**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

по дисциплине «**Интернет технологии и распределенная обработка данных**»

на тему: «**Проверочное задание на тему: Расчет средней успеваемости студентов**»

Выполнили: студент гр. ИП-42

Саприко Артём

Принял: ст. преподаватель

Кухаренко А.А.

Гомель 2019

Лабораторная работа №1

Цель работы: Задание предназначено для проверки уровня владения навыками разработки приложений среди студентов, чтобы можно было оценить состояние текущей подготовки, уровень написанного исходного кода, приемы и практики, паттерны, которые применяются и т.п.

Мною было разработано консольное приложение на языке С#, платформа .NET Core 2.x.

Приложение состоит из двух модулей Lab1 и Lab1.CLI.

В Lab1 были подключены библиотеки CsvHelper, OfficeOpenXml, Newtonsoft.Json.

В Lab1.CLI библиотека CommandLine.

**Листинг программы**

**Классы и интерфейсы модуля Lab1**

namespace Lab1.config

{

public struct Config

{

public string InputFileName { get; set; }

public string OutputFileName { get; set; }

public FileType OutputFileType { get; set; }

}

}

namespace Lab1.config

{

public enum FileType

{

Excel,

Json

}

}

namespace Lab1.config

{

public interface IConfigurationProvider

{

Config Config { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using CsvHelper;

using Lab1.config;

namespace Lab1.processor.reader

{

public class CsvStudentsReader : IReader<Student>

{

private readonly Config \_config;

public CsvStudentsReader(IConfigurationProvider configurationProvider)

{

\_config = configurationProvider.Config;

}

public ICollection<Student> ReadStudents()

{

if (!File.Exists(\_config.InputFileName)) throw new FileNotFoundException("Input file does not exist");

if (Path.GetExtension(\_config.InputFileName) != Constants.Csv)

throw new FormatException($"Wrong input file format. Expected {Constants.Csv}");

var records = new List<Student>();

using (var reader = new StreamReader(\_config.InputFileName))

using (var csv = new CsvReader(reader))

{

csv.Read();

csv.ReadHeader();

var lessons = new List<string>(csv.Context.HeaderRecord)

.Where(h => h != Constants.FirstName && h != Constants.LastName && h != Constants.MiddleName)

.ToList();

while (csv.Read())

{

var student = new Student

{

FirstName = csv.GetField(Constants.FirstName),

LastName = csv.GetField(Constants.LastName),

MiddleName = csv.GetField(Constants.MiddleName)

};

foreach (var lesson in lessons)

{

int.TryParse(csv.GetField(lesson), out var mark);

student.LessonMarks.Add(lesson, mark);

}

records.Add(student);

}

}

return records;

}

}

}

using System.Collections.Generic;

namespace Lab1.processor.reader

{

public interface IReader<T>

{

ICollection<T> ReadStudents();

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

using OfficeOpenXml;

using Lab1.config;

namespace Lab1.processor.writer

{

public class ExcelWriter : IWriter<Report>

{

private readonly Config \_config;

public ExcelWriter(IConfigurationProvider configurationProvider)

{

\_config = configurationProvider.Config;

}

public void Write(Report report)

{

if (!File.Exists(\_config.OutputFileName)) throw new FileNotFoundException("Output file does not exist");

if (Path.GetExtension(\_config.OutputFileName) != Constants.Excel)

throw new FormatException($"Wrong output file format. Expected {Constants.Excel}");

var table = new DataTable();

table.Columns.Add(new DataColumn(Constants.FirstName, typeof(string)));

table.Columns.Add(new DataColumn(Constants.LastName, typeof(string)));

table.Columns.Add(new DataColumn(Constants.MiddleName, typeof(string)));

var lessonIds = new Dictionary<string, int>();

var curId = 3;

foreach (var lesson in report.MeanSubjectMarks.Keys)

{

lessonIds.Add(lesson, curId++);

table.Columns.Add(new DataColumn(lesson, typeof(double)));

}

table.Columns.Add(new DataColumn(Constants.Average, typeof(double)));

foreach (var studentReport in report.StudentReports)

{

var row = table.NewRow();

row[0] = studentReport.Student.FirstName;

row[1] = studentReport.Student.LastName;

row[2] = studentReport.Student.MiddleName;

var cur = 3;

foreach (var mark in studentReport.Student.LessonMarks) row[cur++] = mark.Value;

row[cur] = studentReport.MeanSubjectsMark;

table.Rows.Add(row);

}

var totalRow = table.NewRow();

foreach (var (lesson, mark) in report.MeanSubjectMarks) totalRow[lessonIds[lesson]] = mark;

table.Rows.Add(totalRow);

SaveExcel(table);

}

private void SaveExcel(DataTable table)

{

var fi = new FileInfo(\_config.OutputFileName);

if (fi.Exists) fi.Delete();

using (var pck = new ExcelPackage(fi))

{

var worksheet = pck.Workbook.Worksheets.Add("Students");

worksheet.Cells.LoadFromDataTable(table, true);

pck.Save();

}

}

}

}

namespace Lab1.processor.writer

{

public interface IWriter<in T>

{

void Write(T report);

}

}

using System;

using Lab1.config;

namespace Lab1.processor.writer

{

public interface IWriterResolver

{

void RegisterWriter<T>(FileType fileType, IWriter<T> writer);

IWriter<T> ResolveWriter<T>(FileType fileType);

}

}

using System;

using System.IO;

using Newtonsoft.Json;

using Lab1.config;

namespace Lab1.processor.writer

{

public class JsonWriter : IWriter<Report>

{

private readonly Config \_config;

public JsonWriter(IConfigurationProvider configurationProvider)

{

\_config = configurationProvider.Config;

}

public void Write(Report report)

{

if (!File.Exists(\_config.OutputFileName)) throw new FileNotFoundException("Output file does not exist");

if (Path.GetExtension(\_config.OutputFileName) != Constants.Json)

throw new FormatException($"Wrong output file format. Expected {Constants.Json}");

var json = JsonConvert.SerializeObject(report);

using (var sw = new StreamWriter(\_config.OutputFileName))

{

sw.WriteLine(json);

}

}

}

}

using System.Collections.Generic;

using Lab1.config;

namespace Lab1.processor.writer

{

public class ReportWriterResolver : IWriterResolver

{

private readonly IDictionary<FileType, IWriter<Report>> \_writers = new Dictionary<FileType, IWriter<Report>>();

void IWriterResolver.RegisterWriter<T>(FileType fileType, IWriter<T> writer)

{

if (!(writer is IWriter<Report> reportWriter)) return;

\_writers.Add(fileType, reportWriter);

}

public IWriter<T> ResolveWriter<T>(FileType fileType)

{

if (typeof(T) != typeof(Report)) return null;

if (!\_writers.ContainsKey(fileType)) return null;

return \_writers[fileType] as IWriter<T>;

}

}

}

namespace Lab1.processor

{

public static class Constants

{

public const string Csv = ".csv";

public const string Json = ".json";

public const string Excel = ".xlsx";

public const string FirstName = "Имя";

public const string LastName = "Фамилия";

public const string MiddleName = "Отчество";

public const string Average = "Средняя отметка";

}

}

using System.Collections.Generic;

namespace Lab1.processor

{

public interface IProcessor

{

Report BuildReport(ICollection<Student> students);

}

}

using System.Collections.Generic;

using System.Linq;

namespace Lab1.processor

{

public class Processor : IProcessor

{

public Report BuildReport(ICollection<Student> students)

{

var report = new Report

{

StudentReports = students.Select(s => new StudentReport

{ Student = s, MeanSubjectsMark = s.LessonMarks.Values.Average() }).ToList()

};

var lessons = new Dictionary<string, ICollection<int>>();

foreach (var student in students)

foreach (var (lesson, mark) in student.LessonMarks)

{

if (!lessons.ContainsKey(lesson)) lessons.Add(lesson, new List<int>());

lessons[lesson].Add(mark);

}

report.MeanSubjectMarks = lessons.ToDictionary(p => p.Key, p => p.Value.Average());

return report;

}

}

}

using System.Collections.Generic;

namespace Lab1.processor

{

public class Report

{

public ICollection<StudentReport> StudentReports { get; set; }

public IDictionary<string, double> MeanSubjectMarks { get; set; } = new Dictionary<string, double>();

}

public class StudentReport

{

public Student Student { get; set; }

public double MeanSubjectsMark { get; set; }

}

}

using System.Collections.Generic;

using System.Text;

namespace Lab1.processor

{

public class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string MiddleName { get; set; }

public IDictionary<string, int> LessonMarks { get; set; } = new Dictionary<string, int>();

}

}

**Классы и интерфейсы модуля Lab1.CLI**

using CommandLine;

using Lab1.config;

namespace Lab1.CLI

{

public class ConfigOptions

{

[Option('i', "input", Required = true, HelpText = "Input file path (D:\\input.csv)")]

public string InputFileName { get; set; }

[Option('o', "output", Required = true, HelpText = "Output file path (D:\\output.json)")]

public string OutputFileName { get; set; }

[Option('t', "type", Required = false, Default = FileType.Excel, HelpText = "Output file type (Excel|Json)")]

public FileType OutputFileType { get; set; }

}

}

using Lab1.config;

namespace Lab1.CLI

{

public class ConsoleConfigurationProvider : IConfigurationProvider

{

public Config Config { get; set; }

}

}

using System;

using System.IO;

using CommandLine;

using Lab1.config;

using Lab1.processor;

using Lab1.processor.reader;

using Lab1.processor.writer;

namespace Lab1.CLI

{

public class Program

{

private static readonly IWriterResolver WriterResolver = new ReportWriterResolver();

public static void Main(string[] args)

{

Parser.Default.ParseArguments<ConfigOptions>(args)

.WithParsed(RunProcessor);

}

private static void RegisterWriters(IConfigurationProvider configurationProvider)

{

WriterResolver.RegisterWriter(FileType.Excel, new ExcelWriter(configurationProvider));

WriterResolver.RegisterWriter(FileType.Json, new JsonWriter(configurationProvider));

}

private static void RunProcessor(ConfigOptions opts)

{

var config = new Config

{

InputFileName = opts.InputFileName,

OutputFileName = opts.OutputFileName,

OutputFileType = opts.OutputFileType

};

var configurationProvider = new ConsoleConfigurationProvider { Config = config };

RegisterWriters(configurationProvider);

var processor = new Processor();

var reader = new CsvStudentsReader(configurationProvider);

try

{

var students = reader.ReadStudents();

var report = processor.BuildReport(students);

var writer = WriterResolver.ResolveWriter<Report>(config.OutputFileType);

writer.Write(report);

}

catch (FileNotFoundException e)

{

Console.WriteLine($"File not found!!! {e.Message}");

}

catch (FormatException e)

{

Console.WriteLine($"Wrong file format!!! {e.Message}");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

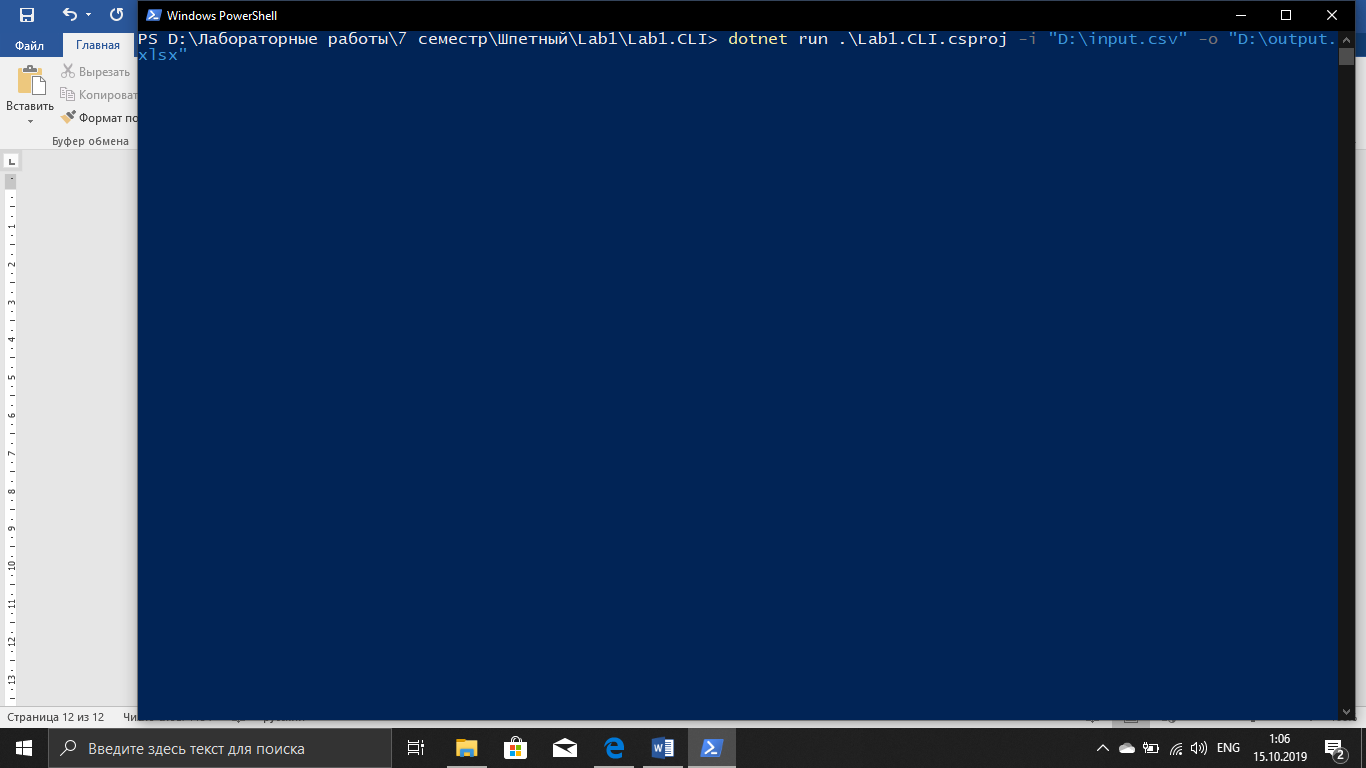
}

}

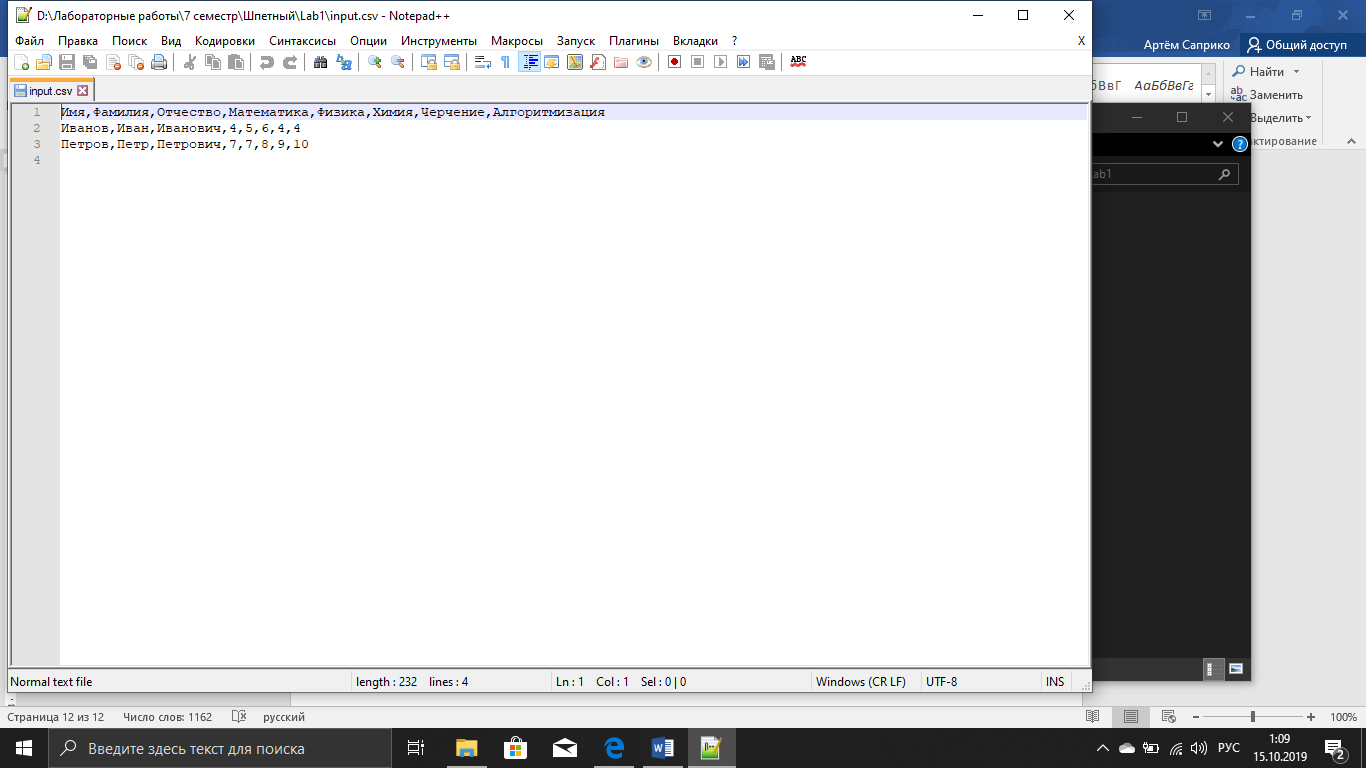
}

}

Работа осуществляется с помощью параметров консольно (Power Shell).

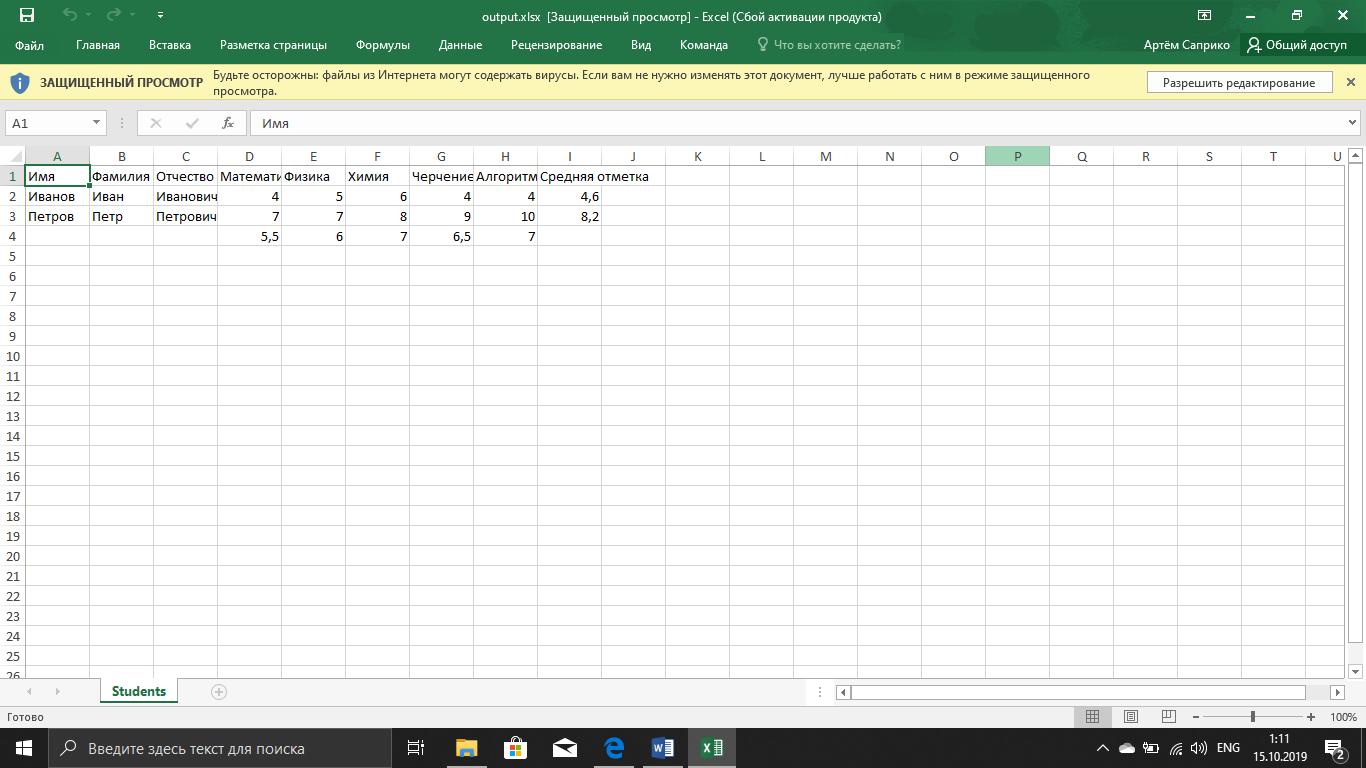


Входной csv-файл:



Выходные файлы (json и xlsx):

{"StudentReports":[{"Student":{"FirstName":"Иванов","LastName":"Иван","MiddleName":"Иванович","LessonMarks":{"Математика":4,"Физика":5,"Химия":6,"Черчение":4,"Алгоритмизация":4}},"MeanSubjectsMark":4.6},{"Student":{"FirstName":"Петров","LastName":"Петр","MiddleName":"Петрович","LessonMarks":{"Математика":7,"Физика":7,"Химия":8,"Черчение":9,"Алгоритмизация":10}},"MeanSubjectsMark":8.2}],"MeanSubjectMarks":{"Математика":5.5,"Физика":6.0,"Химия":7.0,"Черчение":6.5,"Алгоритмизация":7.0}}



Вывод: Вся поставленная работа была выполнена и протестирована.