МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ   
УНИВЕРСИТЕТ»

Факультет информационных систем и технологий

Кафедра Информационные системы

Отчет по практике

защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка прописью)

Руководитель   
практики от   
университета \_\_ст.преподаватель Эгов Е.Н.\_\_

(должность)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (ФИО)

**ОТЧЕТ по** \_\_\_**УЧЕБНОЙ** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ практике

(вид практики)

Студента \_\_Эмирян Владимира Леонидовича\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО)

Направление (специальность,

профиль) подготовки 09.03.04 Программная инженерия

Группа \_\_ПИбд-11\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Место прохождения практики УлГТУ, кафедра Информационные системы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование профильной организации, подразделение университета)

Ульяновск 2020 г.

Оглавление

1F. Красивые часы - 1 3

1G. Красивые часы - 2 4

1H. Выбирайте типы данных с умом 5

1I. Округление 6

3I. Второй максимум последовательности 7

3J. Дисперсия последовательности 8

4F. Проверка на простоту - 1 10

4G. Проверка на простоту – 2. 11

4H. Перебор перестановок 12

5G. Сортировка выбором 17

5H. Наилучший участок 19

5I. Умножение матриц 21

5J. Сапёр 23

6A. Символ в код 28

6B. Количество букв 29

6C. Древний шифр 30

6D. Макс и перестановочный шифр 31

6E. Постиранный пароль 33

6F. C++ и Java 35

6G. Слова 37

# 1F. Красивые часы - 1

**Задание:**

Дано время, отображаемое электронными часами. Определите общее число минут, прошедших с полуночи.

**Входные данные:**

Ввод содержит целые числа H и M (00 ≤ H ≤ 23, 00 ≤ M ≤ 59), разделённые двоеточием — соответственно количество часов и минут, отображаемое часами. Числа H и M записаны с использованием двух десятичных цифр каждое (возможно, с ведущими нулями).

**Выходные данные:**

Выведите одно целое число — количество минут, прошедших с полуночи.

**Код:**

using System;  
  
public class Test  
{  
  public static void Main()  
  {  
    System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
      
      string str = Console.ReadLine();  
      char[] time = new char[5];  
      time = str.ToCharArray();  
      int hrs = 0;  
      int min = 0;  
      int res = 0;  
        
      str = "";  
      str = str + time[0] + time[1];  
      hrs = Convert.ToInt32(str);  
      str = "";  
      str = str + time[3] + time[4];  
      min = Convert.ToInt32(str);  
      res = hrs \* 60 + min;  
      Console.WriteLine(res);  
  }  
}

# 1G. Красивые часы - 2

**Задание:**

Дано общее число минут, прошедших с полуночи. Определите время, отображаемое электронными часами.

**Входные данные:**

Ввод содержит целое число M (0 ≤ M ≤ 1439) — количество минут, прошедших с полуночи.

**Выходные данные:**

Выведите два целых числа, разделённые двоеточием, — количество часов и минут, отображаемое часами. Числа должны быть записаны с использованием двух десятичных цифр каждое (возможно, с ведущими нулями).

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int time = Convert.ToInt32(Console.ReadLine());  
        int hrs = 0;  
        int min = 0;  
  
        hrs = time / 60;  
        min = time - (hrs \* 60);  
  
        string n1 = Convert.ToString(hrs);  
        string n2 = Convert.ToString(min);  
  
        if(hrs >= 0 && hrs < 10)  
    {  
            n1 = "0" + n1;  
    }  
  
        if (min >= 0 && min < 10)  
        {  
            n2 = "0" + n2;  
        }  
  
        Console.WriteLine("{0}:{1}", n1, n2);  
    }  
}

# 1H. Выбирайте типы данных с умом

**Задание:**

Вычислите произведение двух целых чисел.

**Входные данные:**

Ввод содержит целые числа A и B ( - 106 ≤ A, B ≤ 106).

**Выходные данные:**

Выведите значение A × B.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string time = Console.ReadLine();  
        string n1 = "";  
        string n2 = "";  
        int len = time.Length;  
        long a;  
        long b;  
  
        char[] ch = new char[len];  
  
        ch = time.ToCharArray();  
  
        int i = 0;  
        while(ch[i] != 32)  
    {  
            n1 += ch[i];  
            i++;  
    }  
  
        i++;  
        while (i != len)  
        {  
            n2 += ch[i];  
            i++;  
        }  
  
        a = Convert.ToInt64(n1);  
        b = Convert.ToInt64(n2);  
        Console.WriteLine(a \* b);  
    }  
}

# 1I. Округление

**Задание:**

Даны целые числа A и B. Выведите их частное:

· с округлением до целого;

· с округлением до 2 знаков после запятой;

· с округлением до 6 знаков после запятой;

· с округлением до 10 знаков после запятой.

**Входные данные:**

Ввод содержит целые числа A и B (1 ≤ A, B ≤ 1000).

**Выходные данные:**

Выведите в отдельных строках одно целое и три вещественных числа — значения A/B с необходимым округлением.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string time = Console.ReadLine();  
        string n1 = "";  
        string n2 = "";  
        int len = time.Length;  
        double a;  
        double b;  
  
        char[] ch = new char[len];  
  
        ch = time.ToCharArray();  
  
        int i = 0;  
        while(ch[i] != 32)  
    {  
            n1 += ch[i];  
            i++;  
    }  
  
        i++;  
        while (i != len)  
        {  
            n2 += ch[i];  
            i++;  
        }  
  
        a = Convert.ToDouble(n1);  
        b = Convert.ToDouble(n2);  
  
        double res = a / b;  
        Console.WriteLine(Math.Round(res));  
        Console.WriteLine(String.Format("{0:N2}", Math.Round(res, 2, MidpointRounding.AwayFromZero)));  
        Console.WriteLine(String.Format("{0:N6}", Math.Round(res, 6, MidpointRounding.AwayFromZero)));  
        Console.WriteLine(String.Format("{0:N10}", Math.Round(res, 10, MidpointRounding.AwayFromZero)));  
    }  
}

# 3I. Второй максимум последовательности

**Задание:**

Дана последовательность положительных целых чисел, завершающаяся числом 0. Определите второй по величине этой последовательности.

При решении этой задачи нельзя использовать массивы.

**Входные данные:**

Ввод содержит два или более целых чисел Ai (1 ≤ Ai ≤ 2·10^6) — элементы последовательности. После всех чисел Ai следует число 0.

Общее количество чисел Ai во входных данных не превосходит 2·106.

**Выходные данные:**

Выведите одно целое число — второй максимум последовательности.

**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if (fnum.Length == 0 && c == '0')  
                return 0;  
            if (c != ' ')  
                fnum += c;  
        } while (c != ' ');  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int a = -1;  
        int max1 = 0;  
        int max2 = 0;  
  
        while(a != 0)  
    {  
            a = readNum();  
            if (a >= max1)  
            {  
                max2 = max1;  
                max1 = a;  
            }  
            else if (a >= max2)  
            {  
                max2 = a;  
            }  
        }  
  
        Console.WriteLine(max2);  
    }  
}

# 3J. Дисперсия последовательности

**Задание:**

Дана последовательность положительных целых чисел, завершающаяся числом 0. Определите дисперсию этой последовательности. Чтобы избежать погрешностей вещественных чисел, **при вычислениях округляйте *A* и *D* до меньших целых**. При решении этой задачи нельзя использовать массивы.  
**Входные данные:**  
Ввод содержит два или более целых чисел *Ai* (1 ≤ *Ai* ≤ 2·106) — элементы последовательности. После всех чисел *Ai* следует число 0. Общее количество чисел *Ai* во входных данных не превосходит 2·106.  
**Выходные данные:**  
Выведите одно целое число — дисперсию последовательности.  
**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if (fnum.Length == 0 && c == '0')  
                return 0;  
            if (c != ' ')  
                fnum += c;  
        } while (c != ' ');  
        return int.Parse(fnum);  
    }  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int a, n = 0;  
        long sa = 0, sk = 0;  
        a = readNum();  
        while (a != 0)  
        {  
            sa += (long)a;  
            sk += (long)a\* a;  
            n++;  
            a = readNum();  
        }  
        Console.WriteLine ((long)(n \* (long)(sa / n) \* (long)(sa / n) + sk - 2 \* (long)(sa / n) \* sa) / (n - 1));  
    }  
}

# 4F. Проверка на простоту - 1

**Задание**:

Дано натуральное число N. Проверьте, является ли оно простым.

Число называется простым, если оно имеет только 2 различных делителя.

**Входные данные:**

Ввод содержит целое число N (1 ≤ N ≤ 10^6).

**Выходные данные:**

Если N является простым, выведите YES, иначе выведите NO.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int a = Convert.ToInt32(Console.ReadLine());  
        string str = "YES";  
  
        if(a == 1)  
    {  
            str = "NO";  
    }  
      
        for(int i = a - 1; i > 1; i--)  
    {  
            if(a % i == 0)  
      {  
                str = "NO";  
      }  
    }  
      
  
        Console.WriteLine(str);  
    }  
}

# 4G. Проверка на простоту – 2.

**Задание:**

Дано натуральное число N. Проверьте, является ли оно простым.

Число называется простым, если оно имеет только 2 различных делителя.

Для оптимизации проверки воспользуйтесь тем фактом, что если N = A \* B, то min(A,B)  ≤  √N.

**Входные данные:**

Ввод содержит целое число N (1 ≤ N ≤ 10^12).

**Выходные данные:**

Если N является простым, выведите YES, иначе выведите NO.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        long a = Convert.ToInt64(Console.ReadLine());  
        string str = "YES";  
  
        if(a == 1)  
    {  
            str = "NO";  
    }  
  
        long sqr = (long)Math.Sqrt(a);  
  
        for(long i = sqr; i > 1; i--)  
    {  
            if(a % i == 0)  
      {  
                str = "NO";  
      }  
    }  
      
  
        Console.WriteLine(str);  
    }

# 5F. Наиболее частый элемент – 2

**Задание:**

Дан массив, состоящий из N целых чисел, упорядоченных по неубыванию. Определите, какое число встречается в массиве чаще всего.

**Входные данные:**

Первая строка содержит целое число N (1 ≤ N ≤ 10^5) — размер массива.

Вторая строка содержит N целых чисел Ai (1 ≤ Ai ≤ 109, Ai - 1 ≤ Ai ≤ Ai + 1) — элементы массива.

**Выходные данные:**

Выведите одно целое число — наиболее часто встречающийся элемент массива. Если подходящих ответов несколько, выведите максимальный из них.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            /\*if (fnum.Length == 0 && c == '0')  
                return 0;\*/  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, k = 1, kmax = 0, nextNumber = 0, temp = 0;  
        int n = Convert.ToInt32(Console.ReadLine());  
  
        number = readNum();  
        for (int i = 0; i < n - 1; i++)  
        {  
            nextNumber = readNum();  
            if (number == nextNumber)  
            {  
                k++;  
            }  
            else if (k >= kmax)  
            {  
                kmax = k;  
                temp = number;  
                k = 1;  
            }  
            else  
      {  
                k = 1;  
            }  
            number = nextNumber;  
        }  
        if (k >= kmax)  
        {  
            kmax = k;  
            temp = number;  
        }  
  
        Console.WriteLine(temp);  
    }  
}

# 5G. Сортировка выбором

**Задание:**

Дан массив A, состоящий из N целых чисел. Отсортируйте его по не убыванию при помощи сортировки выбором:

· Определите минимальный из элементов A0, ..., AN - 1 и обменяйте его с A0;

· Определите минимальный из элементов A1, ..., AN - 1 и обменяйте его с A1;

· ...

· Определите минимальный из элементов AN - 2, AN - 1 и обменяйте его с AN - 2

**Входные данные:**

Первая строка содержит целое число N (1 ≤ N ≤ 10^4) — размер массива.

Вторая строка содержит N целых чисел Ai ( - 10^9 ≤ Ai ≤ 10^9) — элементы массива.

**Выходные данные:**

Выведите N целых чисел — элементы массива в порядке не убывания.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, temp = 0;  
        int n = Convert.ToInt32(Console.ReadLine());  
        int[] array = new int[n];  
  
        for(int i = 0; i < n; i++)  
    {  
            number = readNum();  
            array[i] = number;  
    }  
  
        int k = 0;  
        for (int i = 0; i < n; i++)  
        {  
            int min = array[k];  
            int kmin = k;  
            for(int j = k + 1; j < n; j++)  
      {  
                if (array[j] < min)  
                {  
                    min = array[j];  
                    kmin = j;  
                }  
            }  
            temp = array[k];  
            array[k] = array[kmin];  
            array[kmin] = temp;  
            k++;  
        }  
  
        for (int i = 0; i < n; i++)  
        {  
            Console.Write(array[i] + " ");  
        }  
    }  
}

# 5H. Наилучший участок

**Задание:**

Дан целочисленный двумерный массив размера N × M. Строки массива нумеруются с единицы сверху вниз, столбцы — с единицы слева направо. Определите в данном массиве участок 3 × 3, сумма элементов которого является максимальной, и выведите координаты левого верхнего угла этого участка.

**Входные данные:**

Первая строка содержит целые числа N и M (3 ≤ N, M ≤ 100) — количество строк и столбцов массива соответственно.

Следующие N строк описывают массив. Каждая из них содержит M целых чисел Aij ( - 1000 ≤ Aij ≤ 1000) — элементы массива.

**Выходные данные:**

Выведите два целых числа R и C — соответственно номер строки и номер столбца левого верхнего элемента наилучшего участка. Если подходящих ответов несколько, выведите ответ с наименьшим значением R; если при этом подходящих ответов всё ещё несколько, выведите ответ с наименьшим значением C.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, temp = 0;  
        int n1 = readNum();  
        int m1 = readNum();  
        int[,] array1 = new int[n1, m1];  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m1; j++)  
            {  
                array1[i, j] = readNum();  
            }  
        }  
  
        int maxSum = -10000;  
        int iTemp = 0;  
        int jTemp = 0;  
        for (int i = 0; i < n1 - 2; i++)  
        {  
            for (int j = 0; j < m1 - 2; j++)  
            {  
                int sum = array1[i, j] + array1[i + 1, j] + array1[i + 2, j] + array1[i, j + 1] + array1[i + 1, j + 1] + array1[i + 2, j + 1] + array1[i, j + 2] + array1[i + 1, j + 2] + array1[i + 2, j + 2];  
                if (sum > maxSum)  
                {  
                    iTemp = i + 1;  
                    jTemp = j + 1;  
                    maxSum = sum;  
                }  
            }  
        }  
  
        Console.WriteLine(iTemp + " " + jTemp);  
    }  
}

# 5I. Умножение матриц

**Задание:**

Даны целочисленные матрицы A и B. Определите их произведение A × B.

**Входные данные:**

Первая строка содержит целые числа NA и MA (1 ≤ NA, MA ≤ 100) — размеры матрицы A.

Следующие NA строк описывают матрицу A. Каждая их них содержит MA целых чисел Aij ( - 100 ≤ Aij ≤ 100) — элементы матрицы.

Следующая строка содержит целые числа NB и MB (1 ≤ NB, MB ≤ 100, NB = MA) — размеры матрицы B.

Следующие NB строк описывают матрицу B. Каждая их них содержит MB целых чисел Bij ( - 100 ≤ Bij ≤ 100) — элементы матрицы.

**Выходные данные:**

Выведите NA строк, каждая из которых содержит MB целых чисел — элементы результирующей матрицы.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, temp = 0;  
        int n1 = readNum();  
        int m1 = readNum();  
        int[,] array1 = new int[n1, m1];  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m1; j++)  
            {  
                array1[i, j] = readNum();  
            }  
        }  
  
        int n2 = readNum();  
        int m2 = readNum();  
        int[,] array2 = new int[n2, m2];  
  
        for (int i = 0; i < n2; i++)  
        {  
            for (int j = 0; j < m2; j++)  
            {  
                array2[i, j] = readNum();  
            }  
        }  
  
    int[,] array3 = new int[101, 101];  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m2; j++)  
            {  
                for (int k = 0; k < m1; k++)  
                {  
                    array3[i, j] += array1[i, k] \* array2[k, j];  
                }  
            }  
        }  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m2; j++)  
            {  
                Console.Write(array3[i, j] + " ");  
            }  
            Console.WriteLine();  
        }  
    }  
}

# 5J. Сапёр

**Задание:**

Игра «Сапёр». Поле для её простейшего варианта имеет размер 8 × 8. 10 клеток поля содержат мины, а в каждой из остальных клеток находится цифра от 0 до 8, равная количеству мин в соседних клетках. Вам известно расположение мин на поле. Восстановите все цифры.

**Входные данные:**

Ввод содержит 8 строк, описывающих игровое поле. Каждая из них содержит 8 символов «.» (клетка пуста) либо «\*» (клетка содержит мину).

Гарантируется, что поле содержит ровно 10 мин.

**Выходные данные:**

Выведите 8 строк, каждая из которых содержит 8 символов: клетки, содержащие мины, должны содержать символ «\*», остальные клетки — цифру от 0 до 8, соответствующую количеству мин в соседних клетках.

**Код:**

using System;

public class Test

{

static int readNum()

{

char c;

string fnum = "";

do

{

c = Convert.ToChar(Console.Read());

if ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57))

fnum += c;

} while ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57));

if (fnum == "") return 0;

return int.Parse(fnum);

}

public static void Main()

{

System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");

int[,] array = new int[8, 8];

char[,] res = new char[8, 8];

for (int i = 0; i < 8; i++)

{

string str = Console.In.ReadLine();

char[] temp = str.ToCharArray();

for (int j = 0; j < 8; j++)

{

if (temp[j] == '\*')

{

array[i, j] = 1;

}

else

{

array[i, j] = 0;

}

}

}

if (array[0, 0] == 0)

{

res[0, 0] += (char)(array[0, 1] + array[1, 0] + array[1, 1]);

}

else

res[0, 0] = '\*';

if (array[0, 7] == 0)

{

res[0, 7] += (char)(array[0, 6] + array[1, 6] + array[1, 7]);

}

else

res[0, 7] = '\*';

if (array[7, 0] == 0)

{

res[7, 0] += (char)(array[6, 0] + array[7, 1] + array[6, 1]);

}

else

res[7, 0] = '\*';

if (array[7, 7] == 0)

{

res[7, 7] += (char)(array[7, 6] + array[6, 7] + array[6, 6]);

}

else

res[7, 7] = '\*';

for (int i = 1; i < 7; i++)

{

if (array[0, i] == 0)

{

res[0, i] += (char)(array[0, i - 1] + array[0, i + 1] + array[1, i] + array[1, i - 1] + array[1, i + 1]);

}

else

res[0, i] = '\*';

}

for (int i = 1; i < 7; i++)

{

if (array[i, 0] == 0)

{

res[i, 0] += (char)(array[i - 1, 0] + array[i + 1, 0] + array[i, 1] + array[i - 1, 1] + array[i + 1, 1]);

}

else

res[i, 0] = '\*';

}

for (int i = 1; i < 7; i++)

{

if (array[i, 7] == 0)

{

res[i, 7] += (char)(array[i - 1, 7] + array[i + 1, 7] + array[i, 6] + array[i - 1, 6] + array[i + 1, 6]);

}

else

res[i, 7] = '\*';

}

for (int i = 1; i < 7; i++)

{

if (array[7, i] == 0)

{

res[7, i] += (char)(array[7, i - 1] + array[7, i + 1] + array[6, i] + array[6, i - 1] + array[6, i + 1]);

}

else

res[7, i] = '\*';

}

for (int i = 1; i < 7; i++)

{

for (int j = 1; j < 7; j++)

{

if (array[i, j] == 0)

{

res[i, j] += (char)(array[i - 1, j - 1] + array[i - 1, j] + array[i - 1, j + 1] + array[i, j - 1] + array[i, j + 1] + array[i + 1, j - 1] + array[i + 1, j] + array[i + 1, j + 1]);

}

else

res[i, j] = '\*';

}

}

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

if (res[i, j] == '\*')

Console.Write((char)res[i, j]);

else

Console.Write((int)res[i, j]);

}

Console.WriteLine();

}

}

}

# 6A. Символ в код

**Задание:**

Дан символ. Выведите его номер в кодировочной таблице ASCII.

**Входные данные:**

Ввод содержит один символ, ASCII-код которого находится в диапазоне [33; 126].

**Выходные данные:**

Выведите ASCII-код указанного символа.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        char c = (char)Console.Read();  
        Console.WriteLine((int)c);  
    }  
}

# 6B. Количество букв

**Задание:**

Дана строка. Определите, сколько её символов являются буквами.

**Входные данные:**

Ввод содержит строку S (1 ≤ |S| ≤ 105), состоящую из латинских букв, цифр и знаков препинания.

**Выходные данные:**

Выведите одно целое число — количество букв в строке S.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str = Console.ReadLine();  
        char[] ch = str.ToCharArray();  
        int k = 0;  
  
        for(int i = 0; i < str.Length; i++)  
    {  
            if((ch[i] >= 'A' && ch[i] <= 'Z') || (ch[i] >= 'a' && ch[i] <= 'z'))   
            {  
                k++;  
        }  
    }  
        Console.WriteLine(k);  
    }  
}

# 

# 6C. Древний шифр

**Задание:**

Одним из самых старых известных методов шифрования является атбаш, при помощи которого зашифрованы многие христианские тексты, например, часть свитков Мёртвого моря. Этот шифр достаточно прост: первая буква алфавита («a») в нём заменяется на последнюю («z»), вторая («b») — на предпоследнюю («y»), ..., последняя («z») — на первую («a»). Формально, буква, находящаяся на позиции X от начала алфавита, заменяется буквой, находящейся на позиции X от конца алфавита. Символы, отличные от букв, не изменяются.

Сможете ли вы расшифровать сообщение, зашифрованное этим древним методом?

**Входные данные:**

Ввод содержит одну строку длиной от 1 до 100 символов, состоящую из больших и малых латинских букв, пробелов и знаков препинания, — зашифрованное сообщение.

**Выходные данные:**

Выведите расшифрованное сообщение. Регистр букв должен быть сохранён.

**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string array = Console.ReadLine();  
        char[] a = array.ToCharArray();  
  
        for (int i = 0; i < array.Length; i++)  
        {  
            if (a[i] >= 'A' && a[i] <= 'Z')  
                a[i] = (char)('Z' - a[i] + 'A');  
            else if (a[i] >= 'a' && a[i] <= 'z')  
                a[i] = (char)('z' - a[i] + 'a');  
        }  
  
        Console.WriteLine(a);  
    }  
}

# 6D. Макс и перестановочный шифр

**Задание:**

В этой программе сам пользователь будет указывать, на какую букву должна заменяться каждая из 26 букв латинского алфавита. Регистр букв при шифровании сохраняется, а пробелы и знаки препинания остаются без изменений.

**Входные данные:**

Первая строка содержит 26 различных целых чисел Ci (1 ≤ Ci ≤ 26). Буква, имеющая в алфавите номер i, заменяется на букву с номером Ci.

Вторая строка содержит текст S (1 ≤ |S| ≤ 105), состоящий из латинских букв, пробелов и знаков препинания — зашифрованное сообщение.

**Выходные данные:**

Выведите расшифрованный текст.

**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57));  
        if (fnum == "") return 0;  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int[] key = new int[26];  
        int i = 0;  
        string str = Console.In.ReadLine();  
        string[] str2 = str.Split(' ');  
        foreach (var str3 in str2)  
        {  
            key[i] = Convert.ToInt32(str3);  
            i++;  
        }  
        string strTemp = Console.In.ReadLine();  
        char[] a = strTemp.ToCharArray();  
        int n = strTemp.Length;  
        for (i = 0; i < n; i++)  
        {  
            if (a[i] >= 'A' && a[i] <= 'Z')  
                for (int j = 0; j < 27; j++)  
                {  
                    if ((key[j] - 1) == (a[i] - 'A'))  
                    {  
                        a[i] = (char)('A' + j);  
                        break;  
                    }  
                }  
            else if (a[i] >= 'a' && a[i] <= 'z')  
                for (int j = 0; j < 27; j++)  
                {  
                    if ((key[j] - 1) == (a[i] - 'a'))  
                    {  
                        a[i] = (char)('a' + j);  
                        break;  
                    }  
                }  
        }  
        Console.WriteLine(a);  
    }  
}

# 6E. Постиранный пароль

**Задание:**

У вас есть набор букв, сколько предполагаемых “паролей” могли быть составлены из этого набора букв?

Помогите Максу справиться с его задачей.

**Входные данные:**

Первая строка содержит строку S (1 ≤ |S| ≤ 100), состоящую из строчных латинских букв.

Вторая строка содержит целое число N (0 ≤ N ≤ 1000) — количество вариантов пароля.

Следующие N строк описывают варианты пароля. Каждая из них содержит строку Pi (1 ≤ |Pi| ≤ 20), состоящую из строчных латинских букв.

**Выходные данные:**

Выведите количество вариантов пароля, которые могли быть составлены из имеющихся букв.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string strVocabulary = Console.ReadLine();  
        char[] vocabulary = strVocabulary.ToCharArray();  
        int n = Convert.ToInt32(Console.ReadLine());  
        int res = 0;  
        char[] tempVocabulary = new char[101];  
        for (int i = 0; i < n; i++)  
        {  
            string tempStr = Console.ReadLine();  
            char[] password = tempStr.ToCharArray();  
            vocabulary.CopyTo(tempVocabulary, 0);  
            int check = 0;  
            for (int j = 0; j < tempStr.Length; j++)  
            {  
                for (int k = 0; k < strVocabulary.Length; k++)  
                {  
                    if (password[j] == tempVocabulary[k])  
                    {  
                        tempVocabulary[k] = (char)45;  
                        check++;  
                        break;  
                    }  
                }  
            }  
  
            if (check == tempStr.Length)  
            {  
                res++;  
            }  
        }  
  
        Console.WriteLine(res);  
    }  
}

# 

# 6F. C++ и Java

**Задание:**

Некоторые студенты предпочитают использовать при решении задач практики язык программирования C++, некоторые — язык программирования Java. И у того, и у другого имеются как преимущества, так и недостатки. Если не удаётся составить решение на одном языке, в отдельных случаях имеет смысл попробовать использовать другой.

Для именования переменных в C++ используется следующее соглашение: идентификатор записывается только строчными буквами, отдельные слова в составе идентификатора разделяются символом подчёркивания. Примерами идентификаторов языка C++ являются «selection\_sort», «key\_value\_pair» и «max\_f\_p\_s».

Для именования переменных в Java используется так называемый «camelCase»: отдельные слова в составе идентификатора записываются слитно, первые буквы всех слов кроме начального являются заглавными. Примерами идентификаторов языка Java являются «selectionSort», «keyValuePair» и «maxFPS».

Правильно измените имя переменной.

**Входные данные:**

Ввод содержит строку S (1 ≤ |S| ≤ 100) — имя переменной. Гарантируется, что строка S является корректным идентификатором в рамках рассматриваемых стилей.

**Выходные данные:**

Если имя переменной соответствует стилю C++, выведите его аналог для стиля Java. Если имя переменной соответствует стилю Java, выведите его аналог для стиля C++. Если вариантов ответа несколько, выведите любой.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str1 = Console.ReadLine();  
        char[] chStr = str1.ToCharArray();  
        string str = "";  
        int f = 0;  
  
        for (int i = 0; i < str1.Length; i++)  
        {  
            if (chStr[i] == '\_')  
      {  
                f = 1;  
      }  
        }  
          
  
        for (int i = 0; i < str1.Length; i++)  
    {  
           if(f == 0)  
       {  
                if (chStr[i] >= 'a' && chStr[i] <= 'z')  
                {  
                    str += chStr[i];  
                }  
                else  
                {  
                    str += '\_';  
                    str += (char)((chStr[i] - 'A') + 'a');  
                }  
           }  
           else  
       {  
                if ((chStr[i] >= 'a' && chStr[i] <= 'z') || (chStr[i] >= 'A' && chStr[i] <= 'Z'))  
                {  
                    str += chStr[i];  
                }  
                else  
                {  
                    str += (char)((chStr[i + 1] - 'a') + 'A');  
                    i++;  
                }  
           }  
    }  
        Console.WriteLine(str);  
    }  
}

# 6G. Слова

**Задание:**

Дан текст. Переведите все содержащиеся в нём слова в верхний регистр и выведите по отдельности, сохраняя их порядок.

**Входные данные:**

Ввод содержит строку S (1 ≤ |S| ≤ 1000), состоящую из латинских букв, пробелов и знаков препинания. Гарантируется, что строка содержит не менее одного слова.

**Выходные данные:**

Выведите в отдельных строках переведённые в верхний регистр слова, содержащиеся в строке S, в порядке появления. Повторяющиеся слова следует выводить несколько раз.

**Код:**

using System;  
  
public class Test  
{  
    static bool isalpha(char c)  
    {  
        if (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z')  
        {  
            return true;  
        }  
        return false;  
  
    }  
  
    static char upreg(char c)  
    {  
        if (c >= 'a' && c <= 'z')  
        {  
            return (char)(c - 'a' + 'A');  
        }  
        else  
        {  
            return c;  
        }  
  
  
    }  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str = Console.ReadLine();  
        str += '.';  
        string word = "";  
        for (int i = 0; i < str.Length; i++)  
        {  
            if (isalpha(str[i]))  
            {  
                word += upreg(str[i]);  
            }  
            else  
            {  
                if (word != "")  
                {  
                    Console.WriteLine(word);  
                    word = "";  
                }  
            }  
        }

# 6I. Автоформатирование

**Задание:**

Составьте программу, которая автоматически форматирует текст по следующим правилам:

* В тексте не должно быть двух или более пробелов подряд;
* Перед точками не должно быть пробелов, между точкой и следующим словом должен быть одиночный пробел;
* Первые буквы предложений должны быть заглавными, остальные буквы — строчными.

**Входные данные:**

Ввод содержит строку *S* (10 ≤ |*S*| ≤ 1000), состоящую из латинских букв, пробелов и точек, — текст, который нужно отформатировать. Гарантируется, что исходный текст начинается с буквы и заканчивается точкой, а в отформатированном тексте не будет двух подряд идущих точек.  
**Выходные данные:**  
Выведите отформатированный текст.  
**Код:**

using System;  
  
public class Test  
{  
    static bool isalpha(char c)  
    {  
        if (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z')  
        {  
            return true;  
        }  
        return false;  
  
    }  
  
    static char toup(char c)  
    {  
        if (c >= 'a' && c <= 'z')  
        {  
            return (char)(c - 'a' + 'A');  
        }  
        else  
        {  
            return c;  
        }  
  
  
    }  
  
    static char tolow(char c)  
    {  
        if (c >= 'A' && c <= 'Z')  
        {  
            return (char)(c - 'A' + 'a');  
        }  
        else  
        {  
            return c;  
        }  
    }  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str = Console.ReadLine();  
        char[] ch = str.ToCharArray();  
        string tmp = "";  
        int f = 1;  
        for(int i = 0; i < str.Length; i++)  
        {  
          if (isalpha(ch[i]))   
          {  
            if (f == 1) {  
              if(i > 0)   
              {  
                tmp += ' ';    
              }  
                tmp += toup(ch[i]);  
                f = 0;  
              }  
              else {  
                tmp += tolow(ch[i]);  
            }      
          }  
          else if(ch[i] == '.')   
          {  
            tmp += '.';  
            f = 1;  
          }  
          else if (ch[i] == ' ')   
          {  
            if(i > 0 && i < str.Length - 1)   
            {  
              if(ch[i - 1] == '.')  
              {  
                ch[i] = '.';  
              }  
              else if(isalpha(ch[i + 1]) && f != 1)  
              {  
                tmp += ' ';  
              }  
            }  
          }  
            
           
        }  
        if(ch[str.Length - 1] != '.')  
      {  
          tmp += '.';    
      }  
      Console.WriteLine(tmp);  
          
    }  
}МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ

Кафедра «Информационные системы»

Дисциплина «Технологии программирования»

**ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ**

**«МАГАЗИН ОХРАННЫХ СИСТЕМ»**

выполнил: студент

гр. ПИбд-21

Эмирян В. Л.

проверил: ст. преподаватель

Е.Н. Эгов

Ульяновск

2021 г.

# Описание основного приложения

## Назначение

Приложение способно вести учет устройств и комплектаций на производстве, создавать заказы, указывая заказчика, выводить отчеты по устройствам, комплектациям и заказам в формате word, excel и pdf соответственно. Также выводит список сообщений с рабочего почтового адреса, уведомляя о появлении нового заказа. Имеется функция сохранения данных в формате xml или json.

## Работа со справочниками

В справочнике с материалами можно добавлять, удалять, изменять устройства, а также обновлять их список.

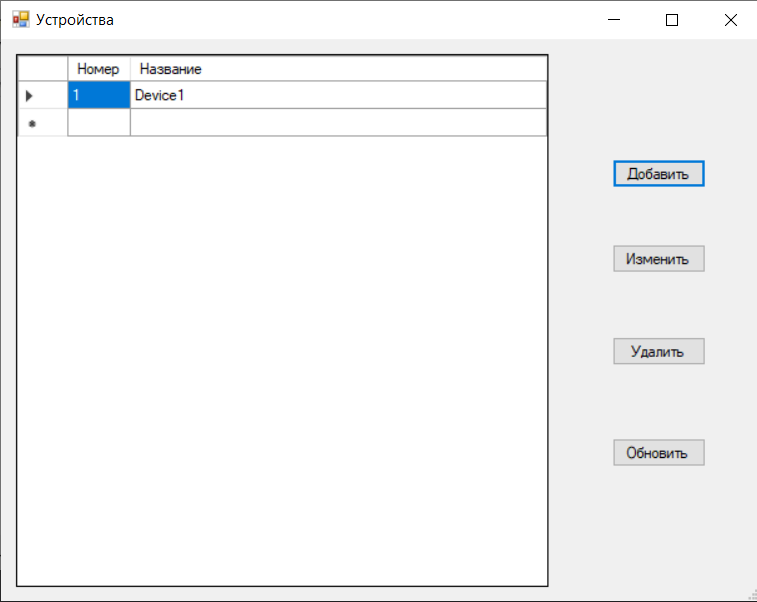


Рисунок 1 «Справочник с устройствами»

При нажатии на кнопку добавить или изменить откроется окно, в котором надо добавить или изменить название соответственно, после чего надо не забыть нажать на кнопку сохранить.

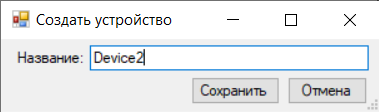


Рисунок 2 «Окно добавления и изменения устройств»

При нажатии на кнопку удалить появляется всплывающее окно, в котором требуется подтвердить или отменить удаление.

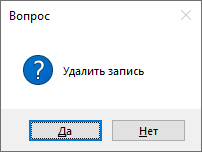


Рисунок 3 «Удаление записи»

При нажатии на кнопку обновления в таблицу помещаются недостающие записи, но в общем случае список обновляется сам.

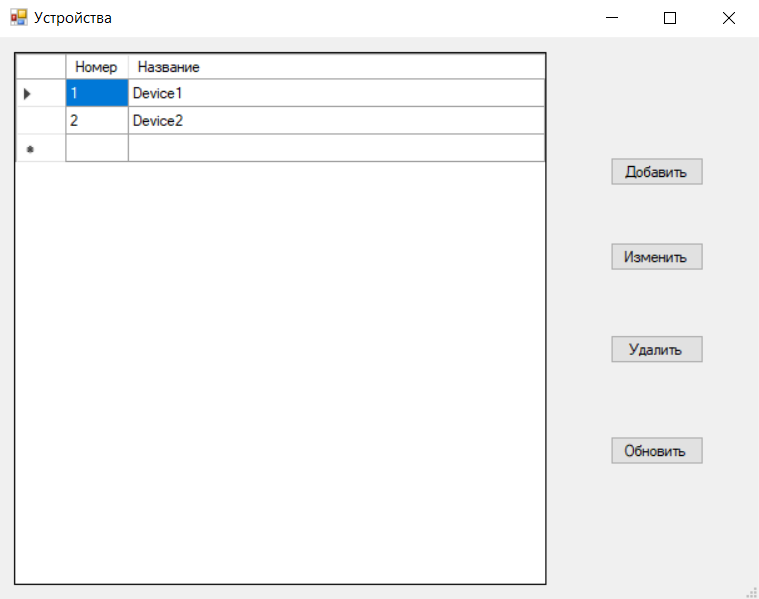


Рисунок 4 «Обновление списка»

В справочнике с комплектациями можно добавлять, удалять, изменять комплектакции, а также обновлять список комплектаций.

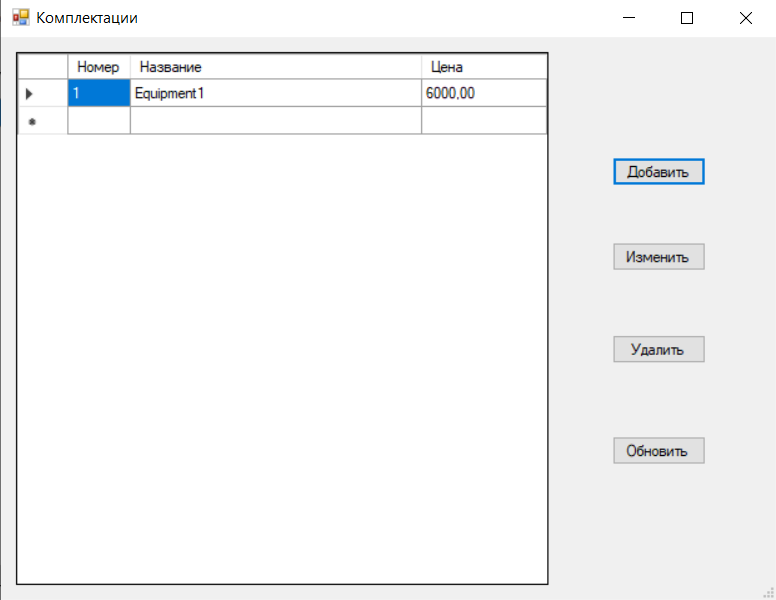


Рисунок 5 «Справочник с комплектациями»

При нажатии на кнопку добавить или изменить откроется окно, в котором нам надо будет добавить или изменить название и цену, а также добавить, удалить и изменить устройства, содержащиеся в комплектации, обновить список этих устройств и нажать кнопку сохранить.

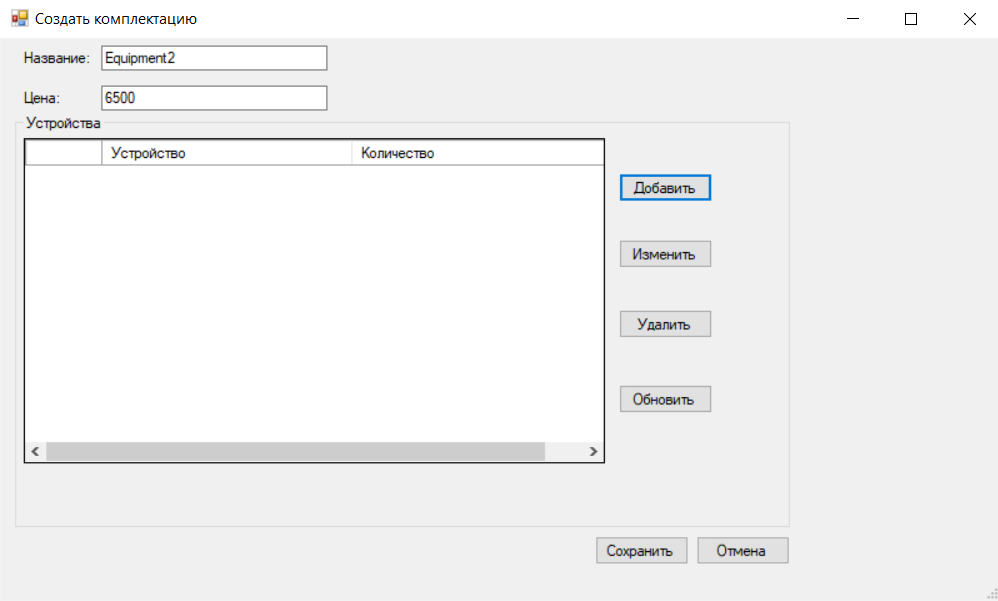


Рисунок 6 «Окно добавления и изменения комплектаций»

При нажатии на кнопку добавить или изменить откроется окно, в котором нам надо будет добавить или изменить устройство комплектации и нажать кнопку сохранить, иначе изменения не сохранятся или запись не добавится в список.

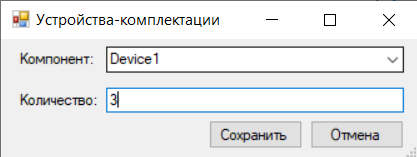


Рисунок 7 «Окно добавления и изменения устройств комплектации»

При нажатии на кнопку удалить появляется всплывающее окно, в котором требуется подтвердить или отменить удаление.

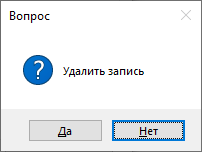


Рисунок 8 «Удаление материала из комплектации»

При нажатии на кнопку обновления в таблицу помещаются недостающие записи, но в общем случае список обновляется сам.

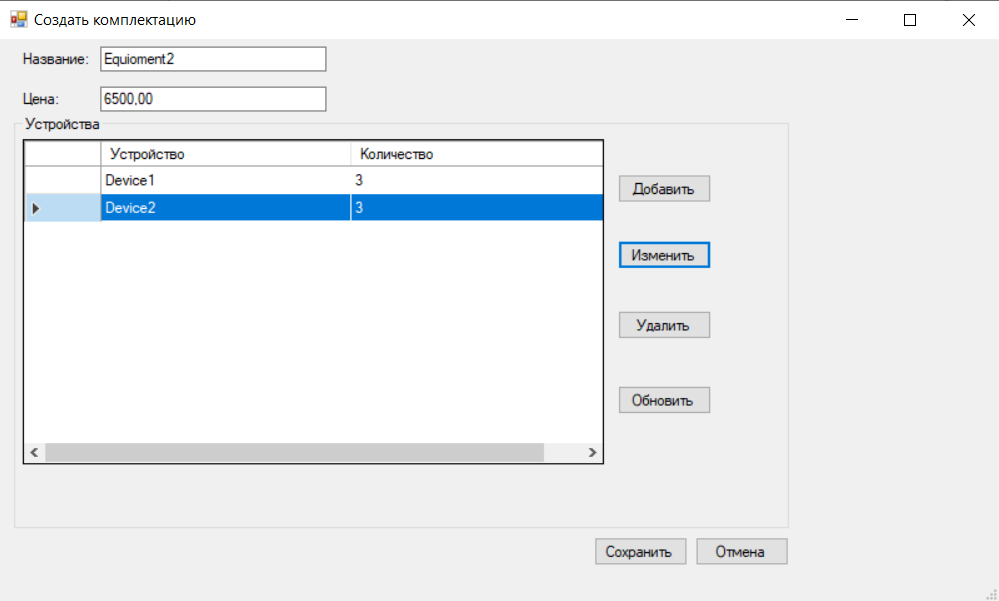
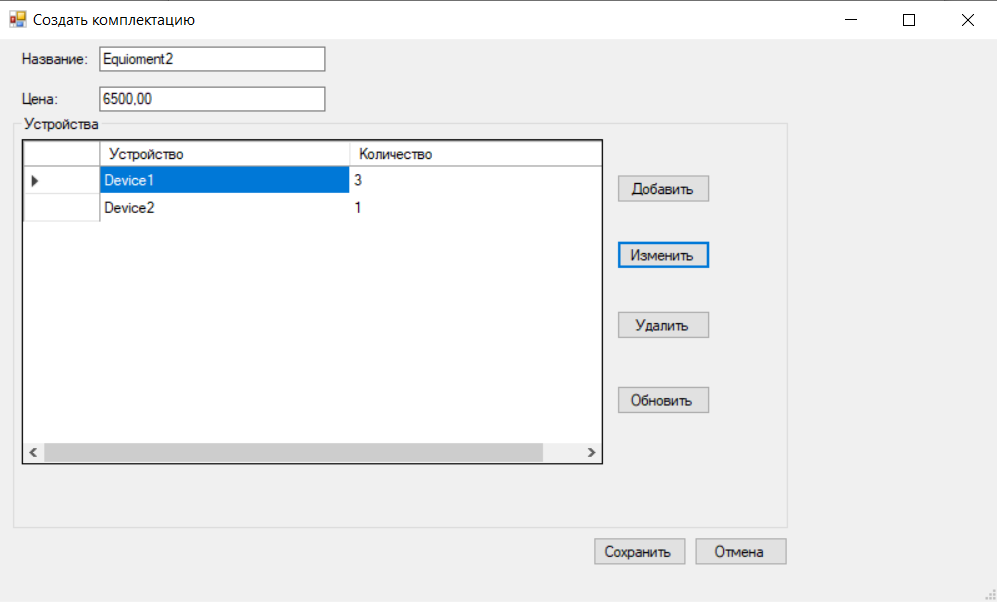


Рисунок 9 «Обновление списка устройств комплектации»



**Рисунок 10 «Изменение списка устройств комплектации»**

При нажатии на кнопку удалить появляется всплывающее окно, в котором требуется подтвердить или отменить удаление.

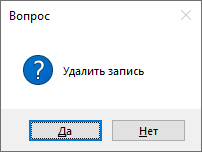


Рисунок 11 «Удаление записи»

При нажатии на кнопку обновления в таблицу помещаются недостающие записи, но в общем случае список обновляется сам.

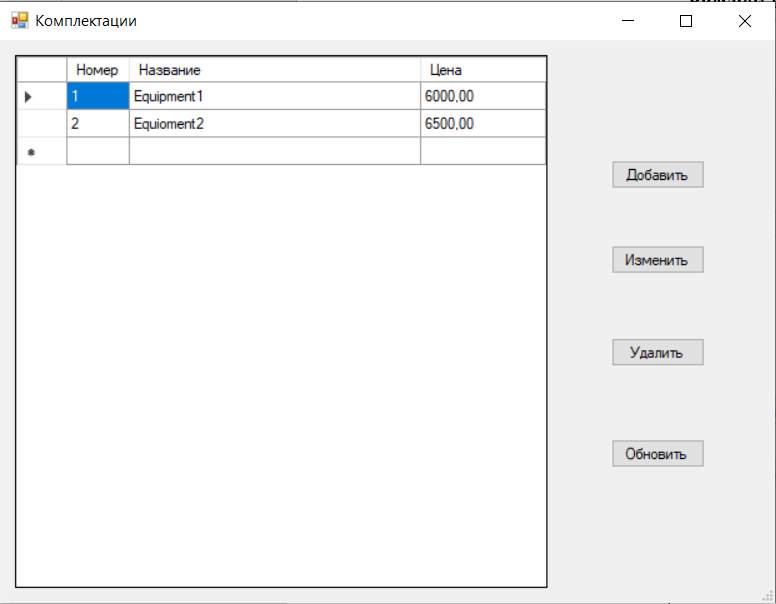


Рисунок 12 «Обновление списка»

В справочнике с клиентами можно удалять клиентов и обновлять список клиентов.

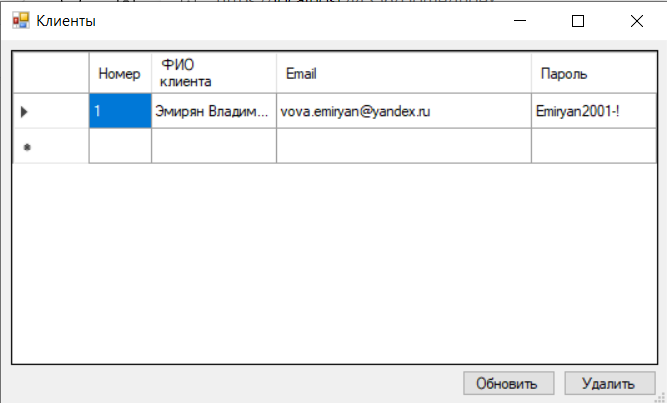


Рисунок 13 «Справочник с клиентами»

При нажатии на кнопку удалить появляется всплывающее окно, в котором требуется подтвердить или отменить удаление.

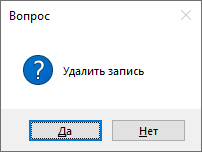


Рисунок 14 «Удаление записи»

При нажатии на кнопку обновления в таблицу помещаются недостающие записи, но в общем случае список обновляется сам.

В справочнике с исполнителями можно добавлять, удалять, изменять исполнителей, а также обновлять список исполнителей.

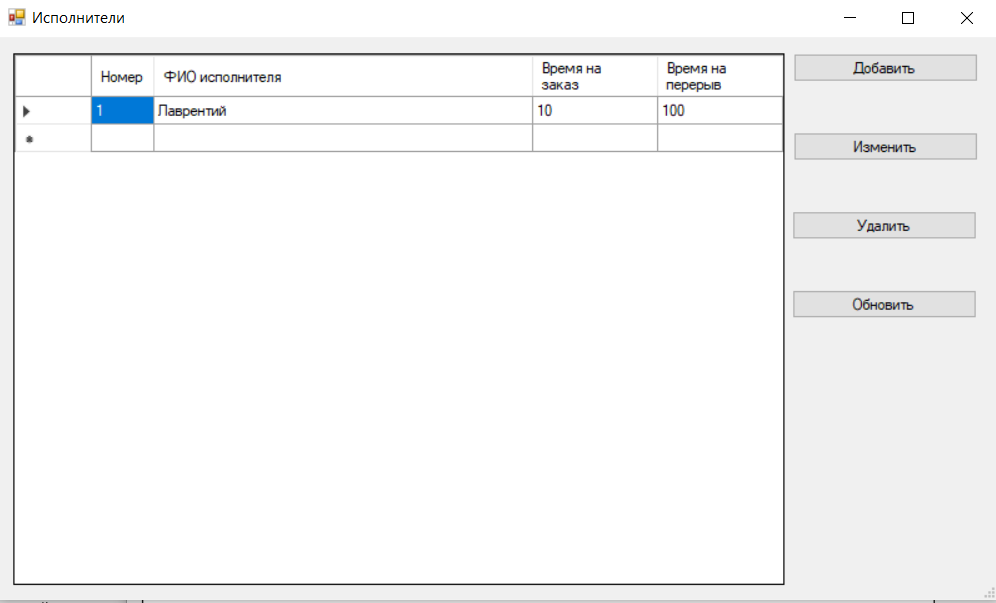


Рисунок 15 «Справочник с исполнителями»

При нажатии на кнопку добавить или изменить откроется окно, в котором нам надо будет добавить или изменить имя, время работы и время отдыха и нажать кнопку сохранить, иначе изменения не сохранятся или запись не добавится в список.

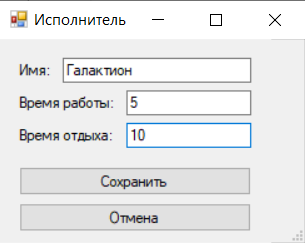


Рисунок 16 «Окно добавления и изменения исполнителей»

При нажатии на кнопку удалить появляется всплывающее окно, в котором требуется подтвердить или отменить удаление.

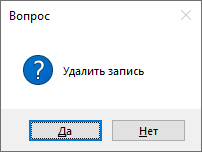


Рисунок 17 «Удаление записи»

При нажатии на кнопку обновления в таблицу помещаются недостающие записи, но в общем случае список обновляется сам.

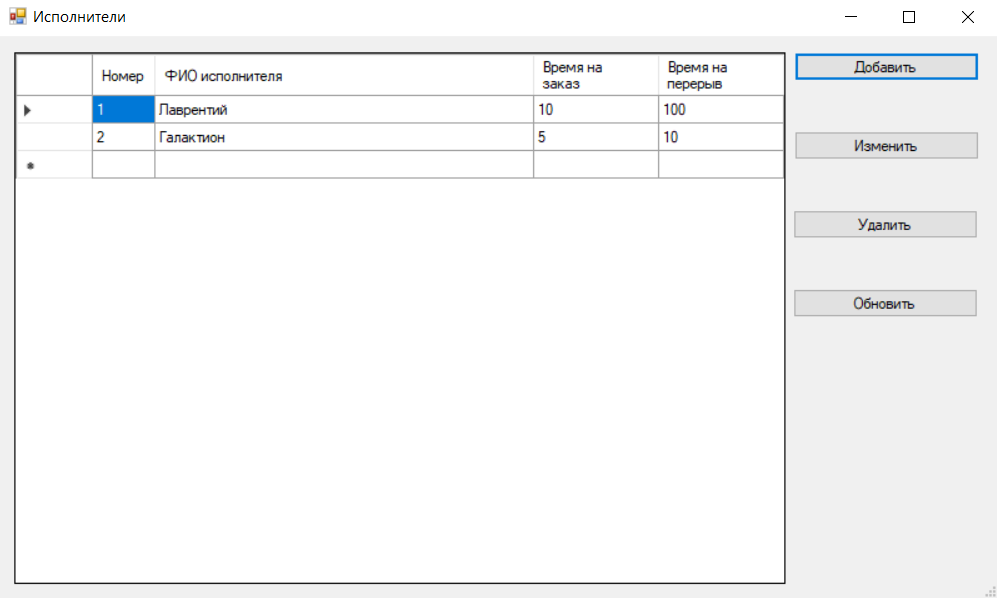


Рисунок 18 «Обновление списка»

## Работа с заказами

Для создания заказа требуется нажать на кнопку «Создать заказ» на основной форме программы.

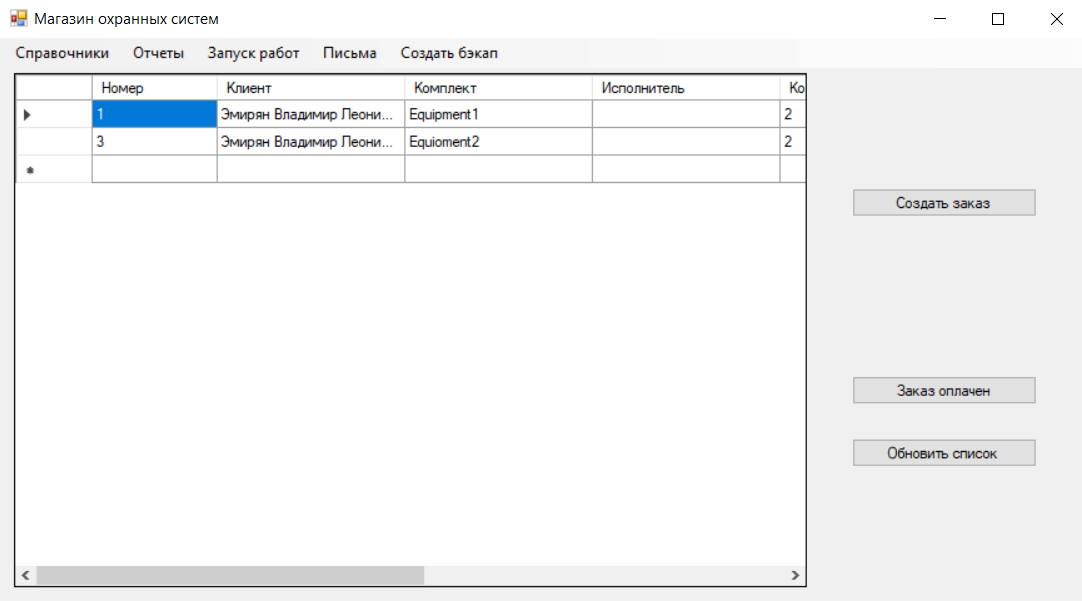


Рисунок 24 «Основное окно»

Далее нужно выбрать клиента, комплектацию и количество комплектаций в заказе. Сумма заказа считается в закрытом для изменений поле, после этого нужно нажать кнопку сохранить, иначе заказ не создастся.

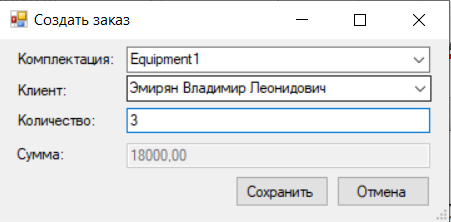


Рисунок 25 «Окно создания заказа»

После создания заказ переходит в статут «Принят». Далее нужно нажать пункт меню «Запуск работ», после чего заказ перейдет в статус «Выполняется», и по истечению выполнения в статус «Готов».

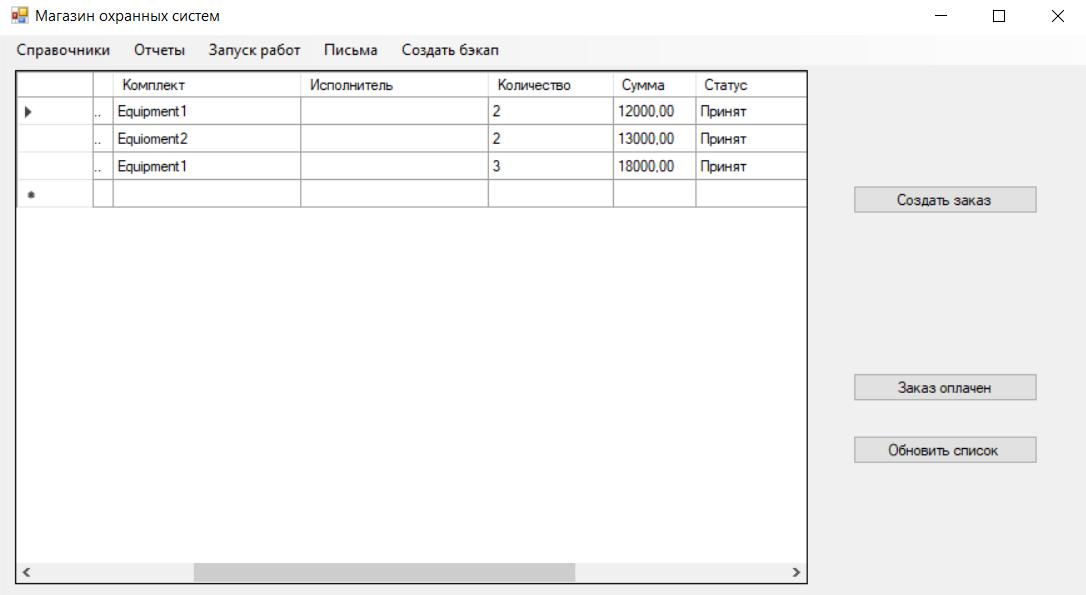


Рисунок 26 «Основное окно после добавления нового заказа»

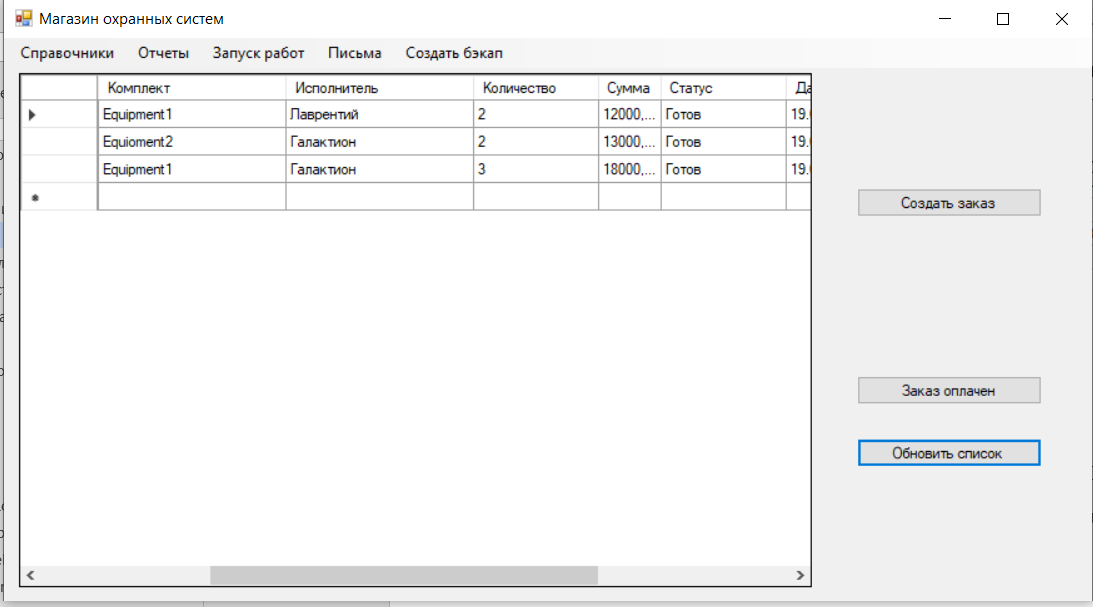


Рисунок 27 «Основное окно после начала работ»

Для перевода заказа в статус «Оплачен» нужно нажать кнопку «Оплатить заказ».

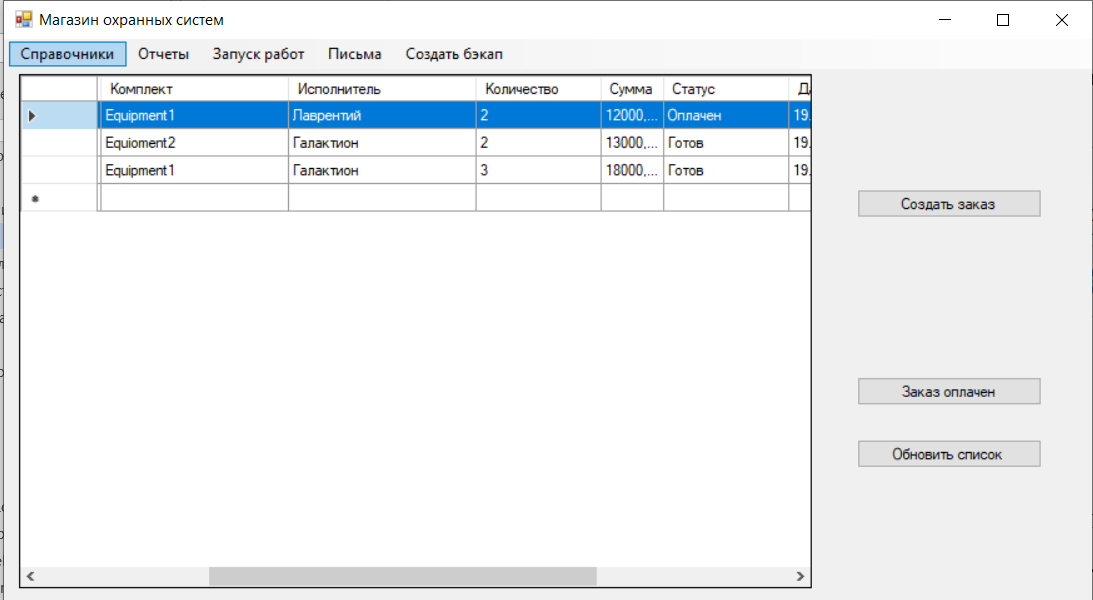


Рисунок 28 «Основное окно после оплаты заказа»

# Описание клиентского приложения

## Регистрация и вход в систему

Для регистрации пользователя необходимо открыть клиентское приложение и нажать пункт меню «Регистрация», после чего ввести логин (рабочая почта), пароль (от 8 до 50 символов, должен содержать латинские буквы, цифры и символы) и ФИО, затем нажать синюю кнопку «Регистрация».

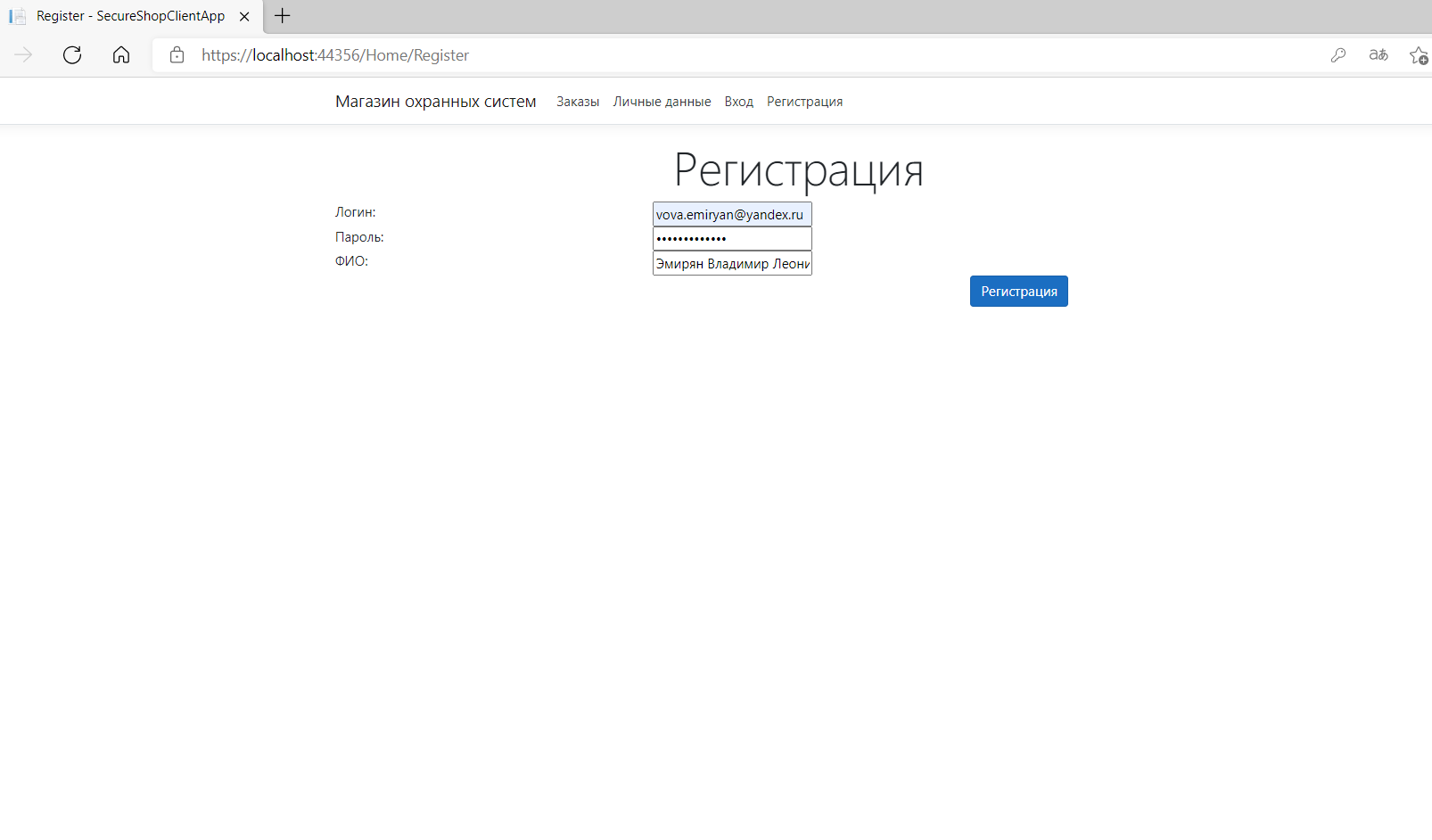


Рисунок 29 «Регистрация пользователя»

Для входа в систему нужно нажать пункт меню «Вход», а затем ввести свой логин и пароль от аккаунта и нажать синюю кнопку «Вход».

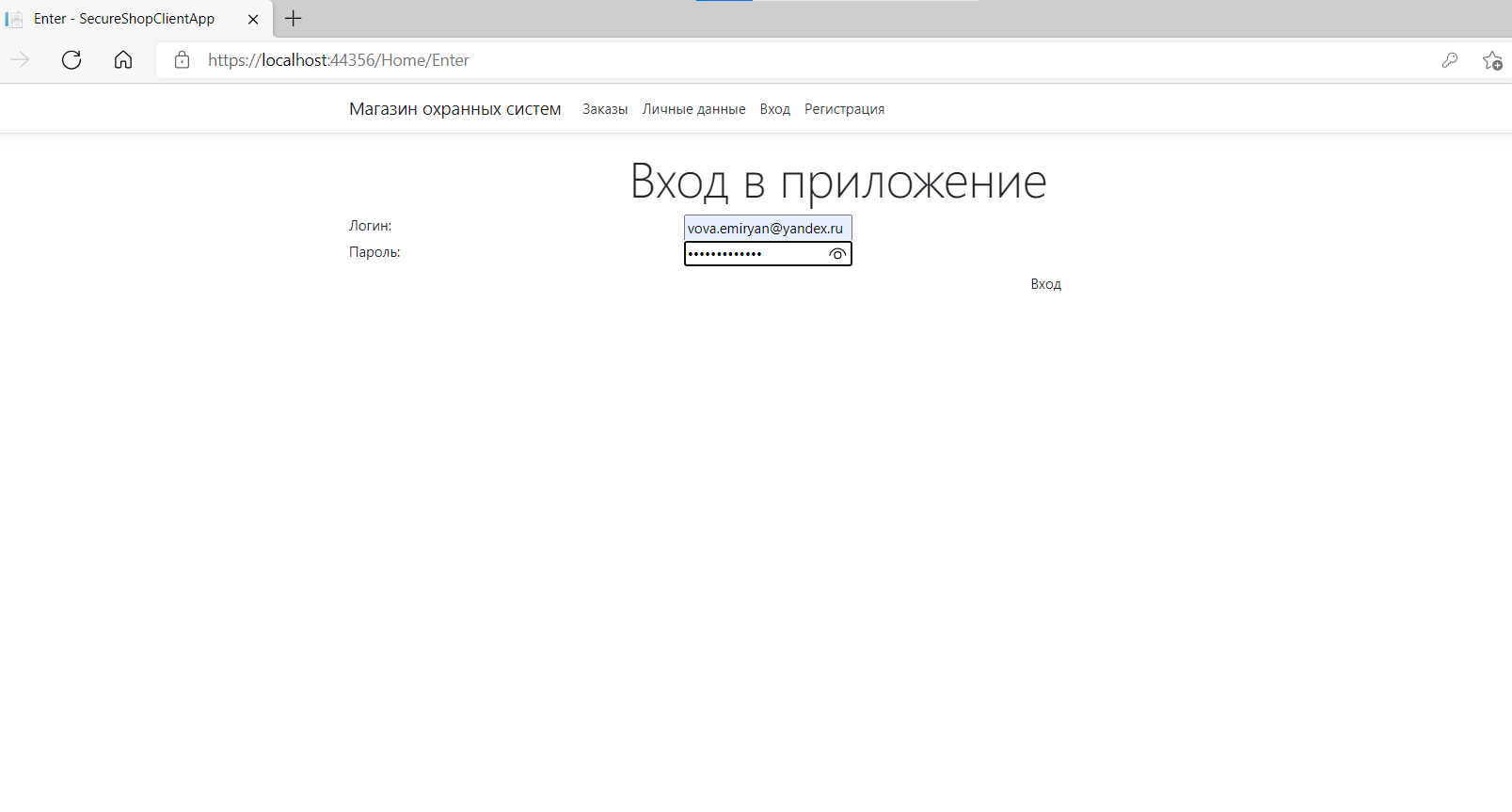


Рисунок 30 «Вход в приложение»

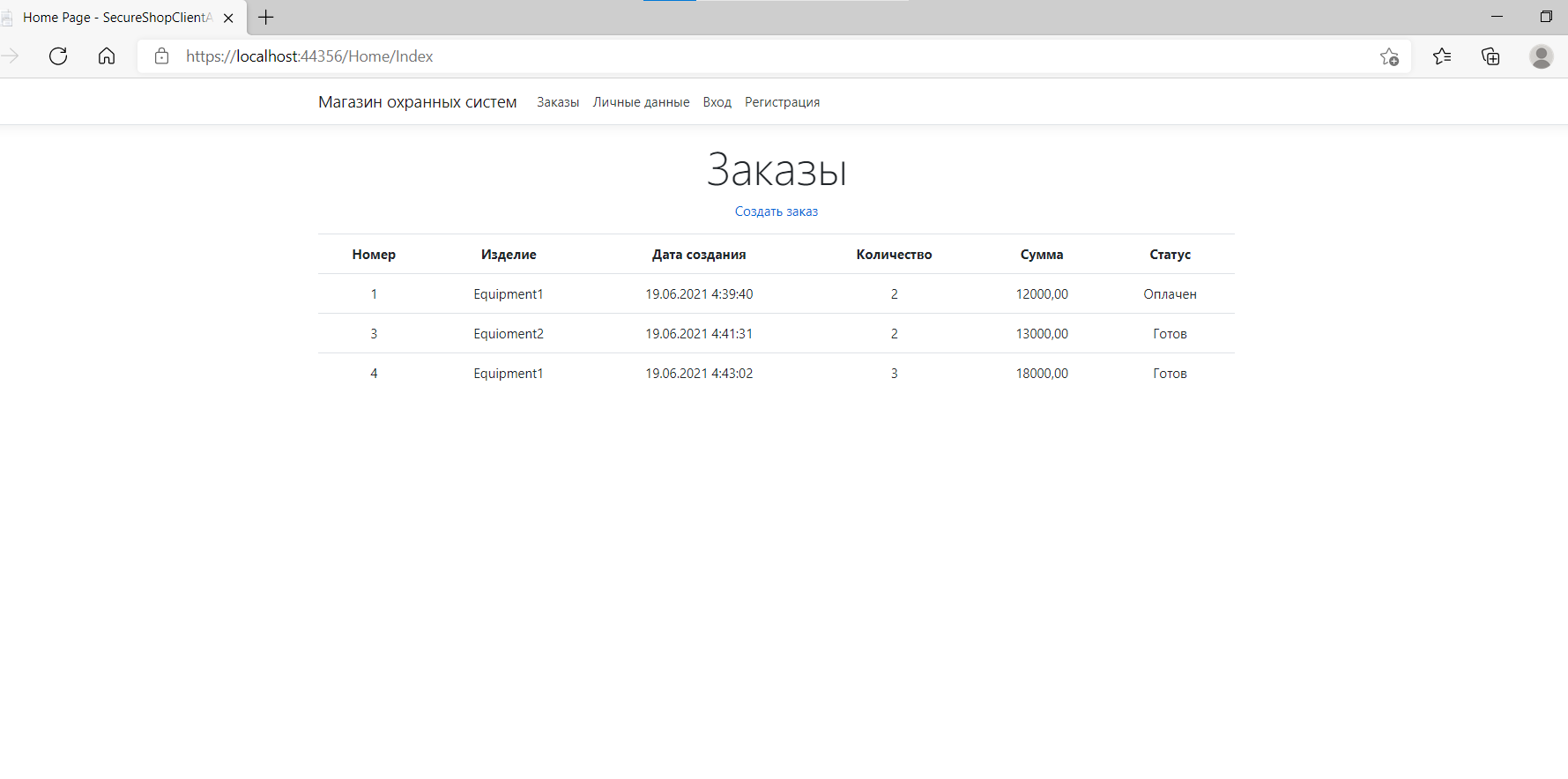


Рисунок 31 «Результат входа в приложение»

## Учетные данные аккаунта

Для смены учетных данных необходимо нажать на пункт меню «Личные данные» или ссылку «Личные данные» в подвале сайта, после чего менять любые данные на странице с личными данными, а затем нажать сохранить, иначе изменения не сохраняться.

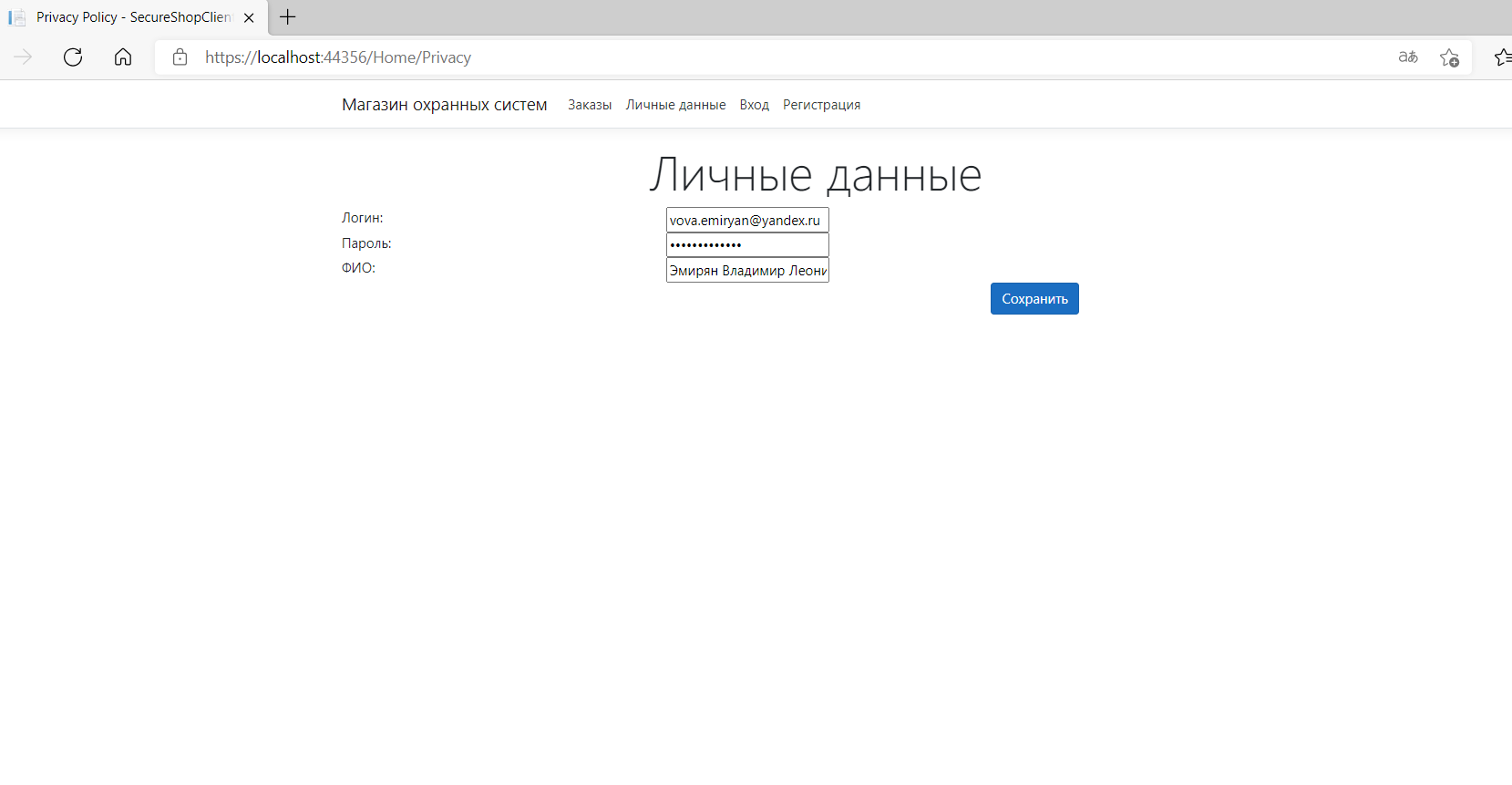


Рисунок 32 «Изменение личных данных»

## Работа с заказами

Для создания заказа необходимо нажать на ссылку «Создать заказ» на странице с заказами.

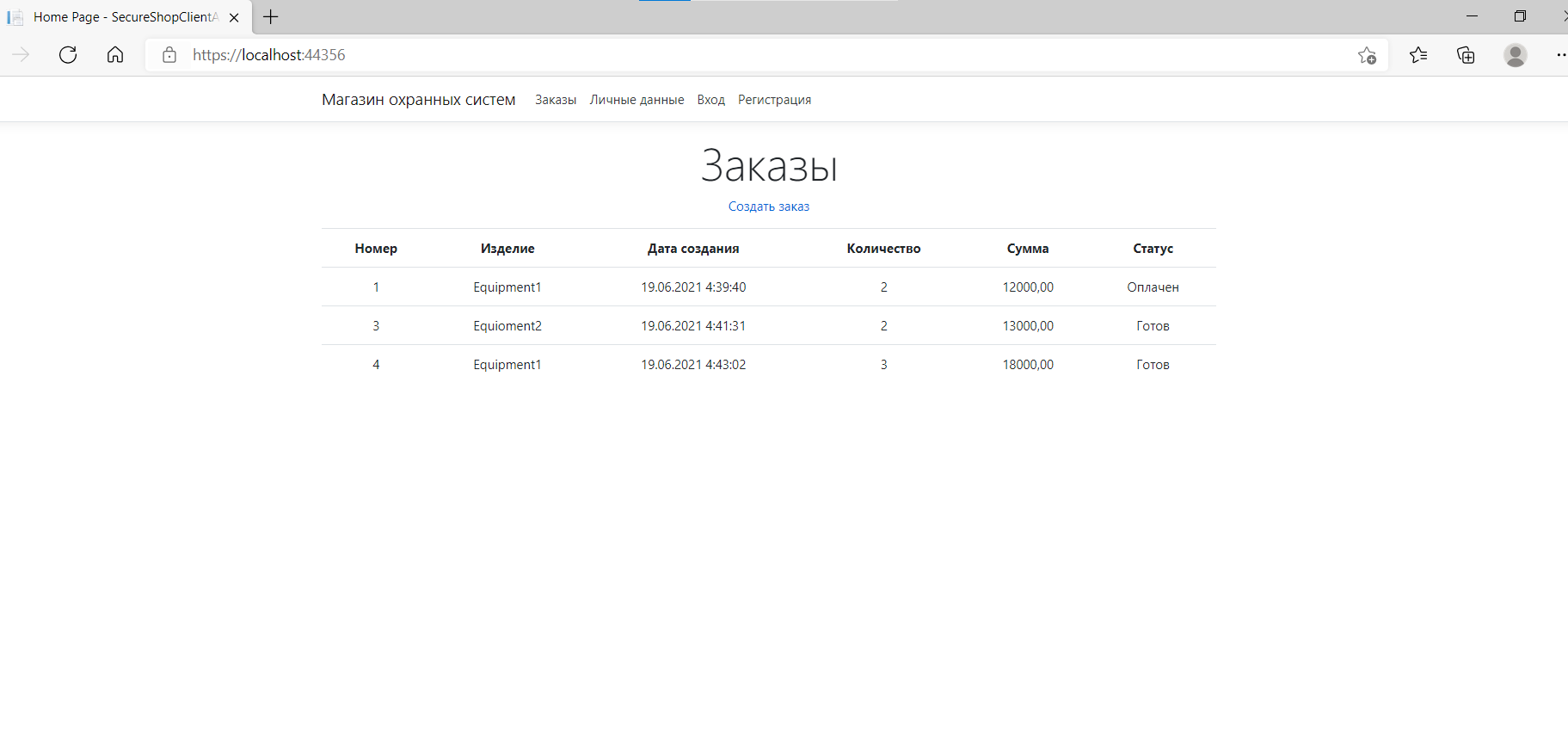


Рисунок 33 «Страница заказов»

Далее выбрать комплектация и количество комплектаций в заказе и нажать кнопку «Создать».

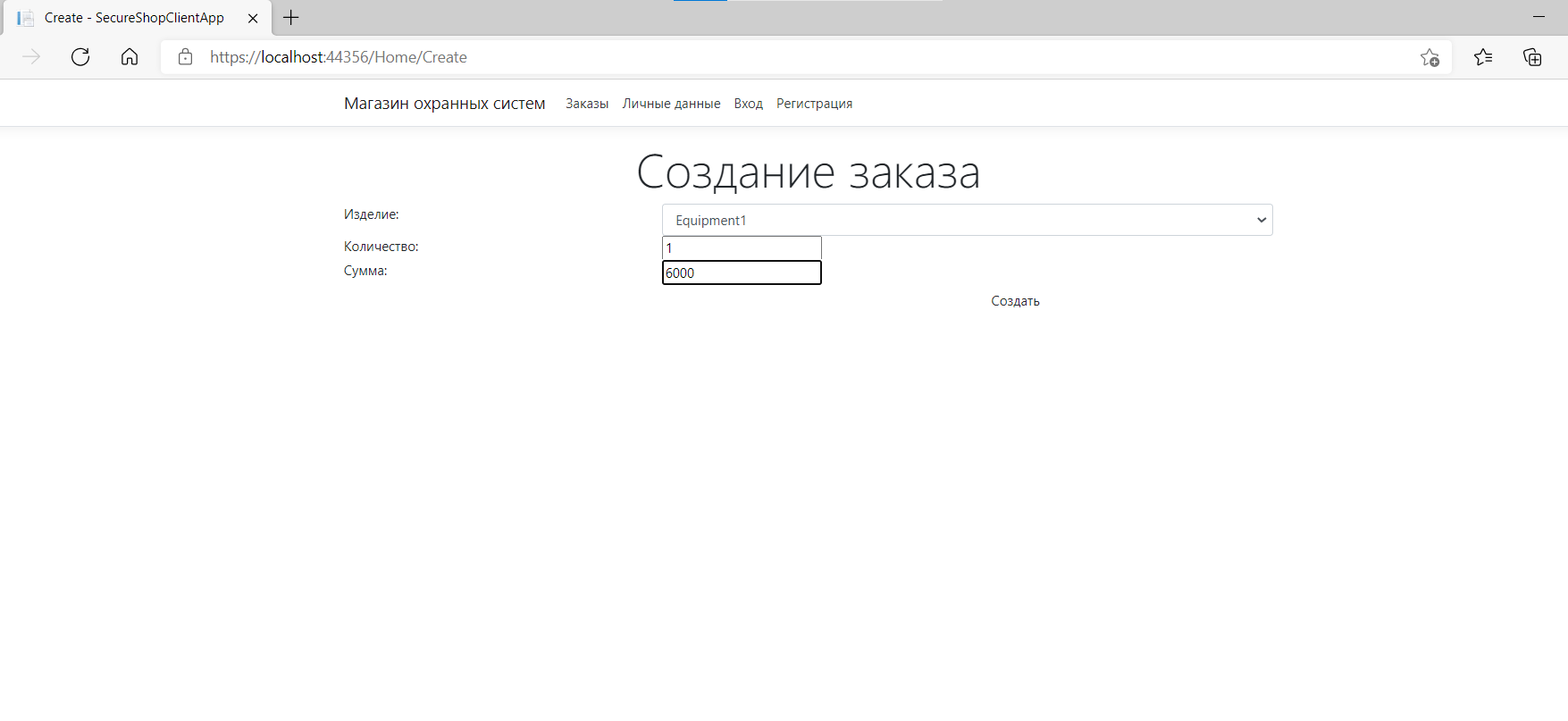


Рисунок 34 «Страница создания заказа»

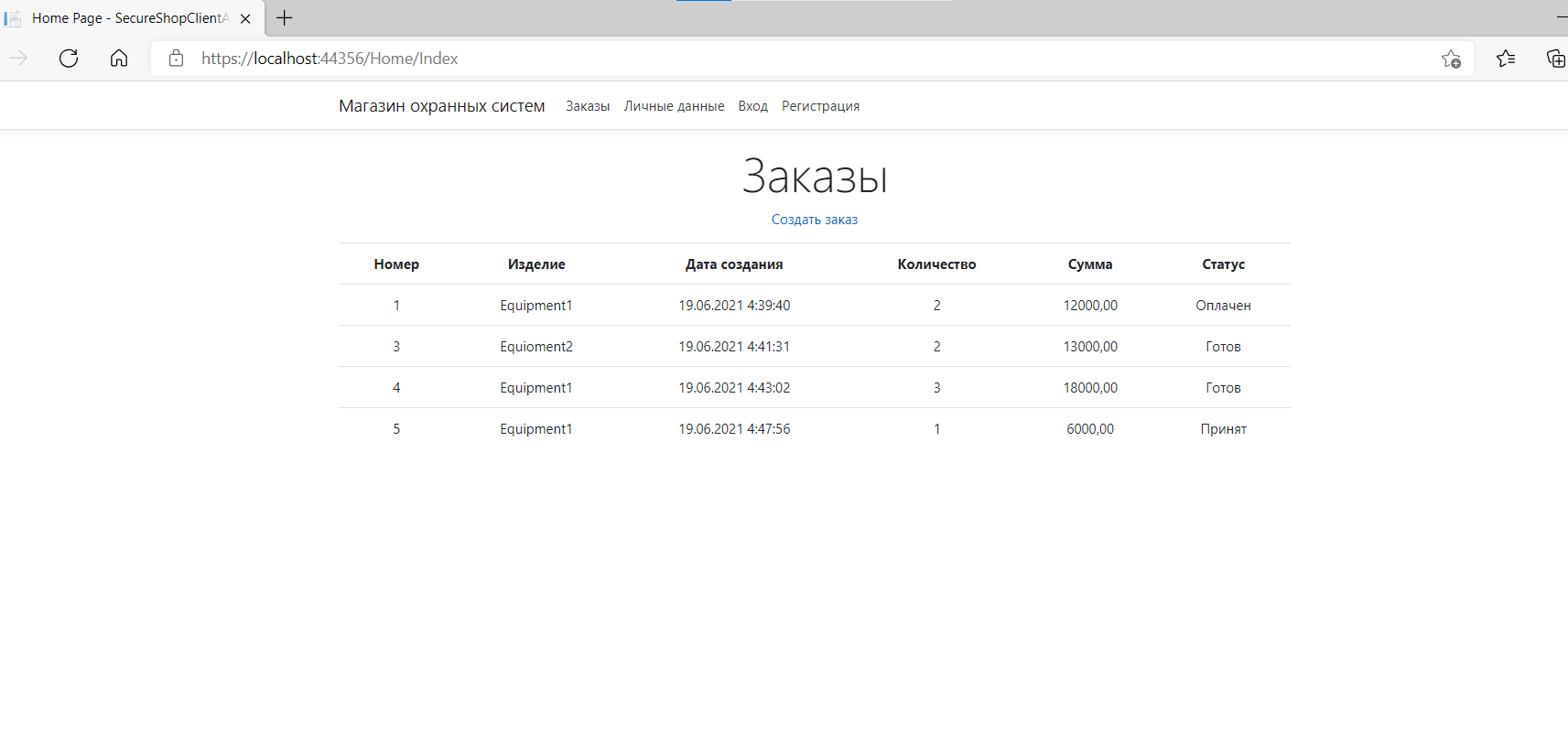
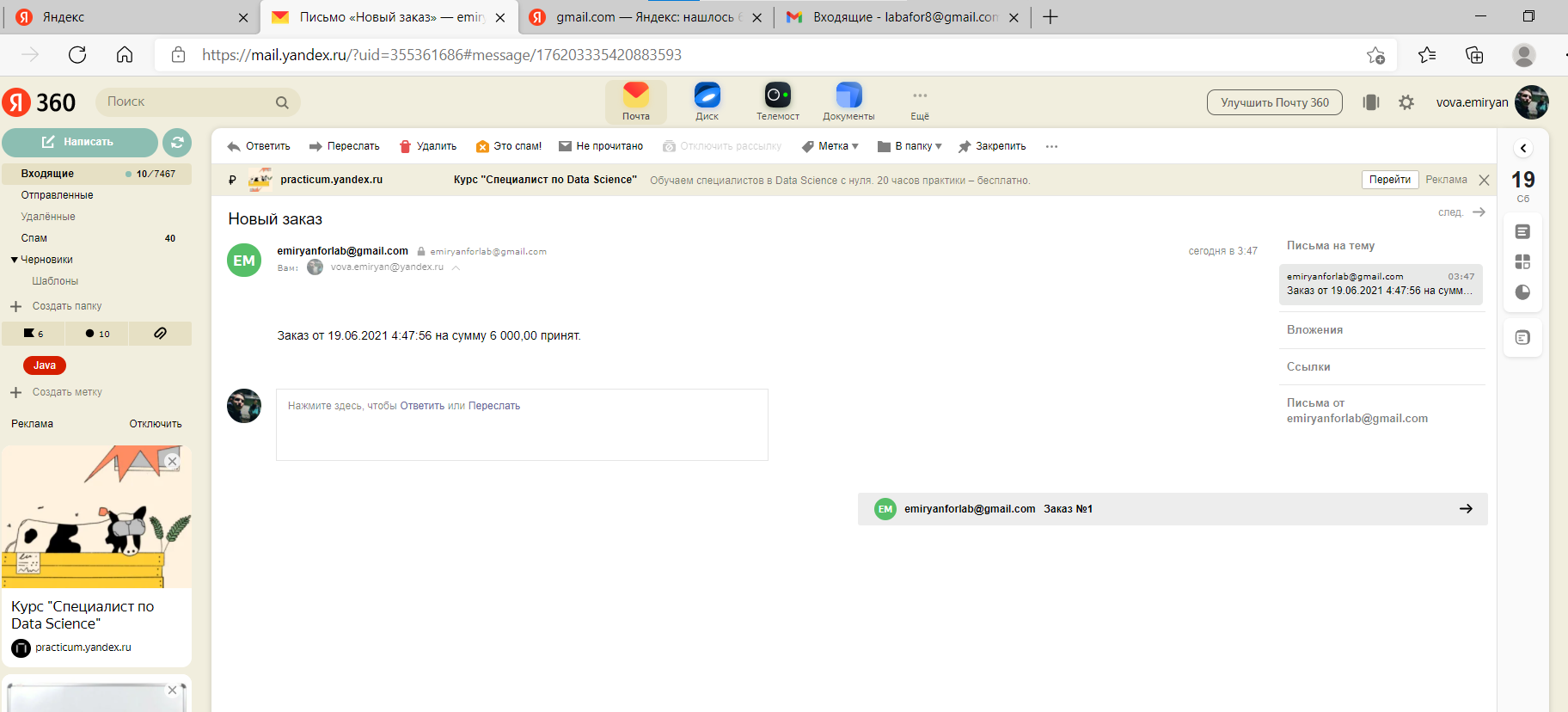


Рисунок 35 «Страница заказов после создания нового заказа»



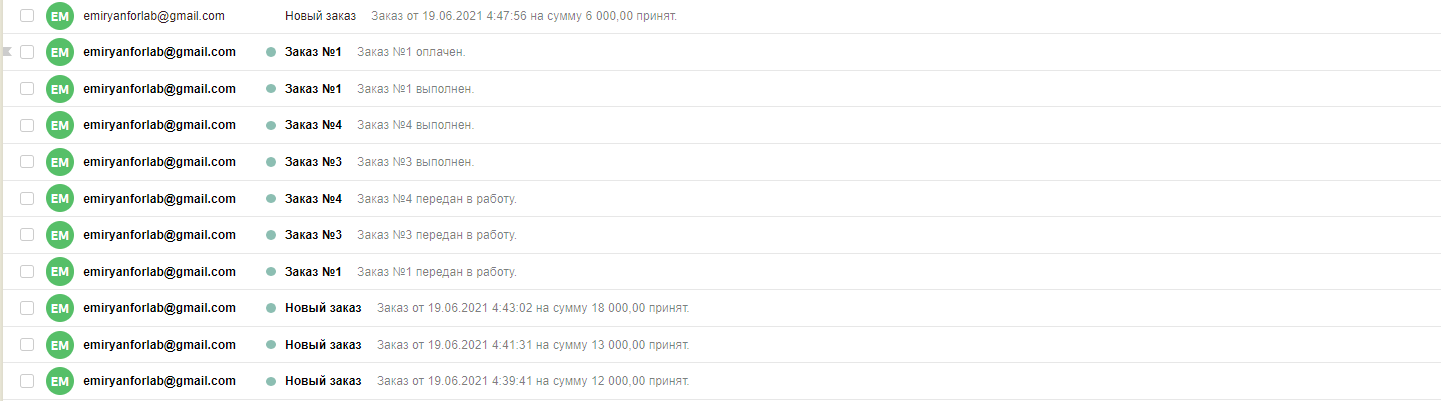


Рисунок 36 «Письмо на почте указанной как логин клиента»

# Листинг

## Код основной части

## Проект GiftShopBusinessLogic

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

/// <summary>

/// Данные для смены статуса заказа

/// </summary>

public class ChangeStatusBindingModel

{

public int OrderId { get; set; }

public int? ImplementerId { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Runtime.Serialization;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

[DataContract]

public class ClientBindingModel

{

[DataMember]

public int? Id { get; set; }

[DataMember]

public string ClientFIO { get; set; }

[DataMember]

public string Email { get; set; }

[DataMember]

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Runtime.Serialization;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

[DataContract]

public class CreateOrderBindingModel

{

[DataMember]

public int ClientId { get; set; }

[DataMember]

public int EquipmentId { get; set; }

[DataMember]

public int Count { get; set; }

[DataMember]

public decimal Sum { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

/// <summary>

/// Компонент, требуемый для изготовления комплектации

/// </summary>

public class DeviceBindingModel

{

public int? Id { get; set; }

public string DeviceName { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

public class EquipmentBindingModel

{

public int? Id { get; set; }

public string EquipmentName { get; set; }

public decimal Price { get; set; }

public Dictionary<int, (string, int)> EquipmentDevices { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

/// <summary>

/// Исполнитель, выполняющий заказы

/// </summary>

public class ImplementerBindingModel

{

public int? Id { get; set; }

public string Name { get; set; }

public int WorkingTime { get; set; }

public int PauseTime { get; set; }

}

}

using System;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.BindingModels

{

public class MessageInfoBindingModel

{

[DataMember]

public int? ClientId { get; set; }

[DataMember]

public string MessageId { get; set; }

[DataMember]

public string FromMailAddress { get; set; }

[DataMember]

public string Subject { get; set; }

[DataMember]

public string Body { get; set; }

[DataMember]

public DateTime DateDelivery { get; set; }

}

}

using SecureShopBusinessLogic.Enums;

using System;

namespace SecureShopBusinessLogic.BindingModels

{

/// <summary>

/// Заказ

/// </summary>

public class OrderBindingModel

{

public int? Id { get; set; }

public int? ClientId { get; set; }

public int EquipmentId { get; set; }

public int? ImplementerId { get; set; }

public int Count { get; set; }

public decimal Sum { get; set; }

public OrderStatus Status { get; set; }

public DateTime DateCreate { get; set; }

public DateTime? DateImplement { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

public bool? FreeOrders { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.BindingModels

{

public class ReportBindingModel

{

public string FileName { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using System;

using System.Collections.Generic;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class ClientLogic

{

private readonly IClientStorage \_clientStorage;

public ClientLogic(IClientStorage clientStorage)

{

\_clientStorage = clientStorage;

}

public List<ClientViewModel> Read(ClientBindingModel model)

{

if (model == null)

{

return \_clientStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<ClientViewModel> { \_clientStorage.GetElement(model) };

}

return \_clientStorage.GetFilteredList(model);

}

public void CreateOrUpdate(ClientBindingModel model)

{

var element = \_clientStorage.GetElement(new ClientBindingModel

{

Email = model.Email

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Этот email уже используется");

}

if (model.Id.HasValue)

{

\_clientStorage.Update(model);

}

else

{

\_clientStorage.Insert(model);

}

}

public void Delete(ClientBindingModel model)

{

var element = \_clientStorage.GetElement(new ClientBindingModel { Id = model.Id });

if (element == null)

{

throw new Exception("Клиент не найден");

}

\_clientStorage.Delete(model);

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class DeviceLogic

{

private readonly IDeviceStorage \_deviceStorage;

public DeviceLogic(IDeviceStorage deviceStorage)

{

\_deviceStorage = deviceStorage;

}

public List<DeviceViewModel> Read(DeviceBindingModel model)

{

if (model == null)

{

return \_deviceStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<DeviceViewModel> { \_deviceStorage.GetElement(model)};

}

return \_deviceStorage.GetFilteredList(model);

}

public void CreateOrUpdate(DeviceBindingModel model)

{

var element = \_deviceStorage.GetElement(new DeviceBindingModel

{

DeviceName = model.DeviceName

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть компонент с таким названием");

}

if (model.Id.HasValue)

{

\_deviceStorage.Update(model);

}

else

{

\_deviceStorage.Insert(model);

}

}

public void Delete(DeviceBindingModel model)

{

var element = \_deviceStorage.GetElement(new DeviceBindingModel { Id = model.Id });

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_deviceStorage.Delete(model);

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class EquipmentLogic

{

private readonly IEquipmentStorage \_equipmentStorage;

public EquipmentLogic(IEquipmentStorage equipmentStorage)

{

\_equipmentStorage = equipmentStorage;

}

public List<EquipmentViewModel> Read(EquipmentBindingModel model)

{

if (model == null)

{

return \_equipmentStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<EquipmentViewModel> { \_equipmentStorage.GetElement(model) };

}

return \_equipmentStorage.GetFilteredList(model);

}

public void CreateOrUpdate(EquipmentBindingModel model)

{

var element = \_equipmentStorage.GetElement(new EquipmentBindingModel { EquipmentName = model.EquipmentName});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть комплектация с таким названием");

}

if (model.Id.HasValue)

{

\_equipmentStorage.Update(model);

}

else

{

\_equipmentStorage.Insert(model);

}

}

public void Delete(EquipmentBindingModel model)

{

var element = \_equipmentStorage.GetElement(new EquipmentBindingModel { Id = model.Id });

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_equipmentStorage.Delete(model);

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using System;

using System.Collections.Generic;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class ImplementerLogic

{

private readonly IImplementerStorage \_implementerStorage;

public ImplementerLogic(IImplementerStorage implementerStorage)

{

\_implementerStorage = implementerStorage;

}

public List<ImplementerViewModel> Read(ImplementerBindingModel model)

{

if (model == null)

{

return \_implementerStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<ImplementerViewModel> { \_implementerStorage.GetElement(model) };

}

return \_implementerStorage.GetFilteredList(model);

}

public void CreateOrUpdate(ImplementerBindingModel model)

{

var element = \_implementerStorage.GetElement(new ImplementerBindingModel

{

Name = model.Name

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть исполнитель с таким именем");

}

if (model.Id.HasValue)

{

\_implementerStorage.Update(model);

}

else

{

\_implementerStorage.Insert(model);

}

}

public void Delete(ImplementerBindingModel model)

{

var element = \_implementerStorage.GetElement(new ImplementerBindingModel { Id = model.Id });

if (element == null)

{

throw new Exception("Исполнитель не найден");

}

\_implementerStorage.Delete(model);

}

}

}

using MailKit.Net.Pop3;

using MailKit.Security;

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Mail;

using System.Text;

using System.Threading.Tasks;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.HelperModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class MailLogic

{

private static string smtpClientHost;

private static int smtpClientPort;

private static string mailLogin;

private static string mailPassword;

private readonly IMessageInfoStorage \_messageInfoStorage;

private readonly IClientStorage \_clientStorage;

public MailLogic(IMessageInfoStorage messageInfoStorage, IClientStorage clientStorage)

{

\_messageInfoStorage = messageInfoStorage;

\_clientStorage = clientStorage;

}

public List<MessageInfoViewModel> Read(MessageInfoBindingModel model)

{

if (model == null)

{

return \_messageInfoStorage.GetFullList();

}

return \_messageInfoStorage.GetFilteredList(model);

}

public void CreateOrder(MessageInfoBindingModel model)

{

var client = \_clientStorage.GetElement(new ClientBindingModel { Email = model.FromMailAddress });

model.ClientId = client?.Id;

\_messageInfoStorage.Insert(model);

}

public static void MailConfig(MailConfig config)

{

smtpClientHost = config.SmtpClientHost;

smtpClientPort = config.SmtpClientPort;

mailLogin = config.MailLogin;

mailPassword = config.MailPassword;

}

public static async void MailSendAsync(MailSendInfo info)

{

if (string.IsNullOrEmpty(smtpClientHost) || smtpClientPort == 0)

{

return;

}

if (string.IsNullOrEmpty(mailLogin) || string.IsNullOrEmpty(mailPassword))

{

return;

}

if (string.IsNullOrEmpty(info.MailAddress) || string.IsNullOrEmpty(info.Subject) || string.IsNullOrEmpty(info.Text))

{

return;

}

using (var objMailMessage = new MailMessage())

{

using (var objSmtpClient = new SmtpClient(smtpClientHost, smtpClientPort))

{

try

{

objMailMessage.From = new MailAddress(mailLogin);

objMailMessage.To.Add(new MailAddress(info.MailAddress));

objMailMessage.Subject = info.Subject;

objMailMessage.Body = info.Text;

objMailMessage.SubjectEncoding = Encoding.UTF8;

objMailMessage.BodyEncoding = Encoding.UTF8;

objSmtpClient.UseDefaultCredentials = false;

objSmtpClient.EnableSsl = true;

objSmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

objSmtpClient.Credentials = new NetworkCredential(mailLogin, mailPassword);

await Task.Run(() => objSmtpClient.Send(objMailMessage));

}

catch (Exception)

{

throw;

}

}

}

}

public static async void MailCheck(MailCheckInfo info)

{

if (string.IsNullOrEmpty(info.PopHost) || info.PopPort == 0)

{

return;

}

if (string.IsNullOrEmpty(mailLogin) || string.IsNullOrEmpty(mailPassword))

{

return;

}

if (info.Storage == null)

{

return;

}

using (var client = new Pop3Client())

{

await Task.Run(() =>

{

try

{

client.Connect(info.PopHost, info.PopPort, SecureSocketOptions.SslOnConnect);

client.Authenticate(mailLogin, mailPassword);

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

{

var message = client.GetMessage(i);

foreach (var mail in message.From.Mailboxes)

{

info.Storage.Insert(new MessageInfoBindingModel

{

ClientId = info.ClientStorage.GetElement(new ClientBindingModel { Email = mail.Address })?.Id,

DateDelivery = message.Date.DateTime,

MessageId = message.MessageId,

FromMailAddress = mail.Address,

Subject = message.Subject,

Body = message.TextBody

});

}

}

}

finally

{

client.Disconnect(true);

}

});

}

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Enums;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopBusinessLogic.HelperModels;

using System;

using System.Collections.Generic;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class OrderLogic

{

private readonly IOrderStorage \_orderStorage;

private readonly IClientStorage \_clientStorage;

private readonly object locker = new object();

public OrderLogic(IOrderStorage orderStorage, IClientStorage clientStorage)

{

\_orderStorage = orderStorage;

\_clientStorage = clientStorage;

}

public OrderLogic(IOrderStorage orderStorage)

{

\_orderStorage = orderStorage;

}

public List<OrderViewModel> Read(OrderBindingModel model)

{

if (model == null)

{

return \_orderStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<OrderViewModel> { \_orderStorage.GetElement(model) };

}

return \_orderStorage.GetFilteredList(model);

}

public void CreateOrder(CreateOrderBindingModel model)

{

\_orderStorage.Insert(new OrderBindingModel

{

EquipmentId = model.EquipmentId,

ClientId = model.ClientId,

Count = model.Count,

Sum = model.Sum,

DateCreate = DateTime.Now,

Status = OrderStatus.Принят

});

MailLogic.MailSendAsync(new MailSendInfo

{

MailAddress = \_clientStorage.GetElement(new ClientBindingModel { Id = model.ClientId })?.Email,

Subject = $"Новый заказ",

Text = $"Заказ от {DateTime.Now} на сумму {model.Sum:N2} принят."

});

}

public void TakeOrderInWork(ChangeStatusBindingModel model)

{

lock (locker)

{

var order = \_orderStorage.GetElement(new OrderBindingModel

{

Id = model.OrderId

});

if (order == null)

{

throw new Exception("Не найден заказ");

}

if (order.Status != OrderStatus.Принят)

{

throw new Exception("Заказ не в статусе \"Принят\"");

}

if (order.ImplementerId.HasValue)

{

throw new Exception("У заказа уже есть исполнитель");

}

\_orderStorage.Update(new OrderBindingModel

{

Id = order.Id,

EquipmentId = order.EquipmentId,

ClientId = order.ClientId,

ImplementerId = model.ImplementerId,

Count = order.Count,

Sum = order.Sum,

DateCreate = order.DateCreate,

Status = OrderStatus.Выполняется

});

MailLogic.MailSendAsync(new MailSendInfo

{

MailAddress = \_clientStorage.GetElement(new ClientBindingModel { Id = order.ClientId })?.Email,

Subject = $"Заказ №{order.Id}",

Text = $"Заказ №{order.Id} передан в работу."

});

}

}

public void FinishOrder(ChangeStatusBindingModel model)

{

var order = \_orderStorage.GetElement(new OrderBindingModel { Id = model.OrderId });

if (order == null)

{

throw new Exception("Заказ не найден");

}

if (order.Status != OrderStatus.Выполняется)

{

throw new Exception("Заказ не в статусе \"Выполняется\"");

}

\_orderStorage.Update(new OrderBindingModel

{

Id = order.Id,

EquipmentId = order.EquipmentId,

ClientId = order.ClientId,

ImplementerId = order.ImplementerId,

Count = order.Count,

Sum = order.Sum,

DateCreate = order.DateCreate,

DateImplement = DateTime.Now,

Status = OrderStatus.Готов

});

MailLogic.MailSendAsync(new MailSendInfo

{

MailAddress = \_clientStorage.GetElement(new ClientBindingModel { Id = order.ClientId })?.Email,

Subject = $"Заказ №{order.Id}",

Text = $"Заказ №{order.Id} выполнен."

});

}

public void PayOrder(ChangeStatusBindingModel model)

{

var order = \_orderStorage.GetElement(new OrderBindingModel { Id = model.OrderId });

if (order == null)

{

throw new Exception("Заказ не найден");

}

if (order.Status != OrderStatus.Готов)

{

throw new Exception("Заказ не в статусе \"Готов\"");

}

\_orderStorage.Update(new OrderBindingModel

{

Id = order.Id,

EquipmentId = order.EquipmentId,

ClientId = order.ClientId,

ImplementerId = order.ImplementerId,

Count = order.Count,

Sum = order.Sum,

DateCreate = order.DateCreate,

DateImplement = order.DateImplement,

Status = OrderStatus.Оплачен

});

MailLogic.MailSendAsync(new MailSendInfo

{

MailAddress = \_clientStorage.GetElement(new ClientBindingModel { Id = order.ClientId })?.Email,

Subject = $"Заказ №{order.Id}",

Text = $"Заказ №{order.Id} оплачен."

});

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.HelperModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using System.Linq;

using SecureShopBusinessLogic.BusinessLogic;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class ReportLogic

{

private readonly IDeviceStorage \_deviceStorage; private readonly IEquipmentStorage \_equipmentStorage; private readonly IOrderStorage \_orderStorage;

public ReportLogic(IEquipmentStorage equipmentStorage, IDeviceStorage deviceStorage, IOrderStorage orderStorage)

{

\_equipmentStorage = equipmentStorage;

\_deviceStorage = deviceStorage;

\_orderStorage = orderStorage;

}

/// <summary>

/// Получение списка компонент с указанием, в каких устройствах используются

/// </summary>

/// <returns></returns>

public List<ReportEquipmentDeviceViewModel> GetEquipmentDevice()

{

var devices = \_deviceStorage.GetFullList();

var equipments = \_equipmentStorage.GetFullList();

var list = new List<ReportEquipmentDeviceViewModel>();

foreach (var equipment in equipments)

{

var record = new ReportEquipmentDeviceViewModel

{

EquipmentName = equipment.EquipmentName,

Devices = new List<Tuple<string, int>>(),

TotalCount = 0

};

foreach (var device in devices)

{

if (equipment.EquipmentDevices.ContainsKey(device.Id))

{

record.Devices.Add(new Tuple<string, int>(device.DeviceName, equipment.EquipmentDevices[device.Id].Item2));

record.TotalCount += equipment.EquipmentDevices[device.Id].Item2;

}

}

list.Add(record);

}

return list;

}

/// <summary>

/// Получение списка заказов за определенный период

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

public List<ReportOrdersViewModel> GetOrders(ReportBindingModel model)

{

return \_orderStorage.GetFilteredList(new OrderBindingModel

{

DateFrom = model.DateFrom,

DateTo = model.DateTo

})

.Select(x => new ReportOrdersViewModel

{

DateCreate = x.DateCreate,

EquipmentName = x.EquipmentName,

Count = x.Count,

Sum = x.Sum,

Status = x.Status

})

.ToList();

}

/// <summary>

/// Сохранение компонент в файл-Word

/// </summary>

/// <param name="model"></param>

public void SaveDevicesToWordFile(ReportBindingModel model)

{

SaveToWord.CreateDoc(new WordInfo

{

FileName = model.FileName,

Title = "Список устройств",

Equipments = \_equipmentStorage.GetFullList()

});

}

/// <summary>

/// Сохранение компонент с указаеним продуктов в файл-Excel

/// </summary>

/// <param name="model"></param>

public void SaveEquipmentDeviceToExcelFile(ReportBindingModel model)

{

SaveToExcel.CreateDoc(new ExcelInfo

{

FileName = model.FileName,

Title = "Список устройств-комплектаций",

EquipmentDevices = GetEquipmentDevice()

});

}

/// <summary>

/// Сохранение заказов в файл-Pdf

/// </summary>

/// <param name="model"></param>

public void SaveOrdersToPdfFile(ReportBindingModel model)

{

SaveToPdf.CreateDoc(new PdfInfo

{

FileName = model.FileName,

Title = "Список заказов",

DateFrom = model.DateFrom.Value,

DateTo = model.DateTo.Value,

Orders = GetOrders(model)

});

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.HelperModels;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Office2010.Excel;

using DocumentFormat.OpenXml.Office2013.Excel;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Spreadsheet;

using System.Linq;

namespace SecureShopBusinessLogic.BusinessLogics

{

static class SaveToExcel

{

public static void CreateDoc(ExcelInfo info)

{

using (SpreadsheetDocument spreadsheetDocument = SpreadsheetDocument.Create(info.FileName, SpreadsheetDocumentType.Workbook))

{

// Создаем книгу (в ней хранятся листы)

WorkbookPart workbookpart = spreadsheetDocument.AddWorkbookPart(); workbookpart.Workbook = new Workbook();

CreateStyles(workbookpart);

// Получаем/создаем хранилище текстов для книги

SharedStringTablePart shareStringPart =

spreadsheetDocument.WorkbookPart.GetPartsOfType<SharedStringTablePart>().Count() > 0

?

spreadsheetDocument.WorkbookPart.GetPartsOfType<SharedStringTablePart>().First()

:

spreadsheetDocument.WorkbookPart.AddNewPart<SharedStringTablePart>();

// Создаем SharedStringTable, если его нет

if (shareStringPart.SharedStringTable == null)

{

shareStringPart.SharedStringTable = new SharedStringTable();

}

// Создаем лист в книгу

WorksheetPart worksheetPart = workbookpart.AddNewPart<WorksheetPart>(); worksheetPart.Worksheet = new Worksheet(new SheetData());

// Добавляем лист в книгу

Sheets sheets =

spreadsheetDocument.WorkbookPart.Workbook.AppendChild<Sheets>(new Sheets()); Sheet sheet = new Sheet()

{

Id = spreadsheetDocument.WorkbookPart.GetIdOfPart(worksheetPart),

SheetId = 1,

Name = "Лист"

};

sheets.Append(sheet);

InsertCellInWorksheet(new ExcelCellParameters

{

Worksheet = worksheetPart.Worksheet,

ShareStringPart = shareStringPart,

ColumnName = "A",

RowIndex = 1,

Text = info.Title,

StyleIndex = 2U

});

MergeCells(new ExcelMergeParameters

{

Worksheet = worksheetPart.Worksheet,

CellFromName = "A1",

CellToName = "C1"

});

uint rowIndex = 2;

foreach (var eq in info.EquipmentDevices)

{

InsertCellInWorksheet(new ExcelCellParameters

{

Worksheet = worksheetPart.Worksheet,

ShareStringPart = shareStringPart,

ColumnName = "A",

RowIndex = rowIndex,

Text = eq.EquipmentName,

StyleIndex = 0U

});

rowIndex++;

foreach (var equipment in eq.Devices)

{

InsertCellInWorksheet(new ExcelCellParameters

{

Worksheet = worksheetPart.Worksheet,

ShareStringPart = shareStringPart,

ColumnName = "B",

RowIndex = rowIndex,

Text = equipment.Item1,

StyleIndex = 1U

});

InsertCellInWorksheet(new ExcelCellParameters

{

Worksheet = worksheetPart.Worksheet,

ShareStringPart = shareStringPart,

ColumnName = "C",

RowIndex = rowIndex,

Text = equipment.Item2.ToString(),

StyleIndex = 1U

});

rowIndex++;

}

InsertCellInWorksheet(new ExcelCellParameters

{

Worksheet = worksheetPart.Worksheet,

ShareStringPart = shareStringPart,

ColumnName = "C",

RowIndex = rowIndex,

Text = eq.TotalCount.ToString(),

StyleIndex = 0U

});

rowIndex++;

}

workbookpart.Workbook.Save();

}

}

/// <summary>

/// Настройка стилей для файла

/// </summary>

/// <param name="workbookpart"></param>

private static void CreateStyles(WorkbookPart workbookpart)

{

WorkbookStylesPart sp = workbookpart.AddNewPart<WorkbookStylesPart>(); sp.Stylesheet = new Stylesheet();

Fonts fonts = new Fonts() { Count = (UInt32Value)2U, KnownFonts = true }; Font fontUsual = new Font();

fontUsual.Append(new FontSize() { Val = 12D });

fontUsual.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Theme = (UInt32Value)1U

});

fontUsual.Append(new FontName() { Val = "Times New Roman" }); fontUsual.Append(new FontFamilyNumbering() { Val = 2 }); fontUsual.Append(new FontScheme() { Val = FontSchemeValues.Minor });

Font fontTitle = new Font(); fontTitle.Append(new Bold()); fontTitle.Append(new FontSize() { Val = 14D });

fontTitle.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Theme

= (UInt32Value)1U

});

fontTitle.Append(new FontName() { Val = "Times New Roman" }); fontTitle.Append(new FontFamilyNumbering() { Val = 2 }); fontTitle.Append(new FontScheme() { Val = FontSchemeValues.Minor });

fonts.Append(fontUsual); fonts.Append(fontTitle);

Fills fills = new Fills() { Count = (UInt32Value)2U }; Fill fill1 = new Fill();

fill1.Append(new PatternFill() { PatternType = PatternValues.None });

Fill fill2 = new Fill();

fill2.Append(new PatternFill() { PatternType = PatternValues.Gray125 });

fills.Append(fill1); fills.Append(fill2);

Borders borders = new Borders() { Count = (UInt32Value)2U };

Border borderNoBorder = new Border(); borderNoBorder.Append(new LeftBorder()); borderNoBorder.Append(new RightBorder()); borderNoBorder.Append(new TopBorder()); borderNoBorder.Append(new BottomBorder()); borderNoBorder.Append(new DiagonalBorder());

Border borderThin = new Border();

LeftBorder leftBorder = new LeftBorder() { Style = BorderStyleValues.Thin }; leftBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = (UInt32Value)64U

});

RightBorder rightBorder = new RightBorder()

{

Style = BorderStyleValues.Thin

};

rightBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = (UInt32Value)64U

});

TopBorder topBorder = new TopBorder() { Style = BorderStyleValues.Thin }; topBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = (UInt32Value)64U

});

BottomBorder bottomBorder = new BottomBorder() { Style = BorderStyleValues.Thin };

bottomBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = (UInt32Value)64U });

borderThin.Append(leftBorder); borderThin.Append(rightBorder); borderThin.Append(topBorder); borderThin.Append(bottomBorder); borderThin.Append(new DiagonalBorder());

borders.Append(borderNoBorder); borders.Append(borderThin);

CellStyleFormats cellStyleFormats = new CellStyleFormats() { Count = (UInt32Value)1U };

CellFormat cellFormatStyle = new CellFormat() { NumberFormatId = (UInt32Value)0U, FontId = (UInt32Value)0U, FillId = (UInt32Value)0U, BorderId = (UInt32Value)0U };

cellStyleFormats.Append(cellFormatStyle);

CellFormats cellFormats = new CellFormats() { Count = (UInt32Value)3U }; CellFormat cellFormatFont = new CellFormat()

{

NumberFormatId =

(UInt32Value)0U,

FontId = (UInt32Value)0U,

FillId = (UInt32Value)0U,

BorderId = (UInt32Value)0U,

FormatId = (UInt32Value)0U,

ApplyFont = true

};

CellFormat cellFormatFontAndBorder = new CellFormat() { NumberFormatId = (UInt32Value)0U, FontId = (UInt32Value)0U, FillId = (UInt32Value)0U, BorderId = (UInt32Value)1U, FormatId = (UInt32Value)0U, ApplyFont = true, ApplyBorder = true };

CellFormat cellFormatTitle = new CellFormat() { NumberFormatId = (UInt32Value)0U, FontId = (UInt32Value)1U, FillId = (UInt32Value)0U, BorderId = (UInt32Value)0U, FormatId = (UInt32Value)0U, Alignment = new Alignment() { Vertical = VerticalAlignmentValues.Center, WrapText = true, Horizontal = HorizontalAlignmentValues.Center }, ApplyFont = true };

cellFormats.Append(cellFormatFont); cellFormats.Append(cellFormatFontAndBorder); cellFormats.Append(cellFormatTitle);

CellStyles cellStyles = new CellStyles() { Count = (UInt32Value)1U }; cellStyles.Append(new CellStyle()

{

Name = "Normal",

FormatId =

(UInt32Value)0U,

BuiltinId = (UInt32Value)0U

});

DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats differentialFormats = new DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats() { Count = (UInt32Value)0U };

TableStyles tableStyles = new TableStyles() { Count = (UInt32Value)0U, DefaultTableStyle = "TableStyleMedium2", DefaultPivotStyle = "PivotStyleLight16" };

StylesheetExtensionList stylesheetExtensionList = new StylesheetExtensionList();

StylesheetExtension stylesheetExtension1 = new StylesheetExtension() { Uri = "{EB79DEF2-80B8-43e5-95BD-54CBDDF9020C}" };

stylesheetExtension1.AddNamespaceDeclaration("x14", "http://schemas.microsoft.com/office/spreadsheetml/2009/9/main");

stylesheetExtension1.Append(new SlicerStyles() { DefaultSlicerStyle = "SlicerStyleLight1" });

StylesheetExtension stylesheetExtension2 = new StylesheetExtension() { Uri = "{9260A510-F301-46a8-8635-F512D64BE5F5}" };

stylesheetExtension2.AddNamespaceDeclaration("x15", "http://schemas.microsoft.com/office/spreadsheetml/2010/11/main");

stylesheetExtension2.Append(new TimelineStyles() { DefaultTimelineStyle = "TimeSlicerStyleLight1" });

stylesheetExtensionList.Append(stylesheetExtension1); stylesheetExtensionList.Append(stylesheetExtension2);

sp.Stylesheet.Append(fonts); sp.Stylesheet.Append(fills); sp.Stylesheet.Append(borders); sp.Stylesheet.Append(cellStyleFormats); sp.Stylesheet.Append(cellFormats); sp.Stylesheet.Append(cellStyles); sp.Stylesheet.Append(differentialFormats); sp.Stylesheet.Append(tableStyles); sp.Stylesheet.Append(stylesheetExtensionList);

}

/// <summary>

/// Добавляем новую ячейку в лист

/// </summary>

/// <param name="cellParameters"></param>

private static void InsertCellInWorksheet(ExcelCellParameters cellParameters)

{

SheetData sheetData = cellParameters.Worksheet.GetFirstChild<SheetData>();

// Ищем строку, либо добавляем ее

Row row;

if (sheetData.Elements<Row>().Where(r => r.RowIndex == cellParameters.RowIndex).Count() != 0)

{

row = sheetData.Elements<Row>().Where(r => r.RowIndex == cellParameters.RowIndex).First();

}

else

{

row = new Row() { RowIndex = cellParameters.RowIndex }; sheetData.Append(row);

}

// Ищем нужную ячейку

Cell cell;

if (row.Elements<Cell>().Where(c => c.CellReference.Value == cellParameters.CellReference).Count() > 0)

{

cell = row.Elements<Cell>().Where(c => c.CellReference.Value == cellParameters.CellReference).First();

}

else

{

// Все ячейки должны быть последовательно друг за другом расположены

// нужно определить, после какой вставлять

Cell refCell = null;

foreach (Cell rowCell in row.Elements<Cell>())

{

if (string.Compare(rowCell.CellReference.Value, cellParameters.CellReference, true) > 0)

{

refCell = rowCell; break;

}

}

Cell newCell = new Cell()

{

CellReference = cellParameters.CellReference

};

row.InsertBefore(newCell, refCell);

cell = newCell;

}

// вставляем новый текст

cellParameters.ShareStringPart.SharedStringTable.AppendChild(new

SharedStringItem(new Text(cellParameters.Text))); cellParameters.ShareStringPart.SharedStringTable.Save();

cell.CellValue = new CellValue((cellParameters.ShareStringPart.SharedStringTable.Elements<SharedStringItem>().Count() - 1).ToString());

cell.DataType = new EnumValue<CellValues>(CellValues.SharedString); cell.StyleIndex = cellParameters.StyleIndex;

}

/// <summary>

/// Объединение ячеек

/// </summary>

/// <param name="mergeParameters"></param>

private static void MergeCells(ExcelMergeParameters mergeParameters)

{

MergeCells mergeCells;

if (mergeParameters.Worksheet.Elements<MergeCells>().Count() > 0)

{

mergeCells = mergeParameters.Worksheet.Elements<MergeCells>().First();

}

else

{

mergeCells = new MergeCells();

if (mergeParameters.Worksheet.Elements<CustomSheetView>().Count() > 0)

{

mergeParameters.Worksheet.InsertAfter(mergeCells, mergeParameters.Worksheet.Elements<CustomSheetView>().First());

}

else

{

mergeParameters.Worksheet.InsertAfter(mergeCells, mergeParameters.Worksheet.Elements<SheetData>().First());

}

}

MergeCell mergeCell = new MergeCell()

{

Reference = new StringValue(mergeParameters.Merge)

};

mergeCells.Append(mergeCell);

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.HelperModels;

using MigraDoc.DocumentObjectModel;

using MigraDoc.DocumentObjectModel.Tables;

using MigraDoc.Rendering;

namespace SecureShopBusinessLogic.BusinessLogic

{

class SaveToPdf

{

public static void CreateDoc(PdfInfo info)

{

Document document = new Document();

DefineStyles(document);

Section section = document.AddSection();

Paragraph paragraph = section.AddParagraph(info.Title);

paragraph.Format.SpaceAfter = "1cm";

paragraph.Format.Alignment = ParagraphAlignment.Center;

paragraph.Style = "NormalTitle";

paragraph = section.AddParagraph($"с {info.DateFrom.ToShortDateString()} по { info.DateTo.ToShortDateString()}");

paragraph.Format.SpaceAfter = "1cm";

paragraph.Format.Alignment = ParagraphAlignment.Center;

paragraph.Style = "Normal";

var table = document.LastSection.AddTable();

List<string> columns = new List<string> { "3cm", "6cm", "3cm", "2cm", "3cm"

};

foreach (var elem in columns)

{

table.AddColumn(elem);

}

CreateRow(new PdfRowParameters

{

Table = table,

Texts = new List<string> { "Дата заказа", "Комплект", "Количество", "Сумма", "Статус" },

Style = "NormalTitle",

ParagraphAlignment = ParagraphAlignment.Center

});

foreach (var order in info.Orders)

{

CreateRow(new PdfRowParameters

{

Table = table,

Texts = new List<string> { order.DateCreate.ToShortDateString(), order.EquipmentName, order.Count.ToString(), order.Sum.ToString(), order.Status.ToString() },

Style = "Normal",

ParagraphAlignment = ParagraphAlignment.Left

});

}

PdfDocumentRenderer renderer = new PdfDocumentRenderer(true, PdfSharp.Pdf.PdfFontEmbedding.Always)

{

Document = document

};

renderer.RenderDocument();

renderer.PdfDocument.Save(info.FileName);

}

/// <summary>

/// Создание стилей для документа

/// </summary>

/// <param name="document"></param>

private static void DefineStyles(Document document)

{

Style style = document.Styles["Normal"];

style.Font.Name = "Times New Roman";

style.Font.Size = 14;

style = document.Styles.AddStyle("NormalTitle", "Normal");

style.Font.Bold = true;

}

/// <summary>

/// Создание и заполнение строки

/// </summary>

/// <param name="rowParameters"></param>

private static void CreateRow(PdfRowParameters rowParameters)

{

Row row = rowParameters.Table.AddRow();

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

{

FillCell(new PdfCellParameters

{

Cell = row.Cells[i],

Text = rowParameters.Texts[i],

Style = rowParameters.Style,

BorderWidth = 0.5,

ParagraphAlignment = rowParameters.ParagraphAlignment

});

}

}

/// <summary>

/// Заполнение ячейки

/// </summary>

/// <param name="cellParameters"></param>

private static void FillCell(PdfCellParameters cellParameters)

{

cellParameters.Cell.AddParagraph(cellParameters.Text);

if (!string.IsNullOrEmpty(cellParameters.Style))

{

cellParameters.Cell.Style = cellParameters.Style;

}

cellParameters.Cell.Borders.Left.Width = cellParameters.BorderWidth;

cellParameters.Cell.Borders.Right.Width = cellParameters.BorderWidth;

cellParameters.Cell.Borders.Top.Width = cellParameters.BorderWidth;

cellParameters.Cell.Borders.Bottom.Width = cellParameters.BorderWidth;

cellParameters.Cell.Format.Alignment = cellParameters.ParagraphAlignment;

cellParameters.Cell.VerticalAlignment = VerticalAlignment.Center;

}

}

}

using SecureShopBusinessLogic.HelperModels;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using System.Collections.Generic;

namespace SecureShopBusinessLogic.BusinessLogics

{

static class SaveToWord

{

/// <summary>

/// Создание документа

/// </summary>

/// <param name="info"></param>

public static void CreateDoc(WordInfo info)

{

using (WordprocessingDocument wordDocument = WordprocessingDocument.Create(info.FileName, WordprocessingDocumentType.Document))

{

MainDocumentPart mainPart = wordDocument.AddMainDocumentPart(); mainPart.Document = new Document();

Body docBody = mainPart.Document.AppendChild(new Body());

docBody.AppendChild(CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { (info.Title, new WordTextProperties { Bold = true, Size = "24", }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationValues = JustificationValues.Center

}

}));

foreach (var equipment in info.Equipments)

{

docBody.AppendChild(CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ($"{equipment.EquipmentName}: ", new WordTextProperties { Size = "24", Bold = true }), (equipment.Price.ToString(), new WordTextProperties { Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationValues = JustificationValues.Both

}

}));

}

docBody.AppendChild(CreateSectionProperties());

wordDocument.MainDocumentPart.Document.Save();

}

}

/// <summary>

/// Настройки страницы

/// </summary>

/// <returns></returns>

private static SectionProperties CreateSectionProperties()

{

SectionProperties properties = new SectionProperties();

PageSize pageSize = new PageSize

{

Orient = PageOrientationValues.Portrait

};

properties.AppendChild(pageSize);

return properties;

}

/// <summary>

/// Создание абзаца с текстом

/// </summary>

/// <param name="paragraph"></param>

/// <returns></returns>

private static Paragraph CreateParagraph(WordParagraph paragraph)

{

if (paragraph != null)

{

Paragraph docParagraph = new Paragraph();

docParagraph.AppendChild(CreateParagraphProperties(paragraph.TextProperties)); foreach (var run in paragraph.Texts)

{

Run docRun = new Run();

RunProperties properties = new RunProperties(); properties.AppendChild(new FontSize { Val = run.Item2.Size }); if (run.Item2.Bold)

{

properties.AppendChild(new Bold());

}

docRun.AppendChild(properties);

docRun.AppendChild(new Text { Text = run.Item1, Space = SpaceProcessingModeValues.Preserve });

docParagraph.AppendChild(docRun);

}

return docParagraph;

}

return null;

}

/// <summary>

/// Задание форматирования для абзаца

/// </summary>

/// <param name="paragraphProperties"></param>

/// <returns></returns>

private static ParagraphProperties CreateParagraphProperties(WordTextProperties paragraphProperties)

{

if (paragraphProperties != null)

{

ParagraphProperties properties = new ParagraphProperties();

properties.AppendChild(new Justification()

{

Val = paragraphProperties.JustificationValues

});

properties.AppendChild(new SpacingBetweenLines

{

LineRule = LineSpacingRuleValues.Auto

});

properties.AppendChild(new Indentation()); ParagraphMarkRunProperties paragraphMarkRunProperties = new

ParagraphMarkRunProperties();

if (!string.IsNullOrEmpty(paragraphProperties.Size))

{

paragraphMarkRunProperties.AppendChild(new FontSize { Val = paragraphProperties.Size });

}

properties.AppendChild(paragraphMarkRunProperties);

return properties;

}

return null;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.IO.Compression;

using System.Linq;

using System.Reflection;

using System.Runtime.Serialization.Json;

using System.Xml.Linq;

namespace SecureShopBusinessLogic.BusinessLogics

{

public abstract class SecureShopBackUpAbstractLogic

{

public void CreateArchive(string folderName)

{

try

{

DirectoryInfo dirInfo = new DirectoryInfo(folderName);

if (dirInfo.Exists)

{

foreach (FileInfo file in dirInfo.GetFiles())

{

file.Delete();

}

}

string fileName = $"{folderName}.zip";

if (File.Exists(fileName))

{

File.Delete(fileName);

}

// берем сборку, чтобы от нее создавать объекты

Assembly assem = GetAssembly();

// вытаскиваем список классов для сохранения

var dbsets = GetFullList();

// берем метод для сохранения (из базвого абстрактного класса)

MethodInfo method = GetType().BaseType.GetTypeInfo().GetDeclaredMethod("SaveToFileXml");

foreach (var set in dbsets)

{

// создаем объект из класса для сохранения

var elem = assem.CreateInstance(set.PropertyType.GenericTypeArguments[0].FullName);

// генерируем метод, исходя из класса

MethodInfo generic = method.MakeGenericMethod(elem.GetType());

// вызываем метод на выполнение

generic.Invoke(this, new object[] { folderName });

}

// архивируем

ZipFile.CreateFromDirectory(folderName, fileName);

// удаляем папку

dirInfo.Delete(true);

}

catch (Exception)

{

// делаем проброс

throw;

}

}

private void SaveToFileXml<T>(string folderName) where T : class, new()

{

var records = GetList<T>();

T obj = new T();

var typeName = obj.GetType().Name;

if (records != null)

{

var root = new XElement(typeName + 's');

foreach (var record in records)

{

var elem = new XElement(typeName);

foreach (var mem in obj.GetType().GetMembers()

.Where(rec => rec.MemberType != MemberTypes.Method &&

rec.MemberType != MemberTypes.Constructor &&

!rec.ToString().Contains(".Models.")))

{

elem.Add(new XElement(mem.Name, record.GetType().GetProperty(mem.Name)?.GetValue(record) ?? "null"));

}

root.Add(elem);

}

XDocument xDocument = new XDocument(root);

xDocument.Save(string.Format("{0}/{1}.xml", folderName, typeName));

}

}

protected abstract Assembly GetAssembly();

protected abstract List<PropertyInfo> GetFullList();

protected abstract List<T> GetList<T>() where T : class, new();

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

namespace SecureShopBusinessLogic.BusinessLogics

{

public class WorkModeling

{

private readonly IImplementerStorage \_implementerStorage;

private readonly IOrderStorage \_orderStorage;

private readonly OrderLogic \_orderLogic;

private readonly Random rnd;

public WorkModeling(IImplementerStorage implementerStorage, IOrderStorage

orderStorage, OrderLogic orderLogic)

{

this.\_implementerStorage = implementerStorage;

this.\_orderStorage = orderStorage;

this.\_orderLogic = orderLogic;

rnd = new Random(1000);

}

public void DoWork()

{

var implementers = \_implementerStorage.GetFullList();

var orders = \_orderStorage.GetFilteredList(new OrderBindingModel

{

FreeOrders = true

});

foreach (var implementer in implementers)

{

WorkerWorkAsync(implementer, orders);

}

}

private async void WorkerWorkAsync(ImplementerViewModel implementer,

List<OrderViewModel> orders)

{

// ищем заказы, которые уже в работе (вдруг исполнителя прервали)

var runOrders = await Task.Run(() => \_orderStorage.GetFilteredList(new

OrderBindingModel

{ ImplementerId = implementer.Id }));

foreach (var order in runOrders)

{

// делаем работу заново

Thread.Sleep(implementer.WorkingTime \* rnd.Next(1, 5) \* order.Count);

\_orderLogic.FinishOrder(new ChangeStatusBindingModel

{

OrderId = order.Id

});

// отдыхаем

Thread.Sleep(implementer.PauseTime);

}

await Task.Run(() =>

{

foreach (var order in orders)

{

// пытаемся назначить заказ на исполнителя

try

{

\_orderLogic.TakeOrderInWork(new ChangeStatusBindingModel

{

OrderId = order.Id,

ImplementerId = implementer.Id

});

// делаем работу

Thread.Sleep(implementer.WorkingTime \* rnd.Next(1, 5) \*

order.Count);

\_orderLogic.FinishOrder(new ChangeStatusBindingModel

{

OrderId =

order.Id

});

// отдыхаем

Thread.Sleep(implementer.PauseTime);

}

catch (Exception) { }

}

});

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.Enums

{

public enum OrderStatus

{

Принят = 0,

Выполняется = 1,

Готов = 2 ,

Оплачен = 3

}

}

using System;

using System.Collections.Generic;

using System.Text;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Spreadsheet;

namespace SecureShopBusinessLogic.HelperModels

{

class ExcelCellParameters

{

public Worksheet Worksheet { get; set; }

public string ColumnName { get; set; }

public uint RowIndex { get; set; }

public UInt32Value StyleIndex { get; set; }

public string Text { get; set; }

public SharedStringTablePart ShareStringPart { get; set; }

public string CellReference => $"{ColumnName}{RowIndex}";

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.HelperModels

{

class ExcelInfo

{

public string FileName { get; set; }

public string Title { get; set; }

public List<ReportEquipmentDeviceViewModel> EquipmentDevices { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using DocumentFormat.OpenXml.Spreadsheet;

namespace SecureShopBusinessLogic.HelperModels

{

class ExcelMergeParameters

{

public Worksheet Worksheet { get; set; }

public string CellFromName { get; set; }

public string CellToName { get; set; }

public string Merge => $"{CellFromName}:{CellToName}";

}

}

using SecureShopBusinessLogic.Interfaces;

namespace SecureShopBusinessLogic.HelperModels

{

public class MailCheckInfo

{

public string PopHost { get; set; }

public int PopPort { get; set; }

public IMessageInfoStorage Storage { get; set; }

public IClientStorage ClientStorage { get; set; }

}

}

using SecureShopBusinessLogic.Interfaces;

namespace SecureShopBusinessLogic.HelperModels

{

public class MailConfig

{

public string SmtpClientHost { get; set; }

public int SmtpClientPort { get; set; }

public string MailLogin { get; set; }

public string MailPassword { get; set; }

}

}

namespace SecureShopBusinessLogic.HelperModels

{

public class MailSendInfo

{

public string MailAddress { get; set; }

public string Subject { get; set; }

public string Text { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using MigraDoc.DocumentObjectModel;

using MigraDoc.DocumentObjectModel.Tables;

namespace SecureShopBusinessLogic.HelperModels

{

class PdfCellParameters

{

public Cell Cell { get; set; }

public string Text { get; set; }

public string Style { get; set; }

public ParagraphAlignment ParagraphAlignment { get; set; }

public Unit BorderWidth { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.HelperModels

{

class PdfInfo

{

public string FileName { get; set; }

public string Title { get; set; }

public DateTime DateFrom { get; set; }

public DateTime DateTo { get; set; }

public List<ReportOrdersViewModel> Orders { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using MigraDoc.DocumentObjectModel;

using MigraDoc.DocumentObjectModel.Tables;

namespace SecureShopBusinessLogic.HelperModels

{

class PdfRowParameters

{

public Table Table { get; set; }

public List<string> Texts { get; set; }

public string Style { get; set; }

public ParagraphAlignment ParagraphAlignment { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.HelperModels

{

class WordInfo

{

public string FileName { get; set; }

public string Title { get; set; }

public List<EquipmentViewModel> Equipments { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.HelperModels

{

class WordParagraph

{

public List<(string, WordTextProperties)> Texts { get; set; }

public WordTextProperties TextProperties { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using DocumentFormat.OpenXml.Wordprocessing;

namespace SecureShopBusinessLogic.HelperModels

{

class WordParagraphProperties

{

public string Size { get; set; }

public bool Bold { get; set; }

public JustificationValues JustificationValues { get; set; }

}

}

using DocumentFormat.OpenXml.Wordprocessing;

namespace SecureShopBusinessLogic.HelperModels

{

class WordTextProperties

{

public string Size { get; set; }

public bool Bold { get; set; }

public JustificationValues JustificationValues { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.Interfaces

{

public interface IClientStorage

{

List<ClientViewModel> GetFullList();

List<ClientViewModel> GetFilteredList(ClientBindingModel model);

ClientViewModel GetElement(ClientBindingModel model);

void Insert(ClientBindingModel model);

void Update(ClientBindingModel model);

void Delete(ClientBindingModel model);

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.Interfaces

{

public interface IDeviceStorage

{

List<DeviceViewModel> GetFullList();

List<DeviceViewModel> GetFilteredList(DeviceBindingModel model);

DeviceViewModel GetElement(DeviceBindingModel model);

void Insert(DeviceBindingModel model);

void Update(DeviceBindingModel model);

void Delete(DeviceBindingModel model);

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.Interfaces

{

public interface IEquipmentStorage

{

List<EquipmentViewModel> GetFullList();

List<EquipmentViewModel> GetFilteredList(EquipmentBindingModel model);

EquipmentViewModel GetElement(EquipmentBindingModel model);

void Insert(EquipmentBindingModel model);

void Update(EquipmentBindingModel model);

void Delete(EquipmentBindingModel model);

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

using System.Collections.Generic;

namespace SecureShopBusinessLogic.Interfaces

{

public interface IImplementerStorage

{

List<ImplementerViewModel> GetFullList();

List<ImplementerViewModel> GetFilteredList(ImplementerBindingModel model);

ImplementerViewModel GetElement(ImplementerBindingModel model);

void Insert(ImplementerBindingModel model);

void Update(ImplementerBindingModel model);

void Delete(ImplementerBindingModel model);

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

using System.Collections.Generic;

namespace SecureShopBusinessLogic.Interfaces

{

public interface IMessageInfoStorage

{

List<MessageInfoViewModel> GetFullList();

List<MessageInfoViewModel> GetFilteredList(MessageInfoBindingModel model);

void Insert(MessageInfoBindingModel model);

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopBusinessLogic.Interfaces

{

public interface IOrderStorage

{

List<OrderViewModel> GetFullList();

List<OrderViewModel> GetFilteredList(OrderBindingModel model);

OrderViewModel GetElement(OrderBindingModel model);

void Insert(OrderBindingModel model);

void Update(OrderBindingModel model);

void Delete(OrderBindingModel model);

}

}

using SecureShopBusinessLogic.Attributes;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.ViewModels

{

[DataContract]

public class ClientViewModel

{

[Column(title: "Номер", width: 50)]

[DataMember]

public int Id { get; set; }

[Column(title: "ФИО клиента", width: 100)]

[DataMember]

public string ClientFIO { get; set; }

[Column(title: "Email", gridViewAutoSize: GridViewAutoSize.Fill)]

[DataMember]

public string Email { get; set; }

[Column(title: "Пароль", width: 100)]

[DataMember]

public string Password { get; set; }

}

}

using SecureShopBusinessLogic.Attributes;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.ViewModels

{

public class DeviceViewModel

{

[Column(title: "Номер", width: 50)]

[DataMember]

public int Id { get; set; }

[Column(title: "Название", gridViewAutoSize: GridViewAutoSize.Fill)]

[DataMember]

public string DeviceName { get; set; }

}

}

using SecureShopBusinessLogic.Attributes;

using System.Collections.Generic;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.ViewModels

{

public class EquipmentViewModel

{

[Column(title: "Номер", width: 50)]

[DataMember]

public int Id { get; set; }

[Column(title: "Название", gridViewAutoSize: GridViewAutoSize.Fill)]

[DataMember]

public string EquipmentName { get; set; }

[Column(title: "Цена", width: 100)]

[DataMember]

public decimal Price { get; set; }

[DataMember]

public Dictionary<int, (string, int)> EquipmentDevices { get; set; }

}

}

using SecureShopBusinessLogic.Attributes;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.ViewModels

{

public class ImplementerViewModel

{

[Column(title: "Номер", width: 50)]

[DataMember]

public int Id { get; set; }

[Column(title: "ФИО исполнителя", gridViewAutoSize: GridViewAutoSize.Fill)]

[DataMember]

public string Name { get; set; }

[Column(title: "Время на заказ", width: 100)]

[DataMember]

public int WorkingTime { get; set; }

[Column(title:"Время на перерыв", width: 100)]

[DataMember]

public int PauseTime { get; set; }

}

}

using SecureShopBusinessLogic.Attributes;

using System;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.ViewModels

{

public class MessageInfoViewModel

{

[Column(title: "Номер", width: 50)]

[DataMember]

public string MessageId { get; set; }

[Column(title: "Отправитель", width: 100)]

[DataMember]

public string SenderName { get; set; }

[Column(title: "Дата письма", width: 100)]

[DataMember]

public DateTime DateDelivery { get; set; }

[Column(title: "Заголовок", width: 100)]

[DataMember]

public string Subject { get; set; }

[Column(title: "Текст", gridViewAutoSize: GridViewAutoSize.Fill)]

[DataMember]

public string Body { get; set; }

}

}

using SecureShopBusinessLogic.Attributes;

using SecureShopBusinessLogic.Enums;

using System;

using System.Runtime.Serialization;

namespace SecureShopBusinessLogic.ViewModels

{

[DataContract]

public class OrderViewModel

{

[Column(title: "Номер", width: 100)]

[DataMember]

public int Id { get; set; }

[DataMember]

public int ClientId { get; set; }

[DataMember]

public int EquipmentId { get; set; }

[DataMember]

public int? ImplementerId { get; set; }

[Column(title: "Клиент", width: 150)]

[DataMember]

public string ClientFIO { get; set; }

[Column(title: "Комплект", width: 150)]

[DataMember]

public string EquipmentName { get; set; }

[Column(title: "Исполнитель", width: 150)]

[DataMember]

public string ImplementerName { get; set; }

[Column(title: "Количество", width: 100)]

[DataMember]

public int Count { get; set; }

[Column(title: "Сумма", width: 50)]

[DataMember]

public decimal Sum { get; set; }

[Column(title: "Статус", width: 100)]

[DataMember]

public OrderStatus Status { get; set; }

[Column(title: "Дата создания", width: 150)]

[DataMember]

public DateTime DateCreate { get; set; }

[Column(title: "Дата выполнения", width: 150)]

[DataMember]

public DateTime? DateImplement { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopBusinessLogic.ViewModels

{

public class ReportEquipmentDeviceViewModel

{

public string EquipmentName { get; set; }

public int TotalCount { get; set; }

public List<Tuple<string, int>> Devices { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.Enums;

namespace SecureShopBusinessLogic.ViewModels

{

public class ReportOrdersViewModel

{

public DateTime DateCreate { get; set; }

public string EquipmentName { get; set; }

public int Count { get; set; }

public decimal Sum { get; set; }

public OrderStatus Status { get; set; }

}

}

## Проект GiftShopClientApp

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.ViewModels;

using SecureShopClientApp.Models;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using SecureShopClientApp;

using SecureShopClientApp.Models;

namespace SecureShopClientApp.Controllers

{

public class HomeController : Controller

{

public HomeController()

{

}

public IActionResult Index()

{

if (Program.Client == null)

{

return Redirect("~/Home/Enter");

}

return

View(APIClient.GetRequest<List<OrderViewModel>>($"api/main/getorders?clientId={Program.Client.Id}"));

}

[HttpGet]

public IActionResult Privacy()

{

if (Program.Client == null)

{

return Redirect("~/Home/Enter");

}

return View(Program.Client);

}

[HttpPost]

public void Privacy(string login, string password, string fio)

{

if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password)

&& !string.IsNullOrEmpty(fio))

{

APIClient.PostRequest("api/client/updatedata", new ClientBindingModel

{

Id = Program.Client.Id,

ClientFIO = fio,

Email = login,

Password = password

});

Program.Client.ClientFIO = fio;

Program.Client.Email = login;

Program.Client.Password = password;

Response.Redirect("Index");

return;

}

throw new Exception("Введите логин, пароль и ФИО");

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel

{

RequestId = Activity.Current?.Id ??

HttpContext.TraceIdentifier

});

}

[HttpGet]

public IActionResult Enter()

{

return View();

}

[HttpPost]

public void Enter(string login, string password)

{

if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password))

{

Program.Client =

APIClient.GetRequest<ClientViewModel>($"api/client/login?login={login}&password={password}");

if (Program.Client == null)

{

throw new Exception("Неверный логин/пароль");

}

Response.Redirect("Index");

return;

}

throw new Exception("Введите логин, пароль");

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public void Register(string login, string password, string fio)

{

if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password)

&& !string.IsNullOrEmpty(fio))

{

APIClient.PostRequest("api/client/register", new

ClientBindingModel

{

ClientFIO = fio,

Email = login,

Password = password

});

Response.Redirect("Enter");

return;

}

throw new Exception("Введите логин, пароль и ФИО");

}

[HttpGet]

public IActionResult Create()

{

ViewBag.Equipments =

APIClient.GetRequest<List<EquipmentViewModel>>("api/main/getequipmentlist");

return View();

}

[HttpPost]

public void Create(int equipment, int count, decimal sum)

{

if (count == 0 || sum == 0)

{

return;

}

APIClient.PostRequest("api/main/createorder", new CreateOrderBindingModel

{

EquipmentId = equipment,

ClientId = Program.Client.Id,

Sum = sum,

Count = count

});

Response.Redirect("Index");

}

[HttpPost]

public decimal Calc(decimal count, int equipment)

{

EquipmentViewModel prod =

APIClient.GetRequest<EquipmentViewModel>($"api/main/getequipment?equipmentId={equipment}");

return count \* prod.Price;

}

}

}

using System;

namespace SecureShopClientApp.Models

{

public class ErrorViewModel

{

public string RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

using Microsoft.Extensions.Configuration;

using Newtonsoft.Json;

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

namespace SecureShopClientApp

{

public static class APIClient

{

private static readonly HttpClient client = new HttpClient();

public static void Connect(IConfiguration configuration)

{

client.BaseAddress = new Uri(configuration["IPAddress"]);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

public static T GetRequest<T>(string requestUrl)

{

var response = client.GetAsync(requestUrl);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

public static void PostRequest<T>(string requestUrl, T model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (!response.Result.IsSuccessStatusCode)

{

throw new Exception(result);

}

}

}

}

using Microsoft.Extensions.Configuration;

using Newtonsoft.Json;

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

namespace SecureShopClientApp

{

public static class APIClient

{

private static readonly HttpClient client = new HttpClient();

public static void Connect(IConfiguration configuration)

{

client.BaseAddress = new Uri(configuration["IPAddress"]);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

public static T GetRequest<T>(string requestUrl)

{

var response = client.GetAsync(requestUrl);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

public static void PostRequest<T>(string requestUrl, T model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (!response.Result.IsSuccessStatusCode)

{

throw new Exception(result);

}

}

}

}

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using SecureShopClientApp;

namespace SecureShowClientApp

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration; APIClient.Connect(configuration);

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for equipmention scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection(); app.UseStaticFiles();

app.UseRouting(); app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - SecureShopClientApp</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/site.css" />

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

</head>

<body>

<header>

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">

<div class="container">

<a class="navbar-brand" asp-area="" asp-controller="Home" asp- action="Index">Магазин охранных систем</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data- target=".navbar-collapse" aria-controls="navbarSupportedContent"

aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">

<ul class="navbar-nav flex-grow-1">

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Заказы</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Личные данные</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Enter">Вход</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Register">Регистрация</a>

</li>

</ul>

</div>

</div>

</nav>

</header>

<div class="container">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

<footer class="border-top footer text-muted">

<div class="container">

&copy; 2020 - Магазин охранных систем - <a asp-area="" asp- controller="Home" asp-action="Privacy">Личные данные</a>

</div>

</footer>

<script src="~/js/site.js" asp-append-version="true"></script> @RenderSection("Scripts", required: false)

</body>

</html>

@model ErrorViewModel

@{

ViewData["Title"] = "Error";

}

<h1 class="text-danger">Error.</h1>

<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)

{

<p>

<strong>Request ID:</strong> <code>@Model.RequestId</code>

</p>

}

<h3>Development Mode</h3>

<p>

Swapping to <strong>Development</strong> environment will display more detailed information about the error that occurred.

</p>

<p>

<strong>The Development environment shouldn't be enabled for deployed applications.</strong>

It can result in displaying sensitive information from exceptions to end users.

For local debugging, enable the <strong>Development</strong> environment by setting the <strong>ASPNETCORE\_ENVIRONMENT</strong> environment variable to <strong>Development</strong>

and restarting the app.

</p>

@{

ViewData["Title"] = "Create";

}

<div class="text-center">

<h2 class="display-4">Создание заказа</h2>

</div>

<form method="post">

<div class="row">

<div class="col-4">Устройство:</div>

<div class="col-8">

<select id="equipment" name="equipment" class="form-control" asp-items="@(new SelectList(@ViewBag.Equipments,"Id", "EquipmentName"))"></select>

</div>

</div>

<div class="row">

<div class="col-4">Количество:</div>

<div class="col-8"><input type="text" name="count" id="count" /></div>

</div>

<div class="row">

<div class="col-4">Сумма:</div>

<div class="col-8">

<input type="text" id="sum" name="sum" readonly="true" />

</div>

</div>

<div class="row">

<div class="col-8"></div>

<div class="col-4"><input type="submit" value="Создать" class="btn btn- primary" /></div>

</div>

</form>

<script>

$('#equipment').on('change', function () {

check();

});

$('#count').on('change', function () {

check();

});

function check() {

var count = $('#count').val();

var equipment = $('#equipment').val();

if (count && equipment) {

$.ajax({

method: "POST",

url: "/Home/Calc",

data: { count: count, equipment: equipment },

success: function (result) {

$("#sum").val(result);

}

});

};

}

</script>

@{

ViewData["Title"] = "Enter";

}

<div class="text-center">

<h2 class="display-4">Вход в приложение</h2>

</div>

<form method="post">

<div class="row">

<div class="col-4">Логин:</div>

<div class="col-8"><input type="text" name="login" /></div>

</div>

<div class="row">

<div class="col-4">Пароль:</div>

<div class="col-8"><input type="password" name="password" /></div>

</div>

<div class="row">

<div class="col-8"></div>

<div class="col-4"><input type="submit" value="Вход" class="btn btn- primary" /></div>

</div>

</form>

@using SecureShopBusinessLogic.ViewModels

@model List<OrderViewModel>

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1 class="display-4">Заказы</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Авторизируйтесь</h3> return;

}

<p>

<a asp-action="Create">Создать заказ</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер

</th>

<th>

Устройство

</th>

<th>

Дата создания

</th>

<th>

Количество

</th>

<th>

Сумма

</th>

<th>

Статус

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Id)

</td>

<td>

@Html.DisplayFor(modelItem =>

item.EquipmentName)

</td>

<td>

@Html.DisplayFor(modelItem =>

item.DateCreate)

</td>

<td>

@Html.DisplayFor(modelItem => item.Count)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@Html.DisplayFor(modelItem => item.Status)

</td>

</tr>

}

</tbody>

</table>

}

</div>

@using SecureShopBusinessLogic.ViewModels

@model List<MessageInfoViewModel>

@{ ViewData["Title"] = "Mail"; }

<div class="text-center">

<h1 class="display-4">Письма</h1>

</div>

<div class="text-center">

@{ if (Model == null)

{

<h3 class="display-4">Авторизируйтесь</h3> return;

}

<table class="table">

<thead>

<tr>

<th>

Отправитель

</th>

<th>

Заголовок

</th>

<th>

Дата письма

</th>

<th>

Текст

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.SenderName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Subject)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateDelivery)

</td>

<td>

@Html.DisplayFor(modelItem => item.Body)

</td>

</tr>

}

</tbody>

</table> }

</div>

@using SecureShopBusinessLogic.ViewModels

@model ClientViewModel

@{

ViewData["Title"] = "Privacy Policy";

}

<div class="text-center">

<h2 class="display-4">Личные данные</h2>

</div>

<form method="post">

<div class="row">

<div class="col-4">Логин:</div>

<div class="col-8"><input type="text" name="login" value="@Model.Email" /></div>

</div>

<div class="row">

<div class="col-4">Пароль:</div>

<div class="col-8"><input type="password" name="password" value="@Model.Password" /></div>

</div>

<div class="row">

<div class="col-4">ФИО:</div>

<div class="col-8"><input type="text" name="fio" value="@Model.ClientFIO" /></div>

</div>

<div class="row">

<div class="col-8"></div>

<div class="col-4"><input type="submit" value="Сохранить" class="btn btn-primary" /></div>

</div>

</form>

@{

ViewData["Title"] = "Register";

}

<div class="text-center">

<h2 class="display-4">Регистрация</h2>

</div>

<form method="post">

<div class="row">

<div class="col-4">Логин:</div>

<div class="col-8"><input type="text" name="login" /></div>

</div>

<div class="row">

<div class="col-4">Пароль:</div>

<div class="col-8"><input type="password" name="password" /></div>

</div>

<div class="row">

<div class="col-4">ФИО:</div>

<div class="col-8"><input type="text" name="fio" /></div>

</div>

<div class="row">

<div class="col-8"></div>

<div class="col-4"><input type="submit" value="Регистрация" class="btn btn-primary" /></div>

</div>

</form>

## Проект GiftShopDatabaseImplement

using SecureShopBusinessLogic.BusinessLogics;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

namespace SecureShopDatabaseImplement.Implements

{

public class BackUpLogic : SecureShopBackUpAbstractLogic

{

protected override Assembly GetAssembly()

{

return typeof(BackUpLogic).Assembly;

}

protected override List<PropertyInfo> GetFullList()

{

using (var context = new SecureShopDatabase())

{

Type type = context.GetType();

return type.GetProperties().Where(x =>

x.PropertyType.FullName.StartsWith("Microsoft.EntityFrameworkCore.DbSet")).ToList();

}

}

protected override List<T> GetList<T>()

{

using (var context = new SecureShopDatabase())

{

return context.Set<T>().ToList();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopDatabaseImplement.Models;

namespace SecureShopDatabaseImplement.Implements

{

public class ClientStorage : IClientStorage

{

public List<ClientViewModel> GetFullList()

{

using (var context = new SecureShopDatabase())

{

return context.Clients

.Select(CreateModel).ToList();

}

}

public List<ClientViewModel> GetFilteredList(ClientBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

return context.Clients

.Where(rec =>

rec.ClientFIO.Contains(model.ClientFIO) || (rec.Email.Equals(model.Email) && rec.Password.Equals(model.Password)))

.Select(CreateModel).ToList();

}

}

public ClientViewModel GetElement(ClientBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

var client = context.Clients

.FirstOrDefault(rec => rec.Email.Equals(model.Email) || rec.Id == model.Id);

return client != null ?

CreateModel(client) : null;

}

}

public void Insert(ClientBindingModel model)

{

using (var context = new SecureShopDatabase())

{

context.Clients.Add(CreateModel(model, new Client()));

context.SaveChanges();

}

}

public void Update(ClientBindingModel model)

{

using (var context = new SecureShopDatabase())

{

var element = context.Clients.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Client not found");

}

CreateModel(model, element);

context.SaveChanges();

}

}

public void Delete(ClientBindingModel model)

{

using (var context = new SecureShopDatabase())

{

Client element = context.Clients.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Clients.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Client not found");

}

}

}

private Client CreateModel(ClientBindingModel model, Client client)

{

client.ClientFIO = model.ClientFIO;

client.Email = model.Email;

client.Password = model.Password;

return client;

}

private ClientViewModel CreateModel(Client client)

{

return new ClientViewModel

{

Id = client.Id,

ClientFIO = client.ClientFIO,

Email = client.Email,

Password = client.Password

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopDatabaseImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopDatabaseImplement.Implements

{

public class DeviceStorage : IDeviceStorage

{

public List<DeviceViewModel> GetFullList()

{

using (var context = new SecureShopDatabase())

{

return context.Devices

.Select(rec => new DeviceViewModel

{

Id = rec.Id,

DeviceName = rec.DeviceName

})

.ToList();

}

}

public List<DeviceViewModel> GetFilteredList(DeviceBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

return context.Devices

.Where(rec => rec.DeviceName.Contains(model.DeviceName))

.Select(rec => new DeviceViewModel

{

Id = rec.Id,

DeviceName = rec.DeviceName

})

.ToList();

}

}

public DeviceViewModel GetElement(DeviceBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

var component = context.Devices

.FirstOrDefault(rec => rec.DeviceName == model.DeviceName ||

rec.Id == model.Id);

return component != null ?

new DeviceViewModel

{

Id = component.Id,

DeviceName = component.DeviceName

} :

null;

}

}

public void Insert(DeviceBindingModel model)

{

using (var context = new SecureShopDatabase())

{

context.Devices.Add(CreateModel(model, new Device()));

context.SaveChanges();

}

}

public void Update(DeviceBindingModel model)

{

using (var context = new SecureShopDatabase())

{

var element = context.Devices.FirstOrDefault(rec => rec.Id ==

model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

}

public void Delete(DeviceBindingModel model)

{

using (var context = new SecureShopDatabase())

{

Device element = context.Devices.FirstOrDefault(rec => rec.Id ==

model.Id);

if (element != null)

{

context.Devices.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

}

private Device CreateModel(DeviceBindingModel model, Device device)

{

device.DeviceName = model.DeviceName;

return device;

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace SecureShopDatabaseImplement.Implements

{

public class EquipmentStorage : IEquipmentStorage

{

public List<EquipmentViewModel> GetFullList()

{

using (var context = new SecureShopDatabase())

{

return context.Equipments

.Include(rec => rec.EquipmentDevice)

.ThenInclude(rec => rec.Device)

.ToList()

.Select(rec => new EquipmentViewModel

{

Id = rec.Id,

EquipmentName = rec.EquipmentName,

Price = rec.Price,

EquipmentDevices = rec.EquipmentDevice

.ToDictionary(recEquipmentDevices => recEquipmentDevices.DeviceId,

recEquipmentDevices => (recEquipmentDevices.Device?.DeviceName,

recEquipmentDevices.Count))

})

.ToList();

}

}

public List<EquipmentViewModel> GetFilteredList(EquipmentBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

return context.Equipments

.Include(rec => rec.EquipmentDevice)

.ThenInclude(rec => rec.Device)

.Where(rec => rec.EquipmentName.Contains(model.EquipmentName))

.ToList()

.Select(rec => new EquipmentViewModel

{

Id = rec.Id,

EquipmentName = rec.EquipmentName,

Price = rec.Price,

EquipmentDevices = rec.EquipmentDevice

.ToDictionary(recEquipmentDevices => recEquipmentDevices.DeviceId,

recEquipmentDevices => (recEquipmentDevices.Device?.DeviceName,

recEquipmentDevices.Count))

})

.ToList();

}

}

public EquipmentViewModel GetElement(EquipmentBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

var equipment = context.Equipments

.Include(rec => rec.EquipmentDevice)

.ThenInclude(rec => rec.Device)

.FirstOrDefault(rec => rec.EquipmentName == model.EquipmentName ||

rec.Id == model.Id);

return equipment != null ?

new EquipmentViewModel

{

Id = equipment.Id,

EquipmentName = equipment.EquipmentName,

Price = equipment.Price,

EquipmentDevices = equipment.EquipmentDevice

.ToDictionary(recEquipmentDevice => recEquipmentDevice.DeviceId,

recEquipmentDevice => (recEquipmentDevice.Device?.DeviceName,

recEquipmentDevice.Count))

} :

null;

}

}

public void Insert(EquipmentBindingModel model)

{

using (var context = new SecureShopDatabase())

{

using (var transaction = context.Database.BeginTransaction())

{

try

{

CreateModel(model, new Equipment(), context);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

}

}

public void Update(EquipmentBindingModel model)

{

using (var context = new SecureShopDatabase())

{

using (var transaction = context.Database.BeginTransaction())

{

try

{

var equipment = context.Equipments.FirstOrDefault(rec => rec.Id == model.Id);

if (equipment == null)

{

throw new Exception("Комплектация не найдена");

}

CreateModel(model, equipment, context);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

}

}

public void Delete(EquipmentBindingModel model)

{

using (var context = new SecureShopDatabase())

{

var Device = context.Equipments.FirstOrDefault(rec => rec.Id == model.Id);

if (Device == null)

{

throw new Exception("Материал не найден");

}

context.Equipments.Remove(Device);

context.SaveChanges();

}

}

private Equipment CreateModel(EquipmentBindingModel model, Equipment equipment, SecureShopDatabase context)

{

equipment.EquipmentName = model.EquipmentName;

equipment.Price = model.Price;

if (equipment.Id == 0)

{

context.Equipments.Add(equipment);

context.SaveChanges();

}

if (model.Id.HasValue)

{

var equipmentDevice = context.EquipmentDevices

.Where(rec => rec.EquipmentId == model.Id.Value)

.ToList();

context.EquipmentDevices.RemoveRange(equipmentDevice

.Where(rec => !model.EquipmentDevices.ContainsKey(rec.EquipmentId))

.ToList());

context.SaveChanges();

foreach (var updateDevice in equipmentDevice)

{

updateDevice.Count = model.EquipmentDevices[updateDevice.DeviceId].Item2;

model.EquipmentDevices.Remove(updateDevice.EquipmentId);

}

context.SaveChanges();

}

foreach (var equipmentDevice in model.EquipmentDevices)

{

context.EquipmentDevices.Add(new EquipmentDevice

{

EquipmentId = equipment.Id,

DeviceId = equipmentDevice.Key,

Count = equipmentDevice.Value.Item2

});

context.SaveChanges();

}

return equipment;

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopDatabaseImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopDatabaseImplement.Implements

{

public class ImplementerStorage : IImplementerStorage

{

public List<ImplementerViewModel> GetFullList()

{

using (var context = new SecureShopDatabase())

{

return context.Implementers

.Select(CreateModel).ToList();

}

}

public List<ImplementerViewModel> GetFilteredList(ImplementerBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

return context.Implementers

.Where(rec =>

rec.Name.Contains(model.Name))

.Select(CreateModel).ToList();

}

}

public ImplementerViewModel GetElement(ImplementerBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

var implementer = context.Implementers

.FirstOrDefault(rec => rec.Id == model.Id || rec.Name == model.Name);

return implementer != null ?

CreateModel(implementer) : null;

}

}

public void Insert(ImplementerBindingModel model)

{

using (var context = new SecureShopDatabase())

{

context.Implementers.Add(CreateModel(model, new Implementer()));

context.SaveChanges();

}

}

public void Update(ImplementerBindingModel model)

{

using (var context = new SecureShopDatabase())

{

var element = context.Implementers.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Исполнитель не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

}

public void Delete(ImplementerBindingModel model)

{

using (var context = new SecureShopDatabase())

{

Implementer element = context.Implementers.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Implementers.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Исполнитель не найден");

}

}

}

private Implementer CreateModel(ImplementerBindingModel model, Implementer implementer)

{

implementer.Name = model.Name;

implementer.WorkingTime = model.WorkingTime;

implementer.PauseTime = model.PauseTime;

return implementer;

}

private ImplementerViewModel CreateModel(Implementer implementer)

{

return new ImplementerViewModel

{

Id = implementer.Id,

Name = implementer.Name,

WorkingTime = implementer.WorkingTime,

PauseTime = implementer.PauseTime

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopDatabaseImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopDatabaseImplement.Implements

{

public class MessageInfoStorage : IMessageInfoStorage

{

public List<MessageInfoViewModel> GetFullList()

{

using (var context = new SecureShopDatabase())

{

return context.Messages

.Select(rec => new MessageInfoViewModel

{

MessageId = rec.MessageId,

SenderName = rec.SenderName,

DateDelivery = rec.DateDelivery,

Subject = rec.Subject,

Body = rec.Body

})

.ToList();

}

}

public List<MessageInfoViewModel> GetFilteredList(MessageInfoBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

return context.Messages

.Where(rec => (model.ClientId.HasValue && rec.ClientId == model.ClientId) ||

(!model.ClientId.HasValue && rec.DateDelivery.Date == model.DateDelivery.Date))

.Select(rec => new MessageInfoViewModel

{

MessageId = rec.MessageId,

SenderName = rec.SenderName,

DateDelivery = rec.DateDelivery,

Subject = rec.Subject,

Body = rec.Body

})

.ToList();

}

}

public void Insert(MessageInfoBindingModel model)

{

using (var context = new SecureShopDatabase())

{

MessageInfo element = context.Messages.FirstOrDefault(rec => rec.MessageId == model.MessageId);

if (element != null)

{

throw new Exception("Уже есть письмо с таким идентификатором");

}

context.Messages.Add(new MessageInfo

{

MessageId = model.MessageId,

ClientId = model.ClientId,

SenderName = model.FromMailAddress,

DateDelivery = model.DateDelivery,

Subject = model.Subject,

Body = model.Body

});

context.SaveChanges();

}

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Enums;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using SecureShopDatabaseImplement.Models;

namespace SecureShopDatabaseImplement.Implements

{

public class OrderStorage : IOrderStorage

{

public List<OrderViewModel> GetFullList()

{

using (var context = new SecureShopDatabase())

{

return context.Orders

.Include(rec => rec.Equipment)

.Include(rec => rec.Client)

.Include(rec => rec.Implementer)

.Select(CreateModel).ToList();

}

}

public List<OrderViewModel> GetFilteredList(OrderBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

return context.Orders

.Include(rec => rec.Equipment)

.Include(rec => rec.Client)

.Include(rec => rec.Implementer)

.Where(rec => (!model.DateFrom.HasValue && !model.DateTo.HasValue &&

rec.DateCreate.Date == model.DateCreate.Date) ||

(model.DateFrom.HasValue && model.DateTo.HasValue &&

rec.DateCreate.Date >= model.DateFrom.Value.Date && rec.DateCreate.Date <=

model.DateTo.Value.Date) ||

(model.ClientId.HasValue && rec.ClientId == model.ClientId) ||

(model.FreeOrders.HasValue && model.FreeOrders.Value && rec.Status ==

OrderStatus.Принят) ||

(model.ImplementerId.HasValue && rec.ImplementerId ==

model.ImplementerId && rec.Status == OrderStatus.Выполняется))

.Select(CreateModel).ToList();

}

}

public OrderViewModel GetElement(OrderBindingModel model)

{

if (model == null)

{

return null;

}

using (var context = new SecureShopDatabase())

{

var order = context.Orders

.Include(rec => rec.Equipment)

.Include(rec => rec.Client)

.Include(rec => rec.Implementer)

.FirstOrDefault(rec => rec.Id == model.Id);

return order != null ?

CreateModel(order) :

null;

}

}

public void Insert(OrderBindingModel model)

{

using (var context = new SecureShopDatabase())

{

context.Orders.Add(CreateModel(model, new Order()));

context.SaveChanges();

}

}

public void Update(OrderBindingModel model)

{

using (var context = new SecureShopDatabase())

{

var order = context.Orders.FirstOrDefault(rec => rec.Id == model.Id);

if (order == null)

{

throw new Exception("Заказ не найден");

}

CreateModel(model, order);

context.SaveChanges();

}

}

public void Delete(OrderBindingModel model)

{

using (var context = new SecureShopDatabase())

{

var order = context.Orders.FirstOrDefault(rec => rec.Id == model.Id);

if (order == null)

{

throw new Exception("Заказ не найден");

}

context.Orders.Remove(order);

context.SaveChanges();

}

}

private OrderViewModel CreateModel(Order order)

{

return new OrderViewModel

{

Id = order.Id,

EquipmentId = order.EquipmentId,

ClientId = order.ClientId.Value,

ImplementerId = order.ImplementerId,

ClientFIO = order.Client.ClientFIO,

EquipmentName = order.Equipment.EquipmentName,

Count = order.Count,

Sum = order.Sum,

Status = order.Status,

DateCreate = order.DateCreate,

DateImplement = order?.DateImplement,

ImplementerName = order.ImplementerId.HasValue ?

order.Implementer.Name : string.Empty

};

}

private Order CreateModel(OrderBindingModel model, Order order)

{

order.EquipmentId = model.EquipmentId;

order.ClientId = model.ClientId.Value;

order.ImplementerId = model.ImplementerId;

order.Count = model.Count;

order.Status = model.Status;

order.Sum = model.Sum;

order.DateCreate = model.DateCreate;

order.DateImplement = model.DateImplement;

return order;

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Text;

namespace SecureShopDatabaseImplement.Models

{

public class Client

{

public int Id { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string Password { get; set; }

[Required]

public string ClientFIO { get; set; }

[ForeignKey("ClientId")]

public virtual List<Order> Orders { get; set; }

[ForeignKey("ClientId")]

public List<MessageInfo> Messages { get; set; }

}

}

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShopDatabaseImplement.Models

{

/// <summary>

/// Компонент, требуемый для изготовления устройства

/// </summary>

public class Device

{

public int Id { get; set; }

[Required]

public string DeviceName { get; set; }

[ForeignKey("DeviceId")]

public virtual List<EquipmentDevice> EquipmentDevices { get; set; }

}

}

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShopDatabaseImplement.Models

{

public class Equipment

{

public int Id { get; set; }

[Required]

public string EquipmentName { get; set; }

[Required]

public decimal Price { get; set; }

[ForeignKey("EquipmentId")]

public virtual List<EquipmentDevice> EquipmentDevice { get; set; }

[ForeignKey("EquipmentId")]

public virtual List<Order> Orders { get; set; }

}

}

using System.ComponentModel.DataAnnotations;

namespace SecureShopDatabaseImplement.Models

{

/// <summary>

/// Сколько компонентов, требуется при изготовлении устройства

/// </summary>

public class EquipmentDevice

{

public int Id { get; set; }

public int EquipmentId { get; set; }

public int DeviceId { get; set; }

[Required]

public int Count { get; set; }

public virtual Device Device { get; set; }

public virtual Equipment Equipment { get; set; }

}

}

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShopDatabaseImplement.Models

{

public class Implementer

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

public int WorkingTime { get; set; }

[Required]

public int PauseTime { get; set; }

[ForeignKey("ImplementerId")]

public List<Order> Orders { get; set; }

}

}

using System;

using System.ComponentModel.DataAnnotations;

namespace SecureShopDatabaseImplement.Models

{

public class MessageInfo

{

[Key]

public string MessageId { get; set; }

public int? ClientId { get; set; }

public string SenderName { get; set; }

public DateTime DateDelivery { get; set; }

public string Subject { get; set; }

public string Body { get; set; }

public virtual Client Client { get; set; }

}

}

using SecureShopBusinessLogic.Enums;

using System;

using System.ComponentModel.DataAnnotations;

namespace SecureShopDatabaseImplement.Models

{

public class Order

{

public int Id { get; set; }

public int EquipmentId { get; set; }

public int? ClientId { get; set; }

public int? ImplementerId { get; set; }

[Required]

public int Count { get; set; }

[Required]

public decimal Sum { get; set; }

[Required]

public OrderStatus Status { get; set; }

[Required]

public DateTime DateCreate { get; set; }

public DateTime? DateImplement { get; set; }

public virtual Equipment Equipment { get; set; }

public virtual Client Client { get; set; }

public virtual Implementer Implementer { get; set; }

}

}

using SecureShopDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

namespace SecureShopDatabaseImplement

{

public class SecureShopDatabase : DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (optionsBuilder.IsConfigured == false)

{

optionsBuilder.UseSqlServer(@"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=SecureShopDatabase;Integrated Security=True;MultipleActiveResultSets=True;");

}

base.OnConfiguring(optionsBuilder);

}

public virtual DbSet<Device> Devices { set; get; }

public virtual DbSet<Equipment> Equipments { set; get; }

public virtual DbSet<EquipmentDevice> EquipmentDevices { set; get; }

public virtual DbSet<Order> Orders { set; get; }

public virtual DbSet<Client> Clients { set; get; }

public virtual DbSet<Implementer> Implementers { set; get; }

public virtual DbSet<MessageInfo> Messages { set; get; }

}

## Проект GiftShopFileImplement

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopFileImplement.Models;

namespace SecureShopFileImplement.Implements

{

public class ClientStorage : IClientStorage

{

private readonly FileDataListSingleton source;

public ClientStorage()

{

source = FileDataListSingleton.GetInstance();

}

public List<ClientViewModel> GetFullList()

{

return source.Clients.Select(CreateModel).ToList();

}

public List<ClientViewModel> GetFilteredList(ClientBindingModel model)

{

if (model == null)

{

return null;

}

return source.Clients.Where(rec =>

rec.ClientFIO.Contains(model.ClientFIO)).Select(CreateModel).ToList();

}

public ClientViewModel GetElement(ClientBindingModel model)

{

if (model == null)

{

return null;

}

var client = source.Clients

.FirstOrDefault(rec => rec.Email == model.Email || rec.Id == model.Id);

return client != null ? CreateModel(client) : null;

}

public void Insert(ClientBindingModel model)

{

int maxId = source.Clients.Count > 0 ? source.Clients.Max(rec => rec.Id) : 0;

var element = new Client { Id = maxId + 1 };

source.Clients.Add(CreateModel(model, element));

}

public void Update(ClientBindingModel model)

{

var element = source.Clients.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Client not found");

}

CreateModel(model, element);

}

public void Delete(ClientBindingModel model)

{

Client element = source.Clients.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

source.Clients.Remove(element);

}

else

{

throw new Exception("Client not found");

}

}

private Client CreateModel(ClientBindingModel model, Client client)

{

client.ClientFIO = model.ClientFIO;

client.Email = model.Email;

client.Password = model.Password;

return client;

}

private ClientViewModel CreateModel(Client client)

{

return new ClientViewModel

{

Id = client.Id,

ClientFIO = client.ClientFIO,

Email = client.Email,

Password = client.Password

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopFileImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopFileImplement.Implements

{

public class DeviceStorage : IDeviceStorage

{

private readonly FileDataListSingleton source;

public DeviceStorage()

{

source = FileDataListSingleton.GetInstance();

}

public List<DeviceViewModel> GetFullList()

{

return source.Devices

.Select(CreateModel)

.ToList();

}

public List<DeviceViewModel> GetFilteredList(DeviceBindingModel model)

{

if (model == null)

{

return null;

}

return source.Devices

.Where(rec => rec.DeviceName.Contains(model.DeviceName))

.Select(CreateModel)

.ToList();

}

public DeviceViewModel GetElement(DeviceBindingModel model)

{

if (model == null)

{

return null;

}

var device = source.Devices

.FirstOrDefault(rec => rec.DeviceName == model.DeviceName ||

rec.Id == model.Id);

return device != null ? CreateModel(device) : null;

}

public void Insert(DeviceBindingModel model)

{

int maxId = source.Devices.Count > 0 ? source.Devices.Max(rec => rec.Id) : 0;

var element = new Device { Id = maxId + 1 }; source.Devices.Add(CreateModel(model, element));

}

public void Update(DeviceBindingModel model)

{

var element = source.Devices.FirstOrDefault(rec => rec.Id == model.Id); if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

}

public void Delete(DeviceBindingModel model)

{

Device element = source.Devices.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

source.Devices.Remove(element);

}

else

{

throw new Exception("Элемент не найден");

}

}

private Device CreateModel(DeviceBindingModel model, Device device)

{

device.DeviceName = model.DeviceName;

return device;

}

private DeviceViewModel CreateModel(Device device)

{

return new DeviceViewModel

{

Id = device.Id,

DeviceName = device.DeviceName

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopFileImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopFileImplement.Implements

{

public class EquipmentStorage : IEquipmentStorage

{

private readonly FileDataListSingleton source;

public EquipmentStorage()

{

source = FileDataListSingleton.GetInstance();

}

public List<EquipmentViewModel> GetFullList()

{

return source.Equipments

.Select(CreateModel)

.ToList();

}

public List<EquipmentViewModel> GetFilteredList(EquipmentBindingModel model)

{

if (model == null)

{

return null;

}

return source.Equipments

.Where(rec => rec.EquipmentName.Contains(model.EquipmentName))

.Select(CreateModel)

.ToList();

}

public EquipmentViewModel GetElement(EquipmentBindingModel model)

{

if (model == null)

{

return null;

}

var equipment = source.Equipments

.FirstOrDefault(rec => rec.EquipmentName == model.EquipmentName || rec.Id

== model.Id);

return equipment != null ? CreateModel(equipment) : null;

}

public void Insert(EquipmentBindingModel model)

{

int maxId = source.Equipments.Count > 0 ? source.Devices.Max(rec => rec.Id) : 0;

var element = new Equipment

{

Id = maxId + 1,

EquipmentDevices = new

Dictionary<int, int>()

};

source.Equipments.Add(CreateModel(model, element));

}

public void Update(EquipmentBindingModel model)

{

var element = source.Equipments.FirstOrDefault(rec => rec.Id == model.Id); if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

}

public void Delete(EquipmentBindingModel model)

{

Equipment element = source.Equipments.FirstOrDefault(rec => rec.Id == model.Id); if (element != null)

{

source.Equipments.Remove(element);

}

else

{

throw new Exception("Элемент не найден");

}

}

private Equipment CreateModel(EquipmentBindingModel model, Equipment equipment)

{

equipment.EquipmentName = model.EquipmentName;

equipment.Price = model.Price;

// удаляем убранные

foreach (var key in equipment.EquipmentDevices.Keys.ToList())

{

if (!model.EquipmentDevices.ContainsKey(key))

{

equipment.EquipmentDevices.Remove(key);

}

}

// обновляем существуюущие и добавляем новые

foreach (var device in model.EquipmentDevices)

{

if (equipment.EquipmentDevices.ContainsKey(device.Key))

{

equipment.EquipmentDevices[device.Key] = model.EquipmentDevices[device.Key].Item2;

}

else

{

equipment.EquipmentDevices.Add(device.Key, model.EquipmentDevices[device.Key].Item2);

}

}

return equipment;

}

private EquipmentViewModel CreateModel(Equipment equipment)

{

return new EquipmentViewModel

{

Id = equipment.Id,

EquipmentName = equipment.EquipmentName,

Price = equipment.Price,

EquipmentDevices = equipment.EquipmentDevices

.ToDictionary(recPC => recPC.Key, recPC => (source.Devices

.FirstOrDefault(recC => recC.Id ==recPC.Key)?.DeviceName, recPC.Value))

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopFileImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopFileImplement.Implements

{

class ImplementerStorage : IImplementerStorage

{

private readonly FileDataListSingleton source;

public ImplementerStorage()

{

source = FileDataListSingleton.GetInstance();

}

public List<ImplementerViewModel> GetFullList()

{

return source.Implementers

.Select(CreateModel).ToList();

}

public List<ImplementerViewModel> GetFilteredList(ImplementerBindingModel model)

{

if (model == null)

{

return null;

}

return source.Implementers

.Where(rec =>

rec.Name.Contains(model.Name))

.Select(CreateModel).ToList();

}

public ImplementerViewModel GetElement(ImplementerBindingModel model)

{

if (model == null)

{

return null;

}

var implementer = source.Implementers

.FirstOrDefault(rec => rec.Id == model.Id);

return implementer != null ?

CreateModel(implementer) : null;

}

public void Insert(ImplementerBindingModel model)

{

int maxId = source.Implementers.Count > 0 ? source.Clients.Max(rec => rec.Id) : 0;

var element = new Implementer { Id = maxId + 1 };

source.Implementers.Add(CreateModel(model, element));

}

public void Update(ImplementerBindingModel model)

{

var element = source.Implementers.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Исполнитель не найден");

}

CreateModel(model, element);

}

public void Delete(ImplementerBindingModel model)

{

Implementer element = source.Implementers.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

source.Implementers.Remove(element);

}

else

{

throw new Exception("Исполнитель не найден");

}

}

private Implementer CreateModel(ImplementerBindingModel model, Implementer implementer)

{

implementer.Name = model.Name;

implementer.WorkingTime = model.WorkingTime;

implementer.PauseTime = model.PauseTime;

return implementer;

}

private ImplementerViewModel CreateModel(Implementer implementer)

{

return new ImplementerViewModel

{

Id = implementer.Id,

Name = implementer.Name,

WorkingTime = implementer.WorkingTime,

PauseTime = implementer.PauseTime

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Enums;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopFileImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopFileImplement.Implements

{

public class OrderStorage : IOrderStorage

{

private readonly FileDataListSingleton source;

public OrderStorage()

{

source = FileDataListSingleton.GetInstance();

}

public List<OrderViewModel> GetFullList()

{

return source.Orders

.Select(CreateModel)

.ToList();

}

public List<OrderViewModel> GetFilteredList(OrderBindingModel model)

{

if (model == null)

{

return null;

}

return source.Orders.Where(rec => (!model.DateFrom.HasValue &&

!model.DateTo.HasValue && rec.DateCreate.Date == model.DateCreate.Date) ||

(model.DateFrom.HasValue && model.DateTo.HasValue && rec.DateCreate.Date >=

model.DateFrom.Value.Date && rec.DateCreate.Date <= model.DateTo.Value.Date) ||

(model.ClientId.HasValue && rec.ClientId == model.ClientId) ||

(model.FreeOrders.HasValue && model.FreeOrders.Value && rec.Status == OrderStatus.Принят) ||

(model.ImplementerId.HasValue && rec.ImplementerId == model.ImplementerId && rec.Status == OrderStatus.Выполняется))

.Select(CreateModel)

.ToList();

}

public OrderViewModel GetElement(OrderBindingModel model)

{

if (model == null)

{

return null;

}

foreach (var order in source.Orders)

{

if (order.Id == model.Id || order.EquipmentId ==

model.EquipmentId)

{

return CreateModel(order);

}

}

return null;

}

public void Insert(OrderBindingModel model)

{

Order tempOrder = new Order

{

Id = 1

};

foreach (var order in source.Orders)

{

if (order.Id >= tempOrder.Id)

{

tempOrder.Id = order.Id + 1;

}

}

source.Orders.Add(CreateModel(model, tempOrder));

}

public void Update(OrderBindingModel model)

{

Order tempOrder = null;

foreach (var order in source.Orders)

{

if (order.Id == model.Id)

{

tempOrder = order;

}

}

if (tempOrder == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, tempOrder);

}

public void Delete(OrderBindingModel model)

{

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

{

if (source.Orders[i].Id == model.Id)

{

source.Orders.RemoveAt(i);

return;

}

}

throw new Exception("Элемент не найден");

}

private Order CreateModel(OrderBindingModel model, Order order)

{

order.EquipmentId = model.EquipmentId;

order.Count = model.Count;

order.Sum = model.Sum;

order.Status = model.Status;

order.DateCreate = model.DateCreate;

order.DateImplement = model.DateImplement;

return order;

}

private OrderViewModel CreateModel(Order order)

{

return new OrderViewModel

{

Id = order.Id,

EquipmentId = order.EquipmentId,

Count = order.Count,

Sum = order.Sum,

Status = order.Status,

DateCreate = order.DateCreate,

DateImplement = order.DateImplement

};

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopFileImplement.Models

{

public class Client

{

public int Id { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string ClientFIO { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopFileImplement.Models

{

public class Device

{

public int Id { get; set; }

public string DeviceName { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopFileImplement.Models

{

public class Equipment

{

public int Id { get; set; }

public string EquipmentName { get; set; }

public decimal Price { get; set; }

public Dictionary<int, int> EquipmentDevices { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopFileImplement.Models

{

public class Implementer

{

public int Id { get; set; }

public string Name { get; set; }

public int WorkingTime { get; set; }

public int PauseTime { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.Enums;

namespace SecureShopFileImplement.Models

{

public class Order

{

public int Id { get; set; }

public int EquipmentId { get; set; }

public int? ClientId { get; set; }

public int? ImplementerId { get; set; }

public int Count { get; set; }

public decimal Sum { get; set; }

public OrderStatus Status { get; set; }

public DateTime DateCreate { get; set; }

public DateTime? DateImplement { get; set; }

}

}

## Проект GiftShopListImplement

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopListImplement.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopListImplement.Implements

{

public class ClientStorage : IClientStorage

{

private readonly DataListSingleton source;

public ClientStorage()

{

source = DataListSingleton.GetInstance();

}

public List<ClientViewModel> GetFullList()

{

List<ClientViewModel> result = new List<ClientViewModel>();

foreach (var client in source.Clients)

{

result.Add(CreateModel(client));

}

return result;

}

public List<ClientViewModel> GetFilteredList(ClientBindingModel model)

{

if (model == null)

{

return null;

}

List<ClientViewModel> result = new List<ClientViewModel>();

foreach (var client in source.Clients)

{

if (client.ClientFIO.Contains(model.ClientFIO))

{

result.Add(CreateModel(client));

}

}

return result;

}

public ClientViewModel GetElement(ClientBindingModel model)

{

if (model == null)

{

return null;

}

foreach (var client in source.Clients)

{

if (client.Id == model.Id || client.ClientFIO == model.ClientFIO)

{

return CreateModel(client);

}

}

return null;

}

public void Insert(ClientBindingModel model)

{

Client tempClient = new Client { Id = 1 };

foreach (var client in source.Clients)

{

if (client.Id >= tempClient.Id)

{

tempClient.Id = client.Id + 1;

}

}

source.Clients.Add(CreateModel(model, tempClient));

}

public void Update(ClientBindingModel model)

{

Client tempClient = null;

foreach (var client in source.Clients)

{

if (client.Id == model.Id)

{

tempClient = client;

}

}

if (tempClient == null)

{

throw new Exception("Клиент не найден");

}

CreateModel(model, tempClient);

}

public void Delete(ClientBindingModel model)

{

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

{

if (source.Clients[i].Id == model.Id.Value)

{

source.Clients.RemoveAt(i);

return;

}

}

throw new Exception("Клиент не найден");

}

private Client CreateModel(ClientBindingModel model, Client client)

{

client.ClientFIO = model.ClientFIO;

client.Email = model.Email;

client.Password = model.Password;

return client;

}

private ClientViewModel CreateModel(Client client)

{

return new ClientViewModel

{

Id = client.Id,

ClientFIO = client.ClientFIO,

Email = client.Email,

Password = client.Password

};

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopListImplement.Models;

namespace SecureShopListImplement.Implements

{

public class DeviceStorage : IDeviceStorage

{

private readonly DataListSingleton source;

public DeviceStorage()

{

source = DataListSingleton.GetInstance();

}

public List<DeviceViewModel> GetFullList()

{

List<DeviceViewModel> result = new List<DeviceViewModel>();

foreach (var device in source.Devices)

{

result.Add(CreateModel(device));

}

return result;

}

public List<DeviceViewModel> GetFilteredList(DeviceBindingModel model)

{

if (model == null)

{

return null;

}

List<DeviceViewModel> result = new List<DeviceViewModel>();

foreach (var device in source.Devices)

{

if (device.DeviceName.Contains(model.DeviceName))

{

result.Add(CreateModel(device));

}

}

return result;

}

public DeviceViewModel GetElement(DeviceBindingModel model)

{

if (model == null)

{

return null;

}

foreach (var device in source.Devices)

{

if (device.Id == model.Id || device.DeviceName ==

model.DeviceName)

{

return CreateModel(device);

}

}

return null;

}

public void Insert(DeviceBindingModel model)

{

Device tempDevice = new Device { Id = 1 };

foreach (var device in source.Devices)

{

if (device.Id >= tempDevice.Id)

{

tempDevice.Id = device.Id + 1;

}

}

source.Devices.Add(CreateModel(model, tempDevice));

}

public void Update(DeviceBindingModel model)

{

Device tempDevice = null;

foreach (var device in source.Devices)

{

if (device.Id == model.Id)

{

tempDevice = device;

}

}

if (tempDevice == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, tempDevice);

}

public void Delete(DeviceBindingModel model)

{

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

{

if (source.Devices[i].Id == model.Id.Value)

{

source.Devices.RemoveAt(i);

return;

}

}

throw new Exception("Элемент не найден");

}

private Device CreateModel(DeviceBindingModel model, Device component)

{

component.DeviceName = model.DeviceName;

return component;

}

private DeviceViewModel CreateModel(Device device)

{

return new DeviceViewModel

{

Id = device.Id,

DeviceName = device.DeviceName

};

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopListImplement.Models;

namespace SecureShopListImplement.Implements

{

public class EquipmentStorage : IEquipmentStorage

{

private readonly DataListSingleton source;

public EquipmentStorage()

{

source = DataListSingleton.GetInstance();

}

public List<EquipmentViewModel> GetFullList()

{

List<EquipmentViewModel> result = new List<EquipmentViewModel>();

foreach (var device in source.Equipments)

{

result.Add(CreateModel(device));

}

return result;

}

public List<EquipmentViewModel> GetFilteredList(EquipmentBindingModel model)

{

if (model == null)

{

return null;

}

List<EquipmentViewModel> result = new List<EquipmentViewModel>();

foreach (var equipment in source.Equipments)

{

if (equipment.EquipmentName.Contains(model.EquipmentName))

{

result.Add(CreateModel(equipment));

}

}

return result;

}

public EquipmentViewModel GetElement(EquipmentBindingModel model)

{

if (model == null)

{

return null;

}

foreach (var equipment in source.Equipments)

{

if (equipment.Id == model.Id || equipment.EquipmentName ==

model.EquipmentName)

{

return CreateModel(equipment);

}

}

return null;

}

public void Insert(EquipmentBindingModel model)

{

Equipment tempEquipment = new Equipment

{

Id = 1,

EquipmentDevices = new

Dictionary<int, int>()

};

foreach (var equipment in source.Equipments)

{

if (equipment.Id >= tempEquipment.Id)

{

tempEquipment.Id = equipment.Id + 1;

}

}

source.Equipments.Add(CreateModel(model, tempEquipment));

}

public void Update(EquipmentBindingModel model)

{

Equipment tempEquipment = null;

foreach (var equipment in source.Equipments)

{

if (equipment.Id == model.Id)

{

tempEquipment = equipment;

}

}

if (tempEquipment == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, tempEquipment);

}

public void Delete(EquipmentBindingModel model)

{

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

{

if (source.Equipments[i].Id == model.Id)

{

source.Equipments.RemoveAt(i);

return;

}

}

throw new Exception("Элемент не найден");

}

private Equipment CreateModel(EquipmentBindingModel model, Equipment equipment)

{

equipment.EquipmentName = model.EquipmentName;

equipment.Price = model.Price;

// удаляем убранные

foreach (var key in equipment.EquipmentDevices.Keys.ToList())

{

if (!model.EquipmentDevices.ContainsKey(key))

{

equipment.EquipmentDevices.Remove(key);

}

}

// обновляем существуюущие и добавляем новые

foreach (var device in model.EquipmentDevices)

{

if (equipment.EquipmentDevices.ContainsKey(device.Key))

{

equipment.EquipmentDevices[device.Key] =

model.EquipmentDevices[device.Key].Item2;

}

else

{

equipment.EquipmentDevices.Add(device.Key,

model.EquipmentDevices[device.Key].Item2);

}

}

return equipment;

}

private EquipmentViewModel CreateModel(Equipment equipment)

{

// требуется дополнительно получить список компонентов для устройства с названиями и их количество

Dictionary<int, (string, int)> equipmentDevices = new Dictionary<int, (string, int)>();

foreach (var eq in equipment.EquipmentDevices)

{

string deviceName = string.Empty;

foreach (var device in source.Devices)

{

if (eq.Key == device.Id)

{

deviceName = device.DeviceName;

break;

}

}

equipmentDevices.Add(eq.Key, (deviceName, eq.Value));

}

return new EquipmentViewModel

{

Id = equipment.Id,

EquipmentName = equipment.EquipmentName,

Price = equipment.Price,

EquipmentDevices = equipmentDevices

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopListImplement.Models;

using System;

using System.Collections.Generic;

namespace SecureShopListImplement.Implements

{

class ImplementerStorage : IImplementerStorage

{

private readonly DataListSingleton source;

public ImplementerStorage()

{

source = DataListSingleton.GetInstance();

}

public List<ImplementerViewModel> GetFullList()

{

List<ImplementerViewModel> result = new List<ImplementerViewModel>();

foreach (var implementer in source.Implementers)

{

result.Add(CreateModel(implementer));

}

return result;

}

public List<ImplementerViewModel> GetFilteredList(ImplementerBindingModel model)

{

if (model == null)

{

return null;

}

List<ImplementerViewModel> result = new List<ImplementerViewModel>();

foreach (var implementer in source.Implementers)

{

if (implementer.Name.Contains(model.Name))

{

result.Add(CreateModel(implementer));

}

}

return result;

}

public ImplementerViewModel GetElement(ImplementerBindingModel model)

{

if (model == null)

{

return null;

}

foreach (var implementer in source.Implementers)

{

if (implementer.Id == model.Id)

{

return CreateModel(implementer);

}

}

return null;

}

public void Insert(ImplementerBindingModel model)

{

Implementer tempImplementer = new Implementer { Id = 1 };

foreach (var implementer in source.Implementers)

{

if (implementer.Id >= tempImplementer.Id)

{

tempImplementer.Id = implementer.Id + 1;

}

}

source.Implementers.Add(CreateModel(model, tempImplementer));

}

public void Update(ImplementerBindingModel model)

{

Implementer tempImplementer = null;

foreach (var implementer in source.Implementers)

{

if (implementer.Id == model.Id)

{

tempImplementer = implementer;

}

}

if (tempImplementer == null)

{

throw new Exception("Исполнитель не найден");

}

CreateModel(model, tempImplementer);

}

public void Delete(ImplementerBindingModel model)

{

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

{

if (source.Implementers[i].Id == model.Id.Value)

{

source.Implementers.RemoveAt(i);

return;

}

}

throw new Exception("Исполнитель не найден");

}

private Implementer CreateModel(ImplementerBindingModel model, Implementer implementer)

{

implementer.Name = model.Name;

implementer.WorkingTime = model.WorkingTime;

implementer.PauseTime = model.PauseTime;

return implementer;

}

private ImplementerViewModel CreateModel(Implementer implementer)

{

return new ImplementerViewModel

{

Id = implementer.Id,

Name = implementer.Name,

WorkingTime = implementer.WorkingTime,

PauseTime = implementer.PauseTime

};

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.Enums;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.ViewModels;

using SecureShopListImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopListImplement.Implements

{

public class OrderStorage : IOrderStorage

{

private readonly DataListSingleton source;

public OrderStorage()

{

source = DataListSingleton.GetInstance();

}

public List<OrderViewModel> GetFullList()

{

List<OrderViewModel> result = new List<OrderViewModel>();

foreach (var order in source.Orders)

{

result.Add(CreateModel(order));

}

return result;

}

public List<OrderViewModel> GetFilteredList(OrderBindingModel model)

{

if (model == null)

{

return null;

}

List<OrderViewModel> result = new List<OrderViewModel>();

foreach (var order in source.Orders)

{

if ((!model.DateFrom.HasValue && !model.DateTo.HasValue && order.DateCreate.Date == model.DateCreate.Date) ||

(model.DateFrom.HasValue && model.DateTo.HasValue && order.DateCreate.Date >= model.DateFrom.Value.Date && order.DateCreate.Date <= model.DateTo.Value.Date) ||

(model.ClientId.HasValue && order.ClientId == model.ClientId) ||

(model.FreeOrders.HasValue && model.FreeOrders.Value && order.Status == OrderStatus.Принят) ||

(model.ImplementerId.HasValue && order.ImplementerId == model.ImplementerId && order.Status == OrderStatus.Выполняется))

{

result.Add(CreateModel(order));

}

}

return result;

}

public OrderViewModel GetElement(OrderBindingModel model)

{

if (model == null)

{

return null;

}

foreach (var order in source.Orders)

{

if (order.Id == model.Id || order.EquipmentId ==

model.EquipmentId)

{

return CreateModel(order);

}

}

return null;

}

public void Insert(OrderBindingModel model)

{

Order tempOrder = new Order

{

Id = 1

};

foreach (var order in source.Orders)

{

if (order.Id >= tempOrder.Id)

{

tempOrder.Id = order.Id + 1;

}

}

source.Orders.Add(CreateModel(model, tempOrder));

}

public void Update(OrderBindingModel model)

{

Order tempOrder = null;

foreach (var order in source.Orders)

{

if (order.Id == model.Id)

{

tempOrder = order;

}

}

if (tempOrder == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, tempOrder);

}

public void Delete(OrderBindingModel model)

{

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

{

if (source.Orders[i].Id == model.Id)

{

source.Orders.RemoveAt(i);

return;

}

}

throw new Exception("Элемент не найден");

}

private Order CreateModel(OrderBindingModel model, Order order)

{

order.EquipmentId = model.EquipmentId;

order.ClientId = model.ClientId;

order.ImplementerId = model.ImplementerId;

order.Count = model.Count;

order.Status = model.Status;

order.Sum = model.Sum;

order.DateCreate = model.DateCreate;

order.DateImplement = model.DateImplement;

return order;

}

private OrderViewModel CreateModel(Order order)

{

string giftName = null;

foreach (var gift in source.Equipments)

{

if (gift.Id == order.EquipmentId)

{

giftName = gift.EquipmentName;

}

}

string clientFIO = null;

foreach (var client in source.Clients)

{

if (client.Id == order.ClientId)

{

clientFIO = client.ClientFIO;

}

}

string implementerName = null;

foreach (var implementer in source.Implementers)

{

if (implementer.Id == order.ImplementerId)

{

implementerName = implementer.Name;

}

}

return new OrderViewModel

{

Id = order.Id,

EquipmentId = order.EquipmentId,

EquipmentName = giftName,

ClientId = order.ClientId.Value,

ClientFIO = clientFIO,

ImplementerId = order.ImplementerId.Value,

ImplementerName = implementerName,

Count = order.Count,

Sum = order.Sum,

Status = order.Status,

DateCreate = order.DateCreate,

DateImplement = order.DateImplement

};

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopListImplement.Models

{

public class Client

{

public int Id { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string ClientFIO { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopListImplement.Models

{

public class Device

{

public int Id { get; set; }

public string DeviceName { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopListImplement.Models

{

public class Equipment

{

public int Id { get; set; }

public string EquipmentName { get; set; }

public decimal Price { get; set; }

public Dictionary<int, int> EquipmentDevices { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace SecureShopListImplement.Models

{

public class Implementer

{

public int Id { get; set; }

public string Name { get; set; }

public int WorkingTime { get; set; }

public int PauseTime { get; set; }

}

}

using SecureShopBusinessLogic.Enums;

using System;

namespace SecureShopListImplement.Models

{

public class Order

{

public int Id { get; set; }

public int EquipmentId { get; set; }

public int? ClientId { get; set; }

public int? ImplementerId { get; set; }

public int Count { get; set; }

public decimal Sum { get; set; }

public OrderStatus Status { get; set; }

public DateTime DateCreate { get; set; }

public DateTime? DateImplement { get; set; }

}

}

using SecureShopListImplement.Models;

using System.Collections.Generic;

namespace SecureShopListImplement

{

public class DataListSingleton

{

private static DataListSingleton instance;

public List<Device> Devices { get; set; }

public List<Order> Orders { get; set; }

public List<Equipment> Equipments { get; set; }

public List<Client> Clients { get; set; }

public List<Implementer> Implementers { get; set; }

private DataListSingleton()

{

Devices = new List<Device>();

Orders = new List<Order>();

Equipments = new List<Equipment>();

Clients = new List<Client>();

Implementers = new List<Implementer>();

}

public static DataListSingleton GetInstance()

{

if (instance == null)

{

instance = new DataListSingleton();

}

return instance;

}

}

}

## Проект GiftShopRestApi

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace SecureShopRestApi

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

using SecureShopBusinessLogic.BusinessLogics;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.Interfaces;

using SecureShopDatabaseImplement.Implements;

using SecureShopBusinessLogic.HelperModels;

namespace SecureShopRestApi

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddTransient<IClientStorage, ClientStorage>();

services.AddTransient<IOrderStorage, OrderStorage>();

services.AddTransient<IEquipmentStorage, EquipmentStorage>();

services.AddTransient<IMessageInfoStorage, MessageInfoStorage>();

services.AddTransient<OrderLogic>();

services.AddTransient<ClientLogic>();

services.AddTransient<EquipmentLogic>();

services.AddTransient<MailLogic>();

services.AddControllers().AddNewtonsoftJson();

MailLogic.MailConfig(new MailConfig

{

SmtpClientHost = "smtp.gmail.com",

SmtpClientPort = 587,

MailLogin = "emiryanforlab@gmail.com",

MailPassword = "123qweQWE"

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

using System;

namespace SecureShopRestApi

{

public class WeatherForecast

{

public DateTime Date { get; set; }

public int TemperatureC { get; set; }

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

public string Summary { get; set; }

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.ViewModels;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

namespace SecureShopRestApi.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class ClientController : ControllerBase

{

private readonly ClientLogic \_clientLogic;

private readonly MailLogic \_mailLogic;

private readonly int \_passwordMaxLength = 50;

private readonly int \_passwordMinLength = 10;

public ClientController(ClientLogic clientLogic, MailLogic mailLogic)

{

\_clientLogic = clientLogic;

\_mailLogic = mailLogic;

}

[HttpGet]

public ClientViewModel Login(string login, string password) => \_clientLogic.Read(new ClientBindingModel { Email = login, Password = password })?[0];

[HttpGet]

public List<MessageInfoViewModel> GetMessages(int clientId) => \_mailLogic.Read(new MessageInfoBindingModel { ClientId = clientId });

[HttpPost]

public void Register(ClientBindingModel model)

{

CheckData(model);

\_clientLogic.CreateOrUpdate(model);

}

[HttpPost]

public void UpdateData(ClientBindingModel model)

{

CheckData(model);

\_clientLogic.CreateOrUpdate(model);

}

private void CheckData(ClientBindingModel model)

{

if (!Regex.IsMatch(model.Email, @"^[A-Za-z0-9]+(?:[.\_%+-])?[A-Za-z0-9.\_-]+[A-Za-z0-9]@[A-Za-z0-9]+(?:[.-])?[A-Za-z0-9.\_-]+\.[A-Za-z]{2,6}$"))

{

throw new Exception(" ");

}

if (model.Password.Length > \_passwordMaxLength || model.Password.Length < \_passwordMinLength || !Regex.IsMatch(model.Password, @"^((\w+\d+\W+)|(\w+\W+\d+)|(\d+\w+\W+)|(\d+\W+\w+)|(\W+\w+\d+)|(\W+\d+\w+))[\w\d\W]\*$"))

{

throw new Exception($" {\_passwordMinLength} {\_passwordMaxLength} , ");

}

}

}

}

using Microsoft.AspNetCore.Mvc;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.ViewModels;

using System.Collections.Generic;

using System.Linq;

namespace SecureShopRestApi.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class MainController : ControllerBase

{

private readonly OrderLogic \_order; private readonly EquipmentLogic \_equipment; private readonly OrderLogic \_main;

public MainController(OrderLogic order, EquipmentLogic equipment, OrderLogic main)

{

\_order = order;

\_equipment = equipment;

\_main = main;

}

[HttpGet]

public List<EquipmentViewModel> GetEquipmentList() => \_equipment.Read(null)?.ToList();

[HttpGet]

public EquipmentViewModel GetEquipment(int equipmentId) => \_equipment.Read(new EquipmentBindingModel { Id = equipmentId })?[0];

[HttpGet]

public List<OrderViewModel> GetOrders(int clientId) => \_order.Read(new OrderBindingModel { ClientId = clientId });

[HttpPost]

public void CreateOrder(CreateOrderBindingModel model) =>

\_main.CreateOrder(model);

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace SecureShopRestApi.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet]

public IEnumerable<WeatherForecast> Get()

{

var rng = new Random();

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateTime.Now.AddDays(index),

TemperatureC = rng.Next(-20, 55),

Summary = Summaries[rng.Next(Summaries.Length)]

})

.ToArray();

}

}

}

## Проект GiftShopView

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using System;

using System.Windows.Forms;

using Unity;

namespace SecureShopView

{

public partial class FormClients : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly ClientLogic logic;

public FormClients(ClientLogic logic)

{

InitializeComponent();

this.logic = logic;

LoadData();

}

private void LoadData()

{

try

{

Program.ConfigGrid(logic.Read(null), dataGridView);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void ButtonDelete\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

if (MessageBox.Show("Удалить запись", "Вопрос", MessageBoxButtons.YesNo,

MessageBoxIcon.Question) == DialogResult.Yes)

{

int id =

Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

logic.Delete(new ClientBindingModel { Id = id });

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

LoadData();

}

}

}

private void ButtonUpdate\_Click(object sender, EventArgs e)

{

LoadData();

}

}

}

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.ViewModels;

using Unity;

namespace SecureShopView

{

public partial class FormCreateOrder : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly EquipmentLogic \_logicP;

private readonly OrderLogic \_logicO;

private readonly ClientLogic \_logicC;

public FormCreateOrder(EquipmentLogic logicE, OrderLogic logicO, ClientLogic logicC)

{

InitializeComponent();

\_logicP = logicE;

\_logicO = logicO;

\_logicC = logicC;

}

private void FormCreateOrder\_Load(object sender, EventArgs e)

{

try

{

List<EquipmentViewModel> list = \_logicP.Read(null);

if (list != null)

{

comboBoxEquipment.DisplayMember = "EquipmentName";

comboBoxEquipment.ValueMember = "Id";

comboBoxEquipment.DataSource = list;

comboBoxEquipment.SelectedItem = null;

}

List<ClientViewModel> listClients = \_logicC.Read(null);

if (listClients != null)

{

comboBoxClient.DisplayMember = "ClientFIO";

comboBoxClient.ValueMember = "Id";

comboBoxClient.DataSource = listClients;

comboBoxClient.SelectedItem = null;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void CalcSum()

{

if (comboBoxEquipment.SelectedValue != null && !string.IsNullOrEmpty(textBoxCount.Text))

{

try

{

int id = Convert.ToInt32(comboBoxEquipment.SelectedValue);

EquipmentViewModel equipment = \_logicP.Read(new EquipmentBindingModel { Id = id })?[0];

int count = Convert.ToInt32(textBoxCount.Text);

textBoxSum.Text = (count \* equipment?.Price ?? 0).ToString();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void TextBoxCount\_TextChanged(object sender, EventArgs e)

{

CalcSum();

}

private void ComboBoxEquipment\_SelectedIndexChanged(object sender, EventArgs e)

{

CalcSum();

}

private void ButtonSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBoxCount.Text))

{

MessageBox.Show("Заполните поле Количество", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (comboBoxEquipment.SelectedValue == null)

{

MessageBox.Show("Выберите комплектацию", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

\_logicO.CreateOrder(new CreateOrderBindingModel

{

ClientId = Convert.ToInt32(comboBoxClient.SelectedValue),

EquipmentId = Convert.ToInt32(comboBoxEquipment.SelectedValue),

Count = Convert.ToInt32(textBoxCount.Text),

Sum = Convert.ToDecimal(textBoxSum.Text)

});

MessageBox.Show("Сохранение прошло успешно", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ButtonCancel\_Click(object sender, EventArgs e)

{

DialogResult = DialogResult.Cancel;

Close();

}

}

}

using System;

using System.Windows.Forms;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using Unity;

namespace SecureShopView

{

public partial class FormDevice : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

public int Id { set { id = value; } }

private readonly DeviceLogic logic;

private int? id;

public FormDevice(DeviceLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void FormDevice\_Load(object sender, EventArgs e)

{

if (id.HasValue)

{

try

{

var view = logic.Read(new DeviceBindingModel { Id = id })?[0];

if (view != null)

{

deviceTextBox.Text = view.DeviceName;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

private void ButtonSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(deviceTextBox.Text))

{

MessageBox.Show("Заполните название", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

logic.CreateOrUpdate(new DeviceBindingModel

{

Id = id,

DeviceName = deviceTextBox.Text

});

MessageBox.Show("Сохранение прошло успешно", "Сообщение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void ButtonCancel\_Click(object sender, EventArgs e)

{

DialogResult = DialogResult.Cancel;

Close();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Unity;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

namespace SecureShopView

{

public partial class FormDevices : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly DeviceLogic logic;

public FormDevices(DeviceLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void FormDevice\_Load(object sender, EventArgs e)

{

LoadData();

}

private void LoadData()

{

try

{

Program.ConfigGrid(logic.Read(null), dataGridView);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void ButtonAdd\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormDevice>();

if (form.ShowDialog() == DialogResult.OK)

{

LoadData();

}

}

private void ButtonEdit\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

var form = Container.Resolve<FormDevice>();

form.Id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

if (form.ShowDialog() == DialogResult.OK)

{

LoadData();

}

}

}

private void ButtonDel\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

if (MessageBox.Show("Удалить запись", "Вопрос", MessageBoxButtons.YesNo,

MessageBoxIcon.Question) == DialogResult.Yes)

{

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

logic.Delete(new DeviceBindingModel { Id = id });

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

LoadData();

}

}

}

private void ButtonUpdate\_Click(object sender, EventArgs e)

{

LoadData();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Unity;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.ViewModels;

namespace SecureShopView

{

public partial class FormEquipment : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

public int Id { set { id = value; } }

private readonly EquipmentLogic logic;

private int? id;

private Dictionary<int, (string, int)> equipmentComponents;

public FormEquipment(EquipmentLogic service)

{

InitializeComponent();

this.logic = service;

}

private void FormEquipment\_Load(object sender, EventArgs e)

{

if (id.HasValue)

{

try

{

EquipmentViewModel view = logic.Read(new EquipmentBindingModel{Id =id.Value})?[0];

if (view != null)

{

textBoxName.Text = view.EquipmentName;

textBoxPrice.Text = view.Price.ToString();

equipmentComponents = view.EquipmentDevices;

LoadData();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

equipmentComponents = new Dictionary<int, (string, int)>();

}

}

private void LoadData()

{

try

{

if (equipmentComponents != null)

{

dataGridView.Rows.Clear();

foreach (var pc in equipmentComponents)

{

dataGridView.Rows.Add(new object[] { pc.Key, pc.Value.Item1, pc.Value.Item2 });

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void ButtonAdd\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormEquipmentDevices>();

if (form.ShowDialog() == DialogResult.OK)

{

if (equipmentComponents.ContainsKey(form.Id))

{

equipmentComponents[form.Id] = (form.ComponentName, form.Count);

}

else

{

equipmentComponents.Add(form.Id, (form.ComponentName, form.Count));

}

LoadData();

}

}

private void ButtonEdit\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

var form = Container.Resolve<FormEquipmentDevices>();

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

form.Id = id;

form.Count = equipmentComponents[id].Item2;

if (form.ShowDialog() == DialogResult.OK)

{

equipmentComponents[form.Id] = (form.ComponentName, form.Count);

LoadData();

}

}

}

private void ButtonDel\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

if (MessageBox.Show("Удалить запись", "Вопрос", MessageBoxButtons.YesNo,

MessageBoxIcon.Question) == DialogResult.Yes)

{

try

{

equipmentComponents.Remove(Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value));

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

LoadData();

}

}

}

private void ButtonUpdate\_Click(object sender, EventArgs e)

{

LoadData();

}

private void ButtonSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBoxName.Text))

{

MessageBox.Show("Заполните название", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (string.IsNullOrEmpty(textBoxPrice.Text))

{

MessageBox.Show("Заполните цену", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (equipmentComponents == null || equipmentComponents.Count == 0)

{

MessageBox.Show("Заполните компоненты", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

logic.CreateOrUpdate(new EquipmentBindingModel{Id = id, EquipmentName = textBoxName.Text, Price = Convert.ToDecimal(textBoxPrice.Text), EquipmentDevices = equipmentComponents});

MessageBox.Show("Сохранение прошло успешно", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ButtonCancel\_Click(object sender, EventArgs e)

{

DialogResult = DialogResult.Cancel;

Close();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.ViewModels;

using Unity;

namespace SecureShopView

{

public partial class FormEquipmentDevices : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

public int Id

{

get { return Convert.ToInt32(comboBoxDevice.SelectedValue); }

set { comboBoxDevice.SelectedValue = value; }

}

public string ComponentName { get { return comboBoxDevice.Text; } }

public int Count

{

get { return Convert.ToInt32(textBoxCount.Text); }

set

{

textBoxCount.Text = value.ToString();

}

}

public FormEquipmentDevices(DeviceLogic logic)

{

InitializeComponent();

List<DeviceViewModel> list = logic.Read(null);

if (list != null)

{

comboBoxDevice.DisplayMember = "DeviceName";

comboBoxDevice.ValueMember = "Id";

comboBoxDevice.DataSource = list;

comboBoxDevice.SelectedItem = null;

}

}

private void ButtonSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBoxCount.Text))

{

MessageBox.Show("Заполните поле Количество", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (comboBoxDevice.SelectedValue == null)

{

MessageBox.Show("Выберите компонент", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

DialogResult = DialogResult.OK;

Close();

}

private void ButtonCancel\_Click(object sender, EventArgs e)

{

DialogResult = DialogResult.Cancel;

Close();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using Unity;

namespace SecureShopView

{

public partial class FormEquipments : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly EquipmentLogic logic;

public FormEquipments(EquipmentLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void FormEquipments\_Load(object sender, EventArgs e)

{

LoadData();

}

private void LoadData()

{

try

{

Program.ConfigGrid(logic.Read(null), dataGridView);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void ButtonAdd\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormEquipment>();

if (form.ShowDialog() == DialogResult.OK)

{

LoadData();

}

}

private void ButtonEdit\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

var form = Container.Resolve<FormEquipment>();

form.Id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

if (form.ShowDialog() == DialogResult.OK)

{

LoadData();

}

}

}

private void ButtonDelete\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

if (MessageBox.Show("Удалить запись", "Вопрос", MessageBoxButtons.YesNo,

MessageBoxIcon.Question) == DialogResult.Yes)

{

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

logic.Delete(new EquipmentBindingModel { Id = id });

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

LoadData();

}

}

}

java лучше чем C#

private void ButtonUpdate\_Click(object sender, EventArgs e)

{

LoadData();

}

private void dataGridView\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

LoadData();

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using System;

using System.Windows.Forms;

using Unity;

namespace SecureShopView

{

public partial class FormImplementer : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

public int Id { set { id = value; } }

private readonly ImplementerLogic logic;

private int? id;

public FormImplementer(ImplementerLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void FormImplementer\_Load(object sender, EventArgs e)

{

if (id.HasValue)

{

try

{

var view = logic.Read(new ImplementerBindingModel { Id = id })?[0];

if (view != null)

{

textBoxName.Text = view.Name;

textBoxWorkTime.Text = view.WorkingTime.ToString();

textBoxPauseTime.Text = view.PauseTime.ToString();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

private void ButtonCancel\_Click(object sender, EventArgs e)

{

DialogResult = DialogResult.Cancel;

Close();

}

private void ButtonSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBoxName.Text))

{

MessageBox.Show("Вы не ввели имя", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (string.IsNullOrEmpty(textBoxWorkTime.Text))

{

MessageBox.Show("Вы не ввели время работы", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (string.IsNullOrEmpty(textBoxPauseTime.Text))

{

MessageBox.Show("Вы не ввели время перерыва", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

logic.CreateOrUpdate(new ImplementerBindingModel

{

Id = id,

Name = textBoxName.Text,

WorkingTime = Convert.ToInt32(textBoxWorkTime.Text),

PauseTime = Convert.ToInt32(textBoxPauseTime.Text)

});

MessageBox.Show("Сохранено", "Message", MessageBoxButtons.OK, MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch (Exception ex)

{

.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

С

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopView;

using System;

using System.Windows.Forms;

using Unity;

namespace SecureShopView

{

public partial class FormImplementers : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly ImplementerLogic logic;

public FormImplementers(ImplementerLogic logic)

{

InitializeComponent();

this.logic = logic;

LoadData();

}

private void LoadData()

{

try

{

каракатица

Program.ConfigGrid(logic.Read(null), dataGridView);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void ButtonEdit\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

var form = Container.Resolve<FormImplementer>();

form.Id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

if (form.ShowDialog() == DialogResult.OK)

{

LoadData();

}

}

}

private void ButtonDelete\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

if (MessageBox.Show("Delete entry", "Question", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)

{

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

logic.Delete(new ImplementerBindingModel { Id = id });

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

LoadData();

}

}

}

private void ButtonUpdate\_Click(object sender, EventArgs e)

{

LoadData();

}

private void ButtonAdd\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormImplementer>();

if (form.ShowDialog() == DialogResult.OK)

{

LoadData();

}

}

private void FormImplementers\_Load(object sender, EventArgs e)

{

LoadData();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Unity;

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

namespace SecureShopView

{

public partial class FormMain : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly OrderLogic \_orderLogic;

private readonly ReportLogic \_reportLogic;

private readonly WorkModeling \_workModeling;

private readonly SecureShopBackUpAbstractLogic \_backUpAbstractLogic;

public FormMain(OrderLogic orderLogic, ReportLogic reportLogic, WorkModeling workModeling, SecureShopBackUpAbstractLogic backUpAbstractLogic)

{

InitializeComponent();

\_orderLogic = orderLogic;

\_reportLogic = reportLogic;

\_workModeling = workModeling;

\_backUpAbstractLogic = backUpAbstractLogic;

LoadData();

}

private void FormMain\_Load(object sender, EventArgs e)

{

LoadData();

}

private void LoadData()

{

try

{

Program.ConfigGrid(\_orderLogic.Read(null), dataGridView);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void DevicesToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormDevices>();

form.ShowDialog();

}

private void EquipmentsToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormEquipments>();

form.ShowDialog();

}

private void ButtonCreateOrder\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormCreateOrder>();

form.ShowDialog();

LoadData();

}

private void ButtonTakeOrderInWork\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

\_orderLogic.TakeOrderInWork(new ChangeStatusBindingModel { OrderId = id });

LoadData();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void ButtonOrderReady\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

\_orderLogic.FinishOrder(new ChangeStatusBindingModel { OrderId = id });

LoadData();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void ButtonPayOrder\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count == 1)

{

int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);

try

{

\_orderLogic.PayOrder(new ChangeStatusBindingModel { OrderId = id });

LoadData();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void DevicesToolStripMenuReportItem\_Click(object sender, EventArgs e)

{

using (var dialog = new SaveFileDialog { Filter = "docx|\*.docx" })

{

if (dialog.ShowDialog() == DialogResult.OK)

{

\_reportLogic.SaveDevicesToWordFile(new ReportBindingModel

{

FileName = dialog.FileName

});

MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

private void DeviceEquipmentsToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormReportEquipmentDevices>(); form.ShowDialog();

}

private void OrdersToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormReportOrders>(); form.ShowDialog();

}

private void ButtonRef\_Click(object sender, EventArgs e)

{

LoadData();

}

private void ClientsToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormClients>();

form.ShowDialog();

}

private void ImplementersToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormImplementers>();

form.ShowDialog();

}

private void StartWorkToolStripMenuItem\_Click(object sender, EventArgs e)

{

\_workModeling.DoWork();

LoadData();

}

private void MessagesToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = Container.Resolve<FormMessages>();

form.ShowDialog();

}

private void CreateBackUpToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

if (\_backUpAbstractLogic != null)

{

var fbd = new FolderBrowserDialog();

if (fbd.ShowDialog() == DialogResult.OK)

{

\_backUpAbstractLogic.CreateArchive(fbd.SelectedPath);

MessageBox.Show("Бекап создан", "Сообщение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

using System;

using System.Windows.Forms;

using SecureShopBusinessLogic.BusinessLogics;

using Unity;

namespace SecureShopView

{

public partial class FormMessages : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly MailLogic logic;

public FormMessages(MailLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void FormMessages\_Load(object sender, EventArgs e)

{

try

{

Program.ConfigGrid(logic.Read(null), dataGridView);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using System;

using System.Windows.Forms;

using Unity;

namespace SecureShopView

{

public partial class FormReportEquipmentDevices : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly ReportLogic logic;

public FormReportEquipmentDevices(ReportLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void FormReportEquipmentDevices\_Load(object sender, EventArgs e)

{

try

{

var dict = logic.GetEquipmentDevice();

if (dict != null)

{

dataGridView.Rows.Clear();

foreach (var elem in dict)

{

dataGridView.Rows.Add(new object[] { elem.EquipmentName, "", "" });

foreach (var listElem in elem.Devices)

{

dataGridView.Rows.Add(new object[] { "", listElem.Item1, listElem.Item2 });

}

dataGridView.Rows.Add(new object[] { "Итого", "", elem.TotalCount });

dataGridView.Rows.Add(new object[] { });

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ButtonSaveToExcel\_Click(object sender, EventArgs e)

{

using (var dialog = new SaveFileDialog { Filter = "xlsx|\*.xlsx" })

{

if (dialog.ShowDialog() == DialogResult.OK)

{

try

{

logic.SaveEquipmentDeviceToExcelFile(new ReportBindingModel

{

FileName = dialog.FileName

});

MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

}

}

using SecureShopBusinessLogic.BindingModels;

using SecureShopBusinessLogic.BusinessLogics;

using Microsoft.Reporting.WinForms;

using System;

using System.Windows.Forms;

using Unity;

namespace SecureShopView

{

public partial class FormReportOrders : Form

{

[Dependency]

public new IUnityContainer Container { get; set; }

private readonly ReportLogic logic;

public FormReportOrders(ReportLogic logic)

{

InitializeComponent();

this.logic = logic;

}

private void buttonCreateReport\_Click(object sender, EventArgs e)

{

if (dateTimePicker1.Value.Date >= dateTimePicker2.Value.Date)

{

MessageBox.Show("Дата начала должна быть меньше даты окончания",

"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

ReportParameter parameter = new ReportParameter("ReportParameterPeriod",

"c " +

dateTimePicker1.Value.ToShortDateString() +

" по " +

dateTimePicker2.Value.ToShortDateString());

reportViewer1.LocalReport.SetParameters(parameter);

var dataSource = logic.GetOrders(new ReportBindingModel

{

DateFrom = dateTimePicker1.Value,

DateTo = dateTimePicker2.Value

});

ReportDataSource source = new ReportDataSource("DataSetOrders",

dataSource);

if (source != null)

{

reportViewer1.LocalReport.DataSources.Clear();

}

reportViewer1.LocalReport.DataSources.Add(source);

reportViewer1.RefreshReport();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

private void buttonSave\_Click(object sender, EventArgs e)

{

if (dateTimePicker1.Value.Date >= dateTimePicker2.Value.Date)

{

MessageBox.Show("Дата начала должна быть меньше даты окончания",

"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

using (var dialog = new SaveFileDialog { Filter = "pdf|\*.pdf" })

{

if (dialog.ShowDialog() == DialogResult.OK)

{

try

{

logic.SaveOrdersToPdfFile(new ReportBindingModel

{

FileName = dialog.FileName,

DateFrom = dateTimePicker1.Value,

DateTo = dateTimePicker2.Value

});

MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

private void FormReportOrders\_Load(object sender, EventArgs e)

{

}

}

}

using SecureShopBusinessLogic.BusinessLogics;

using SecureShopBusinessLogic.Interfaces;

using SecureShopBusinessLogic.HelperModels;

using SecureShopDatabaseImplement.Implements;

using System;

using System.Windows.Forms;

using Unity;

using Unity.Lifetime;

using System.Configuration;

using System.Threading;

using SecureShopView;

using System.Collections.Generic;

using SecureShopBusinessLogic.Attributes;

namespace SecureShopView

{

static class Program

{

[STAThread]

static void Main()

{

var container = BuildUnityContainer();

MailLogic.MailConfig(new MailConfig

{

SmtpClientHost = ConfigurationManager.AppSettings["SmtpClientHost"],

SmtpClientPort = Convert.ToInt32(ConfigurationManager.AppSettings["SmtpClientPort"]),

MailLogin = ConfigurationManager.AppSettings["MailLogin"],

MailPassword = ConfigurationManager.AppSettings["MailPassword"],

});

var timer = new System.Threading.Timer(new TimerCallback(MailCheck), new MailCheckInfo

{

PopHost = ConfigurationManager.AppSettings["PopHost"],

PopPort = Convert.ToInt32(ConfigurationManager.AppSettings["PopPort"]),

Storage = container.Resolve<IMessageInfoStorage>(),

ClientStorage = container.Resolve<IClientStorage>()

}, 0, 10000);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(container.Resolve<FormMain>());

}

private static IUnityContainer BuildUnityContainer()

{

var currentContainer = new UnityContainer();

currentContainer.RegisterType<IDeviceStorage, DeviceStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<IOrderStorage, OrderStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<IEquipmentStorage, EquipmentStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<IClientStorage, ClientStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<DeviceLogic>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<OrderLogic>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<EquipmentLogic>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<ClientLogic>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<ReportLogic>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<IImplementerStorage, ImplementerStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<ImplementerStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<IMessageInfoStorage, MessageInfoStorage>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<MailLogic>(new HierarchicalLifetimeManager());

currentContainer.RegisterType<SecureShopBackUpAbstractLogic, BackUpLogic>(new HierarchicalLifetimeManager());

return currentContainer;

}

private static void MailCheck(object obj)

{

MailLogic.MailCheck((MailCheckInfo)obj);

}

public static void ConfigGrid<T>(List<T> data, DataGridView grid)

{

var type = typeof(T);

var config = new List<string>();

grid.Columns.Clear();

foreach (var prop in type.GetProperties())

{

// получаем список атрибутов

var attributes =

prop.GetCustomAttributes(typeof(ColumnAttribute), true);

if (attributes != null && attributes.Length > 0)

{

foreach (var attr in attributes)

{

// ищем нужный нам атрибут

if (attr is ColumnAttribute columnAttr)

{

config.Add(prop.Name);

var column = new DataGridViewTextBoxColumn

{

Name = prop.Name,

ReadOnly = true,

HeaderText = columnAttr.Title,

Visible = columnAttr.Visible,

Width = columnAttr.Width

};

if (columnAttr.GridViewAutoSize != GridViewAutoSize.None)

{

column.AutoSizeMode = (DataGridViewAutoSizeColumnMode)Enum.Parse(typeof(DataGridViewAutoSizeColumnMode), columnAttr.GridViewAutoSize.ToString());

}

grid.Columns.Add(column);

}

}

}

}

// добавляем строки

foreach (var elem in data)

{

List<object> objs = new List<object>();

foreach (var conf in config)

{

var value = elem.GetType().GetProperty(conf).GetValue(elem);

objs.Add(value);

}

grid.Rows.Add(objs.ToArray());

}

}

}

}

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ   
УНИВЕРСИТЕТ»

Факультет информационных систем и технологий

Кафедра Информационные системы

Отчет по практике

защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка прописью)

Руководитель   
практики от   
университета \_\_ст.преподаватель Эгов Е.Н.\_\_

(должность)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (ФИО)

**ОТЧЕТ по** \_\_\_**УЧЕБНОЙ** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ практике

(вид практики)

Студента \_\_Эмирян Владимира Леонидовича\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО)

Направление (специальность,

профиль) подготовки 09.03.04 Программная инженерия

Группа \_\_ПИбд-11\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Место прохождения практики УлГТУ, кафедра Информационные системы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование профильной организации, подразделение университета)

Ульяновск 2020 г.

Оглавление

1F. Красивые часы - 1 3

1G. Красивые часы - 2 4

1H. Выбирайте типы данных с умом 5

1I. Округление 6

3I. Второй максимум последовательности 7

3J. Дисперсия последовательности 8

4F. Проверка на простоту - 1 10

4G. Проверка на простоту – 2. 11

4H. Перебор перестановок 12

5G. Сортировка выбором 17

5H. Наилучший участок 19

5I. Умножение матриц 21

5J. Сапёр 23

6A. Символ в код 28

6B. Количество букв 29

6C. Древний шифр 30

6D. Макс и перестановочный шифр 31

6E. Постиранный пароль 33

6F. C++ и Java 35

6G. Слова 37

# 1F. Красивые часы - 1

**Задание:**

Дано время, отображаемое электронными часами. Определите общее число минут, прошедших с полуночи.

**Входные данные:**

Ввод содержит целые числа H и M (00 ≤ H ≤ 23, 00 ≤ M ≤ 59), разделённые двоеточием — соответственно количество часов и минут, отображаемое часами. Числа H и M записаны с использованием двух десятичных цифр каждое (возможно, с ведущими нулями).

**Выходные данные:**

Выведите одно целое число — количество минут, прошедших с полуночи.

**Код:**

using System;  
  
public class Test  
{  
  public static void Main()  
  {  
    System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
      
      string str = Console.ReadLine();  
      char[] time = new char[5];  
      time = str.ToCharArray();  
      int hrs = 0;  
      int min = 0;  
      int res = 0;  
        
      str = "";  
      str = str + time[0] + time[1];  
      hrs = Convert.ToInt32(str);  
      str = "";  
      str = str + time[3] + time[4];  
      min = Convert.ToInt32(str);  
      res = hrs \* 60 + min;  
      Console.WriteLine(res);  
  }  
}

# 1G. Красивые часы - 2

**Задание:**

Дано общее число минут, прошедших с полуночи. Определите время, отображаемое электронными часами.

**Входные данные:**

Ввод содержит целое число M (0 ≤ M ≤ 1439) — количество минут, прошедших с полуночи.

**Выходные данные:**

Выведите два целых числа, разделённые двоеточием, — количество часов и минут, отображаемое часами. Числа должны быть записаны с использованием двух десятичных цифр каждое (возможно, с ведущими нулями).

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int time = Convert.ToInt32(Console.ReadLine());  
        int hrs = 0;  
        int min = 0;  
  
        hrs = time / 60;  
        min = time - (hrs \* 60);  
  
        string n1 = Convert.ToString(hrs);  
        string n2 = Convert.ToString(min);  
  
        if(hrs >= 0 && hrs < 10)  
    {  
            n1 = "0" + n1;  
    }  
  
        if (min >= 0 && min < 10)  
        {  
            n2 = "0" + n2;  
        }  
  
        Console.WriteLine("{0}:{1}", n1, n2);  
    }  
}

# 1H. Выбирайте типы данных с умом

**Задание:**

Вычислите произведение двух целых чисел.

**Входные данные:**

Ввод содержит целые числа A и B ( - 106 ≤ A, B ≤ 106).

**Выходные данные:**

Выведите значение A × B.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string time = Console.ReadLine();  
        string n1 = "";  
        string n2 = "";  
        int len = time.Length;  
        long a;  
        long b;  
  
        char[] ch = new char[len];  
  
        ch = time.ToCharArray();  
  
        int i = 0;  
        while(ch[i] != 32)  
    {  
            n1 += ch[i];  
            i++;  
    }  
  
        i++;  
        while (i != len)  
        {  
            n2 += ch[i];  
            i++;  
        }  
  
        a = Convert.ToInt64(n1);  
        b = Convert.ToInt64(n2);  
        Console.WriteLine(a \* b);  
    }  
}

# 1I. Округление

**Задание:**

Даны целые числа A и B. Выведите их частное:

· с округлением до целого;

· с округлением до 2 знаков после запятой;

· с округлением до 6 знаков после запятой;

· с округлением до 10 знаков после запятой.

**Входные данные:**

Ввод содержит целые числа A и B (1 ≤ A, B ≤ 1000).

**Выходные данные:**

Выведите в отдельных строках одно целое и три вещественных числа — значения A/B с необходимым округлением.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string time = Console.ReadLine();  
        string n1 = "";  
        string n2 = "";  
        int len = time.Length;  
        double a;  
        double b;  
  
        char[] ch = new char[len];  
  
        ch = time.ToCharArray();  
  
        int i = 0;  
        while(ch[i] != 32)  
    {  
            n1 += ch[i];  
            i++;  
    }  
  
        i++;  
        while (i != len)  
        {  
            n2 += ch[i];  
            i++;  
        }  
  
        a = Convert.ToDouble(n1);  
        b = Convert.ToDouble(n2);  
  
        double res = a / b;  
        Console.WriteLine(Math.Round(res));  
        Console.WriteLine(String.Format("{0:N2}", Math.Round(res, 2, MidpointRounding.AwayFromZero)));  
        Console.WriteLine(String.Format("{0:N6}", Math.Round(res, 6, MidpointRounding.AwayFromZero)));  
        Console.WriteLine(String.Format("{0:N10}", Math.Round(res, 10, MidpointRounding.AwayFromZero)));  
    }  
}

# 3I. Второй максимум последовательности

**Задание:**

Дана последовательность положительных целых чисел, завершающаяся числом 0. Определите второй по величине этой последовательности.

При решении этой задачи нельзя использовать массивы.

**Входные данные:**

Ввод содержит два или более целых чисел Ai (1 ≤ Ai ≤ 2·10^6) — элементы последовательности. После всех чисел Ai следует число 0.

Общее количество чисел Ai во входных данных не превосходит 2·106.

**Выходные данные:**

Выведите одно целое число — второй максимум последовательности.

**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if (fnum.Length == 0 && c == '0')  
                return 0;  
            if (c != ' ')  
                fnum += c;  
        } while (c != ' ');  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int a = -1;  
        int max1 = 0;  
        int max2 = 0;  
  
        while(a != 0)  
    {  
            a = readNum();  
            if (a >= max1)  
            {  
                max2 = max1;  
                max1 = a;  
            }  
            else if (a >= max2)  
            {  
                max2 = a;  
            }  
        }  
  
        Console.WriteLine(max2);  
    }  
}

# 3J. Дисперсия последовательности

**Задание:**

Дана последовательность положительных целых чисел, завершающаяся числом 0. Определите дисперсию этой последовательности. Чтобы избежать погрешностей вещественных чисел, **при вычислениях округляйте *A* и *D* до меньших целых**. При решении этой задачи нельзя использовать массивы.  
**Входные данные:**  
Ввод содержит два или более целых чисел *Ai* (1 ≤ *Ai* ≤ 2·106) — элементы последовательности. После всех чисел *Ai* следует число 0. Общее количество чисел *Ai* во входных данных не превосходит 2·106.  
**Выходные данные:**  
Выведите одно целое число — дисперсию последовательности.  
**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if (fnum.Length == 0 && c == '0')  
                return 0;  
            if (c != ' ')  
                fnum += c;  
        } while (c != ' ');  
        return int.Parse(fnum);  
    }  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int a, n = 0;  
        long sa = 0, sk = 0;  
        a = readNum();  
        while (a != 0)  
        {  
            sa += (long)a;  
            sk += (long)a\* a;  
            n++;  
            a = readNum();  
        }  
        Console.WriteLine ((long)(n \* (long)(sa / n) \* (long)(sa / n) + sk - 2 \* (long)(sa / n) \* sa) / (n - 1));  
    }  
}

# 4F. Проверка на простоту - 1

**Задание**:

Дано натуральное число N. Проверьте, является ли оно простым.

Число называется простым, если оно имеет только 2 различных делителя.

**Входные данные:**

Ввод содержит целое число N (1 ≤ N ≤ 10^6).

**Выходные данные:**

Если N является простым, выведите YES, иначе выведите NO.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int a = Convert.ToInt32(Console.ReadLine());  
        string str = "YES";  
  
        if(a == 1)  
    {  
            str = "NO";  
    }  
      
        for(int i = a - 1; i > 1; i--)  
    {  
            if(a % i == 0)  
      {  
                str = "NO";  
      }  
    }  
      
  
        Console.WriteLine(str);  
    }  
}

# 4G. Проверка на простоту – 2.

**Задание:**

Дано натуральное число N. Проверьте, является ли оно простым.

Число называется простым, если оно имеет только 2 различных делителя.

Для оптимизации проверки воспользуйтесь тем фактом, что если N = A \* B, то min(A,B)  ≤  √N.

**Входные данные:**

Ввод содержит целое число N (1 ≤ N ≤ 10^12).

**Выходные данные:**

Если N является простым, выведите YES, иначе выведите NO.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        long a = Convert.ToInt64(Console.ReadLine());  
        string str = "YES";  
  
        if(a == 1)  
    {  
            str = "NO";  
    }  
  
        long sqr = (long)Math.Sqrt(a);  
  
        for(long i = sqr; i > 1; i--)  
    {  
            if(a % i == 0)  
      {  
                str = "NO";  
      }  
    }  
      
  
        Console.WriteLine(str);  
    }

# 5F. Наиболее частый элемент – 2

**Задание:**

Дан массив, состоящий из N целых чисел, упорядоченных по неубыванию. Определите, какое число встречается в массиве чаще всего.

**Входные данные:**

Первая строка содержит целое число N (1 ≤ N ≤ 10^5) — размер массива.

Вторая строка содержит N целых чисел Ai (1 ≤ Ai ≤ 109, Ai - 1 ≤ Ai ≤ Ai + 1) — элементы массива.

**Выходные данные:**

Выведите одно целое число — наиболее часто встречающийся элемент массива. Если подходящих ответов несколько, выведите максимальный из них.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            /\*if (fnum.Length == 0 && c == '0')  
                return 0;\*/  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, k = 1, kmax = 0, nextNumber = 0, temp = 0;  
        int n = Convert.ToInt32(Console.ReadLine());  
  
        number = readNum();  
        for (int i = 0; i < n - 1; i++)  
        {  
            nextNumber = readNum();  
            if (number == nextNumber)  
            {  
                k++;  
            }  
            else if (k >= kmax)  
            {  
                kmax = k;  
                temp = number;  
                k = 1;  
            }  
            else  
      {  
                k = 1;  
            }  
            number = nextNumber;  
        }  
        if (k >= kmax)  
        {  
            kmax = k;  
            temp = number;  
        }  
  
        Console.WriteLine(temp);  
    }  
}

# 5G. Сортировка выбором

**Задание:**

Дан массив A, состоящий из N целых чисел. Отсортируйте его по не убыванию при помощи сортировки выбором:

· Определите минимальный из элементов A0, ..., AN - 1 и обменяйте его с A0;

· Определите минимальный из элементов A1, ..., AN - 1 и обменяйте его с A1;

· ...

· Определите минимальный из элементов AN - 2, AN - 1 и обменяйте его с AN - 2

**Входные данные:**

Первая строка содержит целое число N (1 ≤ N ≤ 10^4) — размер массива.

Вторая строка содержит N целых чисел Ai ( - 10^9 ≤ Ai ≤ 10^9) — элементы массива.

**Выходные данные:**

Выведите N целых чисел — элементы массива в порядке не убывания.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, temp = 0;  
        int n = Convert.ToInt32(Console.ReadLine());  
        int[] array = new int[n];  
  
        for(int i = 0; i < n; i++)  
    {  
            number = readNum();  
            array[i] = number;  
    }  
  
        int k = 0;  
        for (int i = 0; i < n; i++)  
        {  
            int min = array[k];  
            int kmin = k;  
            for(int j = k + 1; j < n; j++)  
      {  
                if (array[j] < min)  
                {  
                    min = array[j];  
                    kmin = j;  
                }  
            }  
            temp = array[k];  
            array[k] = array[kmin];  
            array[kmin] = temp;  
            k++;  
        }  
  
        for (int i = 0; i < n; i++)  
        {  
            Console.Write(array[i] + " ");  
        }  
    }  
}

# 5H. Наилучший участок

**Задание:**

Дан целочисленный двумерный массив размера N × M. Строки массива нумеруются с единицы сверху вниз, столбцы — с единицы слева направо. Определите в данном массиве участок 3 × 3, сумма элементов которого является максимальной, и выведите координаты левого верхнего угла этого участка.

**Входные данные:**

Первая строка содержит целые числа N и M (3 ≤ N, M ≤ 100) — количество строк и столбцов массива соответственно.

Следующие N строк описывают массив. Каждая из них содержит M целых чисел Aij ( - 1000 ≤ Aij ≤ 1000) — элементы массива.

**Выходные данные:**

Выведите два целых числа R и C — соответственно номер строки и номер столбца левого верхнего элемента наилучшего участка. Если подходящих ответов несколько, выведите ответ с наименьшим значением R; если при этом подходящих ответов всё ещё несколько, выведите ответ с наименьшим значением C.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, temp = 0;  
        int n1 = readNum();  
        int m1 = readNum();  
        int[,] array1 = new int[n1, m1];  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m1; j++)  
            {  
                array1[i, j] = readNum();  
            }  
        }  
  
        int maxSum = -10000;  
        int iTemp = 0;  
        int jTemp = 0;  
        for (int i = 0; i < n1 - 2; i++)  
        {  
            for (int j = 0; j < m1 - 2; j++)  
            {  
                int sum = array1[i, j] + array1[i + 1, j] + array1[i + 2, j] + array1[i, j + 1] + array1[i + 1, j + 1] + array1[i + 2, j + 1] + array1[i, j + 2] + array1[i + 1, j + 2] + array1[i + 2, j + 2];  
                if (sum > maxSum)  
                {  
                    iTemp = i + 1;  
                    jTemp = j + 1;  
                    maxSum = sum;  
                }  
            }  
        }  
  
        Console.WriteLine(iTemp + " " + jTemp);  
    }  
}

# 5I. Умножение матриц

**Задание:**

Даны целочисленные матрицы A и B. Определите их произведение A × B.

**Входные данные:**

Первая строка содержит целые числа NA и MA (1 ≤ NA, MA ≤ 100) — размеры матрицы A.

Следующие NA строк описывают матрицу A. Каждая их них содержит MA целых чисел Aij ( - 100 ≤ Aij ≤ 100) — элементы матрицы.

Следующая строка содержит целые числа NB и MB (1 ≤ NB, MB ≤ 100, NB = MA) — размеры матрицы B.

Следующие NB строк описывают матрицу B. Каждая их них содержит MB целых чисел Bij ( - 100 ≤ Bij ≤ 100) — элементы матрицы.

**Выходные данные:**

Выведите NA строк, каждая из которых содержит MB целых чисел — элементы результирующей матрицы.

**Код:**

using System;  
using System.Data.SqlTypes;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int number = 0, temp = 0;  
        int n1 = readNum();  
        int m1 = readNum();  
        int[,] array1 = new int[n1, m1];  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m1; j++)  
            {  
                array1[i, j] = readNum();  
            }  
        }  
  
        int n2 = readNum();  
        int m2 = readNum();  
        int[,] array2 = new int[n2, m2];  
  
        for (int i = 0; i < n2; i++)  
        {  
            for (int j = 0; j < m2; j++)  
            {  
                array2[i, j] = readNum();  
            }  
        }  
  
    int[,] array3 = new int[101, 101];  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m2; j++)  
            {  
                for (int k = 0; k < m1; k++)  
                {  
                    array3[i, j] += array1[i, k] \* array2[k, j];  
                }  
            }  
        }  
  
        for (int i = 0; i < n1; i++)  
        {  
            for (int j = 0; j < m2; j++)  
            {  
                Console.Write(array3[i, j] + " ");  
            }  
            Console.WriteLine();  
        }  
    }  
}

# 5J. Сапёр

**Задание:**

Игра «Сапёр». Поле для её простейшего варианта имеет размер 8 × 8. 10 клеток поля содержат мины, а в каждой из остальных клеток находится цифра от 0 до 8, равная количеству мин в соседних клетках. Вам известно расположение мин на поле. Восстановите все цифры.

**Входные данные:**

Ввод содержит 8 строк, описывающих игровое поле. Каждая из них содержит 8 символов «.» (клетка пуста) либо «\*» (клетка содержит мину).

Гарантируется, что поле содержит ровно 10 мин.

**Выходные данные:**

Выведите 8 строк, каждая из которых содержит 8 символов: клетки, содержащие мины, должны содержать символ «\*», остальные клетки — цифру от 0 до 8, соответствующую количеству мин в соседних клетках.

**Код:**

using System;

public class Test

{

static int readNum()

{

char c;

string fnum = "";

do

{

c = Convert.ToChar(Console.Read());

if ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57))

fnum += c;

} while ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57));

if (fnum == "") return 0;

return int.Parse(fnum);

}

public static void Main()

{

System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");

int[,] array = new int[8, 8];

char[,] res = new char[8, 8];

for (int i = 0; i < 8; i++)

{

string str = Console.In.ReadLine();

char[] temp = str.ToCharArray();

for (int j = 0; j < 8; j++)

{

if (temp[j] == '\*')

{

array[i, j] = 1;

}

else

{

array[i, j] = 0;

}

}

}

if (array[0, 0] == 0)

{

res[0, 0] += (char)(array[0, 1] + array[1, 0] + array[1, 1]);

}

else

res[0, 0] = '\*';

if (array[0, 7] == 0)

{

res[0, 7] += (char)(array[0, 6] + array[1, 6] + array[1, 7]);

}

else

res[0, 7] = '\*';

if (array[7, 0] == 0)

{

res[7, 0] += (char)(array[6, 0] + array[7, 1] + array[6, 1]);

}

else

res[7, 0] = '\*';

if (array[7, 7] == 0)

{

res[7, 7] += (char)(array[7, 6] + array[6, 7] + array[6, 6]);

}

else

res[7, 7] = '\*';

for (int i = 1; i < 7; i++)

{

if (array[0, i] == 0)

{

res[0, i] += (char)(array[0, i - 1] + array[0, i + 1] + array[1, i] + array[1, i - 1] + array[1, i + 1]);

}

else

res[0, i] = '\*';

}

for (int i = 1; i < 7; i++)

{

if (array[i, 0] == 0)

{

res[i, 0] += (char)(array[i - 1, 0] + array[i + 1, 0] + array[i, 1] + array[i - 1, 1] + array[i + 1, 1]);

}

else

res[i, 0] = '\*';

}

for (int i = 1; i < 7; i++)

{

if (array[i, 7] == 0)

{

res[i, 7] += (char)(array[i - 1, 7] + array[i + 1, 7] + array[i, 6] + array[i - 1, 6] + array[i + 1, 6]);

}

else

res[i, 7] = '\*';

}

for (int i = 1; i < 7; i++)

{

if (array[7, i] == 0)

{

res[7, i] += (char)(array[7, i - 1] + array[7, i + 1] + array[6, i] + array[6, i - 1] + array[6, i + 1]);

}

else

res[7, i] = '\*';

}

for (int i = 1; i < 7; i++)

{

for (int j = 1; j < 7; j++)

{

if (array[i, j] == 0)

{

res[i, j] += (char)(array[i - 1, j - 1] + array[i - 1, j] + array[i - 1, j + 1] + array[i, j - 1] + array[i, j + 1] + array[i + 1, j - 1] + array[i + 1, j] + array[i + 1, j + 1]);

}

else

res[i, j] = '\*';

}

}

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

if (res[i, j] == '\*')

Console.Write((char)res[i, j]);

else

Console.Write((int)res[i, j]);

}

Console.WriteLine();

}

}

}

# 6A. Символ в код

**Задание:**

Дан символ. Выведите его номер в кодировочной таблице ASCII.

**Входные данные:**

Ввод содержит один символ, ASCII-код которого находится в диапазоне [33; 126].

**Выходные данные:**

Выведите ASCII-код указанного символа.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        char c = (char)Console.Read();  
        Console.WriteLine((int)c);  
    }  
}

# 6B. Количество букв

**Задание:**

Дана строка. Определите, сколько её символов являются буквами.

**Входные данные:**

Ввод содержит строку S (1 ≤ |S| ≤ 105), состоящую из латинских букв, цифр и знаков препинания.

**Выходные данные:**

Выведите одно целое число — количество букв в строке S.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str = Console.ReadLine();  
        char[] ch = str.ToCharArray();  
        int k = 0;  
  
        for(int i = 0; i < str.Length; i++)  
    {  
            if((ch[i] >= 'A' && ch[i] <= 'Z') || (ch[i] >= 'a' && ch[i] <= 'z'))   
            {  
                k++;  
        }  
    }  
        Console.WriteLine(k);  
    }  
}

# 

# 6C. Древний шифр

**Задание:**

Одним из самых старых известных методов шифрования является атбаш, при помощи которого зашифрованы многие христианские тексты, например, часть свитков Мёртвого моря. Этот шифр достаточно прост: первая буква алфавита («a») в нём заменяется на последнюю («z»), вторая («b») — на предпоследнюю («y»), ..., последняя («z») — на первую («a»). Формально, буква, находящаяся на позиции X от начала алфавита, заменяется буквой, находящейся на позиции X от конца алфавита. Символы, отличные от букв, не изменяются.

Сможете ли вы расшифровать сообщение, зашифрованное этим древним методом?

**Входные данные:**

Ввод содержит одну строку длиной от 1 до 100 символов, состоящую из больших и малых латинских букв, пробелов и знаков препинания, — зашифрованное сообщение.

**Выходные данные:**

Выведите расшифрованное сообщение. Регистр букв должен быть сохранён.

**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n'))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n'));  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string array = Console.ReadLine();  
        char[] a = array.ToCharArray();  
  
        for (int i = 0; i < array.Length; i++)  
        {  
            if (a[i] >= 'A' && a[i] <= 'Z')  
                a[i] = (char)('Z' - a[i] + 'A');  
            else if (a[i] >= 'a' && a[i] <= 'z')  
                a[i] = (char)('z' - a[i] + 'a');  
        }  
  
        Console.WriteLine(a);  
    }  
}

# 6D. Макс и перестановочный шифр

**Задание:**

В этой программе сам пользователь будет указывать, на какую букву должна заменяться каждая из 26 букв латинского алфавита. Регистр букв при шифровании сохраняется, а пробелы и знаки препинания остаются без изменений.

**Входные данные:**

Первая строка содержит 26 различных целых чисел Ci (1 ≤ Ci ≤ 26). Буква, имеющая в алфавите номер i, заменяется на букву с номером Ci.

Вторая строка содержит текст S (1 ≤ |S| ≤ 105), состоящий из латинских букв, пробелов и знаков препинания — зашифрованное сообщение.

**Выходные данные:**

Выведите расшифрованный текст.

**Код:**

using System;  
  
public class Test  
{  
    static int readNum()  
    {  
        char c;  
        string fnum = "";  
        do  
        {  
            c = Convert.ToChar(Console.Read());  
            if ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57))  
                fnum += c;  
        } while ((c != ' ') && (c != '\n') && (c >= 48) && (c <= 57));  
        if (fnum == "") return 0;  
        return int.Parse(fnum);  
    }  
  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        int[] key = new int[26];  
        int i = 0;  
        string str = Console.In.ReadLine();  
        string[] str2 = str.Split(' ');  
        foreach (var str3 in str2)  
        {  
            key[i] = Convert.ToInt32(str3);  
            i++;  
        }  
        string strTemp = Console.In.ReadLine();  
        char[] a = strTemp.ToCharArray();  
        int n = strTemp.Length;  
        for (i = 0; i < n; i++)  
        {  
            if (a[i] >= 'A' && a[i] <= 'Z')  
                for (int j = 0; j < 27; j++)  
                {  
                    if ((key[j] - 1) == (a[i] - 'A'))  
                    {  
                        a[i] = (char)('A' + j);  
                        break;  
                    }  
                }  
            else if (a[i] >= 'a' && a[i] <= 'z')  
                for (int j = 0; j < 27; j++)  
                {  
                    if ((key[j] - 1) == (a[i] - 'a'))  
                    {  
                        a[i] = (char)('a' + j);  
                        break;  
                    }  
                }  
        }  
        Console.WriteLine(a);  
    }  
}

# 6E. Постиранный пароль

**Задание:**

У вас есть набор букв, сколько предполагаемых “паролей” могли быть составлены из этого набора букв?

Помогите Максу справиться с его задачей.

**Входные данные:**

Первая строка содержит строку S (1 ≤ |S| ≤ 100), состоящую из строчных латинских букв.

Вторая строка содержит целое число N (0 ≤ N ≤ 1000) — количество вариантов пароля.

Следующие N строк описывают варианты пароля. Каждая из них содержит строку Pi (1 ≤ |Pi| ≤ 20), состоящую из строчных латинских букв.

**Выходные данные:**

Выведите количество вариантов пароля, которые могли быть составлены из имеющихся букв.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string strVocabulary = Console.ReadLine();  
        char[] vocabulary = strVocabulary.ToCharArray();  
        int n = Convert.ToInt32(Console.ReadLine());  
        int res = 0;  
        char[] tempVocabulary = new char[101];  
        for (int i = 0; i < n; i++)  
        {  
            string tempStr = Console.ReadLine();  
            char[] password = tempStr.ToCharArray();  
            vocabulary.CopyTo(tempVocabulary, 0);  
            int check = 0;  
            for (int j = 0; j < tempStr.Length; j++)  
            {  
                for (int k = 0; k < strVocabulary.Length; k++)  
                {  
                    if (password[j] == tempVocabulary[k])  
                    {  
                        tempVocabulary[k] = (char)45;  
                        check++;  
                        break;  
                    }  
                }  
            }  
  
            if (check == tempStr.Length)  
            {  
                res++;  
            }  
        }  
  
        Console.WriteLine(res);  
    }  
}

# 

# 6F. C++ и Java

**Задание:**

Некоторые студенты предпочитают использовать при решении задач практики язык программирования C++, некоторые — язык программирования Java. И у того, и у другого имеются как преимущества, так и недостатки. Если не удаётся составить решение на одном языке, в отдельных случаях имеет смысл попробовать использовать другой.

Для именования переменных в C++ используется следующее соглашение: идентификатор записывается только строчными буквами, отдельные слова в составе идентификатора разделяются символом подчёркивания. Примерами идентификаторов языка C++ являются «selection\_sort», «key\_value\_pair» и «max\_f\_p\_s».

Для именования переменных в Java используется так называемый «camelCase»: отдельные слова в составе идентификатора записываются слитно, первые буквы всех слов кроме начального являются заглавными. Примерами идентификаторов языка Java являются «selectionSort», «keyValuePair» и «maxFPS».

Правильно измените имя переменной.

**Входные данные:**

Ввод содержит строку S (1 ≤ |S| ≤ 100) — имя переменной. Гарантируется, что строка S является корректным идентификатором в рамках рассматриваемых стилей.

**Выходные данные:**

Если имя переменной соответствует стилю C++, выведите его аналог для стиля Java. Если имя переменной соответствует стилю Java, выведите его аналог для стиля C++. Если вариантов ответа несколько, выведите любой.

**Код:**

using System;  
  
public class Test  
{  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str1 = Console.ReadLine();  
        char[] chStr = str1.ToCharArray();  
        string str = "";  
        int f = 0;  
  
        for (int i = 0; i < str1.Length; i++)  
        {  
            if (chStr[i] == '\_')  
      {  
                f = 1;  
      }  
        }  
          
  
        for (int i = 0; i < str1.Length; i++)  
    {  
           if(f == 0)  
       {  
                if (chStr[i] >= 'a' && chStr[i] <= 'z')  
                {  
                    str += chStr[i];  
                }  
                else  
                {  
                    str += '\_';  
                    str += (char)((chStr[i] - 'A') + 'a');  
                }  
           }  
           else  
       {  
                if ((chStr[i] >= 'a' && chStr[i] <= 'z') || (chStr[i] >= 'A' && chStr[i] <= 'Z'))  
                {  
                    str += chStr[i];  
                }  
                else  
                {  
                    str += (char)((chStr[i + 1] - 'a') + 'A');  
                    i++;  
                }  
           }  
    }  
        Console.WriteLine(str);  
    }  
}

# 6G. Слова

**Задание:**

Дан текст. Переведите все содержащиеся в нём слова в верхний регистр и выведите по отдельности, сохраняя их порядок.

**Входные данные:**

Ввод содержит строку S (1 ≤ |S| ≤ 1000), состоящую из латинских букв, пробелов и знаков препинания. Гарантируется, что строка содержит не менее одного слова.

**Выходные данные:**

Выведите в отдельных строках переведённые в верхний регистр слова, содержащиеся в строке S, в порядке появления. Повторяющиеся слова следует выводить несколько раз.

**Код:**

using System;  
  
public class Test  
{  
    static bool isalpha(char c)  
    {  
        if (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z')  
        {  
            return true;  
        }  
        return false;  
  
    }  
  
    static char upreg(char c)  
    {  
        if (c >= 'a' && c <= 'z')  
        {  
            return (char)(c - 'a' + 'A');  
        }  
        else  
        {  
            return c;  
        }  
  
  
    }  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str = Console.ReadLine();  
        str += '.';  
        string word = "";  
        for (int i = 0; i < str.Length; i++)  
        {  
            if (isalpha(str[i]))  
            {  
                word += upreg(str[i]);  
            }  
            else  
            {  
                if (word != "")  
                {  
                    Console.WriteLine(word);  
                    word = "";  
                }  
            }  
        }

# 6I. Автоформатирование

**Задание:**

Составьте программу, которая автоматически форматирует текст по следующим правилам:

* В тексте не должно быть двух или более пробелов подряд;
* Перед точками не должно быть пробелов, между точкой и следующим словом должен быть одиночный пробел;
* Первые буквы предложений должны быть заглавными, остальные буквы — строчными.

**Входные данные:**

Ввод содержит строку *S* (10 ≤ |*S*| ≤ 1000), состоящую из латинских букв, пробелов и точек, — текст, который нужно отформатировать. Гарантируется, что исходный текст начинается с буквы и заканчивается точкой, а в отформатированном тексте не будет двух подряд идущих точек.  
**Выходные данные:**  
Выведите отформатированный текст.  
**Код:**

using System;  
  
public class Test  
{  
    static bool isalpha(char c)  
    {  
        if (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z')  
        {  
            return true;  
        }  
        return false;  
  
    }  
  
    static char toup(char c)  
    {  
        if (c >= 'a' && c <= 'z')  
        {  
            return (char)(c - 'a' + 'A');  
        }  
        else  
        {  
            return c;  
        }  
  
  
    }  
  
    static char tolow(char c)  
    {  
        if (c >= 'A' && c <= 'Z')  
        {  
            return (char)(c - 'A' + 'a');  
        }  
        else  
        {  
            return c;  
        }  
    }  
    public static void Main()  
    {  
        System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");  
  
        string str = Console.ReadLine();  
        char[] ch = str.ToCharArray();  
        string tmp = "";  
        int f = 1;  
        for(int i = 0; i < str.Length; i++)  
        {  
          if (isalpha(ch[i]))   
          {  
            if (f == 1) {  
              if(i > 0)   
              {  
                tmp += ' ';    
              }  
                tmp += toup(ch[i]);  
                f = 0;  
              }  
              else {  
                tmp += tolow(ch[i]);  
            }      
          }  
          else if(ch[i] == '.')   
          {  
            tmp += '.';  
            f = 1;  
          }  
          else if (ch[i] == ' ')   
          {  
            if(i > 0 && i < str.Length - 1)   
            {  
              if(ch[i - 1] == '.')  
              {  
                ch[i] = '.';  
              }  
              else if(isalpha(ch[i + 1]) && f != 1)  
              {  
                tmp += ' ';  
              }  
            }  
          }  
            
           
        }  
        if(ch[str.Length - 1] != '.')  
      {  
          tmp += '.';    
      }  
      Console.WriteLine(tmp);  
          
    }  
}МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ

Кафедра «Информационные системы»

Дисциплина «Технологии программирования»