**КОМПЬЮТЕРНАЯ АКАДЕМИЯ «ШАГ»**

|  |  |  |
| --- | --- | --- |
|  | **КОМПЬЮТЕРНАЯ АКАДЕМИЯ**  **КОМПЬЮТЕРНАЯ АКАДЕМИЯ** |  |

**Компьютерная академия «ШАГ»**

**Одесский филиал**

**Кафедра Разработки программного обеспечения**

**КУРСОВОЙ ПРОЕКТ ПО ADO.NET**

**«Органайзер перерывов»**

**Студентки группы EKO\_16\_П3  
Александр Буженко**

**Руководитель курсового проекта:  
Полянский Виталий Владилинович**

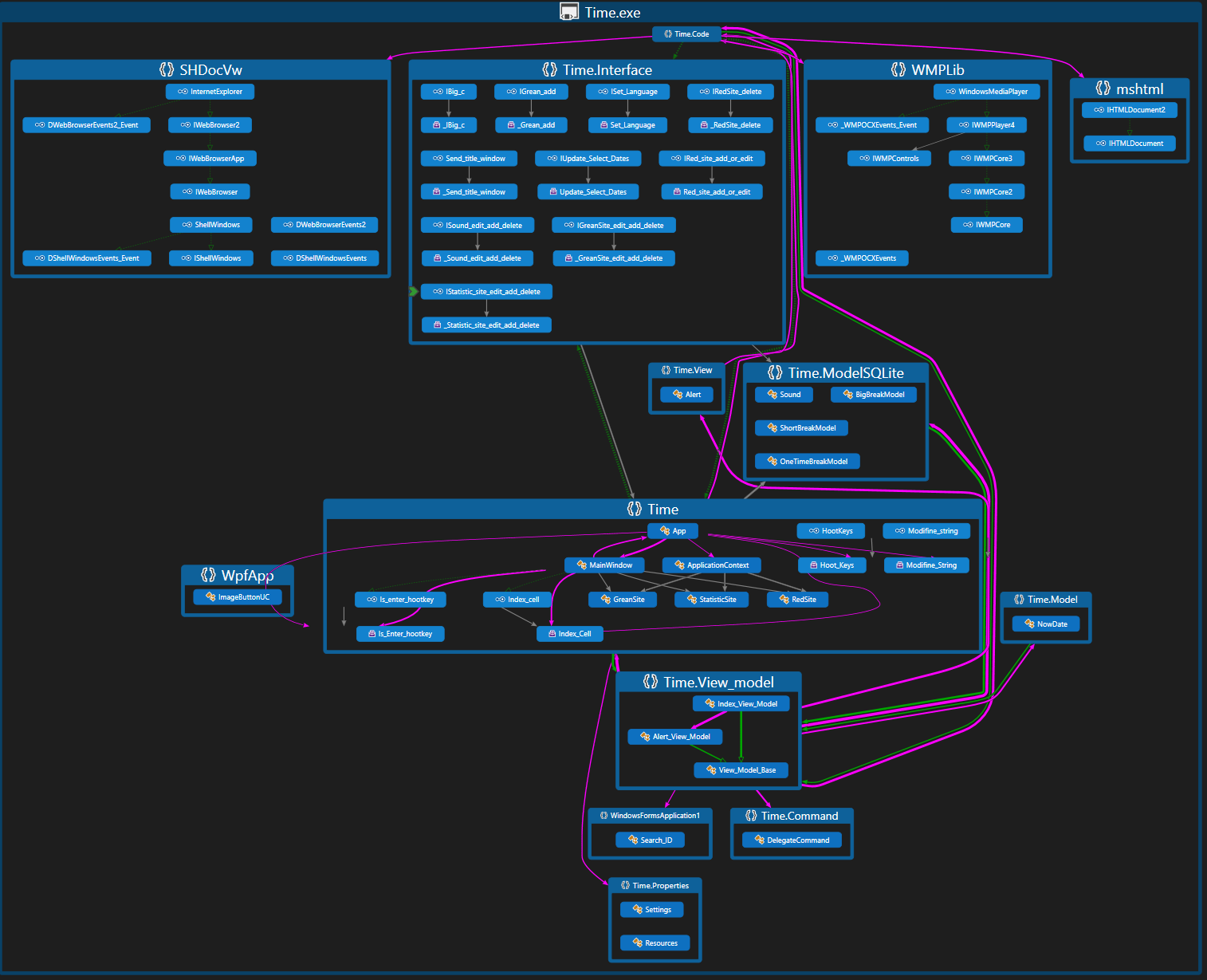
**Одесса 2019**

**АННОТАЦИЯ**

Программа предоставляет следующие возможности пользователям при работе с портативным компьютером.

1. Пользователь может выставлять интервал, время, и тип перерыва(быстрый,долгий).
2. Не позволяет пользователю заходить на сайты, которые он отметил как нежелательные при работе.
3. Сохранение нужных сайтов для работы в нужных браузерах (выставление того в каком браузере открыть тот или иной сайт)
4. Синхронизация с календарём google.
5. Ведение статистики и её сохранение.
6. Google аккаунт.

Диаграмма классов



Код

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Forms;

namespace Time.Code

{

public class globalKeyboardHook : Modifine\_string, HootKeys

{

public delegate int keyboardHookProc(int code, int wParam, ref keyboardHookStruct lParam);

public struct keyboardHookStruct

{

public int vkCode;

public int scanCode;

public int flags;

public int time;

public int dwExtraInfo;

}

const int WH\_KEYBOARD\_LL = 13;

const int WM\_KEYDOWN = 0x100;

const int WM\_KEYUP = 0x101;

const int WM\_SYSKEYDOWN = 0x104;

const int WM\_SYSKEYUP = 0x105;

public List<Keys> HookedKeys = new List<Keys>();

IntPtr hhook = IntPtr.Zero;

IntPtr hhook\_site = IntPtr.Zero;

public event System.Windows.Forms.KeyEventHandler KeyDown;

public event System.Windows.Forms.KeyEventHandler KeyUp;

public event Modifine\_String \_Modifine\_string;

public event Hoot\_Keys hoot\_Keys;

Convert\_Key \_convert;

string last\_hootkey = "";

int? \_Index\_Cell;

public void Set\_index\_cell(int? i)

{

try

{

\_Index\_Cell = i;

}

catch

{

\_Index\_Cell = null;

}

}

bool \_is\_edit\_Cell;

public void Is\_edit\_Cell(bool i)

{

\_is\_edit\_Cell = i;

}

public globalKeyboardHook()

{

\_convert = new Convert\_Key();

hook();

}

~globalKeyboardHook()

{

unhook();

}

#region keyboar

private static keyboardHookProc callbackDelegate;

public void hook()

{

if (callbackDelegate != null) throw new InvalidOperationException("Can't hook more than once");

IntPtr hInstance = LoadLibrary("User32");

callbackDelegate = new keyboardHookProc(hookProc);

hhook = SetWindowsHookEx(WH\_KEYBOARD\_LL, callbackDelegate, hInstance, 0);

hhook\_site = SetWindowsHookEx(WH\_KEYBOARD\_LL, callbackDelegate, hInstance, 0);

if (hhook == IntPtr.Zero)

throw new Win32Exception();

}

public void unhook()

{

if (callbackDelegate == null) return;

bool ok = UnhookWindowsHookEx(hhook);

if (!ok)

throw new Win32Exception();

callbackDelegate = null;

}

public int hookProc(int code, int wParam, ref keyboardHookStruct lParam)

{

if (code >= 0)

{

Keys key = (Keys)lParam.vkCode;

if (!HookedKeys.Contains(key) && key != Keys.Enter && key != Keys.Back)

{

System.Windows.Forms.KeyEventArgs kea = new System.Windows.Forms.KeyEventArgs(key);

if (kea.KeyCode != Keys.Back

&& (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN))/\* && (KeyDown != null)\*/

{

if (\_is\_edit\_Cell && \_Index\_Cell == 1)

{

\_Modifine\_string.Invoke(\_convert.KeyDown(key.ToString()));

return 1;

}

else

{

try

{

var now\_hootkey = \_convert.KeyDown(key.ToString());

if (now\_hootkey.CompareTo(last\_hootkey) != 0)

hoot\_Keys.Invoke(now\_hootkey);

last\_hootkey = now\_hootkey;

}

catch

{

\_convert.Clean();

}

}

// KeyDown(this, kea);

}

else if (kea.KeyCode != Keys.Back && (wParam == WM\_KEYUP || wParam == WM\_SYSKEYUP) /\*&& (KeyUp != null)\*/)

{

if (\_is\_edit\_Cell && \_Index\_Cell == 1)

{

\_convert.KeyUp(key.ToString());

return 1;

}

else

{

\_convert.KeyUp(key.ToString());

}

// KeyUp(this, kea);

}

// if (kea.Handled)

// return 1;

}

}

return CallNextHookEx(hhook, code, wParam, ref lParam);

}

#endregion keyboar

[DllImport("user32.dll")]

static extern IntPtr SetWindowsHookEx(int idHook, keyboardHookProc callback, IntPtr hInstance, uint threadId);

[DllImport("user32.dll")]

static extern bool UnhookWindowsHookEx(IntPtr hInstance);

[DllImport("user32.dll")]

static extern int CallNextHookEx(IntPtr idHook, int nCode, int wParam, ref keyboardHookStruct lParam);

[DllImport("kernel32.dll")]

static extern IntPtr LoadLibrary(string lpFileName);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.Drawing;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace Time.Code

{

public static class MouseHook

{

#region Declarations

public static event System.Windows.Forms.MouseEventHandler MouseDown;

public static event MouseEventHandler MouseUp;

public static event MouseEventHandler MouseMove;

[StructLayout(LayoutKind.Sequential)]

struct MOUSEHOOKSTRUCT

{

public POINT pt;

public IntPtr hwnd;

public int wHitTestCode;

public IntPtr dwExtraInfo;

}

[StructLayout(LayoutKind.Sequential)]

struct MSLLHOOKSTRUCT

{

public POINT pt;

public int mouseData;

public int flags;

public int time;

public IntPtr dwExtraInfo;

}

[StructLayout(LayoutKind.Sequential)]

struct POINT

{

public int X;

public int Y;

public POINT(int x, int y)

{

this.X = x;

this.Y = y;

}

public static implicit operator Point(POINT p)

{

return new Point(p.X, p.Y);

}

public static implicit operator POINT(Point p)

{

return new POINT(p.X, p.Y);

}

}

const int WM\_LBUTTONDOWN = 0x201;

const int WM\_LBUTTONUP = 0x202;

const int WM\_MOUSEMOVE = 0x0200;

const int WM\_MOUSEWHEEL = 0x020A;

const int WM\_RBUTTONDOWN = 0x0204;

const int WM\_RBUTTONUP = 0x0205;

const int WM\_MBUTTONUP = 0x208;

const int WM\_MBUTTONDOWN = 0x207;

const int WM\_XBUTTONDOWN = 0x20B;

const int WM\_XBUTTONUP = 0x20C;

static IntPtr hHook = IntPtr.Zero;

static IntPtr hModule = IntPtr.Zero;

static bool hookInstall = false;

static bool localHook = false;

static API.HookProc hookDel;

#endregion

/// <summary>

/// Hook install method.

/// </summary>

public static void InstallHook()

{

if (IsHookInstalled) return;

hModule = Marshal.GetHINSTANCE(AppDomain.CurrentDomain.GetAssemblies()[0].GetModules()[0]);

hookDel = new API.HookProc(HookProcFunction);

if (localHook)

hHook = API.SetWindowsHookEx(API.HookType.WH\_MOUSE,

hookDel, IntPtr.Zero, AppDomain.GetCurrentThreadId()); // Если подчеркивает необращай внимание, так надо.

else

hHook = API.SetWindowsHookEx(API.HookType.WH\_MOUSE\_LL,

hookDel, hModule, 0);

if (hHook != IntPtr.Zero)

hookInstall = true;

else

throw new Win32Exception("Can't install low level keyboard hook!");

}

/// <summary>

/// If hook installed return true, either false.

/// </summary>

public static bool IsHookInstalled

{

get { return hookInstall && hHook != IntPtr.Zero; }

}

/// <summary>

/// Module handle in which hook was installed.

/// </summary>

public static IntPtr ModuleHandle

{

get { return hModule; }

}

/// <summary>

/// If true local hook will installed, either global.

/// </summary>

public static bool LocalHook

{

get { return localHook; }

set

{

if (value != localHook)

{

if (IsHookInstalled)

throw new Win32Exception("Can't change type of hook than it install!");

localHook = value;

}

}

}

/// <summary>

/// Uninstall hook method.

/// </summary>

public static void UnInstallHook()

{

if (IsHookInstalled)

{

if (!API.UnhookWindowsHookEx(hHook))

throw new Win32Exception("Can't uninstall low level keyboard hook!");

hHook = IntPtr.Zero;

hModule = IntPtr.Zero;

hookInstall = false;

}

}

/// <summary>

/// Hook process messages.

/// </summary>

/// <param name="nCode"></param>

/// <param name="wParam"></param>

/// <param name="lParam"></param>

/// <returns></returns>

static IntPtr HookProcFunction(int nCode, IntPtr wParam, IntPtr lParam)

{

if (nCode == 0)

{

MSLLHOOKSTRUCT mhs = (MSLLHOOKSTRUCT)Marshal.PtrToStructure(lParam, typeof(MSLLHOOKSTRUCT));

switch (wParam.ToInt32())

{

case WM\_LBUTTONDOWN:

if (MouseDown != null)

MouseDown(null,

new MouseEventArgs(MouseButtons.Left,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_LBUTTONUP:

if (MouseUp != null)

MouseUp(null,

new MouseEventArgs(MouseButtons.Left,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_MBUTTONDOWN:

if (MouseDown != null)

MouseDown(null,

new MouseEventArgs(MouseButtons.Middle,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_MBUTTONUP:

if (MouseUp != null)

MouseUp(null,

new MouseEventArgs(MouseButtons.Middle,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_MOUSEWHEEL:

if (!localHook)

{

if (MouseMove != null)

MouseMove(null,

new MouseEventArgs(MouseButtons.None, mhs.time,

mhs.pt.X, mhs.pt.Y, mhs.mouseData >> 16));

}

break;

case WM\_RBUTTONDOWN:

if (MouseDown != null)

MouseDown(null,

new MouseEventArgs(MouseButtons.Right,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_RBUTTONUP:

if (MouseUp != null)

MouseUp(null,

new MouseEventArgs(MouseButtons.Right,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_XBUTTONDOWN:

if (MouseDown != null)

MouseDown(null,

new MouseEventArgs(API.HIWORD(mhs.mouseData) == 1 ? MouseButtons.XButton1 : MouseButtons.XButton2,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

case WM\_XBUTTONUP:

if (MouseUp != null)

MouseUp(null,

new MouseEventArgs(API.HIWORD(mhs.mouseData) == 1 ? MouseButtons.XButton1 : MouseButtons.XButton2,

1,

mhs.pt.X,

mhs.pt.Y,

0));

break;

default:

break;

}

}

return API.CallNextHookEx(hHook, nCode, wParam, lParam);

}

}

static class API

{

public delegate IntPtr HookProc(int nCode, IntPtr wParam, [In] IntPtr lParam);

[DllImport("user32.dll")]

public static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode, IntPtr wParam, [In] IntPtr lParam);

[DllImport("user32.dll", SetLastError = true)]

public static extern IntPtr SetWindowsHookEx(HookType hookType, HookProc lpfn,

IntPtr hMod, int dwThreadId);

[DllImport("user32.dll", SetLastError = true)]

public static extern bool UnhookWindowsHookEx(IntPtr hhk);

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern IntPtr GetModuleHandle(string lpModuleName);

public enum HookType : int

{

WH\_JOURNALRECORD = 0,

WH\_JOURNALPLAYBACK = 1,

WH\_KEYBOARD = 2,

WH\_GETMESSAGE = 3,

WH\_CALLWNDPROC = 4,

WH\_CBT = 5,

WH\_SYSMSGFILTER = 6,

WH\_MOUSE = 7,

WH\_HARDWARE = 8,

WH\_DEBUG = 9,

WH\_SHELL = 10,

WH\_FOREGROUNDIDLE = 11,

WH\_CALLWNDPROCRET = 12,

WH\_KEYBOARD\_LL = 13,

WH\_MOUSE\_LL = 14

}

public static int LOWORD(int x)

{

return x & 0xffff;

}

public static int HIWORD(int x)

{

return (x >> 16) & 0xffff;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using Time.Interface;

namespace Time.Code

{

// Добавить textBox1 на форму

delegate void WinEventDelegate(IntPtr hWinEventHook, uint eventType, IntPtr hwnd, int idObject, int idChild, uint dwEventThread, uint dwmsEventTime);

public partial class My\_form : Send\_title\_window

{

WinEventDelegate dele;

IntPtr m\_hhook;

public My\_form()

{

// dele = new WinEventDelegate(WinEventProc);

// m\_hhook = SetWinEventHook(EVENT\_SYSTEM\_FOREGROUND, EVENT\_SYSTEM\_FOREGROUND, IntPtr.Zero, dele, 0, 0, WINEVENT\_OUTOFCONTEXT);

}

private void WinEventProc(IntPtr hWinEventHook, uint eventType, IntPtr hwnd, int idObject, int idChild, uint dwEventThread, uint dwmsEventTime)

{

string wTitle = GetActiveWindowTitle();

send\_title\_window.Invoke(wTitle);

}

public string GetActiveWindowTitle()

{

const int nChars = 256;

IntPtr handle = IntPtr.Zero;

StringBuilder Buff = new StringBuilder(nChars);

handle = GetForegroundWindow();

if (GetWindowText(handle, Buff, nChars) > 0)

{

return Buff.ToString();

}

return null;

}

[DllImport("user32.dll")]

static extern IntPtr SetWinEventHook(uint eventMin, uint eventMax, IntPtr hmodWinEventProc, WinEventDelegate lpfnWinEventProc, uint idProcess, uint idThread, uint dwFlags);

private const uint WINEVENT\_OUTOFCONTEXT = 0;

private const uint EVENT\_SYSTEM\_FOREGROUND = 3;

public event \_Send\_title\_window send\_title\_window;

[DllImport("user32.dll")]

static extern IntPtr GetForegroundWindow();

[DllImport("user32.dll")]

static extern int GetWindowText(IntPtr hWnd, StringBuilder text, int count);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Code

{

public class Convert\_Key

{

public void Clean()

{

my\_Hootkey.Clear();

hootkey = "";

}

List<string> my\_Hootkey;

string hootkey = "";

public Convert\_Key(){

hootkey = "";

my\_Hootkey = new List<string>();

}

public string KeyDown(string key)

{

if (my\_Hootkey.Find(x => x == key) == null)

{

my\_Hootkey.Add(key);

hootkey = "";

my\_Hootkey.ForEach(x => { hootkey += x + "+"; });

hootkey = hootkey.Substring(0, hootkey.Length - 1);

}

return hootkey;

}

public void KeyUp(string key)

{

my\_Hootkey.Remove(key);

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace WindowsFormsApplication1

{

public partial class Search\_ID

{

public Search\_ID()

{

}

[DllImport("user32.dll", SetLastError = true)]

static extern int GetWindowText(IntPtr hWnd, StringBuilder lpString, int nMaxCount);

[DllImport("USER32.DLL", CharSet = CharSet.Unicode)]

public static extern IntPtr FindWindow(string lpClassName,

string lpWindowName);

[DllImport("USER32.DLL")]

public static extern bool SetForegroundWindow(IntPtr hWnd);

delegate bool EnumWindowsProc(IntPtr hWnd, IntPtr lParam);

[DllImport("user32.dll", SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

static extern bool EnumWindows(EnumWindowsProc lpEnumFunc, IntPtr lParam);

[DllImport("user32.dll", SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

static extern bool IsWindowVisible(IntPtr hWnd);

[DllImport("user32.dll", SetLastError = true)]

static extern int GetWindowTextLength(IntPtr hWnd);

List<IntPtr> ListHandles = new List<IntPtr>();

public void btn\_Click()

{

EnumWindows((hWnd, lParam) =>

{

if ((IsWindowVisible(hWnd) && GetWindowTextLength(hWnd) != 0) && GetWindowText(hWnd).StartsWith("Калькулятор"))

{

ListHandles.Add(hWnd);

foreach (var calc in ListHandles)

{

SetForegroundWindow(calc);

//действие

}

}

return true;

}, IntPtr.Zero);

}

string GetWindowText(IntPtr hWnd)

{

int len = GetWindowTextLength(hWnd) + 1;

StringBuilder sb = new StringBuilder(len);

len = GetWindowText(hWnd, sb, len);

return sb.ToString(0, len);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Code

{

class Site\_opening

{

static public void Open(string i)

{

System.Diagnostics.Process.Start(i);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Automation;

using System.Diagnostics;

using System.Threading;

using NDde.Client;

using System.Web;

using System.IO;

using OpenQA.Selenium;

using OpenQA.Selenium.Firefox;

namespace Time.Code

{

class SiteBloc

{

#region Chrome

public static string GetURL(string i)

{

if (i != null)

{

string my\_browser = Parse\_str(i);

if (my\_browser.CompareTo(" Google Chrome") == 0)

{

try

{

AutomationElement root =null;

AutomationElement textP=null;

object vpi = null;

var outer = Task.Factory.StartNew(() => // внешняя задача

{

try

{

root = AutomationElement.RootElement.FindFirst(TreeScope.Children, new PropertyCondition(AutomationElement.ClassNameProperty, "Chrome\_WidgetWin\_1"));

textP = root.FindFirst(TreeScope.Descendants, new PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Edit));

vpi = textP.GetCurrentPropertyValue(ValuePatternIdentifiers.ValueProperty);

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message + " || " + "GetURL Chrome " + i + "//");

#endif

}

});

outer.Wait();

return vpi.ToString();

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message + " || " + "GetURL Chrome "+ i+"//");

#endif

}

}

else if (my\_browser.CompareTo(" Internet Explorer") == 0)

{

try

{

string myLocalLink = null;

var outer = Task.Factory.StartNew(() => // внешняя задача

{

SHDocVw.InternetExplorer browser;

mshtml.IHTMLDocument2 myDoc;

SHDocVw.ShellWindows shellWindows = new SHDocVw.ShellWindows();

string filename;

foreach (SHDocVw.InternetExplorer ie in shellWindows)

{

filename = System.IO.Path.GetFileNameWithoutExtension(ie.FullName).ToLower();

if ((filename == "iexplore"))

{

browser = ie;

myDoc = browser.Document;

myLocalLink = myDoc.url;

break;

}

}

});

outer.Wait();

return myLocalLink;

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message + " || " + "GetURL iexplore " + i);

}

}

else if (my\_browser.CompareTo(" Mozilla Firefox") == 0)

{

var proz = Process.GetProcesses();

for(int num=0; num < proz.Length-1; num++)

{

try

{

string temp\_str = GetBrowsedUrl(proz[num]);

if(temp\_str!=null)

{

return temp\_str;

}

}

catch (Exception ex) {

System.Windows.MessageBox.Show(ex.Message + " || " + "GetURL Firefox " + i);

}

}

}

}

return null;

}

static string GetBrowsedUrl(Process process)

{

if (process.ProcessName == "firefox")

{

if (process == null)

return null;

if (process.MainWindowHandle == IntPtr.Zero)

return null;

AutomationElement element = AutomationElement.FromHandle(process.MainWindowHandle);

if (element == null)

return null;

var doc = element.FindFirst(TreeScope.Subtree, new PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Document));

if (doc == null)

return null;

string \_url\_ = null;

\_url\_ = ((ValuePattern)doc.GetCurrentPattern(ValuePattern.Pattern)).Current.Value as string;

return \_url\_;

//var doc = element.FindAll(TreeScope.Subtree, new PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Document));

//if (doc == null)

// return null;

//string \_url\_=null;

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

// \_url\_=((ValuePattern) doc[i].GetCurrentPattern(ValuePattern.Pattern)).Current.Value as string;

//return \_url\_;

}

return null;

}

#endregion chrome

static string Parse\_str(string i)

{

string temp = null;

string[] words = i.Split(new char[] { '-' });

temp = words[words.Length-1];

return temp;

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace Time.Code

{

public class BindingErrorTraceListener : TraceListener

{

private readonly StringBuilder \_messageBuilder = new StringBuilder();

public override void Write(string message)

{

\_messageBuilder.Append(message);

}

public override void WriteLine(string message)

{

try

{

Write(message);

MessageBox.Show(\_messageBuilder.ToString(), "Binding error", MessageBoxButton.OK, MessageBoxImage.Warning);

\_messageBuilder.Clear();

}

catch

{

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace Time.Code

{

public static class PasswordBoxAssistant

{

public static readonly DependencyProperty BoundPassword =

DependencyProperty.RegisterAttached("BoundPassword", typeof(string), typeof(PasswordBoxAssistant), new PropertyMetadata(string.Empty, OnBoundPasswordChanged));

public static readonly DependencyProperty BindPassword = DependencyProperty.RegisterAttached(

"BindPassword", typeof(bool), typeof(PasswordBoxAssistant), new PropertyMetadata(false, OnBindPasswordChanged));

private static readonly DependencyProperty UpdatingPassword =

DependencyProperty.RegisterAttached("UpdatingPassword", typeof(bool), typeof(PasswordBoxAssistant), new PropertyMetadata(false));

private static void OnBoundPasswordChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)

{

PasswordBox box = d as PasswordBox;

// only handle this event when the property is attached to a PasswordBox

// and when the BindPassword attached property has been set to true

if (d == null || !GetBindPassword(d))

{

return;

}

// avoid recursive updating by ignoring the box's changed event

box.PasswordChanged -= HandlePasswordChanged;

string newPassword = (string)e.NewValue;

if (!GetUpdatingPassword(box))

{

box.Password = newPassword;

}

box.PasswordChanged += HandlePasswordChanged;

}

private static void OnBindPasswordChanged(DependencyObject dp, DependencyPropertyChangedEventArgs e)

{

// when the BindPassword attached property is set on a PasswordBox,

// start listening to its PasswordChanged event

PasswordBox box = dp as PasswordBox;

if (box == null)

{

return;

}

bool wasBound = (bool)(e.OldValue);

bool needToBind = (bool)(e.NewValue);

if (wasBound)

{

box.PasswordChanged -= HandlePasswordChanged;

}

if (needToBind)

{

box.PasswordChanged += HandlePasswordChanged;

}

}

private static void HandlePasswordChanged(object sender, RoutedEventArgs e)

{

PasswordBox box = sender as PasswordBox;

// set a flag to indicate that we're updating the password

SetUpdatingPassword(box, true);

// push the new password into the BoundPassword property

SetBoundPassword(box, box.Password);

SetUpdatingPassword(box, false);

}

public static void SetBindPassword(DependencyObject dp, bool value)

{

dp.SetValue(BindPassword, value);

}

public static bool GetBindPassword(DependencyObject dp)

{

return (bool)dp.GetValue(BindPassword);

}

public static string GetBoundPassword(DependencyObject dp)

{

return (string)dp.GetValue(BoundPassword);

}

public static void SetBoundPassword(DependencyObject dp, string value)

{

dp.SetValue(BoundPassword, value);

}

private static bool GetUpdatingPassword(DependencyObject dp)

{

return (bool)dp.GetValue(UpdatingPassword);

}

private static void SetUpdatingPassword(DependencyObject dp, bool value)

{

dp.SetValue(UpdatingPassword, value);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Code

{

public enum Type\_alert { Big,Short,One,Message}

public class AlertType

{

public int time { set; get; }

public Type\_alert type { set; get; }

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using WMPLib;

namespace Time.Code

{

class MusicPath

{

static WindowsMediaPlayer wmp = new WindowsMediaPlayer();

public static List<string> Get\_Paths()

{

FolderBrowserDialog folderBrowserDialog1 = new FolderBrowserDialog();

DialogResult result = folderBrowserDialog1.ShowDialog();

if (result == DialogResult.OK)

{

string folderName = folderBrowserDialog1.SelectedPath;

return Directory.GetFiles(folderName, "\*.mp3", SearchOption.TopDirectoryOnly).ToList();

}

return null;

}

public static void Play(string path)

{

Stop();

if (path != null)

{

wmp.URL = path;

wmp.controls.play();

}

}

public static void Stop()

{

wmp.controls.stop();

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Media;

namespace Time.Code

{

public class StoreDataSetPaginator : DocumentPaginator

{

private List<StatisticSite> dt;

private Typeface typeface;

private double fontSize;

private double margin;

private Size pageSize;

public override Size PageSize

{

get

{

return pageSize;

}

set

{

pageSize = value;

PaginateData();

}

}

public StoreDataSetPaginator(List<StatisticSite> statisticSites, Typeface typeface, double fontSize, double margin, Size pageSize)

{

this.dt = statisticSites;

this.typeface = typeface;

this.fontSize = fontSize;

this.margin = margin;

this.pageSize = pageSize;

PaginateData();

}

private int pageCount;

private int rowsPerPage;

private void PaginateData()

{

// Создать тестовую строку для измерения

FormattedText text = GetFormattedText("A");

// Подсчитать строки, которые умещаются на странице

rowsPerPage = (int)((pageSize.Height - margin \* 2) / text.Height);

// Оставить строку для заголовка

rowsPerPage -= 1;

pageCount = (int)Math.Ceiling((double)dt.Count / rowsPerPage);

}

private FormattedText GetFormattedText(string text)

{

return GetFormattedText(text, typeface);

}

private FormattedText GetFormattedText(string text, Typeface typeface)

{

return new FormattedText(

text, CultureInfo.CurrentCulture, FlowDirection.LeftToRight,

typeface, fontSize, Brushes.Black);

}

// Всегда возвращает true, потому что количество страниц обновляется

// немедленно и синхронно, когда изменяется размер страницы.

// Никогда не находится в неопределенном состоянии

public override bool IsPageCountValid

{

get { return true; }

}

public override int PageCount

{

get { return pageCount; }

}

public override IDocumentPaginatorSource Source

{

get { return null; }

}

public override DocumentPage GetPage(int pageNumber)

{

// Создать тестовую строку для измерения

FormattedText text = GetFormattedText("A");

text.MaxTextWidth = 30;

// Размеры столбцов относительно ширины символа "A"

double col1\_X = margin;

double col2\_X = col1\_X + text.Width \* 30;

// Вычислить диапазон строк, которые попадают в эту страницу

int minRow = pageNumber \* rowsPerPage;

int maxRow = minRow + rowsPerPage;

// Создать визуальный элемент для страницы

DrawingVisual visual = new DrawingVisual();

// Установить позицию в верхний левый угол печатаемой области

Point point = new Point(margin, margin);

using (DrawingContext dc = visual.RenderOpen())

{

// Нарисовать заголовки столбцов

Typeface columnHeaderTypeface = new Typeface(typeface.FontFamily, FontStyles.Normal, FontWeights.Bold, FontStretches.Normal);

point.X = col1\_X;

text = GetFormattedText("Name", columnHeaderTypeface);

dc.DrawText(text, point);

text = GetFormattedText("Type", columnHeaderTypeface);

point.X = col2\_X;

dc.DrawText(text, point);

// Нарисовать линию подчеркивания

dc.DrawLine(new Pen(Brushes.Black, 2),

new Point(margin, margin + text.Height),

new Point(pageSize.Width - margin, margin + text.Height));

point.Y += text.Height;

// Нарисовать значения столбцов

for (int i = minRow; i < maxRow; i++)

{

// Проверить конец последней (частично заполненной) страницы

if (i > (dt.Count - 1)) break;

point.X = col1\_X;

text = GetFormattedText(dt[i].Name.ToString());

dc.DrawText(text, point);

// Добавить второй столбец

text = GetFormattedText(dt[i].Status.ToString());

point.X = col2\_X;

dc.DrawText(text, point);

point.Y += text.Height;

}

}

return new DocumentPage(visual, pageSize, new Rect(pageSize), new Rect(pageSize));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time.Code

{

class Test\_Element : View\_Model\_Base

{

string name;

public string Name

{

get

{

return name;

}

set

{

name = value;

OnPropertyChanged(nameof(Name));

}

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace Time.Code

{

class TopWindow

{

[DllImport("user32.dll", SetLastError = true)]

public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

[DllImport("user32.dll", SetLastError = true)]

static extern long SetWindowPos(IntPtr hWnd, IntPtr hWndInsertAfter, int X, int Y, int cx, int cy, uint uFlags);

[DllImport("user32.dll")]

[return: MarshalAs(UnmanagedType.Bool)]

static extern bool SetForegroundWindow(IntPtr hWnd);

[DllImport("user32.dll")]

[return: MarshalAs(UnmanagedType.Bool)]

static extern bool ShowWindow(IntPtr hWnd, int showWindowCommand);

public void Top\_Window(string name)

{

uint flags = 0x0001 | 0x0002;

IntPtr w = FindWindow(null, name);

SetWindowPos(w, (IntPtr)(-1), 1, 1, 0, 0, flags);

}

public void Top\_All\_Window(string name)

{

Process[] procs = Process.GetProcessesByName(name);

foreach (Process p in procs)

{

ShowWindow(p.MainWindowHandle, 1);

SetForegroundWindow(p.MainWindowHandle);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

namespace Time.Command

{

public class DelegateCommand : ICommand

{

public DelegateCommand(Action<object> execute, Predicate<object> canExecute)

{

\_execute = execute ?? throw new ArgumentNullException("execute");

\_canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return \_canExecute != null ? \_canExecute(parameter) : true;

}

public void Execute(object parameter)

{

\_execute(parameter);

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

Action<object> \_execute;

Predicate<object> \_canExecute;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time

{

public delegate void Hoot\_Keys(string i);

interface HootKeys

{

event Hoot\_Keys hoot\_Keys;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void \_IBig\_c();

public interface IBig\_c

{

event \_IBig\_c big\_c;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void \_Grean\_add();

public interface IGrean\_add

{

event \_Grean\_add Grean\_add;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void \_GreanSite\_edit\_add\_delete(List<GreanSite> element);

public interface IGreanSite\_edit\_add\_delete

{

event \_GreanSite\_edit\_add\_delete greanSite\_edit\_add\_delete;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time

{

public delegate void Index\_Cell(int? i);

public interface Index\_cell

{

event Index\_Cell index\_cell;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void Red\_site\_add\_or\_edit(bool \_is);

public interface IRed\_site\_add\_or\_edit

{

event Red\_site\_add\_or\_edit red\_site\_add\_or\_edit;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void \_RedSite\_delete(List<RedSite> element);

public interface IRedSite\_delete

{

event \_RedSite\_delete red\_site\_delete;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time

{

public delegate void Is\_Enter\_hootkey(bool i);

public interface Is\_enter\_hootkey

{

event Is\_Enter\_hootkey Is\_enter\_hootkey;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void Set\_Language(string i);

public interface ISet\_Language

{

event Set\_Language set\_Language\_;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.ModelSQLite;

namespace Time.Interface

{

public delegate void \_Sound\_edit\_add\_delete(List<Sound> i);

interface ISound\_edit\_add\_delete

{

event \_Sound\_edit\_add\_delete sound\_edit\_add\_delete;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void \_Statistic\_site\_edit\_add\_delete(List<StatisticSite> i);

interface IStatistic\_site\_edit\_add\_delete

{

event \_Statistic\_site\_edit\_add\_delete statistic\_site\_edit\_add\_delete;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time

{

public delegate void Modifine\_String(string i);

interface Modifine\_string

{

event Modifine\_String \_Modifine\_string;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void \_Send\_title\_window(string i);

interface Send\_title\_window

{

event \_Send\_title\_window send\_title\_window;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.Interface

{

public delegate void Update\_Select\_Dates(List<DateTime> i);

interface IUpdate\_Select\_Dates

{

event Update\_Select\_Dates Update\_select\_dates;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.Entity;

using Time.ModelSQLite;

namespace Time

{

class ApplicationContext : DbContext

{

public ApplicationContext() : base("DefaultConnection")

{

}

public DbSet<GreanSite> GreanSites { get; set; }

public DbSet<RedSite> RedSites { get; set; }

public DbSet<StatisticSite> StatisticSites { get; set; }

public DbSet<OneTimeBreakModel> OneTimeBreakModels { get; set; }

public DbSet<ShortBreakModel> ShortBreakModels { get; set; }

public DbSet<BigBreakModel> BigBreakModels { get; set; }

public DbSet<Sound> Sounds { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time.ModelSQLite

{

class BigBreakModel : View\_Model\_Base

{

public int Id { get; set; }

bool isActiveBig;

int numberTimeBig;

int numberDurationBig;

string oldTimeAtBreakBig;

int bigSoundId;

bool isActiveSound;

bool strictMode;

public bool IsActiveBig

{

get { return isActiveBig; }

set

{

isActiveBig = value;

OnPropertyChanged(nameof(IsActiveBig));

}

}

public int NumberTimeBig

{

get { return numberTimeBig; }

set

{

numberTimeBig = value;

OnPropertyChanged(nameof(NumberTimeBig));

}

}

public int NumberDurationBig

{

get { return numberDurationBig; }

set

{

numberDurationBig = value;

OnPropertyChanged(nameof(NumberDurationBig));

}

}

public string OldTimeAtBreakBig

{

get { return oldTimeAtBreakBig; }

set

{

oldTimeAtBreakBig = value;

OnPropertyChanged(nameof(OldTimeAtBreakBig));

}

}

public int BigSoundId

{

get { return bigSoundId; }

set

{

bigSoundId = value;

OnPropertyChanged(nameof(BigSoundId));

}

}

public bool IsActiveSound

{

get { return isActiveSound; }

set

{

isActiveSound = value;

OnPropertyChanged(nameof(IsActiveSound));

}

}

public bool StrictMode

{

get { return strictMode; }

set

{

strictMode = value;

OnPropertyChanged(nameof(StrictMode));

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time

{

public class GreanSite: View\_Model\_Base

{

private string name;

private string hootkey;

private string url;

public int Id { get; set; }

public string Name

{

set

{

name = value;

OnPropertyChanged(nameof(Name));

}

get

{

return name;

}

}

public string Hootkey

{

set

{

hootkey = value;

OnPropertyChanged("Hootkey");

}

get

{

return hootkey;

}

}

public string URL

{

set

{

url = value;

OnPropertyChanged("URL");

}

get

{

return url;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time.ModelSQLite

{

class OneTimeBreakModel : View\_Model\_Base

{

private bool is\_active\_one;

private string time\_break\_one;

private string duration\_break\_one;

private string old\_time\_at\_break\_one;

private int oneSoundId;

private bool isSoundActive;

public int Id { get; set; }

public bool IsActiveOne

{

set

{

is\_active\_one = value;

OnPropertyChanged(nameof(IsActiveOne));

}

get

{

return is\_active\_one;

}

}

public string TimeBreakOne

{

set

{

time\_break\_one = value;

OnPropertyChanged("TimeBreakOne");

}

get

{

return time\_break\_one;

}

}

public string DurationBreakOne

{

set

{

duration\_break\_one = value;

OnPropertyChanged("DurationBreakOne");

}

get

{

return duration\_break\_one;

}

}

public string OldTimeAtBreakOne

{

set

{

old\_time\_at\_break\_one = value;

OnPropertyChanged("OldTimeAtBreakOne");

}

get

{

return old\_time\_at\_break\_one;

}

}

public int OneSoundId

{

set

{

oneSoundId = value;

OnPropertyChanged("OneSoundId");

}

get

{

return oneSoundId;

}

}

public bool IsSoundActive

{

set

{

isSoundActive = value;

OnPropertyChanged("IsSoundActive");

}

get

{

return isSoundActive;

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time

{

public class RedSite : View\_Model\_Base

{

private string name;

private string url;

public int Id { get; set; }

public string Name

{

set

{

name = value;

OnPropertyChanged(nameof(Name));

}

get

{

return name;

}

}

public string URL

{

set

{

url = value;

OnPropertyChanged("URL");

}

get

{

return url;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time.ModelSQLite

{

class ShortBreakModel : View\_Model\_Base

{

public int Id { get; set; }

private bool isActiveShort;

private int numberTimeShort;

private int numberDurationShort;

private string oldTimeAtBreakShort;

private int shortSoundId;

private bool isSoundActive;

public bool IsActiveShort

{

get

{

return isActiveShort;

}

set

{

isActiveShort = value;

OnPropertyChanged(nameof(IsActiveShort));

}

}

public int NumberTimeShort

{

get

{

return numberTimeShort;

}

set

{

numberTimeShort = value;

OnPropertyChanged(nameof(NumberTimeShort));

}

}

public int NumberDurationShort

{

get

{

return numberDurationShort;

}

set

{

numberDurationShort = value;

OnPropertyChanged(nameof(NumberDurationShort));

}

}

public string OldTimeAtBreakShort

{

get

{

return oldTimeAtBreakShort;

}

set

{

oldTimeAtBreakShort = value;

OnPropertyChanged(nameof(OldTimeAtBreakShort));

}

}

public int ShortSoundId

{

get

{

return shortSoundId;

}

set

{

shortSoundId = value;

OnPropertyChanged(nameof(ShortSoundId));

}

}

public bool IsSoundActive

{

get

{

return isSoundActive;

}

set

{

isSoundActive = value;

OnPropertyChanged(nameof(IsSoundActive));

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time.ModelSQLite

{

public class Sound : View\_Model\_Base

{

private string path;

private string name;

public int Id { get; set; }

public string Path

{

set

{

path = value;

OnPropertyChanged(nameof(Path));

}

get

{

return path;

}

}

public string Name

{

set

{

name = value;

OnPropertyChanged(nameof(Name));

}

get

{

return name;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Time.View\_model;

namespace Time

{

public class StatisticSite : View\_Model\_Base

{

private string name;

private string time;

private string status;

private string url;

public int Id { get; set; }

public string Name

{

set

{

name = value;

OnPropertyChanged(nameof(Name));

}

get

{

return name;

}

}

public string Time

{

set

{

time = value;

OnPropertyChanged(nameof(Time));

}

get

{

return time;

}

}

public string Status

{

set

{

status = value;

OnPropertyChanged(nameof(Status));

}

get

{

return status;

}

}

public string URL

{

set

{

url = value;

OnPropertyChanged(nameof(URL));

}

get

{

return url;

}

}

}

}

using MahApps.Metro.Controls;

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 Time.Code;

namespace Time.View

{

/// <summary>

/// Interaction logic for Alert.xaml

/// </summary>

public partial class Alert : Window

{

TopWindow \_top\_windows = new TopWindow();

public Alert()

{

InitializeComponent();

}

public Action StartBreakBig;

public Type\_alert type;

private void MetroWindow\_Activated(object sender, EventArgs e)

{

}

#region CANCEL\_1

public void Activate\_CANCEL\_1()

{

Alert\_button\_cancel1.Visibility = Visibility.Visible;

}

public void Deactivate\_CANCEL\_1()

{

Alert\_button\_cancel1.Visibility = Visibility.Collapsed;

}

#endregion

#region OK

public void Activ\_OK()

{

Alert\_button\_ok.Visibility = Visibility.Visible;

}

public void Deactiv\_OK()

{

Alert\_button\_ok.Visibility = Visibility.Collapsed;

}

#endregion

#region NEXT

public void Activ\_NEXT()

{

Alert\_button\_next.Visibility = Visibility.Visible;

}

public void Deactiv\_NEXT()

{

Alert\_button\_next.Visibility = Visibility.Collapsed;

}

#endregion

#region Cancel

public void Activ\_CANCEL()

{

Alert\_button\_cancel.Visibility = Visibility.Visible;

}

public void Deactiv\_CANCEL()

{

Alert\_button\_cancel.Visibility = Visibility.Collapsed;

}

#endregion

#region Style window

public void Activ\_Style()

{

GWindow.WindowStyle = WindowStyle.ToolWindow;

}

public void Deactiv\_Style()

{

GWindow.WindowStyle = WindowStyle.None;

}

#endregion

#region Info small

public void Activated\_Small\_info()

{

Info.Visibility = Visibility.Visible;

}

public void Deactivated\_Small\_info()

{

Info.Visibility = Visibility.Collapsed;

}

#endregion

#region window

public void Center()

{

double screenHeight = SystemParameters.FullPrimaryScreenHeight + 70;

double screenWidth = SystemParameters.FullPrimaryScreenWidth;

this.Top = (screenHeight - this.Height) / 0x00000002;

this.Left = (screenWidth - this.Width) / 0x00000002;

}

public void Transareds()

{

GWindow.AllowsTransparency = true;

// GWindow.Background = new SolidColorBrush(Colors.Transparent);

this.Height = SystemParameters.FullPrimaryScreenHeight + 70;

this.Width = SystemParameters.FullPrimaryScreenWidth;

this.Opacity = 0.8;

this.Info.FontSize = 48;

this.Time.FontSize = 48;

this.title.FontSize = 28;

this.Alert\_button\_cancel1.Opacity = 2.0;

}

public void Activated\_message\_style()

{

Alert\_button\_next.Visibility = Visibility.Collapsed;

Time.Visibility = Visibility.Collapsed;

Alert\_button\_cancel.Visibility = Visibility.Collapsed;

Alert\_button\_ok.Visibility = Visibility.Collapsed;

Alert\_button\_ok1.Visibility = Visibility.Visible;

}

#endregion

private void GWindow\_Loaded(object sender, RoutedEventArgs e)

{

if (type != Type\_alert.Message)

{

\_top\_windows.Top\_Window("Alert");

\_top\_windows.Top\_All\_Window("Alert");

StartBreakBig();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using Time.Code;

using Time.Command;

namespace Time.View\_model

{

class Alert\_View\_Model : View\_Model\_Base

{

#region Actions

public Action Activate\_OK;

public Action Activate\_NEXT;

public Action Activate\_CANCEL;

public Action Activate\_CANCEL1;

public Action Activate\_Info\_small;

public Action Activate\_Info\_big;

public Action Deactivate\_OK;

public Action Deactivate\_NEXT;

public Action Deactivate\_CANCEL;

public Action Deactivate\_CANCEL1;

public Action Deactivate\_Info\_small;

public Action Deactivate\_Info\_big;

public Action Closenig;

public Action Disposes;

#endregion Actions

#region Variables

string text\_info;

public string Text\_info { get { return text\_info; } set { text\_info = value; OnPropertyChanged(nameof(Text\_info)); } }

public Type\_alert type { set; get; }

string times;

public string Times { get { return times; } set { times = value; OnPropertyChanged(nameof(Times)); } }

public double time\_s { set; get; }

public bool Message\_result=false;

TimerCallback big\_Callback;

Timer big\_timer;

#endregion Variables

#region Function

void TickBreak(object o)

{

try

{

App.Current.Dispatcher.Invoke((Action)delegate // <--- HERE

{

int x = (int)o;

time\_s = (time\_s - x);

int temp\_my\_s = (int)time\_s;

int temp\_second = 0;

int temp\_mitutes = 0;

for (; temp\_my\_s >= 60; temp\_my\_s -= 60)

{

temp\_mitutes++;

}

temp\_second = temp\_my\_s;

if (type == Type\_alert.Big)

{

if(temp\_second<10)

Times = String.Format("{0}:{1}{2}", temp\_mitutes,0, temp\_second);

else

Times = String.Format("{0}:{1}", temp\_mitutes, temp\_second);

}

else if (type == Type\_alert.Short)

Times = String.Format("{0}", temp\_second);

else if (type == Type\_alert.One)

{

if (temp\_second < 10)

Times = String.Format("{0}:{1}{2}", temp\_mitutes, 0, temp\_second);

else

Times = String.Format("{0}:{1}", temp\_mitutes, temp\_second);

if (time\_s < 1)

{

MusicPath.Stop();

Closenig();

Disposes();

big\_timer.Change(System.Threading.Timeout.Infinite, 0);

}

}

if (time\_s < 1)

{

MusicPath.Stop();

Closenig();

big\_timer.Change(System.Threading.Timeout.Infinite, 0);

}

//Times = Time;

});

}

catch (Exception ex)

{

MusicPath.Stop();

MessageBox.Show(ex.Message, "TickBreak");

}

}

public void ActiveteTime()

{

big\_Callback = new TimerCallback(TickBreak);

// создаем таймер

big\_timer = new System.Threading.Timer(big\_Callback, 1, 0, 1000);

}

#endregion Function

#region Command

#region cancel

private DelegateCommand \_Command\_button\_cancel;

public ICommand Button\_clik\_button\_cancel

{

get

{

if (\_Command\_button\_cancel == null)

{

\_Command\_button\_cancel = new DelegateCommand(Execute\_button\_cancel, CanExecute\_button\_cancel);

}

return \_Command\_button\_cancel;

}

}

private void Execute\_button\_cancel(object o)

{

if (type == Type\_alert.Big || type == Type\_alert.One || type == Type\_alert.Short)

{

MusicPath.Stop();

Closenig();

if (type == Type\_alert.One)

Disposes();

big\_timer.Change(System.Threading.Timeout.Infinite, 0);

}

}

private bool CanExecute\_button\_cancel(object o)

{

return true;

}

#endregion cancel

#region OK

private DelegateCommand \_Command\_button\_ok;

public ICommand Button\_clik\_button\_ok1

{

get

{

if (\_Command\_button\_ok == null)

{

\_Command\_button\_ok = new DelegateCommand(Execute\_button\_ok, CanExecute\_button\_ok);

}

return \_Command\_button\_ok;

}

}

private void Execute\_button\_ok(object o)

{

Message\_result = true;

Closenig();

}

private bool CanExecute\_button\_ok(object o)

{

return true;

}

#endregion OK

#region next

private DelegateCommand \_Command\_button\_next;

public ICommand Button\_clik\_button\_next

{

get

{

if (\_Command\_button\_next == null)

{

\_Command\_button\_next = new DelegateCommand(Execute\_button\_next, CanExecute\_button\_next);

}

return \_Command\_button\_next;

}

}

private void Execute\_button\_next(object o)

{

var i = 0;

}

private bool CanExecute\_button\_next(object o)

{

return true;

}

#endregion next

#endregion Command

}

}

#define test

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Data;

using System.Data.Entity;

using System.Diagnostics;

using System.Windows;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using Time.Code;

using Time.Command;

using WindowsFormsApplication1;

using System.Globalization;

using System.Windows.Forms;

using Time.ModelSQLite;

using System.Threading;

using Time.View;

using System.Runtime.InteropServices;

using System.IO;

using Google.Apis.Auth.OAuth2;

using Google.Apis.Calendar.v3;

using Google.Apis.Services;

using Google.Apis.Util.Store;

using Google.Apis.Calendar.v3.Data;

using Time.Model;

using Microsoft.Win32;

namespace Time.View\_model

{

class Index\_View\_Model : View\_Model\_Base

{

Search\_ID \_search\_ID;

ApplicationContext db;

ResourceDictionary dict = new ResourceDictionary();

public Index\_View\_Model()

{

}

public Index\_View\_Model(ApplicationContext temp)

{

\_search\_ID = new Search\_ID();

db = temp;

db.GreanSites.Load();

db.RedSites.Load();

db.StatisticSites.Load();

db.OneTimeBreakModels.Load();

db.ShortBreakModels.Load();

db.BigBreakModels.Load();

db.Sounds.Load();

//list\_GSite = db.GreanSites.ToList();

// list\_RSite = db.RedSites.ToList();

list\_StatisticSite = db.StatisticSites.ToList();

// list\_sounds = db.Sounds.ToList();

list\_Test\_list = new List<Test\_Element>();

my\_big\_model = new ObservableCollection<BigBreakModel>(db.BigBreakModels.ToList());

my\_short\_model = new ObservableCollection<ShortBreakModel>(db.ShortBreakModels.ToList());

my\_ome\_model = new ObservableCollection<OneTimeBreakModel>(db.OneTimeBreakModels.ToList());

List\_sound = new ObservableCollection<Sound>(db.Sounds.ToList());

Test\_list = new ObservableCollection<Test\_Element>(list\_Test\_list);

Grean = new ObservableCollection<GreanSite>(db.GreanSites.ToList());

Red = new ObservableCollection<RedSite>(db.RedSites.ToList());

StatisticSite = new ObservableCollection<Time.StatisticSite>(list\_StatisticSite);

My\_list = new ObservableCollection<NowDate>();

Select\_Index\_Sound\_type\_timer = -1;

// MouseHook.MouseDown += new System.Windows.Forms.MouseEventHandler(MouseHook\_MouseDown);

MouseHook.MouseUp += new System.Windows.Forms.MouseEventHandler(MouseHook\_MouseUp);

MouseHook.InstallHook();

Check\_Path();

if (Time\_timer\_textbox == "0")

{

Time\_timer\_textbox = "\_\_:\_\_:\_\_";

}

if (Time\_duration\_textbox == "0")

{

Time\_duration\_textbox = "\_\_:\_\_:\_\_";

}

}

public void View\_model\_up()

{

Is\_Small = Is\_Small;

Is\_small\_timer = Is\_small\_timer;

Is\_big\_timer = Is\_big\_timer;

Auto\_start = Auto\_start;

}

public void Set\_Language(string \_language)

{

switch (\_language)

{

case "ru-RU":

dict.Source = new Uri(String.Format("Resources/lang.{0}.xaml", \_language), UriKind.Relative);

break;

default:

dict.Source = new Uri("Resources/lang.xaml", UriKind.Relative);

break;

}

}

void MouseHook\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

Statistic\_procces();

}

#region Saves

void SaveGrean()

{

Grean.ToList().ForEach(y =>

{

if (y.Name == null)

y.Name = "None";

if (y.Hootkey == null)

y.Hootkey = "None";

if (y.URL == null)

y.URL = "None";

bool isAdd = true;

db.GreanSites.ToList().ForEach(x =>

{

try

{

if (y.Id == x.Id)

{

isAdd = false;

}

}

catch

{

isAdd = false;

}

});

if (isAdd)

db.GreanSites.Add(y);

});

db.SaveChanges();

}

void SaveRed()

{

Red.ToList().ForEach(y =>

{

if (y.Name == null)

y.Name = "None";

if (y.URL == null)

y.URL = "None";

bool isAdd = true;

db.RedSites.ToList().ForEach(x =>

{

try

{

if (y.Id == x.Id)

{

isAdd = false;

}

}

catch

{

isAdd = false;

}

});

if (isAdd)

db.RedSites.Add(y);

});

db.SaveChanges();

}

void SaveStatisticSite()

{

StatisticSite.ToList().ForEach(y =>

{

bool isAdd = true;

db.StatisticSites.ToList().ForEach(x =>

{

try

{

if (y.Id == x.Id)

{

isAdd = false;

}

}

catch

{

isAdd = false;

}

});

if (isAdd)

db.StatisticSites.Add(y);

});

db.SaveChanges();

}

void SaveSoundTimer()

{

List\_sound.ToList().ForEach(y =>

{

bool isAdd = true;

db.Sounds.ToList().ForEach(x =>

{

try

{

if (y.Id == x.Id)

{

isAdd = false;

}

}

catch

{

isAdd = false;

}

});

if (isAdd)

db.Sounds.Add(y);

});

db.SaveChanges();

}

void SaveTime()

{

}

public void Closing()

{

SaveGrean();

SaveRed();

SaveStatisticSite();

SaveSoundTimer();

MouseHook.UnInstallHook();

db.Dispose();

}

#endregion Saves

#region Timer

#region type time

int Get\_minutes\_type\_big(int i)

{

switch (i)

{

case 0:

#if test

return 1;//20

#endif

return 20;//20

case 1:

return 25;

case 2:

return 30;

case 3:

return 35;

case 4:

return 40;

case 5:

return 45;

case 6:

return 50;

case 7:

return 60;

case 8:

return 90;

case 9:

return 120;

}

return -1;

}

int Get\_minutes\_duration\_big(int i)

{

switch (i)

{

case 0:

#if test

return 1;//20

#endif

return 2;//20

case 1:

return 3;

case 2:

return 4;

case 3:

return 5;

case 4:

return 7;

case 5:

return 10;

case 6:

return 15;

case 7:

return 20;

}

return -1;

}

int Get\_minutes\_type\_short(int i)

{

switch (i)

{

case 0:

#if test

return 1;//20

#endif

return 5;//20

case 1:

return 6;

case 2:

return 10;

case 3:

return 15;

case 4:

return 20;

case 5:

return 30;

}

return -1;

}

int Get\_seconds\_duration\_short(int i)

{

switch (i)

{

case 0:

return 8;

case 1:

return 20;

}

return -1;

}

#endregion type time

#region Variables

#region is activ big timer

Alert viewBig;

Alert\_View\_Model view\_modelBig;

TimerCallback big\_Callback;

System.Threading.Timer big\_timer;

public bool Is\_big\_timer

{

get

{

return my\_big\_model[0].IsActiveBig;

}

set

{

try

{

if (value)

BigT\_D();

else

BigT\_A();

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Ups...Is\_Small");

#endif

}

my\_big\_model[0].IsActiveBig = value;

my\_big\_model[0].IsActiveBig = Set\_Timer\_big(my\_big\_model[0].IsActiveBig);

OnPropertyChanged(nameof(Is\_big\_timer));

}

}

public int Index\_time\_big

{

get

{

return my\_big\_model[0].NumberTimeBig;

}

set

{

my\_big\_model[0].NumberTimeBig = value;

OnPropertyChanged(nameof(Index\_time\_big));

}

}

public int Index\_duration\_big

{

get

{

return my\_big\_model[0].NumberDurationBig;

}

set

{

my\_big\_model[0].NumberDurationBig = value;

OnPropertyChanged(nameof(Index\_duration\_big));

}

}

#endregion is activ big timer

#region is activ small timer

Alert viewShort;

Alert\_View\_Model view\_modelShort;

TimerCallback short\_Callback;

System.Threading.Timer short\_timer;

public bool Is\_small\_timer

{

get

{

return my\_short\_model[0].IsActiveShort;

}

set

{

try

{

if (value)

Short\_D();

else

Short\_A();

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Ups...Is\_Small");

#endif

}

my\_short\_model[0].IsActiveShort = Set\_Timer\_short(value);

OnPropertyChanged(nameof(Is\_small\_timer));

}

}

public int Index\_time\_short

{

get

{

return my\_short\_model[0].NumberTimeShort;

}

set

{

my\_short\_model[0].NumberTimeShort = value;

OnPropertyChanged(nameof(Index\_time\_short));

}

}

public int Index\_duration\_short

{

get

{

return my\_short\_model[0].NumberDurationShort;

}

set

{

my\_short\_model[0].NumberDurationShort = value;

OnPropertyChanged(nameof(Index\_duration\_short));

}

}

#endregion is activ small timer

#region is activ strict type

public bool Is\_strict\_type

{

get

{

return my\_big\_model[0].StrictMode;

}

set

{

my\_big\_model[0].StrictMode = value;

OnPropertyChanged(nameof(Is\_strict\_type));

}

}

#endregion is activ strict type

#region small timer

Alert viewOne;

Alert\_View\_Model view\_modelOne;

TimerCallback One\_Callback;

System.Threading.Timer One\_timer;

public string Time\_timer\_textbox

{

get

{

return my\_ome\_model[0].TimeBreakOne;

}

set

{

if (value.Length < 9)

my\_ome\_model[0].TimeBreakOne = Converts(value);

OnPropertyChanged(nameof(Time\_timer\_textbox));

}

}

public string Time\_duration\_textbox

{

get

{

return my\_ome\_model[0].DurationBreakOne;

}

set

{

if (value.Length < 9)

my\_ome\_model[0].DurationBreakOne = Converts(value);

OnPropertyChanged(nameof(Time\_duration\_textbox));

}

}

string Converts(string str)

{

int second = 0;

int minutes = 0;

int hours = 0;

var list = str.Split(':');

if (list[2][0] != '\_')

{

var i = Convert.ToInt32(list[2][0].ToString()) \* 10;

if (i > 59)

{

i = i - 60;

minutes++;

}

second += i;

}

if (list[2][1] != '\_')

{

var i = Convert.ToInt32(list[2][1].ToString());

second += i;

}

if (list[1][0] != '\_')

{

var temp\_m = Convert.ToInt32(list[1][0].ToString()) \* 10;

if (temp\_m > 59)

{

hours++;

temp\_m = temp\_m - 60;

}

minutes += temp\_m;

}

if (list[1][1] != '\_')

{

var temp\_m = Convert.ToInt32(list[1][1].ToString());

minutes += temp\_m;

}

if (list[0][0] != '\_')

hours += Convert.ToInt32(list[0][0].ToString()) \* 10;

if (list[0][1] != '\_')

hours += Convert.ToInt32(list[0][1].ToString());

if (hours > 24)

hours = 24;

var end\_s = (second >= 10 ? second.ToString() : "\_" + (second == 0 ? "\_" : second.ToString()));

var end\_m = (minutes >= 10 ? minutes.ToString() : "\_" + (minutes == 0 ? "\_" : minutes.ToString()));

var end\_h = (hours >= 10 ? hours.ToString() : "\_" + (hours == 0 ? "\_" : hours.ToString()));

return String.Format("{0}:{1}:{2}", end\_h, end\_m, end\_s);

}

#endregion small timer

#region is one timer

public bool Is\_Small

{

get

{

return my\_ome\_model[0].IsActiveOne;

}

set

{

try

{

if (value)

OneTimes\_D();

else

OneTimes\_A();

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Ups...Is\_Small");

#endif

}

my\_ome\_model[0].IsActiveOne = Set\_Timer\_one(value);

OnPropertyChanged(nameof(Is\_Small));

}

}

#endregion is one timer

#region is activ sound

int select\_Index\_Sound\_type\_timer = -1;

public int Select\_Index\_Sound\_type\_timer

{

get

{

return select\_Index\_Sound\_type\_timer;

}

set

{

select\_Index\_Sound\_type\_timer = value;

OnPropertyChanged(nameof(Select\_Index\_Sound\_type\_timer));

OnPropertyChanged(nameof(Is\_Activ\_sound));

OnPropertyChanged(nameof(Selected\_Item\_sound));

}

}

public bool Is\_Activ\_sound

{

get

{

if (select\_Index\_Sound\_type\_timer == 0)

{

return my\_big\_model[0].IsActiveSound;

}

else if (select\_Index\_Sound\_type\_timer == 1)

{

return my\_short\_model[0].IsSoundActive;

}

else if (select\_Index\_Sound\_type\_timer == 2)

{

return my\_ome\_model[0].IsSoundActive;

}

return false;

}

set

{

if (select\_Index\_Sound\_type\_timer == 0)

{

my\_big\_model[0].IsActiveSound = value;

}

else if (select\_Index\_Sound\_type\_timer == 1)

{

my\_short\_model[0].IsSoundActive = value;

}

else if (select\_Index\_Sound\_type\_timer == 2)

{

my\_ome\_model[0].IsSoundActive = value;

}

OnPropertyChanged(nameof(Is\_Activ\_sound));

}

}

#endregion is activ sound

#endregion Variables

#region Function

bool Set\_Timer\_big(bool i)

{

try

{

if (Index\_time\_big != -1 && Index\_duration\_big != -1 && i)

{

if (big\_timer != null)

big\_timer.Change(System.Threading.Timeout.Infinite, 0);

int mitute\_time = Get\_minutes\_type\_big(Index\_time\_big);

int mitute\_duration = Get\_minutes\_duration\_big(Index\_duration\_big);

long mil = mitute\_time \* 60000;

AlertType my\_alert = new AlertType() { time = mitute\_duration, type = Type\_alert.Big };

big\_Callback = new TimerCallback(Alert\_box);

// создаем таймер

#if test

big\_timer = new System.Threading.Timer(big\_Callback, my\_alert, 0, mil);

#else

big\_timer = new System.Threading.Timer(big\_Callback, my\_alert, mil, mil);

#endif

return true;

}

else

{

if (!i && big\_timer != null)

big\_timer.Change(System.Threading.Timeout.Infinite, 0);

return false;

}

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Ups...");

#endif

return false;

}

}

bool Set\_Timer\_short(bool i)

{

try

{

if (Index\_time\_short != -1 && Index\_duration\_short != -1 && i)

{

if (short\_timer != null)

short\_timer.Dispose();

int mitute\_time = Get\_minutes\_type\_short(Index\_time\_short);

int seconde\_duration = Get\_seconds\_duration\_short(Index\_duration\_short);

long mil = mitute\_time \* 60000;

AlertType my\_alert = new AlertType() { time = seconde\_duration, type = Type\_alert.Short };

short\_Callback = new TimerCallback(Alert\_box);

// создаем таймер

#if test

short\_timer = new System.Threading.Timer(short\_Callback, my\_alert, 0, mil);

#else

short\_timer = new System.Threading.Timer(short\_Callback, my\_alert, mil, mil);

#endif

return true;

}

else

{

if (!i && short\_timer != null)

short\_timer.Dispose();

return false;

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message, "Ups...");

return false;

}

}

bool Set\_Timer\_one(bool i)

{

try

{

var times = DateTime.Parse(Time\_timer\_textbox.Replace('\_', '0')).TimeOfDay;

var duration = DateTime.Parse(Time\_duration\_textbox.Replace('\_', '0')).TimeOfDay;

if ((duration.Hours > 0 || duration.Minutes > 0 || duration.Seconds > 0) &&

(times.Hours > 0 || times.Minutes > 0 || times.Seconds > 0) && i)

{

if (One\_timer != null)

One\_timer.Dispose();

int seconde\_duration = duration.Hours \* 60 \* 60 + duration.Minutes \* 60 + duration.Seconds;

long mil = times.Hours \* 60 \* 60000 + times.Minutes \* 60000 + times.Seconds \* 1000;

AlertType my\_alert = new AlertType() { time = seconde\_duration, type = Type\_alert.One };

One\_Callback = new TimerCallback(Alert\_box);

// создаем таймер

#if test

One\_timer = new System.Threading.Timer(One\_Callback, my\_alert, 0, mil);

#else

One\_timer = new System.Threading.Timer(One\_Callback, my\_alert, mil, mil);

#endif

return true;

}

else

{

if (!i && One\_timer != null)

{

One\_timer.Dispose();

}

return false;

}

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Ups...Set\_Timer\_one");

#endif

return false;

}

}

void DisposesOne()

{

One\_timer.Dispose();

Is\_Small = false;

}

void Alert\_box(object obj)

{

var temp = (AlertType)obj;

App.Current.Dispatcher.Invoke((Action)delegate // <--- HERE

{

if (temp.type == Type\_alert.Big)

{

viewBig = new Alert();

view\_modelBig = new Alert\_View\_Model() { time\_s = temp.time \* 60, type = Type\_alert.Big };

viewBig.StartBreakBig = new Action(view\_modelBig.ActiveteTime);

viewBig.Deactiv\_Style();

viewBig.Transareds();

viewBig.Center();

view\_modelBig.Activate\_OK = new Action(viewBig.Activ\_OK);

view\_modelBig.Deactivate\_OK = new Action(viewBig.Deactiv\_OK);

view\_modelBig.Deactivate\_OK();

view\_modelBig.Activate\_NEXT = new Action(viewBig.Activ\_NEXT);

view\_modelBig.Deactivate\_NEXT = new Action(viewBig.Deactiv\_NEXT);

view\_modelBig.Deactivate\_NEXT();

view\_modelBig.Activate\_CANCEL = new Action(viewBig.Activ\_CANCEL);

view\_modelBig.Deactivate\_CANCEL = new Action(viewBig.Deactiv\_CANCEL);

view\_modelBig.Deactivate\_CANCEL();

view\_modelBig.Activate\_CANCEL1 = new Action(viewBig.Activate\_CANCEL\_1);

view\_modelBig.Deactivate\_CANCEL1 = new Action(viewBig.Deactivate\_CANCEL\_1);

view\_modelBig.Activate\_CANCEL1();

view\_modelBig.Activate\_Info\_small = new Action(viewBig.Activated\_Small\_info);

view\_modelBig.Deactivate\_Info\_small = new Action(viewBig.Deactivated\_Small\_info);

view\_modelBig.Text\_info = dict["Alert\_message\_1\_big"].ToString();

if (Is\_strict\_type)

{

view\_modelBig.Deactivate\_CANCEL1();

}

view\_modelBig.Closenig = new Action(viewBig.Close);

viewBig.DataContext = view\_modelBig;

viewBig.Show();

try

{

if (my\_big\_model[0].IsActiveSound && my\_big\_model[0].BigSoundId != -1)

MusicPath.Play(List\_sound.Where(x => x.Id == my\_big\_model[0].BigSoundId).First().Path);

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Alert\_box - my\_big\_model");

#endif

}

}

else if (temp.type == Type\_alert.Short)

{

viewShort = new Alert();

view\_modelShort = new Alert\_View\_Model() { time\_s = temp.time, type = Type\_alert.Short };

viewShort.StartBreakBig = new Action(view\_modelShort.ActiveteTime);

viewShort.Deactiv\_Style();

viewShort.Center();

view\_modelShort.Activate\_OK = new Action(viewShort.Activ\_OK);

view\_modelShort.Deactivate\_OK = new Action(viewShort.Deactiv\_OK);

view\_modelShort.Deactivate\_OK();

view\_modelShort.Activate\_NEXT = new Action(viewShort.Activ\_NEXT);

view\_modelShort.Deactivate\_NEXT = new Action(viewShort.Deactiv\_NEXT);

view\_modelShort.Deactivate\_NEXT();

view\_modelShort.Activate\_CANCEL = new Action(viewShort.Activ\_CANCEL);

view\_modelShort.Deactivate\_CANCEL = new Action(viewShort.Deactiv\_CANCEL);

view\_modelShort.Deactivate\_CANCEL();

view\_modelShort.Closenig = new Action(viewShort.Close);

view\_modelShort.Text\_info = dict["Alert\_message\_1"].ToString();

viewShort.DataContext = view\_modelShort;

viewShort.Show();

try

{

if (my\_short\_model[0].IsSoundActive && my\_short\_model[0].ShortSoundId != -1)

MusicPath.Play(List\_sound.Where(x => x.Id == my\_short\_model[0].ShortSoundId).First().Path);

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Alert\_box - my\_big\_model");

#endif

}

}

if (temp.type == Type\_alert.One)

{

viewOne = new Alert();

view\_modelOne = new Alert\_View\_Model() { time\_s = temp.time, type = Type\_alert.One };

viewOne.StartBreakBig = new Action(view\_modelOne.ActiveteTime);

viewOne.Deactiv\_Style();

viewOne.Center();

view\_modelOne.Activate\_OK = new Action(viewOne.Activ\_OK);

view\_modelOne.Deactivate\_OK = new Action(viewOne.Deactiv\_OK);

view\_modelOne.Deactivate\_OK();

view\_modelOne.Activate\_NEXT = new Action(viewOne.Activ\_NEXT);

view\_modelOne.Deactivate\_NEXT = new Action(viewOne.Deactiv\_NEXT);

view\_modelOne.Deactivate\_NEXT();

view\_modelOne.Activate\_CANCEL = new Action(viewOne.Activ\_CANCEL);

view\_modelOne.Deactivate\_CANCEL = new Action(viewOne.Deactiv\_CANCEL);

view\_modelOne.Closenig = new Action(viewOne.Close);

view\_modelOne.Disposes = new Action(DisposesOne);

view\_modelOne.Text\_info = dict["Alert\_message\_1"].ToString();

viewOne.DataContext = view\_modelOne;

try

{

if (my\_ome\_model[0].IsSoundActive && my\_ome\_model[0].OneSoundId != -1)

if (Selected\_Item\_sound != null)

MusicPath.Play(List\_sound.Where(x => x.Id == my\_ome\_model[0].OneSoundId).First().Path);

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message, "Alert\_box - my\_big\_model");

#endif

}

One\_timer.Change(System.Threading.Timeout.Infinite, 0);

Is\_Small = false;

viewOne.Show();

}

});

}

#endregion Function

#region List

#region sesion

ObservableCollection<OneTimeBreakModel> my\_ome\_model { set; get; }

ObservableCollection<ShortBreakModel> my\_short\_model { set; get; }

ObservableCollection<BigBreakModel> my\_big\_model { set; get; }

#endregion sesion

#region Sound list

//List<Sound> list\_sounds;

public ObservableCollection<Sound> List\_sound

{

get;

set;

}

public Sound Selected\_Item\_sound

{

get

{

if (select\_Index\_Sound\_type\_timer == 0)

{

var i = List\_sound.Where(x => x.Id == my\_big\_model[0].BigSoundId);

if (i != null && i.Count() > 0)

{

return i.First();

}

}

else if (select\_Index\_Sound\_type\_timer == 1)

{

var i = List\_sound.Where(x => x.Id == my\_short\_model[0].ShortSoundId);

if (i != null && i.Count() > 0)

{

return i.First();

}

}

else if (select\_Index\_Sound\_type\_timer == 2)

{

var i = List\_sound.Where(x => x.Id == my\_ome\_model[0].OneSoundId);

if (i != null && i.Count() > 0)

{

return i.First();

}

}

return List\_sound[0];

}

set

{

if (select\_Index\_Sound\_type\_timer == 0)

{

if (value != null)

my\_big\_model[0].BigSoundId = value.Id;

else

my\_big\_model[0].BigSoundId = -1;

}

else if (select\_Index\_Sound\_type\_timer == 1)

{

if (value != null)

my\_short\_model[0].ShortSoundId = value.Id;

else

my\_short\_model[0].ShortSoundId = -1;

}

else if (select\_Index\_Sound\_type\_timer == 2)

{

if (value != null)

my\_ome\_model[0].OneSoundId = value.Id;

else

my\_ome\_model[0].OneSoundId = -1;

}

OnPropertyChanged(nameof(Selected\_Item\_sound));

}

}

void Check\_Path()

{

List\_sound.ToList().ForEach(

x =>

{

string path = x.Path;

FileInfo fileInf = new FileInfo(path);

if (!fileInf.Exists)

{

List\_sound.Remove(x);

db.Sounds.Remove(x);

}

}

);

db.SaveChanges();

}

#endregion Sound list

#endregion List

#region Command

#region path

private DelegateCommand \_Command\_path;

public ICommand Button\_clik\_path

{

get

{

if (\_Command\_path == null)

{

\_Command\_path = new DelegateCommand(Execute\_path, CanExecute\_path);

}

return \_Command\_path;

}

}

private void Execute\_path(object o)

{

var list\_sound\_temp = MusicPath.Get\_Paths();

if (list\_sound\_temp != null && list\_sound\_temp.Count > 0)

list\_sound\_temp.ForEach(x => List\_sound.Add(new Sound() { Path = x, Name = Path.GetFileName(x) }));

}

private bool CanExecute\_path(object o)

{

return true;

}

#endregion path

#endregion Command

#region Action

public Action Short\_A;

public Action Short\_D;

public Action OneTimes\_A;

public Action OneTimes\_D;

public Action BigT\_A;

public Action BigT\_D;

#endregion Action

#endregion Timer

#region filter

#region List

#region Red

public ObservableCollection<RedSite> Red

{ get; set; }

//public List<RedSite> list\_RSite

//{ get; set; }

#endregion Red

#region grean

public ObservableCollection<GreanSite> Grean

{ get; set; }

//public List<GreanSite> list\_GSite

//{ get; set; }

GreanSite \_Selected\_Grean = null;

public GreanSite Selected\_Grean

{

get { return \_Selected\_Grean; }

set

{

\_Selected\_Grean = value;

OnPropertyChanged(nameof(Selected\_Grean));

}

}

#endregion grean

#region statistic

public ObservableCollection<StatisticSite> StatisticSite

{ get; set; }

public List<StatisticSite> list\_StatisticSite

{ get; set; }

#endregion statistic

#region Test

public ObservableCollection<Test\_Element> Test\_list

{ get; set; }

public List<Test\_Element> list\_Test\_list

{ get; set; }

#endregion Test

#endregion List

#region pole

string \_last\_url = "";

string \_name\_title;

public string name\_title

{

set

{

\_name\_title = value;

OnPropertyChanged(nameof(name\_title));

}

get

{

return \_name\_title;

}

}

string \_temp\_url;

#region Date

List<DateTime> my\_dates = new List<DateTime>();

string date\_statistic\_title = "";

public string Date\_statistic\_title

{

set

{

date\_statistic\_title = value;

OnPropertyChanged(nameof(Date\_statistic\_title));

}

get

{

return date\_statistic\_title;

}

}

DateTime date\_statistic = DateTime.Now;

public DateTime Date\_statistic

{

set

{

date\_statistic = value;

OnPropertyChanged(nameof(Date\_statistic));

}

get

{

return date\_statistic;

}

}

#endregion Date

#endregion

#region site

#region Hoot

public void Modefine\_string(string i)

{

if (Selected\_Grean != null)

{

Selected\_Grean.Hootkey = i;

OnPropertyChanged(nameof(Selected\_Grean));

}

}

public void Hoot\_keys(string i)

{

if (i != null)

{

Grean.ToList().ForEach(x =>

{

if (x.Hootkey.CompareTo(i) == 0)

{

Site\_opening.Open(x.URL);

}

});

}

}

#endregion Hoot

#region statistic

public void Update\_range\_date(List<DateTime> i)

{

my\_dates = i;

Date\_statistic\_title = my\_dates.First() != null ?

(my\_dates.First().ToString().Split(' ')[0] + (

my\_dates.Count > 1 ?

"-" + my\_dates[my\_dates.Count - 1].ToString().Split(' ')[0]

: " "))

: " ";

Set\_List\_Statistic();

}

private void Set\_List\_Statistic()

{

var lits\_temp = db.StatisticSites.ToList();

List<StatisticSite> list = new List<Time.StatisticSite>();

var start = (my\_dates.First());

if (start != null)

{

var end = (my\_dates[my\_dates.Count - 1]);

if (end != start)

{

if (start > end)

{

var temp = DateTime.Parse(start.ToString());

start = end;

end = temp;

}

for (var i = 0; i < lits\_temp.Count; i++)

{

if (DateTime.Parse(lits\_temp[i].Time.Split(' ')[0]) >= DateTime.Parse(start.ToString().Split(' ')[0])

&& DateTime.Parse(lits\_temp[i].Time.Split(' ')[0]) <= DateTime.Parse(end.ToString().Split(' ')[0]))

{

list.Add(lits\_temp[i]);

}

}

}

else

{

for (var i = 0; i < lits\_temp.Count; i++)

{

if (DateTime.Parse(lits\_temp[i].Time.Split(' ')[0]) <= DateTime.Parse(end.ToString().Split(' ')[0]))

{

list.Add(lits\_temp[i]);

}

}

}

}

if (list.ToList().Count > 0)

{

list\_StatisticSite = list.ToList();

StatisticSite = new ObservableCollection<Time.StatisticSite>(list\_StatisticSite);

}

else

{

list\_StatisticSite = new List<StatisticSite>();

StatisticSite = new ObservableCollection<Time.StatisticSite>(list\_StatisticSite);

}

OnPropertyChanged(nameof(StatisticSite));

}

private async void Statistic\_procces()

{

My\_form \_my\_form = new My\_form();

name\_title = \_my\_form.GetActiveWindowTitle();

try

{

\_temp\_url = SiteBloc.GetURL(name\_title);

if (\_temp\_url != null && \_last\_url != \_temp\_url)

{

\_last\_url = \_temp\_url;

await Task.Run(() => Add\_statistic(\_temp\_url));

}

}

catch (Exception ex)

{

#if test

System.Windows.MessageBox.Show(ex.Message + " || " + "MouseHook\_MouseDown");

#endif

}

}

string Get\_status(string url)

{

int len = Grean.ToList().Count;

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

{

if (url.IndexOf(Grean[i].URL) != -1)

{

return "Green";

}

}

len = Red.ToList().Count;

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

{

if (url.IndexOf(Red[i].URL) != -1)

{

return "Red";

}

}

return "Yellow";

}

void Add\_statistic(string url)

{

var outer = Task.Factory.StartNew(() => // внешняя задача

{

var temp = new StatisticSite();

temp.Status = Get\_status(url);

temp.URL = url;

temp.Time = DateTime.Now.ToString();

if (temp.Status.CompareTo("Grean") == 0)

{

App.Current.Dispatcher.Invoke((Action)delegate // <--- HERE

{

int numb = Grean.ToList().FindIndex(x => url.IndexOf(x.URL) != -1);

temp.Name = Grean.ToList()[numb].Name;

});

}

else if (temp.Status.CompareTo("Red") == 0)

{

App.Current.Dispatcher.Invoke((Action)delegate // <--- HERE

{

int numb = Red.ToList().FindIndex(x => url.IndexOf(x.URL) != -1);

Alert\_Red\_Site(Red.ToList()[numb].Name);

temp.Name = Red.ToList()[numb].Name;

});

}

else

{

temp.Name = name\_title;

}

App.Current.Dispatcher.Invoke((Action)delegate // <--- HERE

{

StatisticSite.Add(temp);

Test\_list.Add(new Test\_Element() { Name = name\_title });

OnPropertyChanged(nameof(StatisticSite));

SaveStatisticSite();

});

});

}

#endregion statistic

async void Alert\_Red\_Site(string name)

{

await Task.Run(() => {

App.Current.Dispatcher.Invoke((Action)delegate // <--- HERE

{

ShowMessage(name + " " + dict["Message\_text"].ToString());

});

});

/\*MessageBox.Show( dict["Message\_text"].ToString()+name, dict["Message\_title"].ToString())\*/

}

public void Set\_Edit\_Site(bool i)

{

SaveRed();

}

#endregion site

#region Command

#region \_Command\_print

private DelegateCommand \_Command\_print;

public ICommand Button\_clik\_print

{

get

{

if (\_Command\_print == null)

{

\_Command\_print = new DelegateCommand(Execute\_print, CanExecute\_print);

}

return \_Command\_print;

}

}

private void Execute\_print(object o)

{

System.Windows.Controls.PrintDialog printDialog = new System.Windows.Controls.PrintDialog();

if (printDialog.ShowDialog() == true)

{

// Вспомогательные действия для извлечения таблицы

// из XML-файла, используя ADO.NET

StoreDataSetPaginator paginator = new StoreDataSetPaginator(StatisticSite.ToList(),

new Typeface("Calibri"), 24, 96 \* 0.75,

new Size(printDialog.PrintableAreaWidth, printDialog.PrintableAreaHeight));

printDialog.PrintDocument(paginator, "Печать с помощью классов визуального уровня");

}

}

private bool CanExecute\_print(object o)

{

return true;

}

#endregion \_Command\_print

#region Delete\_Statistic

public void Delete\_Statistic(List<StatisticSite> element)

{

// db.StatisticSites.ToList().ForEach(x => db.StatisticSites.Remove(x));

if (ShowMessage(dict["Alert\_message\_delete"].ToString()))

{

Delete\_elements\_statistic(element);

}

}

void Delete\_elements\_statistic(List<StatisticSite> element)

{

element.ForEach(x =>

{

StatisticSite.Remove(x);

db.StatisticSites.Remove(x);

OnPropertyChanged(nameof(StatisticSite));

});

db.SaveChanges();

}

#endregion Delete\_Statistic

#region Delete\_Sound

public void Delete\_Sound(List<Sound> element)

{

// db.StatisticSites.ToList().ForEach(x => db.StatisticSites.Remove(x));

if (ShowMessage(dict["Alert\_message\_delete"].ToString()))

{

Delete\_elements\_sound(element);

}

}

void Delete\_elements\_sound(List<Sound> element)

{

element.ForEach(x =>

{

try

{

List\_sound.Remove(x);

db.Sounds.Remove(x);

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message, "Delete\_elements\_sound");

}

OnPropertyChanged(nameof(List\_sound));

});

db.SaveChanges();

}

#endregion Delete\_Sound

#region Delete\_Grean

public void Delete\_Grean(List<GreanSite> element)

{

// db.StatisticSites.ToList().ForEach(x => db.StatisticSites.Remove(x));

if (ShowMessage(dict["Alert\_message\_delete"].ToString()))

{

Delete\_elements\_grean(element);

}

}

void Delete\_elements\_grean(List<GreanSite> element)

{

element.ForEach(x =>

{

Grean.Remove(x);

db.GreanSites.Remove(x);

OnPropertyChanged(nameof(Grean));

});

db.SaveChanges();

}

#endregion Delete\_Grean

#region Delete\_Red

public void Delete\_Red(List<RedSite> element)

{

// db.StatisticSites.ToList().ForEach(x => db.StatisticSites.Remove(x));

if (ShowMessage(dict["Alert\_message\_delete"].ToString()))

{

Delete\_elements\_red(element);

}

}

bool ShowMessage(string info)

{

Alert message = new Alert();

message.type = Type\_alert.Message;

Alert\_View\_Model message\_model = new Alert\_View\_Model() { type = Type\_alert.Message };

message\_model.Closenig = new Action(message.Close);

message.Activated\_message\_style();

message\_model.Text\_info = info;

message.DataContext = message\_model;

message.ShowDialog();

return message\_model.Message\_result;

}

void Delete\_elements\_red(List<RedSite> element)

{

element.ForEach(x =>

{

Red.Remove(x);

db.RedSites.Remove(x);

OnPropertyChanged(nameof(Grean));

});

db.SaveChanges();

}

#endregion Delete\_Red

#region Add\_Grean

public void Add\_Grean()

{

Add\_elements\_Grean();

}

void Add\_elements\_Grean()

{

var i = Grean;

SaveGrean();

}

#endregion Add\_Grean

#endregion Command

#endregion filter

#region Settings

bool auto\_start;

public bool Auto\_start

{

get

{

return auto\_start;

}

set

{

RegistryKey saveKey = Registry.CurrentUser.OpenSubKey(@"SOFTWARE\Microsoft\Windows\CurrentVersion\Run\", true);

if (value || (value == false && saveKey.GetValue("Time") != null))

{

saveKey.SetValue("Time", System.Windows.Forms.Application.ExecutablePath);

saveKey.Close();

//RegistryKey key = Registry.CurrentUser.OpenSubKey(@"SOFTWARE\Microsoft\Windows\CurrentVersion\Run\", true);

//string i = System.Windows.Forms.Application.ExecutablePath;

//key.SetValue("Time", System.Windows.Forms.Application.ExecutablePath);

auto\_start = true;

}

else

{

// var key = Microsoft.Win32.Registry.CurrentUser.OpenSubKey(@"SOFTWARE\Microsoft\Windows\CurrentVersion\Run\", true);

if (saveKey.GetValue("Time") != null)

{

saveKey.DeleteValue("Time");

auto\_start = value;

}

}

OnPropertyChanged(nameof(Auto\_start));

}

}

#endregion

#region Event

CalendarService service;

void Login()

{

string[] Scopes = { CalendarService.Scope.CalendarReadonly };

string ApplicationName = "Google Calendar API .NET Quickstart";

UserCredential credential;

using (var stream =

new FileStream("credentials.json", FileMode.Open, FileAccess.Read))

{

// The file token.json stores the user's access and refresh tokens, and is created

// automatically when the authorization flow completes for the first time.

string credPath = "token.json";

credential = GoogleWebAuthorizationBroker.AuthorizeAsync(

GoogleClientSecrets.Load(stream).Secrets,

Scopes,

"user",

CancellationToken.None,

new FileDataStore(credPath, true)).Result;

Console.WriteLine("Credential file saved to: " + credPath);

}

// Create Google Calendar API service.

service = new CalendarService(new BaseClientService.Initializer()

{

HttpClientInitializer = credential,

ApplicationName = ApplicationName,

});

Set\_events(10.ToString(), DateTime.Now);

}

void Set\_events(string i, DateTime days)

{

EventsResource.ListRequest request = service.Events.List("primary");

request.TimeMin = new DateTime(days.Year, days.Month, days.Day, 0, 0, 0);

request.TimeMax = new DateTime(days.Year, days.Month, days.Day, 23, 0, 0);

request.ShowDeleted = false;

request.SingleEvents = true;

// request.MaxResults = int.Parse(i);

request.OrderBy = EventsResource.ListRequest.OrderByEnum.StartTime;

// List events.

Events events = request.Execute();

Logins = events.Summary;

if (events.Items != null && events.Items.Count > 0)

{

foreach (var eventItem in events.Items)

{

My\_list.Add(new NowDate() { Name = eventItem.Summary, Time = eventItem.Start.DateTime.ToString() });

OnPropertyChanged(nameof(My\_list));

}

}

}

string login = "";

public string Logins

{

get { return login; }

set { login = value; OnPropertyChanged(nameof(Logins)); }

}

public ObservableCollection<NowDate> My\_list { get; set; }

DateTime selected\_date = DateTime.Now;

public DateTime Selected\_date

{

get

{

return selected\_date;

}

set

{

My\_list.Clear();

selected\_date = value;

Set\_events(10.ToString(), selected\_date);

OnPropertyChanged(nameof(Selected\_date));

}

}

#region \_Command\_print

private DelegateCommand \_Command\_sing\_in;

public ICommand Button\_clik\_sing\_in

{

get

{

if (\_Command\_sing\_in == null)

{

\_Command\_sing\_in = new DelegateCommand(Execute\_sing\_in, CanExecute\_sing\_in);

}

return \_Command\_sing\_in;

}

}

private void Execute\_sing\_in(object o)

{

Login();

}

private bool CanExecute\_sing\_in(object o)

{

return true;

}

#endregion \_Command\_print

#endregion Event

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Time.View\_model

{

public abstract class View\_Model\_Base : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

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

}

}

}

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.Entity;

using System.Data.SQLite;

using System.Diagnostics;

using System.Globalization;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using Time.Code;

using Time.Interface;

using Time.View\_model;

namespace Time

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application, ISet\_Language

{

static App my\_this;

private static List<CultureInfo> m\_Languages = new List<CultureInfo>();

public event Set\_Language set\_Language\_;

public static List<CultureInfo> Languages

{

get

{

return m\_Languages;

}

}

public App()

{

my\_this = this;

PresentationTraceSources.DataBindingSource.Listeners.Add(new BindingErrorTraceListener());

PresentationTraceSources.DataBindingSource.Switch.Level = SourceLevels.Error;

App.LanguageChanged += App\_LanguageChanged;

m\_Languages.Clear();

m\_Languages.Add(new CultureInfo("en-US")); //Нейтральная культура для этого проекта

m\_Languages.Add(new CultureInfo("ru-RU"));

}

#region language

public static event EventHandler LanguageChanged;

public static CultureInfo Language

{

get

{

return System.Threading.Thread.CurrentThread.CurrentUICulture;

}

set

{

if (value == null) throw new ArgumentNullException("value");

if (value == System.Threading.Thread.CurrentThread.CurrentUICulture) return;

//1. Меняем язык приложения:

System.Threading.Thread.CurrentThread.CurrentUICulture = value;

//2. Создаём ResourceDictionary для новой культуры

ResourceDictionary dict = new ResourceDictionary();

my\_this.set\_Language\_.Invoke(value.Name);

switch (value.Name)

{

case "ru-RU":

dict.Source = new Uri(String.Format("Resources/lang.{0}.xaml", value.Name), UriKind.Relative);

break;

default:

dict.Source = new Uri("Resources/lang.xaml", UriKind.Relative);

break;

}

//3. Находим старую ResourceDictionary и удаляем его и добавляем новую ResourceDictionary

ResourceDictionary oldDict = (from d in Application.Current.Resources.MergedDictionaries

where d.Source != null && d.Source.OriginalString.StartsWith("Resources/lang.")

select d).First();

if (oldDict != null)

{

int ind = Application.Current.Resources.MergedDictionaries.IndexOf(oldDict);

Application.Current.Resources.MergedDictionaries.Remove(oldDict);

Application.Current.Resources.MergedDictionaries.Insert(ind, dict);

}

else

{

Application.Current.Resources.MergedDictionaries.Add(dict);

}

//4. Вызываем евент для оповещения всех окон.

LanguageChanged(Application.Current, new EventArgs());

}

}

private void App\_LanguageChanged(Object sender, EventArgs e)

{

Time.Properties.Settings.Default.DefaultLanguage = Language;

Time.Properties.Settings.Default.Save();

}

#endregion language

globalKeyboardHook KListener;

// My\_form \_my\_fomr;

private void Application\_Startup(object sender, StartupEventArgs e)

{

MainWindow view = new MainWindow();

ApplicationContext myLite = new ApplicationContext();

Index\_View\_Model viewModel = new Index\_View\_Model(myLite);

view.DataContext = viewModel;

view.closing = new Action(viewModel.Closing);

KListener = new globalKeyboardHook();

view.Is\_enter\_hootkey += new Is\_Enter\_hootkey(KListener.Is\_edit\_Cell);

view.Update\_select\_dates += new Interface.Update\_Select\_Dates(viewModel.Update\_range\_date);

view.statistic\_site\_edit\_add\_delete += new \_Statistic\_site\_edit\_add\_delete(viewModel.Delete\_Statistic);

view.sound\_edit\_add\_delete += new \_Sound\_edit\_add\_delete(viewModel.Delete\_Sound);

view.greanSite\_edit\_add\_delete += new \_GreanSite\_edit\_add\_delete(viewModel.Delete\_Grean);

view.Grean\_add += new \_Grean\_add(viewModel.Add\_Grean);

view.red\_site\_add\_or\_edit += new Red\_site\_add\_or\_edit(viewModel.Set\_Edit\_Site);

view.red\_site\_delete += new \_RedSite\_delete(viewModel.Delete\_Red);

view.index\_cell += new Index\_Cell(KListener.Set\_index\_cell);

KListener.hoot\_Keys += new Hoot\_Keys(viewModel.Hoot\_keys);

KListener.\_Modifine\_string += new Modifine\_String(viewModel.Modefine\_string);

viewModel.BigT\_A = new Action(view.BigT\_A);

viewModel.BigT\_D = new Action(view.BigT\_D);

viewModel.Short\_A = new Action(view.Short\_A);

viewModel.Short\_D = new Action(view.Short\_D);

viewModel.OneTimes\_A = new Action(view.OneTimes\_A);

viewModel.OneTimes\_D = new Action(view.OneTimes\_D);

view.view\_model\_up = new Action(viewModel.View\_model\_up);

set\_Language\_ += new Set\_Language(viewModel.Set\_Language);

set\_Language\_.Invoke("en-US");

view.ShowDialog();

}

private void Application\_Exit(object sender, ExitEventArgs e)

{

KListener.unhook();

m\_Languages.Clear();

PresentationTraceSources.DataBindingSource.Listeners.Clear();

}

}

}

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 MahApps.Metro.Controls;

using MahApps.Metro.Controls.Dialogs;

using System.Windows.Interop;

using System.Runtime.InteropServices;

using System.Globalization;

using Time.Interface;

using Time.View\_model;

using System.Data;

using System.Collections;

using Time.ModelSQLite;

namespace Time

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : MetroWindow, Index\_cell, Is\_enter\_hootkey,

IUpdate\_Select\_Dates, IRed\_site\_add\_or\_edit, IStatistic\_site\_edit\_add\_delete,

ISound\_edit\_add\_delete, IGrean\_add, IGreanSite\_edit\_add\_delete, IRedSite\_delete

{

public Action closing;

private System.Windows.Forms.NotifyIcon m\_notifyIcon;

public event Index\_Cell index\_cell;

public event Is\_Enter\_hootkey Is\_enter\_hootkey;

public event Update\_Select\_Dates Update\_select\_dates;

public event Red\_site\_add\_or\_edit red\_site\_add\_or\_edit;

public event \_Statistic\_site\_edit\_add\_delete statistic\_site\_edit\_add\_delete;

public event \_Sound\_edit\_add\_delete sound\_edit\_add\_delete;

public event \_Grean\_add Grean\_add;

public event \_GreanSite\_edit\_add\_delete greanSite\_edit\_add\_delete;

public event \_RedSite\_delete red\_site\_delete;

public MainWindow()

{

InitializeComponent();

// initialise code here

m\_notifyIcon = new System.Windows.Forms.NotifyIcon();

m\_notifyIcon.BalloonTipText = "The app has been minimised. Click the tray icon to show.";

m\_notifyIcon.BalloonTipTitle = "The App";

m\_notifyIcon.Text = "The App";

m\_notifyIcon.Icon = new System.Drawing.Icon("ic\_timer\_128\_28821.ico");

m\_notifyIcon.Click += new EventHandler(m\_notifyIcon\_Click);

App.LanguageChanged += LanguageChanged;

CultureInfo currLang = App.Languages[0];

//Заполняем меню смены языка:

menuLanguage.Items.Clear();

foreach (var lang in App.Languages)

{

MenuItem menuLang = new MenuItem();

menuLang.Header = lang.DisplayName;

menuLang.Tag = lang;

menuLang.IsChecked = lang.Equals(currLang);

menuLang.Click += ChangeLanguageClick;

menuLanguage.Items.Add(menuLang);

}

}

#region trey

private WindowState m\_storedWindowState = WindowState.Normal;

void OnStateChanged(object sender, EventArgs args)

{

if (WindowState == WindowState.Minimized)

{

Hide();

if (m\_notifyIcon != null)

m\_notifyIcon.ShowBalloonTip(2000);

}

else

m\_storedWindowState = WindowState;

}

void OnIsVisibleChanged(object sender, DependencyPropertyChangedEventArgs args)

{

CheckTrayIcon();

}

void m\_notifyIcon\_Click(object sender, EventArgs e)

{

Show();

WindowState = m\_storedWindowState;

}

void CheckTrayIcon()

{

ShowTrayIcon(!IsVisible);

}

void ShowTrayIcon(bool show)

{

if (m\_notifyIcon != null)

m\_notifyIcon.Visible = show;

}

#endregion

#region Language

private void LanguageChanged(Object sender, EventArgs e)

{

CultureInfo currLang = App.Language;

//Отмечаем нужный пункт смены языка как выбранный язык

foreach (MenuItem i in menuLanguage.Items)

{

CultureInfo ci = i.Tag as CultureInfo;

i.IsChecked = ci != null && ci.Equals(currLang);

}

}

private void ChangeLanguageClick(Object sender, EventArgs e)

{

MenuItem mi = sender as MenuItem;

if (mi != null)

{

CultureInfo lang = mi.Tag as CultureInfo;

if (lang != null)

{

App.Language = lang;

}

}

}

#endregion Language

private void MetroWindow\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

m\_notifyIcon.Dispose();

m\_notifyIcon = null;

closing();

}

#region List grean

private void LIST\_RowEditEnding(object sender, DataGridRowEditEndingEventArgs e)

{

Is\_enter\_hootkey.Invoke(false);

Grean\_add.Invoke();

}

private void LIST\_BeginningEdit(object sender, DataGridBeginningEditEventArgs e)

{

Is\_enter\_hootkey.Invoke(true);

}

private void LIST\_CurrentCellChanged(object sender, EventArgs e)

{

try

{

index\_cell.Invoke(LIST.CurrentCell.Column.DisplayIndex);

}

catch

{

}

}

#endregion List grean

#region List red

private void LIST\_R\_RowEditEnding(object sender, DataGridRowEditEndingEventArgs e)

{

red\_site\_add\_or\_edit.Invoke(true);

}

#endregion List red

#region Date

private void DateTimePicker\_SelectedDateChanged(object sender, TimePickerBaseSelectionChangedEventArgs<DateTime?> e)

{

}

private void PickerCombo\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

Comboboxes.SelectedIndex = -1;

}

private void DropCalendar\_SelectedDatesChanged(object sender, SelectionChangedEventArgs e)

{

try

{

Update\_select\_dates.Invoke(DropCalendar.SelectedDates.ToList());

}

catch (Exception ex) { MessageBox.Show(ex.Message, "Ups...DropCalendar\_SelectedDatesChanged"); }

}

#endregion Date

#region delegete

private void CheckBox\_Checked(object sender, RoutedEventArgs e)

{

}

private void CheckBox\_Checked\_1(object sender, RoutedEventArgs e)

{

}

private void Short\_Click(object sender, RoutedEventArgs e)

{

if (Short.IsChecked != null && Short.IsChecked == true)

{

ShortTime.IsEnabled = false;

ShortDuration.IsEnabled = false;

}

else

{

ShortTime.IsEnabled = true;

ShortDuration.IsEnabled = true;

}

}

public void Short\_A()

{

ShortTime.IsEnabled = true;

ShortDuration.IsEnabled = true;

}

public void Short\_D()

{

ShortTime.IsEnabled = false;

ShortDuration.IsEnabled = false;

}

private void OneTimes\_Click(object sender, RoutedEventArgs e)

{

if (OneTimes.IsChecked != null && OneTimes.IsChecked == true)

{

mask.IsEnabled = false;

mask1.IsEnabled = false;

}

else

{

mask.IsEnabled = true;

mask1.IsEnabled = true;

}

}

public void OneTimes\_A()

{

mask.IsEnabled = true;

mask1.IsEnabled = true;

}

public void OneTimes\_D()

{

mask.IsEnabled = false;

mask1.IsEnabled = false;

}

private void BigT\_Click(object sender, RoutedEventArgs e)

{

if (BigT.IsChecked != null && BigT.IsChecked == true)

{

BreeakT.IsEnabled = false;

BreeakD.IsEnabled = false;

strictmode.IsEnabled = false;

}

else

{

BreeakT.IsEnabled = true;

BreeakD.IsEnabled = true;

strictmode.IsEnabled = true;

}

}

public void BigT\_A()

{

BreeakT.IsEnabled = true;

BreeakD.IsEnabled = true;

strictmode.IsEnabled = true;

}

public void BigT\_D()

{

BreeakT.IsEnabled = false;

BreeakD.IsEnabled = false;

strictmode.IsEnabled = false;

}

#endregion

private void DataGrid\_RowEditEnding(object sender, DataGridRowEditEndingEventArgs e)

{

}

private void DataGrid\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Delete)

{

DataGrid dg = sender as DataGrid;

List<StatisticSite> my\_stat = new List<StatisticSite>();

IList rows = dg.SelectedItems;

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

{

my\_stat.Add((dg.SelectedItems[i]) as StatisticSite);

}

statistic\_site\_edit\_add\_delete.Invoke(my\_stat);

}

}

private void DataGrid\_Sound\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Delete)

{

DataGrid dg = sender as DataGrid;

List<Sound> my\_stat = new List<Sound>();

IList rows = dg.SelectedItems;

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

{

my\_stat.Add((dg.SelectedItems[i]) as Sound);

}

sound\_edit\_add\_delete.Invoke(my\_stat);

}

}

private void DataGrid\_Grean\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Delete)

{

DataGrid dg = sender as DataGrid;

List<GreanSite> my\_stat = new List<GreanSite>();

IList rows = dg.SelectedItems;

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

{

my\_stat.Add((dg.SelectedItems[i]) as GreanSite);

}

greanSite\_edit\_add\_delete.Invoke(my\_stat);

}

}

private void DataGrid\_Red\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Delete)

{

DataGrid dg = sender as DataGrid;

List<RedSite> my\_stat = new List<RedSite>();

IList rows = dg.SelectedItems;

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

{

my\_stat.Add((dg.SelectedItems[i]) as RedSite);

}

red\_site\_delete.Invoke(my\_stat);

}

}

bool is\_check = true;

public Action view\_model\_up;

private void MetroWindow\_Activated(object sender, EventArgs e)

{

if (is\_check)

{

view\_model\_up();

is\_check = false;

}

}

private void MetroWindow\_Loaded(object sender, RoutedEventArgs e)

{

}

}

}