed)
            visibilityOrCollapse = Visibility.Visible;

        return visibilityOrCollapse;
    }
}

private TextBox themeName;
public TextBox ThemeName
{
    get
    {
        return themeName;
    }
    set
    {
        themeName = value;
        NotifyPropertyChanged(nameof(ThemeName));
    }
}
```

```
private TextBox courseName = null;
public TextBox CourseName
{
    get
    {
        return courseName;
    }
    set
    {
        courseName = value;
        NotifyPropertyChanged(nameof(CourseName));
    }
}

private TextBox lessonName = null;
public TextBox LessonName
{
    get
    {
        return lessonName;
    }
    set
```

```
        {
            lessonName = value;
            NotifyPropertyChanged(nameof(LessonName));
        }
    }

    private StackPanel stackPanel;
    public StackPanel StackPanel { get { return stackPanel; } set { stackPanel = value; NotifyPropertyChanged(nameof(StackPanel)); } }

    private RelayCommand createNewTheme;
    public RelayCommand CreateNewTheme
    {
        get
        {
            return createNewTheme ?? new RelayCommand(obj =>
            {

                mainGrid.Background = Brushes.Gray;
                StackPanel.Visibility = Visibility.Collapsed;
                StackPanel.Opacity = 0.99;
```

```
        StackPanel.MaxHeight = 100001;


        TreeViewCoursesForListBox = TreeViewCoursesForListBox;


        //StackPanel = obj as StackPanel;
        //StackPanel.IsEnabled = false;


        //foreach (var item in TreeViewThemes)
        //{
        //    TreeViewThemesForListBox.Add(item);
        //}


        ////trViewItm = treeViewThemes.IndexOf(sender);
        //if (CourseName == null)
        //{
        //    if (trViewItm != -1)
        //    {
        //        courseName.KeyDown += CourseNameKeyDown;
        //        treeViewThemes.Add(CourseName);
        //    }
        //}
    });
}
```

```
}

private string selectCurrentCourse;

private RelayCommand selectCourse;
public RelayCommand SelectCourse
{
   get
   {
      return selectCourse ?? new RelayCommand(obj =>
      {
         var course = obj as ListBox;
         if (course.SelectedItem != null &&
(TreeViewCoursesForListBox[TreeViewCoursesForListBox.IndexOf(course.Select
edItem)] as TreeViewItem).Header.ToString() != null)
         selectCurrentCourse =
(TreeViewCoursesForListBox[TreeViewCoursesForListBox.IndexOf(course.Select
edItem)] as TreeViewItem).Header.ToString();

         if (course.SelectedItem != null)
         {
            var index =
TreeViewCoursesForListBox.IndexOf(course.SelectedItem);
            if (index != -1)
```

```
{
    StackPanel.Opacity = 0.99;


    Thickness thickness = new Thickness(3, 3, 3, 3);
    ThemeName = new TextBox
    {
        Text = "",
        FontSize = 12,
        MaxLength = 50,
        FontWeight = FontWeights.Bold,
        BorderThickness = thickness
    };
    ThemeName.KeyDown += HndleEnterPressTheme;
    treeViewThemes[index].Add(ThemeName);
    themeIndex = index;
    StackPanel.Visibility = Visibility.Visible;
    StackPanel.MaxHeight = 100000;
    (treeViewCourses[index] as TreeViewItem).IsExpanded = true;
    mainGrid.Background = null;
}
else
{
    ShowMessageToUser("Ошибка! курс не найден.");
```

```
                }
            }
            else
            {
                ShowMessageToUser("Курс не выбран!");
            }
        });
    }
}


private RelayCommand createNewCourse;
public RelayCommand CreateNewCourse
{
    get
    {
        return createNewCourse ?? new RelayCommand(obj =>
        {
            btnCreateNew.Visibility = Visibility.Collapsed;

            if (CourseName == null || CourseName.Visibility ==
Visibility.Collapsed)
            {
```

```
                StackPanel = obj as StackPanel;
                Thickness thickness = new Thickness(3, 3, 3, 3);
                CourseName = new TextBox
                {
                   Text = "",
                   FontSize = 14,
                   MaxLength = 50,
                   FontWeight = FontWeights.Bold,
                   BorderThickness = thickness
                };
                CourseName.KeyDown += HndleEnterPressCourse;

                stackPanel.Children.Insert(stackPanel.Children.Count - 1,
CourseName);

            }
        });
    }
}

private RelayCommand createNewTest;
public RelayCommand CreateNewTest
{
```

```
        get
        {
            return createNewTest ?? new RelayCommand(obj =>
            {
                CurrentMainWindow.Visibility = Visibility.Collapsed;


                OpenTestConstructorWindowMethod();
            });
        }
    }

    private RelayCommand closeHamburgerMenu;
    public RelayCommand CloseHamburgerMenu
    {
        get
        {
            return closeHamburgerMenu ?? new RelayCommand(obj =>
            {
                var HamburgerMenuControl = obj as HamburgerMenu;


HamburgerMenuControl.SetCurrentValue(HamburgerMenu.IsPaneOpenProperty,
false);
```

```
        });
    }
}

private RelayCommand createNewLesson;
public RelayCommand CreateNewLesson
{
    get
    {
        return createNewCourse ?? new RelayCommand(obj =>
        {
            bool isReason = true;

            foreach (var item in TreeViewThemes)
            {
                if (item.Any(x => x == treeView.SelectedItem))
                {

                    //StackPanel = obj as StackPanel;
                    //StackPanel.Visibility = Visibility.Collapsed;
                    //StackPanel.IsEnabled = false;
                    //StackPanel.Opacity = 0.99;
```

```
                //StackPanel.MaxHeight = 100001;


                CurrentTheme = new
List<Theme>(DataWorker.GetAllThemes()).FirstOrDefault(th => th.ThemeName
== (treeView.SelectedItem as TreeViewItem).Header.ToString());
                CurrentMainWindow.Visibility = Visibility.Collapsed;
                isReason = false;
                OpenCreateLessonWindowMethod();
                break;
            }
        }
        if (isReason)
        ShowMessageToUser("Выберите тему!");

        //foreach (var item in TreeViewThemes)
        //{
        //    TreeViewThemesForListBox.Add(item);
        //}

        ////trViewItm = treeViewThemes.IndexOf(sender);
        //if (CourseName == null)
        //{
        //    if (trViewItm != -1)
```

```
            //    {
            //        courseName.KeyDown += CourseNameKeyDown;
            //        treeViewThemes.Add(CourseName);
            //    }
            //}
        });
    }
}

private MainWindow mainWindowThis;

private RelayCommand loadMainWindow;
public RelayCommand LoadMainWindow
{
    get
    {
        return loadMainWindow ?? new RelayCommand(obj =>
        {
            CurrentMainWindow = obj as MainWindow;
        });
    }
}
#endregion
```

```
        #region EVENTS CALENDAR


        private CalenderBackground background;
        private Calendar eventsCalendar;
        private List<(DateTime, string, string)> eventDates;


        private RelayCommand
loadedSelectedDatesFromDataBaseForEventsCalendar;
        public RelayCommand LoadedSelectedDatesFromDataBaseForEventsCalendar
        {
           get
           {
              return loadedSelectedDatesFromDataBaseForEventsCalendar ?? new
RelayCommand(obj =>
              {
                 eventsCalendar = obj as Calendar;
                 background = new CalenderBackground(eventsCalendar);
                 background.AddOverlay("circle",
@"C:\Users\KazernoyTys\Desktop\Repa\ISB_CorpTrainingSystem\src\TrainingSys
tem\TrainingSystem\Image\circle.png");
                 background.AddOverlay("tjek",
@"C:/Users/KazernoyTys/Desktop/Repa/ISB_CorpTrainingSystem/src/TrainingSys
tem/TrainingSystem/Image/Tjek.png");
```

```
            //background.AddOverlay("cross",
@"C:\Users\KazernoyTys\Desktop\Repa\ISB_CorpTrainingSystem\src\TrainingSys
tem\TrainingSystem\Image\cross.png");

            //background.AddOverlay("box",
@"C:\Users\KazernoyTys\Desktop\Repa\ISB_CorpTrainingSystem\src\TrainingSys
tem\TrainingSystem\Image\box.png");

            //background.AddOverlay("gray",
@"C:\Users\KazernoyTys\Desktop\Repa\ISB_CorpTrainingSystem\src\TrainingSys
tem\TrainingSystem\Image\gray.png");

            eventDates = new List<(DateTime, string, string)>();

            //background.grayoutweekends = "gray";


            var eventEmployees = DataWorker.GetAllEventEmployee().Where(ee
=> ee.IdEmployee == CurrentUser.IdEmployee);

            var events = DataWorker.GetAllEvents();


            foreach (var e in events)

            {

                for (var i = 0; i < eventEmployees.Count(); ++i)

                {

                    if (e.IdEvent == eventEmployees.ElementAt(i).IdEvent)

                    {

                        for (var j = 0; j <=
e.EventEndDate.Subtract(e.EventStartDate).Days; ++j)

                        {
```

```
                    background.AddDate(e.EventStartDate.AddDays((double)j),
"tjek");
                    eventDates.Add((e.EventStartDate.AddDays((double)j),
e.EventName, e.EventDescription));
                }



                break;
            }
        }
    }

        eventsCalendar.Background = background.GetBackground();
    });
  }
}

private RelayCommand calenderOnDisplayDateChanged;
public RelayCommand CalenderOnDisplayDateChanged
{
  get
  {
    return calenderOnDisplayDateChanged ?? new RelayCommand(obj =>
```

```
            {
                eventsCalendar = obj as Calendar;
                (obj as Calendar).Background = background.GetBackground();
            });
        }
    }
    private RelayCommand slectedDatesFromEventsCalendar;
    public RelayCommand SelectedDatesFromEventsCalendar
    {
        get
        {
            return slectedDatesFromEventsCalendar ?? new RelayCommand(obj =>
            {
                var element = obj as Label;

                if (eventsCalendar.SelectedDate != null)
                {
                    if (eventDates.Any((x) => x.Item1.Date ==
eventsCalendar.SelectedDate))
                    {
                        DateTime date = (DateTime)eventsCalendar.SelectedDate;
                        element.Content = date.ToString("d") + " " +
eventDates.FirstOrDefault(x => x.Item1 == eventsCalendar.SelectedDate).Item2;
```

```
                    grigForContentObjectStudy.Children.Clear();
                    grigForContentObjectStudy.Children.Add(new Label
                    {
                        Content = eventDates.FirstOrDefault(x => x.Item1 ==
eventsCalendar.SelectedDate).Item3,
                        BorderBrush = Brushes.Gray,
                        BorderThickness = new Thickness(1, 1, 1, 1),
                        FontSize = 16
                    });
                }
                else
                {
                    DateTime date = (DateTime)eventsCalendar.SelectedDate;

                    element.Content = date.ToString("d") + " - ничего не
запланировано";
                    grigForContentObjectStudy.Children.Clear();
                }
                element.BorderBrush = Brushes.Black;

            }
```

```
            });
        }
    }


    #endregion

    private ObservableCollection<Grid> userPassabilitiesList = new
ObservableCollection<Grid>();
    public ObservableCollection<Grid> UserPassabilitiesList { get { return
userPassabilitiesList; } set { userPassabilitiesList = value;
NotifyPropertyChanged(nameof(UserPassabilitiesList)); } }
    private RelayCommand userPassabilityLoaded;
    public RelayCommand UserPassabilityLoaded
    {
        get
        {
            return userPassabilityLoaded ?? new RelayCommand(obj =>
            {
                UserPassabilitiesList.Clear();
                ListView listView = obj as ListView;
```

```
            List <CoursePassability> coursePassability =
DataWorker.GetAllCoursesPassability().Where(w => w.IdEmployee ==
CurrentUser.IdEmployee).ToList();

            List <LessonPassability> lessonPassability =
DataWorker.GetAllLessonsPassability().Where(w => w.IdEmployee ==
CurrentUser.IdEmployee).ToList();

            List <ThemePassability> themePassability =
DataWorker.GetAllThemesPassability().Where(w => w.IdEmployee ==
CurrentUser.IdEmployee).ToList();

            List <TestPassability> testPassability =
DataWorker.GetAllTestsPassability().Where(w => w.IdEmployee ==
CurrentUser.IdEmployee).ToList();

            foreach (CoursePassability passability in coursePassability)

            {

               UserPassabilitiesList.Add(GenerateGridForPassability(passability));

            }

            foreach (LessonPassability passability in lessonPassability)

            {

               UserPassabilitiesList.Add(GenerateGridForPassability(passability));

            }

            foreach (ThemePassability passability in themePassability)

            {

               UserPassabilitiesList.Add(GenerateGridForPassability(passability));

            }

            foreach (TestPassability passability in testPassability)
```

```
            {
                UserPassabilitiesList.Add(GenerateGridForPassability(passability));
            }
        });
    }
}

public Grid GenerateGridForPassability(object obj)
{
    Grid grid = new Grid();
    grid.ColumnDefinitions.Add(new ColumnDefinition()
    {
        Width = new GridLength(1, GridUnitType.Star)
    });
    grid.ColumnDefinitions.Add(new ColumnDefinition()
    {
        Width = new GridLength(1, GridUnitType.Star)
    });
    grid.ColumnDefinitions.Add(new ColumnDefinition()
    {
        Width = new GridLength(1, GridUnitType.Star),
        MaxWidth = 300
    });
```

```
switch (obj.GetType().Name)
{
    case "CoursePassability":
        {
            CoursePassability passability = obj as CoursePassability;
            Label name = new Label { Content = "Курс " +
DataWorker.GetAllCourses().Where(w => w.IdCourse ==
passability.IdCourse).FirstOrDefault().CourseName };
            Label progressText = new Label { Content =
passability.Progress.ToString() + '%' };
            progressText.HorizontalAlignment = HorizontalAlignment.Right;
            Style style = new Style();
            ProgressBar progress = new ProgressBar { Value =
passability.Progress, Style = style, MaxWidth = 300 };
            if (passability.Progress <= 50)
            {
                progress.Foreground = new SolidColorBrush(Color.FromRgb(225,
Convert.ToByte(passability.Progress * 3 + 75), 75));
            }
            else
            {
                progress.Foreground = new
SolidColorBrush(Color.FromRgb(Convert.ToByte(375 - 3 * passability.Progress),
225, 75));
```

```
                }
                grid.Children.Add(name);
                name.SetValue(Grid.ColumnProperty, 0);
                grid.Children.Add(progressText);
                progressText.SetValue(Grid.ColumnProperty, 1);
                grid.Children.Add(progress);
                progress.SetValue(Grid.ColumnProperty, 2);
                break;
            }
        case "LessonPassability":
            {
                LessonPassability passability = obj as LessonPassability;
                Label name = new Label { Content = "Урок " +
DataWorker.GetAllLessons().Where(w => w.IdLesson ==
passability.IdLesson).FirstOrDefault().LessonName };
                Label progressText = new Label { Content =
passability.Progress.ToString() + '%' };
                progressText.HorizontalAlignment = HorizontalAlignment.Right;
                Style style = new Style();
                ProgressBar progress = new ProgressBar { Value =
passability.Progress, Style = style, MaxWidth = 300 };
                if (passability.Progress <= 50)
                {
```

```
                    progress.Foreground = new SolidColorBrush(Color.FromRgb(225,
Convert.ToByte(passability.Progress * 3 + 75), 75));
                }
                else
                {
                    progress.Foreground = new
SolidColorBrush(Color.FromRgb(Convert.ToByte(375 - 3 * passability.Progress),
225, 75));
                }
                grid.Children.Add(name);
                name.SetValue(Grid.ColumnProperty, 0);
                grid.Children.Add(progressText);
                progressText.SetValue(Grid.ColumnProperty, 1);
                grid.Children.Add(progress);
                progress.SetValue(Grid.ColumnProperty, 2);
                break;
            }
        case "ThemePassability":
            {
                ThemePassability passability = obj as ThemePassability;
                Label name = new Label { Content = "Тема " +
DataWorker.GetAllThemes().Where(w => w.IdTheme ==
passability.IdTheme).FirstOrDefault().ThemeName };
```

```
                Label progressText = new Label { Content =
passability.Progress.ToString() + '%' };
                progressText.HorizontalAlignment = HorizontalAlignment.Right;
                Style style = new Style();
                ProgressBar progress = new ProgressBar { Value =
passability.Progress, Style = style, MaxWidth = 300 };
                if (passability.Progress <= 50)
                {
                    progress.Foreground = new SolidColorBrush(Color.FromRgb(225,
Convert.ToByte(passability.Progress * 3 + 75), 75));
                }
                else
                {
                    progress.Foreground = new
SolidColorBrush(Color.FromRgb(Convert.ToByte(375 - 3 * passability.Progress),
225, 75));
                }
                grid.Children.Add(name);
                name.SetValue(Grid.ColumnProperty, 0);
                grid.Children.Add(progressText);
                progressText.SetValue(Grid.ColumnProperty, 1);
                grid.Children.Add(progress);
                progress.SetValue(Grid.ColumnProperty, 2);
                break;
```

```
            }
        case "TestPassability":
            {
                TestPassability passability = obj as TestPassability;
                Label name = new Label { Content = "Тест " +
DataWorker.GetAllTests().Where(w => w.IdTest ==
passability.IdTest).FirstOrDefault().TestName };
                Label progressText = new Label { Content =
passability.Progress.ToString() + '%' };
                progressText.HorizontalAlignment = HorizontalAlignment.Right;
                Style style = new Style();
                ProgressBar progress = new ProgressBar { Value =
passability.Progress, Style = style, MaxWidth = 300 };
                if (passability.Progress <= 50)
                {
                    progress.Foreground = new SolidColorBrush(Color.FromRgb(225,
Convert.ToByte(passability.Progress * 3 + 75), 75));
                }
                else
                {
                    progress.Foreground = new
SolidColorBrush(Color.FromRgb(Convert.ToByte(375 - 3 * passability.Progress),
225, 75));
                }
```

```
                grid.Children.Add(name);
                name.SetValue(Grid.ColumnProperty, 0);
                grid.Children.Add(progressText);
                progressText.SetValue(Grid.ColumnProperty, 1);
                grid.Children.Add(progress);
                progress.SetValue(Grid.ColumnProperty, 2);
                break;
            }
        }
        return grid;
    }

  }
}
```

Окно авторизации (LoginWindowVM)

```
using GongSolutions.Wpf.DragDrop;
using MahApps.Metro.Controls;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using TrainingSystem.Messages;
using TrainingSystem.Model;
using TrainingSystem.Resources;
using TrainingSystem.Services;
using TrainingSystem.View;

namespace TrainingSystem.ViewModel
{
```

```
internal class LoginWindowVM : BaseVM
{
    private readonly IMessenger _Messenger;
    public LoginWindowVM(IMessenger messenger)
    {
        _Messenger = messenger;
        _Messenger.Subscribe<CurrentUserChangedMessage>(this,
CurrentUserChanged);
    }

    private Employee currentUser;
    public Employee CurrentUser
    {
        get { return currentUser; }
        set { currentUser = value; NotifyPropertyChanged(nameof(CurrentUser)); }
    }

    protected override void NotifyPropertyChanged([CallerMemberName] string
propertyName = "")
    {
        base.NotifyPropertyChanged(propertyName);
        if (propertyName == nameof(CurrentUser))
        {
            Properties.Settings.Default.Save();
            _Messenger.Send(new CurrentUserChangedMessage(CurrentUser));
        }
    }


    /// <summary>
```

```
        /// Команда авторизации пользователя
        /// </summary>
        private RelayCommand loginOn;
        public RelayCommand LoginOn
        {
            get
            {
                return loginOn ?? new RelayCommand(obj =>
                {
                    Window wnd = obj as Window;
                    bool chekedIs = true;
                    string resultStr = "";
                    if (UserLogin == null || UserLogin == "" || UserLogin.Replace(" ",
"").Length == 0)
                    {
                        SetRedBlockControl(wnd, "tbUserLogin");
                        chekedIs = false;
                    }
                    if (UserPassword == null || UserPassword == "" ||
UserPassword.Replace(" ", "").Length == 0)
                    {
                        SetRedBlockControl(wnd, "tbUserPassword");
                    }
                    else if (chekedIs)
                    {

                        resultStr = DataWorker.Login(UserLogin, UserPassword);
                        if (resultStr == LocalizedStrings.Instance["SignedIn"])
                        {
                            Task.Run(() => { ShowMessageToUser(resultStr); });
```

```
                    SetNullPropertyValues();
                    CurrentUser = (DataWorker.GetAllEmployees().First(el =>
el.Login == UserLogin &&
                                                        el.Password ==
UserPassword));
                    OpenMainWindowMethod();
                    wnd.Close();
                }
                else
                {
                    ShowMessageToUser(resultStr);
                    UserPassword = null;
                }
                //else
                //{
                //    resultStr = DataWorker.Login(UserLogin, UserPassword);
                //    ShowMessageToUser(resultStr);
                //    SetNullPropertyValues();
                //    if (resultStr == LocalizedStrings.Instance["SignedIn"])
                //    {
                //        OpenMainWindowMethod();
                //        wnd.Close();
                //    }
                //}

            }
        }
        );
    }
}
```

```
    }
}
```

Окно прохождения теста (TestPassingWindowVM)

```
using GongSolutions.Wpf.DragDrop;
```

```
using System.Windows.Navigation;
using MahApps.Metro.Controls;
using Microsoft.Toolkit.Mvvm.ComponentModel;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using TrainingSystem.Messages;
using TrainingSystem.Model;
using TrainingSystem.Resources;
using TrainingSystem.Services;
using TrainingSystem.View;
using MenuItem = TrainingSystem.ViewModel.MenuItem;
using System.Runtime.CompilerServices;
using TrainingSystem.Model.StudyObjectFactory;
using Microsoft.Toolkit;
using System.IO;
```

```
using System.Windows.Markup;
using System.IO.Packaging;
using System.Xml.Linq;
using Microsoft.Win32;

namespace TrainingSystem.ViewModel
{
    internal class TestPassingWindowVM : BaseVM
    {
        private readonly IMessenger _Messenger;
        public TestPassingWindowVM(IMessenger messenger)
        {
            _Messenger = messenger;
            _Messenger.Subscribe<CurrentUserChangedMessage>(this,
CurrentUserChanged);
            _Messenger.Subscribe<CurrentThemeChangedMessage>(this,
CurrentThemeChanged);
            _Messenger.Subscribe<CurrentMainWindowMessage>(this,
CurrentMainWindowChanged);
            _Messenger.Subscribe<ChosenTestMessage>(this, TestIdChosen);
        }

        private List<string> userAnswers = new List<string>();
```

```
public List<string> UserAnswers { get { return userAnswers; } set { userAnswers = value; NotifyPropertyChanged(nameof(UserAnswers)); } }

private Window testPassingWindow;

public Window TestPassingWindow { get { return testPassingWindow; } set { testPassingWindow = value; NotifyPropertyChanged(nameof(TestPassingWindow)); } }

private RelayCommand loadTestPassingWindow;

public RelayCommand LoadTestPassingWindow
{
    get
    {
        return loadTestPassingWindow ?? new RelayCommand(obj =>
        {
            TestPassingWindow = obj as Window;
            TestPassingWindow.BeginInit();

            TheTest = DataWorker.GetAllTests().Where(w => w.IdTest == ChosenTestId).FirstOrDefault();

            TestPassingWindow.Title = TheTest.TestName;
            foreach (TestQuestions testQuestion in DataWorker.GetAllTestQuestions().Where(w => w.IdTest == TheTest.IdTest))
```

```
            {
                TestQuestions.Add(DataWorker.GetAllQuestions().Where(w =>
w.IdQuestion == testQuestion.IdQuestion).FirstOrDefault());
                UserAnswers.Add("");
            }

            int i = 1;
            foreach (Question question in TestQuestions)
            {
                Label label = new Label
                {
                    Content = "Вопрос " + i.ToString(),
                    Tag = question.IdQuestion
                };
                i++;
                QuestionLabelsList.Add(label);
            }
        });
    }
}

private RelayCommand returnToMainWindow;
public RelayCommand ReturnToMainWindow
```

```
{
    get
    {
        return returnToMainWindow ?? new RelayCommand(obj =>
        {
            TestPassingWindow.Close();
            QuestionLabelsList.Clear();
            TestQuestions.Clear();
            UserAnswers.Clear();
            if (CurrentMainWindow == null)
            {
                //CurrentMainWindow = new MainWindow();
                CurrentMainWindow.Show();
            }
            else
            {
                CurrentMainWindow.Visibility = Visibility.Visible;
            }
        });
    }
}

private Test theTest = new Test();
```

```
public Test TheTest { get { return theTest; } set { theTest = value; NotifyPropertyChanged(nameof(TheTest)); } }

private List<Question> testQuestions = new List<Question>();

public List<Question> TestQuestions { get { return testQuestions; } set { testQuestions = value; NotifyPropertyChanged(nameof(TestQuestions)); } }

private ObservableCollection<Label> questionLabelsList = new ObservableCollection<Label>();

public ObservableCollection<Label> QuestionLabelsList { get { return questionLabelsList; } set { questionLabelsList = value; NotifyPropertyChanged(nameof(QuestionLabelsList)); } }

private ListBox lbQuestions;

private RelayCommand listBoxLoaded;

public RelayCommand ListBoxLoaded

{

   get

   {

      return listBoxLoaded ?? new RelayCommand(obj =>

      {

         lbQuestions = obj as ListBox;

      });
```

```
    }
}

public void SaveUserAnswer()
{
    FrameworkElement answerElement = new FrameworkElement();
    Question question = new Question();
    foreach (FrameworkElement element in GridQuestionBody.Children)
    {
        if (element.Tag != null)
        {
            answerElement = element;
            question = element.Tag as Question;
            break;
        }
    }
    if (TestQuestions.IndexOf(question) != -1)
    {
        switch (HelpFunctionsVM.GetTextBeforeWave(question))
        {
            case "ManyVariantsTypeOfQuestion~":
                {
                    foreach (Grid grid in (answerElement as ListBox).Items)
```

```
                {
                    if ((grid.Children[0] as CheckBox).IsChecked == true)
                    {
                        UserAnswers[TestQuestions.IndexOf(question)] +=
(grid.Children[1] as TextBox).Text + ";";
                    }
                }
                break;
            }
        case "OneVariantTypeOfQuestion~":
            {
                foreach (Grid grid in (answerElement as ListBox).Items)
                {
                    if ((grid.Children[0] as RadioButton).IsChecked == true)
                    {
                        UserAnswers[TestQuestions.IndexOf(question)] +=
(grid.Children[1] as TextBox).Text;
                    }
                }
                break;
            }
        case "TextTypeOfQuestion~":
            {
```

```
                TextBox textBox = (TextBox)answerElement;
                UserAnswers[TestQuestions.IndexOf(question)] = textBox.Text;
                //UserAnswers[TestQuestions.IndexOf(question)] += new
TextRange(textBox.Document.ContentStart, textBox.Document.ContentEnd).Text;
                //UserAnswers[TestQuestions.IndexOf(question)] =
UserAnswers[TestQuestions.IndexOf(question)].Remove(UserAnswers[TestQuestio
ns.IndexOf(question)].Length);
                break;
            }
        case "MissingTextTypeOfQestion~":
            {
                StackPanel panel = (StackPanel)answerElement;
                foreach (FrameworkElement element in panel.Children)
                {
                    if (element.GetType().Name == "TextBox")
                    {
                        UserAnswers[TestQuestions.IndexOf(question)] += (element
as TextBox).Text + ";";
                    }
                }
                break;
            }
        case "SetRightOrderTypeOfQuestion~":
            {
```

```
                    ListBox listBox = (ListBox)answerElement;
                    for (int i = 1; i <= listBox.Items.Count; i++)
                    {
                        Grid row = listBox.Items[i - 1] as Grid;
                        TextBox textBox = (TextBox)row.Children[0];
                        UserAnswers[TestQuestions.IndexOf(question)] +=
textBox.Text + ";";
                    }
                    break;
                }
            case "MatchingTypeOfQuestion~":
                {



                    break;
                }
            default:
                {
                    break;
                }
        }
    }
    GridListForListbox.Clear();
```

```
}

private RelayCommand listBoxQuestionsSelectionChanged;
public RelayCommand ListBoxQuestionsSelectionChanged
{
   get
   {
      return listBoxQuestionsSelectionChanged ?? new RelayCommand(obj =>
      {

         SaveUserAnswer();

         Question question = new Question();
         foreach (FrameworkElement element in GridQuestionBody.Children)
         {
            if (element.Tag != null)
            {
               question = element.Tag as Question;
               break;
            }
         }
         if (QuestionLabelsList.Count != 0)
         {
```

```
            question = TestQuestions[lbQuestions.SelectedIndex];
            if (question != null)
            {
                BuildGridForQuestionBody(question);
            }
        }
    });
  }
}

private RelayCommand endTestPassing;
public RelayCommand EndTestPassing
{
  get
  {
    return endTestPassing ?? new RelayCommand(obj =>
    {
      SaveUserAnswer();

      int j = 0;
      for (int i = 0; i < TestQuestions.Count; i++)
      {
        if(TestQuestions[i].Answer == UserAnswers[i])
```

```
                {
                    j++;
                }
            }
            double mark;
            if (TestQuestions.Count != 0)
            {
                mark = Convert.ToDouble(j) /
Convert.ToDouble(TestQuestions.Count) * 100;
            }
            else
            {
                mark = 0;
            }
            string score = j.ToString();
            switch(score[score.Length - 1])
            {
                case '1':
                    {
                        score += " балл";
                        break;
                    }
                case '2': case '3': case '4':
```

```
                {
                    score += " балла";
                    break;
                }
            default:
                {
                    score += " баллов";
                    break;
                }
        }
        if (MessageBox.Show("Вы набрали " + score + " из " +
TestQuestions.Count.ToString() + " (" + mark.ToString() + "%)" + "\nСохранить
результат?", "Question", MessageBoxButton.YesNo, MessageBoxImage.Question)
== MessageBoxResult.Yes)
        {
            string result =
DataWorker.CreateNewTestPassability(TheTest.IdTest, CurrentUser.IdEmployee,
mark, "", null);
            if (result == "Уже существует")
            {
                MessageBox.Show("Ошибка создания записи");
            }
            else
            {
```

```
                    if(result == "Успешно!")
                    {
                        MessageBox.Show("Запись успешно сохранена");
                        ReturnToMainWindow.Execute(null);
                    }
                }
            }
            else
            {
                ReturnToMainWindow.Execute(null);
            }
        });
    }
}

    private Grid gridQuestionBody;
    public Grid GridQuestionBody { get { return gridQuestionBody; } set {
gridQuestionBody = value; NotifyPropertyChanged(nameof(GridQuestionBody)); }
}

    private RelayCommand loadGridQuestionBody;
    public RelayCommand LoadGridQuestionBody
    {
```

```
        get
        {
            return loadGridQuestionBody ?? new RelayCommand(obj =>
            {
                GridQuestionBody = obj as Grid;
            });
        }
    }
    private ObservableCollection<Grid> gridListForListbox = new
ObservableCollection<Grid>();
    public ObservableCollection<Grid> GridListForListbox { get { return
gridListForListbox; } set { gridListForListbox = value;
NotifyPropertyChanged(nameof(GridListForListbox)); } }



    public void BuildGridForQuestionBody(Question question)
    {
        string typeOfQuestion = HelpFunctionsVM.GetTextBeforeWave(question);
        GridQuestionBody.Children.Clear();
        GridQuestionBody.RowDefinitions.Clear();
        GridQuestionBody.ColumnDefinitions.Clear();
        GridQuestionBody.ShowGridLines = true;
        GridQuestionBody.RowDefinitions.Add(new RowDefinition()
```

```
{
    Height = new GridLength(1, GridUnitType.Star)
});
GridQuestionBody.RowDefinitions.Add(new RowDefinition()
{
    Height = new GridLength(10, GridUnitType.Star)
});
Label labelQuestionName = new Label()
{
    Content = question.QuestionName,
    HorizontalAlignment = HorizontalAlignment.Center,
    FontWeight = FontWeights.Bold,
    FontSize = 18
};
GridQuestionBody.Children.Add(labelQuestionName);
labelQuestionName.SetValue(Grid.RowProperty, 0);
List<string> variants;
switch (HelpFunctionsVM.GetTextBeforeWave(question))
{
    case "ManyVariantsTypeOfQuestion~":
        {
            ListBox listBox = new ListBox()
            {
```

```
                ItemsSource = GridListForListbox,
                Tag = question
            };
            GridQuestionBody.Children.Add(listBox);
            listBox.SetValue(Grid.RowProperty, 2);
            variants =
HelpFunctionsVM.GetListOfTextWithBreaks(HelpFunctionsVM.GetTextAfterWav
e(question));
            List<string> userVariants =
HelpFunctionsVM.GetListOfTextWithBreaks(UserAnswers[TestQuestions.IndexOf
(question)]);
            foreach (string variant in variants)
            {
                GridListForListbox.Add(GenerateGridForListBox(question,
variant));
                if(userVariants.Contains(variant))
                {
                    Grid grid = GridListForListbox.LastOrDefault();
                    CheckBox checkBox = grid.Children[0] as CheckBox;
                    checkBox.IsChecked = true;
                }
            }
            break;
        }
```

```
            case "OneVariantTypeOfQuestion~":
                {
                    ListBox listBox = new ListBox()
                    {
                        ItemsSource = GridListForListbox,
                        Tag = question
                    };
                    GridQuestionBody.Children.Add(listBox);
                    listBox.SetValue(Grid.RowProperty, 2);
                    variants =
HelpFunctionsVM.GetListOfTextWithBreaks(HelpFunctionsVM.GetTextAfterWav
e(question));
                    string userVariant =
UserAnswers[TestQuestions.IndexOf(question)];
                    foreach (string variant in variants)
                    {
                        GridListForListbox.Add(GenerateGridForListBox(question,
variant));
                        if(userVariant == variant)
                        {
                            Grid grid = GridListForListbox.LastOrDefault();
                            RadioButton radioButton = grid.Children[0] as RadioButton;
                            radioButton.IsChecked = true;
                        }
```

```
                }
                break;
            }
        case "TextTypeOfQuestion~":
            {
                TextBox textBox = new TextBox
                {
                    FontSize = 18,
                    Tag = question
                };
                TextBoxHelper.SetWatermark(textBox, "Введите ответ здесь");
                textBox.Text = UserAnswers[TestQuestions.IndexOf(question)];
                //textBox.Document.Blocks.Clear();

//textBox.Document.Blocks.Remove(textBox.Document.Blocks.LastBlock);
                //textBox.Document.Blocks.Add(new Paragraph(new
Run(UserAnswers[TestQuestions.IndexOf(question)])));
                GridQuestionBody.Children.Add(textBox);
                textBox.SetValue(Grid.RowProperty, 2);
                break;
            }
        case "MissingTextTypeOfQestion~":
            {
```

```
string str = "";
int j = 0;
for (int i = 0; i < question.Filling.Length; i++)
{
   if (question.Filling[i] == '~')
   {
      j = i;
      break;
   }
}
StackPanel stackPanel = new StackPanel()
{
   Orientation = Orientation.Horizontal,
   Margin = new Thickness(5, 10, 5, 0),
   Tag = question
};
int k = 0;
List<string> list =
HelpFunctionsVM.GetListOfTextWithBreaks(UserAnswers[TestQuestions.IndexOf
(question)]);
for (int i = j + 1; i < question.Filling.Length; i++)
{
   if (question.Filling[i] == '#')
```

```
{
    Label label = new Label()
    {
        Content = str,
    };
    stackPanel.Children.Add(label);

    str = "";
    TextBox textBox = new TextBox()
    {
        TextWrapping = TextWrapping.Wrap,
        MinWidth = 100,
        MaxWidth = 200,
        Height = 30,
        VerticalAlignment = VerticalAlignment.Top,
    };
    if(UserAnswers[TestQuestions.IndexOf(question)] != "")
    {
        textBox.Text = list[k];
        k++;
    }
    TextBoxHelper.SetWatermark(textBox, "Впишите фрагмент");
    stackPanel.Children.Add(textBox);
```

```
            }
            else
            {
                str += question.Filling[i];
            }
            if (i == question.Filling.Length - 1)
            {
                Label label = new Label()
                {
                    Content = str,
                };
                stackPanel.Children.Add(label);
                str = "";
            }
        }
        GridQuestionBody.Children.Add(stackPanel);
        stackPanel.SetValue(Grid.RowProperty, 1);
        break;
    }
case "SetRightOrderTypeOfQuestion~":
    {
        ListBox listBox = new ListBox()
        {
```

```
                    ItemsSource = GridListForListbox,
                    Tag = question
                };
                GridQuestionBody.Children.Add(listBox);
                listBox.SetValue(Grid.RowProperty, 1);
                variants =
HelpFunctionsVM.GetListOfTextWithBreaks(HelpFunctionsVM.GetTextAfterWav
e(question));
                if (UserAnswers[TestQuestions.IndexOf(question)] != "")
                {
                    variants =
HelpFunctionsVM.GetListOfTextWithBreaks(UserAnswers[TestQuestions.IndexOf
(question)]);
                }
                foreach (string variant in variants)
                {
                    GridListForListbox.Add(GenerateGridForListBox(question,
variant));
                }
                break;
            }
        case "MatchingTypeOfQuestion~":
            {
```

```
                break;
            }
        default:
            {
                break;
            }
    }
    UserAnswers[TestQuestions.IndexOf(question)] = "";
}

private RelayCommand radioButtonChanged;
public RelayCommand RadioButtonChanged
{
    get
    {
        return radioButtonChanged ?? new RelayCommand(obj =>
        {
            Grid gottenGrid = obj as Grid;
            foreach (Grid grid in GridListForListbox)
            {
                if (grid != gottenGrid && (grid.Children[0] as
RadioButton).IsChecked == true)
                {
```

```
                (grid.Children[0] as RadioButton).IsChecked = false;
            }
        }
    });
  }
}

public Grid GenerateGridForListBox(Question question, string text)///////это в тоже
{
  Grid grid = new Grid();
  grid.ShowGridLines = true;
  switch (HelpFunctionsVM.GetTextBeforeWave(question))
  {
    case "ManyVariantsTypeOfQuestion~":
      {

        grid.ColumnDefinitions.Add(new ColumnDefinition()
        {
          Width = new GridLength(1, GridUnitType.Star)
        });
        grid.ColumnDefinitions.Add(new ColumnDefinition()
        {
```

```
            Width = new GridLength(20, GridUnitType.Star)
        });

        CheckBox checkBox = new CheckBox
        {
            HorizontalAlignment = HorizontalAlignment.Center,
            VerticalAlignment = VerticalAlignment.Center
        };
        grid.Children.Add(checkBox);
        checkBox.SetValue(Grid.ColumnProperty, 0);
        TextBox textBox = new TextBox
        {
            FontSize = 18,
            Text = text,
            IsReadOnly = true
        };

        grid.Children.Add(textBox);
        textBox.SetValue(Grid.ColumnProperty, 1);
        break;
    }
case "OneVariantTypeOfQuestion~":
    {
```

```
grid.ColumnDefinitions.Add(new ColumnDefinition()
{
    Width = new GridLength(1, GridUnitType.Star)
});
grid.ColumnDefinitions.Add(new ColumnDefinition()
{
    Width = new GridLength(20, GridUnitType.Star)
});

RadioButton radioButton = new RadioButton
{
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    Command = RadioButtonChanged,
    CommandParameter = grid
};
grid.Children.Add(radioButton);
radioButton.SetValue(Grid.ColumnProperty, 0);
TextBox textBox = new TextBox
{
    FontSize = 18,
    Text = text,
    IsReadOnly = true
```

```
        };

        grid.Children.Add(textBox);
        textBox.SetValue(Grid.ColumnProperty, 1);
        break;
    }
case "SetRightOrderTypeOfQuestion~":
    {

        grid.ColumnDefinitions.Add(new ColumnDefinition()
        {
            Width = new GridLength(19, GridUnitType.Star)
        });
        grid.ColumnDefinitions.Add(new ColumnDefinition()
        {
            Width = new GridLength(1, GridUnitType.Star)
        });



        TextBox textBox = new TextBox
        {
            FontSize = 18,
```

```
    Text = text,
    IsReadOnly = true
};
grid.Children.Add(textBox);
textBox.SetValue(Grid.ColumnProperty, 0);

Grid miniGrid = new Grid();
miniGrid.RowDefinitions.Add(new RowDefinition()
{
    Height = new GridLength(1, GridUnitType.Star)
});
miniGrid.RowDefinitions.Add(new RowDefinition()
{
    Height = new GridLength(1, GridUnitType.Star)
});

Button buttonUp = new Button();
buttonUp.Content = "▲";
miniGrid.Children.Add(buttonUp);
buttonUp.SetValue(Grid.RowProperty, 0);
buttonUp.Command = RowGoUpInListView;
buttonUp.CommandParameter = grid;
```

```
                Button buttonDown = new Button();
                buttonDown.Content = "▼";
                miniGrid.Children.Add(buttonDown);
                buttonDown.SetValue(Grid.RowProperty, 1);
                buttonDown.Command = RowGoDownInListView;
                buttonDown.CommandParameter = grid;

                grid.Children.Add(miniGrid);
                miniGrid.SetValue(Grid.ColumnProperty, 1);
                break;
            }
    }
    return grid;
}

private RelayCommand rowGoUpInListView;
public RelayCommand RowGoUpInListView
{
    get
    {
        return rowGoUpInListView ?? new RelayCommand(obj =>
        {
            Grid gridRow = obj as Grid;
```

```
                int indexOfGottenRow = GridListForListbox.IndexOf(gridRow);
                if (indexOfGottenRow != 0)
                {
                    var buf = gridRow;
                    GridListForListbox[indexOfGottenRow] =
GridListForListbox[indexOfGottenRow - 1];
                    GridListForListbox[indexOfGottenRow - 1] = buf;
                }
            });
        }
    }

    private RelayCommand rowGoDownInListView;
    public RelayCommand RowGoDownInListView
    {
        get
        {
            return rowGoDownInListView ?? new RelayCommand(obj =>
            {
                Grid gridRow = obj as Grid;
                int indexOfGottenRow = GridListForListbox.IndexOf(gridRow);
                if (indexOfGottenRow != GridListForListbox.Count - 1)
                {
```

```
                    var buf = gridRow;
                    GridListForListbox[indexOfGottenRow] =
GridListForListbox[indexOfGottenRow + 1];
                    GridListForListbox[indexOfGottenRow + 1] = buf;
                }
            });
        }
    }
  }
}
```