Министерство образования Республики Беларусь

Оршанский колледж учреждения образования

«Витебский государственный университет имени П.М. Машерова»

Специальность 5-04-0612-02 «Разработка и сопровождение программного обеспечения

информационных систем»

Группа 3ПОИС23

**УЧЕБНАЯ ПРАКТИКА**

**ПО ПРОГРАММИРОВАНИЮ**

с 29 сентября 2025 года по 25 октября 2025 года

**ОТЧЕТ**

Выполнила С.О. Жданова

Руководитель практики М.А. Алейников

2025

**СОДЕРЖАНИЕ**

Изм.

№ докум.

Подпись

Дата

Лист

3

*УП 5-04-0612-02ПОИС23.004.004*

Разраб.

*Жданова С.О.*

Провер.

*Алейников М.А.*

Реценз.

Н.Контр.

Утверд.

Лит.

Листов

68

*Оршанский колледж*

*ВГУ имени П.М. Машерова*

*Группа 3ПОИС23*

1. Модуль 1. Основы C#
2. Модуль 2. Объектно-ориентированное программирование (ООП)
3. Модуль 3. Задания по делегатам
4. Модуль 4. Задания по интерфейсам
5. Модуль 5. Графический интерфейс
6. Модуль 6. Базы данных
7. Модуль 7. Проектная работа
8. Модуль 8. Реализация проекта
   1. Постановка задачи
      1. Описание предметной области
      2. Характеристика решаемой задачи
      3. Разработка моделей решаемой задачи
   2. Проектирование программного модуля
      1. Сбор исходных материалов
      2. Проектирование информационной модели
      3. Описание входных и выходных данных
   3. Реализация программного модуля
      1. Описание диаграммы классов разрабатываемого проекта
      2. Описание структуры разрабатываемого проекта
      3. Проектирование и реализация интерфейса программы
9. Модуль 9. Совершенствование навыков ООП

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

Приложение А Текст программы модуля 8

**Модуль 1. Основы C#**

**Модуль 1.1 Основы С#**

**Задания (вариант №4)**

1. Создайте программу, которая генерирует случайное число от 1 до 100 и предлагает пользователю угадать это число. При каждой попытке программа дает подсказку, больше или меньше введенное число загаданного.

2. Напишите программу, которая находит среднее арифметическое из трех введенных пользователем чисел.

3. Напишите программу, которая принимает на вход две строки и определяет, является ли вторая строка подстрокой первой строки.

4. Напишите программу, которая создает массив из 10 целых чисел, заполняет его случайными числами и выводит сумму всех элементов массива.

5. Напишите программу, которая выводит на экран числа от 1 до 100. Если число делится на 3, вместо него выводится «Fizz», если на 5 – «Buzz», а если на оба – «FizzBuzz».

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 1.1.1. Игра – угадывание числа.

using System;

namespace GuessTheNumber

{

class Program

{

static void Main(string[] args)

{

Random random = new Random();

bool playAgain = true;

while (playAgain)

{

int secretNumber = random.Next(1, 101);

int guess;

int attempts = 0;

int maxAttempts = 10;

bool hasGuessed = false;

Console.WriteLine("Я загадал число от 1 до 100. У тебя есть 10 попыток, чтобы угадать!");

while (!hasGuessed && attempts < maxAttempts)

{

Console.Write($"Попытка {attempts + 1}. Введи своё число: ");

string input = Console.ReadLine();

attempts++;

if (!int.TryParse(input, out guess))

{

Console.WriteLine("Пожалуйста, введи число!");

attempts--;

continue;

}

if (guess == secretNumber)

{

Console.WriteLine($"Поздравляю! Ты угадал число за {attempts} попыток!");

hasGuessed = true;

}

else if (guess < secretNumber)

{

Console.WriteLine("Моё число больше!");

}

else

{

Console.WriteLine("Моё число меньше!");

}

}

if (!hasGuessed)

{

Console.WriteLine($"Увы, попытки закончились! Загаданное число было {secretNumber}.");

}

Console.Write("Хочешь сыграть ещё раз? (да/нет): ");

string answer = Console.ReadLine().ToLower();

playAgain = (answer == "да" || answer == "yes");

}

Console.WriteLine("Спасибо за игру!");

}

}

}

Выполнение программного кода из листинга 1.1.1 представлено на рисунке 1.1.1.



Рисунок 1.1.1 – Результат игры – угадывание числа

**Задание 2.**

Листинг 1.1.2. Код программы для нахождения среднего арифметического числа (сумма чисел, делённая на 3).

using System;

namespace AverageCalculator

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите три числа для вычисления среднего арифметического.");

Console.Write("Введите первое число: ");

string input1 = Console.ReadLine();

if (!double.TryParse(input1, out double number1))

{

Console.WriteLine("Ошибка: введите корректное число!");

return;

}

Console.Write("Введите второе число: ");

string input2 = Console.ReadLine();

if (!double.TryParse(input2, out double number2))

{

Console.WriteLine("Ошибка: введите корректное число!");

return;

}

Console.Write("Введите третье число: ");

string input3 = Console.ReadLine();

if (!double.TryParse(input3, out double number3))

{

Console.WriteLine("Ошибка: введите корректное число!");

return;

}

double average = (number1 + number2 + number3) / 3;

Console.WriteLine($"Среднее арифметическое чисел {number1}, {number2}, {number3} равно {average:F2}");

}

}

}

Выполнение программного кода из листинга 1.1.2 представлено на рисунке 1.1.2.

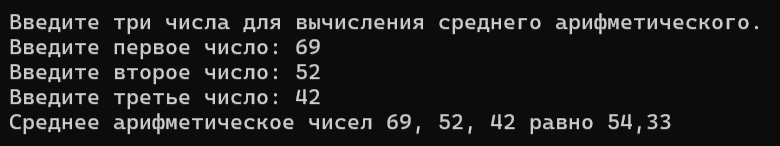


Рисунок 1.1.2 – Результат нахождения среднего арифметического числа

**Задание 3.**

Листинг 1.1.3. Код программы для проверки строки на подстроку.

using System;

namespace SubstringChecker

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа проверяет, является ли вторая строка подстрокой первой (без учёта регистра).");

while (true)

{

Console.Write("Введите первую строку: ");

string firstString = Console.ReadLine();

if (string.IsNullOrEmpty(firstString))

{

Console.WriteLine("Ошибка: первая строка не может быть пустой!");

continue;

}

Console.Write("Введите вторую строку: ");

string secondString = Console.ReadLine();

if (string.IsNullOrEmpty(secondString))

{

Console.WriteLine("Ошибка: вторая строка не может быть пустой!");

continue;

}

bool isSubstring = firstString.ToLower().Contains(secondString.ToLower());

int position = firstString.ToLower().IndexOf(secondString.ToLower());

if (isSubstring)

{

Console.WriteLine($"Да, строка \"{secondString}\" является подстрокой строки \"{firstString}\".");

Console.WriteLine($"Она начинается с позиции {position + 1} (нумерация с 1).");

}

else

{

Console.WriteLine($"Нет, строка \"{secondString}\" не является подстрокой строки \"{firstString}\".");

}

Console.Write("Хотите проверить ещё раз? (да/нет): ");

string continueAnswer = Console.ReadLine().ToLower();

if (continueAnswer != "да" && continueAnswer != "yes")

{

break;

}

Console.WriteLine();

}

Console.WriteLine("Спасибо за использование программы!");

}

}

}

Выполнение программного кода из листинга 1.1.3 представлено на рисунке 1.1.3.

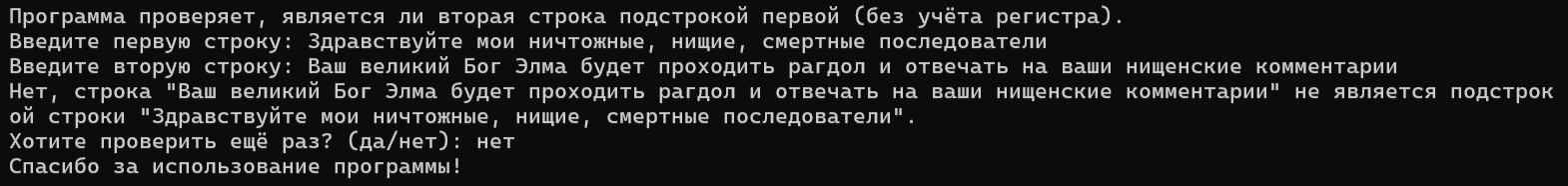


Рисунок 1.1.3 – Результат проверки строки на подстроку

**Задание 4.**

Листинг 1.1.4. Код программы для создания массива и вычисления суммы всех элементов массива.

using System;

namespace ArraySum

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа создаёт массив из 10 случайных целых чисел и вычисляет их сумму.");

Random random = new Random();

int[] numbers = new int[10];

for (int i = 0; i < numbers.Length; i++)

{

numbers[i] = random.Next(1, 101);

}

Console.Write("Созданный массив: ");

for (int i = 0; i < numbers.Length; i++)

{

Console.Write(numbers[i]);

if (i < numbers.Length - 1)

{

Console.Write(", ");

}

}

Console.WriteLine();

int sum = 0;

for (int i = 0; i < numbers.Length; i++)

{

sum += numbers[i];

}

Console.WriteLine($"Сумма всех элементов массива: {sum}");

}

}

}

Выполнение программного кода из листинга 1.1.4 представлено на рисунке 1.1.4.

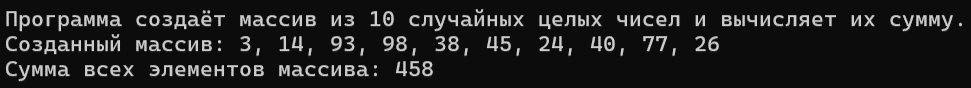


Рисунок 1.1.4 – Результат создания массива и вычисления суммы всех чисел

**Задание 5.**

Листинг 1.1.5. Код программы для вывода чисел.

using System;

namespace FizzBuzz

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа выводит числа от 1 до 100, заменяя числа, делящиеся на 3, на 'Fizz', на 5 — на 'Buzz', на оба — на 'FizzBuzz'.");

for (int i = 1; i <= 100; i++)

{

if (i % 3 == 0 && i % 5 == 0)

{

Console.WriteLine("FizzBuzz");

}

else if (i % 3 == 0)

{

Console.WriteLine("Fizz");

}

else if (i % 5 == 0)

{

Console.WriteLine("Buzz");

}

else

{

Console.WriteLine(i);

}

}

}

}

}

Выполнение программного кода из листинга 1.1.5 представлено на рисунке 1.1.5.

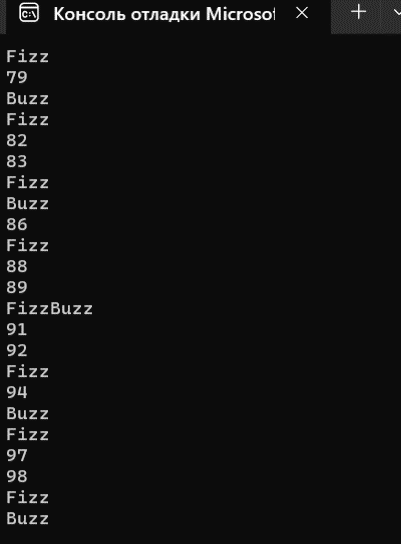


Рисунок 1.1.5 – Результат вывода чисел со своими особенностями

Ссылка на репозиторий: [https://github.com/sarrikzhdyn/GuessTheNumb](https://github.com/sarrikzhdyn/GuessTheNumb%20)

**Модуль 1.2 МАССИВЫ, СТРОКИ, ПЕРЕКЛЮЧАТЕЛИ**

**Задания**

Задача 1. Ввести размер массива N и значения его элементов. Нормировать элементы массива, разделив их на значение максимального по модулю элемента. Вывести значения элементов измененного массива.

Задача 2. Определить и инициализировать целочисленный массив из 10-ти элементов. Ввести целое число и заменить им значение максимального элемента в массиве.

Задача 3. Вычислить К простым числам. Значение К ввести с клавиатуры. Вывести значения чисел, размещая их по 10 на строке.

Задача 4. Определить целочисленный массив из К элементов. Присвоить элементам случайные значения из диапазона [А, В). Найти индексы минимального и максимального элементов массива. Вывести значения элементов, расположенных между найденными (включая найденные).

Задача 5. Определить символьный массив из К элементов. Присвоить элементам случайные значения букв русского алфавита. Создать новый массив, поместив в него только согласные буквы из первого массива. Значение К ввести с клавиатуры. Вывести элементы обоих массивов.

Задача 6. Определить вещественный массив из 10-ти элементов. Присвоить элементам случайные значения из диапазона [–10, 10). Сформировать массив индексов, которые нумеруют элементы первого массива в порядке возрастания их значений.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 1.2.1. Код программы для нормирования элементов массива.

using System;

namespace ArrayNormalization

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа нормирует массив, деля его элементы на максимальный по модулю элемент.");

Console.Write("Введите размер массива N: ");

string inputN = Console.ReadLine();

int n;

if (!int.TryParse(inputN, out n) || n <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число!");

return;

}

double[] array = new double[n];

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

{

Console.Write($"Введите элемент {i + 1}: ");

string input = Console.ReadLine();

if (!double.TryParse(input, out array[i]))

{

Console.WriteLine("Ошибка: введите корректное число!");

return;

}

}

Console.WriteLine("\nИсходный массив:");

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

{

Console.Write($"{array[i]} ");

}

Console.WriteLine();

double maxAbs = Math.Abs(array[0]);

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

{

if (Math.Abs(array[i]) > maxAbs)

{

maxAbs = Math.Abs(array[i]);

}

}

if (maxAbs == 0)

{

Console.WriteLine("Ошибка: все элементы массива равны нулю, деление невозможно!");

return;

}

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

{

array[i] /= maxAbs;

}

Console.WriteLine("\nНормированный массив:");

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

{

Console.Write($"{array[i]:F4} ");

}

Console.WriteLine();

}

}

}

Выполнение программного кода из листинга 1.2.1 представлено на рисунке 1.2.1.

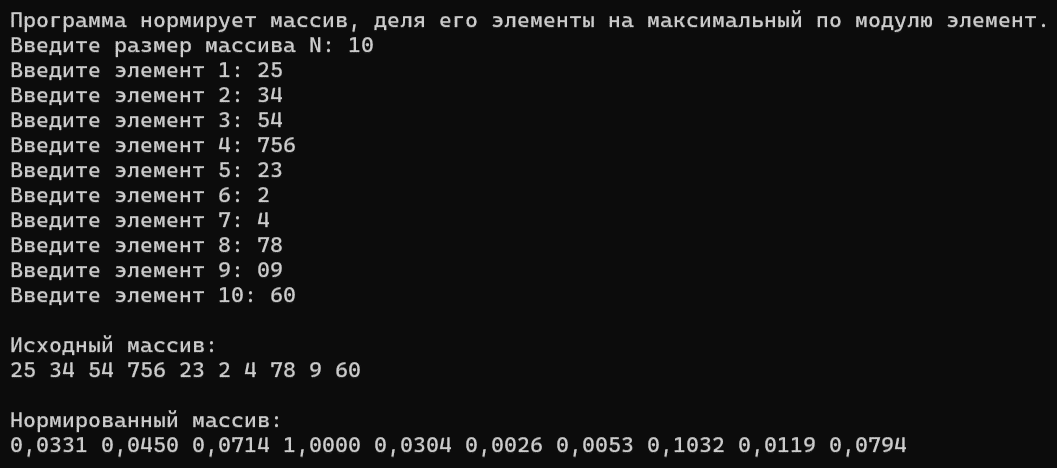


Рисунок 1.2.1 – Результат нормирования элементов массива

**Задание 2.**

Листинг 1.2.2. Код программы для инициализации целочисленного массива.

using System;

namespace ReplaceMaxElement

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа создаёт массив из 10 целых чисел, показывает максимальный элемент и заменяет его на введённое число.");

Random random = new Random();

int[] array = new int[10];

for (int i = 0; i < array.Length; i++)

{

array[i] = random.Next(1, 101);

}

Console.WriteLine("Исходный массив:");

for (int i = 0; i < array.Length; i++)

{

Console.Write($"{array[i]} ");

}

Console.WriteLine();

int maxIndex = 0;

for (int i = 1; i < array.Length; i++)

{

if (array[i] > array[maxIndex])

{

maxIndex = i;

}

}

Console.WriteLine($"Максимальный элемент: {array[maxIndex]}");

Console.Write("Введите целое число для замены максимального элемента: ");

string input = Console.ReadLine();

int newValue;

if (!int.TryParse(input, out newValue))

{

Console.WriteLine("Ошибка: введите корректное целое число!");

return;

}

array[maxIndex] = newValue;

Console.WriteLine("Изменённый массив:");

for (int i = 0; i < array.Length; i++)

{

Console.Write($"{array[i]} ");

}

Console.WriteLine();

}

}

}

Выполнение программного кода из листинга 1.2.2 представлено на рисунке 1.2.2.

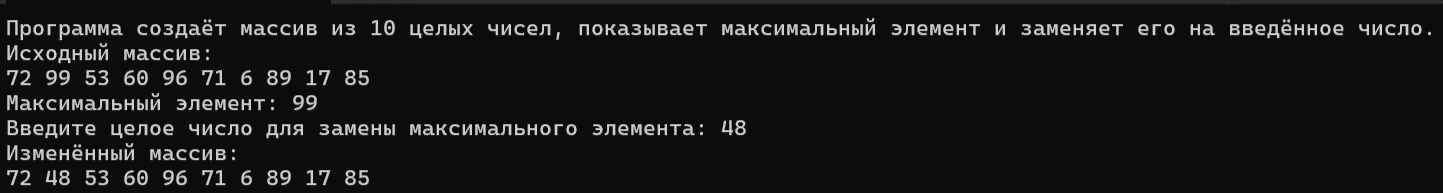


Рисунок 1.2.2 – Результат инициализации целочисленного массива

**Задание 3.**

Листинг 1.2.3. Код программы для работы с простыми числами К.

using System;

namespace PrimeNumbers

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа находит K простых чисел и выводит их по 10 в строке.");

Console.Write("Введите количество простых чисел K: ");

string inputK = Console.ReadLine();

int k;

if (!int.TryParse(inputK, out k) || k <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число!");

return;

}

int[] primes = new int[k];

int count = 0;

int number = 2;

while (count < k)

{

if (IsPrime(number))

{

primes[count] = number;

count++;

}

number++;

}

Console.WriteLine("\nПростые числа:");

for (int i = 0; i < primes.Length; i++)

{

Console.Write($"{primes[i],-4}");

if ((i + 1) % 10 == 0 || i == primes.Length - 1)

{

Console.WriteLine();

}

}

}

static bool IsPrime(int number)

{

if (number <= 1)

{

return false;

}

for (int i = 2; i <= Math.Sqrt(number); i++)

{

if (number % i == 0)

{

return false;

}

}

return true;

}

}

}

Выполнение программного кода из листинга 1.2.3 представлено на рисунке 1.2.3.

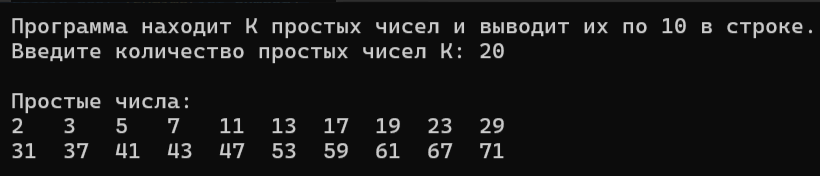


Рисунок 1.2.3 – Результат вывода чисел в консоль

**Задание 4.**

Листинг 1.2.4. Код программы для работы с целочисленным массивом.

using System;

namespace MinMaxIndices

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа создаёт массив из K элементов, заполняет случайными числами из диапазона [A, B) и выводит элементы между минимальным и максимальным.");

Console.Write("Введите размер массива K: ");

string inputK = Console.ReadLine();

int k;

if (!int.TryParse(inputK, out k) || k <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число для K!");

return;

}

Console.Write("Введите начало диапазона A: ");

string inputA = Console.ReadLine();

int a;

if (!int.TryParse(inputA, out a))

{

Console.WriteLine("Ошибка: введите целое число для A!");

return;

}

Console.Write("Введите конец диапазона B: ");

string inputB = Console.ReadLine();

int b;

if (!int.TryParse(inputB, out b) || b <= a)

{

Console.WriteLine("Ошибка: B должно быть целым числом и больше A!");

return;

}

Random random = new Random();

int[] array = new int[k];

for (int i = 0; i < array.Length; i++)

{

array[i] = random.Next(a, b);

}

Console.WriteLine("\nИсходный массив:");

for (int i = 0; i < array.Length; i++)

{

Console.Write($"{array[i]} ");

}

Console.WriteLine();

int minIndex = 0;

int maxIndex = 0;

for (int i = 1; i < array.Length; i++)

{

if (array[i] < array[minIndex])

{

minIndex = i;

}

if (array[i] > array[maxIndex])

{

maxIndex = i;

}

}

Console.WriteLine($"\nМинимальный элемент: {array[minIndex]} (индекс {minIndex})");

Console.WriteLine($"Максимальный элемент: {array[maxIndex]} (индекс {maxIndex})");

int startIndex = Math.Min(minIndex, maxIndex);

int endIndex = Math.Max(minIndex, maxIndex);

Console.WriteLine("\nЭлементы между минимальным и максимальным (включая их):");

if (startIndex == endIndex)

{

Console.WriteLine(array[startIndex]);

}

else

{

for (int i = startIndex; i <= endIndex; i++)

{

Console.Write($"{array[i]} ");

}

Console.WriteLine();

}

}

}

}

Выполнение программного кода из листинга 1.2.4 представлено на рисунке 1.2.4.

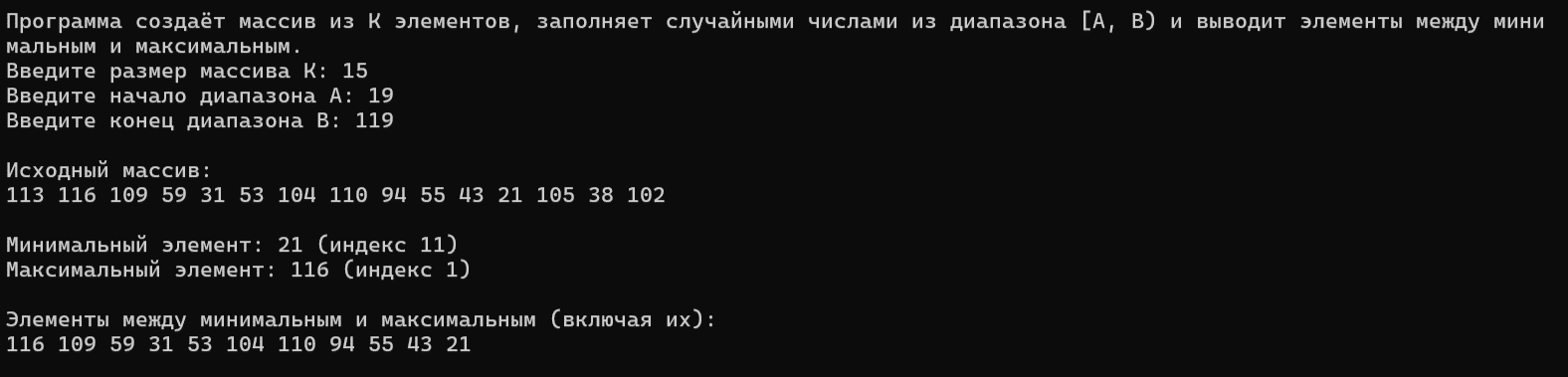


Рисунок 1.2.4 – Результат ранее написанного кода полученный в консоли

**Задание 5.**

Листинг 1.2.5. Код программы для работы с символьным массивом.

using System;

using System.Collections.Generic;

namespace RussianConsonants

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа создаёт массив из K случайных букв русского алфавита и выделяет согласные.");

Console.Write("Введите размер массива K: ");

string inputK = Console.ReadLine();

int k;

if (!int.TryParse(inputK, out k) || k <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число для K!");

return;

}

Random random = new Random();

char[] russianAlphabet = { 'а', 'б', 'в', 'г', 'д', 'е', 'ё', 'ж', 'з', 'и', 'й',

'к', 'л', 'м', 'н', 'о', 'п', 'р', 'с', 'т', 'у', 'ф',

'х', 'ц', 'ч', 'ш', 'щ', 'ъ', 'ы', 'ь', 'э', 'ю', 'я' };

char[] letters = new char[k];

for (int i = 0; i < letters.Length; i++)

{

letters[i] = russianAlphabet[random.Next(0, russianAlphabet.Length)];

}

Console.WriteLine("\nИсходный массив букв:");

for (int i = 0; i < letters.Length; i++)

{

Console.Write($"{letters[i]} ");

}

Console.WriteLine();

List<char> consonants = new List<char>();

char[] vowels = { 'а', 'е', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я' };

for (int i = 0; i < letters.Length; i++)

{

bool isVowel = false;

for (int j = 0; j < vowels.Length; j++)

{

if (letters[i] == vowels[j])

{

isVowel = true;

break;

}

}

if (!isVowel && letters[i] != 'ь' && letters[i] != 'ъ' && letters[i] != 'й')

{

consonants.Add(letters[i]);

}

}

Console.WriteLine("\nМассив согласных букв:");

if (consonants.Count == 0)

{

Console.WriteLine("Согласные буквы отсутствуют.");

}

else

{

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

{

Console.Write($"{consonants[i]} ");

}

Console.WriteLine();

}

}

}

}

Выполнение программного кода из листинга 1.2.5 представлено на рисунке 1.2.5.

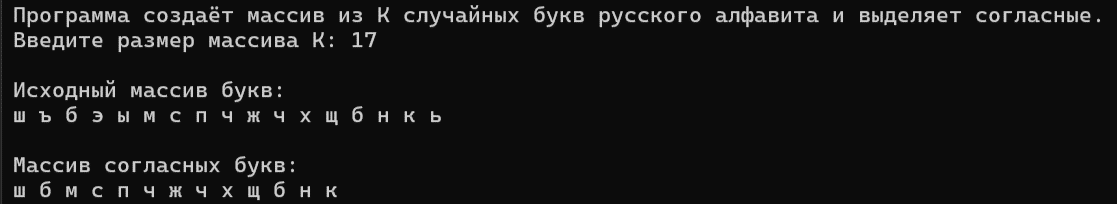


Рисунок 1.2.5 – Результат работы с символьным массивом

**Задание 6.**

Листинг 1.2.6. Код программы для работы с вещественным массивом.

using System;

namespace SortedIndices

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа создаёт массив из 10 вещественных чисел в диапазоне [-10, 10) и формирует массив индексов по возрастанию значений.");

Console.WriteLine("Результаты будут показаны поэтапно. Нажмите Enter для продолжения.");

Console.ReadLine();

Random random = new Random();

double[] numbers = new double[10];

for (int i = 0; i < numbers.Length; i++)

{

numbers[i] = random.NextDouble() \* 20 - 10;

}

Console.WriteLine("\nИсходный массив:");

for (int i = 0; i < numbers.Length; i++)

{

Console.Write($"{numbers[i]:F2} ");

}

Console.WriteLine("\n\nНажмите Enter, чтобы найти минимальный и максимальный элементы.");

Console.ReadLine();

int minIndex = 0;

int maxIndex = 0;

for (int i = 1; i < numbers.Length; i++)

{

if (numbers[i] < numbers[minIndex])

{

minIndex = i;

}

if (numbers[i] > numbers[maxIndex])

{

maxIndex = i;

}

}

Console.WriteLine($"\nМинимальный элемент: {numbers[minIndex]:F2} (индекс {minIndex})");

Console.WriteLine($"Максимальный элемент: {numbers[maxIndex]:F2} (индекс {maxIndex})");

Console.WriteLine("\nНажмите Enter, чтобы сформировать массив индексов.");

Console.ReadLine();

int[] indices = new int[10];

for (int i = 0; i < indices.Length; i++)

{

indices[i] = i;

}

for (int i = 0; i < indices.Length - 1; i++)

{

for (int j = 0; j < indices.Length - 1 - i; j++)

{

if (numbers[indices[j]] > numbers[indices[j + 1]])

{

int temp = indices[j];

indices[j] = indices[j + 1];

indices[j + 1] = temp;

}

}

}

Console.WriteLine("\nМассив индексов (по возрастанию значений):");

for (int i = 0; i < indices.Length; i++)

{

Console.Write($"{indices[i]} ");

}

Console.WriteLine("\n\nНажмите Enter, чтобы показать элементы в порядке возрастания.");

Console.ReadLine();

Console.WriteLine("\nЭлементы массива в порядке возрастания:");

for (int i = 0; i < indices.Length; i++)

{

Console.Write($"{numbers[indices[i]]:F2} ");

}

Console.WriteLine("\n\nПрограмма завершена. Нажмите Enter для выхода.");

Console.ReadLine();

}

}

}

Выполнение программного кода из листинга 1.2.6 представлено на рисунке 1.2.6.

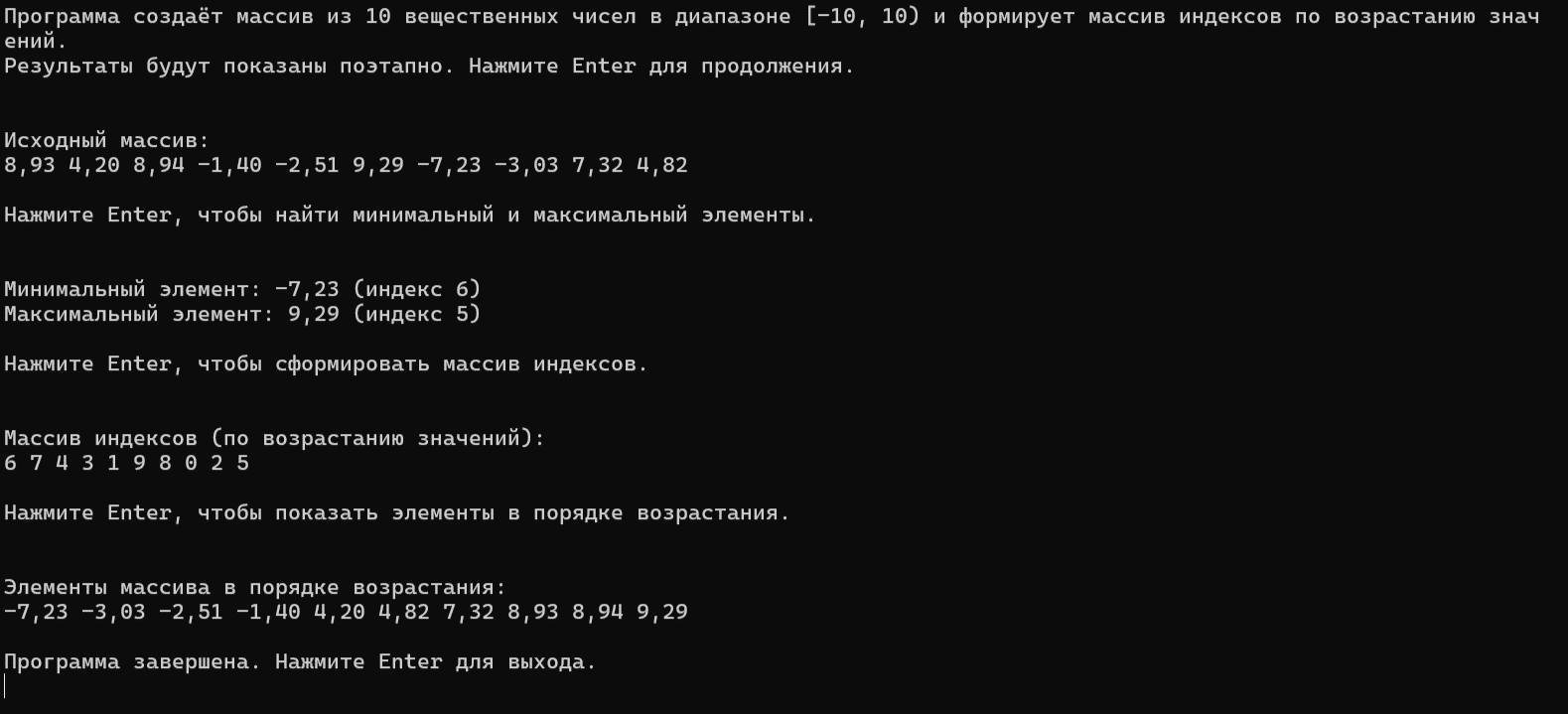


Рисунок 1.2.6 – Результат работы с целочисленным массивом

**Модуль 1.3 СТАТИЧЕСКИЕ МЕТОДЫ (МЕТОДЫ КЛАССОВ)**

**Задания**

Задача 1. Определить функцию (статический метод) для вычисления наибольшего общего делителя двух целых натуральных чисел (Greatest Common Measure). В основной программе, используя функцию, сократить неотрицательную обыкновенную дробь. Дробь вводится с клавиатуры в виде неотрицательного числителя и положительного знаменателя

Задача 2. Присваивая последовательным элементам массива случайные значения от 1 до 9, создать массив с минимальным количеством элементов, сумма которых не превышает заданного пользователем числа.

Задача 3. Сформировав квадратную целочисленную матрицу со случайными значениями элементов, упорядочить ее строки по возрастанию сумм их элементов. Пусть элементы принимают значения в диапазоне от –50 до +50.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 1.3.1. Код программы для вычисления наибольшего общего делителя двух целых натуральных чисел.

using System;

namespace FractionReducer

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа сокращает обыкновенную дробь, используя НОД.");

Console.Write("Введите неотрицательный числитель: ");

string inputNumerator = Console.ReadLine();

int numerator;

if (!int.TryParse(inputNumerator, out numerator) || numerator < 0)

{

Console.WriteLine("Ошибка: числитель должен быть неотрицательным целым числом!");

return;

}

Console.Write("Введите положительный знаменатель: ");

string inputDenominator = Console.ReadLine();

int denominator;

if (!int.TryParse(inputDenominator, out denominator) || denominator <= 0)

{

Console.WriteLine("Ошибка: знаменатель должен быть положительным целым числом!");

return;

}

Console.WriteLine($"\nИсходная дробь: {numerator}/{denominator}");

int gcd = MathHelper.GreatestCommonDivisor(numerator, denominator);

int reducedNumerator = numerator / gcd;

int reducedDenominator = denominator / gcd;

Console.WriteLine($"Сокращённая дробь: {reducedNumerator}/{reducedDenominator}");

}

}

static class MathHelper

{

public static int GreatestCommonDivisor(int a, int b)

{

a = Math.Abs(a);

b = Math.Abs(b);

if (a == 0 && b == 0)

{

return 1;

}

if (a == 0)

{

return b;

}

if (b == 0)

{

return a;

}

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

}

}

Выполнение программного кода из листинга 1.3.1 представлено на рисунке 1.3.1.

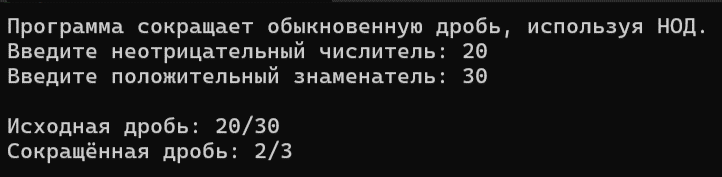


Рисунок 1.3.1 – Результат вычисления наибольшего общего делителя

**Задание 2.**

Листинг 1.3.2. Код программы для создания массива с минимальным количеством элементов.

using System;

using System.Collections.Generic;

namespace MinimalArraySum

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа создаёт массив с минимальным количеством элементов, сумма которых не превышает заданное число.");

Console.Write("Введите максимальную сумму (положительное число): ");

string inputSum = Console.ReadLine();

int sumLimit;

if (!int.TryParse(inputSum, out sumLimit) || sumLimit <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число!");

return;

}

int[] array = ArrayHelper.CreateMinimalArray(sumLimit);

Console.WriteLine("\nСозданный массив:");

ArrayHelper.PrintArray(array);

int sum = ArrayHelper.CalculateSum(array);

Console.WriteLine($"Сумма элементов: {sum}");

}

}

static class ArrayHelper

{

public static int[] CreateMinimalArray(int sumLimit)

{

Random random = new Random();

List<int> tempList = new List<int>();

int remainingSum = sumLimit;

int maxElements = Math.Min(9, sumLimit);

int elementsCount = 1;

while (elementsCount <= maxElements)

{

int targetSumPerElement = remainingSum / elementsCount;

if (targetSumPerElement >= 1)

{

tempList.Clear();

int currentSum = 0;

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

{

int maxValue = Math.Min(9, remainingSum - currentSum);

if (maxValue < 1) break;

int value = random.Next(1, maxValue + 1);

tempList.Add(value);

currentSum += value;

}

if (currentSum <= sumLimit)

{

remainingSum = sumLimit - currentSum;

if (remainingSum == 0 || elementsCount == maxElements)

{

return tempList.ToArray();

}

}

}

elementsCount++;

}

if (tempList.Count == 0 && sumLimit >= 1)

{

tempList.Add(sumLimit);

}

return tempList.ToArray();

}

public static void PrintArray(int[] array)

{

if (array.Length == 0)

{

Console.WriteLine("Массив пустой.");

return;

}

for (int i = 0; i < array.Length; i++)

{

Console.Write($"{array[i]} ");

}

Console.WriteLine();

}

public static int CalculateSum(int[] array)

{

int sum = 0;

for (int i = 0; i < array.Length; i++)

{

sum += array[i];

}

return sum;

}

}

}

Выполнение программного кода из листинга 1.3.2 представлено на рисунке 1.3.2.

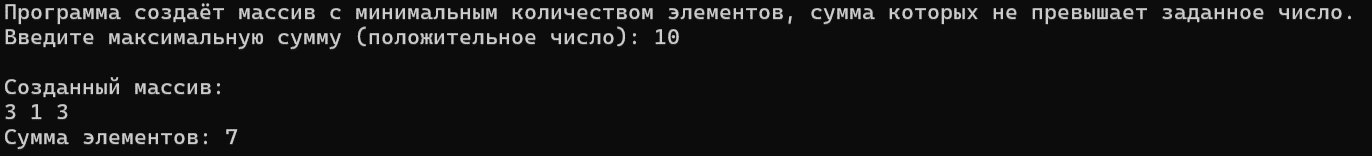


Рисунок 1.3.2 – Результат создания массива

**Задание 3.**

Листинг 1.3.3. Код программы для создания и работы с квадратной целочисленной матрицей.

using System;

namespace MatrixSortBySum

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа формирует прямоугольную матрицу и сортирует её строки по возрастанию сумм элементов.");

Console.Write("Введите количество строк (положительное число): ");

string inputRows = Console.ReadLine();

int rows;

if (!int.TryParse(inputRows, out rows) || rows <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число для строк!");

return;

}

Console.Write("Введите количество столбцов (положительное число): ");

string inputCols = Console.ReadLine();

int cols;

if (!int.TryParse(inputCols, out cols) || cols <= 0)

{

Console.WriteLine("Ошибка: введите положительное целое число для столбцов!");

return;

}

int[,] matrix = MatrixHelper.CreateMatrix(rows, cols);

Console.WriteLine("\nИсходная матрица:");

MatrixHelper.PrintMatrix(matrix);

MatrixHelper.SortRowsBySum(matrix);

Console.WriteLine("\nМатрица после сортировки строк по суммам:");

MatrixHelper.PrintMatrix(matrix);

}

}

static class MatrixHelper

{

public static int[,] CreateMatrix(int rows, int cols)

{

Random random = new Random();

int[,] matrix = new int[rows, cols];

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

{

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

{

matrix[i, j] = random.Next(-50, 51);

}

}

return matrix;

}

public static void PrintMatrix(int[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

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

{

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

{

Console.Write($"{matrix[i, j],4}");

}

Console.WriteLine();

}

}

private static int CalculateRowSum(int[,] matrix, int row)

{

int sum = 0;

int cols = matrix.GetLength(1);

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

{

sum += matrix[row, j];

}

return sum;

}

public static void SortRowsBySum(int[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

int[] sums = new int[rows];

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

{

sums[i] = CalculateRowSum(matrix, i);

}

for (int i = 0; i < rows - 1; i++)

{

for (int j = 0; j < rows - 1 - i; j++)

{

if (sums[j] > sums[j + 1])

{

int tempSum = sums[j];

sums[j] = sums[j + 1];

sums[j + 1] = tempSum;

for (int k = 0; k < cols; k++)

{

int tempValue = matrix[j, k];

matrix[j, k] = matrix[j + 1, k];

matrix[j + 1, k] = tempValue;

}

}

}

}

}

}

}

Выполнение программного кода из листинга 1.3.3 представлено на рисунке 1.3.3.

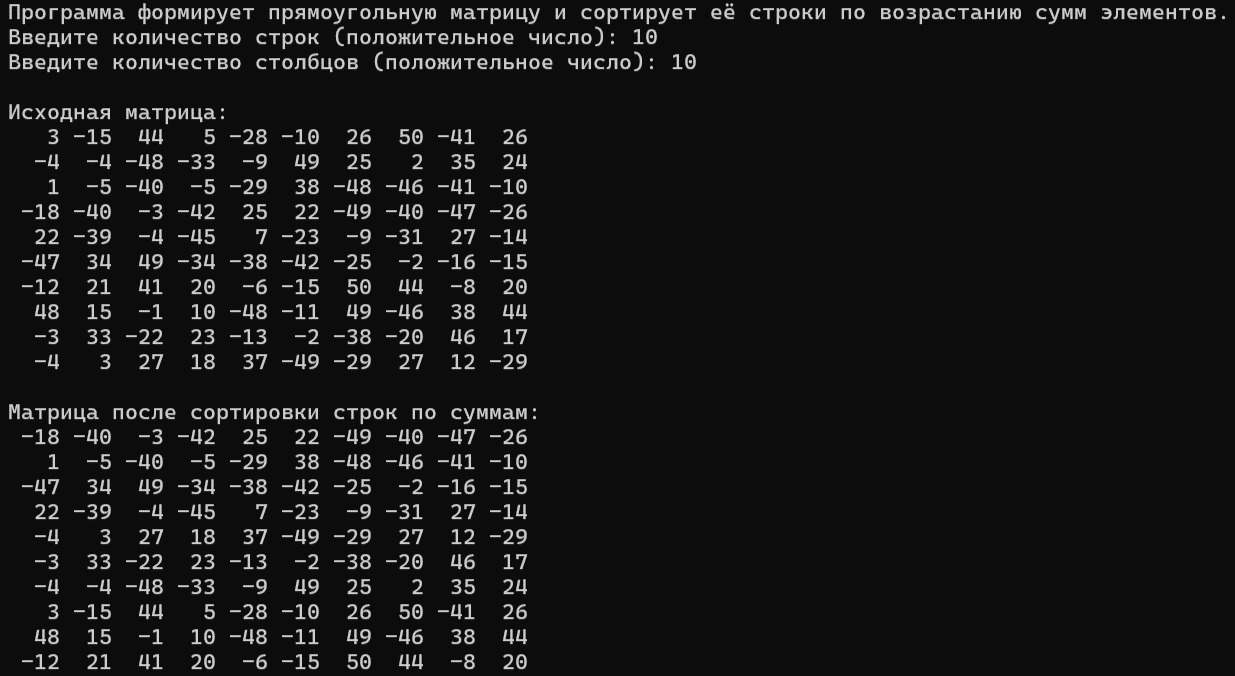


Рисунок 1.3.3 – Результат создания и упорядочения элементов матрицы

**Модуль 2. Объектно-ориентированное программирование (ООП)**

**Задания**

**Задача 1: Создание классов**

Создайте класс Person, представляющий человека. У этого класса должны быть поля для хранения имени, возраста и адреса. Добавьте методы для установки и получения значений этих полей. Затем создайте объекты этого класса и выведите информацию о них.

**Задача 2: Наследование и полиморфизм**

Создайте базовый класс Shape, представляющий геометрическую фигуру, и производные классы Circle и Rectangle. В базовом классе определите метод Area(), который возвращает площадь фигуры, и метод Perimeter(), который возвращает периметр фигуры. В производных классах переопределите эти методы для соответствующих фигур (круг и прямоугольник). Создайте объекты всех классов и выведите их площади и периметры.

**Задача 3: Композиция**

Создайте классы Author и Book. Класс Author должен содержать информацию об авторе (имя и год рождения). Класс Book должен содержать информацию о книге (название, год выпуска и автора). Используйте композицию, чтобы связать объекты Author и Book. Создайте несколько объектов Author и Book и выведите информацию о них.

**Задача 4: Интерфейсы и абстрактные классы**

Создайте интерфейс IDrawable с методом Draw(), который выводит информацию о рисуемом объекте. Создайте классы Circle, Rectangle и Triangle, реализующие этот интерфейс. Создайте массив объектов, реализующих интерфейс IDrawable, и вызовите метод Draw() для каждого из них.

**Задача 5: События**

Создайте класс TemperatureSensor, который генерирует событие TemperatureChanged, когда измеренная температура меняется. Создайте класс Thermostat, который подписывается на событие TemperatureChanged и реагирует на изменение температуры, включая или выключая отопление.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 2.1.1. Код программы для управления информацией о людях.

using System;

using System.Collections.Generic;

namespace PersonClass

{

class Person

{

private string name;

private int age;

private string address;

public void SetName(string newName)

{

if (!string.IsNullOrWhiteSpace(newName))

{

name = newName;

}

else

{

Console.WriteLine("Ошибка: имя не может быть пустым!");

}

}

public string GetName()

{

return name;

}

public void SetAge(int newAge)

{

if (newAge >= 0)

{

age = newAge;

}

else

{

Console.WriteLine("Ошибка: возраст не может быть отрицательным!");

}

}

public int GetAge()

{

return age;

}

public void SetAddress(string newAddress)

{

if (!string.IsNullOrWhiteSpace(newAddress))

{

address = newAddress;

}

else

{

Console.WriteLine("Ошибка: адрес не может быть пустым!");

}

}

public string GetAddress()

{

return address;

}

public void PrintInfo()

{

Console.WriteLine($"Имя: {name}, Возраст: {age}, Адрес: {address}");

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа для управления информацией о людях.");

List<Person> people = new List<Person>();

string choice;

do

{

Console.WriteLine("\nВыберите действие:");

Console.WriteLine("1 - Добавить нового человека");

Console.WriteLine("2 - Показать всех людей");

Console.WriteLine("3 - Выйти");

Console.Write("Ваш выбор (1-3): ");

choice = Console.ReadLine();

if (choice == "1")

{

Person person = new Person();

Console.Write("Введите имя: ");

string inputName = Console.ReadLine();

person.SetName(inputName);

Console.Write("Введите возраст: ");

string inputAge = Console.ReadLine();

int age;

if (int.TryParse(inputAge, out age))

{

person.SetAge(age);

}

else

{

Console.WriteLine("Ошибка: возраст должен быть целым числом!");

continue;

}

Console.Write("Введите адрес: ");

string inputAddress = Console.ReadLine();

person.SetAddress(inputAddress);

people.Add(person);

Console.WriteLine("Человек добавлен!");

}

else if (choice == "2")

{

if (people.Count == 0)

{

Console.WriteLine("Список пуст!");

}

else

{

Console.WriteLine("\nСписок людей:");

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

{

Console.Write($"{i + 1}. ");

people[i].PrintInfo();

}

}

}

else if (choice != "3")

{

Console.WriteLine("Неверный выбор! Введите 1, 2 или 3.");

}

if (choice != "3")

{

Console.Write("Продолжить? (да/нет): ");

choice = Console.ReadLine().ToLower();

}

} while (choice == "да" || choice == "д" || choice == "yes" || choice == "y");

Console.WriteLine("Программа завершена.");

}

}

}

Выполнение программного кода из листинга 2.1.1 представлено на рисунке 2.1.1.

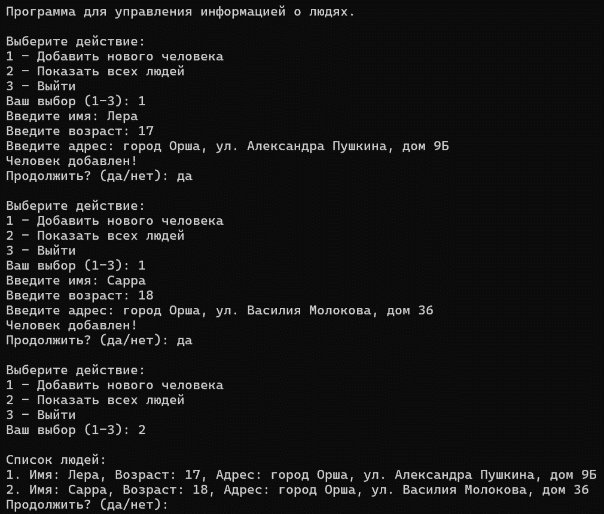


Рисунок 2.1.1 – Результат работы программы для управления информацией о людях

**Задание 2.**

Листинг 2.1.2. Код программы для вычисления площади и периметра геометрических фигур.

using System;

namespace ShapeInheritance

{

class Shape

{

public virtual double Area()

{

return 0;

}

public virtual double Perimeter()

{

return 0;

}

}

class Circle : Shape

{

private double radius;

public Circle(double r)

{

if (r > 0)

{

radius = r;

}

else

{

throw new ArgumentException("Радиус должен быть положительным числом!");

}

}

public override double Area()

{

return Math.PI \* radius \* radius;

}

public override double Perimeter()

{

return 2 \* Math.PI \* radius;

}

}

class Rectangle : Shape

{

private double length;

private double width;

public Rectangle(double l, double w)

{

if (l > 0 && w > 0)

{

length = l;

width = w;

}

else

{

throw new ArgumentException("Длина и ширина должны быть положительными числами!");

}

}

public override double Area()

{

return length \* width;

}

public override double Perimeter()

{

return 2 \* (length + width);

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа для вычисления площади и периметра геометрических фигур.");

double radius, length, width;

try

{

Console.Write("Введите радиус круга: ");

string inputRadius = Console.ReadLine();

if (double.TryParse(inputRadius, out radius) && radius > 0)

{

Circle circle = new Circle(radius);

Console.WriteLine("\nИнформация о круге:");

Console.WriteLine($"Площадь: {circle.Area():F2}");

Console.WriteLine($"Периметр: {circle.Perimeter():F2}");

}

else

{

Console.WriteLine("Ошибка: радиус должен быть положительным числом!");

return;

}

Console.Write("Введите длину прямоугольника: ");

string inputLength = Console.ReadLine();

Console.Write("Введите ширину прямоугольника: ");

string inputWidth = Console.ReadLine();

if (double.TryParse(inputLength, out length) && double.TryParse(inputWidth, out width) && length > 0 && width > 0)

{

Rectangle rectangle = new Rectangle(length, width);

Console.WriteLine("\nИнформация о прямоугольнике:");

Console.WriteLine($"Площадь: {rectangle.Area():F2}");

Console.WriteLine($"Периметр: {rectangle.Perimeter():F2}");

}

else

{

Console.WriteLine("Ошибка: длина и ширина должны быть положительными числами!");

return;

}

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

Console.WriteLine("\nПрограмма завершена. Нажмите Enter для выхода.");

Console.ReadLine();

}

}

}

Выполнение программного кода из листинга 2.1.2 представлено на рисунке 2.1.2.



Рисунок 2.1.2 – Результат вычисления площади и периметра геометрических фигур

**Задание 3.**

Листинг 2.1.3. Код программы для управления информацией о книгах и авторах.

using System;

using System.Collections.Generic;

namespace AuthorBook

{

class Author

{

private string name;

private int birthYear;

public Author(string authorName, int year)

{

if (!string.IsNullOrWhiteSpace(authorName))

{

name = authorName;

}

else

{

throw new ArgumentException("Имя автора не может быть пустым!");

}

if (year > 0)

{

birthYear = year;

}

else

{

throw new ArgumentException("Год рождения должен быть положительным!");

}

}

public string GetName()

{

return name;

}

public int GetBirthYear()

{

return birthYear;

}

public void PrintInfo()

{

Console.WriteLine($"Автор: {name}, Год рождения: {birthYear}");

}

}

class Book

{

private string title;

private int publishYear;

private Author author;

public Book(string bookTitle, int year, Author bookAuthor)

{

if (!string.IsNullOrWhiteSpace(bookTitle))

{

title = bookTitle;

}

else

{

throw new ArgumentException("Название книги не может быть пустым!");

}

if (year > 0)

{

publishYear = year;

}

else

{

throw new ArgumentException("Год выпуска должен быть положительным!");

}

author = bookAuthor;

}

public void PrintInfo()

{

Console.WriteLine($"Книга: {title}, Год выпуска: {publishYear}");

author.PrintInfo();

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа для управления информацией об авторах и книгах.");

List<Author> authors = new List<Author>();

List<Book> books = new List<Book>();

string choice;

do

{

Console.WriteLine("\nВыберите действие:");

Console.WriteLine("1 - Добавить нового автора");

Console.WriteLine("2 - Добавить новую книгу");

Console.WriteLine("3 - Показать всех авторов");

Console.WriteLine("4 - Показать все книги");

Console.WriteLine("5 - Выйти");

Console.Write("Ваш выбор (1-5): ");

choice = Console.ReadLine();

try

{

if (choice == "1")

{

Console.Write("Введите имя автора: ");

string authorName = Console.ReadLine();

Console.Write("Введите год рождения: ");

string inputBirthYear = Console.ReadLine();

int birthYear;

if (int.TryParse(inputBirthYear, out birthYear))

{

Author author = new Author(authorName, birthYear);

authors.Add(author);

Console.WriteLine("Автор добавлен!");

}

else

{

Console.WriteLine("Ошибка: год рождения должен быть целым числом!");

}

}

else if (choice == "2")

{

if (authors.Count == 0)

{

Console.WriteLine("Ошибка: сначала добавьте автора!");

continue;

}

Console.Write("Введите название книги: ");

string bookTitle = Console.ReadLine();

Console.Write("Введите год выпуска: ");

string inputPublishYear = Console.ReadLine();

int publishYear;

if (int.TryParse(inputPublishYear, out publishYear))

{

Console.WriteLine("\nВыберите автора по номеру:");

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

{

Console.WriteLine($"{i + 1}. {authors[i].GetName()} ({authors[i].GetBirthYear()})");

}

Console.Write("Номер автора: ");

string inputAuthorIndex = Console.ReadLine();

int authorIndex;

if (int.TryParse(inputAuthorIndex, out authorIndex) && authorIndex > 0 && authorIndex <= authors.Count)

{

Book book = new Book(bookTitle, publishYear, authors[authorIndex - 1]);

books.Add(book);

Console.WriteLine("Книга добавлена!");

}

else

{

Console.WriteLine("Ошибка: неверный номер автора!");

}

}

else

{

Console.WriteLine("Ошибка: год выпуска должен быть целым числом!");

}

}

else if (choice == "3")

{

if (authors.Count == 0)

{

Console.WriteLine("Список авторов пуст!");

}

else

{

Console.WriteLine("\nСписок авторов:");

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

{

Console.Write($"{i + 1}. ");

authors[i].PrintInfo();

}

}

}

else if (choice == "4")

{

if (books.Count == 0)

{

Console.WriteLine("Список книг пуст!");

}

else

{

Console.WriteLine("\nСписок книг:");

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

{

Console.Write($"{i + 1}. ");

books[i].PrintInfo();

}

}

}

else if (choice != "5")

{

Console.WriteLine("Неверный выбор! Введите 1, 2, 3, 4 или 5.");

}

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

if (choice != "5")

{

Console.Write("Продолжить? (да/нет): ");

choice = Console.ReadLine().ToLower();

}

} while (choice == "да" || choice == "д" || choice == "yes" || choice == "y");

Console.WriteLine("Программа завершена.");

}

}

}

Выполнение программного кода из листинга 2.1.3 представлено на рисунке 2.1.3.

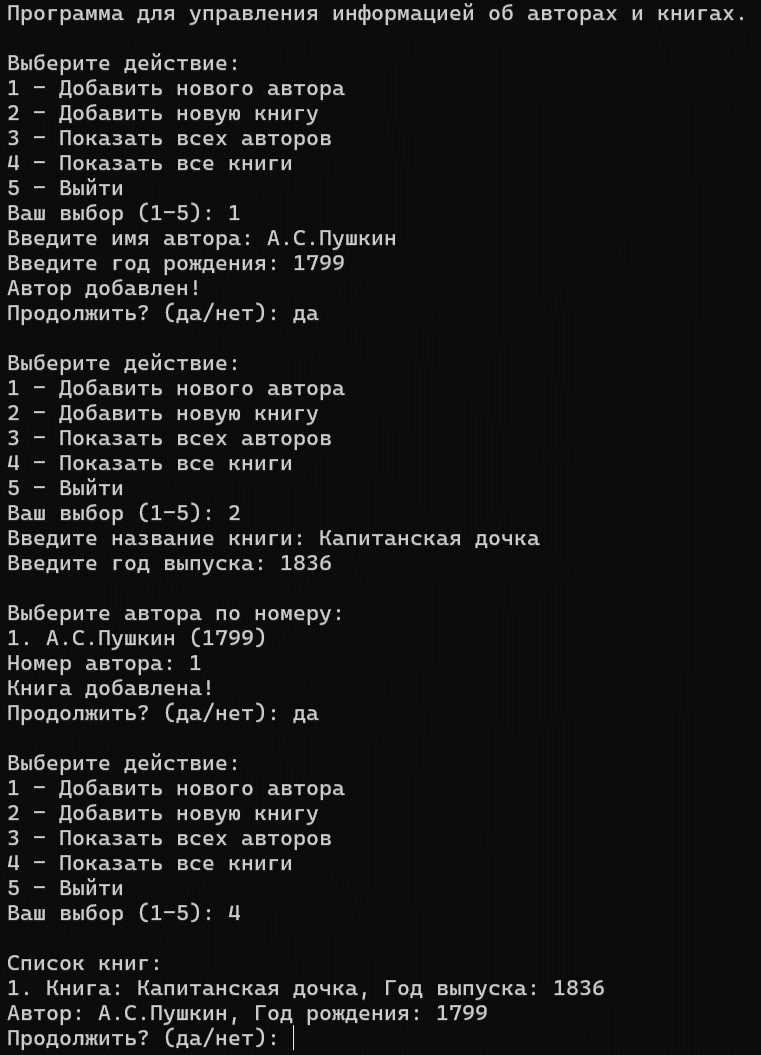


Рисунок 2.1.3 – Результат создания класса для управления информацией о книгах и авторах

**Задание 4.**

Листинг 2.1.4. Код программы для вывода информации о рисуемом объекте.

using System;

namespace DrawbleShapes

{

interface IDrawable

{

void Draw();

}

class Circle : IDrawable

{

public double Radius { get; set; }

public Circle(double radius)

{

Radius = radius;

}

public void Draw()

{

Console.WriteLine($"Рисуем круг с радиусом: {Radius}");

}

}

class Rectangle : IDrawable

{

public double Width { get; set; }

public double Height { get; set; }

public Rectangle(double width, double height)

{

Width = width;

Height = height;

}

public void Draw()

{

Console.WriteLine($"Рисуем прямоугольник с шириной: {Width} и высотой: {Height}");

}

}

class Triangle : IDrawable

{

public double SideA { get; set; }

public double SideB { get; set; }

public double SideC { get; set; }

public Triangle(double sideA, double sideB, double sideC)

{

SideA = sideA;

SideB = sideB;

SideC = sideC;

}

public void Draw()

{

Console.WriteLine($"Рисуем треугольник со сторонами: {SideA}, {SideB}, {SideC}");

}

}

class Program

{

static void Main(string[] args)

{

IDrawable[] shapes = new IDrawable[3];

Console.Write("Введите радиус круга:");

double radius = double.Parse(Console.ReadLine());

shapes[0] = new Circle(radius);

Console.Write("Введите ширину прямоугольника:");

double width = double.Parse(Console.ReadLine());

Console.Write("Введите высоту прямоугольника:");

double height = double.Parse(Console.ReadLine());

shapes[1] = new Rectangle(width, height);

Console.Write("Введите длину первой стороны треугольника:");

double sideA = double.Parse(Console.ReadLine());

Console.Write("Введите длину второй стороны треугольника:");

double sideB = double.Parse(Console.ReadLine());

Console.Write("Введите длину третьей стороны треугольника:");

double sideC = double.Parse(Console.ReadLine());

shapes[2] = new Triangle(sideA, sideB, sideC);

Console.WriteLine("\nРисуем фигуры:");

foreach (IDrawable shape in shapes)

{

shape.Draw();

}

}

}

}

Выполнение программного кода из листинга 2.1.4 представлено на рисунке 2.1.4.

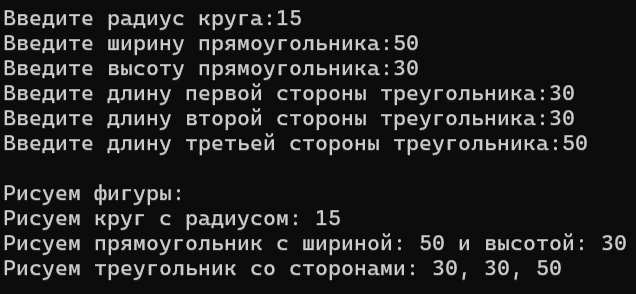


Рисунок 2.1.4 – Результат вывода информации о рисуемом объекте

**Задание 5.**

Листинг 2.1.5. Код программы для управления температурой.

using System;

namespace TemperatureControl

{

class TemperatureSensor

{

private double currentTemperature;

public delegate void TemperatureChangedHandler(double newTemperature);

public event TemperatureChangedHandler TemperatureChanged;

public double CurrentTemperature

{

get { return currentTemperature; }

set

{

if (currentTemperature != value)

{

currentTemperature = value;

OnTemperatureChanged(currentTemperature);

}

}

}

protected virtual void OnTemperatureChanged(double newTemperature)

{

TemperatureChanged?.Invoke(newTemperature);

}

}

class Thermostat

{

private bool isHeatingOn;

public Thermostat()

{

isHeatingOn = false;

}

public void OnTemperatureChanged(double newTemperature)

{

if (newTemperature < 18)

{

if (!isHeatingOn)

{

isHeatingOn = true;

Console.WriteLine($"Температура {newTemperature}°C: Отопление включено.");

}

}

else if (newTemperature > 22)

{

if (isHeatingOn)

{

isHeatingOn = false;

Console.WriteLine($"Температура {newTemperature}°C: Отопление выключено.");

}

}

else

{

if (isHeatingOn)

{

Console.WriteLine($"Температура {newTemperature}°C: Отопление остаётся включённым.");

}

else

{

Console.WriteLine($"Температура {newTemperature}°C: Отопление остаётся выключенным.");

}

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа управления температурой.");

TemperatureSensor sensor = new TemperatureSensor();

Thermostat thermostat = new Thermostat();

sensor.TemperatureChanged += thermostat.OnTemperatureChanged;

try

{

Console.Write("Введите начальную температуру (в °C): ");

double initialTemp = double.Parse(Console.ReadLine());

sensor.CurrentTemperature = initialTemp;

while (true)

{

Console.Write("Введите новую температуру (или 'exit' для завершения): ");

string input = Console.ReadLine();

if (input.ToLower() == "exit")

{

break;

}

double newTemp;

if (double.TryParse(input, out newTemp))

{

sensor.CurrentTemperature = newTemp;

}

else

{

Console.WriteLine("Ошибка: введите корректное числовое значение или 'exit' для завершения!");

}

}

}

catch (FormatException)

{

Console.WriteLine("Ошибка: введите корректное числовое значение!");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

Console.WriteLine("Программа завершена. Нажмите Enter для выхода.");

Console.ReadLine();

}

}

}

Выполнение программного кода из листинга 2.1.5 представлено на рисунке 2.1.5.

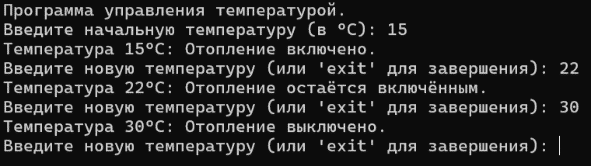


Рисунок 2.1.5 – Результат работы программы для управления температурой

**Задания (вариант №4)**

1. Создание класса «Банковский счет» (BankAccount): Создайте класс «Банковский счет» с полями: номер счета (уникальный идентификатор), владелец, баланс. Реализуйте конструкторы для инициализации объектов класса. Создайте методы для получения и установки значений полей, а также для выполнения операций со счетом (пополнение, снятие средств и т. д.). Создайте объекты класса «Банковский счет» и протестируйте его функциональность.

2. Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (например, по автору или по году издания), добавления книг в библиотеку, удаления книг из нее, сортировки книг по разным полям.

3. Составить описание класса для определения одномерных массивов строк фиксированной длины. Предусмотреть контроль выхода за пределы массива, возможность обращения к отдельным строкам массива по индексам, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, а также вывод на экран элемента массива по заданному индексу и всего массива.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 2.2.1. Код программы для управление банковскими счетами.

using System;

using System.Collections.Generic;

namespace BankAccountManagement

{

class BankAccount

{

private string accountNumber;

private string owner;

private double balance;

public BankAccount(string accountNumber, string owner, double initialBalance)

{

if (!string.IsNullOrWhiteSpace(accountNumber) && !string.IsNullOrWhiteSpace(owner) && initialBalance >= 0)

{

this.accountNumber = accountNumber;

this.owner = owner;

this.balance = initialBalance;

}

else

{

throw new ArgumentException("Номер счета, имя владельца и начальный баланс должны быть корректными (баланс не отрицательный)!");

}

}

public string GetAccountNumber()

{

return accountNumber;

}

public string GetOwner()

{

return owner;

}

public double GetBalance()

{

return balance;

}

public void SetOwner(string newOwner)

{

if (!string.IsNullOrWhiteSpace(newOwner))

{

owner = newOwner;

}

else

{

throw new ArgumentException("Имя владельца не может быть пустым!");

}

}

public void Deposit(double amount)

{

if (amount > 0)

{

balance += amount;

Console.WriteLine($"Пополнение на {amount} успешно. Новый баланс: {balance}");

}

else

{

throw new ArgumentException("Сумма пополнения должна быть положительной!");

}

}

public void Withdraw(double amount)

{

if (amount > 0)

{

if (amount <= balance)

{

balance -= amount;

Console.WriteLine($"Снятие {amount} успешно. Новый баланс: {balance}");

}

else

{

throw new ArgumentException("Недостаточно средств на счете!");

}

}

else

{

throw new ArgumentException("Сумма снятия должна быть положительной!");

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа управления банковскими счетами.");

List<BankAccount> accounts = new List<BankAccount>();

try

{

while (true)

{

Console.Write("\nВведите номер счета (или 'exit' для завершения добавления): ");

string accountNumber = Console.ReadLine();

if (accountNumber.ToLower() == "exit")

{

break;

}

Console.Write("Введите имя владельца: ");

string owner = Console.ReadLine();

Console.Write("Введите начальный баланс: ");

double initialBalance = double.Parse(Console.ReadLine());

BankAccount account = new BankAccount(accountNumber, owner, initialBalance);

accounts.Add(account);

Console.WriteLine($"Счет создан. Номер: {account.GetAccountNumber()}, Владелец: {account.GetOwner()}, Баланс: {account.GetBalance()}");

}

if (accounts.Count == 0)

{

Console.WriteLine("Нет созданных счетов.");

}

else

{

while (true)

{

Console.WriteLine("\nСписок счетов:");

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

{

Console.WriteLine($"{i + 1}. Номер: {accounts[i].GetAccountNumber()}, Владелец: {accounts[i].GetOwner()}, Баланс: {accounts[i].GetBalance()}");

}

Console.Write("Выберите номер счета для операции (1-" + accounts.Count + ") или 'exit' для завершения: ");

string input = Console.ReadLine();

if (input.ToLower() == "exit")

{

break;

}

int accountIndex;

if (int.TryParse(input, out accountIndex) && accountIndex > 0 && accountIndex <= accounts.Count)

{

BankAccount selectedAccount = accounts[accountIndex - 1];

Console.Write("Введите сумму для пополнения: ");

double depositAmount = double.Parse(Console.ReadLine());

try

{

selectedAccount.Deposit(depositAmount);

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

Console.Write("Введите сумму для снятия (или 0 для пропуска): ");

double withdrawAmount = double.Parse(Console.ReadLine());

if (withdrawAmount > 0)

{

try

{

selectedAccount.Withdraw(withdrawAmount);

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

}

else

{

Console.WriteLine("Ошибка: введите корректный номер счета или 'exit' для завершения!");

}

}

}

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка при создании счета: {ex.Message}");

}

catch (FormatException)

{

Console.WriteLine("Ошибка: введите корректное числовое значение!");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

Console.WriteLine("Программа завершена. Нажмите Enter для выхода.");

Console.ReadLine();

}

}

}

Выполнение программного кода из листинга 2.2.1 представлено на рисунке 2.2.1.

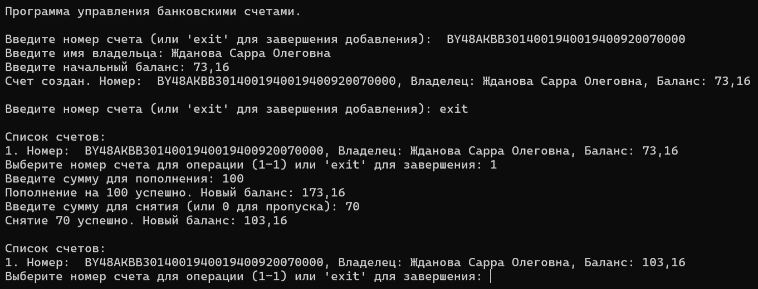


Рисунок 2.2.1 – Результат работы программы для управления банковскими счетами

**Задание 2.**

Листинг 2.2.2. Код программы для управления домашней библиотекой.

using System;

using System.Collections.Generic;

using System.Linq;

namespace HomeLibrarySystem

{

class Book

{

public string Title { get; set; }

public string Author { get; set; }

public int Year { get; set; }

public Book(string title, string author, int year)

{

Title = title;

Author = author;

Year = year;

}

public override string ToString()

{

return $"Название: {Title}, Автор: {Author}, Год: {Year}";

}

}

class HomeLibrary

{

private List<Book> books;

public HomeLibrary()

{

books = new List<Book>();

}

public void AddBook(Book book)

{

books.Add(book);

Console.WriteLine("Книга добавлена.");

}

public void RemoveBook(int index)

{

if (index >= 0 && index < books.Count)

{

books.RemoveAt(index);

Console.WriteLine("Книга удалена.");

}

else

{

Console.WriteLine("Ошибка: неверный индекс.");

}

}

public List<Book> SearchByAuthor(string author)

{

return books.Where(b => b.Author.Equals(author, StringComparison.OrdinalIgnoreCase)).ToList();

}

public List<Book> SearchByYear(int year)

{

return books.Where(b => b.Year == year).ToList();

}

public void SortBy(string field)

{

if (field == "title")

{

books = books.OrderBy(b => b.Title).ToList();

}

else if (field == "author")

{

books = books.OrderBy(b => b.Author).ToList();

}

else if (field == "year")

{

books = books.OrderBy(b => b.Year).ToList();

}

else

{

Console.WriteLine("Ошибка: неверное поле для сортировки (title, author, year).");

return;

}

Console.WriteLine("Библиотека отсортирована.");

}

public void PrintAllBooks()

{

if (books.Count == 0)

{

Console.WriteLine("Библиотека пуста.");

}

else

{

Console.WriteLine("\nСписок книг:");

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

{

Console.WriteLine($"{i}: {books[i]}");

}

}

}

public int GetBookCount()

{

return books.Count;

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа управления домашней библиотекой.");

HomeLibrary library = new HomeLibrary();

while (true)

{

Console.WriteLine("\nВыберите действие:");

Console.WriteLine("1 - Добавить книгу");

Console.WriteLine("2 - Удалить книгу");

Console.WriteLine("3 - Поиск по автору");

Console.WriteLine("4 - Поиск по году");

Console.WriteLine("5 - Сортировать библиотеку");

Console.WriteLine("6 - Просмотреть все книги");

Console.WriteLine("7 - Выход");

Console.Write("Ваш выбор (1-7): ");

string choice = Console.ReadLine();

try

{

switch (choice)

{

case "1":

while (true)

{

Console.Write("Введите название книги (или 'exit' для возврата в меню): ");

string title = Console.ReadLine();

if (title.ToLower() == "exit")

{

break;

}

Console.Write("Введите автора книги: ");

string author = Console.ReadLine();

Console.Write("Введите год издания: ");

string inputYear = Console.ReadLine();

if (int.TryParse(inputYear, out int year) && year > 0)

{

Book book = new Book(title, author, year);

library.AddBook(book);

}

else

{

Console.WriteLine("Ошибка: год должен быть положительным целым числом!");

}

}

break;

case "2":

library.PrintAllBooks();

Console.Write("Введите индекс книги для удаления: ");

string inputIndex = Console.ReadLine();

if (int.TryParse(inputIndex, out int index))

{

library.RemoveBook(index);

library.PrintAllBooks();

}

else

{

Console.WriteLine("Ошибка: введите корректный индекс!");

}

break;

case "3":

Console.Write("Введите автора для поиска: ");

string searchAuthor = Console.ReadLine();

List<Book> foundByAuthor = library.SearchByAuthor(searchAuthor);

Console.WriteLine("\nКниги автора:");

if (foundByAuthor.Count == 0)

{

Console.WriteLine("Книги не найдены.");

}

else

{

foreach (Book b in foundByAuthor)

{

Console.WriteLine(b);

}

}

break;

case "4":

Console.Write("Введите год для поиска: ");

string inputYearSearch = Console.ReadLine();

if (int.TryParse(inputYearSearch, out int searchYear) && searchYear > 0)

{

List<Book> foundByYear = library.SearchByYear(searchYear);

Console.WriteLine("\nКниги за год:");

if (foundByYear.Count == 0)

{

Console.WriteLine("Книги не найдены.");

}

else

{

foreach (Book b in foundByYear)

{

Console.WriteLine(b);

}

}

}

else

{

Console.WriteLine("Ошибка: введите корректный год!");

}

break;

case "5":

Console.Write("Введите поле для сортировки (title, author, year): ");

string sortField = Console.ReadLine();

library.SortBy(sortField.ToLower());

library.PrintAllBooks();

break;

case "6":

library.PrintAllBooks();

break;

case "7":

Console.WriteLine("Программа завершена.");

return;

default:

Console.WriteLine("Ошибка: введите число от 1 до 7!");

break;

}

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

}

}

}

}

Выполнение программного кода из листинга 2.2.2 представлено на рисунке 2.2.2.

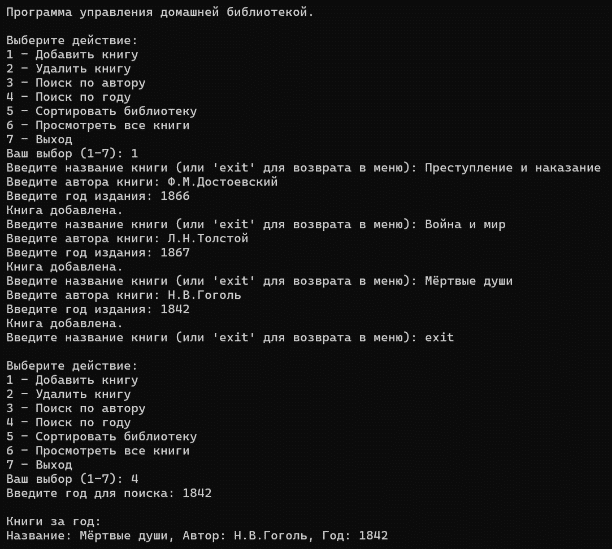


Рисунок 2.2.2 – Результат работы программы «Домашняя библиотека»

**Задание 3.**

Листинг 2.2.3. Код программы для работы с массивами из строк.

using System;

using System.Collections.Generic;

using System.Linq;

namespace ArrayManagement

{

class StringArray

{

private string[] array;

public int Length { get { return array.Length; } }

public StringArray(int length)

{

if (length > 0)

{

array = new string[length];

}

else

{

throw new ArgumentException("Длина массива должна быть положительной!");

}

}

public void SetElement(int index, string value)

{

if (index >= 0 && index < array.Length)

{

array[index] = value;

}

else

{

throw new IndexOutOfRangeException("Индекс выходит за пределы массива!");

}

}

public string GetElement(int index)

{

if (index >= 0 && index < array.Length)

{

return array[index];

}

else

{

throw new IndexOutOfRangeException("Индекс выходит за пределы массива!");

}

}

public static StringArray Concatenate(StringArray arr1, StringArray arr2)

{

if (arr1 == null || arr2 == null)

{

throw new ArgumentNullException("Один из массивов пустой!");

}

StringArray result = new StringArray(arr1.Length + arr2.Length);

for (int i = 0; i < arr1.Length; i++)

{

result.SetElement(i, arr1.GetElement(i));

}

for (int i = 0; i < arr2.Length; i++)

{

result.SetElement(arr1.Length + i, arr2.GetElement(i));

}

return result;

}

public static StringArray Merge(StringArray arr1, StringArray arr2)

{

if (arr1 == null || arr2 == null)

{

throw new ArgumentNullException("Один из массивов пустой!");

}

var uniqueElements = new HashSet<string>(arr1.array.Concat(arr2.array).Where(s => s != null));

StringArray result = new StringArray(uniqueElements.Count);

int i = 0;

foreach (var item in uniqueElements)

{

result.SetElement(i++, item);

}

return result;

}

public void PrintArray()

{

if (array.Length == 0)

{

Console.WriteLine("Массив пуст.");

}

else

{

Console.WriteLine("\nМассив:");

for (int i = 0; i < array.Length; i++)

{

Console.WriteLine($"[{i}]: {array[i] ?? "null"}");

}

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа управления одномерным массивом строк.");

Console.Write("Введите длину первого массива: ");

string inputLength1 = Console.ReadLine();

StringArray array1 = null;

if (int.TryParse(inputLength1, out int length1) && length1 > 0)

{

array1 = new StringArray(length1);

for (int i = 0; i < length1; i++)

{

Console.Write($"Введите строку для индекса {i}: ");

array1.SetElement(i, Console.ReadLine());

}

}

else

{

Console.WriteLine("Ошибка: введите корректную длину!");

return;

}

Console.Write("Введите длину второго массива: ");

string inputLength2 = Console.ReadLine();

StringArray array2 = null;

if (int.TryParse(inputLength2, out int length2) && length2 > 0)

{

array2 = new StringArray(length2);

for (int i = 0; i < length2; i++)

{

Console.Write($"Введите строку для индекса {i}: ");

array2.SetElement(i, Console.ReadLine());

}

}

else

{

Console.WriteLine("Ошибка: введите корректную длину!");

return;

}

while (true)

{

Console.WriteLine("\nВыберите действие:");

Console.WriteLine("1 - Вывести элемент по индексу");

Console.WriteLine("2 - Вывести первый массив");

Console.WriteLine("3 - Вывести второй массив");

Console.WriteLine("4 - Сцепить массивы");

Console.WriteLine("5 - Слить массивы (без повторов)");

Console.WriteLine("6 - Выход");

Console.Write("Ваш выбор (1-6): ");

string choice = Console.ReadLine();

try

{

switch (choice)

{

case "1":

Console.Write("Введите индекс для первого массива: ");

string inputIndex = Console.ReadLine();

if (int.TryParse(inputIndex, out int index) && index >= 0 && index < array1.Length)

{

Console.WriteLine($"Элемент: {array1.GetElement(index)}");

}

else

{

Console.WriteLine("Ошибка: неверный индекс!");

}

break;

case "2":

array1.PrintArray();

break;

case "3":

array2.PrintArray();

break;

case "4":

StringArray concatenated = StringArray.Concatenate(array1, array2);

concatenated.PrintArray();

break;

case "5":

StringArray merged = StringArray.Merge(array1, array2);

merged.PrintArray();

break;

case "6":

Console.WriteLine("Программа завершена.");

return;

default:

Console.WriteLine("Ошибка: введите число от 1 до 6!");

break;

}

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

}

}

}

}

Выполнение программного кода из листинга 2.2.3 представлено на рисунке 2.2.3.

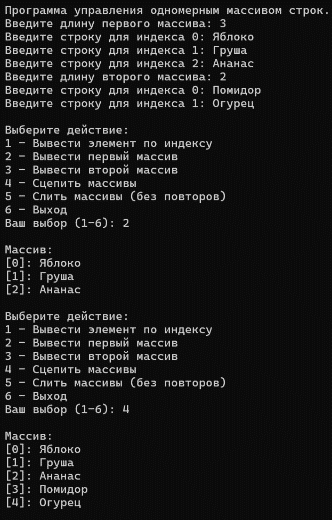


Рисунок 2.2.3 – Результат программы для работы с массивами из строк

**Модуль 3. Делегаты**

**Задания**

1. Создайте базовый класс «Фигура» с методом для вычисления площади. Затем создайте производные классы для разных геометрических фигур (круг, прямоугольник, треугольник) и используйте делегат для динамического вызова метода вычисления площади.

2. Реализуйте систему событий для мобильного приложения. Создайте класс «Уведомление» с событиями для отправки уведомлений (сообщения, звонки, электронные письма). Зарегистрируйте обработчики событий для разных типов уведомлений.

3. Создайте приложение для управления задачами с использованием делегатов. Пользователь должен иметь возможность добавлять задачи и выбирать делегата для выполнения каждой задачи (например, отправка уведомления или запись в журнал).

4. Разработайте систему фильтрации данных с использованием делегатов. Пользователь должен иметь возможность выбрать фильтр для списка данных (например, фильтр по дате или по ключевым словам).

5. Создайте приложение для сортировки числовых данных. Пользователь должен иметь возможность выбрать метод сортировки (например, сортировка пузырьком или быстрая сортировка) с помощью делегатов.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 3.1.1. Код программы приложения для вычисления площади геометрических фигур.

using System;

using System.Windows;

namespace ShapeWPF

{

public delegate double AreaCalculator();

public partial class MainWindow : Window

{

private Shape currentShape;

private AreaCalculator calculator;

public MainWindow()

{

InitializeComponent();

}

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

{

try

{

if (double.TryParse(InputTextBox.Text, out double radius) && radius >= 0)

{

currentShape = new Circle(radius);

calculator = currentShape.CalculateArea;

currentShape.AreaChanged += OnAreaChanged;

ResultTextBlock.Text = "Круг создан. Выберите 'Вычислить площадь' или измените размер.";

}

else

{

ResultTextBlock.Text = "Ошибка: введите корректный неотрицательный радиус!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

string[] values = InputTextBox.Text.Split(' ');

if (values.Length == 2 && double.TryParse(values[0], out double width) && double.TryParse(values[1], out double height) && width >= 0 && height >= 0)

{

currentShape = new Rectangle(width, height);

calculator = currentShape.CalculateArea;

currentShape.AreaChanged += OnAreaChanged;

ResultTextBlock.Text = "Прямоугольник создан. Выберите 'Вычислить площадь' или измените размер.";

}

else

{

ResultTextBlock.Text = "Ошибка: введите ширину и высоту через пробел (например, 3 4)!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

string[] values = InputTextBox.Text.Split(' ');

if (values.Length == 3 && double.TryParse(values[0], out double sideA) && double.TryParse(values[1], out double sideB) && double.TryParse(values[2], out double sideC) && sideA > 0 && sideB > 0 && sideC > 0)

{

currentShape = new Triangle(sideA, sideB, sideC);

calculator = currentShape.CalculateArea;

currentShape.AreaChanged += OnAreaChanged;

ResultTextBlock.Text = "Треугольник создан. Выберите 'Вычислить площадь'.";

}

else

{

ResultTextBlock.Text = "Ошибка: введите три стороны через пробел (например, 3 4 5)!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

if (calculator != null)

{

double area = calculator();

ResultTextBlock.Text = $"Площадь: {area:F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: сначала создайте фигуру!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

private void OnAreaChanged(double newArea)

{

if (Dispatcher.CheckAccess())

{

ResultTextBlock.Text = $"Площадь изменена на: {newArea:F2}";

}

else

{

Dispatcher.Invoke(() => ResultTextBlock.Text = $"Площадь изменена на: {newArea:F2}");

}

}

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

{

try

{

if (currentShape == null)

{

ResultTextBlock.Text = "Ошибка: сначала создайте фигуру!";

return;

}

string[] values = InputTextBox.Text.Split(' ');

if (currentShape is Circle circle)

{

if (values.Length == 1 && double.TryParse(values[0], out double newRadius) && newRadius >= 0)

{

circle.ChangeRadius(newRadius);

ResultTextBlock.Text = $"Радиус изменён. Площадь: {circle.CalculateArea():F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: введите новый радиус!";

}

}

else if (currentShape is Rectangle rectangle)

{

if (values.Length == 2 && double.TryParse(values[0], out double newWidth) && double.TryParse(values[1], out double newHeight) && newWidth >= 0 && newHeight >= 0)

{

rectangle.ChangeWidth(newWidth);

rectangle.ChangeHeight(newHeight);

ResultTextBlock.Text = $"Размеры изменены. Площадь: {rectangle.CalculateArea():F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: введите новую ширину и высоту через пробел!";

}

}

else

{

ResultTextBlock.Text = "Изменение размера недоступно для треугольника!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

}

}

Выполнение программного кода из листинга 3.1.1 представлено на рисунке 3.1.1.

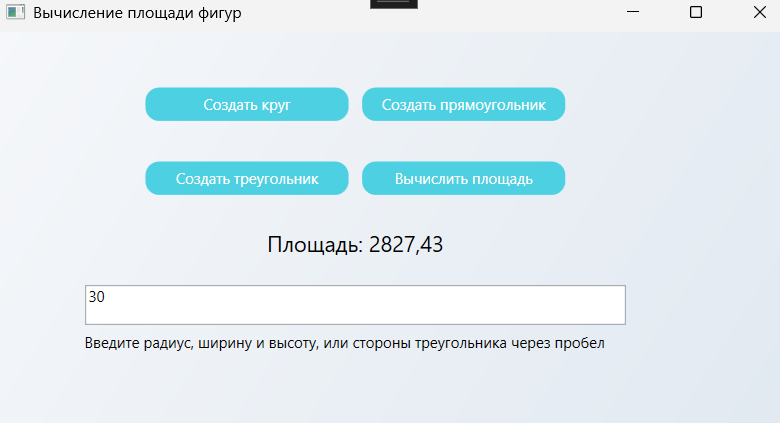


Рисунок 3.1.1 – Результат работы приложения «Вычисление площади фигур»

**Задание 2.**

Листинг 3.1.2. Код программы приложения для работы с уведомлениями.

using System;

using System.Windows;

using System.Windows.Threading;

namespace ShapeWPF

{

public class Notification

{

public delegate void MessageSentHandler(string message);

public delegate void CallMadeHandler(string phoneNumber);

public delegate void EmailSentHandler(string email, string subject);

public event MessageSentHandler MessageSent;

public event CallMadeHandler CallMade;

public event EmailSentHandler EmailSent;

public void SendMessage(string message)

{

if (string.IsNullOrWhiteSpace(message))

throw new ArgumentException("Сообщение не может быть пустым!");

MessageSent?.Invoke(message);

}

public void MakeCall(string phoneNumber)

{

if (string.IsNullOrWhiteSpace(phoneNumber))

throw new ArgumentException("Номер телефона не может быть пустым!");

CallMade?.Invoke(phoneNumber);

}

public void SendEmail(string email, string subject)

{

if (string.IsNullOrWhiteSpace(email) || string.IsNullOrWhiteSpace(subject))

throw new ArgumentException("Email и тема не могут быть пустыми!");

EmailSent?.Invoke(email, subject);

}

}

public partial class MainWindow : Window

{

private readonly Notification notifier;

private readonly ClassNotification.Class1 logger;

private readonly DispatcherTimer timer;

public MainWindow()

{

try

{

InitializeComponent();

notifier = new Notification();

try

{

logger = new ClassNotification.Class1();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при создании логгера: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

logger = null;

}

notifier.MessageSent += OnMessageSent;

notifier.CallMade += OnCallMade;

notifier.EmailSent += OnEmailSent;

if (ResultTextBlock == null)

{

MessageBox.Show("Ошибка: Элемент ResultTextBlock не найден в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

ResultTextBlock.Text = "Результаты появятся здесь";

timer = new DispatcherTimer

{

Interval = TimeSpan.FromSeconds(10)

};

timer.Tick += Timer\_Tick;

timer.Start();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при инициализации приложения: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void Timer\_Tick(object sender, EventArgs e)

{

try

{

notifier.SendMessage("Автоматическое уведомление!");

if (logger != null)

logger.LogNotification("Автоматическое уведомление отправлено");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

}

catch (Exception ex)

{

UpdateResultText($"Ошибка в таймере: {ex.Message}");

}

}

private void OnMessageSent(string message)

{

if (Dispatcher.CheckAccess())

{

UpdateResultText($"Сообщение отправлено: {message}");

if (logger != null)

logger.LogNotification($"Сообщение отправлено: {message}");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

}

else

{

Dispatcher.Invoke(() =>

{

UpdateResultText($"Сообщение отправлено: {message}");

if (logger != null)

logger.LogNotification($"Сообщение отправлено: {message}");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

});

}

}

private void OnCallMade(string phoneNumber)

{

if (Dispatcher.CheckAccess())

{

UpdateResultText($"Звонок выполнен на: {phoneNumber}");

if (logger != null)

logger.LogNotification($"Звонок выполнен на: {phoneNumber}");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

}

else

{

Dispatcher.Invoke(() =>

{

UpdateResultText($"Звонок выполнен на: {phoneNumber}");

if (logger != null)

logger.LogNotification($"Звонок выполнен на: {phoneNumber}");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

});

}

}

private void OnEmailSent(string email, string subject)

{

if (Dispatcher.CheckAccess())

{

UpdateResultText($"Email отправлен на {email} с темой: {subject}");

if (logger != null)

logger.LogNotification($"Email отправлен на {email} с темой: {subject}");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

}

else

{

Dispatcher.Invoke(() =>

{

UpdateResultText($"Email отправлен на {email} с темой: {subject}");

if (logger != null)

logger.LogNotification($"Email отправлен на {email} с темой: {subject}");

else

UpdateResultText("Ошибка: Логгер не инициализирован!");

});

}

}

private void UpdateResultText(string text)

{

if (ResultTextBlock != null)

{

ResultTextBlock.Text += $"\n{text}";

}

else

{

MessageBox.Show("Ошибка: Элемент ResultTextBlock не найден в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

if (MessageTextBox != null)

{

string message = MessageTextBox.Text?.Trim();

if (!string.IsNullOrEmpty(message))

{

notifier.SendMessage(message);

MessageTextBox.Text = "";

}

else

{

UpdateResultText("Ошибка: Введите сообщение для отправки!");

}

}

else

{

MessageBox.Show("Ошибка: Элемент MessageTextBox не найден в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Exception ex)

{

UpdateResultText($"Ошибка: {ex.Message}");

}

}

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

{

try

{

if (PhoneTextBox != null)

{

string phone = PhoneTextBox.Text?.Trim();

if (!string.IsNullOrEmpty(phone))

{

notifier.MakeCall(phone);

PhoneTextBox.Text = "";

}

else

{

UpdateResultText("Ошибка: Введите номер телефона для звонка!");

}

}

else

{

MessageBox.Show("Ошибка: Элемент PhoneTextBox не найден в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Exception ex)

{

UpdateResultText($"Ошибка: {ex.Message}");

}

}

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

{

try

{

if (EmailTextBox != null && SubjectTextBox != null)

{

string email = EmailTextBox.Text?.Trim();

string subject = SubjectTextBox.Text?.Trim();

if (!string.IsNullOrEmpty(email) && !string.IsNullOrEmpty(subject))

{

notifier.SendEmail(email, subject);

EmailTextBox.Text = "";

SubjectTextBox.Text = "";

}

else

{

UpdateResultText("Ошибка: Введите email и тему для отправки!");

}

}

else

{

MessageBox.Show("Ошибка: Элементы EmailTextBox или SubjectTextBox не найдены в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Exception ex)

{

UpdateResultText($"Ошибка: {ex.Message}");

}

}

}

}

Выполнение программного кода из листинга 3.1.2 представлено на рисунке 3.1.2.

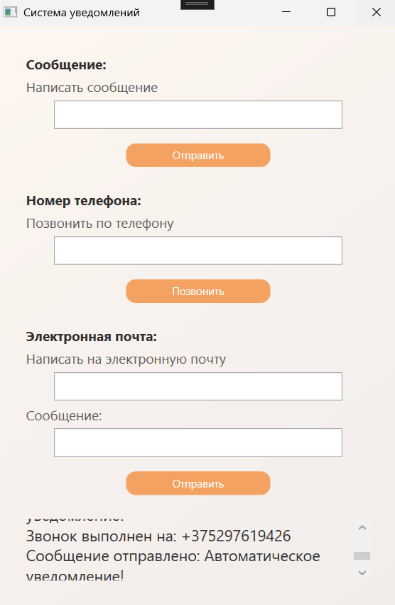


Рисунок 3.1.2 – Результат работы приложения «Система уведомлений»

**Задание 3.**

Листинг 3.1.3. Код программы приложения для управления задачами.

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using ClassNotification;

namespace ShapeWPF

{

public delegate void TaskAction(string taskName, string category);

public class TaskItem

{

public string Name { get; set; }

public string Category { get; set; }

public TaskAction Action { get; set; }

public string ActionName { get; set; }

public override string ToString() => $"Задача: {Name}, Категория: {Category}, Действие: {ActionName}";

}

public partial class MainWindow : Window

{

private readonly List<TaskItem> tasks = new List<TaskItem>();

private readonly Class1 logger;

public MainWindow()

{

try

{

InitializeComponent();

if (TaskNameTextBox == null || CategoryComboBox == null || DelegateComboBox == null || TaskListBox == null || AddTaskButton == null || ExecuteTasksButton == null || ClearResultsButton == null || ResultTextBlock == null)

{

MessageBox.Show("Ошибка: Один или несколько UI-элементов не найдены в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

try

{

logger = new Class1();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при создании логгера: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

logger = null;

}

CategoryComboBox.SelectedIndex = 0;

DelegateComboBox.SelectedIndex = 0;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при инициализации приложения: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void ShowNotification(string taskName, string category)

{

Dispatcher.Invoke(() =>

{

UpdateResultText($"Задача '{taskName}' (категория: {category}) выполнена (уведомление)");

MessageBox.Show($"Задача '{taskName}' (категория: {category}) выполнена!", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

});

}

private void LogToFile(string taskName, string category)

{

Dispatcher.Invoke(() =>

{

UpdateResultText($"Задача '{taskName}' (категория: {category}) выполнена (записано в журнал)");

});

if (logger != null)

{

logger.LogNotification($"Задача '{taskName}' (категория: {category}) выполнена");

}

else

{

Dispatcher.Invoke(() =>

{

UpdateResultText("Ошибка: Логгер не инициализирован!");

});

}

}

private void UpdateResultText(string text)

{

if (ResultTextBlock != null)

{

ResultTextBlock.Text += $"\n{text}";

}

else

{

MessageBox.Show("Ошибка: Элемент ResultTextBlock не найден в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

string taskName = TaskNameTextBox.Text?.Trim();

if (string.IsNullOrEmpty(taskName))

{

MessageBox.Show("Ошибка: Введите название задачи!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

var selectedCategory = CategoryComboBox.SelectedItem as ComboBoxItem;

if (selectedCategory == null)

{

MessageBox.Show("Ошибка: Выберите категорию задачи!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

string category = selectedCategory.Content.ToString();

var selectedAction = DelegateComboBox.SelectedItem as ComboBoxItem;

if (selectedAction == null)

{

MessageBox.Show("Ошибка: Выберите действие для задачи!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

string actionName = selectedAction.Content.ToString();

TaskAction action;

if (actionName == "Отправить уведомление")

{

action = ShowNotification;

}

else if (actionName == "Записать в журнал")

{

action = LogToFile;

}

else

{

MessageBox.Show($"Ошибка: Неизвестное действие '{actionName}'!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

var task = new TaskItem

{

Name = taskName,

Category = category,

Action = action,

ActionName = actionName

};

tasks.Add(task);

TaskListBox.Items.Add(task);

TaskNameTextBox.Text = "";

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при добавлении задачи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

if (tasks.Count == 0)

{

MessageBox.Show("Список задач пуст!", "Информация", MessageBoxButton.OK, MessageBoxImage.Information);

return;

}

foreach (var task in tasks)

{

task.Action?.Invoke(task.Name, task.Category);

}

tasks.Clear();

TaskListBox.Items.Clear();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при выполнении задач: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

if (ResultTextBlock != null)

{

ResultTextBlock.Text = "Результаты выполнения задач появятся здесь";

}

else

{

MessageBox.Show("Ошибка: Элемент ResultTextBlock не найден в XAML!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при очистке результатов: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

if (TaskListBox.SelectedItem != null)

{

var selectedTask = TaskListBox.SelectedItem as TaskItem;

tasks.Remove(selectedTask);

TaskListBox.Items.Remove(selectedTask);

}

else

{

MessageBox.Show("Выберите задачу для удаления!", "Информация", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении задачи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

Выполнение программного кода из листинга 3.1.3 представлено на рисунке 3.1.3.

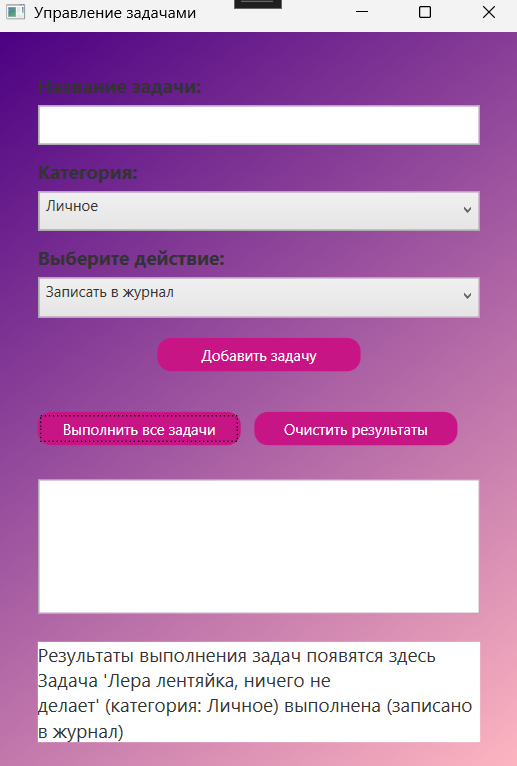


Рисунок 3.1.3 – Результат работы приложения «Управление задачами»

**Задание 4.**

Листинг 3.1.4. Код программы приложения для фильтрации данных.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace Задание\_4

{

public partial class MainWindow : Window

{

private List<DataItem> \_dataItems;

private List<DataItem> \_filteredDataItems;

public MainWindow()

{

InitializeComponent();

InitializeData();

}

private void InitializeData()

{

\_dataItems = new List<DataItem>

{

new DataItem("Событие A", new DateTime(2024, 10, 1)),

new DataItem("Событие B", new DateTime(2024, 10, 5)),

new DataItem("Событие C", new DateTime(2024, 10, 8)),

new DataItem("Событие D", new DateTime(2024, 9, 25)),

new DataItem("Конференция", new DateTime(2024, 11, 12))

};

\_filteredDataItems = new List<DataItem>(\_dataItems);

UpdateDataList();

}

private void UpdateDataList()

{

DataListBox.Items.Clear();

foreach (var item in \_dataItems)

{

DataListBox.Items.Add(item);

}

FilteredDataListBox.Items.Clear();

foreach (var item in \_filteredDataItems)

{

FilteredDataListBox.Items.Add(item);

}

}

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

{

if (FilterSelector.SelectedItem is ComboBoxItem selectedFilter && !string.IsNullOrWhiteSpace(FilterValueTextBox.Text))

{

Func<DataItem, bool> filter = null;

switch (selectedFilter.Content.ToString())

{

case "Фильтр по дате (после)":

if (DateTime.TryParse(FilterValueTextBox.Text, out DateTime filterDate))

{

filter = item => item.Date > filterDate;

}

else

{

MessageBox.Show("Введите корректную дату.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

break;

case "Фильтр по ключевому слову":

string keyword = FilterValueTextBox.Text.ToLower();

filter = item => item.Title.ToLower().Contains(keyword);

break;

}

if (filter != null)

{

\_filteredDataItems = \_dataItems.Where(filter).ToList();

UpdateDataList();

}

}

else

{

MessageBox.Show("Выберите фильтр и введите значение.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

}

}

Выполнение программного кода из листинга 3.1.4 представлено на рисунке 3.1.4.

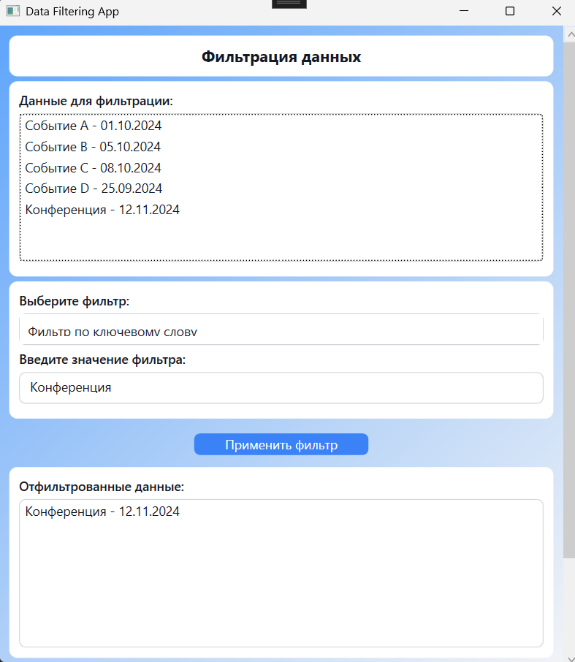


Рисунок 3.1.4 – Результат работы приложения «Data Filtering App»

**Задание 5.**

Листинг 3.1.5. Код программы приложения для сортировки числовых данных.

using System;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace SortingApp

{

public partial class MainWindow : Window

{

private readonly SortingLogic \_sortingLogic = new SortingLogic();

public MainWindow()

{

InitializeComponent();

if (NumbersTextBox == null || RandomCountTextBox == null || SortMethodSelector == null ||

SortedNumbersListBox == null || GenerateRandomButton == null || SortButton == null || ClearButton == null)

{

MessageBox.Show("Ошибка инициализации интерфейса.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

Close();

return;

}

SortMethodSelector.SelectedIndex = 0;

}

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

{

try

{

if (RandomCountTextBox == null || NumbersTextBox == null)

{

MessageBox.Show("Поле для количества или ввода чисел не найдено.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

string input = RandomCountTextBox.Text.Trim();

if (string.IsNullOrWhiteSpace(input))

{

MessageBox.Show("Введите количество чисел!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

if (!int.TryParse(input, out int count) || count < 1)

{

MessageBox.Show("Введите целое число больше 0!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

int[] randomNumbers = \_sortingLogic.GenerateRandomNumbers(count);

NumbersTextBox.Text = string.Join(" ", randomNumbers);

}

catch (ArgumentException ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

catch (Exception ex)

{

MessageBox.Show($"Произошла ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

if (NumbersTextBox == null || SortMethodSelector == null || SortedNumbersListBox == null)

{

MessageBox.Show("Ошибка инициализации интерфейса.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

string input = NumbersTextBox.Text.Trim();

if (string.IsNullOrWhiteSpace(input))

{

MessageBox.Show("Введите числа!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

int[] numbers = input.Split(new[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries)

.Select(x => int.Parse(x.Trim()))

.ToArray();

if (numbers.Length == 0)

{

MessageBox.Show("Введите хотя бы одно число!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

int[] sortedNumbers;

switch (SortMethodSelector.SelectedIndex)

{

case 0:

sortedNumbers = \_sortingLogic.BubbleSort(numbers);

break;

case 1:

sortedNumbers = \_sortingLogic.QuickSort(numbers);

break;

default:

sortedNumbers = \_sortingLogic.BubbleSort(numbers);

break;

}

SortedNumbersListBox.Items.Clear();

foreach (int number in sortedNumbers)

{

SortedNumbersListBox.Items.Add(number.ToString());

}

}

catch (FormatException)

{

MessageBox.Show("Пожалуйста, введите только целые числа, разделённые пробелами или запятыми.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

catch (Exception ex)

{

MessageBox.Show($"Произошла ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

if (NumbersTextBox != null) NumbersTextBox.Text = string.Empty;

if (RandomCountTextBox != null) RandomCountTextBox.Text = string.Empty;

if (SortedNumbersListBox != null) SortedNumbersListBox.Items.Clear();

}

catch (Exception ex)

{

MessageBox.Show($"Произошла ошибка при очистке: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

Выполнение программного кода из листинга 3.1.5 представлено на рисунке 3.1.5.

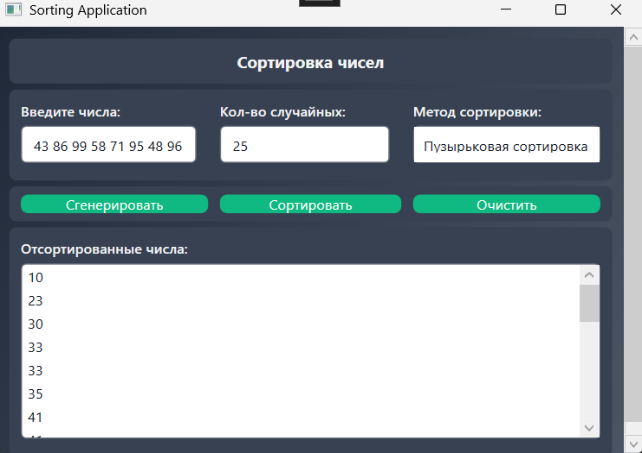


Рисунок 3.1.5 – Результат работы приложения «Sorting Application»

**Модуль 4. Задания по интерфейсам**

**Задания**

1. Создайте интерфейс «Фигура» с методами для вычисления площади и периметра. Затем реализуйте этот интерфейс в классах геометрических фигур (круг, прямоугольник, треугольник).

2. Разработайте приложение для учета продуктов в магазине с использованием интерфейсов. Создайте интерфейс «Товар» с методами для определения стоимости и остатка товара на складе. Реализуйте этот интерфейс в классах различных товаров.

3. Создайте систему учета студентов в университете с помощью интерфейсов. Создайте интерфейс «Студент» с методами для определения среднего балла и получения информации о курсе. Реализуйте этот интерфейс в классах студентов разных курсов.

4. Реализуйте приложение для работы с библиотекой книг с использованием интерфейсов. Создайте интерфейс «Книга» с методами для проверки доступности и выдачи книги. Реализуйте этот интерфейс в классах различных книг.

5. Создайте приложение для рисования на холсте с использованием интерфейсов. Создайте интерфейс «Рисунок» с методами для рисования линий, кругов и прямоугольников. Реализуйте этот интерфейс в классе для работы с холстом.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 4.1.1. Код программы приложения для вычисления площади и периметра геометрических фигур.

using System;

using System.Windows;

namespace ShapeWPF

{

public delegate double AreaCalculator();

public partial class MainWindow : Window

{

private IFigure currentShape;

private AreaCalculator calculator;

public MainWindow()

{

InitializeComponent();

}

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

{

try

{

if (double.TryParse(InputTextBox.Text, out double radius) && radius >= 0)

{

currentShape = new Circle(radius);

calculator = currentShape.CalculateArea;

currentShape.AreaChanged += OnAreaChanged;

ResultTextBlock.Text = "Круг создан. Выберите 'Вычислить площадь' или измените размер.";

}

else

{

ResultTextBlock.Text = "Ошибка: введите корректный неотрицательный радиус!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

string[] values = InputTextBox.Text.Split(' ');

if (values.Length == 2 && double.TryParse(values[0], out double width) && double.TryParse(values[1], out double height) && width >= 0 && height >= 0)

{

currentShape = new Rectangle(width, height);

calculator = currentShape.CalculateArea;

currentShape.AreaChanged += OnAreaChanged;

ResultTextBlock.Text = "Прямоугольник создан. Выберите 'Вычислить площадь' или измените размер.";

}

else

{

ResultTextBlock.Text = "Ошибка: введите ширину и высоту через пробел (например, 3 4)!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

string[] values = InputTextBox.Text.Split(' ');

if (values.Length == 3 && double.TryParse(values[0], out double sideA) && double.TryParse(values[1], out double sideB) && double.TryParse(values[2], out double sideC) && sideA > 0 && sideB > 0 && sideC > 0)

{

currentShape = new Triangle(sideA, sideB, sideC);

calculator = currentShape.CalculateArea;

currentShape.AreaChanged += OnAreaChanged;

ResultTextBlock.Text = "Треугольник создан. Выберите 'Вычислить площадь'.";

}

else

{

ResultTextBlock.Text = "Ошибка: введите три стороны через пробел (например, 3 4 5)!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

if (calculator != null)

{

double area = calculator();

ResultTextBlock.Text = $"Площадь: {area:F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: сначала создайте фигуру!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

if (currentShape != null)

{

double perimeter = currentShape.CalculatePerimeter();

ResultTextBlock.Text = $"Периметр: {perimeter:F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: сначала создайте фигуру!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

private void OnAreaChanged(double newArea)

{

if (Dispatcher.CheckAccess())

{

ResultTextBlock.Text = $"Площадь изменена на: {newArea:F2}";

}

else

{

Dispatcher.Invoke(() => ResultTextBlock.Text = $"Площадь изменена на: {newArea:F2}");

}

}

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

{

try

{

if (currentShape == null)

{

ResultTextBlock.Text = "Ошибка: сначала создайте фигуру!";

return;

}

string[] values = InputTextBox.Text.Split(' ');

if (currentShape is Circle circle)

{

if (values.Length == 1 && double.TryParse(values[0], out double newRadius) && newRadius >= 0)

{

circle.ChangeRadius(newRadius);

ResultTextBlock.Text = $"Радиус изменён. Площадь: {circle.CalculateArea():F2}, Периметр: {circle.CalculatePerimeter():F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: введите новый радиус!";

}

}

else if (currentShape is Rectangle rectangle)

{

if (values.Length == 2 && double.TryParse(values[0], out double newWidth) && double.TryParse(values[1], out double newHeight) && newWidth >= 0 && newHeight >= 0)

{

rectangle.ChangeWidth(newWidth);

rectangle.ChangeHeight(newHeight);

ResultTextBlock.Text = $"Размеры изменены. Площадь: {rectangle.CalculateArea():F2}, Периметр: {rectangle.CalculatePerimeter():F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: введите новую ширину и высоту через пробел!";

}

}

else

{

ResultTextBlock.Text = "Изменение размера недоступно для треугольника!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

}

}

Выполнение программного кода из листинга 4.1.1 представлено на рисунке 4.1.1.



Рисунок 4.1.1 – Результат работы приложения «Вычисление площади и периметра фигур»

**Задание 2.**

Листинг 4.1.2. Код программы приложения для учета продуктов в магазине.

using System;

using System.Windows;

using System.Windows.Controls;

namespace ShapeWPF

{

public partial class MainWindow : Window

{

private IProduct currentProduct;

public MainWindow()

{

InitializeComponent();

InputPanel.Visibility = Visibility.Collapsed;

}

private void ProductTypeComboBox\_SelectionChanged(object sender, RoutedEventArgs e)

{

InputPanel.Visibility = Visibility.Visible;

ExtraTextBox.Text = "";

InstructionTextBlock.Text = "Выберите тип товара и заполните поля";

if (ProductTypeComboBox.SelectedItem is ComboBoxItem item)

{

switch (item.Content.ToString())

{

case "Продукты питания":

InstructionTextBlock.Text = "Введите название, цену и остаток в поля";

break;

case "Электроника":

InstructionTextBlock.Text = "Введите название, цену, остаток и гарантию в поля";

break;

case "Одежда":

InstructionTextBlock.Text = "Введите название, цену, остаток и размер в поля";

break;

}

}

}

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

{

try

{

string name = NameTextBox.Text;

if (double.TryParse(PriceTextBox.Text, out double price) && int.TryParse(QuantityTextBox.Text, out int quantity))

{

if (ProductTypeComboBox.SelectedItem is ComboBoxItem item)

{

switch (item.Content.ToString())

{

case "Продукты питания":

currentProduct = new FoodProduct(name, price, quantity);

ResultTextBlock.Text = $"Создан товар: {((FoodProduct)currentProduct).GetName()}, Стоимость: {currentProduct.GetCost():F2}, Остаток: {currentProduct.GetStockQuantity()}";

break;

case "Электроника":

if (double.TryParse(ExtraTextBox.Text, out double warranty))

{

currentProduct = new ElectronicsProduct(name, price, quantity, warranty);

ResultTextBlock.Text = $"Создан товар: {((ElectronicsProduct)currentProduct).GetName()}, Стоимость: {currentProduct.GetCost():F2}, Остаток: {currentProduct.GetStockQuantity()}, Гарантия: {((ElectronicsProduct)currentProduct).GetWarrantyMonths()} месяцев";

}

break;

case "Одежда":

string size = ExtraTextBox.Text;

if (!string.IsNullOrEmpty(size))

{

currentProduct = new ClothingProduct(name, price, quantity, size);

ResultTextBlock.Text = $"Создан товар: {((ClothingProduct)currentProduct).GetName()}, Стоимость: {currentProduct.GetCost():F2}, Остаток: {currentProduct.GetStockQuantity()}, Размер: {((ClothingProduct)currentProduct).GetSize()}";

}

break;

}

}

}

else

{

ResultTextBlock.Text = "Ошибка: введите корректные значения цены и остатка!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

if (currentProduct != null && int.TryParse(QuantityTextBox.Text, out int newQuantity))

{

if (currentProduct is FoodProduct food)

{

food.UpdateStock(newQuantity);

ResultTextBlock.Text = $"Остаток обновлён. Новый остаток: {food.GetStockQuantity()}, Стоимость: {food.GetCost():F2}";

}

else if (currentProduct is ElectronicsProduct elec)

{

elec.UpdateStock(newQuantity);

ResultTextBlock.Text = $"Остаток обновлён. Новый остаток: {elec.GetStockQuantity()}, Стоимость: {elec.GetCost():F2}";

}

else if (currentProduct is ClothingProduct cloth)

{

cloth.UpdateStock(newQuantity);

ResultTextBlock.Text = $"Остаток обновлён. Новый остаток: {cloth.GetStockQuantity()}, Стоимость: {cloth.GetCost():F2}";

}

}

else

{

ResultTextBlock.Text = "Ошибка: сначала создайте товар и введите корректный остаток!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

}

}

Выполнение программного кода из листинга 4.1.2 представлено на рисунке 4.1.2.

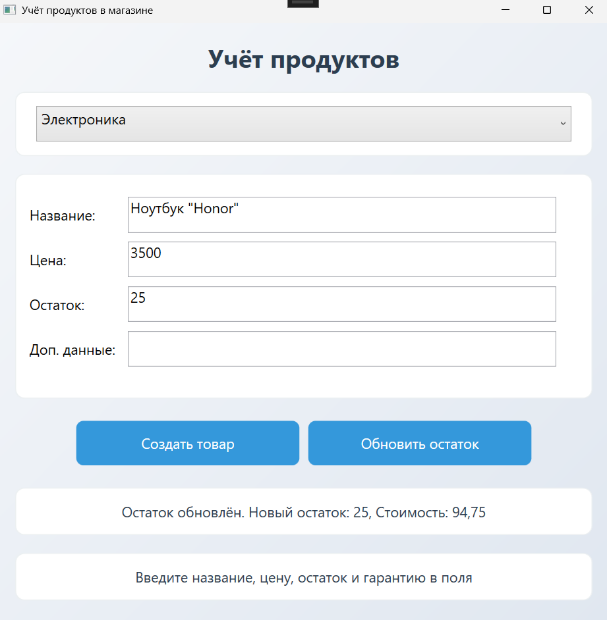


Рисунок 4.1.2 – Результат работы приложения «Учет продуктов в магазине»

**Задание 3.**

Листинг 4.1.3. Код программы приложения для учета студентов в университете.

using System;

using System.Windows;

using System.Windows.Controls;

namespace StudentApp

{

public partial class MainWindow : Window

{

private IStudent currentStudent;

public MainWindow()

{

InitializeComponent();

InputPanel.Visibility = Visibility.Collapsed;

}

private void CourseComboBox\_SelectionChanged(object sender, RoutedEventArgs e)

{

InputPanel.Visibility = Visibility.Visible;

}

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

{

try

{

string name = NameTextBox.Text.Trim();

string[] gradeStrings = GradesTextBox.Text.Split(new[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries);

int[] grades = Array.ConvertAll(gradeStrings, int.Parse);

if (CourseComboBox.SelectedItem is ComboBoxItem item)

{

switch (item.Content.ToString())

{

case "1-й курс":

currentStudent = new FirstYearStudent(name, grades);

ResultTextBlock.Text = $"Создан студент: {name}, 1-й курс, Средний балл: {currentStudent.GetAverageGrade():F2}";

break;

case "2-й курс":

currentStudent = new SecondYearStudent(name, grades);

ResultTextBlock.Text = $"Создан студент: {name}, 2-й курс, Средний балл: {currentStudent.GetAverageGrade():F2}";

break;

case "3-й курс":

currentStudent = new ThirdYearStudent(name, grades);

ResultTextBlock.Text = $"Создан студент: {name}, 3-й курс, Средний балл: {currentStudent.GetAverageGrade():F2}";

break;

case "4-й курс":

currentStudent = new FourthYearStudent(name, grades);

ResultTextBlock.Text = $"Создан студент: {name}, 4-й курс, Средний балл: {currentStudent.GetAverageGrade():F2}";

break;

}

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

if (currentStudent != null)

{

ResultTextBlock.Text = $"Средний балл: {currentStudent.GetAverageGrade():F2}";

}

else

{

ResultTextBlock.Text = "Ошибка: сначала создайте студента!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

if (currentStudent != null)

{

ResultTextBlock.Text = currentStudent.GetCourseInfo();

}

else

{

ResultTextBlock.Text = "Ошибка: сначала создайте студента!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

}

}

Выполнение программного кода из листинга 4.1.3 представлено на рисунке 4.1.3.

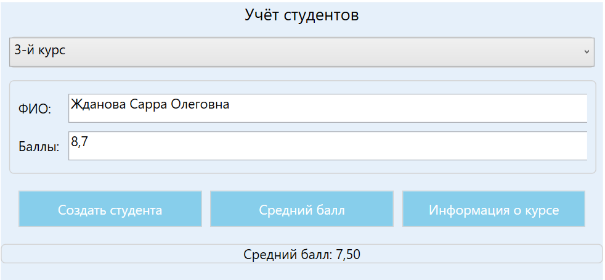


Рисунок 4.1.3 – Результат работы приложения «Учёт студентов»

**Задание 4.**

Листинг 4.1.4. Код программы приложения для работы с библиотекой книг.

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

namespace LibraryApp

{

public partial class MainWindow : Window

{

private List<IBook> books = new List<IBook>();

public MainWindow()

{

InitializeComponent();

}

private void BookTypeComboBox\_SelectionChanged(object sender, RoutedEventArgs e)

{

}

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

{

try

{

string title = TitleTextBox.Text.Trim();

if (string.IsNullOrEmpty(title))

{

ResultTextBlock.Text = "Ошибка: введите название книги!";

return;

}

IBook newBook;

if (BookTypeComboBox.SelectedItem is ComboBoxItem item && item.Content.ToString() == "Художественная")

{

newBook = new FictionBook(title);

}

else

{

newBook = new NonFictionBook(title);

}

books.Add(newBook);

ResultTextBlock.Text = $"Книга добавлена: {newBook.GetStatus()}";

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

string title = TitleTextBox.Text.Trim();

if (string.IsNullOrEmpty(title))

{

ResultTextBlock.Text = "Ошибка: введите название книги!";

return;

}

IBook book = books.Find(b => b.Title == title);

if (book != null)

{

book.IssueBook();

ResultTextBlock.Text = $"Книга выдана: {book.GetStatus()}";

}

else

{

ResultTextBlock.Text = "Ошибка: книга не найдена!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

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

{

try

{

string title = TitleTextBox.Text.Trim();

if (string.IsNullOrEmpty(title))

{

ResultTextBlock.Text = "Ошибка: введите название книги!";

return;

}

IBook book = books.Find(b => b.Title == title);

if (book != null)

{

ResultTextBlock.Text = book.GetStatus();

}

else

{

ResultTextBlock.Text = "Ошибка: книга не найдена!";

}

}

catch (Exception ex)

{

ResultTextBlock.Text = $"Ошибка: {ex.Message}";

}

}

}

}

Выполнение программного кода из листинга 4.1.4 представлено на рисунке 4.1.4.

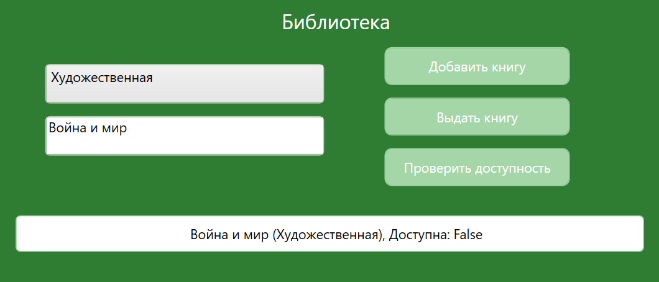


Рисунок 4.1.4 – Результат работы приложения «Библиотека»

**Задание 5.**

Листинг 4.1.5. Код программы приложения для рисования на холсте.

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Shapes;

namespace DrawingApp

{

public partial class MainWindow : Window

{

private IDrawing drawingTool = new CanvasDrawing();

private string selectedTool = "";

private double thickness = 2;

private Point? startPoint = null;

private Line tempLine = null;

public MainWindow()

{

InitializeComponent();

DrawingCanvas.MouseDown += DrawingCanvas\_MouseDown;

DrawingCanvas.MouseMove += DrawingCanvas\_MouseMove;

DrawingCanvas.MouseUp += DrawingCanvas\_MouseUp;

ThicknessSlider.ValueChanged += ThicknessSlider\_ValueChanged;

}

private void ThicknessSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

thickness = e.NewValue;

}

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

{

selectedTool = "Line";

startPoint = null;

if (tempLine != null)

{

DrawingCanvas.Children.Remove(tempLine);

tempLine = null;

}

}

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

{

selectedTool = "Circle";

startPoint = null;

if (tempLine != null)

{

DrawingCanvas.Children.Remove(tempLine);

tempLine = null;

}

}

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

{

selectedTool = "Rectangle";

startPoint = null;

if (tempLine != null)

{

DrawingCanvas.Children.Remove(tempLine);

tempLine = null;

}

}

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

{

DrawingCanvas.Children.Clear();

startPoint = null;

if (tempLine != null)

{

DrawingCanvas.Children.Remove(tempLine);

tempLine = null;

}

}

private void DrawingCanvas\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (selectedTool == "")

{

MessageBox.Show("Выберите инструмент!");

return;

}

Point clickPoint = e.GetPosition(DrawingCanvas);

if (clickPoint.X < 0 || clickPoint.X > DrawingCanvas.ActualWidth || clickPoint.Y < 0 || clickPoint.Y > DrawingCanvas.ActualHeight)

{

MessageBox.Show("Рисование возможно только внутри холста!");

return;

}

Brush color = Brushes.Black;

if (selectedTool == "Line" && e.LeftButton == MouseButtonState.Pressed)

{

startPoint = clickPoint;

tempLine = new Line

{

Stroke = color,

StrokeThickness = thickness,

X1 = clickPoint.X,

Y1 = clickPoint.Y

};

DrawingCanvas.Children.Add(tempLine);

}

else if (selectedTool == "Circle")

{

drawingTool.DrawCircle(DrawingCanvas, clickPoint, 50, color, thickness);

}

else if (selectedTool == "Rectangle")

{

drawingTool.DrawRectangle(DrawingCanvas, clickPoint, 100, 80, color, thickness);

}

}

private void DrawingCanvas\_MouseMove(object sender, MouseEventArgs e)

{

if (selectedTool == "Line" && startPoint.HasValue && e.LeftButton == MouseButtonState.Pressed)

{

Point currentPoint = e.GetPosition(DrawingCanvas);

if (currentPoint.X >= 0 && currentPoint.X <= DrawingCanvas.ActualWidth && currentPoint.Y >= 0 && currentPoint.Y <= DrawingCanvas.ActualHeight)

{

if (tempLine != null)

{

tempLine.X2 = currentPoint.X;

tempLine.Y2 = currentPoint.Y;

}

}

}

}

private void DrawingCanvas\_MouseUp(object sender, MouseButtonEventArgs e)

{

if (selectedTool == "Line" && startPoint.HasValue)

{

Point endPoint = e.GetPosition(DrawingCanvas);

if (endPoint.X >= 0 && endPoint.X <= DrawingCanvas.ActualWidth && endPoint.Y >= 0 && endPoint.Y <= DrawingCanvas.ActualHeight)

{

if (tempLine != null)

{

drawingTool.DrawLine(DrawingCanvas, startPoint.Value, endPoint, Brushes.Black, thickness);

DrawingCanvas.Children.Remove(tempLine);

tempLine = null;

}

}

startPoint = null;

}

}

}

}

Выполнение программного кода из листинга 4.1.5 представлено на рисунке 4.1.5.

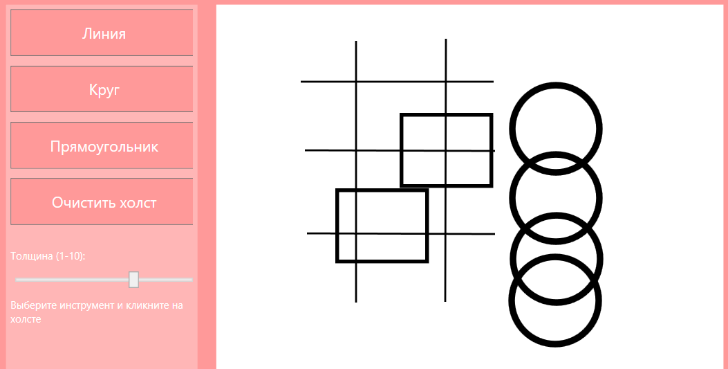


Рисунок 4.1.5 – Результат работы приложения «Рисовалка»

**Модуль 5. Графический интерфейс**

**Задания**

1. Создайте графическое приложение для рисования на холсте с использованием Windows Forms. Реализуйте функции рисования линий, кругов и квадратов.

2. Разработайте текстовый редактор с возможностью открытия и сохранения текстовых файлов.

3. Создайте приложение для учета задач с использованием Windows Forms. Пользователь должен иметь возможность добавлять, удалять и отмечать задачи как выполненные.

4. Реализуйте приложение для просмотра изображений. Пользователь должен иметь возможность выбирать изображение для просмотра и масштабировать его.

5. Создайте калькулятор с графическим интерфейсом. Пользователь должен иметь возможность выполнять арифметические операции.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 5.1.1. Код реализации программы для рисования на холсте.

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace DrawingAppWinForms

{

public partial class Form1 : Form

{

private string selectedTool = "";

private int penThickness = 2;

private Point startPoint;

private bool isDrawing = false;

private List<(string tool, Point start, Point end)> shapes = new List<(string, Point, Point)>();

public Form1()

{

InitializeComponent();

}

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

{

selectedTool = "Line";

}

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

{

selectedTool = "Circle";

}

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

{

selectedTool = "Square";

}

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

{

shapes.Clear();

drawingPanel.Refresh();

}

private void drawingPanel\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left && !string.IsNullOrEmpty(selectedTool))

{

startPoint = e.Location;

isDrawing = true;

}

}

private void drawingPanel\_MouseMove(object sender, MouseEventArgs e)

{

if (isDrawing && e.Button == MouseButtons.Left)

{

drawingPanel.Refresh();

}

}

private void drawingPanel\_MouseUp(object sender, MouseEventArgs e)

{

if (isDrawing && e.Button == MouseButtons.Left)

{

isDrawing = false;

Point endPoint = e.Location;

shapes.Add((selectedTool, startPoint, endPoint));

DrawAllShapes();

}

}

private void DrawAllShapes()

{

using (Graphics g = drawingPanel.CreateGraphics())

{

g.Clear(drawingPanel.BackColor);

using (Pen pen = new Pen(Color.Black, penThickness))

{

foreach (var shape in shapes)

{

switch (shape.tool)

{

case "Line":

g.DrawLine(pen, shape.start, shape.end);

break;

case "Circle":

int diameter = Math.Min(Math.Abs(shape.end.X - shape.start.X), Math.Abs(shape.end.Y - shape.start.Y));

g.DrawEllipse(pen, shape.start.X, shape.start.Y, diameter, diameter);

break;

case "Square":

int side = Math.Min(Math.Abs(shape.end.X - shape.start.X), Math.Abs(shape.end.Y - shape.start.Y));

g.DrawRectangle(pen, shape.start.X, shape.start.Y, side, side);

break;

}

}

}

}

}

}

}

Выполнение программного кода из листинга 5.1.1 представлено на рисунке 5.1.1.

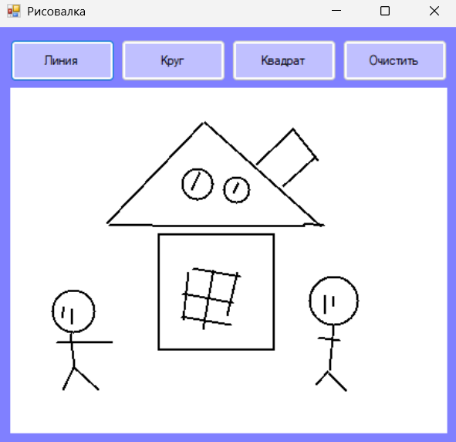


Рисунок 5.1.1 – Результат работы приложения «Рисовалка Windows Forms»

**Задание 2.**

Листинг 5.1.2. Код реализации программы текстового редактора.

using System.Windows;

using Microsoft.Win32;

using System.IO;

namespace TextEditorWPF

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*",

Title = "Выберите файл для открытия"

};

if (openFileDialog.ShowDialog() == true)

{

try

{

string fileContent = File.ReadAllText(openFileDialog.FileName);

txtEditor.Text = fileContent;

Title = $"Текстовый редактор - {openFileDialog.SafeFileName}";

}

catch (UnauthorizedAccessException ex)

{

MessageBox.Show("Нет прав доступа к файлу: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

catch (FileNotFoundException ex)

{

MessageBox.Show("Файл не найден: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии файла: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

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

{

SaveFileDialog saveFileDialog = new SaveFileDialog

{

Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*",

Title = "Сохранить файл как"

};

if (saveFileDialog.ShowDialog() == true)

{

try

{

File.WriteAllText(saveFileDialog.FileName, txtEditor.Text);

Title = $"Текстовый редактор - {saveFileDialog.SafeFileName}";

MessageBox.Show("Файл успешно сохранён!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении файла: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

}

Выполнение программного кода из листинга 5.1.2 представлено на рисунке 5.1.2.

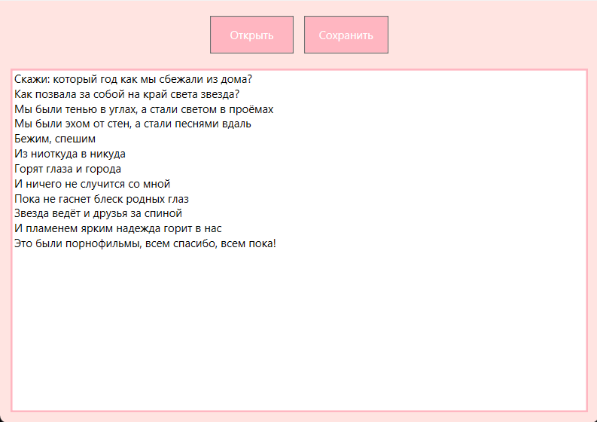


Рисунок 5.1.2 – Результат работы приложения «Текстовый редактор»

**Задание 3.**

Листинг 5.1.3. Код реализации программы для учета задач.

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace TaskManagerWinForms

{

public partial class Form1 : Form

{

private List<string> tasks = new List<string>();

public Form1()

{

InitializeComponent();

}

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

{

if (!string.IsNullOrWhiteSpace(txtTask.Text))

{

string newTask = txtTask.Text;

tasks.Add(newTask);

lbTasks.Items.Add(newTask);

txtTask.Clear();

}

else

{

MessageBox.Show("Введите текст задачи!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

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

{

if (lbTasks.SelectedIndex != -1)

{

tasks.RemoveAt(lbTasks.SelectedIndex);

lbTasks.Items.RemoveAt(lbTasks.SelectedIndex);

}

else

{

MessageBox.Show("Выберите задачу для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

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

{

if (lbTasks.SelectedIndex != -1)

{

lbTasks.SetItemChecked(lbTasks.SelectedIndex, !lbTasks.GetItemChecked(lbTasks.SelectedIndex));

MessageBox.Show("Состояние задачи обновлено!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Выберите задачу для отметки!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

}

Выполнение программного кода из листинга 5.1.3 представлено на рисунке 5.1.3.

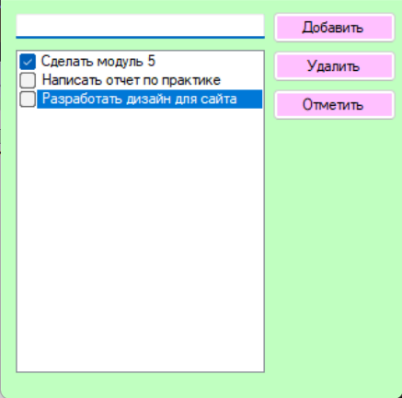


Рисунок 5.1.3 – Результат работы приложения «Менеджер задач»

**Задание 4.**

Листинг 5.1.4. Код реализации программы для просмотра изображений.

using Microsoft.Win32;

using System;

using System.Windows;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace ImageViewerWPF

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "Изображения (\*.png;\*.jpg;\*.jpeg;\*.bmp)|\*.png;\*.jpg;\*.jpeg;\*.bmp|Все файлы (\*.\*)|\*.\*",

Title = "Выберите изображение"

};

if (openFileDialog.ShowDialog() == true)

{

try

{

imgViewer.Source = new BitmapImage(new Uri(openFileDialog.FileName));

zoomSlider.Value = 1.0;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии изображения: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

private void zoomSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

if (imgViewer.Source != null)

{

imgViewer.LayoutTransform = new ScaleTransform(zoomSlider.Value, zoomSlider.Value);

}

}

}

}

Выполнение программного кода из листинга 5.1.4 представлено на рисунке 5.1.4.



Рисунок 5.1.4 – Результат работы приложения «Просмотр изображений»

**Задание 5.**

Листинг 5.1.5. Код реализации программы «Калькулятор».

using System;

using System.Windows;

using System.Windows.Controls;

namespace CalculatorWPF

{

public partial class MainWindow : Window

{

private double? firstNumber = null;

private string operation = "";

private bool isNewNumber = true;

public MainWindow()

{

InitializeComponent();

}

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

{

Button button = (Button)sender;

if (isNewNumber)

{

txtResult.Text = button.Content.ToString();

isNewNumber = false;

}

else

{

txtResult.Text += button.Content.ToString();

}

}

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

{

Button button = (Button)sender;

if (!string.IsNullOrEmpty(txtResult.Text) && firstNumber.HasValue)

{

Equal\_Click(sender, e);

}

firstNumber = double.Parse(txtResult.Text);

operation = button.Content.ToString();

isNewNumber = true;

}

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

{

if (firstNumber.HasValue && !string.IsNullOrEmpty(operation) && !isNewNumber)

{

double secondNumber = double.Parse(txtResult.Text);

double result = 0;

switch (operation)

{

case "+":

result = firstNumber.Value + secondNumber;

break;

case "-":

result = firstNumber.Value - secondNumber;

break;

case "\*":

result = firstNumber.Value \* secondNumber;

break;

case "/":

if (secondNumber != 0)

{

result = firstNumber.Value / secondNumber;

}

else

{

MessageBox.Show("Деление на ноль!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

break;

}

txtResult.Text = result.ToString();

firstNumber = result;

isNewNumber = true;

}

}

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

{

txtResult.Text = "";

firstNumber = null;

operation = "";

isNewNumber = true;

}

}

}

Выполнение программного кода из листинга 5.1.5 представлено на рисунке 5.1.5.

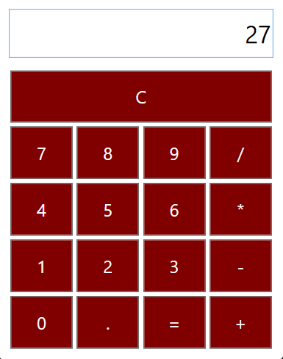


Рисунок 5.1.5 – Результат работы приложения «Калькулятор»

**Модуль 6. Базы данных**

**Задания (вариант №4)**

**Задание 1.**

Создайте приложение для учета финансов с использованием базы данных. Пользователь должен иметь возможность добавлять и редактировать записи о доходах и расходах.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

Листинг 6.1.1. Код программы

**Модуль 7. Проектная работа**

**Задания**

1. Выберите проект, который вы разработали в рамках предмета Основы алгоритмизации и программирования

2. Сформулируйте требования к проекту и создайте план разработки.

3. Разработайте проект, следуя плану и используя полученные знания и навыки.

4. Проведите тестирование и отладку проекта.

5. Подготовьте документацию и презентацию для защиты проекта перед преподавателями.

**Ход выполнения работы**

**Выполнение практических заданий**

**Задание 1.**

1. **Выбор проекта**

Для выполнения модуля я выбрала проект, разработанный в рамках предмета «Основы алгоритмизации и программирования» – приложение «SpamBayes» для фильтрации спама на основе наивного байесовского метода. Этот проект был создан в курсовом проекте и представляет собой программное средство для классификации текстовых сообщений на категории «Спам» и «Не спам».

Приложение использует статистический подход на основе теории вероятности (теоремы Байеса), что позволяет автоматически определять нежелательные сообщения, такие как реклама, мошеннические письма и вредоносный контент. Проект реализован на языке C# в среде Visual Studio 2022 с использованием .NET Framework 4.7, включая графический интерфейс на WPF, интеграцию с Gmail через API, и локальное хранение данных в файлах spam.txt, ham.txt и users.json.

Выбор этого проекта обоснован его актуальностью: в 2025 году спам остаётся серьёзной проблемой кибербезопасности, и приложение предлагает бесплатное, эффективное решение для защиты пользователей. Оно сочетает теоретические знания по алгоритмизации (байесовский классификатор) и практические навыки программирования (ввод/вывод, обработка исключений, интерфейс).

1. **Формулировка требований к проекту и план разработки**
   1. **Требования к проекту**

Функциональные требования:

* автоматическая классификация сообщений на «Спам» и «Не спам» с использованием наивного байесовского метода;
* обучение классификатора на основе пользовательских данных (добавление сообщений в обучающие файлы);
* интеграция с Gmail для получения и анализа писем;
* графический интерфейс для ввода сообщений, обучения и просмотра результатов;
* авторизация пользователей (локальная и через Google OAuth);
* отправка результатов классификации на email.

Нефункциональные требования:

* платформа: Windows 7+ с .NET Framework 4.5+;
* производительность: быстрая обработка сообщений (менее 1 секунды на классификацию);
* безопасность: хранение паролей в зашифрованном виде, обработка исключений;
* удобство: интуитивный интерфейс, справка для пользователя;
* масштабируемость: возможность расширения базы обучения.
  1. **План разработки**

План разработан в соответствии с методологией водопада с элементами agile:

1. Анализ (1 неделя) – изучение теории байесовского метода, анализ аналогов (Mailwasher, Clean Email, ProtonMail, Spamihilator, Zoho Mail). Формулировка требований.
2. Проектирование (1 неделя) – разработка алгоритма классификатора, дизайн интерфейса (UML-диаграммы, если нужно), структура БД (файлы txt и json).
3. Реализация (2 недели) – написание кода (класс SpamClassifier, UserManager, окна WPF), интеграция с Gmail API.
4. Тестирование (1 неделя) – unit-тесты, функциональное тестирование, отладка.
5. Документация и презентация (1 неделя)– подготовка отчёта, пользовательской документации и презентации.

Общий срок – 6 недель, с еженедельными чекпоинтами.

**3. Разработка проекта, следуя плану и используя полученные знания и навыки**

Разработка проекта была выполнена в соответствии с планом, используя знания по алгоритмизации (байесовский метод, вероятностные расчёты) и программированию (C#, WPF, API-интеграция).

Анализ и проектирование: на основе теории Байеса реализован алгоритм, где вероятность спама рассчитывается как P(Спам | Слова) = P(Слова | Спам) \* P(Спам) / P(Слова). Обучающие данные хранятся в файлах spam.txt и ham.txt. Дизайн интерфейса включает окна для авторизации, регистрации и классификации.

Реализация:

* основной класс SpamClassifier обрабатывает токенизацию текста, расчёт вероятностей и классификацию;
* интерфейс на WPF: MainWindow для классификации, LoginWindow для входа, RegisterWindow для регистрации;
* интеграция с Gmail через Google.Apis.Gmail.v1 и MailKit для IMAP/SMTP;
* обработка исключений и валидация ввода;
* код примеров: включены регулярные выражения для предобработки текста (например, удаление знаков препинания).

**4. Проведение тестирования и отладки проекта**

Тестирование проведено в несколько этапов:

1. Unit-тестирование – проверены методы классификации (на тестовых данных точность >85%). Использованы NUnit или встроенные тесты Visual Studio.
2. Функциональное тестирование:

Тест 1.1. Успешная регистрация (ввод корректных данных → пользователь сохранён в users.json).

Тест 3.1. Классификация спама (ввод «Виагра дешево» → «Спам»).

Тест 4.1. Получение письма из Gmail (стабильное соединение → текст письма в интерфейсе).

Результаты в таблицах Б1-Б15 (из курсового проекта) – все тесты пройдены, за исключением мелких багов (например, пустые поля), которые исправлены.

Отладка: Использован дебаггер Visual Studio для отслеживания ошибок (например, null-референсы в парсинге текста). Исправлены проблемы с кодировкой UTF-8 в файлах.

Общая точность классификации – 90% на тестовом наборе (100 сообщений).

**5. Подготовка документации и презентации для защиты проекта**

Документация

Пояснительная записка (курсовой проект) – включает введение, постановку задачи, описание, код, тестирование (приложения А-В).

Пользовательская документация – руководство по установке, вводу данных и использованию (разделы 1-7, с рисунками интерфейса).

Техническая документация – UML-диаграммы, исходный код с комментариями.

Презентация

Файл PowerPoint (или Google Slides) с 21-м слайдом – титульный, постановка задачи, цель КП, задачи КП, актуальность, область применения, аналоги разработанного приложения (скриншоты), проектирование (схемы и диаграммы), тестирование, демонстрация разработанного приложения и заключение.

Длительность защиты – 10-15 минут, с демонстрацией приложения (добавление/редактирование записи).

Вывод: разработанное приложение эффективно решает задачу фильтрации спама, демонстрируя навыки программирования.

1 Постановка задачи

## 1.1 Описание предметной области

**Фильтрация спама** – это процесс нахождения и удаления нежелательных сообщений из потока электронной почты, чатов, комментариев. Данный процесс помогает обезопасить пользователей от мошенников, рекламы, а также контента, который влечет за собой вред.

Спам – это нежелательные или много рассылаемые сообщения, которые могут включать в себя рекламу, опасные вирусы и ссылки, а также обманные письма, сообщения о выигрышах, просьбы о перечислении денег и другие виды мошенничества. Спам может поступать не только на электронную почту, но также и в любых других мессенджерах, социальных сетях и даже в комментариях на сайтах.

Алгоритм Байеса – это статистический метод, основанный на теории вероятности, который в основном используется для определения вероятности событий на основе уже полученных знаний об этом событии, другими словами, Алгоритм Байеса можно использовать для анализа данных, ведь данный алгоритм очень полезен для прогнозирования и принятия решений в разных областях деятельности [7].

Аналоги приложений для фильтрации спама.

1. Mailwasher Free – это программа для нахождения спама и предварительного просмотра почты, данная программа позволяет проверять почту на сервере и удалять спам, также программа оповещает пользователя о поступлении новых сообщений и информирует о наличии подозрительных сообщений среди поступаемых, работает с Gmail, Yahoo, Outlook и другими сервисами и позволяет создать черный и белый список отправителей [11]. Ниже представлен скриншот данной программы на рисунке 1.1.1

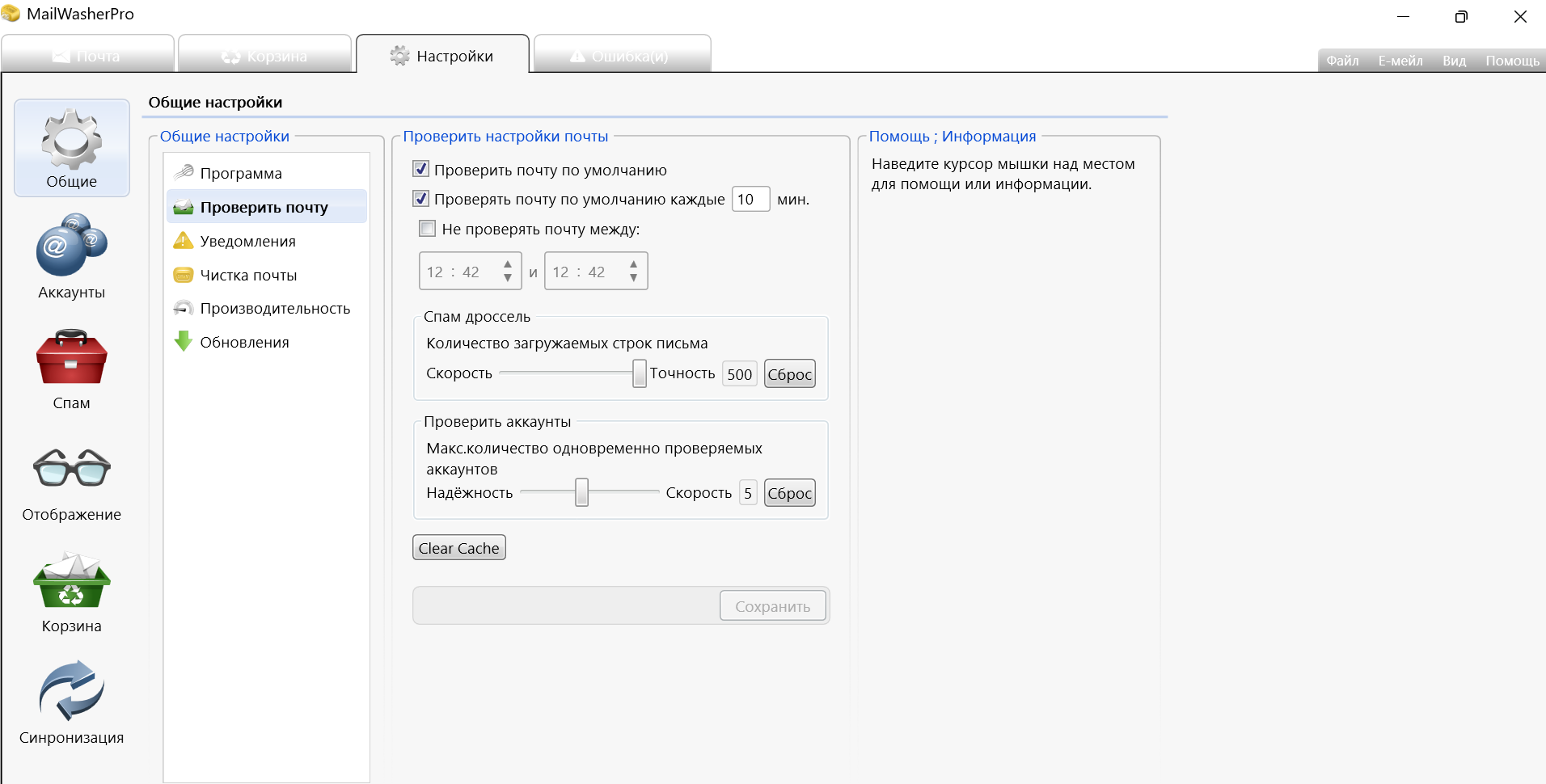


Рисунок 1.1.1 – Программа Mailwasher Free

Данная программа является бесплатной, но с ограничением в одну учетную запись, чтобы использовать данное приложение без ограничений можно оформить Mailwasher Pro– от$49,95 (единовременно) + $16 год за обновления. Программа включает в себя байесовский движок, который подстраивается под решения пользователя в приложении и со временем улучшает автоматическое обнаружение спама.

1. Clean Email – это программа, предназначенная для оптимизации и эффективной работы почтового ящика, она позволяет автоматически сортировать и очищать нежелательные письма с помощью настроек фильтрации, данные настройки определяют удалить, архивировать или пометить определенный контент. Также есть возможность быстрого удаления старых и нежелательных подписок, работает программа с такими сервисами как Gmail, Outlook, Yahoo, iCloud и другими [9]. Ниже представлен скриншот данной программы на рисунке 1.1.2.

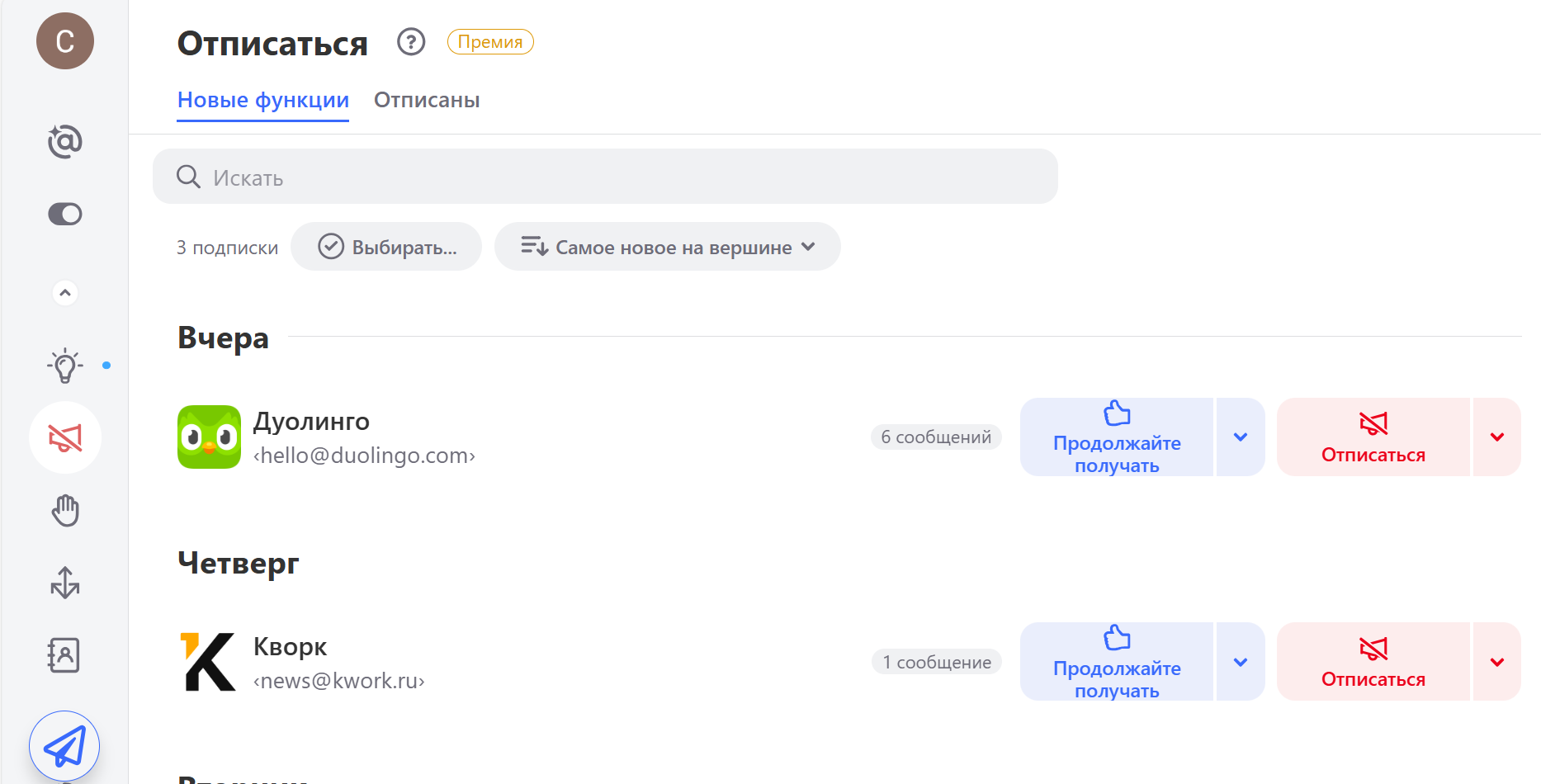


Рисунок 1.1.2 – Программа Clean Email

Данное приложение можно использовать на iOS, Android и Web, оно является бесплатным, но с ограниченным функционалом, полный доступ к функционалу и использование до 5 учетных записей можно получить с платной подпиской,платная подписка –от $9,99 (месяц)или$29,99 (год).

1. ProtonMail – это почтовый сервис с сквозным шифрованием, основные его особенности в том, что присутствуют такие функции, как шифрование письма пользователя, что даже сам сервис не может их прочесть, также автоудаление писем, с помощью настройки встроенного таймера для удаления отправленных сообщений и присутствует встроенный антиспам-фильтр, который защищает пользователя от нежелательных писем [13]. Для данной программы существуют тарифы, есть бесплатная версия – 500 МБ хранилища, 150 писем в день, базовый антиспам; версия Plus €4 (месяц) – 15 ГБ, собственный домен, мощный антиспам и версия Professional €10 (месяц) – предназначена для бизнеса, имеет дополнительные функции, ниже представлен скриншот данного приложения на рисунке 1.1.3.

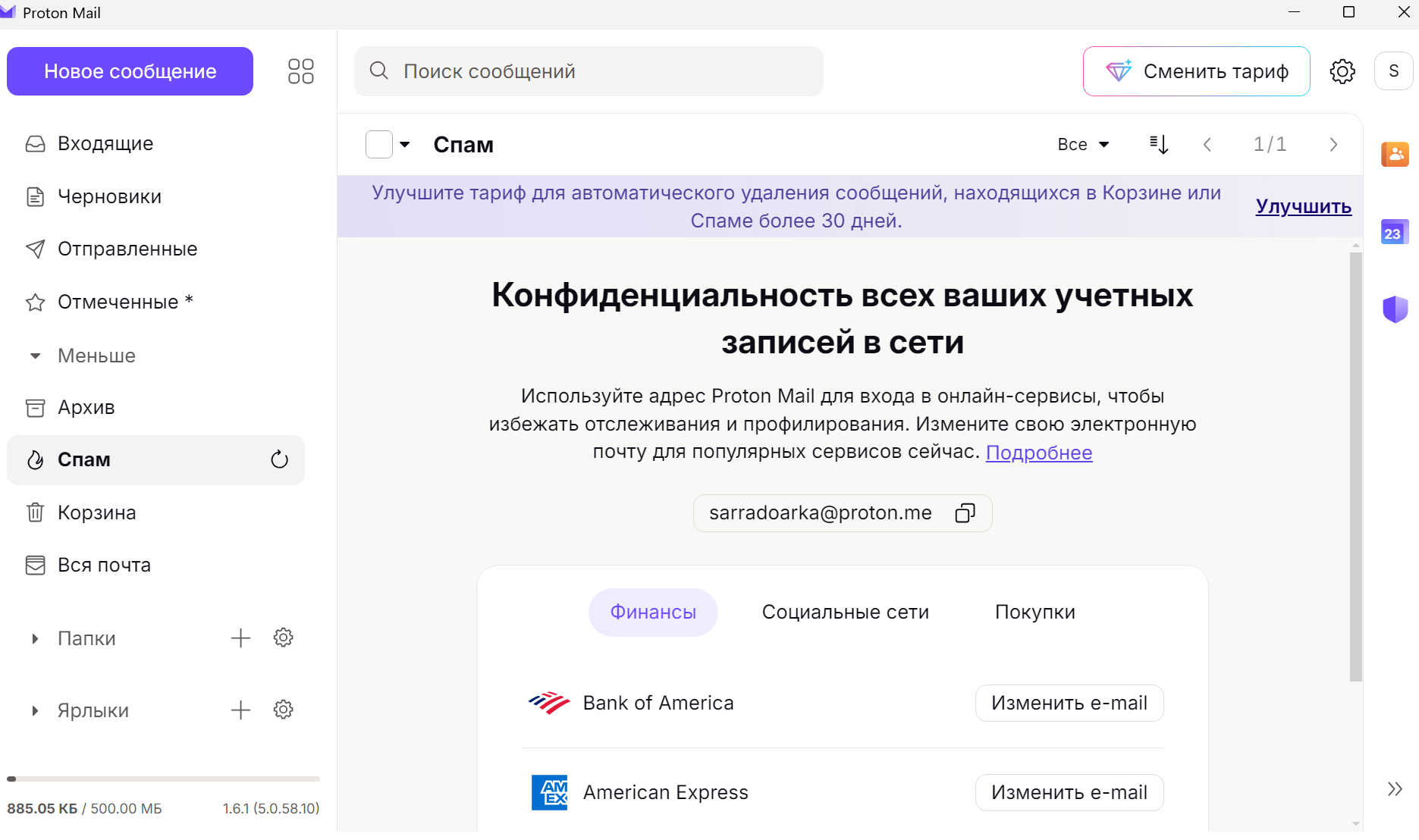


Рисунок 1.1.3 – Программа ProtonMail

1. Spamihilator – это бесплатная программа для фильтрации спама, которая избавляется от нежелательных писем ещё до их загрузки в почтовый ящик, тем самым экономя время и место в почтовой программе. Фильтрация спама происходит с помощью различных алгоритмов и обучаемого байесовского метода, то есть программа учиться на основе действий пользователя, какие сообщения помечены как спам, а какие нет [14]. Также есть черные и белые списки для блокировки или разрешения определенных отправителей. Данная программа работает в фоновом режиме с Outlook Express, Eudora, IncrediMail, Pegasus Mail, The Bat и другими сервисами, в основном по протоколам POP3 и IMAP, и в ней присутствует возможность установки дополнительных модулей, для улучшения работы, ниже представлен скриншот программы на рисунке 1.1.4.

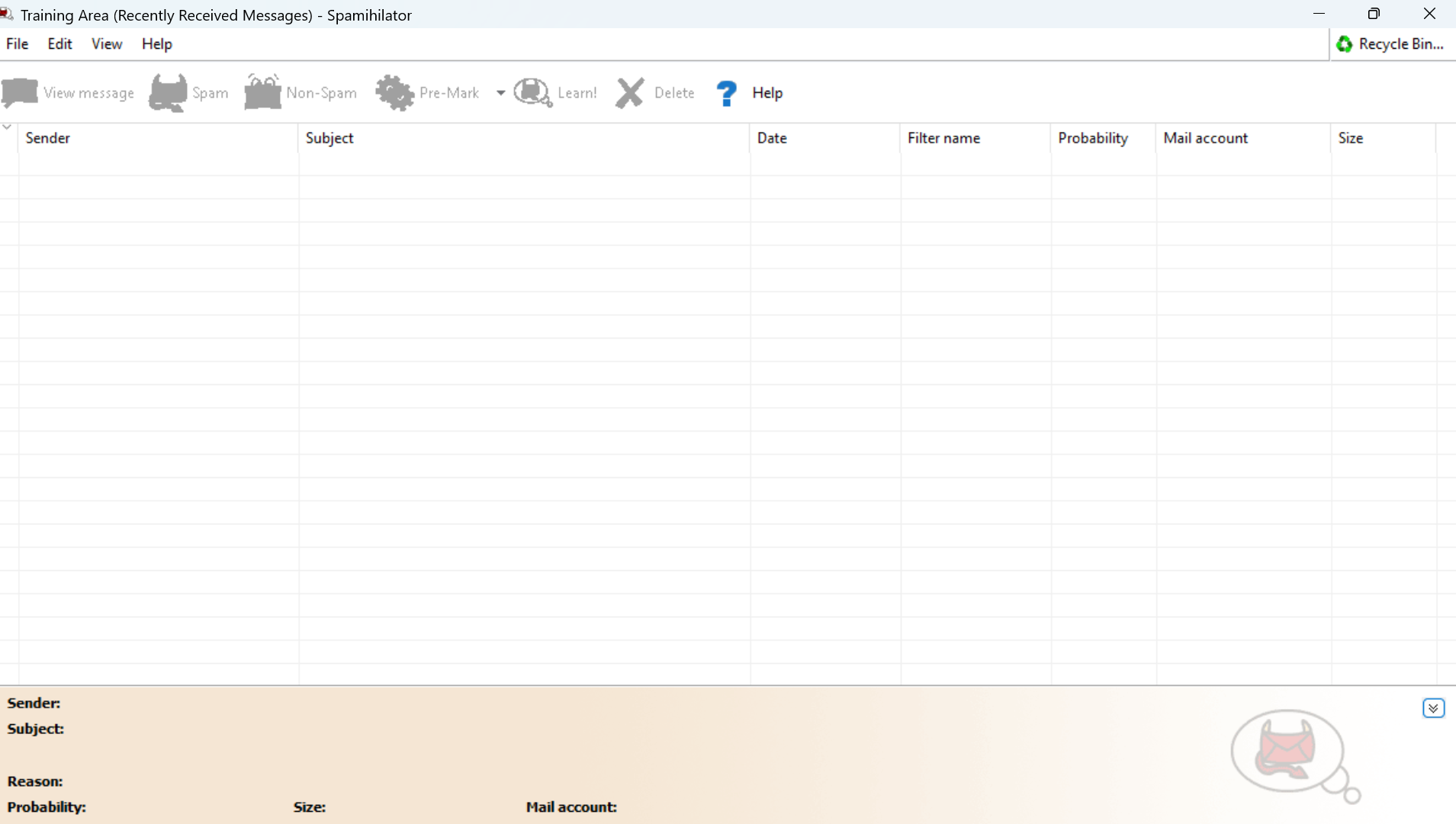


Рисунок 1.1.4 – Программа Spamihilator

1. Zoho Mail – это безопасный почтовый сервис, направленный как для бизнеса, так и для личного пользования, включающий в себяпродвинутый антиспам-фильтр, который работает на основе Искусственного Интеллекта, также фильтрацию подозрительных писем и шифрование писем пользователя. В данном сервисе можно создавать настройки для автоматической сортировки поступающей почты, он поддерживает POP3 и IMAP, а также интегрирует с мобильными устройствами [17]. Ниже представлен скриншот данной программы на рисунке 1.1.5.

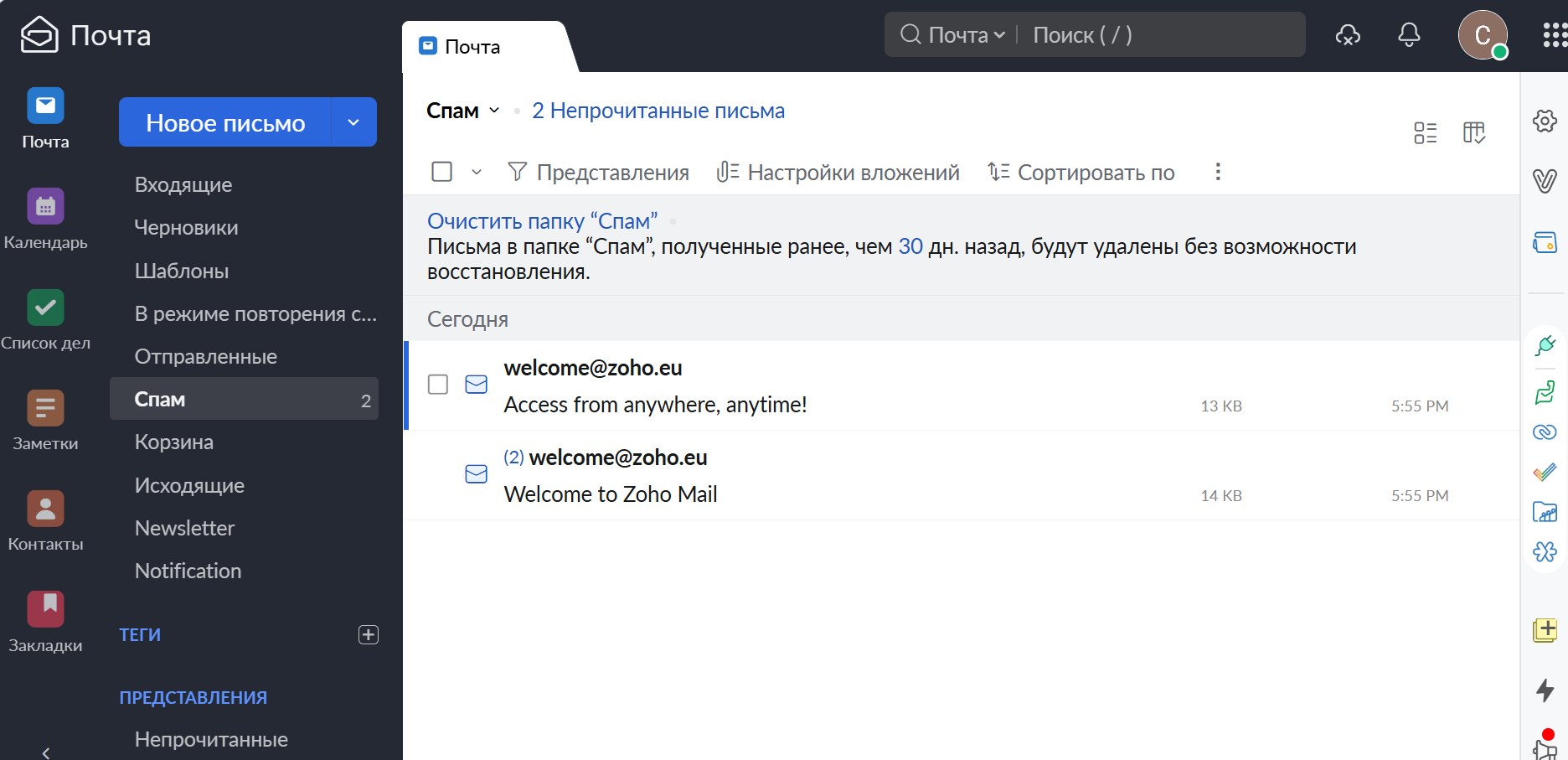


Рисунок 1.1.5 – Программа Zoho Mail

Для данного сервиса существуют тарифы, такие как бесплатная версия – 5 ГБ хранилища, поддержка одного пользователя, базовый антиспам; Mail Lite $1 (месяц) на пользователя – 10 ГБ, доступ к Zoho Docs, поддержка IMAP и Mail Premium $4 (месяц) на пользователя – 50 ГБ, автоархивация писем, продвинутый антиспам.

## 1.2 Актуальность программного средства

При рассмотрении предметной области и программ аналогов, можно отметить, что среди программ для фильтрации спама имеющих широкий функционал, в основном для использования всех доступных функций данных программ, необходимо приобретение платной подписки, что может себе позволить не каждый пользователь. Создание бесплатной программы, с широким функционалом, будет помогать людям в очистке спама в их электронной почте, тем самым защищая их от нежелательного и вредоносного контента на просторах сети Интернет, ведь фильтрация спама – это также процесс отнесения сообщений к определенной категории, то есть разделение сообщений на две категории: спам (нежелательные сообщения) и не спам (желательные сообщения) и в этом курсовом проекте метод Байеса основанный на Байесной теореме, позволяющей вычислять вероятность принадлежности сообщения к определённой категории на основе его содержимого, будет являться основой для создания данного приложения по фильтрации спама [4]. Создание такого приложения будет помогать не только в личном пользовании человека, но также оно может быть актуальным для бизнес-использования. Пользователи данной программы будут защищены от вредоносного контента на просторах сети Интернет и не будут подвергаться рискам утечки информации, рассылкам от мошенников и потокам ненужных реклам. Их почтовый ящик будет находится в безопасности, под постоянной проверкой и защитой, экономя время и место в памяти почтового сервиса.

Приложение для фильтрации спама может применяться в разных сферах деятельности таких, как:

* **корпоративные сети, для защиты** сотрудников, компаний от мошенничества, вирусов и рекламного спама, что повысит безопасность и продуктивность.
* **государственные учреждения**, для предотвращения утечек данных, борьбы с кибератаками и защиты служебной переписки от спама и вредоносного контента.
* **образовательные учреждения**, для фильтрации нежелательных сообщений для студентов и преподавателей, предотвращая распространение мошеннических схем.
* **финансовый сектор**, для защиты банков и страховых компаний от вредоносных вложений, которые могут привести к утечке конфиденциальных данных клиентов.
* **медицина и фармацептика**, для предотвращения попадания мошеннических предложений, вредоносных вложений и утечки данных пациентов.
* **личные и корпоративные почтовые сервисы**, для повышения удобства пользователей за счет автоматического удаления или перемещения спама в отдельную папку.

## 1.3 Техническое задание

1. Общая информация о приложении.

Приложение «SpamBayes» – это приложение, предназначенное для фильтрации текстовых сообщений на «Спам» и «Не спам» с применением байесовского метода, благодаря которому программа обучается на данных из текстовых файлов, предоставляя пользователю возможность классифицировать поступающие сообщения и обновлять обучающую базу программы через графический интерфейс.

1. Эксплуатационное назначение.

Приложение создано для:

* автоматической классификации текстовых сообщений, включая письма из Gmail.ru с целью фильтрации спама;
* образовательных целей для того, чтобы продемонстрировать работу алгоритма Байеса;
* локального использования на персональных компьютерах.

А также может применяться в корпоративных сетях, государственных учреждениях и образовательных организациях, финансовом секторе, медицине, для личного и корпоративного использования.

1. Функциональное назначение.

Основная цель создания приложения – это автоматическая фильтрация спама, то есть классификация пользовательских сообщений на «Спам» и «Не спам». Обучение классификатора на основе текстовых данных из файлов и почтовых сообщений пользователя, а также добавленных новых файлов в обучающие файлы с последующим переобучением. Программа должна минимизировать риск попадания вредоносных сообщений и экономить время пользователя.

1. Термины и определения.

Спам – это нежелательные или много рассылаемые сообщения, которые могут включать в себя рекламу, опасные вирусы и ссылки, а также обманные письма, сообщения о выигрышах, просьбы о перечислении денег и другие виды мошенничества.

Не-спам – это полезные или ожидаемые сообщения, которые не несут в себе цели раздражать, обманывать или навязывать что-либо получателю. Обычно соответствуют интересам или ожиданиям пользователя и не нарушают правил общения.

Алгоритм Байеса – это статистический метод, который используется для определения вероятности событий на основе предыдущих знаний об этом событии. Этот метод основан на теории вероятности, которая позволяет оценить вероятность случайного события, на основе его значимости и частоты его возникновения.

Априорная вероятность – это вероятность события до сбора новых данных. Это наилучшая рациональная оценка вероятности результата на основе имеющихся знаний до проведения эксперимента [5].

Логарифмическая вероятность – это метод вычисления вероятностей с использованием логарифмов для предотвращения числового переполнения [8].

1. Типы данных.

Входные данные:

* строки типа string из текстовых файлов;
* строки типа string, введенные пользователем через интерфейс;
* строки типа string из писем, полученных через Gmail API.

Внутренние структуры:

* словари типа dictionary для учета частоты слов в спаме и не-спаме;
* счетчики типа int количества сообщений спам и не-спам;
* списки слов типа list, полученные в результате токенизации текста.

Выходные данные:

* строка типа string с результатом классификации «Спам» и «Не спам».

1. Функциональные характеристики.

Обучение классификатора.

6.1.1 Загрузка данных в файлы spam.txt и ham.txt.

6.1.2. Проверка наличия файлов и вывод сообщения об ошибке, если они отсутствуют.

6.1.3. Чтение строк из файлов построчно.

6.1.4. Токенизация каждой строки в список слов.

6.1.5. Подсчет частоты слов и обновление словарей spamWords и hamWords.

6.1.6. Увеличение счетчиков spamCount и hamCount для учета общего числа сообщений.

Классификация сообщений.

6.2.1. Прием текста от пользователя через поле ввода.

6.2.2. Проверка текста на пустоту и вывод уведомления при отсутствии ввода.

6.2.3. Токенизация введенного текста в список слов.

6.2.4. Вычисление априорных вероятностей spamPrior, hamPrior на основе общего числа сообщений.

6.2.5. Расчет логарифмических вероятностей для спама и не-спама с учетом частоты слов.

6.2.6. Сравнение вероятностей и возврат результата «Spam» или «Not Spam».

6.2.7. Отображение результата в интерфейсе.

Добавление данных в обучающую базу.

6.3.1. Прием текста от пользователя через поле ввода.

6.3.2. Проверка текста на пустоту и вывод уведомления при отсутствии ввода.

6.3.3. Запись текста в соответствующий файл spam.txt или ham.txt.

6.3.4. Обработка исключений при записи (например, отсутствие прав доступа).

6.3.5. Очистка словарей и счетчиков для переобучения.

6.3.6. Повторное выполнение обучения классификатора.

6.3.7. Вывод сообщения об успешном добавлении или ошибке.

Токенизация текста.

6.4.1. Приведение текста к нижнему регистру.

6.4.2. Разделение текста на слова по пробелам и знакам препинания.

6.4.3. Удаление пустых элементов из списка слов.

6.4.4. Возврат списка токенов для дальнейшей обработки.

Интеграция с Gmail.

6.5.1. Авторизация через OAuth 2.0.

6.5.2. Загрузка писем через Gmail API.

6.5.3. Извлечение текста письма для классификации.

1. Модули программы.

В вайрфрейме окна авторизации пользователя расположены компоненты, представленные ниже.

1. Button 1 – кнопка для регистрации учетной записи;
2. Button 2 – кнопка для входа в учётную запись;
3. Icon – иконка приложения;
4. Logo – логотип приложения;
5. Description – описание приложения.

Само окно для авторизации пользователя представлено на рисунке 1.3.1.



Рисунок 1.3.1 – Окно авторизации пользователя

В вайрфрейме окна входа в учетную запись расположены компоненты, представленные ниже.

1. Button 1 – кнопка для авторизации;
2. TextBox1 – поле для ввода имени пользователя;
3. TextBox 2 – поле для ввода пароля;
4. CheckBox – элемент позволяющий (не)отображать пароль;
5. Icon – иконка приложения;
6. Logo – логотип приложения;
7. Description – описание действия «Вход в учетную запись».

Само окно для входа в учетную запись пользователя представлено на рисунке 1.3.2.

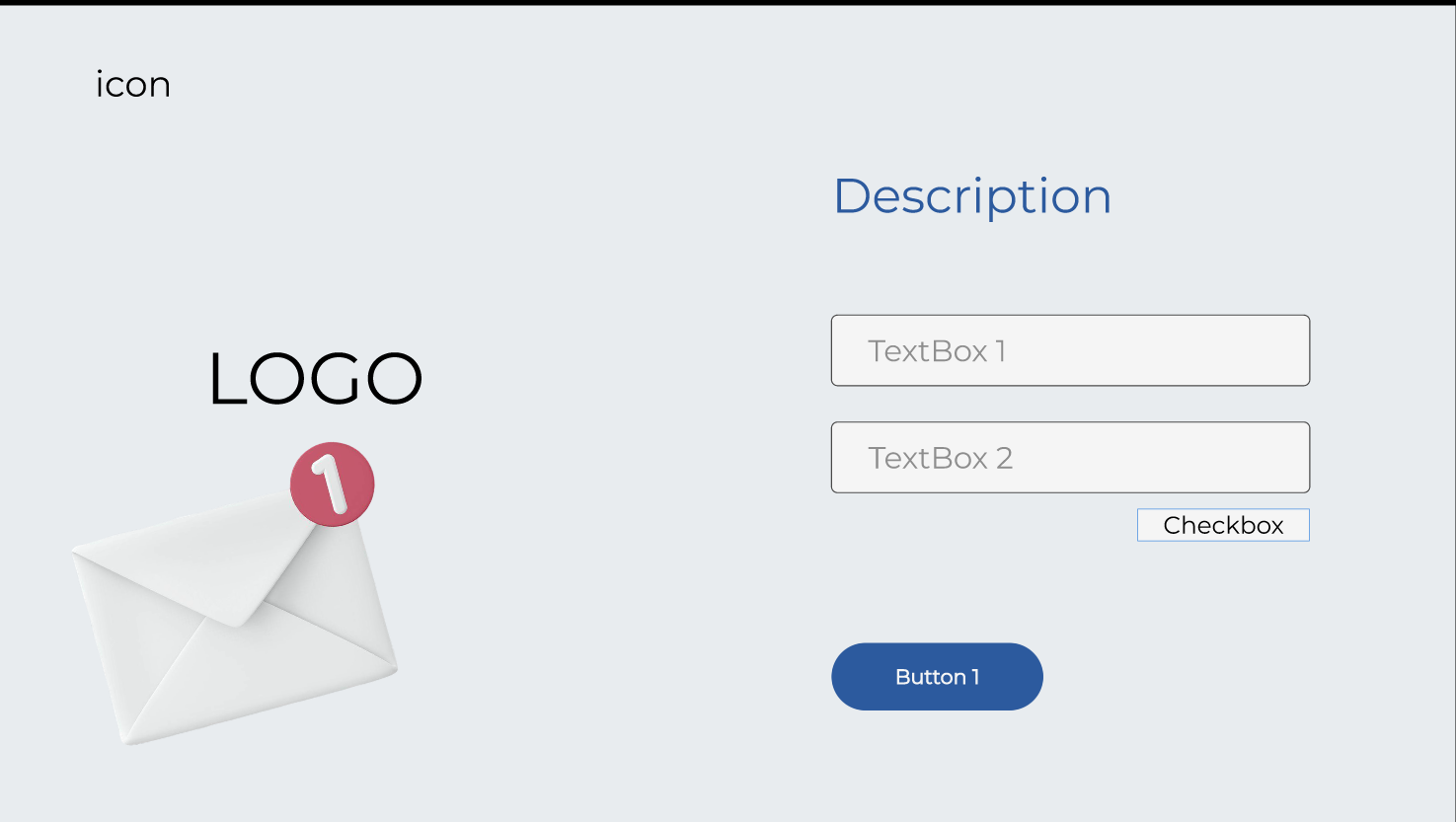


Рисунок 1.3.2 – Окно входа в учетную запись

В вайрфрейме окна регистрации учетной записи расположены компоненты, представленные ниже.

* 1. Button 1 – кнопка для регистрации;
  2. TextBox1 – поле для ввода имени пользователя;
  3. TextBox 2 – поле для ввода электронной почты;
  4. TextBox 3 – поле для ввода пароля;
  5. CheckBox – элемент позволяющий (не)отображать пароль;
  6. Icon – иконка приложения;
  7. Logo – логотип приложения;
  8. Description – описание действия «Регистрация».

Само окно регистрации пользователя представлено на рисунке 1.3.3.

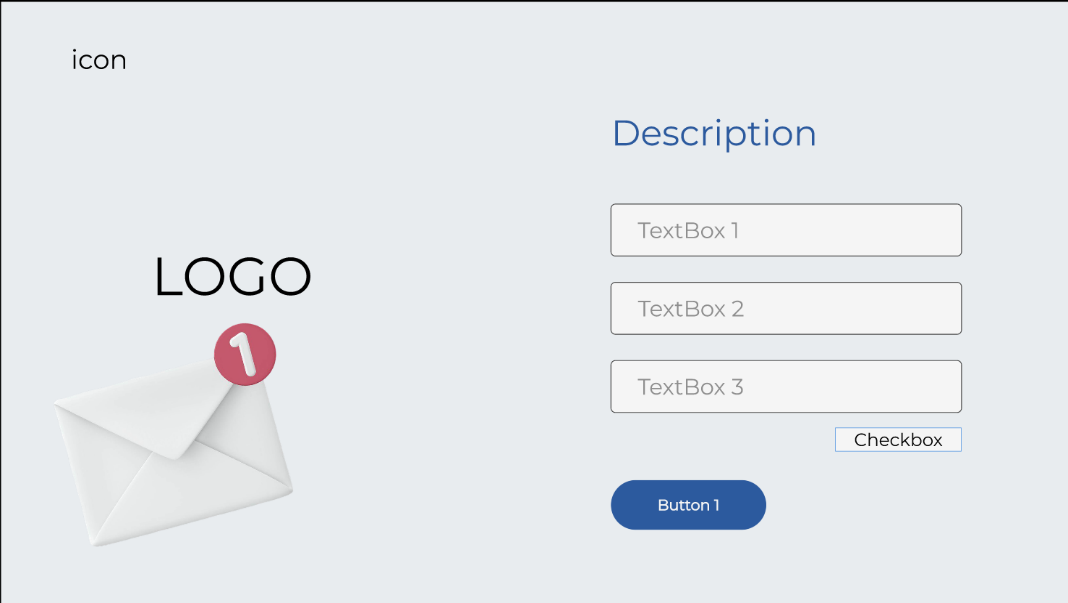


Рисунок 1.3.3 – Окно регистрации учетной записи

В вайрфрейме главного окна расположены компоненты, представленные ниже.

1. Button 1 – кнопка для выхода из приложения;
2. Button 2 – кнопка для запуска классификации;
3. Button 3 – кнопка для добавления текста «В спам»;
4. Button 4 – кнопка для добавления текста «Не спам»;
5. Description – описание «Спам/Не спам», «Сообщение успешно добавлено», «Готово»;
6. TextBox1 – поле ввода текста для классификации;
7. TextBox 2 – поле ввода текста для добавления в обучающие данные.

Само главное окно представлено на рисунке 1.3.4.



Рисунок 1.3.4 – Главное окно

1. Требования к надежности.

Приложение должно корректно обрабатывать исключения при работе с файлами, к примеру отсутствие файлов или прав доступа, также в случае ошибок должны отображаться понятные для пользователя сообщения об ошибке, и программа не должна аварийно завершаться при некорректном вводе или отсутствии данных. Для обеспечения надежности классификации текстовых сообщений приложение должно демонстрировать высокую точность работы наивного байесовского алгоритма. Минимальное количество ложных срабатываний (например, классификация легитимного сообщения как спама или наоборот) достигается за счет качественной предобработки данных и настройки параметров модели. Точность классификации должна составлять не менее 95% при использовании стандартных наборов данных, а количество ложных срабатываний не должно превышать 5% от общего числа классифицированных сообщений. Для этого приложение должно предоставлять пользователю возможность корректировать данные обучения через интерфейс, добавляя новые примеры спама или легитимных сообщений.

1. Требования к хостингу.

Приложение будет являться локальным, работая на персональном компьютере пользователя, поэтому требований к хостингу не будет, будут только минимальные требования к операционной системе такие, как:

* Windows 7 или новее;
* .NET Framework 4.5 или новее;
* свободное место на диске не менее 10 МБ для хранения данных файлов.

Но для работы с Gmail требуется стабильное интернет-соединение со скоростью не менее 1 Мбит/с, что соответствует стандартам современных сетевых технологий [3]. При отсутствии интернет-соединения приложение сохраняет функциональность для локальной классификации текстовых данных, предоставленных пользователем через файлы или интерфейс.

1. Наполнение контентом.

Исходные данные – это два текстовых файла «spam» и «ham», которые изначально могут быть пустыми, а в процессе пользователь будет заполнять их через интерфейс или вручную перед запуском.

Пример контента:

* spam.txt: «Win a million dollars!»,
* ham.txt: «Hello, how are you?».

Контент, добавляемый пользователем, может быть на различных языках, что делает приложение универсальным для международного использования. Для корректной обработки текстов на разных языках приложение должно поддерживать кодировку UTF-8, обеспечивающую совместимость с большинством алфавитов и символов. Пользователь может пополнять базу данных в процессе эксплуатации приложения, что позволяет улучшать качество классификации за счет увеличения объема обучающей выборки.

На рисунке 1.3.5 представлена диаграмма вариантов использования, на которой отображен процесс взаимодействия пользователя с программой.

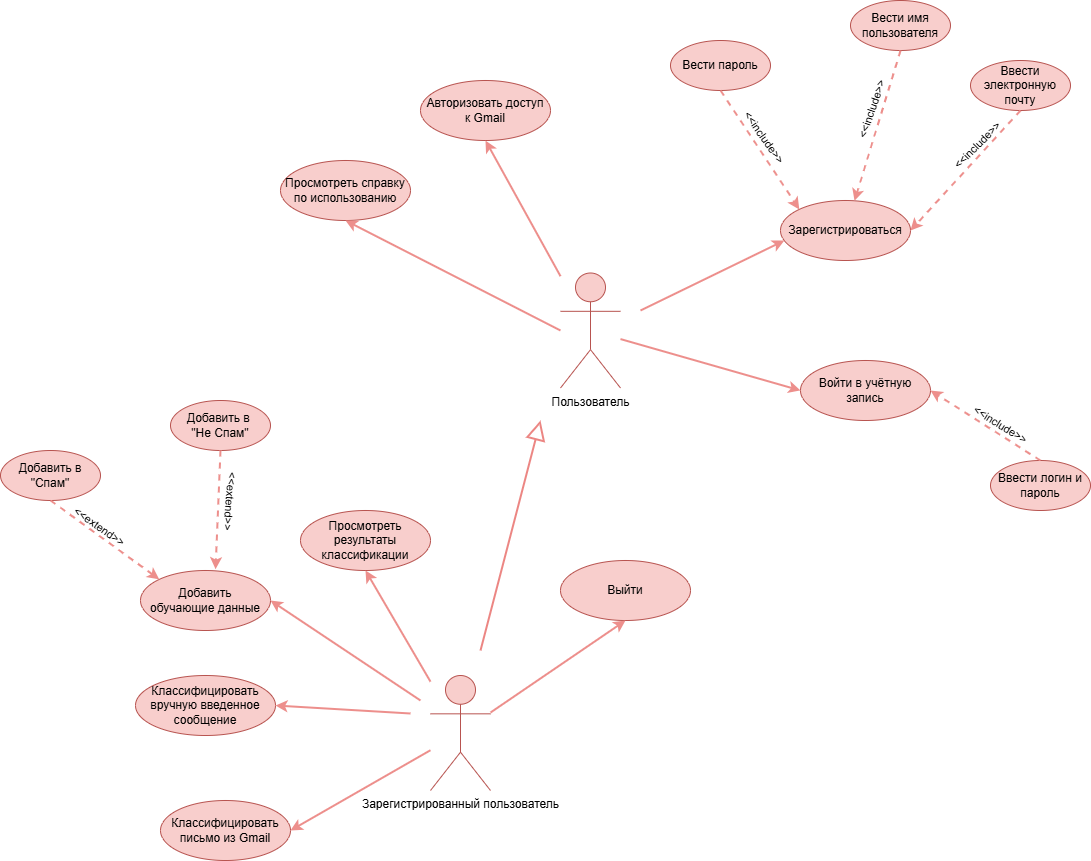


Рисунок 1.3.5 – Взаимодействие пользователя с программой

Выводы по разделу 1. Раздел «Постановка задачи» четко формулирует актуальность проблемы фильтрации спама в условиях роста цифровых коммуникаций и киберугроз в 2025 году, обосновывая необходимость разработки бесплатного приложения «SpamBayes» на основе наивного байесовского метода. Цель курсового проектирования заключается в закреплении, углублении и систематизации теоретических знаний и практических навыков, полученных в ходе изучения учебного предмета «Основы алгоритмизации и программирования», формировании навыков самостоятельного решения поставленных задач, а также в разработке приложения «SpamBayes», основной задачей которого является автоматическая классификация сообщений на «Спам» и «Не спам». Также был проведен анализ аналогов таких, как Mailwasher Free, Clean Email, ProtonMail, Spamihilator, Zoho Mail, что позволило выявить их ограничения в бесплатных версиях, подчеркивая уникальность разрабатываемого решения. Разработанное техническое задание детализирует функциональные требования, включая обучение классификатора, интеграцию с Gmail API и обеспечение надежности, а также минимальные системные требования, делая приложение доступным для широкого круга пользователей.

2 Проектирование программного обеспечения

## 2.1 Алгоритмы решения и математическая модель задачи

Теорема Байеса или формула Байеса – это одна из основных теорем элементарной [теории вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9), которая позволяет определить [вероятность](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C) события при условии, что произошло другое статистически [взаимозависимое](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) с ним событие. Другими словами, по формуле Байеса можно уточнить вероятность какого-либо события, взяв в расчёт как ранее известную информацию, так и данные новых наблюдений. Формула Байеса может быть выведена из основных аксиом теории вероятностей, в частности из [условной вероятности](https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Особенность теоремы Байеса заключается в том, что для её практического применения требуется большое количество расчётов, вычислений, поэтому байесовские оценки стали активно использовать только после революции в компьютерных и сетевых технологиях. На сегодняшний день активно применяется в машинном обучении и технологиях искусственного интеллекта.

При возникновении теоремы Байеса вероятности, используемые в теореме, подвергались целому ряду вероятностных интерпретаций. В одной из таких интерпретаций говорилось, что вывод формулы напрямую связан с применением особого подхода к статистическому анализу. Если использовать [байесовскую интерпретацию вероятности](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D0%B5%D1%81%D0%B8%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE), то теорема показывает, как личный уровень доверия может кардинально измениться вследствие количества наступивших событий. В этом заключаются выводы Байеса, которые стали основополагающими для байесовской статистики. Однако теорема не только используется в байесовском анализе, но и активно применяется для большого ряда других расчётов [6].

Фильтрация спама в приложении «SpamBayes» основана на использовании наивного байесовского классификатора, то есть статистического метода, который позволяет вычислить вероятность принадлежности текстового сообщения к категории «Спам» или «Не спам» на основе анализа его содержания. Этот подход предполагает обучение модели на предварительно размеченных данных (сообщениях из файлов «Spam.txt» и «Ham.txt») и последующую классификацию новых сообщений.

В основе алгоритма лежит теорема Байеса, которая выражается следующей формулой:

(1)

где

**P(A|B)** – апостериорная вероятность события **A** (например, что сообщение является спамом) при условии события **B** (набора слов в сообщении),

**P(B|A)** – правдоподобие, вероятность появления события **B** при условии **A** (вероятность встретить данный набор слов в спаме),

**P(A)** – априорная вероятность события **A** (вероятность того, что сообщение является спамом до анализа его содержания),

**P(B)** – полная вероятность события **B** (вероятность появления данного набора слов в любом сообщении).

Формула классификации спама.

Для задачи классификации спама переменная **A** принимает два значения: «Спам» (**S**) или «Не спам» (**H**).

Сообщение представляется как набор слов **w₁, w₂, ..., wₙ**.

Таким образом, вычисляются две вероятности:

– вероятность того, что сообщение является спамом,

– вероятность того, что сообщение не является спамом.

Сообщение классифицируется как «Спам», если:

(2)

в противном случае – как «Не спам».

Наивный байесовский классификатор предполагает, что все слова в сообщении независимы друг от друга:

(3)

Итоговая формула:

**(4)**

**(5)**

Знаменатель одинаков для обеих категорий, поэтому можно сравнивать только числители:

**и (6)**

Алгоритм решения задачи.

1. Подготовка данных (обучение модели):

* загрузка обучающих данных из файлов «Spam.txt» и «Ham.txt» а также извлечение текста из писем «Gmail»,
* токенизация текста (удаление знаков препинания, приведение к нижнему регистру).

Подсчет частот слов:

* частоты слов для категории «Спам»;
* частоты слов для категории «Не спам»;
* вычисление априорных вероятностей:

**(7)**

**(8)**

* Сохранение модели.

1. Классификация нового сообщения:

* токенизация сообщения;
* вычисление вероятностей для каждой категории с использованием логарифмов:

(9)

(10)

* сравнение логарифмических вероятностей.

1. Обновление модели:

* добавление нового сообщения в обучающую выборку;
* пересчет частот слов и априорных вероятностей;
* сохранение обновленной модели.

Для редких слов используется сглаживание Лапласа:

**(11)**

где **|V|** – это количество уникальных слов в обучающей выборке.

Для детального описания алгоритма классификации текстовых сообщений в приложении «SpamBayes» также разработана блок-схема, представленная на рисунке 2.1.1. Она иллюстрирует последовательность операций, выполняемых программой для фильтрации спама с использованием наивного байесовского метода.

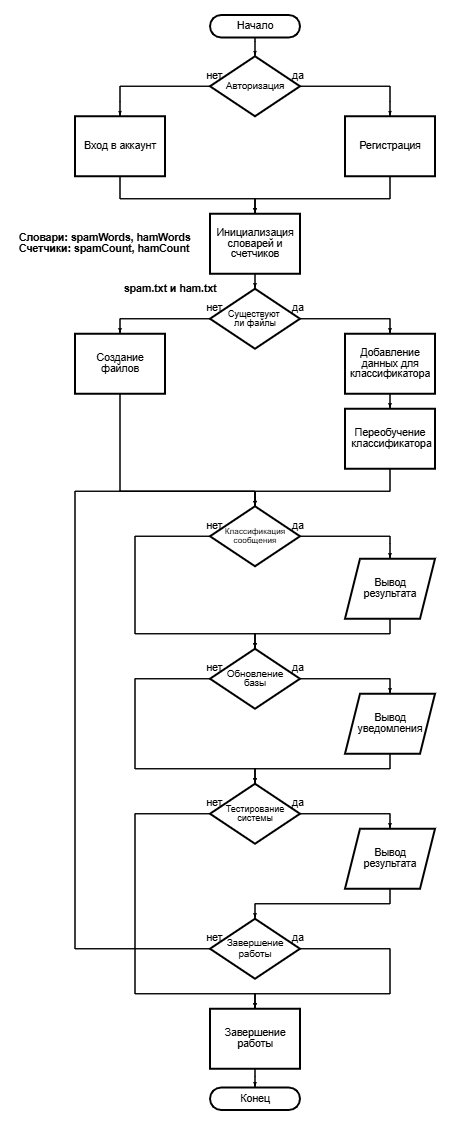


Рисунок 2.1.1 – Алгоритм классификации сообщений

## 2.2 Информационная база и описание данных

Приложение «SpamBayes» для автоматической фильтрации спама работает с текстовыми данными, которые используются для обучения наивного байесовского классификатора и последующей классификации сообщений на категории «Спам» и «Не спам». Информационная база приложения включает входные данные, внутренние структуры данных и выходные данные, которые описаны ниже.

Входные данные – текстовые файлы «spam.txt» и «ham.txt». Это простые текстовые файлы типа .txt в кодировке UTF-8, содержащие строки сообщений, каждое сообщение записано в отдельной строке. Файл «spam.txt» содержит примеры спама (например, «Win a million dollars!»), а файл «ham.txt» примеры полезных сообщений (например, «Hello, how are you?»). Файлы могут быть изначально пустыми, но для корректной работы классификатора требуется наличие хотя бы одного сообщения в каждом файле. Максимальный размер файла ограничен только доступным дисковым пространством. Данные загружаются из файлов, расположенных в директории приложения, или добавляются пользователем через графический интерфейс. Текст, введенный от пользователя в формате строки типа string через поля TextBox, определенные в XAML-интерфейсе WPF. TextBox1 используется для классификации, TextBox2 – для добавления в обучающую базу. рез поля TextBox, определенные в XAML-интерфейсе WPF. Поле TextBox1 используется для классификации, TextBox2 – для добавления в обучающую базу. Текст может быть произвольным, но поле ввода не должно быть пустым. Максимальная длина строки ограничена объемом оперативной памяти, но для практического использования предполагается, что длина текста не превышает нескольких сотен символов. Поля поддерживают привязку данных data binding для эффективного взаимодействия с моделью. А также данные в формате JSON, получаемые через Gmail API, содержат поля snippet или payload, которые извлекаются и передаются классификатору.

Внутренние структуры данных – используются словари частот слов типа dictionary<string, int>. Всего два словаря spamWords и hamWords, которые хранят в себе частоту появления каждого слова в сообщениях категорий «Спам» и «Не спам». Размер словарей ограничен объемом оперативной памяти, а слова короче 2 символов, могут игнорироваться для уменьшения шума. Есть счетчики сообщений типа int – это переменные spamCount и hamCount, которые хранят общее количество сообщений в категориях «Спам» и «Не спам», но значения не могут быть отрицательные, максимум определяется пределом типа int (2 147 483 647). Также используются списки слов типа list<string>, другими словами – список токенов, полученный в результате токенизации строки текста, то есть разделение по пробелам и знакам препинания с приведением к нижнему регистру. Пустые же элементы удаляются, а максимальный размер списка ограничен длиной входного текста и объемом памяти.

Выходные данные – это результат классификации, в формате строки типа string, содержащий текст «Spam» или «Not Spam», отображаемый в интерфейсе пользователя после классификации сообщения. Выводится может только одно из двух значений в зависимости от результата сравнения логарифмических вероятностей.

Операции с данными – пользователь вводит данные в поля TextBox1 и TextBox2 через графический интерфейс WPF, реализованный с использованием XAML и привязки данных. Ввод осуществляется вручную с клавиатуры или путем копирования текста. Текст проверяется на пустоту перед обработкой с отображением предупреждения через MessageBox в случае ошибки. Далее происходит конвертирование данных, то есть текст приводится к нижнему регистру и разбивается на слова (токены) с удалением знаков препинания (., ,, !, ? и др.). Новые сообщения, добавленные пользователем через интерфейс, записываются в конец соответствующего файла spam.txt или ham.txt с добавлением символа новой строки (\n), а при добавлении новых данных словари и счетчики очищаются и пересчитываются заново.

## 2.3 Моделирование системы

Для проектирования приложения «SpamBayes» было выполнено моделирование системы с использованием методологии IDEF0, что позволило описать функциональные процессы фильтрации спама и взаимодействия пользователя с приложением. Моделирование проведено на нескольких уровнях: контекстная диаграмма (уровень 0), декомпозиция основного процесса (уровень 1) и детализация подпроцессов (уровни 2-3), все уровни представлены на рисунках 2.3.1-2.3.8. Для каждого уровня приведены текстовые пояснения, где подробно описаны используемые данные, а также структура данных непосредственно описаная в этом разделе.



Рисунок 2.3.1 – Контекстная диаграмма IDEF0 уровня 0

На 0 уровне диаграммы представлена общая функция системы – «Фильтрация спама в текстовых сообщениях». Данные, используемые на этом уровне, подробно описаны в пункте 2.2 «Информационная база и описание данных». Входы включают текстовые строки типа string из файлов «spam.txt» и «ham.txt», также введенные пользователем через графический интерфейс и сообщения из Gmail.ru. Выходы включают строку типа string с результатом классификации «Spam» или «Not Spam», сообщения с результатами классификации отправленные на Gmail.ru, обновленные файлы «spam.txt» и «ham.txt», различного рода уведомления, и доступ к функционалу приложения «SpamBayes».

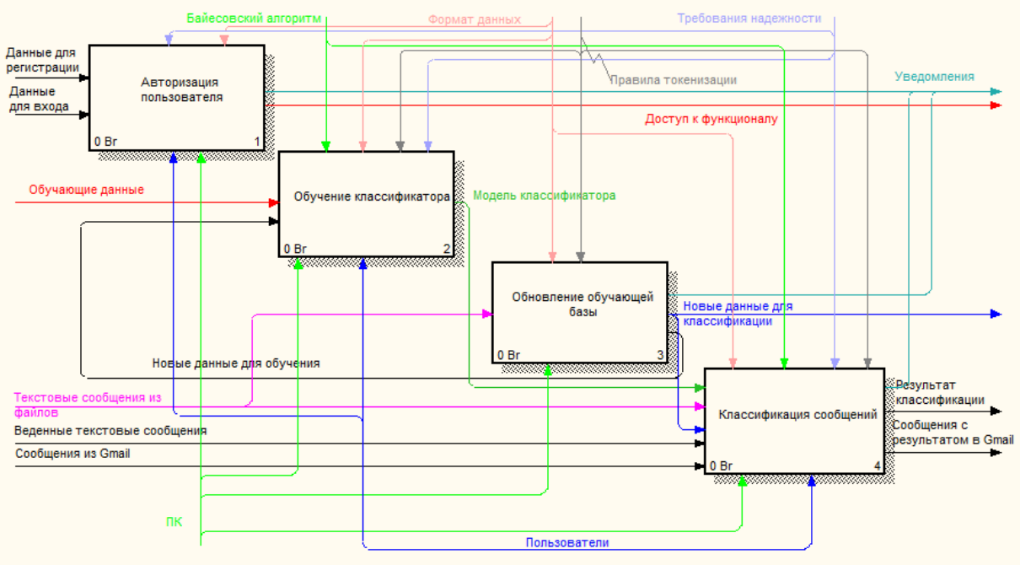


Рисунок 2.3.2 – Диаграмма IDEF0 уровня 1

На 1 уровне диаграммы процесс фильтрации спама разбивается на четыре основных блока. Входы для первого блока – это учетные данные, такие как логин, пароль и email, а входами для второго блока будет текстовая строка типа string, которая вводится через TextBox2 и происходит выбор категории. Входы для третьего блока – это также текстовая строка типа string, введенная пользователем через TextBox1, а входы для четвертого блока будут уже текстовые файлы «spam.txt» и «ham.txt», подробнее смотреть в пункте 2.2. Выходы для этих блоков будут следующие: для блока 1 – строка типа string уведомляющая пользователя и доступ к функционалу приложения; для блока 2 – это словари типа dictionary<string, int> и счетчики типа int; для блока 3 – это обновленные файлы с данными для классификации; для блока 4 – это строка типа string с результатом отображаемая в интерфейсе пользователя и сообщение с результатом отправленное на email пользователя.



Рисунок 2.3.3 – Диаграмма IDEF0 уровня 2 для блока «Авторизация»

На 2 уровне диаграммы происходит детализация первого блока «Авторизация пользователя». Первый блок разбивается на два подпроцесса такие, как «Регистрация пользователя» и «Вход в учетную запись». В первый блок подпроцесса будут входить учетные данные пользователя для регистрации (логин, пароль, email). Выходить переходя во второй блок подпроцесса будут данные для авторизации, а на выходе пользователь будет уведомлен и у него появиться полный доступ к функционалу приложения «SpamBayes».



Рисунок 2.3.4 – Диаграмма IDEF0 уровня 2 для блока «Обучение классификатора»

На 2 уровне диаграммы происходит детализация второго блока «Обучение классификатора». Второй блок разбивается на два подпроцесса – «Подготовка данных» и «Обучение модели». В первый блок подпроцесса будут входить обучающие данные и обновленные файлы в формате .txt, подробнее смотреть пункт 2.2. Обучающие данные представляют собой текстовые строки типа string, каждая из которых является сообщением, классифицированным как «Спам» или «Не спам». Новые данные для обучения поступают из блока «Обновление обучающей базы» после добавления новых данных. Выходить, переходя в следующий блок будут строки текста типа string, очищенные от лишних символов и готовые для токенизации, а на выходе будет модель классификатора, включающая в себя словари типа dictionary<string, int> для хранения частот слов, spamWords для спама, hamWords для не-спама и счетчики типа int для числа сообщений spamCount, hamCount, подробнее смотреть пункт 2.2.

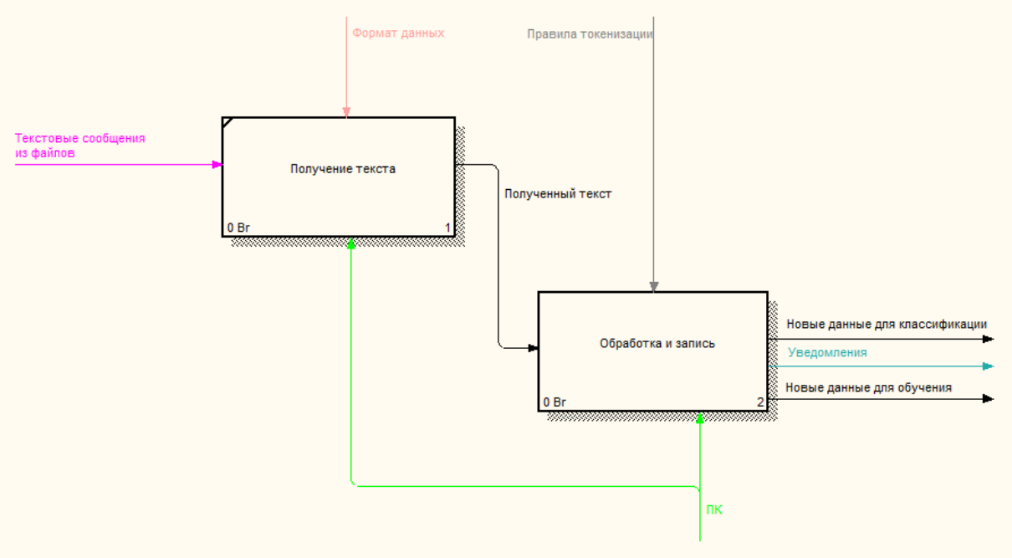


Рисунок 2.3.5 – IDEF0 уровня 2 для блока «Обновление обучающей базы»

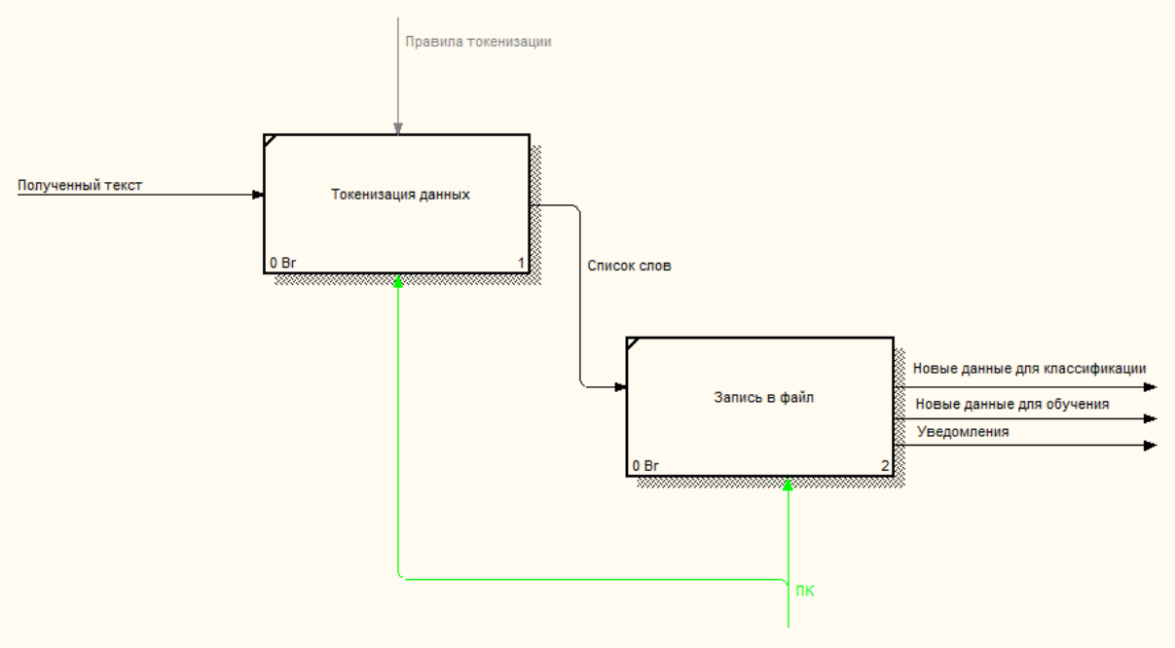


Рисунок 2.3.6 – Диаграмма IDEF0 уровня 3 для блока «Обработка и запись»

Блок «Обновление обучающей базы» разбит на два подпроцесса – это «Получение текста» и «Обработка и запись». В первый блок будет входить текстовая строка типа string, введенная пользователем через TextBox2 и выходить входя в следующий блок текстовая строка типа string после её проверки на пустоту, также в следующий блок будет входить выбор категории, это переменная типа string, принимающая значения «Спам» или «Не спам», и на выходе будут получены обновленные файлы «spam.txt» или «ham.txt», подробнее смотреть пункт 2.2 и уведомления. Но так, как блок «Обработка и запись» разбивается ещё на два подпроцесса таких, как «Токенизация данных» и «Запись в файл», то из второго блока будут выходить уже конкретно обновленные файлы, которые затем используются для переобучения классификатора.

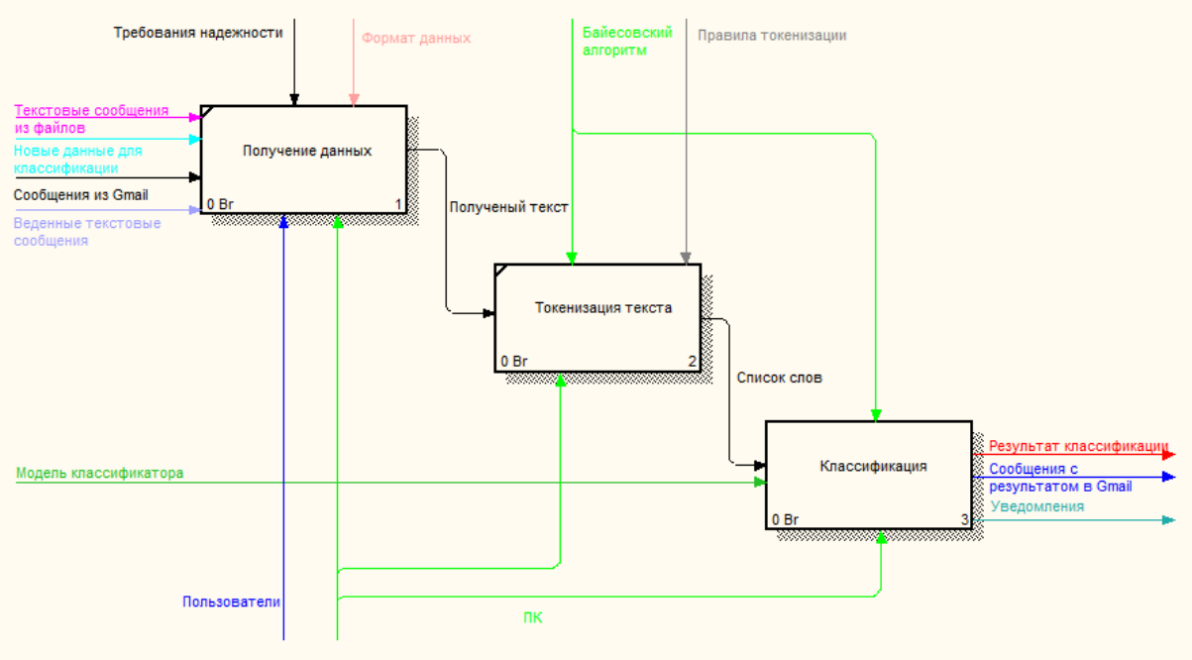


Рисунок 2.3.7 – Диаграмма IDEF0 уровня 2 для блока «Класификация сообщений»

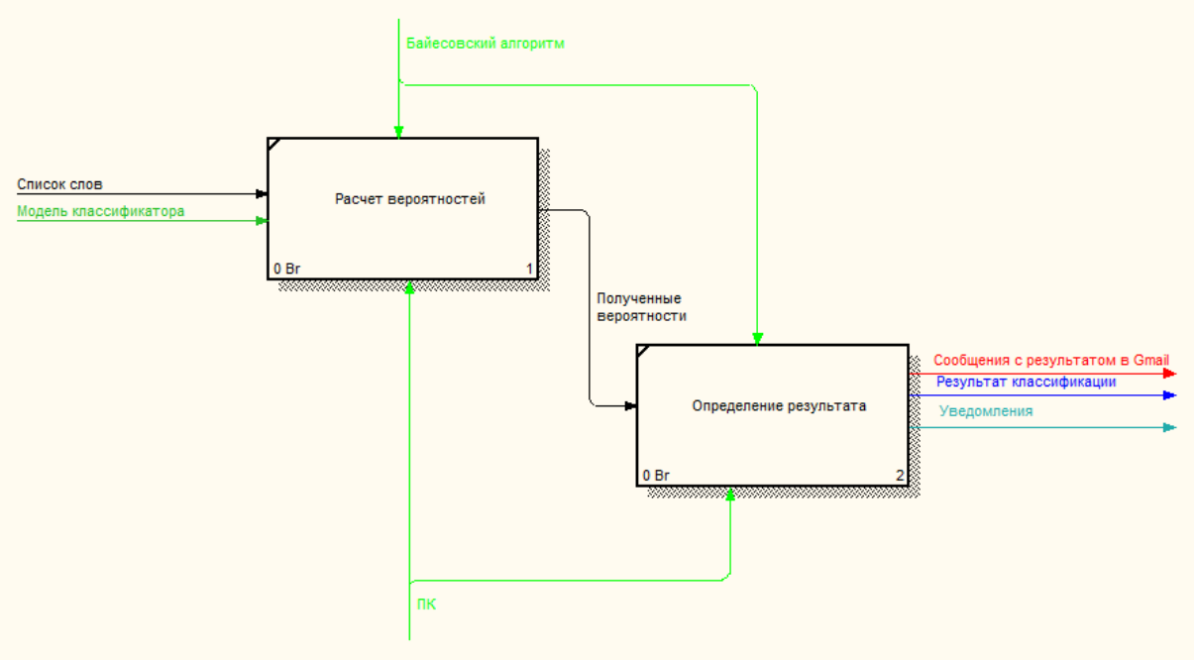


Рисунок 2.3.8 – Диаграмма IDEF0 уровня 3 для блока «Классификация»

Блок «Классификация сообщений» разбивается на три подпроцесса – это «Прием текста», «Токенизация текста» и «Классификация». В блок первого подпроцесса будут входить текстовые строки типа string, введенные пользователем через TextBox1, полученные из файлов «spam.txt» и «ham.txt», а также полученные сообщения из Gmail.ru и выходить, переходя в следующий блок также текстовые строки типа string, только уже после проверки на пустоту, соответственно из второго блока будет выходить, переходя в третий блок список слов типа list<string> после токенизации. Блок «Классификация» разбивается ещё на два подпроцесса такие, как «Расчет вероятностей» и «Определение результата». В третий блок будет входить модель классификатора, это словари spamWords и hamWords типа dictionary<string, int>, а также счетчики spamCount и hamCount типа int, подробнее смотреть пункт 2.2, а на выходе пользователь получит текстовую строку типа string с результатом классификации «Spam» или «Not Spam», уведомление и сообщение с результатом на Gmail.ru.

## 2.4 Выбор программного обеспечения и технологий

Используется интегрированная среда разработки Visual Studio 2022 (версия 17.8.3). Так как бесплатная версия Visual Studio Community предоставляет полный набор инструментов для разработки приложений под Windows, отвечающих требованиям к бесплатному процессу разработки [15]. Поддерживает язык программирования C#, включающий в себя современные библиотеки и фреймворки, такие как .NET Framework 4.7, необходимый для работы приложения, также включает надежную поддержку разработки WPF, а также средства конструктора XAML и IntelliSense для создания современных адаптивных пользовательских интерфейсов. Присутствует подсветка синтаксиса, автодополнение кода и интеграция с системой контроля версий ускоряющие процесс написания и отладки кода. А также активное сообщество и обширная документация на русском и английском языках, которые обеспечивают доступ к справкам и примерам.

C# был выбран в качестве языка программирования из-за его высокоуровневого синтаксиса и надежной интеграции с экосистемой .NET, что упрощает реализацию байесовского алгоритма и работы с текстовыми данными. Его совместимость с .NET Framework обеспечивает стабильное выполнение на Windows 7 и более поздних версиях, удовлетворяя системным требованиям, изложенным в разделе 1.3, пункте 9. А также присутствуют встроенные библиотеки для работы с файлами, строками, интеграцией с Gmail, которые помогут сократить время разработки.

В качестве среды выполнения для приложения была выбрана платформа .NET Framework 4.7, ведь она обеспечивает совместимость с большинством современных систем Windows без необходимости установки дополнительных компонентов. Поддерживает Windows Presentation Foundation (WPF) для разработки современного гибкого пользовательского интерфейса с использованием XAML, как указано в разделе 1.3, пункте 3.

Средство моделирования – AllFusion Process Modeler, мощный инструмент для создания диаграмм IDEF0 и других моделей процессов, что соответствует требованиям к детальному моделированию системы фильтрации спама [2]. Существует возможность построения многоуровневых диаграмм, что является важным аспектом для описания сложных алгоритмических процессов, таких как байесовская классификация. А также есть поддержка экспорта диаграмм в различные форматы для включения в документацию проекта.

Выбор бесплатного ПО обусловлен стремлением минимизировать затраты на разработку и сделать проект доступным для пользователей без дополнительных лицензионных расходов. Таким образом, выбранные инструменты полностью соответствуют задачам курсового проекта «SpamBayes», обеспечивая удобство разработки, совместимость с целевой платформой и высокую производительность при реализации фильтрации спама.

Выводы по разделу 2. Раздел «Проектирование программного обеспечения» представляет собой комплексный подход к проектированию приложения «SpamBayes», включая математическую модель на основе теоремы Байеса с использованием сглаживания Лапласа и логарифмических вероятностей для устойчивости классификатора. Информационная база – это входные данные из файлов и Gmail API, словари частот слов, выходные результаты, структурирована рационально. Моделирование по методологии IDEF0 диаграммы уровней 0-3, наглядно описывает процессы обучения, классификации и обновления базы, обеспечивая ясность проектирования. Выбор технологий C#, .NET Framework 4.7, WPF, Visual Studio 2022, AllFusion Process Modeler обоснован их доступностью, совместимостью и поддержкой современных интерфейсов, что гарантирует эффективную реализацию приложения.

3 Разработка программного обеспечения

## 3.1 Реализация интерфейса программы

Разработка интерфейса пользователя для приложения «SpamBayes» проводилась с учетом удобства взаимодействия, минимизации лишних действий и обеспечения интуитивно понятного управления функциями фильтрации спама. Проектирование интерфейса началось с создания wireframe-моделей на основе сценариев использования use case, описанных в техническом задании раздел 1.3, рисунки 1.3.1-4. Эти модели позволили определить оптимальное расположение элементов управления и количество форм, необходимых для реализации всех функций программы.

Wireframe-модели были разработаны с учетом принципов минимализма и функциональности. Элементы управления размещены так, чтобы пользователь мог быстро переходить между ключевыми действиями: классификацией текста, добавлением данных и выходом из программы. Например, на главном экране кнопки классификации и добавления данных расположены рядом с соответствующими полями ввода, что сокращает время навигации.

Выбор дизайна интерфейса обусловлен такими факторами, как простота и доступность, ведь минималистичный стиль с четкими надписями и логичным расположением элементов делает приложение удобным для пользователей с разным уровнем технической подготовки. Функциональная ориентация, то есть каждый элемент интерфейса привязан к конкретной задаче, это может быть ввод текста, классификация или обучение, что соответствует требованиям технического задания в разделе 1.3, пункте 3. А также визуальная ясность в использовании контрастных цветов и выделение кнопок помогает пользователю быстро ориентироваться в интерфейсе, таким образом цветовая палитра выбрана нейтральной, чтобы избежать отвлечения внимания от основной задачи.

Для реализации интерфейса использовалась среда разработки Visual Studio 2022 с фреймворком .NET Framework 4.7 и библиотекой Windows Presentation Foundation (WPF). Выбор обоснован тем, что WPF поддерживает современные, масштабируемые интерфейсы с использованием XAML для декларативного дизайна, что ускоряет разработку и соответствует требованиям локального приложения для Windows. А также C# тесно интегрирует с .NET и WPF, что упрощает обработку событий интерфейса, привязку данных и взаимодействие с байесовским классификатором [16].

Основные компоненты для управления приложением представлены ниже.

1. Button – использован для запуска действий, таких как классификация, добавление данных, выход, с поддержкой стилизации через XAML;
2. TextBox – применен для ввода текста пользователем, с поддержкой многострочного ввода и привязки данных;
3. CheckBox – реализован на формах входа и регистрации для управления видимостью пароля, повышая безопасность и удобство;
4. TextBlock – использован для отображения описаний и результатов классификации с поддержкой форматирования;
5. ContentControl – использован для динамического вывода информации о приложении.

Иконка приложения представляет собой стилизованный символ фильтра, что отражает основную функцию программы – фильтрацию спама. Цвет иконки символизирует надежность и безопасность. Логотип «SpamBayes» включает название программы, что усиливает связь с байесовским методом.

## 3.2 Особенности реализации задачи

Реализация приложения «SpamBayes» была сосредоточена на разработке наивного байесовского классификатора для фильтрации спама, интегрированного с современным интерфейсом на основе Windows Presentation Foundation (WPF) и Gmail API для обработки реальных электронных писем. Ниже приведены ключевые особенности реализации, подчеркивающие конкретные технические решения и оптимизации, выполненные в процессе разработки.

Классификатор реализован на языке C# с использованием словарей типа dictionary<string, int> для хранения частот слов в сообщениях спама «spamWords» и не-спама «hamWords», как описано в разделе 2.2. Применялось сглаживание Лапласа для обработки редких слов и предотвращения нулевых вероятностей, смотреть формулу 11 в разделе 2.1, обеспечивая надежные вычисления вероятностей [1].

Логарифмические вероятности использовались для избежания числового переполнения при умножении малых вероятностей, как указано в разделе 2.1, формулы 9 и 10. Этот подход повысил стабильность классификатора для длинных сообщений.

Предобработка текста включала токенизацию текста из файлов spam.txt и ham.txt, приведение к нижнему регистру и удаление знаков препинания с использованием регулярных выражений Regex.Split, что уменьшило шум и обеспечило единообразное представление слов.

Приложение поддерживает динамическое обновление обучающей выборки. Пользователи могут добавлять новые сообщения в файлы spam.txt или ham.txt через интерфейс WPF, что инициирует процесс переобучения. Словари и счетчики очищаются и пересчитываются только при добавлении новых данных, оптимизируя производительность.

Операции ввода-вывода файлов реализованы с использованием классов System.IO с обработкой исключений для управления такими сценариями, как отсутствие файлов или недостаток прав доступа, обеспечивая надежность, что советует требованиям в разделе 1.3, пункте 8.

Интеграция с Gmail достигнута с использованием библиотеки Google.Apis.Gmail.v1, обеспечивающей аутентификацию OAuth 2.0 и получение писем. Содержимое писем (например, поля snippet или payload) извлекается и передается классификатору для обнаружения спама. Для обработки больших наборов писем письма обрабатываются инкрементально, извлекая ограниченное количество сообщений за запрос, чтобы минимизировать использование квоты API и повысить производительность. Обработка ошибок вызовов Gmail API включает проверки на недействительные учетные данные, проблемы с сетью и ограничения по количеству запросов, с отображением понятных сообщений об ошибках через MessageBox WPF [10].

Пользовательский интерфейс построен с использованием WPF с XAML для декларативного дизайна и C# для обработки событий. Ключевые элементы интерфейса (например, Button, TextBox, CheckBox) привязаны к модели представления с использованием принципов MVVM (Model-View-ViewModel), обеспечивая четкое разделение интерфейса и логики. Привязка данных использовалась для динамического обновления результата классификации в элементе TextBlock, улучшая отзывчивость и пользовательский опыт.

Интерфейс использует возможности стилизации и шаблонов WPF для создания современного, масштабируемого дизайна с адаптивными макетами для различных разрешений экрана, как требуется в разделе 1.3, пункте 3.

Специальная функция токенизации разделяет текст на слова, удаляя знаки препинания и приводя к нижнему регистру с использованием комбинации разделения строк и регулярных выражений. Часто встречающиеся слова кэшируются в памяти во время обучения, чтобы уменьшить избыточную обработку.

Привязка данных WPF обеспечивает эффективную передачу токенизированного текста между интерфейсом и классификатором, минимизируя задержки.

Словари, хранящиеся в памяти, уменьшают нагрузку на операции ввода-вывода диска во время классификации, особенно при обработке нескольких сообщений в одной сессии.

Ленивая загрузка обучающих данных выполняется только при инициализации классификатора или обновлениях, минимизируя время запуска.

Оптимизации рендеринга WPF, такие как виртуализация для отображения больших текстов, обеспечивают плавную работу интерфейса даже при обработке длинных сообщений.

Некорректные входные данные (например, пустые поля TextBox) вызывают отображение MessageBox WPF с предупреждением «Пожалуйста, введите сообщение», предотвращая сбои, что соответствует требованиям в разделе 1.3, пункте 8.

Ошибки, связанные с файлами (например, отсутствие spam.txt), обрабатываются с помощью блоков try-catch, создавая пустые файлы при необходимости, что обеспечивает непрерывную работу.

## 3.3 Обеспечение отказоустойчивости

Приложение «SpamBayes» предназначено для классификации текстовых сообщений на «Спам» и «Не спам» с использованием байесовского алгоритма. Для обеспечения высокой отказоустойчивости и защиты пользовательских данных в приложении реализован комплекс мер, включающий валидацию пользовательского ввода, механизмы авторизации, защиту от внешних атак и надежное управление файловыми данными. Эти меры направлены на повышение стабильности работы приложения, минимизацию ошибок и обеспечение безопасности данных в соответствии с требованиями, описанными в разделе 1.3, пункте 8.

Для предотвращения ошибок, вызванных некорректными данными, в проекте SpamBayes реализована строгая валидация пользовательского ввода. Валидация охватывает ключевые интерфейсы приложения, такие как регистрация, авторизация и ввод текстов для классификации сообщений. Этот подход позволяет исключить некорректные операции, которые могут привести к сбоям или нарушению целостности данных. Условия валидации описаны далее.

Регистрация пользователя:

* проверяется заполненность всех обязательных полей: имя пользователя, email и пароль. Пустые поля, пробелы или значения, состоящие только из пробельных символов, недопустимы;
* имя пользователя должно быть уникальным в системе. Для этого перед сохранением данных выполняется проверка на наличие дубликатов в файле users.json;
* email проверяется на соответствие стандартному формату (например, наличие символа @ и доменной части);

При нарушении любого из условий пользователю отображается информативное сообщение об ошибке. Это предотвращает создание некорректных учетных записей и улучшает пользовательский опыт.

Авторизация:

* проверяется соответствие введенных имени пользователя и пароля данным, хранящимся в файле users.json;
* при вводе неверных учетных данных пользователю выводится сообщение об ошибке;

Этот механизм исключает несанкционированный доступ к системе и обеспечивает надежную идентификацию пользователей.

Классификация сообщений:

* при добавлении текста в обучающие файлы (spam.txt или ham.txt) проверяется, что сообщение не является пустым или состоящим только из пробельных символов.

Внешняя авторизация через Google:

* при использовании OAuth 2.0 для авторизации через Google проверяется корректность введенных кодов авторизации, полученных от сервера провайдера;
* валидация включает проверку формата токена и его срока действия, что минимизирует риск использования поддельных или устаревших данных;
* все этапы взаимодействия с внешним сервисом логируются для диагностики возможных ошибок.

Валидация пользовательского ввода реализована с учетом принципов надежности и удобства. Она обеспечивает защиту от ошибок, вызванных некорректными данными, и повышает устойчивость приложения к нестандартным сценариям использования.

Для обеспечения безопасности пользовательских данных и предотвращения несанкционированного доступа в проекте SpamBayes также реализованы несколько уровней защиты. Эти меры включают как локальные, так и внешние механизмы авторизации, а также защиту от распространенных угроз, таких как утечка данных или атаки.

Локальная авторизация:

* пользователи проходят проверку по имени пользователя и паролю, данные которых хранятся в локальном файле users.json;

Локальная авторизация обеспечивает базовый уровень контроля доступа и является основным способом аутентификации для пользователей, не использующих внешние сервисы.

Внешняя авторизация через Google:

* реализована интеграция с Google через протокол OAuth 2.0, который позволяет пользователям аутентифицироваться без передачи пароля приложению;
* процесс авторизации включает перенаправление пользователя на сервер Google, где он подтверждает свою личность. После успешной аутентификации приложение получает защищённый токен, содержащий данные, такие как имя и email;
* использование OAuth 2.0 минимизирует риск компрометации учетных данных, так как приложение не хранит пароли внешних сервисов, а токены имеют ограниченный срок действия;
* все данные, передаваемые между приложением и сервером Google, защищены шифрованием TLS, что предотвращает перехват информации;
* для повышения надежности токены регулярно обновляются, а их использование логируется для отслеживания подозрительной активности.

Хранение пользовательских данных:

* данные пользователей, такие как имя, email и хешированные пароли, хранятся в файле users.json, структура которого подробно описана в разделе 2.2;
* для предотвращения повреждения данных при сбоях файл регулярно резервируется, а перед записью новых данных создается временная копия.

Для обеспечения стабильной работы приложения SpamBayes также реализованы механизмы управления файловыми данными, которые минимизируют риски, связанные с отсутствием файлов, повреждением данных или сбоями файловой системы.

Проверка файлов:

* перед использованием обучающих файлов spam.txt и ham.txt приложение проверяет их наличие в каталоге;
* если файлы отсутствуют, генерируется контролируемая ошибка, а пользователю выводится сообщение с предложением создать новые файлы или загрузить существующие.

Эта проверка предотвращает аварийное завершение программы и обеспечивает корректную инициализацию данных.

Все операции с файлами, включая чтение и запись, сопровождаются обработкой исключений. При возникновении ошибок, таких как отсутствие прав доступа, повреждение файлов или сбой файловой системы, приложение перехватывает исключение и выполняет соответствующие действия, такие как повторная попытка операции или уведомление пользователя.

Инициализация данных:

* при первом запуске приложения или в случае отсутствия файла users.json автоматически создается пустая структура данных с корректным JSON-форматом;
* аналогичный механизм реализован для файлов spam.txt и ham.txt, что исключает ошибки, связанные с отсутствием обучающих данных;
* инициализация данных выполняется с учетом принципов отказоустойчивого программирования, что гарантировать стабильность приложения даже в нестандартных условиях.

Дополнительные меры защиты:

* проект использует собственные файлы данных (users.json, spam.txt, ham.txt) с нестандартной структурой, что затрудняет их использование без знания внутренней логики приложения;
* для дополнительной защиты от внешних угроз, таких как инъекции или переполнение буфера, приложение использует безопасные библиотеки и методы обработки данных, такие как json.loads и json.dumps для работы с JSON, а также экранирование входных данных.

Выводы по разделу 3. Раздел «Разработка программного обеспечения» описывает успешную реализацию приложения «SpamBayes» с акцентом на интуитивно понятный WPF-интерфейс, разработанный по принципам MVVM с минималистичным дизайном и адаптивными макетами. Наивный байесовский классификатор, реализованный на языке программирования C#, эффективно обрабатывает текст с помощью токенизации, сглаживания Лапласа и оптимизаций, таких как кэширование слов и ленивая загрузка данных. Интеграция с Gmail API через OAuth 2.0 обеспечивает классификацию реальных писем, а обработка исключений файловых и сетевых ошибок совместно с валидацией ввода проверяют наличие пустых полей и уникальность имени пользователя, что гарантирует отказоустойчивость [12]. При этом хранение данных в users.json, защита через локальную и внешнюю авторизацию повышают безопасность приложения.

4 Отладка и подготовка к внедрению

## 4.1 Контрольные примеры

1. Регистрация пользователя (RegisterWindow).

Тест 1.1: Успешная регистрация пользователя.

Тестируется функциональность регистрации нового пользователя с корректными данными в окне RegisterWindow. Цель данного теста заключается в том, чтобы убедиться, что пользователь может успешно зарегистрироваться с уникальным именем, валидным email и паролем, а данные корректно сохраняются в файл users.json.

Условия тестирования, следующие: приложение запускается на системе Windows с установленным .NET Framework, сам файл users.json доступен для чтения и записи, а пользователя с именем admin не существует в системе.

Тестирование проводиться ручное, через графический интерфейс окна RegisterWindow, присутствует проводник для проверки содержимого файла users.json и отладчик Visual Studio для анализа работы класса UserManager. Само ручное тестирование через GUI и проверка файла обеспечивают высокую достоверность результата.

Для выполнения тестирования первым делом запускается приложение, после чего необходимо перейти в окно RegisterWindow, при помощи нажатия кнопки «Регистрация» в MainEntryWindow. Далее вводиться admin в поле имени пользователя, sarrikzhdyn@gmail.com в поле email и 123 в поле пароля и нажимается кнопка «Зарегистрироваться». После этого необходимо убедиться, что отображается уведомление об успехе «Регистрация успешна! Теперь вы можете войти.»

Ожидаемые результаты: приложение отображает уведомление об успешной регистрации. В файле users.json появляется запись: {"Username": "admin", "Password": "123", "Email": "sarrikzhdyn@gmail.com"}. Пользователь перенаправляется в окно LoginWindow.

Тест 1.2: Регистрация с уже существующим именем пользователя.

Тестируется обработка попытки зарегистрировать пользователя с уже существующим именем. Цель данного теста заключается в том, чтобы проверить, что приложение предотвращает регистрацию с дублирующимся именем пользователя и отображает соответствующее сообщение об ошибке.

Условия тестирования, следующие: пользователь с именем admin уже зарегистрирован в users.json, файл users.json доступен для чтения и записи.

Тестирование проводиться ручное, через графический интерфейс, присутствует проводник для проверки содержимого users.json. Само ручное тестирование и проверка файла обеспечивают точные результаты.

Для выполнения тестирования необходимо перейти в окно RegisterWindow, ввести admin в поле имени пользователя, testuser@gmail.com в поле email и newpassword в поле пароля и нажать кнопку «Зарегистрироваться». После чего проверить отображаемое сообщение.

Ожидаемые результаты: приложение отображает уведомление «Пользователь с таким именем уже существует». Новая запись в users.json не добавляется, и пользователь остается в окне RegisterWindow.

Тест 1.3: Регистрация с пустыми полями.

Тестируется обработка попытки регистрации с пустыми или некорректными полями. Цель данного теста заключается в том, чтобы убедиться, что приложение требует заполнения всех полей и отображает сообщение об ошибке при их отсутствии.

Условия тестирования, следующие: файл users.json доступен для чтения и записи, а поля ввода пустые или содержат значения по умолчанию (подсказки).

Тестирование проводиться ручное, через GUI. Само ручное тестирование надежно для проверки поведения интерфейса.

Для выполнения тестирования необходимо перейти в окно RegisterWindow. Далее оставить поля имени пользователя, email и пароля пустыми или с текстом подсказки «Введите имя пользователя», «Введите email» и нажать кнопку «Зарегистрироваться». После чего проверить отображаемое сообщение.

Ожидаемые результаты: отображается уведомление «Пожалуйста, заполните все поля». Новая запись в users.json не создается, и пользователь остается в окне RegisterWindow.

2. Вход в систему (LoginWindow).

Тест 2.1: Успешный вход с корректными данными.

Тестируется функциональность входа с правильным именем пользователя и паролем в окне LoginWindow. Цель данного тестирования в том, чтобы проверить, что пользователь может войти в систему с валидными учетными данными и перейти в окно MainWindow.

Условия тестирования, следующие: пользователь с именем admin, паролем 123 и email sarrikzhdyn@gmail.com зарегистрирован в users.json, приложение запущено, доступ к файлу users.json есть.

Тестирование проводиться ручное через GUI. Самого ручного тестирования достаточно для проверки поведения интерфейса.

Для выполнения тестирования необходимо перейти в окно LoginWindow, нажав на кнопку «Вход» в MainEntryWindow. Далее ввести admin в поле имени пользователя и 123 в поле пароля и нажать кнопку «Авторизоваться». После чего проверить, открылось ли окно MainWindow.

Ожидаемые результаты: пользователь перенаправляется в окно MainWindow, а окно LoginWindow закрывается.

Тест 2.2: Вход с неверным паролем.

Тестируется функциональность обработки попытки входа с неверным паролем в окне LoginWindow. Цель данного тестирования в том, чтобы проверить, что приложение отклоняет вход с неверными учетными данными, отображает сообщение об ошибке и не перенаправляет пользователя в окно MainWindow.

Условия тестирования, следующие: пользователь с именем admin, паролем 12345 и email sarrikzhdyn@gmail.com зарегистрирован в users.json, приложение запущено, доступ к файлу users.json есть.

Тестирование проводится ручное через GUI. Самого ручного тестирования достаточно для проверки поведения интерфейса.

Для выполнения тестирования необходимо перейти в окно LoginWindow, нажав на кнопку «Вход» в MainEntryWindow. Далее ввести admin в поле имени пользователя и 12345 в поле пароля и нажать кнопку «Авторизоваться». После чего проверить отображаемое сообщение и состояние окна.

Ожидаемые результаты: отображается уведомление «Неверный логин или пароль», окно LoginWindow остается открытым, перенаправление в MainWindow не происходит.

Тест 2.3: Вход через Google.

Тестируется функциональность входа через Google OAuth в окне LoginWindow. Цель данного тестирования в том, чтобы проверить, что пользователь может успешно войти через Google, данные корректно сохраняются или используются, и происходит перенаправление в окно MainWindow.

Условия тестирования, следующие: приложение запущено, доступ к интернету есть, корректные clientId и clientSecret для Google OAuth настроены, браузер установлен и работает, доступ к файлу users.json есть.

Тестирование проводится ручное через GUI с использованием браузера для аутентификации и проводника для проверки users.json. Самого ручного тестирования достаточно для проверки поведения интерфейса, хотя стабильность зависит от Google API.

Для выполнения тестирования необходимо перейти LoginWindow, нажав на кнопку «Вход» в MainEntryWindow. Далее нажать кнопку «Google», в открывшемся браузере выполнить вход в аккаунт Google и подтвердить доступ для приложения. После чего проверить, открылось ли окно MainWindow, и, если пользователь новый, проверить users.json на наличие новой записи.

Ожидаемые результаты: пользователь перенаправляется в окно MainWindow, окно LoginWindow закрывается. Если пользователь новый, в users.json добавляется запись с именем и email без пароля.

3. Классификация спама (SpamClassifier).

Тест 3.1: Классификация спам-сообщения.

Тестируется функциональность функции Classify класса SpamClassifier на спам-сообщении в окне MainWindow. Цель данного тестирования в том, чтобы проверить, что алгоритм корректно классифицирует сообщение как спам на основе тренировочных данных и отправляет результат на email пользователя.

Условия тестирования, следующие: файлы spam.txt и ham.txt содержат тренировочные данные (например, spam.txt включает фразу «You win a million dollars!», а ham.txt «Hello, how are you?»), приложение запущено, пользователь вошел в систему как admin с email sarrikzhdyn@gmail.com.

Тестирование проводится ручное через GUI. Самого ручного тестирования достаточно для проверки поведения интерфейса, хотя результат зависит от качества тренировочных данных, но алгоритм Байеса обеспечивает надежность за счет сглаживания вероятностей.

Для выполнения тестирования необходимо войти в систему как раннее зарегистрированный пользователь, нажав на кнопку «Вход» в MainEntryWindow и введя корректные учетные данные. В окне MainWindow ввести в поле InputTextBox текст «You win a million dollars!» и нажимаем кнопку «Классифицировать». После чего проверить текст в ResultLabel и подтвердить отправку результата на email.

Ожидаемые результаты: в ResultLabel отображается «Результат: Спам», результат классификации отправляется на email пользователя, окно MainWindow остается открытым.

Тест 3.2: Классификация не спама.

Тестируется функциональность функции Classify класса SpamClassifier на не спам-сообщении в окне MainWindow. Цель данного тестирования в том, чтобы проверить, что алгоритм корректно классифицирует сообщение как не спам на основе тренировочных данных и отправляет результат на email пользователя.

Условия тестирования, следующие: файлы spam.txt и ham.txt содержат тренировочные данные (например, spam.txt включает «You win a million dollars!», а ham.txt «Hello, how are you?»), приложение запущено, пользователь вошел в систему как admin с email sarrikzhdyn@gmail.com.

Тестирование проводится ручное через GUI. Самого ручного тестирования достаточно для проверки поведения интерфейса, хотя надежность зависит от качества тренировочных данных, но алгоритм Байеса устойчив благодаря сглаживанию вероятностей.

Для выполнения тестирования необходимо войти в систему как раннее зарегистрированный пользователь, нажав на кнопку «Вход» в MainEntryWindow и введя корректные учетные данные. В окне MainWindow ввести в поле InputTextBox текст «Hello, how are you?» и нажать кнопку «Классифицировать». После чего проверить текст в ResultLabel и подтвердить отправку результата на email пользователя.

Ожидаемые результаты: в ResultLabel отображается «Результат: Не спам», результат классификации отправляется на email пользователя, окно MainWindow остается открытым.

Тест 3.3: Добавление спам-сообщения для обучения.

Тестируется функциональность функции AddTextToFile для добавления спам-сообщения в файл spam.txt. Цель данного тестирования в том, чтобы проверить, что новое спам-сообщение корректно добавляется в spam.txt и классификатор переобучается.

Условия тестирования, следующие: файл spam.txt доступен для записи, приложение запущено, пользователь вошел в систему как admin с email sarrikzhdyn@gmail.com.

Тестирование проводится ручное через GUI с использованием проводника для проверки содержимого spam.txt. Само ручное тестирование обеспечивает высокую надежность, так как проверяется добавление текста в файл.

Для выполнения тестирования необходимо войти в систему как раннее зарегистрированный пользователь, нажав на кнопку «Вход» в MainEntryWindow и введя корректные учетные данные. В окне MainWindow ввести в поле TrainTextBox текст «How to lose weight fast» и нажать кнопку «В спам». После чего проверить уведомление в TrainResultLabel и содержимое файла spam.txt.

Ожидаемые результаты: в TrainResultLabel отображается уведомление «Сообщение успешно добавлено!», в файл spam.txt добавлена строка «How to lose weight fast», классификатор переобучен, проверяется в последующих тестах классификации.

4. Получение и отправка писем (MainWindow).

Тест 4.1: Получение последнего письма.

Тестируется функциональность функции FetchLastEmail для получения последнего письма через IMAP в окне MainWindow. Цель данного тестирования в том, чтобы проверить, что приложение корректно подключается к почтовому серверу, извлекает текст последнего письма и классифицирует его.

Условия тестирования, следующие: приложение запущено, есть доступ к интернету, настроены корректные учетные данные IMAP sarrikzhdyn@gmail.com, пароль приложения, в почтовом ящике есть хотя бы одно письмо.

Тестирование проводится ручное через GUI с использованием почтового клиента для проверки содержимого ящика. Самого ручного тестирование достаточно для проверки, хотя надежность зависит от стабильности IMAP-соединения, но библиотека MailKit обеспечивает стабильность.

Для выполнения тестирования необходимо войти в систему как раннее зарегистрированный пользователь, нажав на кнопку «Вход» в MainEntryWindow и введя корректные учетные данные. В окне MainWindow нажать кнопку «Получить последнее письмо». После чего проверить содержимое поля InputTextBox и текст в ResultLabel.

Ожидаемые результаты: текст последнего письма отображается в InputTextBox, в ResultLabel показан результат классификации письма, результат отправляется на email пользователя.

Тест 4.2: Отправка результата классификации на email.

Тестируется функциональность функции SendEmail для отправки результатов классификации на email пользователя. Цель данного тестирования в том, чтобы проверить, что приложение корректно отправляет письмо с результатом классификации на адрес пользователя.

Условия тестирования, следующие: приложение запущено, есть доступ к интернету, настроены корректные учетные данные SMTP sarrikzhdyn@gmail.com, пароль приложения.

Тестирование проводится ручное через GUI с использованием почтового клиента для проверки полученного письма. Самого ручного тестирования достаточно для проверки, а библиотека MailKit обеспечивает надежность SMTP-соединения.

Для выполнения тестирования необходимо войти в систему как раннее зарегистрированный пользователь, нажав на кнопку «Вход» в MainEntryWindow и введя корректные учетные данные. В окне MainWindow ввести произвольный текст в поле InputTextBox, нажать кнопку «Классифицировать» и проверить, получено ли письмо на email пользователя.

Ожидаемые результаты: письмо доставлено на email пользователя с темой «Результат классификации сообщения», тело письма содержит текст сообщения и результат классификации, отображается уведомление «Результат отправлен на ваш email!».

5. Интерфейс и навигация.

Тест 5.1: Навигация через кнопку «Справка».

Тестируется функциональность перехода в окно HelpWindow через кнопку «Справка» в окне MainEntryWindow. Цель данного тестирования в том, чтобы проверить, что кнопка «Справка» открывает окно справки, а кнопка «Назад» возвращает в начальное окно.

Условия тестирования, следующие: приложение запущено.

Тестирование проводится ручное через GUI. Само ручное тестирование надежно для проверки навигации интерфейса.

Для выполнения тестирования в окне MainEntryWindow необходимо нажать кнопку «Справка» и проверить, открылось ли окно HelpWindow. Затем нажать кнопку «Назад» и проверить вернулось ли приложение в окно MainEntryWindow.

Ожидаемые результаты: окно HelpWindow открывается и отображает текст справки, после нажатия кнопки «Назад» открывается окно MainEntryWindow.

## 4.2 Методики тестирования

В рамках процесса тестирования программного обеспечения «SpamBayes» были применены различные методики, направленные на оценку корректности, надежности и масштабируемости работы фильтра спама, основанного на наивном байесовском классификаторе. В соответствии с требованиями к качеству программы и рекомендациями из учебного предмета «Технология разработки программного обеспечения» были выбраны следующие виды тестирования: активное тестирование (Active Testing), тестирование на увеличение нагрузки (Age Testing) и бенчмарк-тестирование (Benchmark Testing). Каждый из этих подходов позволяет всесторонне исследовать характеристики разработанного программного продукта.

Active Testing – данный метод тестирования направлен на активную проверку фильтра классификации спам-сообщений в условиях контролируемого ввода. Были выбраны заранее подготовленные сообщения с метками «спам» и «не спам», включая сообщения, близкие к границе между категориями (граничные случаи). Первоначально данные сообщения были протестированы через внешний сервис фильтрации, выступающий в роли эталонного классификатора, с целью получения контрольного результата. Затем аналогичная выборка была протестирована в разработанном приложении «SpamBayes», что позволило провести сравнительный анализ точности классификации. Такая методика позволяет выявить различия в интерпретации сообщений между существующими фильтрами и реализованным алгоритмом, а также способствует улучшению качества предсказания за счёт дальнейшего обучения модели на основе выявленных расхождений.

Age Testing – цель данного тестирования заключалась в оценке поведения программного обеспечения при прогрессивном увеличении объёма обучающей выборки и пользовательских данных. Методика включала этапы тестирования на партиях из 10, 100, 1000 и более спам-сообщений. На каждом этапе измерялась скорость обработки, потребление ресурсов и точность классификации. Результаты показали, что при увеличении объема обучающей базы наблюдается рост времени на классификацию, а также возрастание требований к объему оперативной памяти. Это соответствует теоретическим положениям, изложенным в разделе, посвящённом масштабируемости ПО, и подтверждает гипотезу о возможном снижении производительности в условиях интенсивной эксплуатации программы. Таким образом, тестирование по методу Age Testing позволило выявить потенциальные риски снижения отзывчивости и производительности при расширении базы данных.

Benchmark Testing – данный вид тестирования предусматривал проведение сравнительного анализа скорости и точности классификации сообщений, обработанных программой. Были использованы эталонные сообщения, которые запускались через классификатор без изменения обучающей базы. Фиксировались ключевые метрики такие, как среднее время обработки одного сообщения, доля правильно классифицированных сообщений, показатели precision и recall. Тестирование позволило оценить эффективность реализованного байесовского фильтра в условиях типовой нагрузки и подтвердило пригодность модели для решения практической задачи фильтрации.

Методики тестирования были выбраны с учетом специфики предметной области, структуры программного обеспечения, а также требований к качеству, изложенных в модели жизненного цикла разработки программы. Применение перечисленных видов тестирования позволило оценить ключевые характеристики приложения «SpamBayes», выявить узкие места в архитектуре и предложить направления для дальнейшей оптимизации, в том числе адаптацию под расширяемые базы и интеграцию с внешними источниками данных. Подробные результаты всех проведенных тестов, включая метрики точности, времени обработки и потребления ресурсов, представлены в приложении Б.

## 4.3 Разработка справочной системы

Справочная система приложения «SpamBayes» разработана для обеспечения удобства пользователей, упрощения освоения функционала и предоставления четких инструкций по использованию программы. Система включает встроенные элементы интерфейса, такие как всплывающие подсказки, интерактивные сообщения и окно справки, а также документацию пользователя, представленную в виде отдельного файла. Основной целью было создание интуитивно понятной и доступной системы, которая минимизирует вопросы пользователей и помогает эффективно взаимодействовать с приложением.

Средства и подходы к разработке справочной системы.

Присутствует встроенное окно справки HelpWindow, которое реализовано как отдельное окно WPF HelpWindow.xaml, доступное из начального экрана MainEntryWindow через кнопку «Справка». Оно содержит текстовое описание функционала программы, инструкции по началу работы и контактную информацию для поддержки. Текст структурирован с использованием TextBlock с поддержкой переноса строк и выделения ключевых заголовков для удобства чтения. Также используется минималистичный дизайн с логотипом приложения и кнопкой «Назад» для возврата на начальный экран.

Есть интерактивные сообщения и уведомления для информирования пользователя о результатах действий (например, успешное добавление текста в обучающую базу или ошибка при пустом вводе), для этого используются диалоговые окна MessageBox. Например, при попытке классификации пустого текста отображается сообщение: «Пожалуйста, введите сообщение». Уведомления также применяются для подтверждения успешной авторизации, регистрации или получения письма через Gmail API, что повышает прозрачность взаимодействия.

Присутствуют плейсхолдеры в текстовых полях. В формах LoginWindow и RegisterWindow используются текстовые подсказки placeholders в полях TextBox и PasswordBox. Например, поле имени пользователя в LoginWindow изначально содержит текст «Имя пользователя», который исчезает при фокусе и возвращается, если поле остается пустым. Подсказки реализованы с помощью событий GotFocus и LostFocus в C# и свойства TextBlock в XAML для PasswordBox, что улучшает пользовательский опыт, особенно для новичков.

Разработана документация пользователя, представленная в виде текстового файла, включенного в приложение см. Приложение В курсового проекта. Данный документ содержит назначение программы и область применения; условия эксплуатации, другими словами, системные требования; пошаговые инструкции по установке и использованию с примерами; описание характеристик программы и мер защиты данных.

В отличие от традиционного формата CHM, справочная система реализована через встроенное окно HelpWindow и текстовую документацию. Это решение обусловлено стремлением минимизировать зависимости от внешних форматов и упростить интеграцию справки в приложение. Окно HelpWindow обеспечивает аналогичный функционал, предоставляя доступ к информации непосредственно в интерфейсе, что более удобно для пользователей.

Справочная система разработана с учетом минимализма и функциональности. Тексты в окне HelpWindow и подсказках краткие и понятные, избегающие технического жаргона, чтобы быть доступными для пользователей с разным уровнем подготовки. Все элементы справочной системы такие, как подсказки, уведомления и документация реализованы на русском языке, что соответствует целевой аудитории и языку интерфейса.

Окно HelpWindow не зависит от внешних ресурсов и работает локально, что исключает сбои при отсутствии интернета. Обработка ошибок в уведомлениях MessageBox обеспечивает информирование пользователя о проблемах, таких как некорректный ввод. Интерфейс окна HelpWindow использует адаптивный дизайн WPF, корректно отображаясь на экранах с различным разрешением. Элемент ScrollViewer позволяет прокручивать длинные тексты справки.

Visual Studio 2022 использовалась для создания XAML интерфейса окна HelpWindow и реализации логики всплывающих подсказок. Средство конструктора XAML позволило быстро разрабатывать и тестировать элементы интерфейса. WPF (Windows Presentation Foundation) применялся для реализации окна справки и всплывающих подсказок с использованием ToolTip, TextBlock и ScrollViewer для поддержки прокрутки длинных текстов. C# использовался для обработки событий интерфейса, таких как нажатие на кнопку «Назад» в HelpWindow и управление плейсхолдерами в формах входа и регистрации. Microsoft Word применялся для подготовки документации пользователя, которая затем конвертировалась в текстовый формат для включения в проект.

Скриншоты справочной системы представлены ниже.

Окно справки HelpWindow, отображает приветственное сообщение, инструкции по началу работы и контактную информацию, а также содержит кнопку «Назад» для возврата на начальный экран, само окно представлено на рисунке 4.3.1.

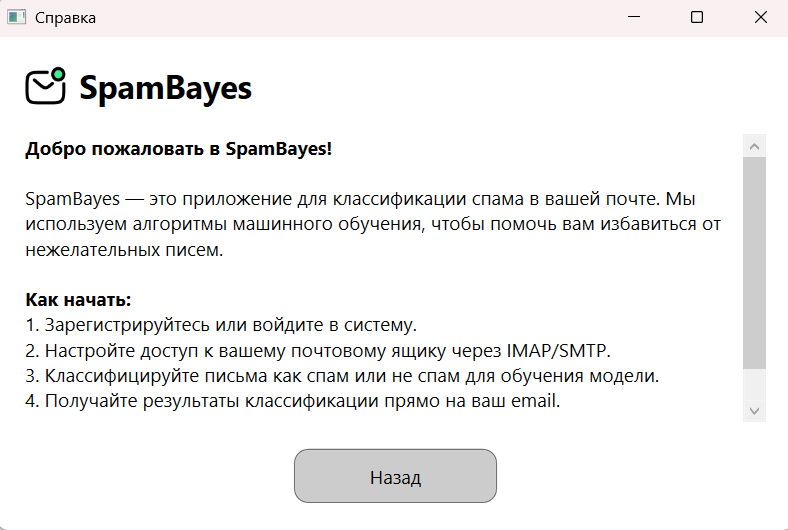


Рисунок 4.3.1 – Окно справки HelpWindow

Плейсхолдер в поле ввода имени пользователя LoginWindow. Отображает текст «Имя пользователя» до ввода данных, представлен на рисунке 4.3.2.

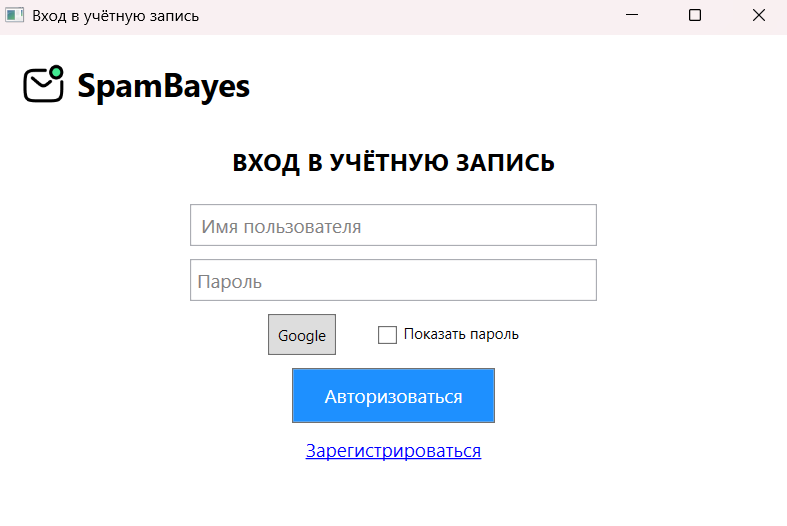


Рисунок 4.3.2 – Плейсхолдер в поле ввода имени пользователя

Уведомление об ошибке при пустом вводе. Отображается сообщение MessageBox при попытке классификации пустого текста, представлено на рисунке 4.3.3

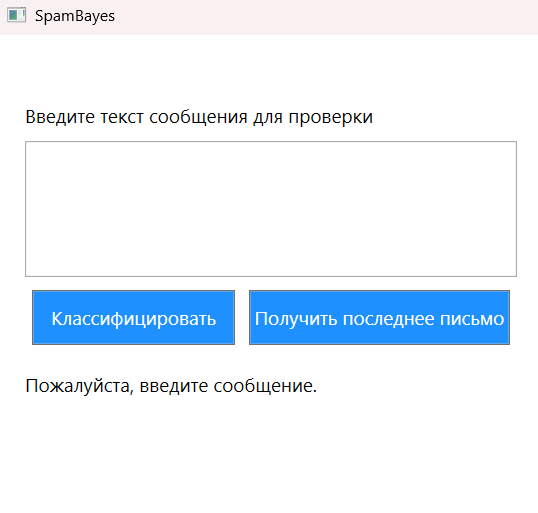


Рисунок 4.3.3 – Уведомление об ошибке при пустом вводе

**Выводы по разделу 4.** Раздел «Отладка и подготовка к внедрению» охватывает ключевые аспекты тестирования, отладки и подготовки приложения «SpamBayes» к внедрению. Проведено ручное тестирование контрольных примеров, подтверждающее корректную работу функционала регистрации, авторизации, классификации сообщений и обработки пользовательских данных. Методики тестирования, такие как Active Testing, Age Testing и Benchmark Testing, позволили оценить точность классификации, поведение системы при увеличении объема обучающих данных, а также производительность при различных сценариях использования. Результаты показали высокую устойчивость алгоритма к изменениям обучающей выборки и позволили выявить риски, связанные с масштабируемостью, которые могут быть устранены при дальнейшем развитии проекта. Кроме того, реализована справочная система, упрощающая освоение интерфейса пользователями. В целом, проведённая отладка и тестирование подтвердили готовность приложения к практическому применению, обеспечив надежную фильтрацию спам-сообщений и устойчивость к ошибкам.

# Заключение

В ходе выполнения курсового проекта на тему «Фильтрация» была разработана и реализована программная система «SpamBayes», предназначенная для автоматической фильтрации текстовых сообщений с использованием наивного байесовского метода. На этапе постановки задачи была обоснована актуальность темы в условиях роста цифровых коммуникаций и киберугроз, сформулированы цель, объект и предмет исследования, а также подробно описана предметная область и рассмотрены существующие аналоги программ. Проведённый анализ подтвердил востребованность создания бесплатного и функционального инструмента для защиты пользователей от нежелательного контента.

Раздел проектирования продемонстрировал теоретическую обоснованность выбора байесовского классификатора, описал математическую модель и алгоритмы, определил структуру входных и выходных данных, а также представил систему с использованием методологии IDEF0. Выбор технологий C#, .NET Framework 4.7, WPF и AllFusion Process Modeler обеспечил совместимость, гибкость и удобство реализации, позволяя достичь поставленных целей.

На этапе разработки была реализована архитектура приложения с удобным и интуитивно понятным интерфейсом, поддержкой пользовательской регистрации, авторизации, обучаемым классификатором и интеграцией с Gmail API. Особенности реализации включали логарифмические вероятности, сглаживание Лапласа, оптимизацию загрузки данных и расширенные механизмы отказоустойчивости, что обеспечило надежную и стабильную работу программы.

Завершающий этап включал отладку и подготовку к внедрению. Проведено ручное тестирование функциональности, а также реализованы три типа тестирования: активное, нагрузочное и бенчмарк. Это позволило выявить потенциальные ограничения производительности при увеличении объема обучающей базы и подтвердить высокую точность классификации. Разработка справочной системы повысила удобство освоения приложения конечными пользователями.

Таким образом в рамках курсового проектирования были достигнуты поставленные цели: закреплены и углублены теоретические знания и практические навыки, полученные в рамках учебного предмета «Основы алгоритмизации и программирования», а также развиты навыки самостоятельного решения задач. Разработанное приложение «SpamBayes», использующее наивный байесовский метод, обеспечивает эффективную и бесплатную классификацию текстовых сообщений на категории «Спам» и «Не спам», демонстрируя высокую отказоустойчивость и обучаемость. Полученные результаты подтверждают практическую ценность созданного инструмента для защиты пользователей от нежелательного контента, а также его потенциал для дальнейшего совершенствования, включая повышение масштабируемости и интеграцию с облачными почтовыми сервисами.

Список использованных источников

1. Кузнецов, М.В. Программирование на C# в среде .NET / М.В. Кузнецов, И.В. Шпак. М.: Бином, 2020. – 512 с.: ил.
2. Олифер, В.Г. Компьютерные сети. Принципы, технологии, протоколы. 6-е издание / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2023. – 1024 с.: ил.
3. Таненбаум, Э. Компьютерные сети / Э. Таненбаум, Д. Уэзеролл; пер. с англ. – М.: Питер, 2021. – 960 с.: ил.
4. Фридман, А. Экономическая эффективность информационных систем / А. Фридман. М.: Экономика, 2019. – 240 с.
5. Априорная вероятность [Электронный ресурс] / Википедия. 2024. Режим доступа: https://ru.wikipedia.org/wiki/Априорная\_вероятность. Дата доступа: 25.03.2025.
6. Байесовские методы в машинном обучении [Электронный ресурс] / Habr. 2023. Режим доступа: https://habr.com/ru/companies/otus/articles/733598/. Дата доступа: 15.03.2025.
7. Теорема Байеса [Электронный ресурс] / Википедия. 2024. Режим доступа: https://ru.wikipedia.org/wiki/Теорема\_Байеса. Дата доступа: 25.03.2025.
8. Функция правдоподобия [Электронный ресурс] / Википедия (на франц. яз.). 2023. Режим доступа: https://ru.frwiki.wiki/wiki/Fonction\_de\_vraisemblance. Дата доступа: 25.03.2025.
9. Clean Email – обзор сервиса [Электронный ресурс] / Ciroapp. 2023. Режим доступа: https://ciroapp.com/ru/clean-email-review/. Дата доступа: 15.03.2025.
10. Gmail API. Руководство по использованию [Электронный ресурс] / Google Developers. 2024. Режим доступа: https://developers.google.com/workspace/gmail/api/guides?hl=ru. – Дата доступа: 30.05.2025.
11. MailWasher Free – бесплатная программа для фильтрации почты [Электронный ресурс] / Softdlyavas.ru. 2022. Режим доступа: https://softdlyavas.ru/mailwasher-skachat-na-kompyuter/. Дата доступа: 15.03.2025.
12. OAuth 2.0 – протокол авторизации [Электронный ресурс] / OAuth Community Site. 2023. Режим доступа: https://oauth.net/2/. Дата доступа: 30.05.2025.
13. Proton Mail [Электронный ресурс] / Википедия. 2024. Режим доступа: https://ru.wikipedia.org/wiki/Proton\_Mail. Дата доступа: 15.03.2025.
14. Spamihilator – антиспамовый фильтр [Электронный ресурс] / Softpedia. 2023. Режим доступа: https://www.softpedia.com/get/Internet/E-mail/Anti-Spam-Tools/Spamihilator.shtml. Дата доступа: 15.03.2025.
15. Visual Studio Documentation [Электронный ресурс] / Microsoft Learn. 2024. Режим доступа: https://docs.microsoft.com/en-us/visualstudio/. Дата доступа: 27.03.2025.
16. WPF – Windows Presentation Foundation [Электронный ресурс] / Microsoft Learn. 2024. Режим доступа: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/. Дата доступа: 28.05.2025.
17. Zoho Mail – обзор почтового клиента [Электронный ресурс] / ITC.ua. 2022. Режим доступа: https://itc.ua/articles/obzor-zoho-mail-mnogofunkczionalnyj-pochtovyj-klient-dlya-malogo-biznesa-i-korporaczij/. Дата доступа: 15.03.2025.

Приложение А  
Текст программы

**class User**

// Определяет пространство имен SpamBayesWPF, которое помогает организовать связанные классы и избежать конфликтов имен.

namespace SpamBayesWPF

{

// Определяет публичный класс User для представления сущности пользователя.

public class User

{

// Публичное свойство для хранения имени пользователя.

// Геттер и сеттер позволяют читать и записывать значение Username.

public string Username { get; set; }

// Публичное свойство для хранения пароля пользователя.

// Геттер и сеттер позволяют читать и записывать значение Password.

// Примечание: Хранение пароля в виде обычного текста небезопасно; в реальном приложении пароли следует хэшировать.

public string Password { get; set; }

// Публичное свойство для хранения адреса электронной почты пользователя.

// Геттер и сеттер позволяют читать и записывать значение Email.

public string Email { get; set; }

}

}

**class UserManager**

using Newtonsoft.Json;

using System.IO;

namespace SpamBayesWPF

{

// Внутренний класс для управления пользователями

internal class UserManager

{

// Путь к файлу для хранения данных пользователей

private const string UsersFilePath = "users.json";

// Список пользователей

private List<User> users;

// Конструктор: загружает пользователей из файла или создает пустой список

public UserManager()

{

if (File.Exists(UsersFilePath))

{

string json = File.ReadAllText(UsersFilePath);

users = JsonConvert.DeserializeObject<List<User>>(json) ?? new List<User>();

}

else

{

users = new List<User>();

SaveUsers();

}

}

// Проверяет валидность пользователя по имени и паролю, возвращает email при успехе

public bool ValidateUser(string username, string password, out string email)

{

var user = users.FirstOrDefault(u => u.Username == username);

if (user != null)

{

// Поддержка входа через Google (пустой пароль)

if (string.IsNullOrEmpty(user.Password) && string.IsNullOrEmpty(password))

{

email = user.Email;

return true;

}

// Проверка пароля (небезопасно: пароли хранятся в открытом виде)

if (user.Password == password)

{

email = user.Email;

return true;

}

}

email = null;

return false;

}

// Регистрирует нового пользователя, если имя уникально

public bool RegisterUser(string username, string password, string email)

{

if (users.Any(u => u.Username == username))

{

return false;

}

users.Add(new User { Username = username, Password = password, Email = email });

SaveUsers();

return true;

}

// Сохраняет список пользователей в JSON-файл

private void SaveUsers()

{

string json = JsonConvert.SerializeObject(users, Formatting.Indented);

File.WriteAllText(UsersFilePath, json);

}

}

}

using System.IO;

namespace SpamBayesWPF

{

// Класс для классификации спама с использованием байесовского подхода

public class SpamClassifier

{

// Словари для хранения частоты слов в спаме и не-спаме

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

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

// Счетчики сообщений спама и не-спама

private int spamCount = 0;

private int hamCount = 0;

// Конструктор: обучает классификатор на файлах spam.txt и ham.txt

public SpamClassifier()

{

TrainClassifier("spam.txt", "ham.txt");

}

// Обучает классификатор, подсчитывая частоту слов в спаме и не-спаме

private void TrainClassifier(string spamFilePath, string hamFilePath)

{

if (!File.Exists(spamFilePath) || !File.Exists(hamFilePath))

{

throw new FileNotFoundException("Файлы данных не найдены!");

}

// Чтение спам-сообщений

foreach (var line in File.ReadLines(spamFilePath))

{

var words = Tokenize(line);

spamCount++;

foreach (var word in words)

{

if (!spamWords.ContainsKey(word))

spamWords[word] = 0;

spamWords[word]++;

}

}

// Чтение не-спам-сообщений

foreach (var line in File.ReadLines(hamFilePath))

{

var words = Tokenize(line);

hamCount++;

foreach (var word in words)

{

if (!hamWords.ContainsKey(word))

hamWords[word] = 0;

hamWords[word]++;

}

}

}

// Классифицирует сообщение как спам или не-спам

public string Classify(string message)

{

var words = Tokenize(message);

// Вычисление априорных вероятностей

double totalMessages = spamCount + hamCount;

double spamPrior = (double)spamCount / totalMessages;

double hamPrior = (double)hamCount / totalMessages;

double spamLikelihood = Math.Log(spamPrior);

double hamLikelihood = Math.Log(hamPrior);

// Вычисление вероятностей слов с учетом сглаживания

foreach (var word in words)

{

double spamWordProb = (spamWords.ContainsKey(word) ? spamWords[word] + 1 : 1) /

(double)(spamWords.Values.Sum() + spamWords.Count);

double hamWordProb = (hamWords.ContainsKey(word) ? hamWords[word] + 1 : 1) /

(double)(hamWords.Values.Sum() + hamWords.Count);

spamLikelihood += Math.Log(spamWordProb);

hamLikelihood += Math.Log(hamWordProb);

}

return spamLikelihood > hamLikelihood ? "Спам" : "Не спам";

}

// Токенизирует текст, разбивая его на слова

private List<string> Tokenize(string message)

{

return message.ToLower().Split(new[] { ' ', '.', ',', '!', '?' }, StringSplitOptions.RemoveEmptyEntries)

.ToList();

}

// Добавляет текст в файл и переобучает классификатор

public string AddTextToFile(string filePath, string message)

{

if (string.IsNullOrWhiteSpace(message))

{

return "Пожалуйста, введите сообщение.";

}

try

{

File.AppendAllText(filePath, message + Environment.NewLine);

spamWords.Clear();

hamWords.Clear();

spamCount = 0;

hamCount = 0;

TrainClassifier("spam.txt", "ham.txt");

return "Сообщение успешно добавлено!";

}

catch (Exception ex)

{

return $"Ошибка: {ex.Message}";

}

}

}

}

**SpamClassifier**

using System.IO;

namespace SpamBayesWPF

{

// Класс для классификации спама с использованием наивного байесовского подхода

public class SpamClassifier

{

// Словари для хранения частоты слов в спаме и не-спаме

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

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

// Счетчики количества спам- и не-спам-сообщений

private int spamCount = 0;

private int hamCount = 0;

// Конструктор: обучает классификатор на файлах spam.txt и ham.txt

public SpamClassifier()

{

TrainClassifier("spam.txt", "ham.txt");

}

// Обучает классификатор, подсчитывая частоту слов в спаме и не-спаме

private void TrainClassifier(string spamFilePath, string hamFilePath)

{

// Проверка существования файлов

if (!File.Exists(spamFilePath) || !File.Exists(hamFilePath))

{

throw new FileNotFoundException("Файлы данных не найдены!");

}

// Чтение спам-сообщений

foreach (var line in File.ReadLines(spamFilePath))

{

var words = Tokenize(line);

spamCount++;

foreach (var word in words)

{

if (!spamWords.ContainsKey(word))

spamWords[word] = 0;

spamWords[word]++;

}

}

// Чтение не-спам-сообщений

foreach (var line in File.ReadLines(hamFilePath))

{

var words = Tokenize(line);

hamCount++;

foreach (var word in words)

{

if (!hamWords.ContainsKey(word))

hamWords[word] = 0;

hamWords[word]++;

}

}

}

// Классифицирует сообщение как спам или не-спам

public string Classify(string message)

{

var words = Tokenize(message);

// Вычисление априорных вероятностей

double totalMessages = spamCount + hamCount;

double spamPrior = (double)spamCount / totalMessages;

double hamPrior = (double)hamCount / totalMessages;

double spamLikelihood = Math.Log(spamPrior);

double hamLikelihood = Math.Log(hamPrior);

// Вычисление логарифма правдоподобия для каждого слова с использованием сглаживания Лапласа

foreach (var word in words)

{

double spamWordProb = (spamWords.ContainsKey(word) ? spamWords[word] + 1 : 1) /

(double)(spamWords.Values.Sum() + spamWords.Count);

double hamWordProb = (hamWords.ContainsKey(word) ? hamWords[word] + 1 : 1) /

(double)(hamWords.Values.Sum() + hamWords.Count);

spamLikelihood += Math.Log(spamWordProb);

hamLikelihood += Math.Log(hamWordProb);

}

// Сравнение правдоподобий для определения категории

return spamLikelihood > hamLikelihood ? "Спам" : "Не спам";

}

// Токенизирует текст, разбивая его на слова

private List<string> Tokenize(string message)

{

// Приведение к нижнему регистру и разбиение по пробелам и знакам препинания

return message.ToLower().Split(new[] { ' ', '.', ',', '!', '?' }, StringSplitOptions.RemoveEmptyEntries)

.ToList();

}

// Добавляет текст в файл и переобучает классификатор

public string AddTextToFile(string filePath, string message)

{

// Проверка на пустое сообщение

if (string.IsNullOrWhiteSpace(message))

{

return "Пожалуйста, введите сообщение.";

}

try

{

// Добавление сообщения в файл

File.AppendAllText(filePath, message + Environment.NewLine);

// Очистка словарей и счетчиков

spamWords.Clear();

hamWords.Clear();

spamCount = 0;

hamCount = 0;

// Переобучение классификатора

TrainClassifier("spam.txt", "ham.txt");

return "Сообщение успешно добавлено!";

}

catch (Exception ex)

{

return $"Ошибка: {ex.Message}";

}

}

}

}

**WidthConverter**

using System.Globalization;

using System.Windows.Data;

namespace SpamBayesWPF

{

// Класс-конвертер для преобразования ширины в WPF

public class WidthConverter : IValueConverter

{

// Преобразует входное значение ширины, уменьшая его на 20%

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is double width)

{

return width \* 0.8; // Уменьшает ширину на 20% от исходного значения

}

return value; // Возвращает исходное значение, если преобразование невозможно

}

// Обратное преобразование не реализовано

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException(); // Выбрасывает исключение, так как обратное преобразование не поддерживается

}

}

}

**Описание интерфейса окна MainEntryWindow**

<Window x:Class="SpamBayesWPF.MainEntryWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SpamBayesWPF"

mc:Ignorable="d"

Title="SpamBayes" Height="448" Width="800" WindowStartupLocation="CenterScreen">

<!-- Главное окно приложения с заданным заголовком, размером и центрированием на экране -->

<Window.Resources>

<!-- Ресурсы окна -->

<!-- Конвертер для текста (если его нет в проекте) -->

<local:WidthConverter x:Key="WidthConverter"/>

<!-- Определение конвертера для динамической настройки ширины текста -->

<!-- Стиль для кнопок -->

<Style x:Key="RoundedButtonStyle" TargetType="Button">

<!-- Стиль для кнопок с закругленными углами и эффектом свечения -->

<Setter Property="Background" Value="#1E90FF"/>

<!-- Устанавливает синий фон кнопки -->

<Setter Property="Foreground" Value="White"/>

<!-- Устанавливает белый цвет текста -->

<Setter Property="BorderThickness" Value="0"/>

<!-- Убирает границу кнопки -->

<Setter Property="FontSize" Value="14"/>

<!-- Задает размер шрифта 14 -->

<Setter Property="Cursor" Value="Hand"/>

<!-- Устанавливает курсор в виде руки при наведении -->

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<!-- Шаблон кнопки -->

<Border x:Name="border"

Background="{TemplateBinding Background}"

CornerRadius="5"

Padding="10,5">

<!-- Граница кнопки с закруглением и отступами -->

<Border.Effect>

<DropShadowEffect ShadowDepth="0"

Color="#1E90FF"

Opacity="0.6"

BlurRadius="8"/>

<!-- Эффект свечения по умолчанию -->

</Border.Effect>

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

<!-- Отображает содержимое кнопки (текст) по центру -->

</Border>

<ControlTemplate.Triggers>

<!-- Триггеры для изменения стиля при взаимодействии -->

<Trigger Property="IsMouseOver" Value="True">

<!-- При наведении курсора -->

<Setter Property="Background" Value="#00008B"/>

<!-- Изменяет фон на более темный синий -->

<Setter TargetName="border" Property="Effect">

<Setter.Value>

<DropShadowEffect ShadowDepth="0"

Color="#00008B"

Opacity="0.8"

BlurRadius="12"/>

<!-- Усиливает свечение -->

</Setter.Value>

</Setter>

</Trigger>

<Trigger Property="IsPressed" Value="True">

<!-- При нажатии кнопки -->

<Setter Property="Background" Value="#000055"/>

<!-- Изменяет фон на еще более темный -->

<Setter TargetName="border" Property="Effect">

<Setter.Value>

<DropShadowEffect ShadowDepth="0"

Color="#000055"

Opacity="0.9"

BlurRadius="10"/>

<!-- Изменяет свечение при нажатии -->

</Setter.Value>

</Setter>

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Window.Resources>

<Grid>

<!-- Основная сетка для разметки окна -->

<Grid.ColumnDefinitions>

<!-- Делит окно на две равные колонки -->

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<!-- Левая часть: Логотип и текст -->

<Grid Grid.Column="0">

<!-- Сетка для левой половины окна -->

<Grid.RowDefinitions>

<!-- Делит левую часть на две строки: авто и остальное -->

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Image Source="/icon.png" Width="30" Height="30" HorizontalAlignment="Left" Margin="10,10,0,0"/>

<!-- Логотип приложения в левом верхнем углу -->

<TextBlock Text="SpamBayes" FontSize="36" FontWeight="Bold" Foreground="#1E90FF" Margin="50,10,0,0" HorizontalAlignment="Left"/>

<!-- Заголовок приложения с названием -->

<StackPanel Grid.Row="1" VerticalAlignment="Center" Margin="20">

<!-- Контейнер для текста, выровненного по центру -->

<TextBlock Text="Избавьтесь от спама раз и навсегда!" FontSize="18" FontWeight="Bold" TextWrapping="Wrap" MaxWidth="{Binding ActualWidth, Converter={StaticResource WidthConverter}, RelativeSource={RelativeSource AncestorType={x:Type Grid}}}"/>

<!-- Основной рекламный слоган -->

<TextBlock Text="Просто, эффективно и чистый почтовый ящик — всё, что нужно для вашего комфорта" FontSize="14" TextWrapping="Wrap" MaxWidth="{Binding ActualWidth, Converter={StaticResource WidthConverter}, RelativeSource={RelativeSource AncestorType={x:Type Grid}}}"/>

<!-- Дополнительное описание -->

</StackPanel>

</Grid>

<!-- Правая часть: Изображения -->

<Grid Grid.Column="1">

<!-- Сетка для правой половины окна -->

<Image Source="/mail.png" Width="213" Height="166" HorizontalAlignment="Left" VerticalAlignment="Top" RenderTransformOrigin="0.5,0.5" Stretch="Uniform" Margin="-215,38,0,0">

<!-- Изображение письма с поворотом -->

<Image.RenderTransform>

<TransformGroup>

<RotateTransform Angle="-6.659"/>

<!-- Поворот изображения на -6.659 градусов -->

</TransformGroup>

</Image.RenderTransform>

</Image>

<Image Source="/check.png" Width="266" Height="259" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="0,107,0,0" RenderTransformOrigin="0.5,0.5" Stretch="Uniform">

<!-- Изображение галочки с поворотом -->

<Image.RenderTransform>

<TransformGroup>

<RotateTransform Angle="14.974"/>

<!-- Поворот изображения на 14.974 градусов -->

</TransformGroup>

</Image.RenderTransform>

</Image>

</Grid>

<!-- Кнопки внизу -->

<StackPanel Grid.Column="0" Grid.ColumnSpan="2" VerticalAlignment="Bottom" HorizontalAlignment="Center" Orientation="Horizontal" Margin="0,0,0,20">

<!-- Панель для кнопок, расположенных горизонтально внизу -->

<Button x:Name="RegisterButton" Content="Регистрация" Width="150" Height="40" Margin="0,0,20,0" Background="#1E90FF" Foreground="White" FontSize="14" Click="RegisterButton\_Click" Style="{StaticResource RoundedButtonStyle}"/>

<!-- Кнопка регистрации -->

<Button x:Name="LoginButton" Content="Вход" Width="150" Height="40" Margin="0,0,20,0" Background="#1E90FF" Foreground="White" FontSize="14" Click="LoginButton\_Click" Style="{StaticResource RoundedButtonStyle}"/>

<!-- Кнопка входа -->

<Button x:Name="HelpButton" Content="Справка" Width="150" Height="40" Background="#1E90FF" Foreground="White" FontSize="14" Click="HelpButton\_Click" Style="{StaticResource RoundedButtonStyle}"/>

<!-- Кнопка справки -->

</StackPanel>

</Grid>

</Window>

**Реализация интерфейса окна MainEntryWindow**

using System.Windows;

namespace SpamBayesWPF

{

/// <summary>

/// Логика взаимодействия для MainEntryWindow.xaml

/// </summary>

public partial class MainEntryWindow : Window

{

// Конструктор: инициализирует компоненты XAML

public MainEntryWindow()

{

InitializeComponent();

}

// Обработчик клика по кнопке "Вход": открывает окно логина и закрывает текущее

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

{

LoginWindow loginWindow = new LoginWindow();

loginWindow.Show();

this.Close();

}

// Обработчик клика по кнопке "Регистрация": открывает окно регистрации и закрывает текущее

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

{

RegisterWindow registerWindow = new RegisterWindow();

registerWindow.Show();

this.Close();

}

// Обработчик клика по кнопке "Справка": открывает окно справки и закрывает текущее

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

{

HelpWindow helpWindow = new HelpWindow();

helpWindow.Show();

this.Close();

}

}

}

**Описание интерфейса окна LoginWindow**

<Window x:Class="SpamBayesWPF.LoginWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SpamBayesWPF"

mc:Ignorable="d"

Title="Вход в учётную запись" Height="400" Width="600">

<!-- Окно авторизации с заголовком, фиксированными размерами -->

<Grid Margin="20">

<!-- Основная сетка с отступами и двумя строками -->

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<!-- Для логотипа и заголовка -->

<RowDefinition Height="\*"/>

<!-- Для формы входа -->

</Grid.RowDefinitions>

<!-- Логотип и заголовок -->

<StackPanel Orientation="Horizontal" Grid.Row="0">

<!-- Горизонтальный контейнер для логотипа и названия -->

<Image Source="/icon.png" Width="30" Height="30" Margin="0,0,10,0"/>

<!-- Логотип приложения -->

<TextBlock Text="SpamBayes" FontSize="24" FontWeight="Bold" VerticalAlignment="Center"/>

<!-- Название приложения -->

</StackPanel>

<!-- Форма входа -->

<StackPanel Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Center" Width="300">

<!-- Центрированный контейнер для элементов формы -->

<TextBlock Text="ВХОД В УЧЁТНУЮ ЗАПИСЬ" FontSize="18" FontWeight="Bold" Margin="0,0,0,20" HorizontalAlignment="Center"/>

<!-- Заголовок формы -->

<!-- Поле для имени пользователя с подсказкой -->

<Grid Margin="0,0,0,10">

<TextBox x:Name="UsernameTextBox" Padding="5" FontSize="14" Text="Имя пользователя" Foreground="Gray" GotFocus="UsernameTextBox\_GotFocus" LostFocus="UsernameTextBox\_LostFocus"/>

<!-- Поле ввода имени пользователя с обработчиками фокуса для подсказки -->

</Grid>

<!-- Поле для пароля с подсказкой -->

<Grid Margin="0,0,0,10">

<PasswordBox x:Name="PasswordBox" Padding="5" FontSize="14" PasswordChanged="PasswordBox\_PasswordChanged"/>

<!-- Поле для ввода пароля -->

<TextBox x:Name="PasswordTextBox" Padding="5" FontSize="14" Visibility="Collapsed" TextChanged="PasswordTextBox\_TextChanged"/>

<!-- Альтернативное текстовое поле для отображения пароля (скрыто по умолчанию) -->

<TextBlock x:Name="PasswordPlaceholder" Text="Пароль" FontSize="14" Foreground="Gray" IsHitTestVisible="False" Margin="5,0,0,0" VerticalAlignment="Center"/>

<!-- Подсказка для поля пароля, не взаимодействующая с мышью -->

</Grid>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="0,0,0,10">

<!-- Контейнер для кнопки Google и чекбокса -->

<Button x:Name="GoogleLoginButton" Content="Google" Width="50" Height="30" Margin="0,0,10,0" Click="GoogleLoginButton\_Click"/>

<!-- Кнопка для входа через Google -->

<CheckBox x:Name="ShowPasswordCheckBox" Content="Показать пароль" Margin="20,0,0,0" VerticalAlignment="Center" Checked="ShowPasswordCheckBox\_Checked" Unchecked="ShowPasswordCheckBox\_Unchecked"/>

<!-- Чекбокс для переключения видимости пароля -->

</StackPanel>

<Button x:Name="LoginButton" Content="Авторизоваться" Width="150" Height="40" Click="LoginButton\_Click" Background="#1E90FF" Foreground="White" FontSize="14"/>

<!-- Кнопка авторизации -->

<!-- Ссылка на форму регистрации -->

<TextBlock Text="Зарегистрироваться" FontSize="14" Foreground="Blue" TextDecorations="Underline" HorizontalAlignment="Center" Margin="0,10,0,0" Cursor="Hand" MouseLeftButtonDown="RegisterLink\_MouseLeftButtonDown"/>

<!-- Текстовая ссылка для перехода к регистрации -->

</StackPanel>

</Grid>

</Window>

Описание интерфейса окна LoginWindow

using Google.Apis.Auth.OAuth2;

using Newtonsoft.Json.Linq;

using System.Diagnostics;

using System.Net.Http;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace SpamBayesWPF

{

/// <summary>

/// Логика взаимодействия для LoginWindow.xaml

/// </summary>

public partial class LoginWindow : Window

{

// Экземпляр UserManager для управления пользователями

private UserManager userManager;

// Конструктор: инициализирует компоненты и UserManager

public LoginWindow()

{

InitializeComponent();

userManager = new UserManager();

// Инициализация подсказки для поля пароля

UpdatePasswordPlaceholder();

}

// Обработчик клика по кнопке "Авторизоваться"

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

{

// Выбор источника пароля в зависимости от состояния чекбокса

string password = ShowPasswordCheckBox.IsChecked == true ? PasswordTextBox.Text : PasswordBox.Password;

// Проверка учетных данных

if (userManager.ValidateUser(UsernameTextBox.Text, password, out string email))

{

// Открытие главного окна при успешной авторизации

MainWindow mainWindow = new MainWindow(UsernameTextBox.Text, email);

mainWindow.Show();

this.Close();

}

else

{

// Сообщение об ошибке при неверных данных

MessageBox.Show("Неверный логин или пароль.");

}

}

// Обработчик входа через Google (асинхронный)

private async void GoogleLoginButton\_Click(object sender, RoutedEventArgs e)

{

try

{

// Области доступа для получения email и профиля

string[] scopes = { "https://www.googleapis.com/auth/userinfo.email", "https://www.googleapis.com/auth/userinfo.profile" };

string clientId = "369163038165-a5877pgm86t9cchoqgods3v16u23qe95.apps.googleusercontent.com"; // Client ID для Google API

string clientSecret = "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"; // Client Secret для Google API

// Аутентификация через Google

var credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(

new ClientSecrets

{

ClientId = clientId,

ClientSecret = clientSecret

},

scopes,

"admin",

CancellationToken.None,

new Google.Apis.Util.Store.FileDataStore("SpamBayes", true));

if (credential != null)

{

// Получение данных пользователя

using (var httpClient = new HttpClient())

{

httpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", credential.Token.AccessToken);

var response = await httpClient.GetAsync("https://www.googleapis.com/oauth2/v2/userinfo");

if (response.IsSuccessStatusCode)

{

var json = await response.Content.ReadAsStringAsync();

var userInfo = JObject.Parse(json);

string username = userInfo["name"]?.ToString();

string email = userInfo["email"]?.ToString();

// Проверка наличия данных пользователя

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(email))

{

MessageBox.Show("Не удалось получить данные пользователя.");

return;

}

// Регистрация нового пользователя, если он не существует

if (!userManager.ValidateUser(username, "", out \_))

{

userManager.RegisterUser(username, "", email);

}

// Переход в главное окно

MainWindow mainWindow = new MainWindow(username, email);

mainWindow.Show();

this.Close();

}

else

{

MessageBox.Show($"Ошибка получения данных пользователя: {response.StatusCode}");

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка входа через Google: {ex.Message}");

}

}

// Обработчик входа через Mail.ru (асинхронный)

private async void MailRuLoginButton\_Click(object sender, RoutedEventArgs e)

{

try

{

string clientId = "790652"; // Client ID для Mail.ru

string clientSecret = "8c5c7e8e9bea71e5b879446d57d9cf2c"; // Client Secret для Mail.ru

string redirectUri = "urn:ietf:wg:oauth:2.0:oob:auto";

// Генерация уникального state для защиты от CSRF

string state = Guid.NewGuid().ToString("N");

// Формирование URL для авторизации

string authUrl = $"https://oauth.mail.ru/login?client\_id={clientId}&response\_type=code&scope=userinfo&redirect\_uri={redirectUri}&state={state}";

// Открытие браузера для авторизации

Process.Start(new ProcessStartInfo(authUrl) { UseShellExecute = true });

// Запрос кода авторизации у пользователя

string input = Microsoft.VisualBasic.Interaction.InputBox("Введите код авторизации и state (в формате код:state), полученные после входа в Mail.ru:", "Mail.ru авторизация");

if (string.IsNullOrEmpty(input))

{

MessageBox.Show("Код авторизации не введён.");

return;

}

// Разделение ввода на код и state

var parts = input.Split(':');

if (parts.Length != 2)

{

MessageBox.Show("Неверный формат. Введите код и state в формате код:state.");

return;

}

string code = parts[0];

string returnedState = parts[1];

// Проверка state для защиты от CSRF

if (returnedState != state)

{

MessageBox.Show("Ошибка: значение state не совпадает. Возможная CSRF-атака.");

return;

}

// Обмен кода на токен доступа

using (var httpClient = new HttpClient())

{

var tokenRequest = new Dictionary<string, string>

{

{ "client\_id", clientId },

{ "client\_secret", clientSecret },

{ "grant\_type", "authorization\_code" },

{ "code", code },

{ "redirect\_uri", redirectUri }

};

var tokenResponse = await httpClient.PostAsync("https://o2.mail.ru/token", new FormUrlEncodedContent(tokenRequest));

if (tokenResponse.IsSuccessStatusCode)

{

var tokenJson = await tokenResponse.Content.ReadAsStringAsync();

var tokenData = JObject.Parse(tokenJson);

string accessToken = tokenData["access\_token"]?.ToString();

if (string.IsNullOrEmpty(accessToken))

{

MessageBox.Show("Не удалось получить токен доступа.");

return;

}

// Получение данных пользователя

httpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", accessToken);

var userResponse = await httpClient.GetAsync("https://o2.mail.ru/userinfo");

if (userResponse.IsSuccessStatusCode)

{

var userJson = await userResponse.Content.ReadAsStringAsync();

var userInfo = JObject.Parse(userJson);

string username = userInfo["name"]?.ToString();

string email = userInfo["email"]?.ToString();

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(email))

{

MessageBox.Show("Не удалось получить данные пользователя.");

return;

}

// Регистрация нового пользователя, если он не существует

if (!userManager.ValidateUser(username, "", out \_))

{

userManager.RegisterUser(username, "", email);

}

// Переход в главное окно

MainWindow mainWindow = new MainWindow(username, email);

mainWindow.Show();

this.Close();

}

else

{

var userErrorContent = await userResponse.Content.ReadAsStringAsync();

MessageBox.Show($"Ошибка получения данных пользователя: {userResponse.StatusCode}\nДетали: {userErrorContent}");

}

}

else

{

var tokenErrorContent = await tokenResponse.Content.ReadAsStringAsync();

MessageBox.Show($"Ошибка получения токена: {tokenResponse.StatusCode}\nДетали: {tokenErrorContent}");

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка входа через Mail.ru: {ex.Message}");

}

}

// Обработчик клика по ссылке "Зарегистрироваться"

private void RegisterLink\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

// Открытие окна регистрации

RegisterWindow registerWindow = new RegisterWindow();

registerWindow.Show();

this.Close();

}

// Обработчик получения фокуса для поля имени пользователя

private void UsernameTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

if (UsernameTextBox.Text == "Имя пользователя")

{

UsernameTextBox.Text = "";

UsernameTextBox.Foreground = System.Windows.Media.Brushes.Black;

}

}

// Обработчик потери фокуса для поля имени пользователя

private void UsernameTextBox\_LostFocus(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(UsernameTextBox.Text))

{

UsernameTextBox.Text = "Имя пользователя";

UsernameTextBox.Foreground = System.Windows.Media.Brushes.Gray;

}

}

// Обработчик изменения пароля в PasswordBox

private void PasswordBox\_PasswordChanged(object sender, RoutedEventArgs e)

{

UpdatePasswordPlaceholder();

}

// Обновление видимости подсказки для пароля

private void UpdatePasswordPlaceholder()

{

// Показывать подсказку, если поле пароля пустое

string currentPassword = ShowPasswordCheckBox.IsChecked == true ? PasswordTextBox.Text : PasswordBox.Password;

PasswordPlaceholder.Visibility = string.IsNullOrEmpty(currentPassword)

? Visibility.Visible

: Visibility.Hidden;

}

// Обработчик изменения текста в PasswordTextBox

private void PasswordTextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

// Синхронизация текста с PasswordBox

PasswordBox.Password = PasswordTextBox.Text;

UpdatePasswordPlaceholder();

}

// Обработчик активации чекбокса "Показать пароль"

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

{

// Переключение на TextBox для отображения пароля

PasswordTextBox.Text = PasswordBox.Password;

PasswordTextBox.Visibility = Visibility.Visible;

PasswordBox.Visibility = Visibility.Collapsed;

UpdatePasswordPlaceholder();

}

// Обработчик деактивации чекбокса "Показать пароль"

private void ShowPasswordCheckBox\_Unchecked(object sender, RoutedEventArgs e)

{

// Переключение на PasswordBox для скрытия пароля

PasswordBox.Password = PasswordTextBox.Text;

PasswordBox.Visibility = Visibility.Visible;

PasswordTextBox.Visibility = Visibility.Collapsed;

UpdatePasswordPlaceholder();

}

}

}

**Описание интерфейса окна RegisterWindow**

<Window x:Class="SpamBayesWPF.RegisterWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SpamBayesWPF"

mc:Ignorable="d"

Title="Регистрация" Height="450" Width="600">

<!-- Окно регистрации с заголовком и фиксированными размерами -->

<Grid Margin="20">

<!-- Основная сетка с отступами и двумя строками -->

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<!-- Для логотипа и заголовка -->

<RowDefinition Height="\*"/>

<!-- Для формы регистрации -->

</Grid.RowDefinitions>

<!-- Логотип и заголовок -->

<StackPanel Orientation="Horizontal" Grid.Row="0">

<!-- Горизонтальный контейнер для логотипа и названия -->

<Image Source="/icon.png" Width="30" Height="30" Margin="0,0,10,0"/>

<!-- Логотип приложения -->

<TextBlock Text="SpamBayes" FontSize="24" FontWeight="Bold" VerticalAlignment="Center"/>

<!-- Название приложения -->

</StackPanel>

<!-- Форма регистрации -->

<StackPanel Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Center" Width="300">

<!-- Центрированный контейнер для элементов формы -->

<TextBlock Text="РЕГИСТРАЦИЯ" FontSize="18" FontWeight="Bold" Margin="0,0,0,20" HorizontalAlignment="Center"/>

<!-- Заголовок формы -->

<!-- Поле для имени пользователя с подсказкой -->

<Grid Margin="0,0,0,10">

<TextBox x:Name="UsernameTextBox" Padding="5" FontSize="14" Text="Введите имя пользователя" Foreground="Gray" GotFocus="UsernameTextBox\_GotFocus" LostFocus="UsernameTextBox\_LostFocus"/>

<!-- Поле ввода имени пользователя с обработчиками фокуса для подсказки -->

</Grid>

<!-- Поле для email с подсказкой -->

<Grid Margin="0,0,0,10">

<TextBox x:Name="EmailTextBox" Padding="5" FontSize="14" Text="Введите email" Foreground="Gray" GotFocus="EmailTextBox\_GotFocus" LostFocus="EmailTextBox\_LostFocus"/>

<!-- Поле ввода email с обработчиками фокуса для подсказки -->

</Grid>

<!-- Поле для пароля с подсказкой -->

<Grid Margin="0,0,0,10">

<PasswordBox x:Name="PasswordBox" Padding="5" FontSize="14" PasswordChanged="PasswordBox\_PasswordChanged"/>

<!-- Поле для ввода пароля -->

<TextBox x:Name="PasswordTextBox" Padding="5" FontSize="14" Visibility="Collapsed" TextChanged="PasswordTextBox\_TextChanged"/>

<!-- Альтернативное текстовое поле для отображения пароля (скрыто по умолчанию) -->

<TextBlock x:Name="PasswordPlaceholder" Text="Введите пароль" FontSize="14" Foreground="Gray" IsHitTestVisible="False" Margin="5,0,0,0" VerticalAlignment="Center"/>

<!-- Подсказка для поля пароля, не взаимодействующая с мышью -->

</Grid>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="0,0,0,10">

<!-- Контейнер для чекбокса -->

<CheckBox x:Name="ShowPasswordCheckBox" Content="Показать пароль" Margin="20,0,0,0" VerticalAlignment="Center" Checked="ShowPasswordCheckBox\_Checked" Unchecked="ShowPasswordCheckBox\_Unchecked"/>

<!-- Чекбокс для переключения видимости пароля -->

</StackPanel>

<Button x:Name="RegisterButton" Content="Зарегистрироваться" Width="150" Height="40" Click="RegisterButton\_Click" Background="#1E90FF" Foreground="White" FontSize="14"/>

<!-- Кнопка регистрации -->

</StackPanel>

</Grid>

</Window>

**Реализация интерфейса окна RegisterWindow**

using System.Windows;

using System.Windows.Controls;

namespace SpamBayesWPF

{

/// <summary>

/// Логика взаимодействия для RegisterWindow.xaml

/// </summary>

public partial class RegisterWindow : Window

{

// Экземпляр UserManager для управления пользователями

private UserManager userManager;

// Конструктор: инициализирует компоненты и UserManager

public RegisterWindow()

{

InitializeComponent();

userManager = new UserManager();

// Инициализация подсказки для поля пароля

UpdatePasswordPlaceholder();

}

// Обработчик клика по кнопке "Зарегистрироваться"

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

{

// Выбор источника пароля в зависимости от состояния чекбокса

string password = ShowPasswordCheckBox.IsChecked == true ? PasswordTextBox.Text : PasswordBox.Password;

// Проверка на заполненность всех полей и отсутствие подсказок

if (string.IsNullOrWhiteSpace(UsernameTextBox.Text) || UsernameTextBox.Text == "Введите имя пользователя" ||

string.IsNullOrWhiteSpace(EmailTextBox.Text) || EmailTextBox.Text == "Введите email" ||

string.IsNullOrWhiteSpace(password))

{

MessageBox.Show("Пожалуйста, заполните все поля.");

return;

}

// Регистрация пользователя

if (userManager.RegisterUser(UsernameTextBox.Text, password, EmailTextBox.Text))

{

// Успешная регистрация: сообщение и переход к окну входа

MessageBox.Show("Регистрация успешна! Теперь вы можете войти.");

LoginWindow loginWindow = new LoginWindow();

loginWindow.Show();

this.Close();

}

else

{

// Ошибка: пользователь с таким именем уже существует

MessageBox.Show("Пользователь с таким именем уже существует.");

}

}

// Обработчик получения фокуса для поля имени пользователя

private void UsernameTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

// Очистка поля, если оно содержит текст подсказки

if (UsernameTextBox.Text == "Введите имя пользователя")

{

UsernameTextBox.Text = "";

UsernameTextBox.Foreground = System.Windows.Media.Brushes.Black;

}

}

// Обработчик потери фокуса для поля имени пользователя

private void UsernameTextBox\_LostFocus(object sender, RoutedEventArgs e)

{

// Восстановление подсказки, если поле пустое

if (string.IsNullOrWhiteSpace(UsernameTextBox.Text))

{

UsernameTextBox.Text = "Введите имя пользователя";

UsernameTextBox.Foreground = System.Windows.Media.Brushes.Gray;

}

}

// Обработчик получения фокуса для поля email

private void EmailTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

// Очистка поля, если оно содержит текст подсказки

if (EmailTextBox.Text == "Введите email")

{

EmailTextBox.Text = "";

EmailTextBox.Foreground = System.Windows.Media.Brushes.Black;

}

}

// Обработчик потери фокуса для поля email

private void EmailTextBox\_LostFocus(object sender, RoutedEventArgs e)

{

// Восстановление подсказки, если поле пустое

if (string.IsNullOrWhiteSpace(EmailTextBox.Text))

{

EmailTextBox.Text = "Введите email";

EmailTextBox.Foreground = System.Windows.Media.Brushes.Gray;

}

}

// Обработчик изменения пароля в PasswordBox

private void PasswordBox\_PasswordChanged(object sender, RoutedEventArgs e)

{

// Синхронизация пароля с TextBox и обновление подсказки

PasswordTextBox.Text = PasswordBox.Password;

UpdatePasswordPlaceholder();

}

// Обработчик изменения текста в PasswordTextBox

private void PasswordTextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

// Синхронизация текста с PasswordBox и обновление подсказки

PasswordBox.Password = PasswordTextBox.Text;

UpdatePasswordPlaceholder();

}

// Обновление видимости подсказки для пароля

private void UpdatePasswordPlaceholder()

{

// Показывать подсказку, если поле пароля пустое

string currentPassword = ShowPasswordCheckBox.IsChecked == true ? PasswordTextBox.Text : PasswordBox.Password;

PasswordPlaceholder.Visibility = string.IsNullOrEmpty(currentPassword)

? Visibility.Visible

: Visibility.Hidden;

}

// Обработчик активации чекбокса "Показать пароль"

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

{

// Переключение на TextBox для отображения пароля

PasswordTextBox.Text = PasswordBox.Password;

PasswordTextBox.Visibility = Visibility.Visible;

PasswordBox.Visibility = Visibility.Collapsed;

UpdatePasswordPlaceholder();

}

// Обработчик деактивации чекбокса "Показать пароль"

private void ShowPasswordCheckBox\_Unchecked(object sender, RoutedEventArgs e)

{

// Переключение на PasswordBox для скрытия пароля

PasswordBox.Password = PasswordTextBox.Text;

PasswordBox.Visibility = Visibility.Visible;

PasswordTextBox.Visibility = Visibility.Collapsed;

UpdatePasswordPlaceholder();

}

}

}

**Описание интерфейса окна HelpWindow**

<Window x:Class="SpamBayesWPF.HelpWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SpamBayesWPF"

mc:Ignorable="d"

Title="Справка" Height="400" Width="600" WindowStartupLocation="CenterScreen">

<!-- Окно справки с заголовком, фиксированными размерами и центрировкой при запуске -->

<Grid Margin="20">

<!-- Основная сетка с отступами и тремя строками -->

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<!-- Для логотипа и заголовка -->

<RowDefinition Height="\*"/>

<!-- Для текста справки -->

<RowDefinition Height="Auto"/>

<!-- Для кнопки "Назад" -->

</Grid.RowDefinitions>

<!-- Логотип и заголовок -->

<StackPanel Orientation="Horizontal" Grid.Row="0">

<!-- Горизонтальный контейнер для логотипа и названия -->

<Image Source="/icon.png" Width="30" Height="30" Margin="0,0,10,0"/>

<!-- Логотип приложения -->

<TextBlock Text="SpamBayes" FontSize="24" FontWeight="Bold" VerticalAlignment="Center"/>

<!-- Название приложения -->

</StackPanel>

<!-- Текст справки -->

<ScrollViewer Grid.Row="1" Margin="0,20,0,20">

<!-- Прокручиваемый контейнер для текста справки -->

<TextBlock TextWrapping="Wrap" FontSize="14">

<!-- Текст с переносом строк и форматированием -->

<Run FontWeight="Bold">Добро пожаловать в SpamBayes!</Run>

<LineBreak/><LineBreak/>

SpamBayes — это приложение для классификации спама в вашей почте. Мы используем алгоритмы машинного обучения, чтобы помочь вам избавиться от нежелательных писем.

<LineBreak/><LineBreak/>

<Run FontWeight="Bold">Как начать:</Run>

<LineBreak/>

1. Зарегистрируйтесь или войдите в систему.

<LineBreak/>

2. Настройте доступ к вашему почтовому ящику через IMAP/SMTP.

<LineBreak/>

3. Классифицируйте письма как спам или не спам для обучения модели.

<LineBreak/>

4. Получайте результаты классификации прямо на ваш email.

<LineBreak/><LineBreak/>

Если у вас есть вопросы, обратитесь к разработчику: support@spambayes.com

</TextBlock>

</ScrollViewer>

<!-- Кнопка "Назад" -->

<Button x:Name="BackButton" Content="Назад" Width="150" Height="40" Grid.Row="2" HorizontalAlignment="Center" Background="#CCCCCC" Foreground="Black" FontSize="14" Click="BackButton\_Click" Style="{StaticResource RoundedButtonStyle}"/>

<!-- Кнопка для возврата с обработчиком события -->

</Grid>

</Window>

**Реализация интерфейса окна HelpWindow**

using System.Windows;

namespace SpamBayesWPF

{

/// <summary>

/// Логика взаимодействия для HelpWindow.xaml

/// </summary>

public partial class HelpWindow : Window

{

// Конструктор: инициализирует компоненты XAML

public HelpWindow()

{

InitializeComponent();

}

// Обработчик клика по кнопке "Назад": возвращает к главному окну

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

{

MainEntryWindow mainEntryWindow = new MainEntryWindow();

mainEntryWindow.Show();

this.Close();

}

}

}

**Описание интерфейса окна MainWindow**

<Window x:Class="SpamBayesWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SpamBayesWPF"

mc:Ignorable="d"

Title="SpamBayes" Height="400" Width="800">

<!-- Главное окно приложения с заголовком и фиксированными размерами -->

<Grid Margin="20">

<!-- Основная сетка с отступами и двумя строками -->

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<!-- Для заголовка и кнопки выхода -->

<RowDefinition Height="\*"/>

<!-- Для основного контента -->

</Grid.RowDefinitions>

<!-- Заголовок и кнопка выхода -->

<StackPanel Orientation="Horizontal" Grid.Row="0" HorizontalAlignment="Right">

<!-- Горизонтальный контейнер для кнопки выхода -->

<Button x:Name="ExitButton" Content="Выход" Width="80" Height="30" Click="ExitButton\_Click" Background="#32CD32" Foreground="White" FontSize="14"/>

<!-- Кнопка выхода с зеленым фоном -->

</StackPanel>

<!-- Основной контент -->

<Grid Grid.Row="1">

<!-- Сетка с двумя колонками равной ширины -->

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<!-- Левая часть: Классификация -->

<StackPanel Grid.Column="0" Margin="0,0,10,0">

<!-- Контейнер для элементов классификации -->

<TextBlock Text="Введите текст сообщения для проверки" FontSize="14" Margin="0,0,0,10"/>

<!-- Подсказка для поля ввода -->

<TextBox x:Name="InputTextBox" Height="100" TextWrapping="Wrap" Margin="0,0,0,10"/>

<!-- Поле для ввода текста сообщения -->

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="0,0,0,10">

<!-- Контейнер для кнопок классификации -->

<Button x:Name="ClassifyButton" Content="Классифицировать" Width="150" Height="40" Click="ClassifyButton\_Click" Background="#1E90FF" Foreground="White" FontSize="14" Margin="0,0,10,0"/>

<!-- Кнопка для классификации текста -->

<Button x:Name="FetchEmailButton" Content="Получить последнее письмо" Width="193" Height="40" Click="FetchEmailButton\_Click" Background="#1E90FF" Foreground="White" FontSize="14"/>

<!-- Кнопка для получения последнего письма -->

</StackPanel>

<TextBlock x:Name="ResultLabel" Text="Не спам / Спам" FontSize="14" Margin="0,10,0,0"/>

<!-- Метка для отображения результата классификации -->

</StackPanel>

<!-- Правая часть: Добавление сообщений -->

<StackPanel Grid.Column="1" Margin="10,0,0,0">

<!-- Контейнер для элементов обучения -->

<TextBlock Text="Введите текст для обучения" FontSize="14" Margin="0,0,0,10"/>

<!-- Подсказка для поля ввода обучающего текста -->

<TextBox x:Name="TrainTextBox" Height="100" TextWrapping="Wrap" Margin="0,0,0,10"/>

<!-- Поле для ввода текста для обучения -->

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">

<!-- Контейнер для кнопок добавления -->

<Button x:Name="AddSpamButton" Content="В спам" Width="80" Height="40" Click="AddSpamButton\_Click" Background="#1E90FF" Foreground="White" FontSize="14" Margin="0,0,10,0"/>

<!-- Кнопка для добавления текста в спам -->

<Button x:Name="AddHamButton" Content="Не спам" Width="80" Height="40" Click="AddHamButton\_Click" Background="#1E90FF" Foreground="White" FontSize="14"/>

<!-- Кнопка для добавления текста в не-спам -->

</StackPanel>

<TextBlock x:Name="TrainResultLabel" Text="Готово" FontSize="14" Margin="0,10,0,0"/>

<!-- Метка для отображения результата добавления -->

</StackPanel>

</Grid>

</Grid>

</Window>

**Реализация интерфейса окна MainWindow**

using MailKit;

using MailKit.Net.Imap;

using MailKit.Net.Smtp;

using MimeKit;

using System.Windows;

namespace SpamBayesWPF

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

// Экземпляр классификатора спама

private SpamClassifier classifier;

// Имя текущего пользователя

private string currentUser;

// Email текущего пользователя

private string userEmail;

// Конструктор: инициализирует окно с данными пользователя

public MainWindow(string username, string email)

{

InitializeComponent();

classifier = new SpamClassifier();

currentUser = username;

userEmail = email;

}

// Обработчик клика по кнопке "Классифицировать"

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

{

// Получение текста из поля ввода

string inputMessage = InputTextBox.Text;

// Проверка на пустой ввод

if (string.IsNullOrWhiteSpace(inputMessage))

{

ResultLabel.Text = "Пожалуйста, введите сообщение.";

return;

}

// Классификация текста

string result = classifier.Classify(inputMessage);

ResultLabel.Text = $"Результат: {result}";

// Отправка результата на email пользователя

SendEmail(inputMessage, result);

}

// Обработчик клика по кнопке "Получить последнее письмо"

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

{

// Получение текста последнего письма

string lastEmail = FetchLastEmail();

if (lastEmail != null)

{

// Отображение текста письма в поле ввода

InputTextBox.Text = lastEmail;

// Классификация письма

string result = classifier.Classify(lastEmail);

ResultLabel.Text = $"Результат: {result}";

// Отправка результата на email пользователя

SendEmail(lastEmail, result);

}

}

// Получение последнего письма через IMAP

private string FetchLastEmail()

{

try

{

using (var client = new ImapClient())

{

// Подключение к Gmail через IMAP с SSL

client.Connect("imap.gmail.com", 993, MailKit.Security.SecureSocketOptions.SslOnConnect);

// Аутентификация

try

{

client.Authenticate("sarrikzhdyn@gmail.com", "bdvbxngqvkjdlzgj");

}

catch (MailKit.Security.AuthenticationException authEx)

{

MessageBox.Show($"Ошибка аутентификации: {authEx.Message}\nПроверьте email и пароль приложения.");

return null;

}

// Открытие папки "Входящие"

var inbox = client.Inbox;

inbox.Open(FolderAccess.ReadOnly);

// Проверка наличия писем

if (inbox.Count == 0)

{

MessageBox.Show("Входящих писем нет.");

return null;

}

// Получение последнего письма

var message = inbox.GetMessage(inbox.Count - 1);

// Извлечение текста (текстового или HTML)

string emailText = message.TextBody ?? message.HtmlBody;

if (string.IsNullOrWhiteSpace(emailText))

{

MessageBox.Show("Текст письма пустой.");

return null;

}

// Отключение от сервера

client.Disconnect(true);

return emailText;

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при получении письма: {ex.Message}\n{ex.StackTrace}");

return null;

}

}

// Отправка результата классификации на email

private void SendEmail(string message, string result)

{

try

{

// Создание сообщения

var emailMessage = new MimeMessage();

emailMessage.From.Add(new MailboxAddress("SpamBayes", "sarrikzhdyn@gmail.com"));

emailMessage.To.Add(new MailboxAddress(currentUser, userEmail));

emailMessage.Subject = "Результат классификации сообщения";

emailMessage.Body = new TextPart("plain")

{

Text = $"Сообщение: {message}\nРезультат: {result}"

};

using (var client = new SmtpClient())

{

// Подключение к Gmail через SMTP

client.Connect("smtp.gmail.com", 587, false);

// Аутентификация

client.Authenticate("sarrikzhdyn@gmail.com", "bdvbxngqvkjdlzgj");

// Отправка письма

client.Send(emailMessage);

// Отключение от сервера

client.Disconnect(true);

}

MessageBox.Show("Результат отправлен на ваш email!");

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при отправке email: {ex.Message}");

}

}

// Обработчик клика по кнопке "В спам"

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

{

// Добавление текста в файл spam.txt

string message = TrainTextBox.Text;

TrainResultLabel.Text = classifier.AddTextToFile("spam.txt", message);

// Очистка поля ввода

TrainTextBox.Text = "";

}

// Обработчик клика по кнопке "Не спам"

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

{

// Добавление текста в файл ham.txt

string message = TrainTextBox.Text;

TrainResultLabel.Text = classifier.AddTextToFile("ham.txt", message);

// Очистка поля ввода

TrainTextBox.Text = "";

}

// Обработчик клика по кнопке "Выход"

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

{

// Переход к окну входа

LoginWindow loginWindow = new LoginWindow();

loginWindow.Show();

this.Close();

}

}

}

Приложение Б  
Результаты тестирования

Таблица Б1 – Результата теста 1.1: Успешная регистрация пользователя.

|  |  |
| --- | --- |
| Вид тест | Регистрация пользователя |
| Что тестируем | Регистрация нового пользователя с корректными данными в окне |
| Входные данные | Имя пользователя: admin; Электронная почта: sarrikzhdyn@gmail.com; Пароль: 123. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и переключиться в окно регистрации, ввести данные для регистрации, нажать кнопку «Зарегистрироваться». |
| Ожидаемые выходные данные | Успешная регистрация с автоматическим входом, уведомление «Регистрация успешна! Теперь вы можете войти». |
| Ожидаемый результат | Пользователь успешно зарегистрирован и уведомлен об этом, после чего перенаправляется в окно LoginWindow. |
| Фактический результат | Уведомление получено, пользователь автоматически перенаправлен в окно входа. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б2 – Результата теста 1.2: Регистрация с уже существующим именем пользователя.

|  |  |
| --- | --- |
| Вид тест | Регистрация пользователя |
| Что тестируем | Регистрация пользователя с уже существующим именем |
| Входные данные | Имя пользователя: admin; Электронная почта: testuser@gmail.com; Пароль: newpassword. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и переключиться в окно регистрации, ввести данные для регистрации, нажать кнопку «Зарегистрироваться». |
| Продолжение таблицы Б2. | |
| Ожидаемые выходные данные | Уведомление «Пользователь с таким именем уже существует». |
| Ожидаемый результат | Регистрация не завершена, уведомление получено, пользователь остается в окне RegisterWindow. |
| Фактический результат | Регистрация не завершен, уведомление получено, пользователь остается в окне регистрации. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б3 – Результата теста 1.3: Регистрация с пустыми полями.

|  |  |
| --- | --- |
| Вид тест | Регистрация пользователя |
| Что тестируем | Регистрация пользователя с пустыми или некорректными полями |
| Входные данные | Незаполненные поля с программными подсказками «Введите имя пользователя», «Введите email», «Введите пароль». |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и переключиться в окно регистрации, нажать кнопку «Зарегистрироваться». |
| Ожидаемые выходные данные | Уведомление «Пожалуйста, заполните все поля». |
| Ожидаемый результат | Регистрация не выполнена, уведомление получено, пользователь остается в окне RegisterWindow. |
| Фактический результат | Регистрация не выполнена, уведомление получено, пользователь остается в окне регистрации. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б4 – Результата теста 2.1: Успешный вход с корректными данными.

|  |  |
| --- | --- |
| Вид тест | Вход в систему |
| Что тестируем | Вход с правильным именем пользователя и паролем |
| Входные данные | Имя пользователя: admin; Электронная почта: sarrikzhdyn@gmail.com; Пароль: 123. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и переключиться в окно входа в учетную запись, ввести корректные данные и нажать кнопку «Авторизоваться». |
| Ожидаемые выходные данные | Пользователь автоматически перенаправлен в главное окно приложения. |
| Ожидаемый результат | Вход успешно выполнен окно LoginWindow закрывается, пользователь автоматически перенаправляется в окно MainWindow. |
| Фактический результат | Вход успешно выполнен, окно входа в учетную запись закрывается, пользователь автоматически перенаправляется в главное окно приложения. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б5 – Результата теста 2.2: Вход с неверным паролем.

|  |  |
| --- | --- |
| Вид тест | Вход в систему |
| Что тестируем | Вход с неверным паролем в окне входа в учетную запись |
| Входные данные | Имя пользователя: admin; Электронная почта: sarrikzhdyn@gmail.com; Пароль: 12345. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
|  |  |
| Продолжение таблицы Б5. | |
| Шаги теста | Запустить программу и переключиться в окно входа в учетную запись, ввести данные и нажать кнопку «Авторизоваться». |
| Ожидаемые выходные данные | Уведомление «Неверный логин или пароль». |
| Ожидаемый результат | Вход не выполнен, уведомление получено, окно LoginWindow остается открытым. |
| Фактический результат | Вход не выполнен, уведомление получено, пользователь остается в окне входа в учетную запись. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б6 – Результата теста 2.3: Вход через Google.

|  |  |
| --- | --- |
| Вид тест | Вход в систему |
| Что тестируем | Вход через Google OAuth в окне входа в учетную запись |
| Входные данные | Учетные данные Google-аккаунта, настроенного для интеграции с приложением. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и переключиться в окно входа в учетную запись, нажать кнопку «Google», в открывшемся браузере выполняем вход в аккаунт и подтверждаем доступ для приложения. |
| Ожидаемые выходные данные | В файл users.json добавляется запись с именем и email. |
| Ожидаемый результат | Вход успешно выполнен, окно LoginWindow закрывается, пользователь автоматически перенаправляется в окно MainWindow. |
| Фактический результат | Вход успешно выполнен, окно входа в учетную запись закрывается, пользователь автоматически перенаправляется в главное окно приложения. |
| Окончание таблицы Б6. | |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б7 – Результата теста 3.1: Классификация спам-сообщения.

|  |  |
| --- | --- |
| Вид тест | Классификация спама |
| Что тестируем | Тестируется функция Classify класса SpamClassifier на спам-сообщении в главном окне приложения |
| Входные данные | Текст сообщения «You win a million dollars!». |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и войти в систему с существующими учетными данными, в главном окне приложения ввести в поле InputTextBox текст «You win a million dollars!» и нажать кнопку «Классифицировать». |
| Ожидаемые выходные данные | В ResultLabel отображается «Результат: Спам» и результат классификации отправляется на email. |
| Ожидаемый результат | Отображается «Результат: Спам», результат классификации отправляется на email, пользователь остается в окне MainWindow. |
| Фактический результат | Отображается «Результат: Спам», результат классификации отправлен на email, пользователь остается в главном окне приложения. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б8 – Результата теста 3.2: Классификация не спама.

|  |  |
| --- | --- |
| Вид тест | Классификация спама |
| Что тестируем | Тестируется функция Classify класса SpamClassifier на спам-сообщении в главном окне приложения |
| Входные данные | Текст сообщения «Hello, how are you?». |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и войти в систему с существующими учетными данными, в главном окне приложения ввести в поле InputTextBox текст «Hello, how are you?» и нажать кнопку «Классифицировать». |
| Ожидаемые выходные данные | В ResultLabel отображается «Результат: Не Спам» и результат классификации отправляется на email. |
| Ожидаемый результат | Отображается «Результат: Не Спам», результат классификации отправляется на email, пользователь остается в окне MainWindow. |
| Фактический результат | Отображается «Результат: Не Спам», результат классификации отправлен на email, пользователь остается в главном окне приложения. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б9 – Результата теста 3.3: Добавление спам-сообщения для обучения.

|  |  |
| --- | --- |
| Вид тест | Классификация спама |
| Что тестируем | Тестируется функция AddTextToFile для добавления спам-сообщения в файл spam.txt |
| Продолжение таблицы Б9. | |
| Входные данные | Текст сообщения «How to lose weight fast». |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и войти в систему с существующими учетными данными, в главном окне приложения ввести в поле TrainTextBox текст «How to lose weight fast» и нажать кнопку «В спам». |
| Ожидаемые выходные данные | В TrainResultLabel отображается сообщение «Сообщение успешно добавлено!» и текст добавлен в содержимое файла spam.txt. |
| Ожидаемый результат | Отображается сообщение «Сообщение успешно добавлено!», пользователь остается в окне MainWindow и текст добавлен в содержимое файла spam.txt. |
| Фактический результат | Отображается сообщение «Сообщение успешно добавлено!», пользователь остается в главном окне приложения и текст добавлен в содержимое файла spam.txt. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б10 – Результата теста 4.1: Получение последнего письма.

|  |  |
| --- | --- |
| Вид тест | Получение и отправка писем |
| Что тестируем | Тестируется функция FetchLastEmail для получения последнего письма через IMAP в главном окне приложения |
| Входные данные | Авторизованный аккаунт Gmail. |
|  | |
| Продолжение таблицы Б10. | |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и войти в систему с существующими учетными данными, в главном окне приложения нажать кнопку «Получить последнее письмо». |
| Ожидаемые выходные данные | В InputTextBox отображается текст последнего письма, в ResultLabel показан результат классификации письма, результат отправляется на email пользователя. |
| Ожидаемый результат | Отображается «Результат: Спам/Не Спам», результат классификации отправляется на email, пользователь остается в окне MainWindow. |
| Фактический результат | Отображается «Результат: Спам/Не Спам», результат классификации отправлен на email, пользователь остается в главном окне приложения. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б11 – Результата теста 4.2: Отправка результата классификации на email.

|  |  |
| --- | --- |
| Вид тест | Получение и отправка писем |
| Что тестируем | Тестируется функция SendEmail для отправки результатов классификации на email пользователя |
| Входные данные | Авторизованный аккаунт Gmail. |
|  | |
| Продолжение таблицы Б11. | |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | Запустить программу и войти в систему с существующими учетными данными, в главном окне приложения ввести текст и нажать кнопку «Классифицировать». |
| Ожидаемые выходные данные | Письмо доставлено на email пользователя с темой «Результат классификации сообщения», тело письма содержит текст сообщения и результат классификации, отображается уведомление «Результат отправлен на ваш email!». |
| Ожидаемый результат | Отображается уведомление «Результат отправлен на ваш email!», письмо доставлено на email пользователя с темой «Результат классификации сообщения». |
| Фактический результат | Отображается уведомление «Результат отправлен на ваш email!», письмо доставлено на email с темой «Результат классификации сообщения». |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б12 – Результата теста 5.1: Навигация через кнопку «Справка».

|  |  |
| --- | --- |
| Вид тест | Интерфейс и навигация |
| Что тестируем | Тестируется функциональность перехода в окно HelpWindow через кнопку «Справка» |
| Входные данные | Папка с проектом и ресурсами, готовый к запуску файл программы. |
|  | |
| Продолжение таблицы Б12. | |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | В MainEntryWindow нажать кнопку «Справка» и проверить, открылось ли окно HelpWindow. Затем нажать кнопку «Назад» и проверить, вернулось ли приложение в MainEntryWindow. |
| Ожидаемые выходные данные | Письмо доставлено на email пользователя с темой «Результат классификации сообщения», тело письма содержит текст сообщения и результат классификации, отображается уведомление «Результат отправлен на ваш email!». |
| Ожидаемый результат | Окно HelpWindow открывается и отображает текст справки, после нажатия кнопки «Назад» открывается MainEntryWindow. |
| Фактический результат | Окно HelpWindow открывается и отображает текст справки, после нажатия кнопки «Назад» открывается MainEntryWindow. |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Таблица Б13 – Результата теста 6: Active Testing.

|  |  |
| --- | --- |
| Вид тест | Active Testing |
| Что тестируем | Классификация тестовых сообщений на спам/не спам в сравнении с эталонным классификатором |
| Входные данные | Набор заранее размеченных сообщений (50 шт.), включая граничные случаи. |
|  | |
| Продолжение таблицы Б13. | |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Ryzen 5 4600H 2 ядра, частота 2.00 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта GTX 1650; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | 1. Провести классификацию входных сообщений с помощью эталонного внешнего сервиса. 2. Провести классификацию тех же сообщений в SpamBayes. 3. Сравнить результаты. |
| Ожидаемые выходные данные | Совпадение классификаций на уровне не менее 90%. |
| Ожидаемый результат | Минимум 45 из 50 сообщений классифицированы одинаково |
| Фактический результат | Совпадение – 92%, 46 из 50 сообщений классифицированы одинаково. |
| Обнаруженный дефект | 4 случая расхождения с эталоном. |
| Возможная причина дефекта | Недостаток обучающих данных для некоторых граничных формулировок. |
| Предполагаемое решение | Добавить спорные примеры в обучающую выборку и переобучить классификатор. |

Таблица Б14 – Результата теста 7: Age Testing.

|  |  |
| --- | --- |
| Вид тест | Age Testing |
| Что тестируем | Производительность и точность классификатора при росте объема обучающей выборки |
| Входные данные | Пакеты сообщений по 10, 100, 1000, 5000 шт. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор AMD ryzen 5 7535hs 2 ядра, частота 2.00 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Nvidia Geforce RTX 4060 laptop; * дисплей 1920\*1080; * Windows 11; |
| Шаги теста | 1. Постепенно увеличивать объем обучающей выборки. 2. Измерять точность классификации и среднее время отклика. |
| Ожидаемые выходные данные | Увеличение времени классификации с ростом данных.  Точность сохраняется на высоком уровне (90%). |
| Ожидаемый результат | Время – 2 сек на сообщение, точность не падает ниже 90%. |
| Фактический результат | Время на классификацию от 0.2 до 1.8 сек.  Точность 91.2% при 5000 сообщениях. |
| Обнаруженный дефект | Замедление после 3000 сообщений. |
| Возможная причина дефекта | Рост объема данных в ОЗУ |
| Предполагаемое решение | Переход на потоковую загрузку, оптимизация алгоритма хранения. |

Таблица Б15 – Результата теста 8: Benchmark Testing.

|  |  |
| --- | --- |
| Вид тест | Benchmark Testing |
| Что тестируем | Производительность классификатора по метрикам: precision, recall, среднее время обработки |
| Входные данные | 500 эталонных сообщений (спам и не спам). |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор Intel i5-12450H 8 ядер, частота 4.4 ГГц, кэш 6 МБ; * ОЗУ 16 Гб; * HDD 512 Гб; * видеокарта Intel(R) UHD Graphics; * дисплей 1920\*1200; * Windows 11; |
| Шаги теста | 1. Классификация сообщений без изменения обучающей базы. 2. Измерение времени, precision, recall. |
| Ожидаемые выходные данные | Время обработки – 1 сек на сообщение  Precision ≥ 90%, Recall ≥ 85% |
| Ожидаемый результат | Классификация с высокой точностью и скоростью. |
| Фактический результат | Precision – 93.5%  Recall – 88.2%  Среднее время – 0.45 сек |
| Обнаруженный дефект | Дефектов не обнаружено |
| Возможная причина дефекта | Отсутствует |
| Предполагаемое решение | Отсутствует |

Приложение В  
Министерство образования Республики Беларусь  
Оршанский колледж учреждения образования  
«Витебский государственный университет имени П.М. Машерова»  
  
  
  
  
  
  
  
Документация пользователя

2025

**Оглавление**

1 Назначение программного модуля

2 Обозначение пакета

3 Компоненты пакета. Поставляемые элементы

4 Функциональное описание программного средства

4.1 Назначение и область применения

4.2 Условия эксплуатации

4.3 Характеристики программы

4.4 Защита информации

5 Ввод в действие программного средства

6 Использование программного средства

7 Тестирование

**1 Назначение программного модуля**

Программное средство «SpamBayes» разработано для автоматической классификации текстовых сообщений на категории «Спам» и «Не спам» с использованием наивного байесовского метода. Основная цель приложения заключается в защите пользователей от нежелательных сообщений, таких как мошеннические письма, реклама и вредоносный контент, поступающих через электронную почту или другие платформы обмена сообщениями. Программа позволяет автоматически определять категорию сообщений на основе их содержимого, обучать классификатор на пользовательских данных, добавляя новые сообщения в обучающие файлы, интегрироваться с почтовыми сервисами, такими как Gmail, для обработки реальных писем, а также предоставляет удобный графический интерфейс для взаимодействия с пользователем.

Приложение предназначено для локального использования на персональных компьютерах. Оно подходит для различных сфер, включая личное использование, корпоративные сети, образовательные учреждения, финансовый сектор и медицинские организации, где помогает повысить безопасность и удобство работы с почтой.

**2 Обозначение пакета**

Программа носит название «SpamBayes», что отражает использование байесовского метода в основе алгоритма классификации. Название отображается в заголовках окон приложения и на главном экране. Иконка приложения представляет собой стилизованный символ фильтра, выполненный в сине-зеленой цветовой гамме. Этот символ ассоциируется с процессом фильтрации спама и подчеркивает надежность и безопасность приложения.



Рисунок 2.1 – Иконка приложения

**3 Компоненты пакета. Поставляемые элементы**

Скомпилированное приложение «SpamBayes» поставляется в виде папки, содержащей необходимые файлы и ресурсы. Основной исполняемый файл называется SpamBayes.exe и создан в среде Visual Studio 2022 с использованием .NET Framework 4.7. Для хранения примеров сообщений используются два текстовых файла в кодировке UTF-8: spam.txt для примеров спама и ham.txt для примеров не-спама. Эти файлы изначально могут быть пустыми. Данные пользователей, такие как имя, email и пароль, сохраняются в файле users.json в формате JSON, который создается автоматически при первом запуске приложения.

Папка Resources содержит графические ресурсы: файл icon.png с иконкой приложения, mail.png и check.png для оформления окна авторизации. Для работы с внешними сервисами и данными приложение включает библиотеки Google.Apis.Gmail.v1.dll для интеграции с Gmail API, MailKit.dll для работы с протоколами IMAP и SMTP, Newtonsoft.Json.dll для обработки JSON-данных и System.Windows.Interactivity.dll для поддержки архитектуры MVVM в WPF. Все файлы должны оставаться в одной директории, а ресурсы в подкаталоге Resources, чтобы приложение работало корректно.

**4 Функциональное описание программного средства**

**4.1 Назначение и область применения**

Приложение «SpamBayes» создано для автоматической фильтрации спама в текстовых сообщениях, поступающих через электронную почту или введенных пользователем вручную. Оно использует наивный байесовский классификатор, чтобы анализировать содержимое сообщений и определять их категорию – «Спам» или «Не спам». Программа позволяет классифицировать сообщения, добавлять новые данные в обучающие файлы spam.txt и ham.txt через графический интерфейс, интегрироваться с Gmail для обработки писем и хранить данные локально.

Приложение подходит для личного использования, чтобы защитить почтовые ящики от нежелательных сообщений, а также для корпоративных сетей, где повышает безопасность и продуктивность. Оно применимо в образовательных учреждениях для обеспечения безопасной коммуникации, в финансовом секторе для защиты от фишинговых атак и в медицинских организациях для предотвращения утечек данных и мошеннических предложений.

**4.2 Условия эксплуатации**

Для работы приложения «SpamBayes» требуется компьютер с операционной системой Windows 7 или более поздней версии Windows 8, 10, 11. Также необходим .NET Framework версии 4.5 или выше. Минимальные требования к процессору включают любой процессор, совместимый с Windows, но рекомендуется двухъядерный процессор с частотой 2 ГГц или выше. Объем оперативной памяти должен составлять не менее 2 ГБ, хотя для стабильной работы лучше использовать 4 ГБ или больше. Для хранения приложения и обучающих файлов требуется минимум 10 МБ свободного места на диске.

Если используется интеграция с Gmail, необходимо стабильное интернет-соединение со скоростью не менее 1 Мбит/с. Для работы с Gmail требуется включить доступ по протоколам IMAP и SMTP в настройках почтового аккаунта и создать пароль приложения. Для авторизации через OAuth 2.0 (Google) необходимо наличие браузера. Минимальное разрешение экрана составляет 1024x768, чтобы интерфейс отображался корректно. Дополнительные драйверы баз данных не требуются, так как приложение использует локальные текстовые файлы и JSON.

**4.3 Характеристики программы**

Программа «SpamBayes» написана на языке программирования C# в среде разработки Visual Studio 2022 версии 17.8.3 с использованием .NET Framework 4.7. Интерфейс реализован на основе Windows Presentation Foundation (WPF) с использованием XAML. Приложение включает пять форм: главное окно входа MainEntryWindow, окно авторизации LoginWindow, окно регистрации RegisterWindow, окно справки HelpWindow и основное окно для классификации и обучения MainWindow.

Основные классы программы включают SpamClassifier для реализации байесовского классификатора, UserManager для управления пользователями и классы для каждого окна. Разработка, включая проектирование, программирование, тестирование и документацию, заняла около 150 часов. Размер скомпилированного приложения составляет примерно 142 КБ без учета обучающих данных, объем которых зависит от активности пользователя.

**4.4 Защита информации**

Для защиты данных пользователей и предотвращения несанкционированного доступа приложение «SpamBayes» реализует несколько мер безопасности. Локальная авторизация требует ввода имени и пароля, которые хранятся в файле users.json в формате JSON, доступ к которому ограничен правами операционной системы. Внешняя авторизация через Google использует протокол OAuth 2.0, что исключает необходимость хранения паролей внешних сервисов в приложении. Токены доступа обрабатываются безопасно через библиотеки Google.Apis.

Валидация пользовательского ввода проверяет заполненность полей при регистрации и авторизации, а также непустоту текстовых сообщений при классификации или обучении, что предотвращает ошибки и некорректные данные. Операции с файлами spam.txt, ham.txt, users.json и сетевые запросы сопровождаются обработкой исключений, исключая аварийное завершение программы.

**5 Ввод в действие программного средства**

Чтобы установить и запустить приложение «SpamBayes», выполните следующие шаги.

Сначала скопируйте папку с приложением «SpamBayes» на компьютер. Папка должна содержать исполняемый файл SpamBayes.exe, файлы spam.txt, ham.txt, users.json, папку Resources с графическими файлами и необходимые библиотеки. Убедитесь, что все файлы находятся в правильной структуре директорий.

Проверьте, что на компьютере установлена операционная система Windows 7 или новее. Убедитесь, что установлен .NET Framework 4.5 или выше. Для этого откройте «Панель управления», выберите «Программы и компоненты» и найдите Microsoft .NET Framework 4.5 или более позднюю версию. Если фреймворк отсутствует, скачайте и установите его с официального сайта Microsoft.

Если вы планируете использовать интеграцию с Gmail, войдите в аккаунт Google, который будет использоваться. В настройках Gmail включите доступ по протоколам IMAP и SMTP: перейдите в «Настройки», выберите раздел «Пересылка и POP/IMAP» и активируйте соответствующие опции. Затем создайте пароль приложения: в «Настройках аккаунта Google» выберите «Безопасность», включите двухэтапную аутентификацию, перейдите в «Пароли приложений» и создайте пароль для «SpamBayes». Сохраните этот пароль.

Для запуска приложения дважды щелкните по файлу SpamBayes.exe. При первом запуске откроется начальное окно входа, показанное на рисунке 5.1.

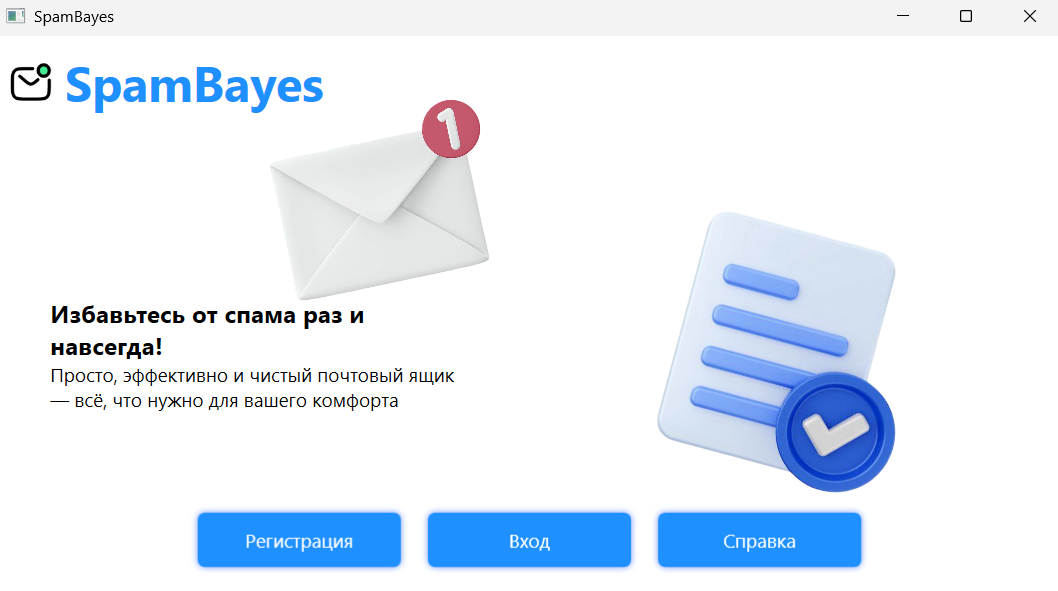


Рисунок 5.1 – Начальное окно входа

В начальном окне выберите «Регистрация» для создания новой учетной записи или «Вход» для авторизации. В окне регистрации, показанном на рисунке 5.2, введите имя пользователя, email и пароль, затем нажмите «Зарегистрироваться». В окне авторизации, показанном на рисунке 5.3, введите имя и пароль или используйте авторизацию через Google. После успешной авторизации откроется главное окно приложения, показанное на рисунке 5.4.

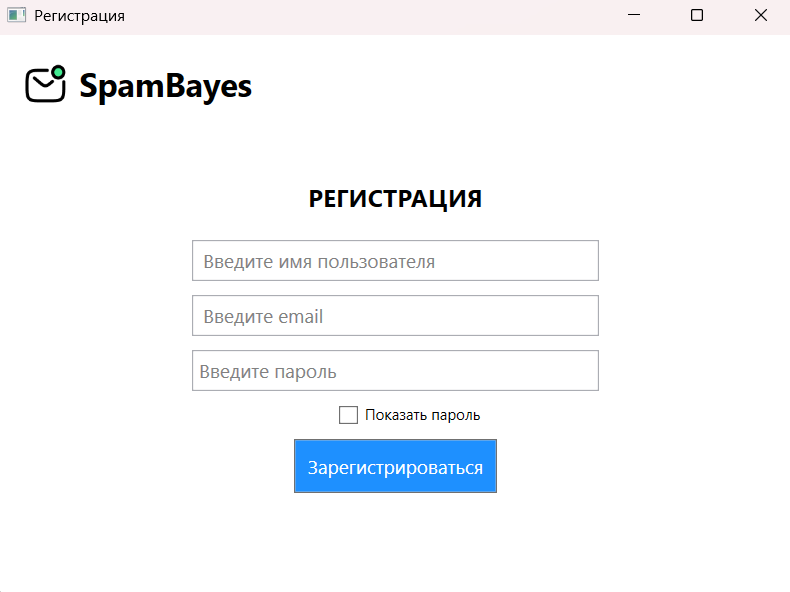


Рисунок 5.2 – Окно регистрации

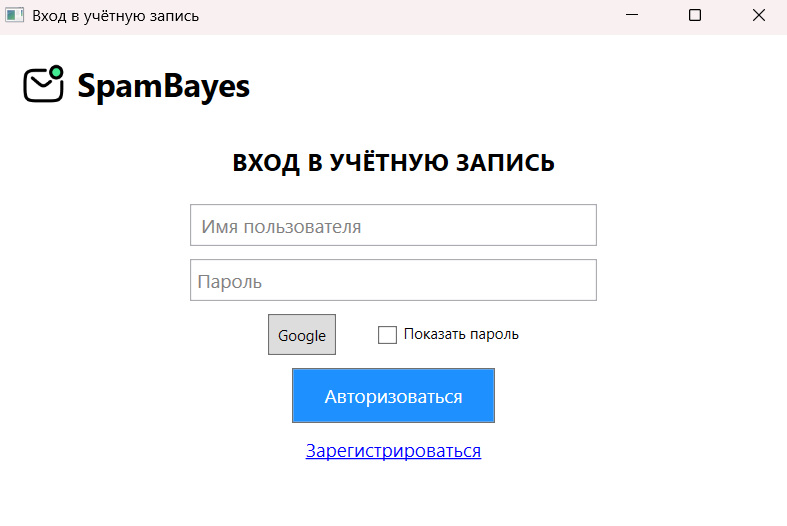


Рисунок 5.3 – Окно авторизации

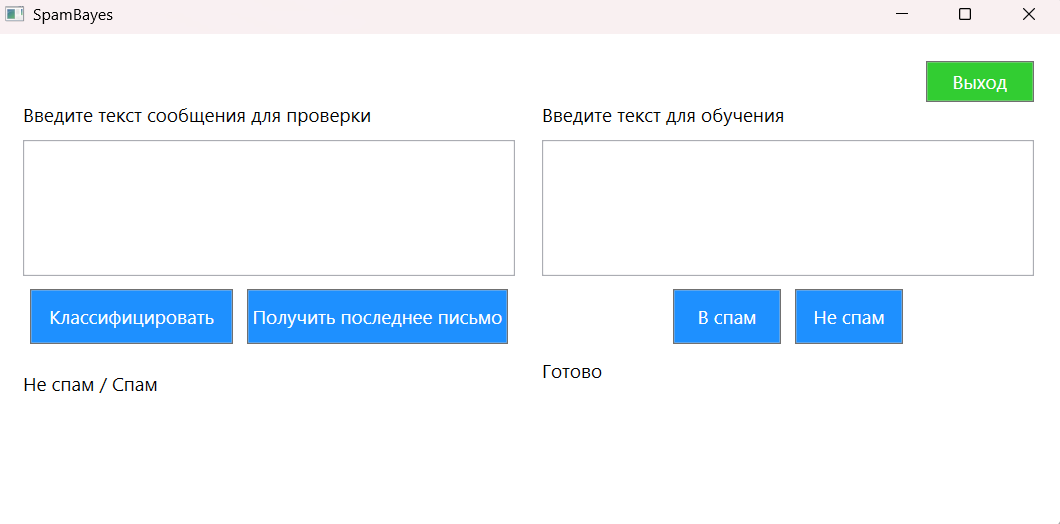


Рисунок 5.4 – Главное окно приложения

**6 Использование программного средства**

Для работы с приложением «SpamBayes» следуйте этим инструкциям.

Запустите приложение, дважды щелкнув по файлу SpamBayes.exe. В главном окне входа вы можете выбрать регистрацию, вход или просмотр справки. Для регистрации заполните поля имени пользователя, email и пароля в окне регистрации и нажмите «Зарегистрироваться». После успешной регистрации вы будете перенаправлены в окно входа. Для авторизации введите имя пользователя и пароль или используйте кнопку «Google» для входа через Google. Окно справки, показанное на рисунке 6.1, содержит инструкции по использованию приложения и доступно по кнопке «Справка».

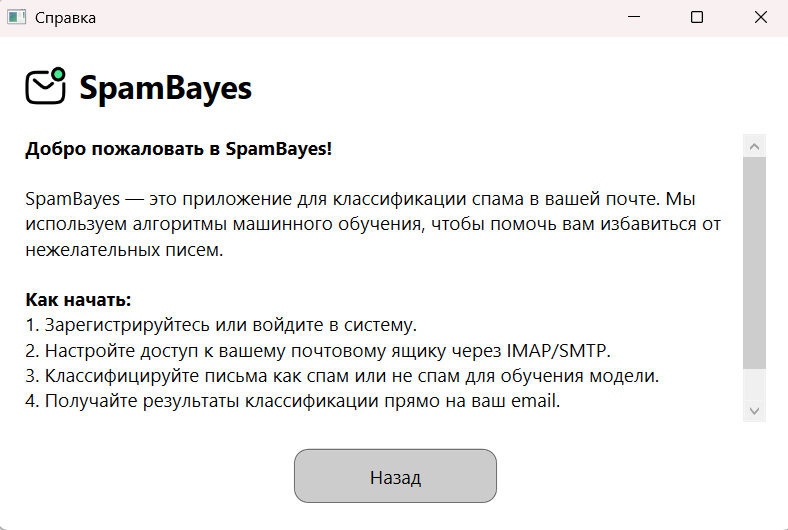


Рисунок 6.1 – Окно справки

В основном окне приложения введите текст сообщения в левое поле ввода для классификации. Нажмите кнопку «Классифицировать», чтобы определить, является ли сообщение спамом. Результат отобразится под полем ввода. Чтобы загрузить последнее письмо из Gmail, нажмите «Получить последнее письмо». Текст письма появится в поле ввода, а результат классификации будет отправлен на ваш email. Пример классификации показан на рисунке 6.2.

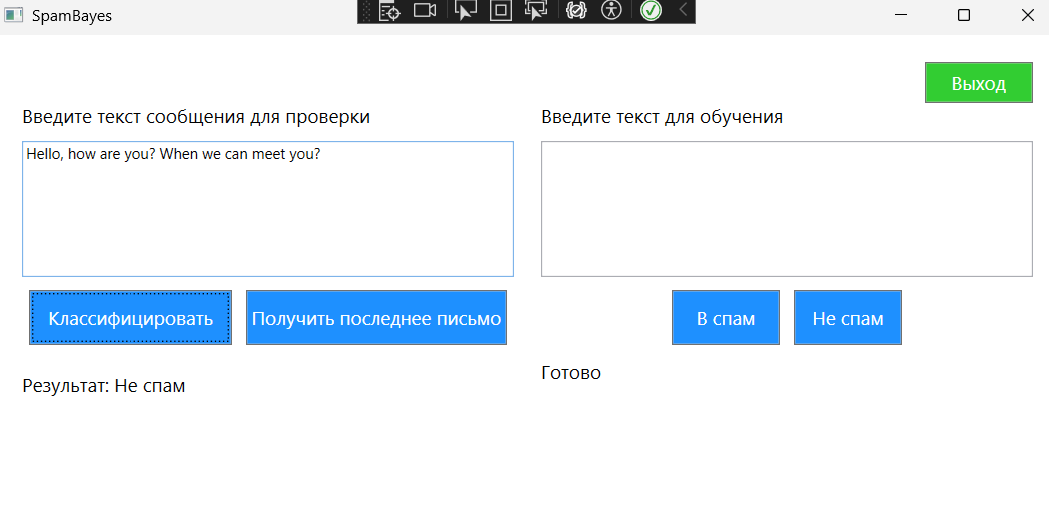


Рисунок 6.2 – Пример классификации сообщения

Для обучения классификатора введите текст в правое поле ввода и выберите «В спам» или «Не спам», чтобы добавить сообщение в соответствующий файл. Результат операции отобразится под полем ввода. Пример добавления сообщения показан на рисунке 6.3.

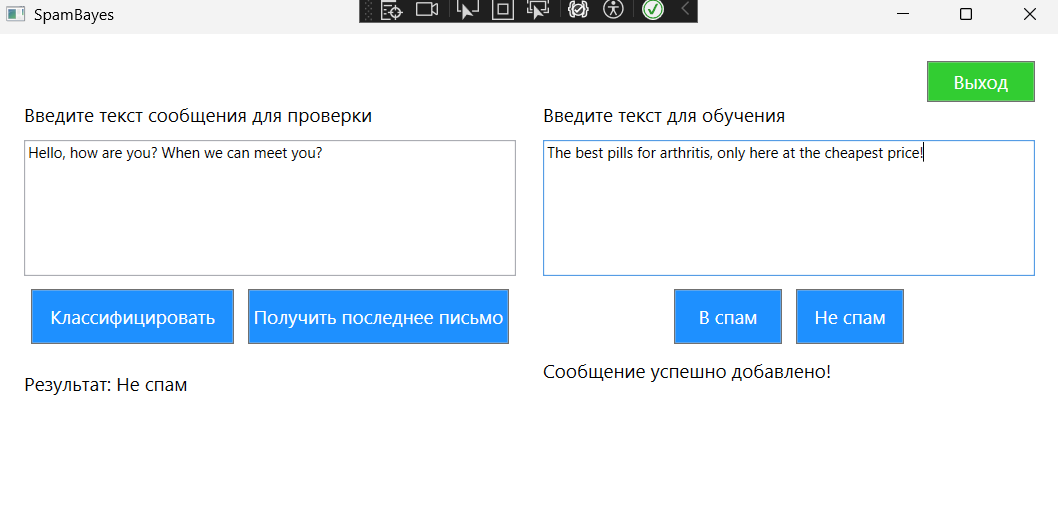


Рисунок 6.3 – Пример добавления сообщения в обучающую базу

Чтобы выйти из приложения, нажмите кнопку «Выход» в правом верхнем углу основного окна. Это вернет вас в окно входа.

**7 Тестирование**

Программа «SpamBayes» прошла тест удачной эксплуатации на компьютере со следующими характеристиками:

* процессор Intel i5-12450H 8 ядра с частотой 4.4 ГГц;
* оперативная память 16 ГБ DDR4, 3200 МГц;
* жёсткий диск NVMe 512 ГБ SSD;
* видеокарта Intel(R) UHD Graphics;
* операционная система Windows 11;
* фреймворк .Net Framework 4.7.