

**ОТЧЕТ**

**ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ №1**

|  |  |  |
| --- | --- | --- |
| Глызин Андрей Сергеевич | | |
| *(Ф.И.О. обучающегося)* | | |
| 02.03.02 Фундаментальная информатика и информационные технологии, 51 Разработка программного обеспечения | | |
| *(направление подготовки (специальность), направленность (профиль))* | | |
|  | | |
| Место прохождения практики | | УФССП России по Кировской области, |
|  | *(наименование организации)* | |
| отдел информатизации и обеспечения информационной безопасности | | |
| *(наименование структурного подразделения организации)* | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Итоговая оценка: |  | | | | |
| Руководитель  практики от университета |  |  |  |  | И.Н. Фищева |
|  | *(дата)* |  | *(подпись)* |  | *(Ф.И.О.)* |

Киров, 2021 г.

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 3](#_Toc76116946)

[ВВЕДЕНИЕ 4](#_Toc76116947)

[ХАРАКТЕРИСТИКА МЕСТА ПРОХОЖДЕНИЯ ПРАКТИКИ 5](#_Toc76116948)

[ОТЧЕТ О ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 6](#_Toc76116949)

[Постановка задания 6](#_Toc76116950)

[Цель, задачи, которые необходимо решить для достижения поставленной цели 6](#_Toc76116951)

[Обзор имеющихся подходов к решению 6](#_Toc76116952)

[Обоснование выбора методов 7](#_Toc76116953)

[Описание решения 7](#_Toc76116954)

[Итоги, заключение 7](#_Toc76116955)

[ЗАКЛЮЧЕНИЕ 9](#_Toc76116956)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 10](#_Toc76116957)

[ПРИЛОЖЕНИЯ 11](#_Toc76116958)

[Листинг проекта, реализующего решение поставленных задач 11](#_Toc76116959)

[Скриншоты графического интерфейса пользователя 29](#_Toc76116960)

# ВВЕДЕНИЕ

Производственная практика проходила в УФССП России по Кировской области, отдел информатизации и обеспечения информационной безопасности с 21.06.2021 г. по 4.07.2021 г., количество недель: 2.

Целью производственной практики является получение профессиональных умений и компетенций в сфере профессиональной деятельности.

В ходе прохождения практики необходимо:

* сформировать полное представление о своей профессии;
* выполнить обязанности на первичных должностях в области применения современных математических методов и информационных технологий;
* получить опыт применения математического аппарата и современных информационных технологий для анализа и решения конкретных производственных задач
* закрепить практические навыки по применению пакетов прикладных программ, разработке прикладного и системного программного обеспечения;
* получить навыки самостоятельного или коллективного решения конкретных профессиональных задач;
* приобрести опыт самостоятельной профессиональной деятельности;
* выполнить индивидуальное задание.

# ХАРАКТЕРИСТИКА МЕСТА ПРОХОЖДЕНИЯ ПРАКТИКИ

Федеральная служба судебных приставов осуществляет функции по обеспечению установленного порядка деятельности судов, исполнению судебных актов, актов других органов и должностных лиц, а также правоприменительные функции и функции по контролю и надзору в установленной сфере деятельности.

Основной вид деятельности отдела информатизации и обеспечения информационной безопасности заключается в

1. Обеспечении функционирования информационных систем в установленной сфере деятельности.
2. Организации работы по созданию и ведению в установленном порядке возбуждении исполнительных производств, автоматизация информационных производственных процессов в Управлении.
3. Осуществлении справочно-информационного сопровождения деятельности структурных подразделений Управления.
4. Обеспечения информационной безопасности Управления.

Описание рабочего места (используемое аппаратное обеспечение):

* Персональный компьютер с предустановленной операционной системой Windows 10

Программное обеспечение, используемое в работе:

* Visual Studio 2019

С рабочих мест обеспечен доступ к Internet.

# ОТЧЕТ О ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

# Постановка задания

Разработка методов, алгоритмов и программного обеспечения, направленных на реализацию мониторинга деятельности работников организации.

# Цель, задачи, которые необходимо решить для достижения поставленной цели

1. Пройти инструктаж по ознакомлению с правилами внутреннего трудового распорядка, охраны труда, техники безопасности, противопожарной безопасности, санитарно-эпидемиологическими правилами и гигиеническими нормативами, а также вводный инструктаж и инструктаж на рабочем месте.
2. Сбор информации о показателях деятельности работников организации, необходимых для учета в системе мониторинга деятельности.
3. Разработка методов, алгоритмов и программного обеспечения, направленных на реализацию мониторинга деятельности работников организации.

# Обзор имеющихся подходов к решению

Передача статистики и сообщение компьютеров в сети было реализовано с помощью протокола TCP/IP.

Для сохранения настроек приложений был использован формат json и библиотека Newtonsoft.Json.

Работа с форматом xlsx была обеспеченна библиотекой ClosedXML.

Для экспорта статистики в pdf формат использовалась библиотека itextsharp.

# Обоснование выбора методов

Использование протокола TCP/IP обусловлено его простотой и минимальным количеством условий для функционирования.

Для работы с используемыми форматами использовались самые популярные и востребованные библиотеки.

# Описание решения

Для анализа показателей деятельности работников организации было реализовано приложение, которое в зависимости от индекса значимости этих критериев оценивает выполнение должностных обязанностей работников организации.

Одним из таких критериев может быть оценка использования времени на рабочем месте. Для сбора и анализа данных было дополнительно реализовано два приложения – Клиент и Сервер. Клиент в течение определённого промежутка времени собирает информацию о количестве времени проведённом в том или ином приложении, а затем передаёт эту информацию по локальной сети на Сервер. Сервер же в свою очередь отвечает за обработку этой информации. Для анализа этих данных необходим список ключевых слов и соответствующих им индексов значимости. Этот список составляется индивидуально каждой организацией в зависимости от её потребностей. После анализа данных они могут быть сопоставлены по именам компьютеров в сети и импортированы в основной xlsx файл.

Главное приложение также имеет функцию построения графиков по полученным данным и их печати.

# Итоги, заключение

В процессе выполнения практики пригодились знания, полученные в ходе изучения ранее пройденных дисциплин.

По окончании практики получены навыки эффективного использования программного обеспечения персонального компьютера, самостоятельного решения практических задач, что привело к систематизации теоретических знаний и позволило закрепить знания, умения и навыки, полученные в процессе теоретической подготовки на практике.

# ЗАКЛЮЧЕНИЕ

Производственная практика способствовала формированию представления о своей профессии, приобретению профессиональных навыков и умению в организации работы в области применения современных методов фундаментальной информатики и информационных технологий, приобретению опыта самостоятельной профессиональной деятельности путём самостоятельного решения поставленных задач.

В ходе практики было реализовано три приложения направленных на решение индивидуального задания.

Наиболее интересным было создание программного кода для анализа собранных данных в соответствии с заданными критериями оценки.

Наиболее сложным было создание программной реализации сообщения клиента и сервера посредством протокола TCP/IP.

По итогу, все поставленные в работе задачи решены, а цель достигнута.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Библиотека официальной технической документации для разработчиков под ОС Microsoft Windows [Электронный ресурс]. URL: <https://docs.microsoft.com/>.

Библиотека официальной технической документации Json.NET [Электронный ресурс]. URL: <https://github.com/JamesNK/Newtonsoft.Json>.

Библиотека официальной технической документации ClosedXML [Электронный ресурс]. URL: <https://github.com/ClosedXML/ClosedXML>.

Библиотека официальной технической документации iText [Электронный ресурс]. URL: https://github.com/itext/itextsharp.

# **ПРИЛОЖЕНИ****Я**

## Листинг проекта, реализующего решение поставленных задач

**namespace** **ActivityTrackingClient** {

**class** **Program** {

**public**

**const** **string** jsonConfigFileName = "client.json";

**static** Settings settings;

**public** **class** **Settings** {

**public** **string** hostname;

**public** **bool** hiddenLoad;

**public** **string** path;

**public** DateTime sendTime;

**public** **Settings**(**string** path) {

**this**.path = path;

}

**public** **void** **Save**() {

File.WriteAllText(path, JObject.FromObject(**this**).ToString());

}

}

**public** **static** **void** **Main**(**string**[] args) {

**if** (File.Exists(jsonConfigFileName)) {

settings = JObject.Parse(File.ReadAllText(jsonConfigFileName)).ToObject < Settings > ();

settings.path = jsonConfigFileName;

} **else** {

settings = **new** Settings(jsonConfigFileName);

settings.Save();

}

HeadersTracking headersTracking = **new** HeadersTracking();

**while** (**true**) {

Console.ReadKey();

HeadersTrackingData trackingData = headersTracking.ExportData();

StatisticsShaper statisticsShaper = **new** StatisticsShaper(trackingData);

**string** xlsxFileName = Environment.MachineName + DateTime.Now.ToString(" dd-MM-yyyy") + ".xlsx";

statisticsShaper.SaveToFile(xlsxFileName);

StatisticSender statisticSender = **new** StatisticSender(xlsxFileName, settings.hostname);

}

}

}

}

**namespace** **ActivityTrackingClient** {

**class** **HeadersTracking** {

**public** **class** **HeadersTrackingData** {

**public** Dictionary < **string**, **int** > statistic {

**get**;

}

**public** **HeadersTrackingData**(Dictionary < **string**, **int** > statistic) {

**this**.statistic = **new** Dictionary < **string**, **int** > (statistic);

}

}

[DllImport("user32.dll")]

**static** **extern** IntPtr **GetForegroundWindow**();

[DllImport("user32.dll")]

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

**public**

**const** **int** checkStateInterval = **1**;

**private** **static** Timer timer;

**private** Dictionary < **string**, **int** > statistic = **new** Dictionary < **string**, **int** > ();

**private** **static** **string** currentHeader;

**private** **static** **int** timeOfCurrentHeader;

**private** **static** **long** workTime = **0**;

**static** **private** **string** **GetActiveWindowTitle**() {

**const** **int** nChars = **256**;

StringBuilder Buff = **new** StringBuilder(nChars);

IntPtr handle = GetForegroundWindow();

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

**return** Buff.ToString();

}

**return** **null**;

}

**public** **HeadersTracking**() {

timer = **new** Timer(**1000**);

timer.Elapsed += CheckState;

timer.Enabled = **true**;

}

**public** HeadersTrackingData **ExportData**() {

**return** **new** **HeadersTrackingData**(**this**.statistic);

}

**private** **void** **CheckState**(**object** sender, ElapsedEventArgs e) {

workTime++;

timeOfCurrentHeader++;

Console.Clear();

DisplayStatistic();

**if** (workTime % checkStateInterval == **0**) {

**string** newHeader = GetActiveWindowTitle();

**if** (currentHeader != newHeader && newHeader != **null**) {

**if** (!statistic.ContainsKey(newHeader)) {

statistic.Add(newHeader, **0**);

}

**if** (currentHeader != **null**)

statistic[currentHeader] += timeOfCurrentHeader;

currentHeader = newHeader;

timeOfCurrentHeader = **0**;

}

}

}

**public** **void** **DisplayStatistic**() {

statistic = statistic.OrderBy(obj => obj.Key).ToDictionary(obj => obj.Key, obj => obj.Value);

**foreach**(**var** item **in** statistic) {

Console.WriteLine(item.Key + " " + item.Value);

}

}

}

}

**namespace** **ActivityTrackingClient** {

**class** **StatisticSender** {

**enum** ServerRespones {

error,

successful

}

**private** **static** **int** timeOut = **5**;

**private** **static** **int** sendAttempts = **5**;

**private** **static** **int** time = **0**;

**private** **static** **bool** skipThisAttempt = **false**;

**private** TcpClient tcpclnt;

**private** NetworkStream stm;

**public** **void** **sendData**(**byte**[] data, NetworkStream stream) {

**int** bufferSize = **1024**;

**byte**[] dataLength = BitConverter.GetBytes(data.Length);

stream.Write(dataLength, **0**, **4**);

**int** bytesSent = **0**;

**int** bytesLeft = data.Length;

**while** (bytesLeft > **0**) {

**int** curDataSize = Math.Min(bufferSize, bytesLeft);

stream.Write(data, bytesSent, curDataSize);

bytesSent += curDataSize;

bytesLeft -= curDataSize;

}

}

**public** **byte**[] **getData**(TcpClient client) {

NetworkStream stream = client.GetStream();

**byte**[] fileSizeBytes = **new** **byte**[**4**];

**int** bytes = stream.Read(fileSizeBytes, **0**, **4**);

**int** dataLength = BitConverter.ToInt32(fileSizeBytes, **0**);

**int** bytesLeft = dataLength;

**byte**[] data = **new** **byte**[dataLength];

**int** bufferSize = **1024**;

**int** bytesRead = **0**;

**while** (bytesLeft > **0**) {

**int** curDataSize = Math.Min(bufferSize, bytesLeft);

**if** (client.Available < curDataSize)

curDataSize = client.Available;

bytes = stream.Read(data, bytesRead, curDataSize);

bytesRead += curDataSize;

bytesLeft -= curDataSize;

}

**return** data;

}

**public** **void** **IncTime**(Object stateInfo) {

time++;

Console.WriteLine("Time to send remaining: " + (timeOut - time));

**if** (time > timeOut) {

time = **0**;

skipThisAttempt = **true**;

**try** {

stm.Close();

tcpclnt.Close();

} **catch** (Exception) {}

}

}

**public** **StatisticSender**(**string** fileName, **string** hostname) {

**var** stateTimer = **new** Timer(IncTime, **null**, **0**, **1000**);

**int** sendAttemptsCurrent = **0**;

**bool** fileSended = **false**;

**while** (sendAttemptsCurrent < sendAttempts && !fileSended) {

**try** {

skipThisAttempt = **false**;

Console.WriteLine("Number of current send attempt : " + sendAttemptsCurrent);

sendAttemptsCurrent++;

tcpclnt = **new** TcpClient();

Console.WriteLine("Connecting ...");

tcpclnt.Connect(hostname, **8001**);

Console.WriteLine("Connected");

**byte**[] file = File.ReadAllBytes(fileName);

Console.WriteLine("Trying to send ...");

stm = tcpclnt.GetStream();

sendData(file, stm);

**if** (skipThisAttempt) {

**continue**;

}

Console.WriteLine("Waiting for reply ...");

**byte**[] serverRespone = getData(tcpclnt);

**if** ((**int**)(BitConverter.ToUInt32(serverRespone)) == (**int**) ServerRespones.successful) {

Console.WriteLine("File has been accepted by server");

fileSended = **true**;

stateTimer.Change(Timeout.Infinite, **1000**);

} **else** Console.WriteLine("Failed to send file");

tcpclnt.Close();

} **catch** (Exception e) {

Console.WriteLine("Error : " + e.Message);

}

}

}

}

}

**namespace** **ActivityTrackingClient** {

**class** **StatisticsShaper** {

XLWorkbook workbook = **new** XLWorkbook();

**public** **StatisticsShaper**(HeadersTrackingData trackingData) {

**var** worksheet = workbook.Worksheets.Add(Environment.MachineName);

List < KeyValuePair < **string**, **int** >> data = trackingData

.statistic

.ToList()

.OrderBy(item => item.Value)

.ToList();

data.RemoveAll(item => item.Value < **5**);

**int** maxLenght = **0**;

**for** (**int** i = **0**; i < data.Count; i++) {

maxLenght = (data.ElementAt(i).Key.Length > maxLenght) ? data.ElementAt(i).Value : maxLenght;

worksheet.Cell("A" + (i + **1**)).Value = data.ElementAt(i).Key;

worksheet.Cell("B" + (i + **1**)).Value = data.ElementAt(i).Value;

}

worksheet.Column("A").Width = maxLenght / **3**;

}

**public** **void** **SaveToFile**(**string** path) {

workbook.SaveAs(path);

}

}

}

**namespace** **ActivityTrackingServer** {

**public** **partial** **class** **MainForm**: Form {

**public** **class** **KeyWord** {

**public** **string** name;

**public** **int** impactIndex;

**public** **bool** bad;

**public** **KeyWord**(**string** name, **int** impactIndex, **bool** bad) {

**this**.name = name;

**this**.impactIndex = impactIndex;

**this**.bad = bad;

}

}

**public** List < **string** > acceptedFiles = **new** List < **string** > ();

**public** List < KeyWord > keyWords;

**public** Dictionary < **string**, **float** > machineStatistics = **new** Dictionary < **string**, **float** > ();

IEnumerable < Color > gradient;

**public**

**const** **string** globalDataFileName = "globaldata.xlsx";

**public**

**const** **string** statisticFileName = "Статистика.xlsx";

**public**

**const** **string** jsonConfigFileName = "settings.json";

**public** **static** Settings settings;

**public** **MainForm**() {

InitializeComponent();

}

**public** **void** **AddTaskToProcessing**(**string** fileName) {

acceptedFiles.Add(fileName);

}

**public** **void** **WriteLog**(**string** text) {

richTextBox1.Invoke(**new** Action(() => {

richTextBox1.Text += text + Environment.NewLine;

}));

}

**private** **void** **Form1\_Load**(**object** sender, EventArgs e) {

gradient = GetGradients(Color.Red, Color.FromArgb(**255**, **255**, **0**), **50**).Union(

GetGradients(Color.FromArgb(**255**, **255**, **0**), Color.FromArgb(**0**, **255**, **0**), **50**)

);

**if** (File.Exists(jsonConfigFileName)) {

settings = JObject.Parse(File.ReadAllText(jsonConfigFileName)).ToObject < Settings > ();

settings.path = jsonConfigFileName;

} **else** settings = **new** Settings(jsonConfigFileName);

keyWords = settings.keyWords;

**foreach**(**var** item **in** keyWords) {

listView1.Items.Add(**new** ListViewItem(**new** **string**[] {

item.name, item.impactIndex.ToString(), item.bad.ToString()

}));

}

Task.Factory.StartNew(() => {

StatisticsReceiver statisticsReceiver = **new** StatisticsReceiver(WriteLog, AddTaskToProcessing);

});

**var** stateTimer = **new** System.Threading.Timer(IncTime, **null**, **0**, **1000**);

}

**public** **void** **IncTime**(Object stateInfo) {

**if** (acceptedFiles.Count != **0**) {

XLWorkbook globalDataFile = **new** XLWorkbook();

**string** currentDataFileName = acceptedFiles.ElementAt(**0**);

Console.WriteLine("opening file: " + currentDataFileName);

XLWorkbook currentDataFile = **new** XLWorkbook(currentDataFileName);

**string** currentDataClientMachineName = currentDataFile.Worksheets.ElementAt(**0**).Name;

**if** (!File.Exists(globalDataFileName)) {

globalDataFile = **new** XLWorkbook();

globalDataFile.Worksheets.Add("empty");

globalDataFile.SaveAs(globalDataFileName);

} **else** {

globalDataFile = **new** XLWorkbook(globalDataFileName);

}

**if** (!globalDataFile.Worksheets.Contains(currentDataClientMachineName)) {

globalDataFile.AddWorksheet(currentDataFile.Worksheets.ElementAt(**0**));

} **else** {

Dictionary < **string**, **int** > globalDataFileTab = **new** Dictionary < **string**, **int** > ();

Dictionary < **string**, **int** > currentDataFileTab = **new** Dictionary < **string**, **int** > ();

**int** counter = **1**;

**var** worksheet = currentDataFile.Worksheet(currentDataClientMachineName);

**while** (!worksheet.Cell("A" + counter).IsEmpty()) {

Console.WriteLine("!!! " + worksheet.Cell("A" + counter).GetString() + " " +

worksheet.Cell("B" + counter).GetString());

currentDataFileTab.Add(

worksheet.Cell("A" + counter).GetString(),

worksheet.Cell("B" + counter).GetValue < **int** > ()

);

counter++;

}

worksheet = globalDataFile.Worksheet(currentDataClientMachineName);

counter = **1**;

**while** (!worksheet.Cell("A" + counter).IsEmpty()) {

globalDataFileTab.Add(

worksheet.Cell("A" + counter).GetString(),

worksheet.Cell("B" + counter).GetValue < **int** > ()

);

counter++;

}

**foreach**(**var** item **in** currentDataFileTab) {

**if** (globalDataFileTab.ContainsKey(item.Key)) {

globalDataFileTab[item.Key] += item.Value;

} **else** {

globalDataFileTab.Add(item.Key, item.Value);

}

}

List < KeyValuePair < **string**, **int** >> outPut = globalDataFileTab.OrderByDescending(item => item.Value).ToList();

**for** (**int** i = **0**; i < outPut.Count; i++) {

WriteLog("!!! " + outPut[i].Key + " " + outPut[i].Value);

worksheet.Cell("A" + (i + **1**)).Value = outPut[i].Key;

worksheet.Cell("B" + (i + **1**)).Value = outPut[i].Value;

}

}

WriteLog("Saving global file ...");

globalDataFile.SaveAs(globalDataFileName);

acceptedFiles.Remove(currentDataFileName);

**while** (acceptedFiles.Count != **0**) {

}

}

}

**private** **void** **button1\_Click**(**object** sender, EventArgs e) {

AddKeyWordDialog addKeyWordDialog = **new** AddKeyWordDialog();

DialogResult dr = addKeyWordDialog.ShowDialog(**this**);

**if** (dr == DialogResult.Cancel) {

addKeyWordDialog.Close();

} **else** **if** (dr == DialogResult.OK) {

KeyWord keyWord = addKeyWordDialog.GetKeyWord();

keyWords.RemoveAll(item => item.name == keyWord.name);

keyWords.Add(addKeyWordDialog.GetKeyWord());

listView1.Items.Add(**new** ListViewItem(**new** **string**[] {

keyWord.name, keyWord.impactIndex.ToString(), keyWord.bad.ToString()

}));

addKeyWordDialog.Close();

MainForm.settings.Save();

}

}

**private** **void** **button2\_Click**(**object** sender, EventArgs e) {

**foreach**(ListViewItem item **in** listView1.SelectedItems) {

keyWords.RemoveAll(item1 => item1.name == listView1.Items[listView1.Items.IndexOf(item)].Text);

listView1.Items.RemoveAt(listView1.Items.IndexOf(item));

MainForm.settings.Save();

}

}

**private** **void** **groupBox1\_Enter**(**object** sender, EventArgs e) {

}

**private** **void** **button3\_Click**(**object** sender, EventArgs e) {

//try

//{

XLWorkbook globalDataFile = **new** XLWorkbook(globalDataFileName);

machineStatistics.Clear();

listView2.Items.Clear();

**for** (**int** i = **1**; i < globalDataFile.Worksheets.Count; i++) {

**var** worksheet = globalDataFile.Worksheets.ElementAt(i);

**string** machineName = worksheet.Name;

**int** counter = **1**;

**int** badValue = **0**;

**int** goodValue = **0**;

**while** (!worksheet.Cell("A" + counter).IsEmpty()) {

**string** header = worksheet.Cell("A" + counter).GetString();

**int** time = worksheet.Cell("B" + counter).GetValue < **int** > ();

counter++;

**foreach**(**var** word **in** keyWords) {

**if** (header.IndexOf(word.name) != -**1**) {

Console.WriteLine(header + word.impactIndex + word.bad);

**if** (word.bad) {

badValue += word.impactIndex \* time;

} **else** {

goodValue += word.impactIndex \* time;

}

}

}

}

**float** **value**;

Console.WriteLine(worksheet + " " + badValue + " " + goodValue);

**if** (badValue + goodValue == **0**) {

**value** = **33**;

} **else**

**value** = **100** - goodValue / (**float**)(badValue + goodValue) \* **100**;

**value** = (**float**) Math.Round(**value**, **2**);

machineStatistics.Add(machineName, **value**);

Console.WriteLine("value " + **value**);

**var** item = **new** ListViewItem(machineName);

item.SubItems.Add(**value**.ToString());

item.SubItems[**1**].BackColor = gradient.ElementAt((**int**) **value** - **1**);

item.UseItemStyleForSubItems = **false**;

listView2.Items.Add(item);

}

//}

//catch (Exception error)

//{

// WriteLog("Error : " + error.Message);

//}

}

**public** **static** IEnumerable < Color > GetGradients(Color start, Color end, **int** steps) {

**int** stepA = ((end.A - start.A) / (steps - **1**));

**int** stepR = ((end.R - start.R) / (steps - **1**));

**int** stepG = ((end.G - start.G) / (steps - **1**));

**int** stepB = ((end.B - start.B) / (steps - **1**));

**for** (**int** i = **0**; i < steps; i++) {

**yield**

**return** Color.FromArgb(start.A + (stepA \* i),

start.R + (stepR \* i),

start.G + (stepG \* i),

start.B + (stepB \* i));

}

}

**private** **void** **button4\_Click**(**object** sender, EventArgs e) {

XLWorkbook statisticFile = **new** XLWorkbook(statisticFileName);

**int** counter = **1**;

**var** statisicWorksheet = statisticFile.Worksheets.ElementAt(**0**);

List < (**string** name, **int** number) > columns = **new** List < (**string** name, **int** number) > ();

**while** (!statisicWorksheet.Column(counter).Cell(**1**).IsEmpty()) {

columns.Add((statisicWorksheet.Column(counter).Cell(**1**).Value.ToString(), counter));

counter++;

}

ImportDataDialog importDataDialog = **new** ImportDataDialog(columns);

DialogResult dr = importDataDialog.ShowDialog(**this**);

**if** (dr == DialogResult.Cancel) {

importDataDialog.Close();

} **else** **if** (dr == DialogResult.OK) {

**var** result = importDataDialog.GetResult();

Console.WriteLine(

statisicWorksheet.Column(result.columnNumberOfMachineNames).Cell(**1**).Value.ToString() + " " +

statisicWorksheet.Column(result.columnNumberOfData).Cell(**1**).Value.ToString()

);

XLWorkbook globalDataFile = **new** XLWorkbook(globalDataFileName);

**foreach**(**var** globalDataFileWorksheet **in** globalDataFile.Worksheets) {

counter = **1**;

**string** machineName = globalDataFileWorksheet.Name;

**while** (!statisicWorksheet.Column(result.columnNumberOfMachineNames).Cell(counter).IsEmpty()) {

**if** (statisicWorksheet.Column(result.columnNumberOfMachineNames).Cell(counter).Value.ToString() ==

machineName) {

Console.WriteLine("Статистика по " + machineName + " может быть сопосотавлена");

statisicWorksheet.Column(result.columnNumberOfData).Cell(counter).Value = machineStatistics[machineName];

}

counter++;

}

}

}

statisticFile.SaveAs(statisticFileName);

}

}

}

**namespace** **ActivityTrackingServer** {

**public** **class** **Settings** {

**public** List < KeyWord > keyWords = **new** List < KeyWord > ();

**public** **string** path;

**public** **Settings**(**string** path) {

**this**.path = path;

}

**public** **void** **Save**() {

File.WriteAllText(path, JObject.FromObject(**this**).ToString());

}

}

}

**namespace** **ActivityTrackingServer** {

**class** **StatisticsReceiver** {

**enum** ServerRespones {

error,

successful

}

**public** **delegate** **void** **WriteLog**(**string** text);

**public** **delegate** **void** **AddTaskToProcessing**(**string** fileName);

**private** WriteLog writeLog;

**private** **static** **int** timeOut = **5**;

**private** **static** **int** time = **0**;

**private** **static** **bool** skipThisAttempt = **false**;

TcpListener listen;

TcpClient client;

**public** **byte**[] **getData**(TcpClient client) {

NetworkStream stream = client.GetStream();

**byte**[] fileSizeBytes = **new** **byte**[**4**];

**int** bytes = stream.Read(fileSizeBytes, **0**, **4**);

**int** dataLength = BitConverter.ToInt32(fileSizeBytes, **0**);

**int** bytesLeft = dataLength;

**byte**[] data = **new** **byte**[dataLength];

**int** bufferSize = **1024**;

**int** bytesRead = **0**;

**while** (bytesLeft > **0**) {

**int** curDataSize = Math.Min(bufferSize, bytesLeft);

**if** (client.Available < curDataSize)

curDataSize = client.Available;

bytes = stream.Read(data, bytesRead, curDataSize);

bytesRead += curDataSize;

bytesLeft -= curDataSize;

}

**return** data;

}

**public** **void** **sendData**(**byte**[] data, NetworkStream stream) {

**int** bufferSize = **1024**;

**byte**[] dataLength = BitConverter.GetBytes(data.Length);

stream.Write(dataLength, **0**, **4**);

**int** bytesSent = **0**;

**int** bytesLeft = data.Length;

**while** (bytesLeft > **0**) {

**int** curDataSize = Math.Min(bufferSize, bytesLeft);

stream.Write(data, bytesSent, curDataSize);

bytesSent += curDataSize;

bytesLeft -= curDataSize;

}

}

**public** **void** **IncTime**(Object stateInfo) {

time++;

Console.WriteLine("Time to receive remaining: " + (timeOut - time));

**if** (time > timeOut) {

time = **0**;

skipThisAttempt = **true**;

client.Close();

listen.Stop();

}

}

**public** **StatisticsReceiver**(WriteLog writeLog, AddTaskToProcessing addTaskToProcessing) {

**this**.writeLog = writeLog;

**var** stateTimer = **new** Timer(IncTime, **null**, **0**, **1000**);

**while** (**true**) {

**try** {

time = **0**;

skipThisAttempt = **false**;

stateTimer.Change(Timeout.Infinite, **1000**);

listen = **new** TcpListener(**8001**);

listen.Start();

writeLog("Waiting for connections...");

client = listen.AcceptTcpClient();

stateTimer.Change(**0**, **1000**);

IPEndPoint remoteIpEndPoint = client.Client.RemoteEndPoint **as** IPEndPoint;

writeLog("Client connected from: " + remoteIpEndPoint.Address);

**byte**[] data = getData(client);

**string** fileName = remoteIpEndPoint.Address + " " + DateTime.Now.ToString("MM-dd-yyyy-fff") + ".xlsx";

File.WriteAllBytes(fileName, data);

addTaskToProcessing(fileName);

writeLog("File saved to disk");

NetworkStream stream = client.GetStream();

sendData(BitConverter.GetBytes((**int**) ServerRespones.successful), stream);

writeLog("Reply has been sended");

client.Close();

listen.Stop();

} **catch** (Exception e) {

**if** (skipThisAttempt == **true**) {

writeLog("Time limit exceeded : " + e.Message);

} **else**

writeLog("Error : " + e.Message);

}

}

}

}

}

**namespace** **ActivityTrackingServer** {

**public** **class** **Settings** {

**public** List < KeyWord > keyWords = **new** List < KeyWord > ();

**public** **string** path;

**public** **Settings**(**string** path) {

**this**.path = path;

}

**public** **void** **Save**() {

File.WriteAllText(path, JObject.FromObject(**this**).ToString());

}

}

}

**namespace** **ActivityTrackingServer** {

**public** **partial** **class** **ImportDataDialog**: Form {

(**int** columnNumberOfMachineNames, **int** columnNumberOfData) result;

**public** **ImportDataDialog**(List < (**string** name, **int** number) > columns) {

InitializeComponent();

**foreach**(**var** item **in** columns) {

comboBox1.Items.Add(item);

comboBox2.Items.Add(item);

}

}

**private** **void** **button1\_Click**(**object** sender, EventArgs e) {

result = (comboBox1.SelectedIndex + **1**, comboBox2.SelectedIndex + **1**);

}

**public**(**int** columnNumberOfMachineNames, **int** columnNumberOfData) GetResult() {

**return** result;

}

**private** **void** **ImportDataDialog\_Load**(**object** sender, EventArgs e) {

}

}

}

**namespace** **ActivityTrackingServer** {

**public** **partial** **class** **AddKeyWordDialog**: Form {

KeyWord keyWord;

**public** **AddKeyWordDialog**() {

InitializeComponent();

}

**private** **void** **button1\_Click**(**object** sender, EventArgs e) {

keyWord = **new** KeyWord(textBox1.Text, **int**.Parse(textBox2.Text), good.Checked);

}

**public** KeyWord **GetKeyWord**() {

**return** keyWord;

}

}

}

**namespace** **ActivityTrackingAnalytics** {

**public** **partial** **class** **MainForm**: Form {

**public** **static** **string** globalDataFileName = "globaldata.xlsx";

**const** **string** statisticFileName = "Статистика.xlsx";

**const** **string** jsonConfigFileName = "ActivityTrackingAnalyticsConfig.json";

IEnumerable < Color > gradient;

**private** Settings settings;

**public** **class** **Settings** {

**public** List < (**string** name, **int** maxValue, **bool** toMaxIsBetter, **int** impactIndex, **int** columnIndex) > EvaluationCriterions =

**new** List < (**string** name, **int** maxValue, **bool** toMaxIsBetter, **int** impactIndex, **int** columnIndex) > ();

**public** **string** path;

**public** **Settings**(**string** path) {

**this**.path = path;

}

**public** **void** **Save**() {

File.WriteAllText(path, JObject.FromObject(**this**).ToString());

}

}

**public** List < (**string** name, **int** maxValue, **bool** toMaxIsBetter, **int** impactIndex, **int** columnIndex) > EvaluationCriterions;

**public** **MainForm**() {

InitializeComponent();

}

**private** **void** **WriteLog**(**string** text) {

richTextBox1.Text += text + Environment.NewLine;

}

**private** **void** **button1\_Click**(**object** sender, EventArgs e) {

XLWorkbook statisticFile = **new** XLWorkbook(statisticFileName);

**int** counter = **1**;

**var** statisicWorksheet = statisticFile.Worksheets.ElementAt(**0**);

List < **string** > columns = **new** List < **string** > ();

**while** (!statisicWorksheet.Column(counter).Cell(**1**).IsEmpty()) {

columns.Add(statisicWorksheet.Column(counter).Cell(**1**).Value.ToString());

counter++;

}

AddEvaluationCriterionDialog adEvaluationCriterionDialog = **new** AddEvaluationCriterionDialog(columns);

DialogResult dr = adEvaluationCriterionDialog.ShowDialog(**this**);

**if** (dr == DialogResult.Cancel) {

adEvaluationCriterionDialog.Close();

} **else** **if** (dr == DialogResult.OK) {

**var** result = adEvaluationCriterionDialog.GetResult();

EvaluationCriterions.Add(result);

settings.Save();

listView1.Items.Add(

**new** **ListViewItem**(**new** **string**[] {

result.name, result.maxValue.ToString(), result.toMaxIsBetter.ToString(), result.impactIndex.ToString(), result.columnIndex.ToString()

}));

}

}

**private** **void** **button3\_Click**(**object** sender, EventArgs e) {

listView2.Items.Clear();

richTextBox1.Clear();

XLWorkbook statisticFile = **new** XLWorkbook(statisticFileName);

**var** worksheet = statisticFile.Worksheets.ElementAt(**0**);

**int** counter = **2**;

List < (**string**, **float**, **float**) > personsStats = **new** List < (**string**, **float**, **float**) > ();

**while** (!worksheet.Cell(counter, **1**).IsEmpty()) {

**string** personeName = worksheet.Cell(counter, **1**).Value.ToString();

Console.WriteLine(personeName);

**float** goodValue = **0**;

**float** badValue = **0**;

**float** maxGoodValue = **0**;

**float** maxBadValue = **0**;

**foreach**(**var** item **in** EvaluationCriterions) {

**float** **value** = (**float**) worksheet.Cell(counter, item.columnIndex).GetDouble();

**float** evaluationValue = **value**;

**if** (evaluationValue > item.maxValue) {

evaluationValue = item.maxValue;

} **else** {

evaluationValue \*= item.impactIndex;

}

**if** (item.toMaxIsBetter) {

maxGoodValue += item.maxValue \* item.impactIndex;

goodValue += evaluationValue;

} **else** {

maxBadValue += item.maxValue \* item.impactIndex;

badValue += evaluationValue;

}

**if** (item.name == "Опоздание (мин)" &&

(evaluationValue > item.maxValue ||

(evaluationValue / item.maxValue > **0.3**)

)) {

WriteLog(personeName + " : сделать выговор из-за опоздания на " + **value** + " минут(ы).");

}

**if** (item.name == "Оценка используемого времени" &&

(evaluationValue / item.maxValue < **0.5**)

) {

WriteLog(personeName + " : плохо использует время на рабочем месте.");

}

}

goodValue = ((maxGoodValue == **0**) ? **0** : goodValue / maxGoodValue) \* **100**;

badValue = ((maxBadValue == **0**) ? **0** : badValue / maxBadValue) \* **100**;

personsStats.Add((personeName, goodValue, badValue));

**var** item\_ = **new** ListViewItem(personeName);

item\_.SubItems.Add(Math.Round(goodValue, **2**).ToString());

item\_.SubItems.Add(Math.Round(badValue, **2**).ToString());

item\_.SubItems[**1**].BackColor = gradient.ElementAt((**int**) goodValue - **1**);

item\_.SubItems[**2**].BackColor = gradient.ElementAt(gradient.Count() - (((**int**) badValue == **100**) ? **99** : (**int**) badValue + **1**));

item\_.UseItemStyleForSubItems = **false**;

listView2.Items.Add(item\_);

counter++;

}

**if** (checkBox1.Checked) {

DetailedStatistics detailedStatistics = **new** DetailedStatistics(personsStats);

detailedStatistics.Show();

}

}

**private** **void** **MainForm\_Load**(**object** sender, EventArgs e) {

WriteLog("Starting...");

gradient = GetGradients(Color.Red, Color.FromArgb(**255**, **255**, **0**), **50**).Union(

GetGradients(Color.FromArgb(**255**, **255**, **0**), Color.FromArgb(**0**, **255**, **0**), **50**)

);

**if** (File.Exists(jsonConfigFileName)) {

settings = JObject.Parse(File.ReadAllText(jsonConfigFileName)).ToObject < Settings > ();

settings.path = jsonConfigFileName;

} **else** settings = **new** Settings(jsonConfigFileName);

EvaluationCriterions = settings.EvaluationCriterions;

**if** (EvaluationCriterions != **null**)

**foreach**(**var** item **in** EvaluationCriterions) {

listView1.Items.Add(

**new** **ListViewItem**(**new** **string**[] {

item.name, item.maxValue.ToString(), item.toMaxIsBetter.ToString(), item.impactIndex.ToString()

}));

}

}

**private** **void** **button2\_Click**(**object** sender, EventArgs e) {

**foreach**(ListViewItem item **in** listView1.SelectedItems) {

EvaluationCriterions.RemoveAll(item1 => item1.name == listView1.Items[listView1.Items.IndexOf(item)].Text);

listView1.Items.RemoveAt(listView1.Items.IndexOf(item));

settings.Save();

}

}

**public** **static** IEnumerable < Color > GetGradients(Color start, Color end, **int** steps) {

**int** stepA = ((end.A - start.A) / (steps - **1**));

**int** stepR = ((end.R - start.R) / (steps - **1**));

**int** stepG = ((end.G - start.G) / (steps - **1**));

**int** stepB = ((end.B - start.B) / (steps - **1**));

**for** (**int** i = **0**; i < steps; i++) {

**yield**

**return** Color.FromArgb(start.A + (stepA \* i),

start.R + (stepR \* i),

start.G + (stepG \* i),

start.B + (stepB \* i));

}

}

}

}

**namespace** **ActivityTrackingAnalytics** {

**public** **partial** **class** **DetailedStatistics**: Form {

List < (**string**, **float**, **float**) > personsStats;

**const** **int** groupsCount = **10**;

**public** **DetailedStatistics**(List < (**string**, **float**, **float**) > personsStats) {

InitializeComponent();

**this**.personsStats = personsStats;

}

**private** **void** **chart1\_Click**(**object** sender, EventArgs e) {

}

**private** **void** **button1\_Click**(**object** sender, EventArgs e) {}

**private** **void** **DetailedStatistics\_Load**(**object** sender, EventArgs e) {

#region MainStats

List < (**int** min, **int** max, **int** count) > stats = **new** List < (**int** min, **int** max, **int** count) > ();

**for** (**int** i = **0**; i < groupsCount; i++) {

stats.Add((i \* **100** / groupsCount, (i + **1**) \* **100** / groupsCount, **0**));

Console.WriteLine(stats[i]);

}

**foreach**(**var** item **in** personsStats) {

**for** (**int** i = **0**; i < stats.Count; i++) {

**if** (item.Item2 > stats[i].min && item.Item2 <= stats[i].max) {

stats[i] = (stats[i].min, stats[i].max, stats[i].count + **1**);

}

}

}

**foreach**(**var** item **in** stats) {

**if** (item.count != **0**)

chart2.Series["Series1"].Points.AddXY((item.min + "-" + item.max), item.count);

}

stats = **new** List < (**int** min, **int** max, **int** count) > ();

**for** (**int** i = **0**; i < groupsCount; i++) {

//stats.Add(((int min, int max, int count)) new ValueTuple<int, int, int>(21, 21, 21));

stats.Add((i \* **100** / groupsCount, (i + **1**) \* **100** / groupsCount, **0**));

Console.WriteLine(stats[i].min + " " + stats[i].max);

}

chart4.Series["Series1"].IsVisibleInLegend = **false**;

**foreach**(**var** item **in** personsStats) {

chart4.Series["Series1"].Points.AddXY(Math.Round(item.Item2), Math.Round(item.Item3));

**for** (**int** i = **0**; i < stats.Count; i++) {

**if** (item.Item3 > stats[i].min && item.Item3 <= stats[i].max) {

stats[i] = (stats[i].min, stats[i].max, stats[i].count + **1**);

}

}

}

**foreach**(**var** item **in** stats) {

**if** (item.count != **0**)

chart1.Series["Series1"].Points.AddXY((item.min + "-" + item.max), item.count);

}

#endregion

**const** **int** maxItems = **4**;

XLWorkbook globalDataFile = **new** XLWorkbook(MainForm.globalDataFileName);

Dictionary < **string**, **int** > keyWords = **new** Dictionary < **string**, **int** > ();

**for** (**int** i = **1**; i < globalDataFile.Worksheets.Count; i++) {

**var** worksheet = globalDataFile.Worksheets.ElementAt(i);

**int** counter = **1**;

**while** (!worksheet.Cell("A" + counter).IsEmpty()) {

**string** header = worksheet.Cell("A" + counter).GetString();

**int** time = worksheet.Cell("B" + counter).GetValue < **int** > ();

counter++;

**if** (keyWords.ContainsKey(header)) {

keyWords[header] += time;

} **else**

keyWords.Add(header, **0**);

}

}

//Console.WriteLine(keyWords.Count);

**var** tmp = keyWords.ToList().OrderBy(item => -item.Value).ToList();

**for** (**int** i = **0**; i <

((maxItems < keyWords.Count) ? maxItems : keyWords.Count); i++) {

//Console.WriteLine(tmp[i].Key+" "+ tmp[i].Value);

chart3.Series["Series1"].Points.AddXY(tmp[i].Key, tmp[i].Value);

}

**if** (maxItems < keyWords.Count) {

**int** count = **0**;

**for** (**int** i = maxItems; i < keyWords.Count; i++) {

count += tmp[i].Value;

}

chart3.Series["Series1"].Points.AddXY("Другие", count);

}

chart3.Series["Series1"]["PieLabelStyle"] = "Disabled";

}

**private** **string** **SaveToPfd**() {

**try** {

iTextSharp.text.Document doc = **new** iTextSharp.text.Document();

**string** path = DateTime.Now.ToString("MM dd yy H-mm-ss") + ".pdf";

PdfWriter.GetInstance(doc, **new** FileStream(path, FileMode.Create));

doc.Open();

BaseFont baseFont = BaseFont.CreateFont(@ "C:\Windows\Fonts\arial.ttf", BaseFont.IDENTITY\_H, BaseFont.NOT\_EMBEDDED);

iTextSharp.text.Font font = **new** iTextSharp.text.Font(baseFont, iTextSharp.text.Font.DEFAULTSIZE, iTextSharp.text.Font.NORMAL);

**var** text = **new** iTextSharp.text.Paragraph("Статистика мониторинга", font);

text.Alignment = Element.ALIGN\_CENTER;

doc.Add(text);

Bitmap bmp = **new** Bitmap(panel1.Width, panel1.Height);

panel1.DrawToBitmap(bmp, **new** System.Drawing.Rectangle(**0**, **0**, panel1.Width, panel1.Height));

iTextSharp.text.Image pic = iTextSharp.text.Image.GetInstance(bmp, System.Drawing.Imaging.ImageFormat.Jpeg);

**if** (pic.Height > pic.Width) {

**float** percentage = **0.0** f;

percentage = **700** / pic.Height;

pic.ScalePercent(percentage \* **100**);

} **else** {

**float** percentage = **0.0** f;

percentage = **540** / pic.Width;

pic.ScalePercent(percentage \* **100**);

}

doc.Add(pic);

doc.Close();

**return** path;

} **catch** (Exception) {

**return** **null**;

}

}

**private** **void** **button1\_Click\_1**(**object** sender, EventArgs e) {

**string** path = SaveToPfd();

**if** (path != **null**) {

MessageBox.Show("Документ с именем " + path + " создан.");

}

}

**private** **void** **button2\_Click**(**object** sender, EventArgs e) {

**string** path = SaveToPfd();

**if** (path != **null**) {

**var** file = File.ReadAllBytes(path);

**var** printQueue = LocalPrintServer.GetDefaultPrintQueue();

using(**var** job = printQueue.AddJob())

using(**var** stream = job.JobStream) {

stream.Write(file, **0**, file.Length);

}

}

}

}

}

**namespace** **ActivityTrackingAnalytics** {

**public** **partial** **class** **AddEvaluationCriterionDialog**: Form {

(**string** name, **int** maxValue, **bool** toMaxIsBetter, **int** impactIndex, **int** columnIndex) result;

**public** **AddEvaluationCriterionDialog**(List < **string** > columns) {

InitializeComponent();

**foreach**(**var** item **in** columns) {

comboBox1.Items.Add(item);

}

}

**private** **void** **button1\_Click**(**object** sender, EventArgs e) {

result = (comboBox1.Text, **int**.Parse(textBox1.Text), radioButton1.Checked, **int**.Parse(textBox2.Text), comboBox1.SelectedIndex + **1**);

}

**public**(**string** name, **int** maxValue, **bool** toMaxIsBetter, **int** impactIndex, **int** columnIndex) GetResult() {

**return** result;

}

**private** **void** **AddEvaluationCriterionDialog\_Load**(**object** sender, EventArgs e) {

}

}

}

## Скриншоты графического интерфейса пользователя



