using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Net;

using System.Text.RegularExpressions;

using System.Web.Compilation;

using DataCollectorScriptLibrary;

using Reporter.Model;

using Reporter.Model.ProcessedItems;

using User = DataCollectorScriptLibrary.User;

using DataCollectorScriptLibrary.DataSet;

using System.Globalization;

public class TRN\_ABC\_snippet : InnerCode

{

public TRN\_ABC\_snippet(CollectionContainer container)

: base(container)

{

}

#region Code

#region Configuration

/\*\*\*\*\*\* HierarchyCacheProviderConfig \*\*\*\*\*\*/

private ReportContext \_ctx;

private const string HierarchyCacheProviderItemSeparator = " >> ";

private const string HierarchyCacheProviderEncloserFormat = "{1} ({0})";

public HashSet<int> ExcludedTaskIds = new HashSet<int>();

public HashSet<ItemType> ExcludedWorkItemTypes = new HashSet<ItemType>();

public double TaskIdsWeight = 1;

public double ItemTypeImportanceWeight = 100;

public bool EnableNetting = true;

public bool EnableIgnoreHolidays = true;

public int lineCounter = 0;

public int TempUserId = 0;

int userGroupLevel1 = 1;

int userGroupLevel2 = 2;

public DateTime TempDateTime;

public DateTime localDate;

public DateTime localEndDate;

public DateTime ReportStartDate;

public DateTime ReportEndDate;

public string OtherTransaction = "Nem besorolt";

public Dictionary<int, int> PositiveTaskIds = new Dictionary<int, int>();

public Dictionary<int, int> NegativeTaskIds = new Dictionary<int, int>();

public Dictionary<string, dictNormaValue> dictNorma = new Dictionary<string, dictNormaValue>();

public Dictionary<string, bool> dictCounter = new Dictionary<string, bool>();

public Dictionary<string, Tuple<bool, DateTime, DateTime>> dictClosedItems = new Dictionary<string, Tuple<bool, DateTime, DateTime>>();

public Dictionary<TransactionNormKey, TranCountWithDuration> dictTransactionNorm = new Dictionary<TransactionNormKey, TranCountWithDuration>();

public Dictionary<string, TimeSpan> dictGlobalTransactionNorms = new Dictionary<string, TimeSpan>();

public Dictionary<DictTRtoIDXKey, DictTRtoIDXValue> copyOfdictKPI = new Dictionary<DictTRtoIDXKey, DictTRtoIDXValue>();

public Dictionary<string, string> WorkflowCode = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase);

public Dictionary<string, TranCountWithDuration> userTransactionNormDeviation = new Dictionary<string, TranCountWithDuration>();

public Dictionary<DictTRtoIDXKey, DictTRtoIDXValue> tempDictKPI = new Dictionary<DictTRtoIDXKey, DictTRtoIDXValue>();

public Dictionary<DictTRtoIDXKey, DictTRtoIDXValue> dictKPI = new Dictionary<DictTRtoIDXKey, DictTRtoIDXValue>();

public class dictNormaValue

{

public double normTime;

public int normCount;

public dictNormaValue()

{

normTime = 0.0;

normCount = 0;

}

}

/\*\*\*\*\*\* WorkitemExtConcatenatorConfig \*\*\*\*\*\*/

Dictionary<ItemType, int> ItemTypeImportance = new Dictionary<ItemType, int>

{

{ItemType.Holiday, 1},

{ItemType.SickLeave, 2},

{ItemType.Manual, 3},

{ItemType.CalendarMeeting, 4},

{ItemType.AdhocMeeting, 5},

{ItemType.Mobile, 6},

{ItemType.Pc, 7},

};

public List<string> OperandNames = new List<string>() { "NormDeviation", "TransactionTime", "CountOfTransaction" };

public bool IsNecessary;

public Dictionary<int, List<string>> ExternalHTMLContent = new Dictionary<int, List<string>>();

DictTRtoIDXValue dictValue;

public HashSet<string> TranNormExludingConditionSet = new HashSet<string>

{

"Produktív",

"OTHER",

"Nem besorolt"

};

public string InputFileName = "INPUT\_ABC\_Keywords 1";

public string InputKeywordSheetName = "kulcsszavak";

public string InputKeywordsRegexSheetName = "regex kulcsszavak";

public string InputUrlSheetName = "url";

public string InputSpecialEmailSheetName = "specialis-email";

public string InputEmailSheetName = "e-mail";

public bool hasClosedTransaction = false;

public string ClosedString = "Lezárt";

public string OpenedString = "Nyitott";

public const String langOther = " Other";

public const String langClosed = " Closed";

#endregion

#region OutputTypeSwitches

public bool OutputHtml = true;

public bool OutputKpi = true;

public bool OutputXls = true;

public bool OutputEV = true;

public bool OutputNorma = true;

public bool WriteDebug = true;

public bool bigData = false;

#endregion

public class heatmapkey : IEquatable<heatmapkey>

{

public int UserId;

public string Username;

public string keyString;

public bool Equals(heatmapkey other)

{

if (ReferenceEquals(null, other)) return false;

if (ReferenceEquals(this, other)) return true;

return UserId == other.UserId && keyString == other.keyString && Username == other.Username;

}

public override bool Equals(object obj)

{

if (ReferenceEquals(null, obj)) return false;

if (ReferenceEquals(this, obj)) return true;

if (obj.GetType() != this.GetType()) return false;

return Equals((heatmapkey)obj);

}

public override int GetHashCode()

{

var hashCode = UserId;

hashCode = (hashCode \* 397) ^ keyString.GetHashCode();

hashCode = (hashCode \* 397) ^ Username.GetHashCode();

return hashCode;

}

}

public class heatmapvalue

{

public HashSet<string> TransCountvalue;

public HashSet<string> pers;

public heatmapvalue()

{

TransCountvalue = new HashSet<string>();

pers = new HashSet<string>();

}

}

public Dictionary<heatmapkey, heatmapvalue> heatmapDictdaysCount = new Dictionary<heatmapkey, heatmapvalue>();

public Dictionary<heatmapkey, heatmapvalue> heatmapDictTrnCNCount = new Dictionary<heatmapkey, heatmapvalue>();

public Dictionary<string, HashSet<string>> InputKeywordsCollection = new Dictionary<string, HashSet<string>>();

public Dictionary<string, HashSet<string>> InputEmailsCollection = new Dictionary<string, HashSet<string>>();

public Dictionary<string, HashSet<string>> InputSpecialEmailsCollection = new Dictionary<string, HashSet<string>>();

public Dictionary<string, HashSet<string>> InputUrlsCollection = new Dictionary<string, HashSet<string>>();

public Dictionary<string, HashSet<Regex>> InputKeywordsRegexCollection = new Dictionary<string, HashSet<Regex>>();

public HashSet<string> InputKeySet = new HashSet<string>();

public void createTransactionHeatmapTrnCNCount()

{

if (dictMain.Count != 0)

{

HashSet<string> keySTR = new HashSet<string>();

foreach (var item in dictMain)

{

ValueDictStatus valst;

if (dictStatus.TryGetValue(item.Key.TransactionId, out valst))

{

string keyStringvar = item.Key.TransactionCategory + "-" + item.Key.TransactionName; //TODO: order by relevant key!!!

var key = new heatmapkey()

{

UserId = item.Key.UserId,

keyString = keyStringvar,

Username = Users.GetName(item.Key.UserId)

};

keySTR.Add(keyStringvar);

heatmapvalue countTRN;

if (!heatmapDictTrnCNCount.TryGetValue(key, out countTRN))

{

heatmapDictTrnCNCount.Add(key, new heatmapvalue());

}

heatmapDictTrnCNCount[key].TransCountvalue.Add(item.Key.TransactionCategory + "-" + item.Key.TransactionName + "-" + item.Key.TransactionId);

}

}

var outputCount = 0;

var sheet = helper.dataSet.Tables.Add("RAW\_trnHeatmaptrnCN");

sheet.Columns.Add("category");

var UserIdComparerByName = new IntComparerByRetreivedObject<string>(helper.GetUserName);

var UserIdList = \_ctx.UserIds;

UserIdList.Sort(UserIdComparerByName);

foreach (var user in UserIdList)

{

sheet.Columns.Add(string.Format("{0}", helper.GetUserName(user)));

}

if (heatmapDictTrnCNCount.Count != 0)

{

foreach (var keySTRItem in keySTR)

{

var row = sheet.NewRow();

var colIdx = 0;

row[colIdx++] = keySTRItem;

foreach (var userItem in UserIdList)

{

var key = new heatmapkey()

{

UserId = userItem,

keyString = keySTRItem,

Username = helper.GetUserName(userItem)

};

heatmapvalue countTRN;

if (heatmapDictTrnCNCount.TryGetValue(key, out countTRN))

{

row[colIdx++] = heatmapDictTrnCNCount[key].TransCountvalue.Count;

}

else

{

row[colIdx++] = 0;

}

}

sheet.Rows.Add(row);

helper.Log(row.ToString());

outputCount++;

}

}

sheet.Hide = true;

}

}

public void createTransactionHeatmapdaysCount()

{

if (dictMain.Count != 0)

{

HashSet<string> keySTR = new HashSet<string>();

foreach (var item in dictMain)

{

ValueDictStatus valst;

if (dictStatus.TryGetValue(item.Key.TransactionId, out valst))

{

DateTime dt = item.Key.DateSplit ?? DateTime.MinValue;

string keyStringvar = dt.ToString("dddd"); //TODO: order by relevant key!!!

var key = new heatmapkey()

{

UserId = item.Key.UserId,

keyString = keyStringvar,

Username = Users.GetName(item.Key.UserId)

};

keySTR.Add(keyStringvar);

heatmapvalue countTRN;

if (!heatmapDictdaysCount.TryGetValue(key, out countTRN))

{

heatmapDictdaysCount.Add(key, new heatmapvalue());

}

heatmapDictdaysCount[key].TransCountvalue.Add(item.Key.TransactionCategory + "-" + item.Key.TransactionName + "-" + item.Key.TransactionId + "-" + dt.ToString("d"));

heatmapDictdaysCount[key].pers.Add(dt.ToString("d"));

}

}

var outputCount = 0;

var sheet = helper.dataSet.Tables.Add("RAW\_trnHeatmapDC");

sheet.Columns.Add("category");

var UserIdComparerByName = new IntComparerByRetreivedObject<string>(helper.GetUserName);

var UserIdList = \_ctx.UserIds;

UserIdList.Sort(UserIdComparerByName);

foreach (var user in UserIdList)

{

sheet.Columns.Add(string.Format("{0}", helper.GetUserName(user)));

}

if (heatmapDictdaysCount.Count != 0)

{

foreach (var keySTRItem in keySTR)

{

var row = sheet.NewRow();

var colIdx = 0;

row[colIdx++] = keySTRItem;

foreach (var userItem in UserIdList)

{

var key = new heatmapkey()

{

UserId = userItem,

keyString = keySTRItem,

Username = helper.GetUserName(userItem)

};

heatmapvalue countTRN;

if (heatmapDictdaysCount.TryGetValue(key, out countTRN) && heatmapDictdaysCount[key].pers.Count != 0)

{

row[colIdx++] = heatmapDictdaysCount[key].TransCountvalue.Count / heatmapDictdaysCount[key].pers.Count;

}

else

{

row[colIdx++] = 0;

}

}

sheet.Rows.Add(row);

helper.Log(row.ToString());

outputCount++;

}

}

sheet.Hide = true;

}

}

public int ShortURLLength = 1;

public string GetURLShort(string URL, int MaxDepth = 3, bool RemoveProtocolInfo = true)

{

int MaxShortURLDepth = MaxDepth;

var URLWorkingCopy = URL;

var URLShorted = new System.Text.StringBuilder();

URLTypes UrlType = URLTypes.TypeNotSet;

if (!String.IsNullOrWhiteSpace(URL))

{

if (RemoveProtocolInfo)

{

if (URL.IndexOf("https://") > -1)

{

URLWorkingCopy = URL.Remove(0, "https://".Length);

UrlType = URLTypes.URL;

}

else if (URL.IndexOf("http://") > -1)

{

URLWorkingCopy = URL.Remove(0, "http://".Length);

UrlType = URLTypes.URL;

}

else if (URL.IndexOf("file://") > -1)

{

URLWorkingCopy = URL.Remove(0, "file://".Length);

UrlType = URLTypes.File;

}

}

if (UrlType == URLTypes.URL && URLWorkingCopy.IndexOf('/') > -1)

{

string[] shortURL = URLWorkingCopy.Split('/');

for (var i = 0; ((i < MaxShortURLDepth) && (i < shortURL.Length)); i++)

{

if (URLShorted.Length == 0)

{

URLShorted.Append(shortURL[i]);

}

else

{

URLShorted.AppendFormat("{0}{1}", '/', shortURL[i]);

}

}

}

else if (UrlType == URLTypes.File && URLWorkingCopy.IndexOf('\\') > -1)

{

return GetFilePathShort(URLWorkingCopy, MaxShortURLDepth);

}

else

{

return URLWorkingCopy;

}

}

return URLShorted.ToString();

}

public string GetFilePathShort(string FilePath, int MaxDepth = 3)

{

int MaxShortFilePathDepth = MaxDepth;

var FilePathWorkingCopy = FilePath;

var PathSorted = new System.Text.StringBuilder();

if (!String.IsNullOrWhiteSpace(FilePath))

{

if (FilePath.IndexOf('\\') > -1)

{

string[] shortFilePath = FilePath.Split('\\');

var maxLength = (shortFilePath.Length < MaxShortFilePathDepth

? shortFilePath.Length

: MaxShortFilePathDepth);

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

{

PathSorted.AppendFormat("{0}{1}", shortFilePath[i], "\\");

}

}

}

return PathSorted.ToString();

}

public void PreProcess()

{

\_ctx = helper.GetReportContext();

// inputfile read

LoadStringStringFromInput(InputFileName, InputKeywordSheetName, InputKeywordsCollection);

LoadStringStringFromInput(InputFileName, InputEmailSheetName, InputEmailsCollection);

LoadStringStringFromInput(InputFileName, InputSpecialEmailSheetName, InputSpecialEmailsCollection);

LoadStringStringFromInput(InputFileName, InputUrlSheetName, InputUrlsCollection);

LoadStringRegexFromInput(InputFileName, InputKeywordsRegexSheetName, InputKeywordsRegexCollection);

}

public void LoadStringStringFromInput(string InputFilename, string InputSheetname,

Dictionary<string, HashSet<string>> TargetCollection)

{

if (string.IsNullOrWhiteSpace(InputFilename) || string.IsNullOrWhiteSpace(InputSheetname) ||

TargetCollection == null) return;

//var helper = Configuration.ReportControl.GetRuntimeContext().helper;

var inputSheet = helper.GetTableFromFile(InputFilename, InputSheetname);

if ((inputSheet == null) || (inputSheet.Rows.Count == 0)) return;

foreach (var row in inputSheet.Rows)

{

var key = row[0].ToString();

var value = row[1].ToString();

if (string.IsNullOrWhiteSpace(key) || string.IsNullOrWhiteSpace(value)) continue;

HashSet<string> InputKeywordValue;

if (!TargetCollection.TryGetValue(key, out InputKeywordValue))

{

InputKeySet.Add(key);

TargetCollection.Add(key, new HashSet<string>());

InputKeywordValue = TargetCollection[key];

}

InputKeywordValue.Add(value);

}

}

private void LoadStringRegexFromInput(string InputFilename, string InputSheetname,

Dictionary<string, HashSet<Regex>> TargetCollection)

{

if (string.IsNullOrWhiteSpace(InputFilename) || string.IsNullOrWhiteSpace(InputSheetname) ||

TargetCollection == null) return;

var regexOptions = RegexOptions.IgnoreCase | RegexOptions.Compiled;

//var helper = Configuration.ReportControl.GetRuntimeContext().helper;

var inputSheet = helper.GetTableFromFile(InputFilename, InputSheetname);

if ((inputSheet == null) || (inputSheet.Rows.Count == 0)) return;

foreach (var row in inputSheet.Rows)

{

var key = row[0].ToString();

var value = row[1].ToString();

if (string.IsNullOrWhiteSpace(key) || string.IsNullOrWhiteSpace(value)) continue;

HashSet<Regex> InputIdPatternValue;

if (!InputKeywordsRegexCollection.TryGetValue(key, out InputIdPatternValue))

{

InputKeySet.Add(key);

TargetCollection.Add(key, new HashSet<Regex>());

InputIdPatternValue = TargetCollection[key];

}

InputIdPatternValue.Add(new Regex(value, regexOptions));

}

}

private HashSet<int> GetTaskIdsFromInputKeys()

{

var result = new HashSet<int>();

var keySet = new HashSet<string>(StringComparer.OrdinalIgnoreCase);

foreach (var key in InputKeySet)

{

if (string.IsNullOrWhiteSpace(key) || keySet.Contains(key))

continue;

var startIndex = key.IndexOf("(", StringComparison.Ordinal);

var lastIndex = key.IndexOf(")", StringComparison.Ordinal);

if (startIndex <= -1 || lastIndex <= -1)

continue;

startIndex = startIndex + 1;

int taskId;

int.TryParse(key.Substring(startIndex, lastIndex - startIndex), out taskId);

keySet.Add(key);

result.Add(taskId);

}

return result;

}

private HashSet<int> ReferencedTaskIds;

private Dictionary<int, List<int>> UsersTaskAssignmentsCache;

public Dictionary<int, List<int>> GetUsersTaskAssignments()

{

var result = new Dictionary<int, List<int>>();

foreach (var userId in \_ctx.UserIds)

{

List<int> taskAssignmentIds;

if (result.TryGetValue(userId, out taskAssignmentIds)) continue;

var userTaskAssignments = helper.GetTaskAssignmentsForUser(userId);

var IntersectOfInputAndUserTaskAssignments = userTaskAssignments

.Join(ReferencedTaskIds,

taskAssignment => taskAssignment.TaskId,

taskId => taskId,

(taskAssignment, taskId) => taskAssignment.TaskId).ToList();

result.Add(userId, IntersectOfInputAndUserTaskAssignments);

}

return result;

}

private bool TryMatchInput(string DataCollector, Dictionary<string, HashSet<string>> InputDictionary,

out string Key, out string MatchedString)

{

Key = "Non-Commodity";

MatchedString = "";

if (string.IsNullOrWhiteSpace(DataCollector))

return false;

foreach (var dictEntry in InputDictionary)

{

foreach (var hashSetItem in dictEntry.Value)

{

if (DataCollector.IndexOf(hashSetItem, StringComparison.OrdinalIgnoreCase) > -1)

{

Key = dictEntry.Key;

MatchedString = hashSetItem;

return true;

}

}

}

return false;

}

private bool TryMatchInputR(string DataCollector, Dictionary<string, HashSet<Regex>> InputDictionary,

out string Key, out string MatchedString)

{

Key = "Non-Commodity";

MatchedString = "";

if (string.IsNullOrWhiteSpace(DataCollector))

return false;

foreach (var dictEntry in InputDictionary)

{

foreach (var hashSetItem in dictEntry.Value)

{

var match = hashSetItem.Match(DataCollector);

if (match.Success)

{

Key = dictEntry.Key;

MatchedString = match.Value;

return true;

}

}

}

return false;

}

private string ConcatenateValuesOfTup(WorkItemExt Xtup)

{

List<string> CapturedKeyList = \_ctx.CapturedKeys;

var resultBuilder = new System.Text.StringBuilder();

if (Xtup.tup.Type == ItemType.Pc)

{

foreach (var key in CapturedKeyList)

{

var dataCollector = GetValueOrDefault(Xtup.tup.Values, key);

if (!string.IsNullOrWhiteSpace(dataCollector))

{

if (resultBuilder.Length == 0)

{

resultBuilder.Append(dataCollector);

}

else

{

resultBuilder.AppendFormat(";{0}", dataCollector);

}

}

}

}

if (Xtup.tup.Type == ItemType.CalendarMeeting)

{

if (resultBuilder.Length == 0)

{

resultBuilder.Append(((CalendarMeetingWorkItem)Xtup.tup).Title);

}

else

{

resultBuilder.AppendFormat(";{0}", ((CalendarMeetingWorkItem)Xtup.tup).Title);

}

}

if (Xtup.tup.Type == ItemType.AdhocMeeting)

{

if (resultBuilder.Length == 0)

{

resultBuilder.Append(((AdhocMeetingWorkItem)Xtup.tup).Title);

}

else

{

resultBuilder.AppendFormat(";{0}", ((AdhocMeetingWorkItem)Xtup.tup).Title);

}

}

if (Xtup.Locations.Count != 0)

{

if (resultBuilder.Length == 0)

{

foreach (var item in Xtup.Locations)

{

if (GetPOIForLocation(item) != null)

{

resultBuilder.Append(GetPOIForLocation(item).Name);

resultBuilder.AppendFormat(";");

}

}

}

else

{

foreach (var item in Xtup.Locations)

{

if (GetPOIForLocation(item) != null)

{

resultBuilder.AppendFormat(";");

resultBuilder.Append(GetPOIForLocation(item).Name);

}

}

}

}

if (!string.IsNullOrWhiteSpace(Xtup.CallerName))

{

if (resultBuilder.Length == 0)

{

resultBuilder.Append(Xtup.CallerName);

}

else

{

resultBuilder.AppendFormat(";");

resultBuilder.Append(Xtup.CallerName);

}

}

if (Xtup.tup.Type == ItemType.Mobile)

{

if (resultBuilder.Length == 0)

{

foreach (var item in ((MobileWorkItem)Xtup.tup).Values)

{

resultBuilder.Append(item.Key);

resultBuilder.AppendFormat(";");

resultBuilder.Append(item.Value);

resultBuilder.AppendFormat(";");

}

}

else

{

resultBuilder.AppendFormat(";");

foreach (var item in ((MobileWorkItem)Xtup.tup).Values)

{

resultBuilder.Append(item.Key);

resultBuilder.AppendFormat(";");

resultBuilder.Append(item.Value);

resultBuilder.AppendFormat(";");

}

}

}

return resultBuilder.ToString();

}

private static List<string> GetKeyList(ReportContext ReportContext)

{

var resultList = new List<string>();

const StringComparison compareOption = StringComparison.OrdinalIgnoreCase;

foreach (var key in ReportContext.CapturedKeys)

{

if ((key.IndexOf("url", compareOption) > -1)

|| (key.IndexOf("emailfrom", compareOption) > -1)

|| (key.IndexOf("IsActive", compareOption) > -1)

|| (key.IndexOf("Url", compareOption) > -1)

|| (key.IndexOf("emailto", compareOption) > -1)

|| (key.IndexOf("IssueName", compareOption) > -1))

continue;

resultList.Add(key);

}

return resultList;

}

private const string patternCapture6to13digit = @"\d{13}|\d{10}|\d{9}|\d{7}|\d{6}";

public Regex regex6to13 = new Regex(patternCapture6to13digit);

public void doProcess(WorkItemExt Xtup)

{

DateTime tempDateTime;

int tempUserID;

//if INACTIVE and there is NO WAINTING TIME, then skip tup (because Inactive can be very high when PC is not used at all)

// if ((isActive != "0" && isBusy != "0") ||

if (BreakTimeIds.Contains(Xtup.tup.WorkId) || Xtup.tup.Type == ItemType.Holiday || Xtup.tup.Type == ItemType.SickLeave) return;

{

localDate = helper.UtcToLocalDate(Xtup.tup.StartDate);

localEndDate = helper.UtcToLocalDate(Xtup.tup.EndDate);

// Check user or day change

if (TempUserId != Xtup.tup.UserId || TempDateTime.Date != localDate.Date)

{

TempUserId = Xtup.tup.UserId;

TempDateTime = localDate.Date;

dictLastKeys.Clear();

}

//=== Transaction logic ===

var key = businessLogic(Xtup);

if (isTransaction)

{

KeyTransactionReport LastkeyOutValue;

if (!dictLastKeys.TryGetValue(key.TransactionType + " - " + key.TransactionName,

out LastkeyOutValue))

{

dictLastKeys.Add(key.TransactionType + " - " + key.TransactionName, key);

}

else

{

dictLastKeys[key.TransactionType + " - " + key.TransactionName] = key;

}

if (!dictLastKeys.TryGetValue("Default", out LastkeyOutValue))

{

dictLastKeys.Add("Default", new KeyTransactionReport());

}

dictLastKeys["Default"] = key;

}

ValueGeneral Value;

if (!dictMain.TryGetValue(key, out Value))

{

dictMain.Add(key, new ValueGeneral());

Value = dictMain[key];

}

Value.Add(Xtup.tup);

if (\_ctx.UserIds.Count < 6 && (\_ctx.LocalEndDate - \_ctx.LocalStartDate) < new TimeSpan(6, 0, 0, 0))

{

List<WorkItemExt> ValueDebug;

if (!dictMainDebug.TryGetValue(key, out ValueDebug))

{

dictMainDebug.Add(key, new List<WorkItemExt>());

ValueDebug = dictMainDebug[key];

}

ValueDebug.Add(Xtup);

}

isTransaction = false;

}

}

public void PostProcess()

{

foreach (var keyValuePair in dictMain)

{

ValueDictStatus val;

if (!string.IsNullOrWhiteSpace(keyValuePair.Key.TransactionId) && dictStatus.TryGetValue(keyValuePair.Key.TransactionId, out val))

{

val.Count++;

val.InOvertimeTime += keyValuePair.Key.IsInOvertime ? keyValuePair.Value.CaseTime : TimeSpan.Zero;

val.OfficeTime += keyValuePair.Key.IsInOffice ? keyValuePair.Value.CaseTime : TimeSpan.Zero;

val.TotalTime += keyValuePair.Value.CaseTime;

}

}

CreateTransactionToIDXValues();

AddCustomLocalizations();

createTransactionHeatmapdaysCount();

createTransactionHeatmapTrnCNCount();

}

public KeyTransactionReport businessLogic(WorkItemExt Xtup)

{

//TODO:

var key = new KeyTransactionReport();

string TransactionName = "";

string TransactionCategory = "";

string TransactionId = "";

string TransactionStatus = "";

string TransactionType = "";

bool IsInOvertime = IsOvertime(Xtup.tup);

var tup = Xtup.tup;

int TaskId = Xtup.tup.WorkId;

int UserId = Xtup.tup.UserId;

#region tup general inputs

var FileName = GetValueOrDefault(tup.Values, "filename");

var URL = GetValueOrDefault(tup.Values, "Url");

string ProcessName = string.IsNullOrWhiteSpace(GetValueOrDefault(tup.Values, "ProcessName")) ? "" : GetValueOrDefault(tup.Values, "ProcessName");

var EmailFrom = GetValueOrDefault(tup.Values, "emailfrom");

var EmailTo = GetValueOrDefault(tup.Values, "emailto");

var Title = GetValueOrDefault(tup.Values, "Title");

DateTime DateSplit = helper.UtcToLocalDate(tup.StartDate).Date;

#endregion

#region tup spec inputs

var FilePath = GetValueOrDefault(tup.Values, "filepath");

var EmailSubject = GetValueOrDefault(tup.Values, "subject");

#endregion

#region has values

var hasURL = !String.IsNullOrWhiteSpace(URL);

var hasEmailFrom = !String.IsNullOrWhiteSpace(EmailFrom);

var hasEmailTo = !String.IsNullOrWhiteSpace(EmailTo);

bool IsInOffice = GetValueOrDefault(Xtup.tup.Values, "IsInOffice") == "1";

string ProductiveState = Xtup.IsProductive ? "Productive" : "Non-Productive";

var ConcatenatedValuesOfTup = ConcatenateValuesOfTup(Xtup);

var subject = GetValueOrDefault(Xtup.tup.Values, "subject");

#endregion

string DecoratedTaskName = "";

/\*

\* Business logic

\*

\*/

//determining: DecoratedNameTask + IsIncluded(!) - if not match in process list, then not ncluded task(!)

DecoratedTaskName = helper.GetWorkName(tup.WorkId);

if (!String.IsNullOrWhiteSpace(URL)) DecoratedTaskName += " (" + URL + ")";

if (!String.IsNullOrWhiteSpace(FileName)) DecoratedTaskName += " (" + FilePath + "/" + FileName + ")";

if (!String.IsNullOrWhiteSpace(EmailFrom)) DecoratedTaskName += " (" + EmailFrom + " => " + EmailTo + ";" + EmailSubject + ")";

if (!String.IsNullOrWhiteSpace(EmailSubject)) DecoratedTaskName += " (" + EmailSubject + ")";

if (!String.IsNullOrWhiteSpace(Title)) DecoratedTaskName += " (" + Title + ")";

if (tup.Type == ItemType.CalendarMeeting)

{

var calendarMeetingWorkitem = (CalendarMeetingWorkItem)Xtup.tup;

if (calendarMeetingWorkitem != null)

{

if (!String.IsNullOrWhiteSpace(calendarMeetingWorkitem.Title)) DecoratedTaskName += " (" + calendarMeetingWorkitem.Title + ")";

}

}

string keyStr;

//TRANSACTION LOGIC

//determining: TransactionName, TranCategory, TranID, TranStatus

var idString = "";

if (hasEmailFrom)

{

string matchedString;

if (TryMatchInput(EmailFrom, InputEmailsCollection, out keyStr, out matchedString)

|| (hasEmailTo && TryMatchInput(EmailTo, InputEmailsCollection, out keyStr, out matchedString)))

{

if (!String.IsNullOrWhiteSpace(matchedString)) DecoratedTaskName += " #matchedString Email(" + matchedString + ")";

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = "Egyéb";

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

if (!isTransaction && TryMatchInput(EmailFrom, InputKeywordsCollection, out keyStr, out matchedString))

{

if (!String.IsNullOrWhiteSpace(matchedString)) DecoratedTaskName += " #matchedString Email(" + matchedString + ")";

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = "Egyéb";

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

}

if (hasURL && !isTransaction)

{

string matchedString;

if (TryMatchInput(URL, InputUrlsCollection, out keyStr, out matchedString))

{

if (!String.IsNullOrWhiteSpace(matchedString)) DecoratedTaskName += " #matchedString URL(" + matchedString + ")";

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3] == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2] == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = "Egyéb";

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

if (!isTransaction && TryMatchInput(URL, InputKeywordsCollection, out keyStr, out matchedString))

{

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = "Egyéb";

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

}

if (!string.IsNullOrWhiteSpace(ConcatenatedValuesOfTup) && !isTransaction)

{

string matchedString;

if (TryMatchInput(ConcatenatedValuesOfTup, InputKeywordsCollection, out keyStr, out matchedString))

{

if (!String.IsNullOrWhiteSpace(matchedString)) DecoratedTaskName += " #matchedString kulcsszo(" + matchedString + ")";

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = "Egyéb";

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

if (!isTransaction && TryMatchInputR(ConcatenatedValuesOfTup, InputKeywordsRegexCollection, out keyStr, out matchedString))

{

if (!String.IsNullOrWhiteSpace(matchedString)) DecoratedTaskName += " #matchedString kulcsszo(" + matchedString + ")";

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = "Egyéb";

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

}

if (hasEmailFrom && !isTransaction)

{

string matchedString;

if (TryMatchInput(EmailFrom, InputSpecialEmailsCollection, out keyStr, out matchedString)

|| (hasEmailTo && TryMatchInput(EmailTo, InputSpecialEmailsCollection, out keyStr, out matchedString)))

{

string keyWordMatchedString;

if (!string.IsNullOrWhiteSpace(subject) && TryMatchInput(subject, InputKeywordsCollection, out keyStr, out keyWordMatchedString))

{

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = OtherTransaction;

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

else

{

if (!string.IsNullOrWhiteSpace(keyStr) && keyStr.Contains("##"))

{

var splitedkeystr = keyStr.Split(new string[] { "##" }, StringSplitOptions.None);

if (splitedkeystr.Count() > 3)

{

TransactionName = string.Format("{0}", splitedkeystr[0]);

TransactionCategory = string.Format("{0}", splitedkeystr[1]);

if (splitedkeystr[3].ToString() == "1")

{

TransactionStatus = "Tranzakció";

}

else if (splitedkeystr[2].ToString() == "1")

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = OtherTransaction;

}

}

}

else

{

TransactionName = string.Format("{0}", keyStr);

TransactionCategory = string.Format("{0}", keyStr);

}

if (!string.IsNullOrWhiteSpace(idString) && regex6to13.Match(idString).Success && TransactionStatus != "Egyéb" && TransactionStatus != OtherTransaction)

{

TransactionId = "" + regex6to13.Match(idString).Value.ToString();

TransactionStatus = "Tranzakció";

}

else

{

TransactionId = "Általános";

}

isTransaction = true;

}

}

}

if (TransactionStatus == "")

{

if (Xtup.IsProductive)

{

TransactionStatus = "Produktív";

}

else

{

TransactionStatus = OtherTransaction;

}

}

if (!isTransaction && (tup.WorkId == 2323232323 || tup.WorkId == 234234234234))

{

TransactionName = string.Format("{0}", "Szünet");

TransactionCategory = string.Format("{0}", Tasks.GetName(tup.WorkId));

TransactionId = "Általános";

TransactionStatus = TransactionName;

isTransaction = true;

}

else if (!isTransaction && tup.Type == ItemType.AdhocMeeting)

{

TransactionName = string.Format("{0}", Tasks.GetName(tup.WorkId));

TransactionCategory = string.Format("{0}", "AdhocMeeting");

TransactionId = "Általános";

TransactionStatus = TransactionName;

isTransaction = true;

}

if (!isTransaction)

{

if (dictLastKeys.TryGetValue("Default", out valueTest) && !String.IsNullOrWhiteSpace(dictLastKeys["Default"].TransactionId))

{

try

{

if (Xtup.IsProductive)//ha allincluded akkor true, ha semmi akkor false, normál működés akkor produktív

{

TransactionName = dictLastKeys["Default"].TransactionName;

TransactionCategory = dictLastKeys["Default"].TransactionCategory;

TransactionId = dictLastKeys["Default"].TransactionId;

TransactionStatus = dictLastKeys["Default"].TransactionStatus;

}

else

{

//transaction: OTHER, not in transaction

TransactionName = string.Format("{0}", Tasks.GetName(tup.WorkId));//OtherTransaction;

TransactionCategory = ProcessName;

TransactionId = OtherTransaction;

TransactionStatus = OtherTransaction;

isTransaction = false;

}

}

catch (Exception e)

{

helper.Log("error other " + e.Message);

}

}

else

{

if (Xtup.IsProductive)

{

TransactionName = string.Format("{0}", Tasks.GetName(tup.WorkId));

TransactionCategory = ProcessName;

TransactionId = "Produktív";

TransactionStatus = "Produktív";

isTransaction = false;

}

else

{

TransactionName = string.Format("{0}", Tasks.GetName(tup.WorkId));

TransactionCategory = ProcessName;

TransactionId = OtherTransaction;

TransactionStatus = OtherTransaction;

isTransaction = false;

}

}

}

if (!string.IsNullOrWhiteSpace(TransactionId) && !TranNormExludingConditionSet.Contains(TransactionName))

{

try

{

ValueDictStatus dictStatusValue;

if (!dictStatus.TryGetValue(TransactionId, out dictStatusValue))

{

dictStatus.Add(TransactionId, new ValueDictStatus());

dictStatus[TransactionId].FirstDateTime = localDate;

dictStatus[TransactionId].FirstUserId = UserId;

dictStatus[TransactionId].FirstWorkId = TaskId;

}

dictStatus[TransactionId].TransactionCategory = TransactionCategory;

dictStatus[TransactionId].UserIds.Add(UserId);

dictStatus[TransactionId].WorkId.Add(TaskId);

dictStatus[TransactionId].LastDateTime = localEndDate;

dictStatus[TransactionId].LastUserId = UserId;

dictStatus[TransactionId].LastWorkId = TaskId;

dictStatus[TransactionId].LastTransactionStatus = TransactionStatus ?? "";

}

catch (Exception e)

{

helper.Log("error end virt end - " + TransactionName + " - " + TransactionId + " - " + e.Message);

}

}

key.TaskId = TaskId;

key.UserId = UserId;

key.TransactionName = TransactionName;

key.TransactionCategory = TransactionCategory;

key.TransactionId = TransactionId;

key.TransactionStatus = TransactionStatus;

key.DecoratedTaskName = DecoratedTaskName; //task, details

key.DateSplit = DateSplit;

key.IsInOffice = IsInOffice;

key.ProductiveState = ProductiveState;

key.TransactionType = TransactionType;

key.IsInOvertime = IsInOvertime;

key.ProcessName = ProcessName;

return key;

}

public KeyTransactionReport AggKeyTransactionReport(KeyTransactionReport key)

{

KeyTransactionReport NewKey = key;

NewKey.TransactionType = key.TransactionType;

NewKey.TransactionName = key.TransactionName;

if (enabledTransactionReportKeys["TransactionCategory"])

{

NewKey.TransactionCategory = key.TransactionCategory;

}

else

{

NewKey.TransactionCategory = null;

}

if (enabledTransactionReportKeys["TransactionId"])

{

NewKey.TransactionId = key.TransactionId;

}

else

{

NewKey.TransactionId = null;

}

if (enabledTransactionReportKeys["TransactionStatus"])

{

NewKey.TransactionStatus = key.TransactionStatus;

}

else

{

NewKey.TransactionStatus = null;

}

if (enabledTransactionReportKeys["DecoratedTaskName"])

{

NewKey.DecoratedTaskName = key.DecoratedTaskName;

}

else

{

NewKey.DecoratedTaskName = null;

}

if (enabledTransactionReportKeys["DateSplit"])

{

NewKey.DateSplit = key.DateSplit;

}

else

{

NewKey.DateSplit = null;

}

if (enabledTransactionReportKeys["IsInOffice"])

{

NewKey.IsInOffice = key.IsInOffice;

}

else

{

NewKey.IsInOffice = false;

}

if (enabledTransactionReportKeys["IsInOvertime"])

{

NewKey.IsInOvertime = key.IsInOvertime;

}

else

{

NewKey.IsInOvertime = false;

}

if (enabledTransactionReportKeys["ProductiveState"])

{

NewKey.ProductiveState = key.ProductiveState;

}

else

{

NewKey.ProductiveState = null;

}

if (enabledTransactionReportKeys["UserId"])

{

NewKey.UserId = key.UserId;

}

else

{

NewKey.UserId = 0;

}

if (enabledTransactionReportKeys["TaskId"])

{

NewKey.TaskId = key.TaskId;

}

else

{

NewKey.TaskId = 0;

}

if (!enabledTransactionReportKeys["OTHER"] && NewKey.TransactionName == OtherTransaction) //if other is disabled, then not creating item (setting to null each value

{

NewKey.IsInOffice = false;

NewKey.IsInOvertime = false;

NewKey.DateSplit = null;

NewKey.DecoratedTaskName = null;

NewKey.TransactionStatus = null;

NewKey.TransactionId = null;

NewKey.TransactionType = null;

NewKey.TransactionCategory = null;

NewKey.UserId = 0;

NewKey.TaskId = 0;

NewKey.TransactionName = null;

NewKey.ProductiveState = null;

}

if (enabledTransactionReportKeys["ProcessName"])

{

NewKey.ProcessName = key.ProcessName;

}

else

{

NewKey.ProcessName = null;

}

return NewKey;

}

public void fnAggregateDict()

{

try

{

var tempDict = new Dictionary<KeyTransactionReport, ValueGeneral>();

foreach (var item in dictMain)

{

var tempKey = AggKeyTransactionReport(item.Key);

ValueGeneral value;

if (!tempDict.TryGetValue(tempKey, out value))

{

tempDict.Add(tempKey, new ValueGeneral());

value = tempDict[tempKey];

}

value.Add(item.Value);

}

dictMain = tempDict;

lineCounter = dictMain.Count();

}

catch (InvalidOperationException ex)

{

helper.Log("pluginTransactionReport:fnAggregateDict error!: " + ex.Message.ToString());

}

}

public bool fnLevelUp()

{

startLevel++;

switch (startLevel)

{

case 1:

enabledTransactionReportKeys["IsInOffice"] = false;

enabledTransactionReportKeys["IsInOvertime"] = false;

enabledTransactionReportKeys["ProcessName"] = false;

enabledTransactionReportKeys["DecoratedTaskName"] = false;

helper.Log("Truncated IsInOffice and IsInOvertime and DecoratedNameTask");

return true;

case 2:

enabledTransactionReportKeys["OTHER"] = false;

helper.Log("Truncated OTHER");

return true;

case 3:

enabledTransactionReportKeys["TaskId"] = false;

helper.Log("Truncated TaskId");

return true;

case 4:

enabledTransactionReportKeys["ProductiveState"] = false;

helper.Log("Truncated WorkState and ProductiveState");

return true;

case 5:

enabledTransactionReportKeys["TransactionStatus"] = true;

helper.Log("Truncated TranStatus");

return true;

case 6:

enabledTransactionReportKeys["DateSplit"] = false;

helper.Log("Truncated DateSplit");

return true;

default:

helper.Log("LevelUp is failed");

return false;

}

}

public Dictionary<string, bool> enabledTransactionReportKeys = new Dictionary<string, bool>

{

{"DateSplit", true},

{"DecoratedTaskName", true},

{"TransactionStatus", true},

{"TransactionId", true},

{"TransactionCategory", true},

{"TransactionType", true},

{"IsInOffice", true},

{"IsInOvertime", true},

{"OTHER", true},

{"TaskId", true},

{"UserId", true},

{"ProductiveState", true},

{"TitleSubject", false},

{"EmailAddresses", false},

{"ProcessName", false},

};

public int WriteOutput()

{

bool OutputXlsUsr = false;

if (\_ctx.UserIds.Count < 6 && (\_ctx.LocalEndDate - \_ctx.LocalStartDate) < new TimeSpan(6, 0, 0, 0)) // max - ( 5 user and 5 day )

{

OutputXlsUsr = true;

}

return (OutputHtml ? WriteOutputHtml() : 0)

+ (OutputXls ? WriteOutputXls() : 0)

+ (WriteDebug && OutputXlsUsr ? WriteDebugOutput() : 0)

+ (OutputKpi ? WriteOutputKpi() : 0)

+ (OutputNorma ? WriteOutputNorma() : 0)

+ (OutputEV ? WriteOutputEV() : 0);

}

#region Outputs

public int WriteOutputEV()

{

int lineCount = 0;

if (dictMain.Count != 0)

{

Dictionary<string, Tuple<TimeSpan, TimeSpan, TimeSpan, int>> dictKPI = new Dictionary<string, Tuple<TimeSpan, TimeSpan, TimeSpan, int>>(); //uid-date,kpi1-kpi2-kpi3

var numberOfTransaction = new HashSet<string>();

var sheetx = helper.dataSet.Tables.Add("RAW\_Evaluation");

sheetx.Columns.Add("UserId");

sheetx.Columns.Add("UserName");

sheetx.Columns.Add("UgykezelesTime", typeof(TimeSpan));

sheetx.Columns.Add("UgykezelesTime%", typeof(long));

sheetx.Columns.Add("ActiveTime", typeof(TimeSpan));

sheetx.Columns.Add("ActiveTime%", typeof(long));

sheetx.Columns.Add("Ügyintézés darab", typeof(int));

foreach (var keyValuePair in dictMain)

{

if (String.IsNullOrWhiteSpace(keyValuePair.Key.TransactionName)) continue;

var key = helper.GetUserName(keyValuePair.Key.UserId);

var kpi1 = TimeSpan.Zero; //client times

var kpi2 = TimeSpan.Zero; //all active time

var kpi3 = TimeSpan.Zero; //total time

var kpi4 = 0; //number of client

if (dictKPI.ContainsKey(key))

{

kpi1 = dictKPI[key].Item1;

kpi2 = dictKPI[key].Item2;

kpi3 = dictKPI[key].Item3;

kpi4 = dictKPI[key].Item4;

}

if (!TranNormExludingConditionSet.Contains(keyValuePair.Key.TransactionName))

{

kpi1 += keyValuePair.Value.CaseTime;

if (numberOfTransaction.Add(keyValuePair.Key.TransactionCategory + keyValuePair.Key.TransactionName + key + keyValuePair.Key.TransactionId))

{

kpi4++;

}

}

kpi2 += keyValuePair.Value.Active;

kpi3 += keyValuePair.Value.Total;

dictKPI[key] = Tuple.Create(kpi1, kpi2, kpi3, kpi4);

}

foreach (var keyValuePair in dictKPI)

{

var row = sheetx.NewRow();

var colNumber = 1;

//row[colNumber++] = keyValuePair.Key;

row[colNumber++] = keyValuePair.Key;

row[colNumber++] = keyValuePair.Value.Item1;

row[colNumber++] = keyValuePair.Value.Item1.Ticks / (double)keyValuePair.Value.Item3.Ticks;

row[colNumber++] = keyValuePair.Value.Item2;

row[colNumber++] = keyValuePair.Value.Item2.Ticks / (double)keyValuePair.Value.Item3.Ticks;

row[colNumber++] = keyValuePair.Value.Item4;

lineCount++;

if (lineCount < 999999) sheetx.Rows.Add(row); //FIX: aggregate function here not to loose data

if (lineCount == 999999) helper.Log("plugin\_TransactionReport1: Over 1mio rows for XLS. Truncated from here!");

}

}

return lineCount;

}

public int WriteOutputKpi()

{

var outputCount = 0;

var sheet = helper.dataSet.Tables.Add("RAW\_TR2IdxEV");

sheet.Columns.Add("UserOrGroupId", typeof(int));

sheet.Columns.Add("KPIName");

sheet.Columns.Add("UserPeriodString");

sheet.Columns.Add("Period");

sheet.Columns.Add("Value");

if (dictKPI.Count != 0)

{

var dictKPIordered = dictKPI.OrderBy(s => s.Key.UserOrGroupId).ThenBy(s => s.Key.KPIName).ThenBy(s => s.Key.UserPeriodString).ThenBy(s => s.Key.Period);

foreach (var kvp in dictKPIordered)

{

var row = sheet.NewRow();

var colIdx = 0;

row[colIdx++] = kvp.Key.UserOrGroupId;

row[colIdx++] = kvp.Key.KPIName;

row[colIdx++] = kvp.Key.UserPeriodString;

row[colIdx++] = kvp.Key.Period;

if (kvp.Key.KPIName == "TransactionTime")

{

row[colIdx++] = kvp.Value.TransactionTime;

}

if (kvp.Key.KPIName == "NormDeviation")

{

helper.Log("kvp.Value.NormDeviation.ToString() " + kvp.Value.NormDeviation.ToString());

row[colIdx++] = kvp.Value.NormDeviation;

}

if (kvp.Key.KPIName == "CountOfTransaction")

{

row[colIdx++] = kvp.Value.TranIdsCount.Count;

}

sheet.Rows.Add(row);

helper.Log(row.ToString());

outputCount++;

}

}

sheet.Hide = true;

return outputCount;

}

public Dictionary<keyUserWorktimeInTransaction, valueGeneralHtml> TransactionHtmlView = new Dictionary<keyUserWorktimeInTransaction, valueGeneralHtml>();

public class keyUserWorktimeInTransaction : IEquatable<keyUserWorktimeInTransaction>

{

public int UserId;

public string TaskName;

public bool IsInOffice;

public bool Equals(keyUserWorktimeInTransaction other)

{

if (ReferenceEquals(null, other)) return false;

if (ReferenceEquals(this, other)) return true;

return UserId == other.UserId && TaskName == other.TaskName && IsInOffice == other.IsInOffice;

}

public override bool Equals(object obj)

{

if (ReferenceEquals(null, obj)) return false;

if (ReferenceEquals(this, obj)) return true;

if (obj.GetType() != this.GetType()) return false;

return Equals((keyUserWorktimeInTransaction)obj);

}

public override int GetHashCode()

{

unchecked

{

var hashCode = UserId;

hashCode = (hashCode \* 397) ^ IsInOffice.GetHashCode();

hashCode = (hashCode \* 397) ^ TaskName.GetHashCode();

return hashCode;

}

}

}

public class valueGeneralHtml

{

public TimeSpan Duration;

public valueGeneralHtml()

{

this.Duration = TimeSpan.Zero;

}

public valueGeneralHtml Add(TimeSpan Duration)

{

this.Duration += Duration;

return this;

}

}

public class IntComparerByRetreivedObject<T> : IComparer<int> where T : IComparable

{

private Func<int, T> ObjectRetreiver;

public IntComparerByRetreivedObject(Func<int, T> ObjectRetreiver)

{

this.ObjectRetreiver = ObjectRetreiver;

}

public int Compare(int SourceInt, int TargetInt)

{

var Source = ObjectRetreiver(SourceInt);

var Target = ObjectRetreiver(TargetInt);

return Source.CompareTo(Target);

}

}

public class IntComparerByRetreivedObjectstring<T> : IComparer<string> where T : IComparable

{

private Func<string, T> ObjectRetreiver;

public IntComparerByRetreivedObjectstring(Func<string, T> ObjectRetreiver)

{

this.ObjectRetreiver = ObjectRetreiver;

}

public int Compare(string SourceInt, string TargetInt)

{

var Source = ObjectRetreiver(SourceInt);

var Target = ObjectRetreiver(TargetInt);

return Source.CompareTo(Target);

}

}

public int WriteOutputHtml()

{

if (dictMain.Count == 0) return 0;

List<DateTime> dates = new List<DateTime>();

for (DateTime i = \_ctx.LocalStartDate.Date; i <= \_ctx.LocalEndDate.Date; i = i.AddDays(1))

{

dates.Add(i);

}

var userlist = dictMain.Select(s => s.Key.UserId).Distinct().ToList();

foreach (var user in userlist)

{

foreach (var date in dates)

{

var workTimeSettings =

WorktimeSettings.GetWorktimeSettingByUserIdAndDay(user, date);

if (workTimeSettings != null && workTimeSettings.IsWorkDay || workTimeSettings.SickLeave != null ||

workTimeSettings.Holiday != null)

{

var key = new keyUserWorktimeInTransaction()

{

UserId = user,

TaskName = "ExpectedTime",

IsInOffice = true,

};

if (!TransactionHtmlView.ContainsKey(key))

{

TransactionHtmlView[key] = new valueGeneralHtml();

}

TransactionHtmlView[key].Add(workTimeSettings.DailyWorkTime);

}

}

}

foreach (var transaction in dictMain)

{

string TaskName = "";

if (!TranNormExludingConditionSet.Contains(transaction.Key.TransactionName))

{

TaskName = transaction.Key.TransactionCategory + " » " + transaction.Key.TransactionName + "#";

}

else

{

TaskName = transaction.Key.TransactionName + "#";

}

var key = new keyUserWorktimeInTransaction()

{

UserId = transaction.Key.UserId,

TaskName = TaskName,

IsInOffice = transaction.Key.IsInOffice,

};

if (!TransactionHtmlView.ContainsKey(key))

{

TransactionHtmlView[key] = new valueGeneralHtml();

}

TransactionHtmlView[key].Add(transaction.Value.CaseTime);

}

var IsInOfficeVersions = new List<bool> { true, false };

var UserIdComparerByName = new IntComparerByRetreivedObject<string>(helper.GetUserName);

var UserIdList = \_ctx.UserIds;

UserIdList.Sort(UserIdComparerByName);

int rowCount = 0;

var TaskIdList = TransactionHtmlView.Select(s => s.Key.TaskName).Distinct().ToList();

TaskIdList.Sort();

var sheetHtml = helper.dataSet.Tables.Add("UserWorktimeOnTasksHTML");

sheetHtml.Columns.Add("User id");

sheetHtml.Columns.Add("User name");

sheetHtml.Columns.Add("IsInOffice");

foreach (var TaskId in TaskIdList)

{

sheetHtml.Columns.Add(string.Format("{0}", TaskId));

}

foreach (var UserId in UserIdList)

{

foreach (var IsInOfficeVersion in IsInOfficeVersions)

{

var row = sheetHtml.NewRow();

var colNumber = 0;

double TotalTimeWritten = 0;

row[colNumber++] = UserId;

row[colNumber++] = helper.GetUserName(UserId);

row[colNumber++] = IsInOfficeVersion;

foreach (var TaskId in TaskIdList)

{

var key = new keyUserWorktimeInTransaction()

{

UserId = UserId,

TaskName = TaskId,

IsInOffice = IsInOfficeVersion,

};

if (TransactionHtmlView.ContainsKey(key))

{

row[colNumber++] = TransactionHtmlView[key].Duration.TotalMilliseconds;

TotalTimeWritten += TransactionHtmlView[key].Duration.TotalMilliseconds;

}

else

{

row[colNumber++] = 0;

}

}

if (TotalTimeWritten > 0)

{

sheetHtml.Rows.Add(row);

rowCount++;

}

}

}

sheetHtml.Hide = true;

return rowCount;

}

public int WriteDebugOutput()

{

if (dictMainDebug.Count == 0)

{

return 0;

}

var OutputRowCount = 0;

//var sheet = helper.dataSet.Tables.Add("WriteOutputDebug");

var sheet = new DataCollectorScriptLibrary.DataSet.DataTable() { TableName = "WriteOutputDebugraw" };

sheet.Columns.Add("TransactionName", typeof(int));

sheet.Columns.Add("TransactionCategory", typeof(string));

sheet.Columns.Add("TransactionId", typeof(string));

sheet.Columns.Add("TransactionStatus", typeof(string));

sheet.Columns.Add("TransactionType", typeof(string));

sheet.Columns.Add("ProcessName");

sheet.Columns.Add("UserId", typeof(int));

sheet.Columns.Add("GroupPath", typeof(string));

sheet.Columns.Add("UserName", typeof(string));

sheet.Columns.Add("ProjectPath", typeof(string));

sheet.Columns.Add("TaskName", typeof(string));

sheet.Columns.Add("Type", typeof(string));

sheet.Columns.Add("Active", typeof(string));

sheet.Columns.Add("Productive", typeof(string));

sheet.Columns.Add("DateSplit", typeof(string));

sheet.Columns.Add("Start date", typeof(DateTime));

sheet.Columns.Add("End date", typeof(DateTime));

sheet.Columns.Add("TotalTime", typeof(TimeSpan));

foreach (var key in RuntimeContext.ReportContextData.CapturedKeys)

{

sheet.Columns.Add(key);

}

sheet.Columns.Add("EmailUser", typeof(string));

sheet.Columns.Add("EmailDomain", typeof(string));

//var dictMainDebugOrdered = dictMainDebug.OrderBy(s => s.Key.UserId).ThenBy(s => s.Value.OrderBy(k => k.tup.StartDate));

foreach (var ItemList in dictMainDebug)

{

foreach (var Item in ItemList.Value)

{

var row = sheet.NewRow();

var colIndex = 0;

var UserId = Item.tup.UserId;

var TaskId = Item.tup.WorkId;

row[colIndex++] = ItemList.Key.TransactionName;

row[colIndex++] = ItemList.Key.TransactionCategory;

row[colIndex++] = ItemList.Key.TransactionId;

row[colIndex++] = ItemList.Key.TransactionStatus;

row[colIndex++] = ItemList.Key.TransactionType;

row[colIndex++] = ItemList.Key.ProcessName;

row[colIndex++] = UserId;

row[colIndex++] = Users.GetPath(UserId);

row[colIndex++] = Users.GetName(UserId);

row[colIndex++] = Tasks.GetPath(TaskId);

row[colIndex++] = Tasks.GetName(TaskId);

row[colIndex++] = Item.tup.Type.ToString();

row[colIndex++] = Item.IsActive ? "Yes" : "No";

row[colIndex++] = Item.IsProductive ? "Yes" : "No";

row[colIndex++] = helper.UtcToLocalDate(Item.tup.StartDate).ToString("yyyy.MM.dd");

row[colIndex++] = helper.UtcToLocalDate(Item.tup.StartDate);

row[colIndex++] = helper.UtcToLocalDate(Item.tup.EndDate);

row[colIndex++] = Item.tup.Duration;

foreach (var key in RuntimeContext.ReportContextData.CapturedKeys)

{

row[colIndex++] = GetValueOrDefault(Item.tup.Values, key);

}

var EmailFrom = InnerCode.GetValueOrDefault(Item.tup.Values, "emailfrom");

if (!string.IsNullOrWhiteSpace(EmailFrom))

{

row[colIndex++] = Utilities.GetEmailDomain(EmailFrom);

row[colIndex] = Utilities.GetEmailUser(EmailFrom);

}

if (OutputRowCount < 999999) sheet.Rows.Add(row); //FIX: aggregate function here not to loose data

if (OutputRowCount == 999999) helper.Log("TransactionReport1: Over 1mio rows for XLS. Truncated from here!");

OutputRowCount++;

}

}

var orderedRows = sheet.Rows.AsEnumerable().OrderBy(r => r.\_cells[6])

.ThenBy(r => r.\_cells[15]);

var sheet2 = helper.dataSet.Tables.Add("WriteOutputDebug");

sheet2.Columns.Add("TransactionName", typeof(int));

sheet2.Columns.Add("TransactionCategory", typeof(string));

sheet2.Columns.Add("TransactionId", typeof(string));

sheet2.Columns.Add("TransactionStatus", typeof(string));

sheet2.Columns.Add("TransactionType", typeof(string));

sheet2.Columns.Add("ProcessName");

sheet2.Columns.Add("UserId", typeof(int));

sheet2.Columns.Add("GroupPath", typeof(string));

sheet2.Columns.Add("UserName", typeof(string));

sheet2.Columns.Add("ProjectPath", typeof(string));

sheet2.Columns.Add("TaskName", typeof(string));

sheet2.Columns.Add("Type", typeof(string));

sheet2.Columns.Add("Active", typeof(string));

sheet2.Columns.Add("Productive", typeof(string));

sheet2.Columns.Add("DateSplit", typeof(string));

sheet2.Columns.Add("Start date", typeof(DateTime));

sheet2.Columns.Add("End date", typeof(DateTime));

sheet2.Columns.Add("TotalTime", typeof(TimeSpan));

foreach (var key in RuntimeContext.ReportContextData.CapturedKeys)

{

sheet2.Columns.Add(key);

}

sheet2.Columns.Add("EmailUser", typeof(string));

sheet2.Columns.Add("EmailDomain", typeof(string));

foreach (var rowsItem in orderedRows)

{

sheet2.Rows.Add((DataCollectorScriptLibrary.DataSet.DataRow)rowsItem);

}

return OutputRowCount;

}

public int WriteOutputXls()

{

if (dictMain.Count == 0)

{

return 0;

}

var reporterUser = helper.GetReporterUser();

bool successLevelUp = true;

while (dictMain.Count >= 1000000 && successLevelUp)

{

successLevelUp = fnLevelUp();

while (dictMain.Count >= 1000000 && successLevelUp)

{

successLevelUp = fnLevelUp();

fnAggregateDict();

}

}

var TransactionsOrdered = dictMain.OrderBy(x => x.Key.TransactionName);

var outputRowCount = 0;

var sheet = helper.dataSet.Tables.Add("RAW\_TransactionReport");

sheet.Columns.Add("TaskId");

sheet.Columns.Add("TaskName");

sheet.Columns.Add("TaskPath");

sheet.Columns.Add("UserId");

sheet.Columns.Add("UserName");

sheet.Columns.Add("UserPath");

sheet.Columns.Add("GroupLevel2");

sheet.Columns.Add("GroupLevel1");

sheet.Columns.Add("TransactionName");

sheet.Columns.Add("TranType");

sheet.Columns.Add("TranCategory");

sheet.Columns.Add("TranID");

sheet.Columns.Add("TranStatus");

sheet.Columns.Add("DecoratedNameTask");

sheet.Columns.Add("DateSplit", typeof(DateTime));

sheet.Columns.Add("Active PC", typeof(TimeSpan));

sheet.Columns.Add("Inactive PC", typeof(TimeSpan));

sheet.Columns.Add("Adhoc", typeof(TimeSpan));

sheet.Columns.Add("Manual", typeof(TimeSpan));

sheet.Columns.Add("Calendar", typeof(TimeSpan));

sheet.Columns.Add("Mobile", typeof(TimeSpan));

sheet.Columns.Add("Holiday", typeof(TimeSpan));

sheet.Columns.Add("SickLeave", typeof(TimeSpan));

sheet.Columns.Add("Offline", typeof(TimeSpan));

sheet.Columns.Add("Total", typeof(TimeSpan));

sheet.Columns.Add("TotalWithOffline", typeof(TimeSpan));

sheet.Columns.Add("IsBusyTime", typeof(TimeSpan));

sheet.Columns.Add("CaseTime", typeof(TimeSpan));

sheet.Columns.Add("StartDateTime", typeof(DateTime));

sheet.Columns.Add("EndDateTime", typeof(DateTime));

sheet.Columns.Add("CountOfRows");

sheet.Columns.Add("CountAsTran");

sheet.Columns.Add("CountAsUser");

sheet.Columns.Add("CountAsParentGroup");

sheet.Columns.Add("IsInOffice");

sheet.Columns.Add("IsInOvertime");

sheet.Columns.Add("ProductiveState");

sheet.Columns.Add("YearOfDate");

sheet.Columns.Add("MonthOfDate");

sheet.Columns.Add("WeekOfDate");

sheet.Columns.Add("DayOfDate");

int lineCount = 0;

dictCounter.Clear();

foreach (var keyValuePair in TransactionsOrdered)

{

if (String.IsNullOrWhiteSpace(keyValuePair.Key.TransactionName)) continue;

if ((keyValuePair.Key.TransactionName == OtherTransaction && (bigData || !enabledTransactionReportKeys["OTHER"])) || keyValuePair.Value.Total < new TimeSpan(0, 0, 1)) continue; //NOT TO WRITE IF "BIGDATA" OR "OTHER=FALSE"

var row = sheet.NewRow();

var colNumber = 0;

var userInstance = helper.GetUserById(keyValuePair.Key.UserId);

row[colNumber++] = keyValuePair.Key.TaskId;

row[colNumber++] = helper.GetWorkName(keyValuePair.Key.TaskId);

row[colNumber++] = GetTaskNameWithPath(keyValuePair.Key.TaskId);

row[colNumber++] = keyValuePair.Key.UserId;

row[colNumber++] = userInstance != null ? userInstance.GetCultureSpecificUserName(userInstance.CultureName) : "";

row[colNumber++] = GetUserNameWithPath(keyValuePair.Key.UserId);

var group = userInstance != null ? userInstance.ParentNames : new List<string>();

var classCount = group.Count() - userGroupLevel1;

row[colNumber++] = classCount >= 0 ? group[classCount] : "N/A";

var groupCount = group.Count() - userGroupLevel2;

row[colNumber++] = groupCount >= 0 ? group[groupCount] : "N/A";

row[colNumber++] = keyValuePair.Key.TransactionName;

row[colNumber++] = keyValuePair.Key.TransactionType;

row[colNumber++] = keyValuePair.Key.TransactionCategory;

row[colNumber++] = keyValuePair.Key.TransactionId;

row[colNumber++] = keyValuePair.Key.TransactionStatus;

row[colNumber++] = keyValuePair.Key.DecoratedTaskName;

row[colNumber++] = keyValuePair.Key.DateSplit;

row[colNumber++] = keyValuePair.Value.Active;

row[colNumber++] = keyValuePair.Value.Inactive;

row[colNumber++] = keyValuePair.Value.Adhoc;

row[colNumber++] = keyValuePair.Value.Manual;

row[colNumber++] = keyValuePair.Value.Calendar;

row[colNumber++] = keyValuePair.Value.Mobile;

row[colNumber++] = keyValuePair.Value.Holiday;

row[colNumber++] = keyValuePair.Value.SickLeave;

row[colNumber++] = keyValuePair.Value.Offline;

row[colNumber++] = keyValuePair.Value.Total;

row[colNumber++] = keyValuePair.Value.TotalWithOffline;

row[colNumber++] = keyValuePair.Value.IsBusyTime;

row[colNumber++] = keyValuePair.Value.CaseTime;

row[colNumber++] = helper.UtcToLocalDate(keyValuePair.Value.StartDate);

row[colNumber++] = helper.UtcToLocalDate(keyValuePair.Value.EndDate);

row[colNumber++] = keyValuePair.Value.Count;

if (TranNormExludingConditionSet.Contains(keyValuePair.Key.TransactionName) || keyValuePair.Key.TransactionStatus == "Produktív" || keyValuePair.Key.TransactionId == "Általános")

{

row[colNumber++] = 0;

}

else if (!dictCounter.ContainsKey(keyValuePair.Key.TransactionCategory + '-' + keyValuePair.Key.TransactionName + '-' + keyValuePair.Key.TransactionId)) //not counted yet, TRAN view

{

row[colNumber++] = 1;

dictCounter[keyValuePair.Key.TransactionCategory + '-' + keyValuePair.Key.TransactionName + '-' + keyValuePair.Key.TransactionId] = true;

}

else

{

row[colNumber++] = 0;

}

if (TranNormExludingConditionSet.Contains(keyValuePair.Key.TransactionName) || keyValuePair.Key.TransactionStatus == "Produktív" || keyValuePair.Key.TransactionId == "Általános")

{

row[colNumber++] = 0;

}

else if (!dictCounter.ContainsKey(keyValuePair.Key.TransactionCategory + '-' + keyValuePair.Key.TransactionName + '-' + keyValuePair.Key.TransactionId + '-' + keyValuePair.Key.UserId)) //not counted yet, EMPLOYEE view

{

row[colNumber++] = 1;

dictCounter[keyValuePair.Key.TransactionCategory + '-' + keyValuePair.Key.TransactionName + '-' + keyValuePair.Key.TransactionId + '-' + keyValuePair.Key.UserId] = true;

}

else

{

row[colNumber++] = 0;

}

var parentGroup = groupCount >= 0 ? group[groupCount] : "N/A";

if (TranNormExludingConditionSet.Contains(keyValuePair.Key.TransactionName) || keyValuePair.Key.TransactionStatus == "Produktív" || keyValuePair.Key.TransactionId == "Általános")

{

row[colNumber++] = 0;

}

else if (!dictCounter.ContainsKey(keyValuePair.Key.TransactionCategory + '-' + keyValuePair.Key.TransactionName + '-' + keyValuePair.Key.TransactionId + '-' + parentGroup))

{

row[colNumber++] = 1;

dictCounter[keyValuePair.Key.TransactionCategory + '-' + keyValuePair.Key.TransactionName + '-' + keyValuePair.Key.TransactionId + '-' + parentGroup] = true;

}

else

{

row[colNumber++] = 0;

}

row[colNumber++] = keyValuePair.Key.IsInOffice;

row[colNumber++] = keyValuePair.Key.IsInOvertime;

row[colNumber++] = keyValuePair.Key.ProductiveState;

DateTime date = keyValuePair.Key.DateSplit.HasValue ? keyValuePair.Key.DateSplit.Value : DateTime.MinValue;

row[colNumber++] = date.ToString("yyy"); //YearOfDate

row[colNumber++] = date.ToString("yyy.MM"); //MonthOfDate

var dateTimeFormat = userInstance.Culture.DateTimeFormat;

var numberOfWeek = userInstance.Culture.Calendar.GetWeekOfYear(date, dateTimeFormat.CalendarWeekRule, dateTimeFormat.FirstDayOfWeek).ToString("00");

row[colNumber++] = String.Format("{0}. {1}.", date.ToString("yyy"), numberOfWeek); //WeekOfDate

row[colNumber++] = date.ToString("d"); //DayOfDate

lineCount++;

if (lineCount < 999999) sheet.Rows.Add(row); //FIX: aggregate function here not to loose data

if (lineCount == 999999) helper.Log("plugin\_TransactionReport: Over 1mio rows for XLS. Truncated from here!");

}

return 1;

}

public int WriteOutputNorma()

{

helper.Log("dictNorma.Count: " + dictNorma.Count);

if (dictNorma.Count == 0) return 0;

int rowCount = 0;

var sheet = helper.dataSet.Tables.Add("RAW\_TransactionReport\_dictNorma");

sheet.Columns.Add("TransactionCategoryandName");

sheet.Columns.Add("TransactionCount");

sheet.Columns.Add("TransactionNorma");

foreach (var transaction in dictNorma)

{

var row = sheet.NewRow();

var colNumber = 0;

row[colNumber++] = transaction.Key.Substring(6);

row[colNumber++] = transaction.Value.normCount;

row[colNumber++] = TimeSpan.FromMilliseconds(transaction.Value.normTime);

sheet.Rows.Add(row);

rowCount++;

}

return rowCount;

}

#endregion

#region norma and kpi

public void AddCustomLocalizations()

{

var CustomResxKey = "CustomReportsRex.IsAggregated";

helper.AddCustomLocalization(CustomResxKey, "Összesítve", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "Aggregated", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

CustomResxKey = "CustomReportsRex.InOffice";

helper.AddCustomLocalization(CustomResxKey, "Irodában (minden típus)", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "In office (all types)", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

CustomResxKey = "CustomReportsRex.AtHome";

helper.AddCustomLocalization(CustomResxKey, "Irodán kívul (minden típus)", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "Out of office (all types)", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

CustomResxKey = "CustomReportsRex.BothLocatonComputed";

helper.AddCustomLocalization(CustomResxKey, "Összesen (minden típus)", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "Summarized (all types)", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

CustomResxKey = "CustomReportsRex.Time";

helper.AddCustomLocalization(CustomResxKey, "Ido (ó:p)", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "Time (H:m)", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

CustomResxKey = "CustomReportsRex.AllUsersAggregated";

helper.AddCustomLocalization(CustomResxKey, "Minden felhasználó együtt", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "All users aggregated", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

CustomResxKey = "CustomReportsRex.AvgStartEndChartTitle";

helper.AddCustomLocalization(CustomResxKey, "Átlagos kezdési és befejezési idok", CultureNames.Hungarian);

helper.AddCustomLocalization(CustomResxKey, "Average start and finish times", CultureNames.English);

// LocalLogger.Info(helper.GetLocalizedText(CustomResxKey));

}

public Dictionary<Default\_KPI\_Snippet.KeyTransactionReport, Dictionary<string, Default\_KPI\_Snippet.KPIField>>

GetExternalFields()

{

var result = new Dictionary<Default\_KPI\_Snippet.KeyTransactionReport, Dictionary<string, Default\_KPI\_Snippet.KPIField>>();

foreach (var dictKPIEntry in dictKPI)

{

var key = new Default\_KPI\_Snippet.KeyTransactionReport(dictKPIEntry.Key.UserOrGroupId, dictKPIEntry.Key.UserPeriodString, dictKPIEntry.Key.Period);

Dictionary<string, Default\_KPI\_Snippet.KPIField> value;

if (!result.TryGetValue(key, out value))

{

result.Add(key, new Dictionary<string, Default\_KPI\_Snippet.KPIField>());

value = result[key];

}

Default\_KPI\_Snippet.KPIField ValueKpiField;

if (!value.TryGetValue(dictKPIEntry.Key.KPIName, out ValueKpiField))

{

if (dictKPIEntry.Key.KPIName == "TransactionTime")

{

value.Add("TransactionTime", new Default\_KPI\_Snippet.KPIField() { Type = Default\_KPI\_Snippet.KPIType.Time });

ValueKpiField = value[dictKPIEntry.Key.KPIName];

}

else if (dictKPIEntry.Key.KPIName == "NormDeviation")

{

value.Add("NormDeviation", new Default\_KPI\_Snippet.KPIField() { Type = Default\_KPI\_Snippet.KPIType.Ratio });

ValueKpiField = value[dictKPIEntry.Key.KPIName];

}

else if (dictKPIEntry.Key.KPIName == "CountOfTransaction")

{

value.Add("CountOfTransaction", new Default\_KPI\_Snippet.KPIField() { Type = Default\_KPI\_Snippet.KPIType.Scalar });

ValueKpiField = value[dictKPIEntry.Key.KPIName];

}

}

if (ValueKpiField == null) continue;

if (dictKPIEntry.Key.KPIName == "TransactionTime")

{

ValueKpiField.Add(dictKPIEntry.Value.TransactionTime);

}

if (dictKPIEntry.Key.KPIName == "CountOfTransaction")

{

ValueKpiField.Add(dictKPIEntry.Value.TranIdsCount.Count);

}

if (dictKPIEntry.Key.KPIName == "NormDeviation")

{

ValueKpiField.Add(dictKPIEntry.Value.NormDeviation);

}

//CountOfTransaction

}

return result;

}

public class TransactionNormKey

{

public int UserId;

public DateTime Day;

public string TransactionName;

public TransactionNormKey(int UserId, DateTime Day, string TransactionName)

{

this.UserId = UserId;

this.Day = Day.Date;

this.TransactionName = TransactionName;

}

}

public class TranCountWithDuration

{

public int Count;

public double Duration;

public TranCountWithDuration()

{

Count = 0;

Duration = 0.0;

}

public double GetNorm()

{

return Duration / Count;

}

public void Add(TranCountWithDuration other)

{

Count += other.Count;

Duration += other.Duration;

}

}

public void CreateTransactionToIDXValues()

{

var totalTimePeriod = ReportStartDate.ToString("yyyy.MM.dd") + " - " +

ReportEndDate.Date.ToString("yyyy.MM.dd");

foreach (var Transaction in dictMain)

{

if (TranNormExludingConditionSet.Contains(Transaction.Key.TransactionName)) continue;

var Group = GetGroups(GroupLevels, Transaction.Key.UserId, helper);

var group1 = Group[0];

var group2 = Group[1];

var dayOfYear = Transaction.Value.StartDate.ToString("yyyy.MM.dd");

var weekOfYear = Transaction.Value.StartDate.Year + "-" +

GetWeekOfYearFromCalendar(GetReportContext(), Transaction.Value.StartDate);

var monthOfYear = Transaction.Value.StartDate.ToString("yyyy.MM");

DictTRtoIDXKey key;

// TransactionTime

{

//UserDay

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = dayOfYear, UserPeriodString = "UserDay", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//UserWeek

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = weekOfYear, UserPeriodString = "UserWeek", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//UserMonth

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = monthOfYear, UserPeriodString = "UserMonth", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

// UserTotal

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = totalTimePeriod, UserPeriodString = "UserTotal", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Day

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = dayOfYear, UserPeriodString = "Group1Day", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Week

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = weekOfYear, UserPeriodString = "Group1Week", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Month

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = monthOfYear, UserPeriodString = "Group1Month", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Total

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = totalTimePeriod, UserPeriodString = "Group1Total", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Day

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = dayOfYear, UserPeriodString = "Group2Day", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Week

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = weekOfYear, UserPeriodString = "Group2Week", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Month

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = monthOfYear, UserPeriodString = "Group2Month", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Total

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = totalTimePeriod, UserPeriodString = "Group2Total", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllDay

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = dayOfYear, UserPeriodString = "AllDay", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllWeek

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = weekOfYear, UserPeriodString = "AllWeek", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllMonth

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = monthOfYear, UserPeriodString = "AllMonth", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllTotal

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = totalTimePeriod, UserPeriodString = "AllTotal", KPIName = "TransactionTime" };

AddTransactionToTR2IDXdict(key, Transaction);

}

if (true) // CountOfTransaction

{

//UserDay

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = dayOfYear, UserPeriodString = "UserDay", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//UserWeek

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = weekOfYear, UserPeriodString = "UserWeek", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//UserMonth

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = monthOfYear, UserPeriodString = "UserMonth", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

// UserTotal

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = totalTimePeriod, UserPeriodString = "UserTotal", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Day

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = dayOfYear, UserPeriodString = "Group1Day", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Week

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = weekOfYear, UserPeriodString = "Group1Week", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Month

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = monthOfYear, UserPeriodString = "Group1Month", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Total

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = totalTimePeriod, UserPeriodString = "Group1Total", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Day

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = dayOfYear, UserPeriodString = "Group2Day", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Week

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = weekOfYear, UserPeriodString = "Group2Week", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Month

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = monthOfYear, UserPeriodString = "Group2Month", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Total

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = totalTimePeriod, UserPeriodString = "Group2Total", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllDay

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = dayOfYear, UserPeriodString = "AllDay", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllWeek

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = weekOfYear, UserPeriodString = "AllWeek", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllMonth

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = monthOfYear, UserPeriodString = "AllMonth", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

//AllTotal

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = totalTimePeriod, UserPeriodString = "AllTotal", KPIName = "CountOfTransaction" };

AddTransactionToTR2IDXdict(key, Transaction);

}

// TransactionUserNorm

ValueDictStatus valst;

if (dictStatus.TryGetValue(Transaction.Key.TransactionId, out valst) && dictStatus[Transaction.Key.TransactionId].UserIds.Count < 2 && dictStatus[Transaction.Key.TransactionId].TotalTime > new TimeSpan(0, 0, 0, 0))

{

dictStatus[Transaction.Key.TransactionId].Closed = true;

//KPI : Norm

var TrNameWithPrefix = "Norm -" + Transaction.Key.TransactionCategory + Transaction.Key.TransactionName;

//UserDay

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = dayOfYear, UserPeriodString = "UserDay", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//UserWeek

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = weekOfYear, UserPeriodString = "UserWeek", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//UserMonth

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = monthOfYear, UserPeriodString = "UserMonth", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//UserTotal

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = totalTimePeriod, UserPeriodString = "UserTotal", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Day

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = dayOfYear, UserPeriodString = "Group1Day", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Week

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = weekOfYear, UserPeriodString = "Group1Week", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Month

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = monthOfYear, UserPeriodString = "Group1Month", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Month

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = totalTimePeriod, UserPeriodString = "Group1Total", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Day

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = dayOfYear, UserPeriodString = "Group2Day", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Week

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = weekOfYear, UserPeriodString = "Group2Week", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Month

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = monthOfYear, UserPeriodString = "Group2Month", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Month

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = totalTimePeriod, UserPeriodString = "Group2Total", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllDay

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = dayOfYear, UserPeriodString = "AllDay", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllWeek

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = weekOfYear, UserPeriodString = "AllWeek", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllMonth

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = monthOfYear, UserPeriodString = "AllMonth", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllTotal

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = totalTimePeriod, UserPeriodString = "AllTotal", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

}

// TransactionTotalNorm

ValueDictStatus valst2;

if (dictStatus.TryGetValue(Transaction.Key.TransactionId, out valst2) && dictStatus[Transaction.Key.TransactionId].TotalTime > new TimeSpan(0, 0, 0, 0))

{

//KPI : Norm

var TrNameWithPrefix = "NorT -" + Transaction.Key.TransactionCategory + "-" + Transaction.Key.TransactionName;

//UserDay

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = dayOfYear, UserPeriodString = "UserDay", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//UserWeek

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = weekOfYear, UserPeriodString = "UserWeek", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//UserMonth

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = monthOfYear, UserPeriodString = "UserMonth", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//UserTotal

key = new DictTRtoIDXKey() { UserOrGroupId = Transaction.Key.UserId, Period = totalTimePeriod, UserPeriodString = "UserTotal", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Day

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = dayOfYear, UserPeriodString = "Group1Day", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Week

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = weekOfYear, UserPeriodString = "Group1Week", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Month

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = monthOfYear, UserPeriodString = "Group1Month", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group1Month

key = new DictTRtoIDXKey() { UserOrGroupId = group1, Period = totalTimePeriod, UserPeriodString = "Group1Total", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Day

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = dayOfYear, UserPeriodString = "Group2Day", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Week

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = weekOfYear, UserPeriodString = "Group2Week", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Month

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = monthOfYear, UserPeriodString = "Group2Month", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//Group2Month

key = new DictTRtoIDXKey() { UserOrGroupId = group2, Period = totalTimePeriod, UserPeriodString = "Group2Total", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllDay

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = dayOfYear, UserPeriodString = "AllDay", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllWeek

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = weekOfYear, UserPeriodString = "AllWeek", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllMonth

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = monthOfYear, UserPeriodString = "AllMonth", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

//AllTotal

key = new DictTRtoIDXKey() { UserOrGroupId = 1, Period = totalTimePeriod, UserPeriodString = "AllTotal", KPIName = TrNameWithPrefix };

AddTransactionToTR2IDXdict(key, Transaction);

}

}

foreach (var kpi in tempDictKPI.Where(k => k.Key.KPIName.StartsWith("Norm -")))

{

var userNorm = kpi.Value.TransactionTime.TotalMilliseconds / kpi.Value.Count;

var TotalForTran =

tempDictKPI[

new DictTRtoIDXKey()

{

UserOrGroupId = 1,

Period = totalTimePeriod,

UserPeriodString = "AllTotal",

KPIName = kpi.Key.KPIName

}];

var processNorm = TotalForTran.TransactionTime.TotalMilliseconds / TotalForTran.Count;

helper.Log(string.Format("Total {0} time: {1}, Total {0} count: {2}, {0}: {3}", kpi.Key.KPIName,

TotalForTran.TransactionTime.ToString("g"), TotalForTran.Count, processNorm));

TranCountWithDuration userTransactionNormDeviationValue;

if (!userTransactionNormDeviation.TryGetValue(kpi.Key.UserOrGroupId + kpi.Key.Period,

out userTransactionNormDeviationValue))

{

userTransactionNormDeviation.Add(kpi.Key.UserOrGroupId + kpi.Key.Period, new TranCountWithDuration());

userTransactionNormDeviationValue = userTransactionNormDeviation[kpi.Key.UserOrGroupId + kpi.Key.Period];

}

userTransactionNormDeviationValue.Duration += (userNorm / processNorm) \* kpi.Value.Count;

userTransactionNormDeviationValue.Count += kpi.Value.Count;

}

foreach (var kpi in tempDictKPI.Where(k => k.Key.KPIName.StartsWith("NorT -")))

{

var userNorm = kpi.Value.TransactionTime.TotalMilliseconds / kpi.Value.Count;

var TotalForTran =

tempDictKPI[

new DictTRtoIDXKey()

{

UserOrGroupId = 1,

Period = totalTimePeriod,

UserPeriodString = "AllTotal",

KPIName = kpi.Key.KPIName

}];

var processNorm = TotalForTran.TransactionTime.TotalMilliseconds / TotalForTran.Count;

helper.Log(string.Format("Total Norma --- {0} time: {1}, Total {0} count: {2}, {0}: {3}", kpi.Key.KPIName, TotalForTran.TransactionTime.ToString("g"), TotalForTran.Count, processNorm));

dictNormaValue val;

if (!dictNorma.TryGetValue(kpi.Key.KPIName, out val) && kpi.Key.Period == totalTimePeriod && kpi.Key.UserPeriodString == "AllTotal")

{

var normVal = new dictNormaValue();

normVal.normCount += kpi.Value.TranIds.Count;

normVal.normTime += processNorm;

dictNorma.Add(kpi.Key.KPIName, normVal);

}

}

foreach (var kpi in tempDictKPI)

{

DictTRtoIDXValue kpiDictValue;

TranCountWithDuration Value;

if (kpi.Key.KPIName.StartsWith("Norm -"))

{

if (userTransactionNormDeviation.TryGetValue(kpi.Key.UserOrGroupId + kpi.Key.Period, out Value))

{

var key = new DictTRtoIDXKey()

{

Period = kpi.Key.Period,

UserOrGroupId = kpi.Key.UserOrGroupId,

UserPeriodString = kpi.Key.UserPeriodString,

KPIName = "NormDeviation"

};

if (!copyOfdictKPI.TryGetValue(key, out kpiDictValue))

{

copyOfdictKPI.Add(key, new DictTRtoIDXValue());

kpiDictValue = copyOfdictKPI[key];

}

kpiDictValue.Count += Value.Count;

kpiDictValue.NormDeviation = 1 - (Value.Duration / Value.Count);

}

}

else if (kpi.Key.KPIName == "TransactionTime")

{

var key = new DictTRtoIDXKey() { Period = kpi.Key.Period, UserOrGroupId = kpi.Key.UserOrGroupId, UserPeriodString = kpi.Key.UserPeriodString, KPIName = "TransactionTime" };

if (!copyOfdictKPI.TryGetValue(key, out kpiDictValue))

{

copyOfdictKPI.Add(key, kpi.Value);

kpiDictValue = copyOfdictKPI[key];

}

}

else if (kpi.Key.KPIName == "CountOfTransaction")

{

var key = new DictTRtoIDXKey() { Period = kpi.Key.Period, UserOrGroupId = kpi.Key.UserOrGroupId, UserPeriodString = kpi.Key.UserPeriodString, KPIName = "CountOfTransaction" };

if (!copyOfdictKPI.TryGetValue(key, out kpiDictValue))

{

copyOfdictKPI.Add(key, kpi.Value);

kpiDictValue = copyOfdictKPI[key];

}

}

}

dictKPI = copyOfdictKPI;

}

public void AddTransactionToTR2IDXdict(DictTRtoIDXKey key, KeyValuePair<KeyTransactionReport, ValueGeneral> Transaction)

{

dictValue = null;

if (!tempDictKPI.TryGetValue(key, out dictValue))

{

tempDictKPI.Add(key, new DictTRtoIDXValue());

dictValue = tempDictKPI[key];

}

dictValue.Add(Transaction);

}

public class DictTRtoIDXKey : IEquatable<DictTRtoIDXKey>

{

//public string TransactionName;

public int UserOrGroupId;

public string Period;

public string UserPeriodString;

public string KPIName;

public override int GetHashCode()

{

unchecked

{

var hash = 17;

hash = hash \* 397 ^ UserOrGroupId.GetHashCode();

hash = hash \* 397 ^ (Period ?? "").GetHashCode();

hash = hash \* 397 ^ (UserPeriodString ?? "").GetHashCode();

hash = hash \* 397 ^ (KPIName ?? "").GetHashCode();

return hash;

}

}

public bool Equals(DictTRtoIDXKey other)

{

return UserOrGroupId.Equals(other.UserOrGroupId)

&& string.Equals(Period, other.Period)

&& string.Equals(UserPeriodString, other.UserPeriodString)

&& string.Equals(KPIName, other.KPIName);

}

public override bool Equals(object other)

{

if (ReferenceEquals(other, null)) return false;

if (ReferenceEquals(this, other)) return true;

return this.Equals((DictTRtoIDXKey)other);

}

}

public class DictTRtoIDXValue

{

public TimeSpan TransactionTime;

private int \_count;

public int Count

{

get

{

\_count = TranIds.Count;

return \_count;

}

set

{

\_count = value;

}

}

public HashSet<string> TranIds;

public HashSet<string> TranIdsCount;

public double NormDeviation;

public DictTRtoIDXValue()

{

Count = 0;

TranIds = new HashSet<string>();

TranIdsCount = new HashSet<string>();

TransactionTime = TimeSpan.Zero;

NormDeviation = 0.0;

}

public void Add(KeyValuePair<KeyTransactionReport, ValueGeneral> transaction)

{

TranIdsCount.Add(transaction.Key.TransactionCategory + "-" + transaction.Key.TransactionName + "-" + transaction.Key.TransactionId);

TranIds.Add(transaction.Key.TransactionId);

TransactionTime += transaction.Value.CaseTime;

}

}

private static DateTime MapDateToMonth(DateTime date)

{

return new DateTime(date.Year, date.Month, 1);

}

#endregion

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* I really do not care about this shit below \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#region TRNInputConfig

//int xlsMaxRowNumber = 1000000;

//double CutTransaction = 0.0; //A tranzakciós kategóriánként ennyi százalékot vágunk le a min és max értékekbol. 0 és 1 közötti értéket vehet fel az elvárt muködéshez.

public bool isTransaction = false;

public const bool isTransactionReport = false;

public const bool isTransactionNormaReport = true;

public const bool isKPIReport = false;

public const bool isEvaluationReport = false;

public const bool isPerformanceEvaluation = false;

private int startLevel = 0;

public string Virt = "Virt - ";

public int VirtUniqueID = 0;

public bool newContactValue = false;

public bool newContactValueOverride = false;

public List<int> GroupLevels;// = new List<int>();

public Dictionary<string, KeyTransactionReport> dictLastKeys = new Dictionary<string, KeyTransactionReport>();

public Dictionary<KeyTransactionReport, ValueGeneral> dictMain = new Dictionary<KeyTransactionReport, ValueGeneral>();

public Dictionary<KeyTransactionReport, List<WorkItemExt>> dictMainDebug = new Dictionary<KeyTransactionReport, List<WorkItemExt>>();

public Dictionary<string, ValueDictStatus> dictStatus = new Dictionary<string, ValueDictStatus>();

public Dictionary<KeyTransactionReport, ValueGeneral> dictMainTemp = new Dictionary<KeyTransactionReport, ValueGeneral>();

public Dictionary<string, ValueDictStatus> dictStatusTemp = new Dictionary<string, ValueDictStatus>();

#endregion

#region TranDictComponents

public KeyTransactionReport valueTest;

public class KeyTransactionReport : IEquatable<KeyTransactionReport>

{

public int TaskId;

public int UserId;

public string TransactionName;

public string TransactionCategory;

public string TransactionId;

public string TransactionStatus;

public string DecoratedTaskName; //task, details

public DateTime? DateSplit;

public bool IsInOffice;

public string ProductiveState;

public string IncludedWorkName;

public string TransactionType;

public bool IsInOvertime;

public string ProcessName;

public bool Equals(KeyTransactionReport other)

{

if (ReferenceEquals(null, other)) return false;

if (ReferenceEquals(this, other)) return true;

return TaskId == other.TaskId && UserId == other.UserId

&& string.Equals(TransactionName, other.TransactionName)

&& string.Equals(TransactionCategory, other.TransactionCategory)

&& string.Equals(TransactionId, other.TransactionId)

&& string.Equals(DecoratedTaskName, other.DecoratedTaskName)

&& DateSplit.Equals(other.DateSplit)

&& IsInOffice == other.IsInOffice

&& string.Equals(ProductiveState, other.ProductiveState)

&& string.Equals(IncludedWorkName, other.IncludedWorkName)

&& string.Equals(TransactionStatus, other.TransactionStatus)

&& string.Equals(TransactionType, other.TransactionType)

&& IsInOvertime == other.IsInOvertime;

}

public override bool Equals(object obj)

{

if (ReferenceEquals(null, obj)) return false;

if (ReferenceEquals(this, obj)) return true;

return obj.GetType() == GetType() && Equals((KeyTransactionReport)obj);

}

public override int GetHashCode()

{

unchecked

{

var hashCode = TaskId;

hashCode = (hashCode \* 397) ^ UserId;

hashCode = (hashCode \* 397) ^ (TransactionName != null ? TransactionName.GetHashCode() : 0);

hashCode = (hashCode \* 397) ^ (TransactionCategory != null ? TransactionCategory.GetHashCode() : 0);

hashCode = (hashCode \* 397) ^ (TransactionId != null ? TransactionId.GetHashCode() : 0);

hashCode = (hashCode \* 397) ^ (DecoratedTaskName != null ? DecoratedTaskName.GetHashCode() : 0);

hashCode = (hashCode \* 397) ^ DateSplit.GetHashCode();

hashCode = (hashCode \* 397) ^ IsInOffice.GetHashCode();

hashCode = (hashCode \* 397) ^ (ProductiveState != null ? ProductiveState.GetHashCode() : 0);

hashCode = (hashCode \* 397) ^ (IncludedWorkName != null ? IncludedWorkName.GetHashCode() : 0);

hashCode = (hashCode \* 397) ^ (TransactionStatus ?? "").GetHashCode();

hashCode = (hashCode \* 397) ^ (TransactionType ?? "").GetHashCode();

hashCode = (hashCode \* 397) ^ IsInOvertime.GetHashCode();

hashCode = (hashCode \* 397) ^ (ProcessName ?? "").GetHashCode();

return hashCode;

}

}

}

public class ValueDictStatus

{

public string RealTransactionId;

public HashSet<string> TransactionKeyId;

public string TransactionCategory;

public bool? Closed;

public DateTime ClosedDateTime;

public DateTime FirstDateTime;

public int? FirstWorkId;

public int? FirstUserId;

public DateTime LastDateTime;

public int LastWorkId;

public int LastUserId;

public string LastTransactionStatus;

public TimeSpan OfficeTime;

public TimeSpan InOvertimeTime;

public TimeSpan TotalTime;

public int? Count;

public HashSet<int> UserIds;

public HashSet<int> WorkId;

public ValueDictStatus()

{

RealTransactionId = "";

TransactionKeyId = new HashSet<string>();

TransactionCategory = "";

Closed = false;

ClosedDateTime = DateTime.MinValue;

FirstDateTime = DateTime.MinValue;

FirstWorkId = 0;

FirstUserId = 0;

LastDateTime = DateTime.MinValue;

LastWorkId = 0;

LastUserId = 0;

LastTransactionStatus = "";

OfficeTime = TimeSpan.Zero;

InOvertimeTime = TimeSpan.Zero;

TotalTime = TimeSpan.Zero;

Count = 0;

UserIds = new HashSet<int>();

WorkId = new HashSet<int>();

}

}

public class ValueGeneral

{

public TimeSpan Active;

public TimeSpan Inactive;

public TimeSpan Adhoc;

public TimeSpan Calendar;

public TimeSpan Manual;

public TimeSpan Mobile;

public TimeSpan Holiday;

public TimeSpan SickLeave;

public TimeSpan Offline;

public TimeSpan Total;

public TimeSpan TotalWithOffline;

public TimeSpan IsBusyTime;

public TimeSpan CaseTime

{

get

{

return Active + Adhoc + Mobile + Calendar;

}

}

public DateTime StartDate;

public DateTime EndDate;

public int Count;

public TimeSpan IsInOffice;

public ValueGeneral()

{

Active = TimeSpan.Zero;

Inactive = TimeSpan.Zero;

Adhoc = TimeSpan.Zero;

Calendar = TimeSpan.Zero;

Manual = TimeSpan.Zero;

Mobile = TimeSpan.Zero;

Holiday = TimeSpan.Zero;

SickLeave = TimeSpan.Zero;

Offline = TimeSpan.Zero;

Total = TimeSpan.Zero;

TotalWithOffline = TimeSpan.Zero;

IsBusyTime = TimeSpan.Zero;

Count = 0;

StartDate = DateTime.MaxValue;

EndDate = DateTime.MinValue;

IsInOffice = TimeSpan.Zero;

}

private TimeSpan offline = TimeSpan.Zero;

public ValueGeneral Add(WorkItem item)

{

var sw = new Stopwatch(); sw.Start();

switch (item.Type)

{

case ItemType.AdhocMeeting:

Adhoc += item.Duration;

break;

case ItemType.CalendarMeeting:

Calendar += item.Duration;

break;

case ItemType.Holiday:

Holiday += item.Duration;

break;

case ItemType.Manual:

Manual += item.Duration;

break;

case ItemType.Mobile:

Mobile += item.Duration;

break;

case ItemType.Pc:

if (item.WorkId != -1)

{

var isActive = GetValueOrDefault(item.Values, "IsActive");

if (isActive == "1") Active += item.Duration;

else /\*if (isActive == "0")\*/ Inactive += item.Duration;

var isBusy = GetValueOrDefault(item.Values, "IsBusyTime");

if (isBusy == "1") IsBusyTime += item.Duration;

}

else

offline += item.Duration;

break;

case ItemType.SickLeave:

SickLeave += item.Duration;

break;

}

Total += item.Duration - offline;

offline = TimeSpan.Zero;

TotalWithOffline += item.Duration;

Count++;

if (StartDate > item.StartDate)

StartDate = item.StartDate;

if (EndDate < item.EndDate)

EndDate = item.EndDate;

if (GetValueOrDefault(item.Values, "IsInOffice") == "1")

{

IsInOffice += item.Duration;

}

return this;

}

public ValueGeneral Add(ValueGeneral other)

{

Active += other.Active;

Inactive += other.Inactive;

Adhoc += other.Adhoc;

Calendar += other.Calendar;

Manual += other.Manual;

Mobile += other.Mobile;

Holiday += other.Holiday;

SickLeave += other.SickLeave;

Offline += other.Offline;

Total += other.Total;

TotalWithOffline += other.TotalWithOffline;

IsBusyTime += other.IsBusyTime;

Count += other.Count;

if (StartDate > other.StartDate)

StartDate = other.StartDate;

if (EndDate < other.EndDate)

EndDate = other.EndDate;

IsInOffice += other.IsInOffice;

return this;

}

}

#endregion

#region GeneralFunctions

public List<int> GetGroups(List<int> arrGroups, int UserID, InnerCode helper)

{

var userGroupLevels = new List<int>();

try

{

foreach (int Levels in arrGroups)

{

if (Levels >= 0)

{

var AllLevelsRtoP = Users.GetParentIds(UserID, HierarchyCacheParentListReturnOrder.RootToParent);

if (AllLevelsRtoP.Count >= Levels)

{

userGroupLevels.Add(AllLevelsRtoP[Levels - 1]);

}

else

{

userGroupLevels.Add(UserID);

}

}

else

{

var AllLevelsPtoR = Users.GetParentIds(UserID, HierarchyCacheParentListReturnOrder.ParentToRoot);

if (AllLevelsPtoR.Count >= Math.Abs(Levels))

{

userGroupLevels.Add(AllLevelsPtoR[Math.Abs(Levels) - 1]);

}

else

{

userGroupLevels.Add(UserID);

}

}

}

return userGroupLevels;

}

catch (Exception e)

{

helper.Log("mivaan " + e);

userGroupLevels.Add(UserID);

userGroupLevels.Add(UserID);

return userGroupLevels;

}

}

// get included URL, processes...

private string GetWorkNameFromTup(string url, string process)

{

if (!String.IsNullOrWhiteSpace(url))

{

if (ExcludedURLList.Any(u => url.IndexOf(u, StringComparison.OrdinalIgnoreCase) > -1))

{

return null;

}

if (ApplicationNameListURL\_used.ContainsKey(url))

{ //found in cache

return ApplicationNameListURL\_used[url];

}

else

{

foreach (var urlTask in ApplicationNameListURL)

{

if (Regex.IsMatch(url, urlTask.Key, RegexOptions.IgnoreCase))

{

ApplicationNameListURL\_used[url] = urlTask.Value; //caching

return urlTask.Value;

}

}

}

}

else if (!String.IsNullOrWhiteSpace(process))

{

if (ApplicationNameListProc.ContainsKey(process)) return ApplicationNameListProc[process];

}

return null;

}

public class TransactionType

{

public string TransactionName;

public HashSet<string> IncludedEmailAddresses;

public TransactionType(string transactionName, IEnumerable<string> includedEmailAddresses = null)

{

TransactionName = transactionName;

if (includedEmailAddresses == null)

{

IncludedEmailAddresses = new HashSet<string>(StringComparer.OrdinalIgnoreCase);

}

else

{

IncludedEmailAddresses = new HashSet<string>(includedEmailAddresses, StringComparer.OrdinalIgnoreCase);

}

}

public bool EvaluationEmailAddresses(string emailFrom, string emailTo, out string emailAddress)

{

if (IncludedEmailAddresses.Count > 0)

{

if (!String.IsNullOrWhiteSpace(emailFrom) && IncludedEmailAddresses.Contains(emailFrom.Trim()))

{

emailAddress = emailFrom;

return true;

}

if (!String.IsNullOrWhiteSpace(emailTo))

{

var splitedEmailTo = emailTo.Split(new[] { ';' });

foreach (var email in splitedEmailTo)

{

if (!String.IsNullOrWhiteSpace(email) && IncludedEmailAddresses.Contains(email.Trim()))

{

emailAddress = email;

return true;

}

}

}

}

emailAddress = "";

return false;

}

}

public string MoreThan24HoursTimeSpanToString(TimeSpan ts, bool seconds)

{

var sb = new System.Text.StringBuilder();

if (ts.TotalHours < 10 && ts.TotalHours > 0) sb.Append("0" + (int)ts.TotalHours);

else if (ts.TotalHours == 0) sb.Append("00");

else sb.Append((ts.Days \* 24 + ts.Hours));

sb.Append(":");

if (ts.Minutes < 10 && ts.Minutes > 0) sb.Append("0" + ts.Minutes.ToString("##"));

else if (ts.Minutes == 0) sb.Append("00");

else sb.Append(ts.Minutes.ToString("##"));

if (seconds)

{

sb.Append(":");

if (ts.Seconds < 10 && ts.Seconds > 0) sb.Append("0" + ts.Seconds.ToString("##"));

else if (ts.Seconds == 0) sb.Append("00");

else sb.Append(ts.Seconds.ToString("##"));

}

return sb.ToString();

}

public string GetUserNameWithPath(int UserId, int Format = 2, string Separator = " » ")

{

if (UserId == 0) return null;

var userName = helper.GetUserName(UserId);

var parentGroups = helper.GetGroupsOfUser(UserId);

var groupPath = "";

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

{

if (!String.IsNullOrWhiteSpace(groupPath)) groupPath += Separator;

groupPath += parentGroups[i];

}

if (Format == 1)

{

return groupPath + Separator + userName;

}

else

{

return groupPath;

}

}

public bool IsWeekend(DateTime date)

{

return new[] { DayOfWeek.Sunday, DayOfWeek.Saturday }.Contains(date.DayOfWeek);

}

public string GetTaskNameWithPath(int TaskId, int Format = 2, string Separator = " » ")

{

if (TaskId == 0) return null;

var workName = helper.GetWorkName(TaskId);

var parentTasks = helper.GetParentTasks(TaskId);

var workPath = "";

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

{

if (workPath == "")

{

workPath = parentTasks[i].Name;

}

else

{

workPath += Separator + parentTasks[i].Name;

}

}

if (Format == 1)

{

return workPath + Separator + workName;

}

else

{

return workPath;

}

}

public int GetWeekOfYearFromCalendar(ReportContext ReportContext, DateTime Date)

{

var currentCultureInfo = CultureInfo.GetCultureInfo(ReportContext.ReportCultureName);

return currentCultureInfo.Calendar.GetWeekOfYear(Date, CalendarWeekRule.FirstFourDayWeek, DayOfWeek.Monday);

}

#endregion

#region included URL, processes...

public HashSet<int> BreakTimeIds = new HashSet<int>

{

};

public HashSet<int> excludeTaskIDs = new HashSet<int>()

{

};

public HashSet<int> enabledTaskIds = new HashSet<int>()

{

};

HashSet<string> ExcludedURLList = new HashSet<string>(StringComparer.OrdinalIgnoreCase)

{

};

private Dictionary<string, string> ApplicationNameListURL\_used = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase);

Dictionary<string, string> ApplicationNameListURL = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase)

{

};

Dictionary<string, string> ApplicationNameListProc = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase)

{

};

Dictionary<string, string> ApplicationNameListFilePath = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase)

{

};

#endregion

public InnerCode helper;

public WorkItemExtConcatenatorConfig ConcatenatorConfig = new WorkItemExtConcatenatorConfig();

public WorkItemExtConcatenatorProviderLinear WorkItemExtConcatenator;

public HierarchyCacheProviderConfig HierarchyCacheProviderDefaultConfig = new HierarchyCacheProviderConfig();

public HierarchyCacheProvider<UserExt, User> Users;

public HierarchyCacheProvider<TaskExt, Task> Tasks;

public IntervalTimerProvider IntervalTimer;

public ReportLogProvider ReportLogger;

public ProductivityProvider Productivity;

public WorktimeSettingsProvider WorktimeSettings;

public RuntimeContextProvider RuntimeContext;

#region Initialization

public bool Configure()

{

ConcatenatorConfig.ItemTypeImportance = ItemTypeImportance;

ConcatenatorConfig.ExcludedTaskIds = ExcludedTaskIds;

ConcatenatorConfig.ExcludedWorkItemTypes = ExcludedWorkItemTypes;

ConcatenatorConfig.TaskIdsWeight = TaskIdsWeight;

ConcatenatorConfig.PositiveTaskIds = PositiveTaskIds;

ConcatenatorConfig.NegativeTaskIds = NegativeTaskIds;

ConcatenatorConfig.DisableNetting = !EnableNetting;

ConcatenatorConfig.EnableIgnoreHolidays = EnableIgnoreHolidays;

HierarchyCacheProviderDefaultConfig.Separator = HierarchyCacheProviderItemSeparator;

HierarchyCacheProviderDefaultConfig.EncloserFormat = HierarchyCacheProviderEncloserFormat;

return true;

}

public bool IsTransactionValidForNorm(string TransactionName)

{

return !TranNormExludingConditionSet.Contains(TransactionName);

}

private bool IsOvertime(WorkItem item)

{

var isOvertimeFromScheadule = IsOvertimeFromSchedule(item);

if (isOvertimeFromScheadule.HasValue)

return isOvertimeFromScheadule.Value;

var isOvertimeFromWorktimeSettings = IsOvertimeFromWorktimeSettings(item);

return isOvertimeFromWorktimeSettings.HasValue ? isOvertimeFromWorktimeSettings.Value : false;

}

private bool? IsOvertimeFromSchedule(WorkItem item) //TODO: Improve performance

{

var schedulesOfUser = helper.GetDailySchedulesForUser(item.UserId, item.StartDate.Date);

var scheduleItemsOfUser = schedulesOfUser.Where(s => s.Items != null)

.SelectMany(s => s.Items, (s, i) => new { i.Start, i.End, i.IsOvertime, s.IsInOffice })

.OrderBy(si => si.Start)

.ToList();

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

{

var rangeStart = i > 0

? Average(scheduleItemsOfUser[i - 1].End, scheduleItemsOfUser[i].Start)

: DateTime.MinValue;

var rangeEnd = i < scheduleItemsOfUser.Count - 1

? Average(scheduleItemsOfUser[i].End, scheduleItemsOfUser[i + 1].Start)

: DateTime.MaxValue;

if (item.StartDate >= rangeStart && item.StartDate < rangeEnd)

return scheduleItemsOfUser[i].IsOvertime;

}

return null;

}

private DateTime Average(DateTime d1, DateTime d2)

{

return d1 + new TimeSpan((d2 - d1).Ticks / 2);

}

private bool? IsOvertimeFromWorktimeSettings(WorkItem item)

{

var day = item.StartDate.Date;

var worktimeSettings = helper.GetWorktimeSettingsForDay(item.UserId, day);

if (worktimeSettings != null)

return item.EndDate > day.AddMinutes(worktimeSettings.CoreTimeEndInMinutes); //TODO: split WorkItem

return null;

}

public bool Initialize()

{

if (!Configure()) return false;

try

{

IntervalTimer = new IntervalTimerProvider(helper);

ReportLogger = new ReportLogProvider("Central", helper);

RuntimeContext = new RuntimeContextProvider(helper, ReportLogger, IntervalTimer);

Users = new HierarchyCacheProvider<UserExt, User>("Users Cache", HierarchyCacheProviderDefaultConfig, RuntimeContext);

Tasks = new HierarchyCacheProvider<TaskExt, Task>("Tasks Cache", HierarchyCacheProviderDefaultConfig, RuntimeContext);

Productivity = new ProductivityProvider("Productivity", RuntimeContext, Users, Tasks, ReportLogger.Log, null);

WorktimeSettings = new WorktimeSettingsProvider("Worktime settings", RuntimeContext);

WorkItemExtConcatenator = new WorkItemExtConcatenatorProviderLinear(ConcatenatorConfig, RuntimeContext, Productivity, WorktimeSettings);

}

catch (Exception e)

{

if (ReportLogger != null && ReportLogger.LogHasContent)

{

ReportLogger.OutputReportLog();

}

helper.Log(string.Format("{0} exception caught during initialization.", e));

return false;

}

return true;

}

public void SetExcelFileName()

{

helper.SetFileName(string.Format("{0}\_{1}\_{2}\_{3}", \_ctx.DefinitionName, \_ctx.CompanyName,

\_ctx.LocalStartDate.ToString("yyyMMdd"),

\_ctx.LocalEndDate.ToString("yyyMMdd")));

}

#endregion

#endregion

}