**Практическое занятие № 13. Использование регулярных выражений**

**1 Цель занятия**

Получить практические навыки использования регулярных выражений для поиска соответствий в строках

**2 Перечень оборудования и программного обеспечения**

Персональный компьютер

Microsoft Office (Word)

Microsoft Visual Studio

**3 Краткие теоретические сведения**

**3.1 Регулярные выражения** – это один из способов поиска подстрок (соответствий) в строках. Осуществляется поиск соответствий с помощью просмотра строки в поисках некоторого шаблона. Поддержка регулярных выражений в .Net выполняется классами пространства имен System.Text.RegularExpressions.

Основные классы:

Regex – постоянное регулярное выражение с заданным шаблоном;

MatchCollection – предоставляет набор успешных соответствий при итеративном применении шаблона регулярного выражения к строке;

Match – предоставляет отдельное успешное совпадение с шаблоном регулярного выражения;

GroupCollection – предоставляет коллекцию найденных групп и возвращает набор групп как одно соответствие;

Group – предоставляет результаты для одной регулярной группы;

CaptureCollection – предоставляет последовательность найденных подстрок и возвращает наборы соответствий отдельно для каждой группы;

Capture – предоставляет результаты отдельного захвата подвыражения.

Основные методы класса Regex:

Matches() – ищет в указанной строке все вхождения шаблона регулярного выражения;

IsMatch() – определяет, обнаружено ли в указанной строке соответствие шаблону регулярного выражения.

Для того чтобы использовать Regex в своих программах необходимо в список используемых пространств имен добавить:

**using System.Text.RegularExpression;**

Далее, в коде самой программы необходимо создать экземпляр Regex:



Все найденные соответствия в тексте будут сохраняться в коллекцию типа MatchCollection:



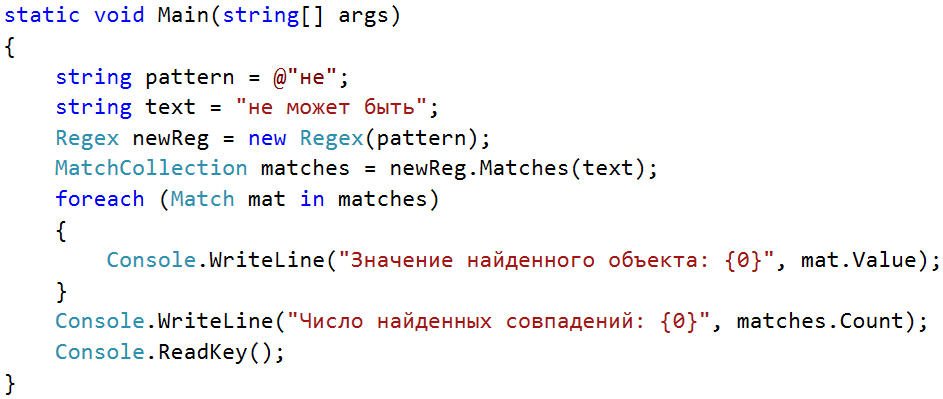
Для поиска в тексте соответствий шаблону регулярного выражения необходимо использовать метод Regex.Matches():



В результате в matches появляются все результаты парсинга (вычленения нужной информации). Мы можем посмотреть, сколько таких результатов (matches.Count), и можем узнать значение конкретного найденного элемента (matches[N].Value).

Значение pattern – образец или шаблон для поиска, например, если необходимо найти слово "не" в строке "не может быть", то pattern для поиска будет выглядеть так: @"не". В простейшем случае, можно обойтись без шаблона поиска, тогда найдено будет значение точно повторяющее pattern. Можно получить как значение найденного совпадения, так и их количество, как видно из примера ниже.

**Пример 1:**

****



Теперь остановимся на условиях поиска, или option. Для начала необходимо создать условие поиска:

RegexOptions option = RegexOptions.<условие поиска>;

Ниже приведены условия поиска.

IgnoreCase – находит совпадения независимо от регистра, т.е. прописными или строчными буквами в строке написано слово.

IgnorePatternWhitespace – устраняет из шаблона неизбежные пробелы и включает комментарии помеченные «#».

Compiled – указывает что регулярное выражение скомпилировано в сборку. Это порождает более быстрое исполнение, но увеличивает время запуска.

CultureInvariant – указывает игнорирование региональных языковых различий.

ExplicitCapture – указывает, что единственные допустимые записи являются явно поименованными или пронумерованными группами в форме (?<name>…).

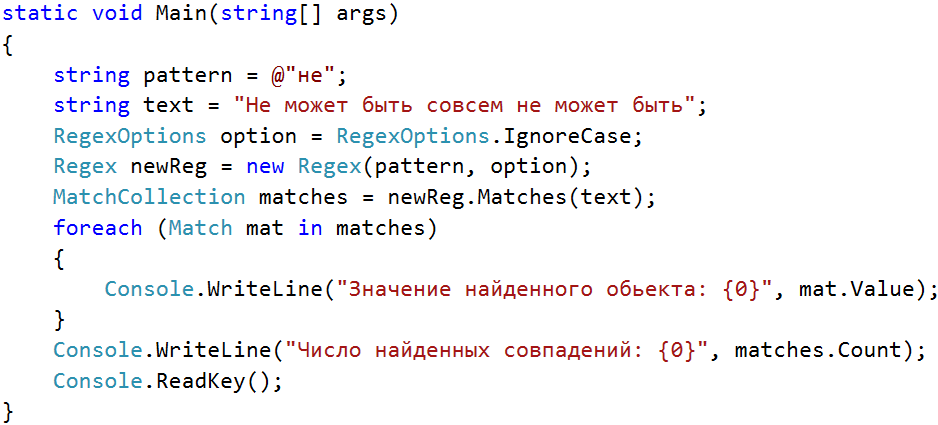
Multiline – многострочный режим. Изменяет значения символов «^» и «$» так что они совпадают соответственно в начале и конце каждой строки, а не только в начале и конце всего текста.

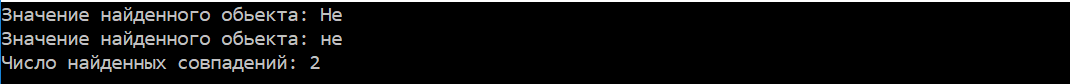
RightToLeft – указывает что поиск будет выполнен справа на лево, а не слева на право.

Singleline – однострочный режим.

None – указывает на отсутствие заданных параметров.

**Пример 2:**





В этом случае будет осуществлен поиск слова "не" не зависимо от того, какими буквами (прописными или строчными) оно написано. И в результате будут найдены оба слова в строке. При желании и опыте, код может быть оптимизирован.

Используя специальные символы можно создавать более сложные шаблоны для поиска:

^ – указывает на то, что поиск должен производиться в начале строки, например шаблон @"^не" найдет "не" в строке, если она начинается с него: "Не может быть совсем не может быть не".

$ – указывает на то что поиск должен производиться в конце строки, шаблон @"не$" найдет "не" в той же самой строке, только если она заканчивается им.

{n} – указывает точное число вхождений в строку, например шаблон @"не{2}" найдет слово "нее" в строке.

{n,} – указывает число вхождений не менее n, т.е. шаблон @"не{2,}" найдет слова "нее", "неее" и т.д.

{n,m} – указывает число вхождений – n и количество символов вхождения – m, то есть шаблон @"не{2,4}" найдет все слова, где количество букв "е" больше 2, но выберет только количество букв "е" равное 4. Т.е. в слове "нееееееееее" он найдет только "нееее".

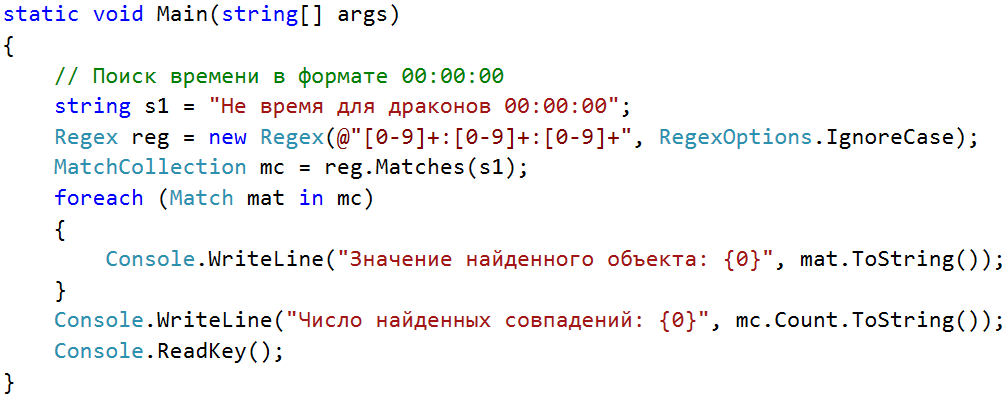
+ – соответствует 1 или более предшествующих выражений. Например, "не+" соответствует "не" и "нее", но не соответствует "н".

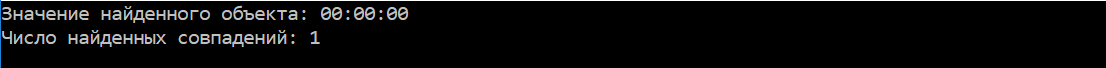
\* – Соответствует 0 или более вхождений предшествующего выражения. Например, "не\*" соответствует "н" и "нее".

? – Соответствует 0 или 1 предшествующих выражений. Например, "бы(ло)?" соответствует "бы" в строке "бы" или "было".

В квадратных скобках можно указать диапазон букв или цифр для поиска, например [A-Z] или [0-9].

**Пример 3:**



****

**Задание.** Дан текстовый файл, состоящий из нескольких строк. Проверить является ли каждая из них электронным адресом. Выделить найденные электронные адреса цветом и выдать сообщение о результате проверки.

Строка электронного адреса не должна содержать пробелов и в общем виде имеет следующий формат: логин@почтовый\_домен, где логин – идентификатор пользователя, уникальный в пределах одного почтового домена. Логин должен состоять из латинских букв (строчных или прописных) и/или цифр; в логине можно использовать символы подчеркивания, минуса и точки; логин должен начинаться только с буквы или цифры. Имя почтового домена состоит из нескольких частей, которые разделяют точками. Последняя часть имени домена должна состоять только из латинских букв (от 2 до 9), предыдущие части могут содержать цифры и дефис (дефис не может быть в начале или конце части):

группа символов1 группа символов2 группа символов3

^[A-Za-z0-9](([\_\.\-]?[a-zA-Z0-9]+)\*)@[A-Za-z0-9](([\.\-]?[a-zA-Z0-9]+)\*)\.([A-Za-z]{2,9})$

Адрес (логин) начинается с цифры или латинской буквы

Группа может начинаться с одного символа \_ . -

Адрес заканчивается этой группой символов

В группе может быть от 2 до 9 латинских букв

Группа может начинаться с одного символа . -

Группа может входить в строку ноль или более раз

Группа содержит одну или более латинскую букву или цифру

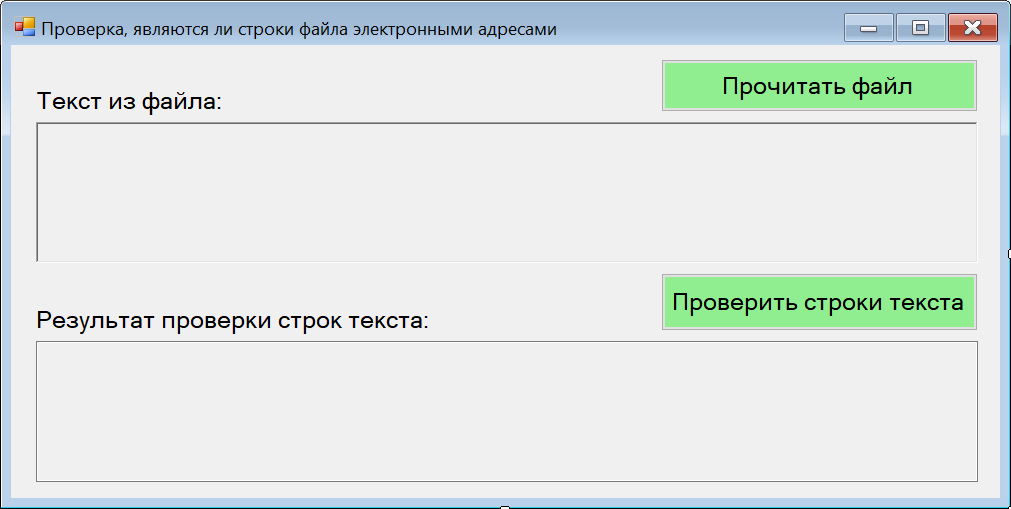
Почтовый домен начинается с цифры или латинской буквы

Группа может входить в строку ноль или более раз

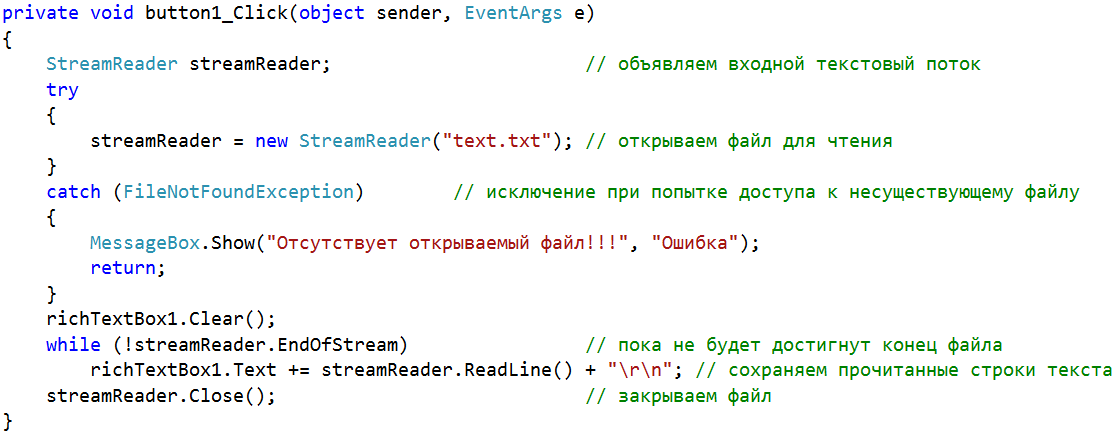
Группа содержит одну или более латинскую букву или цифру

**Решение.**

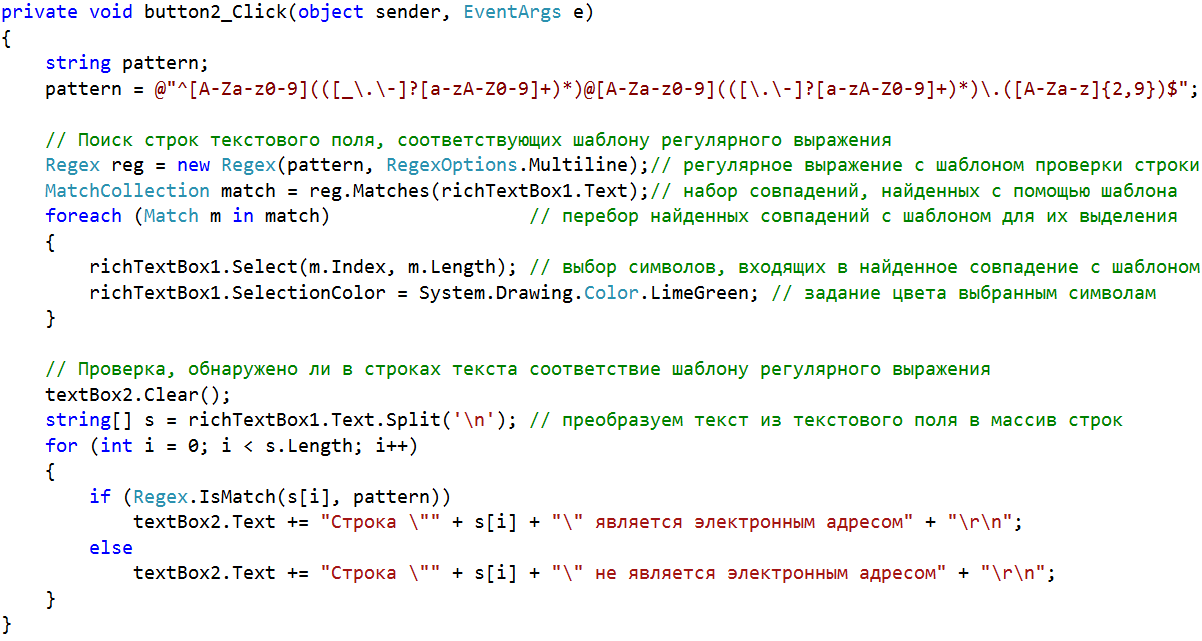
а) Интерфейс приложения Windows Forms:



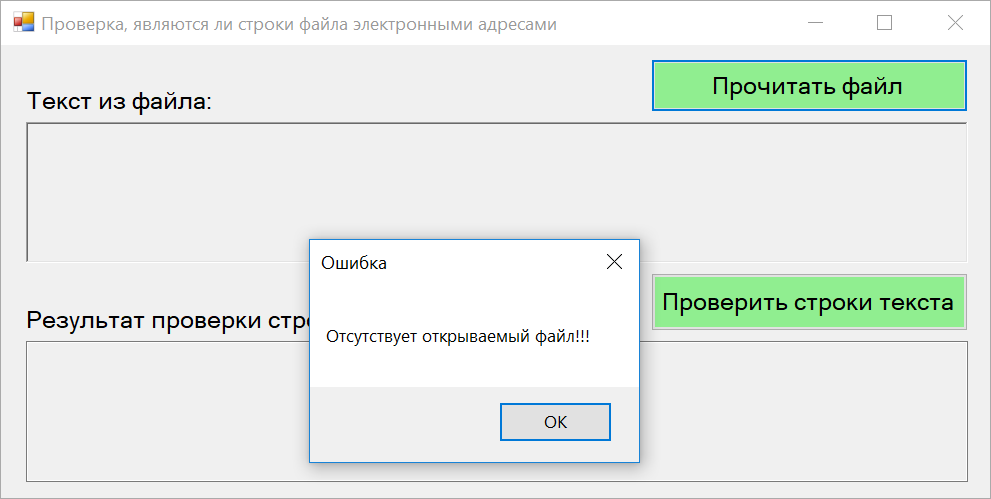
Код, выполняемый при щелчке на кнопку «Прочитать файл»:

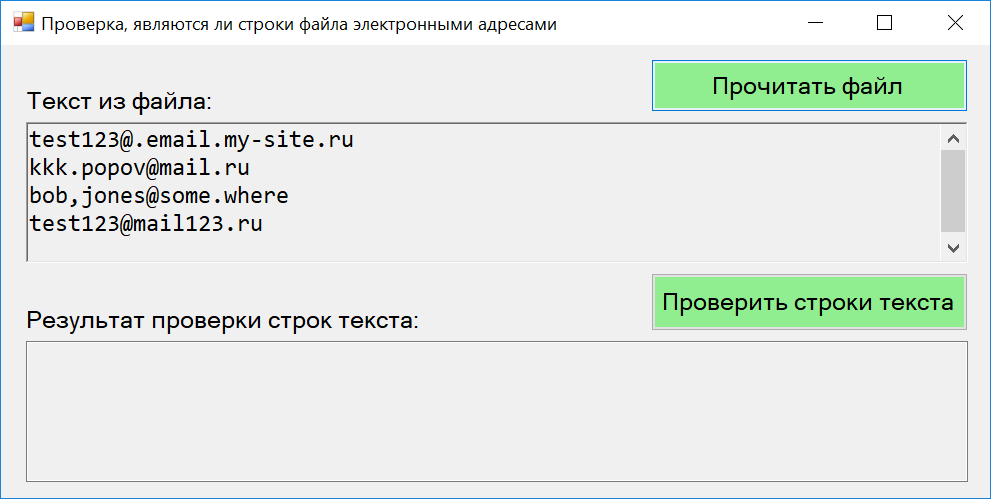


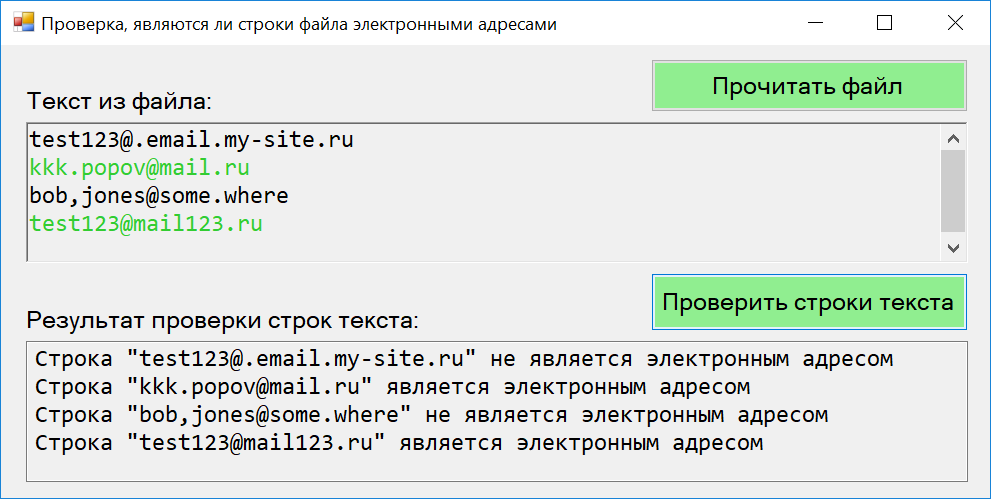
Код, выполняемый при щелчке на кнопку «Проверить строки текста»:



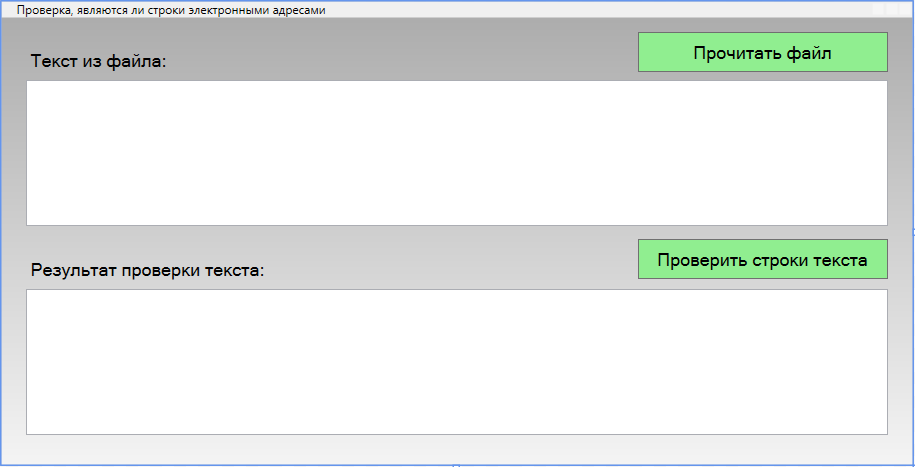
Скрины отлаженной программы:



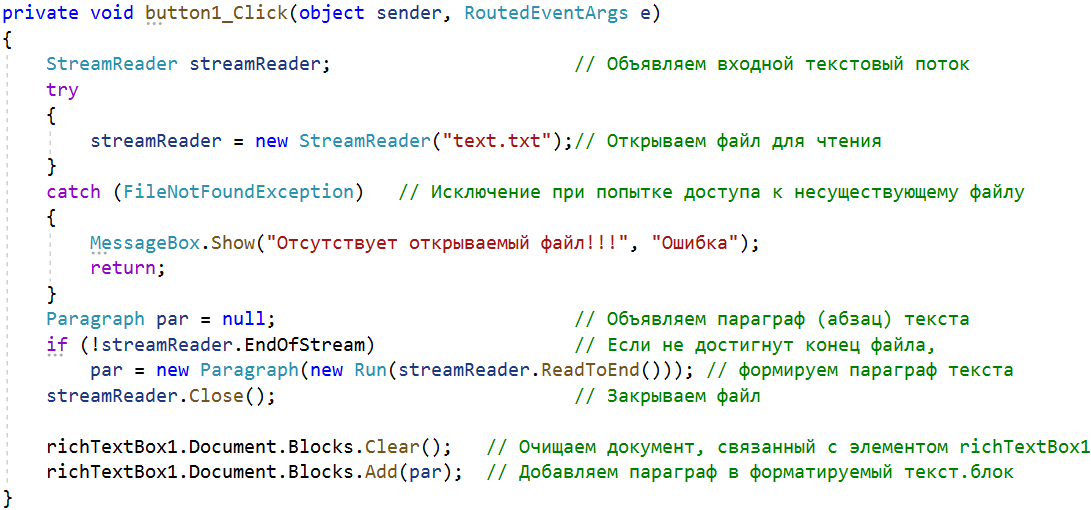




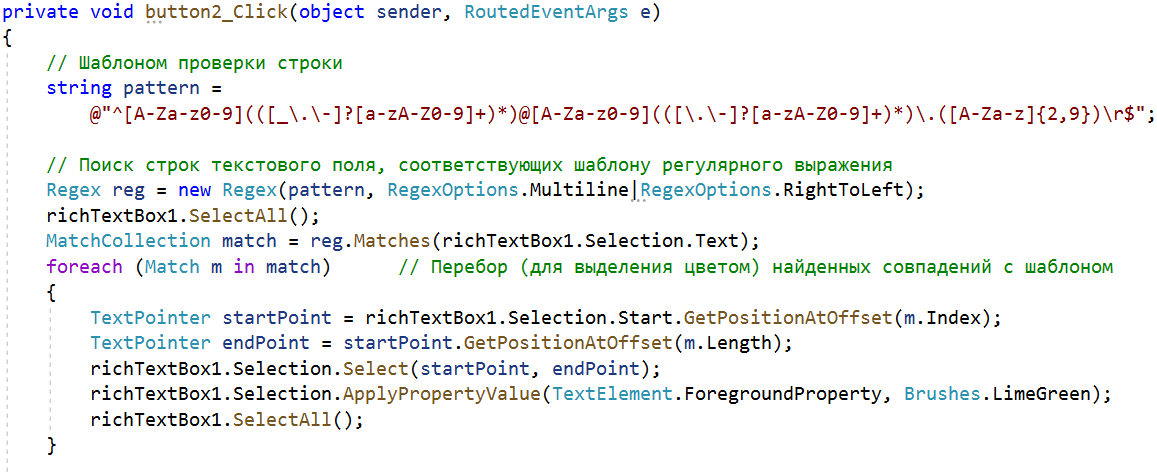
б) Интерфейс приложения WPF:

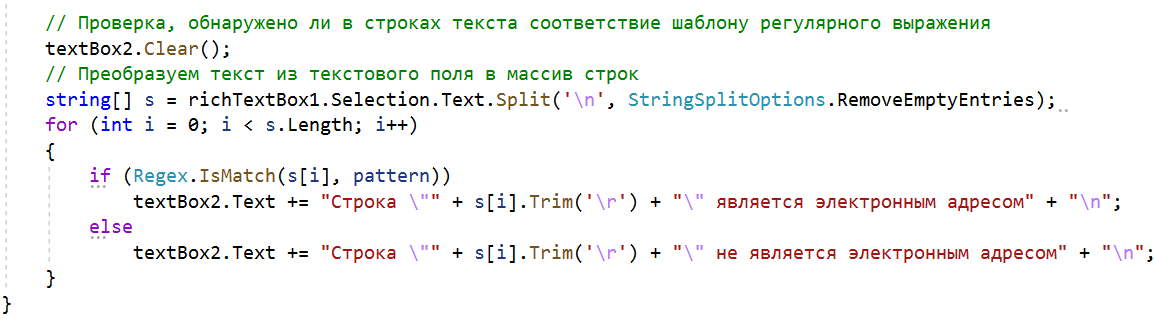


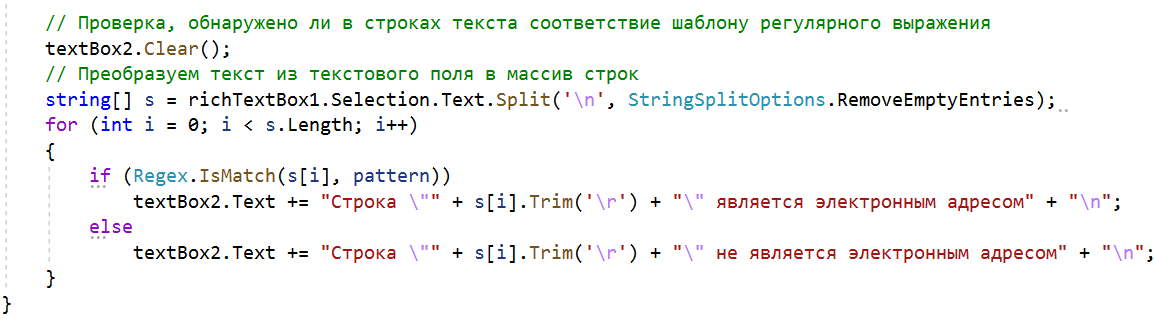
Код, выполняемый при щелчке на кнопку «Прочитать файл»:



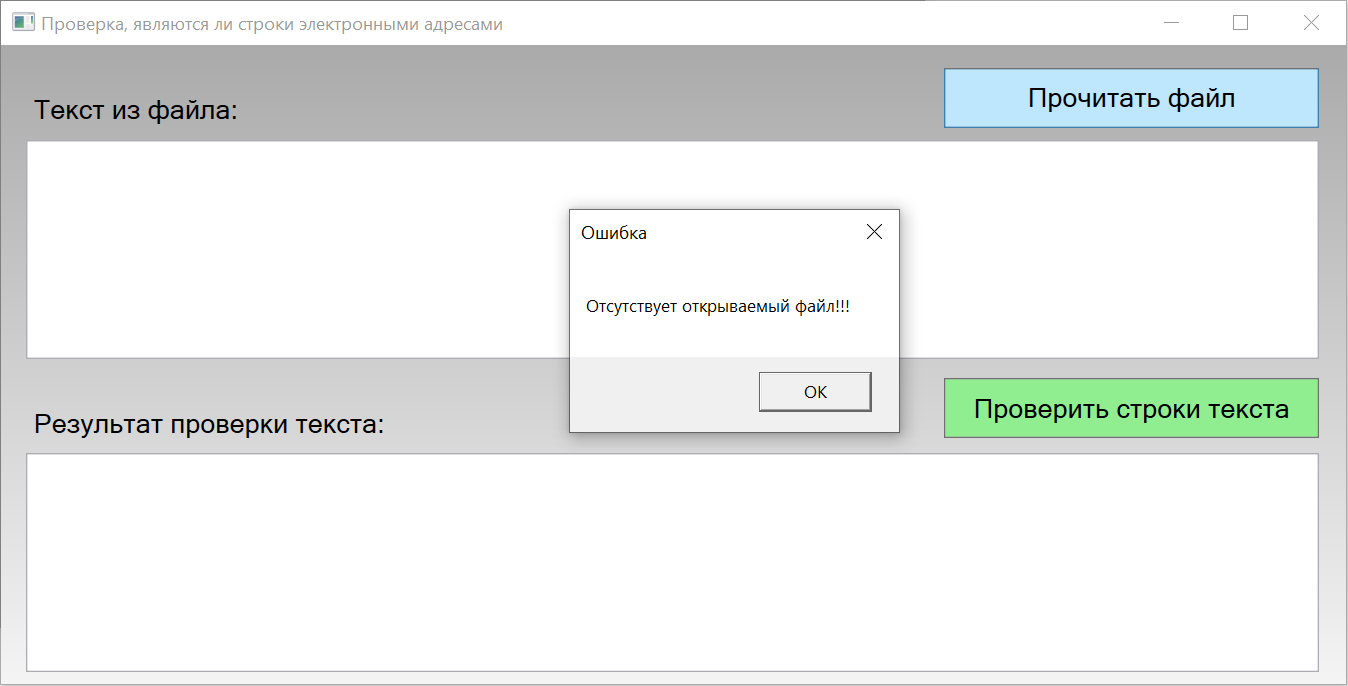
Код, выполняемый при щелчке на кнопку «Проверить строки текста»:

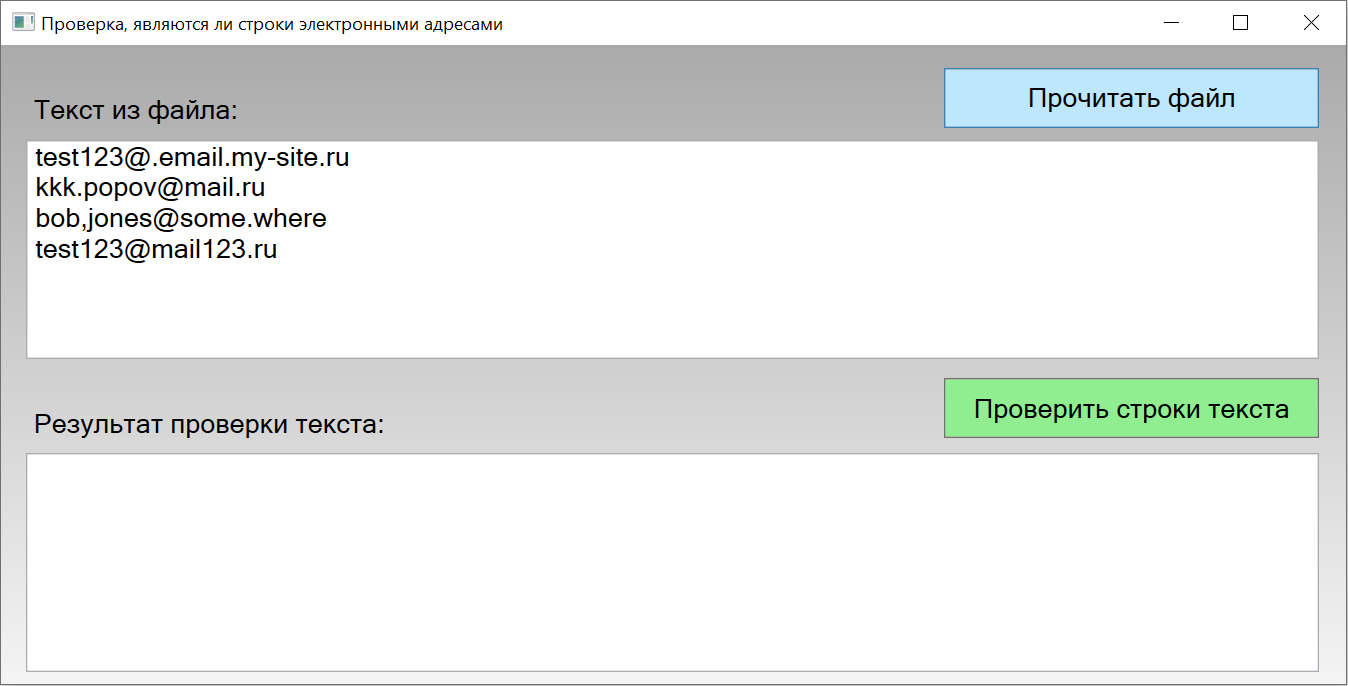


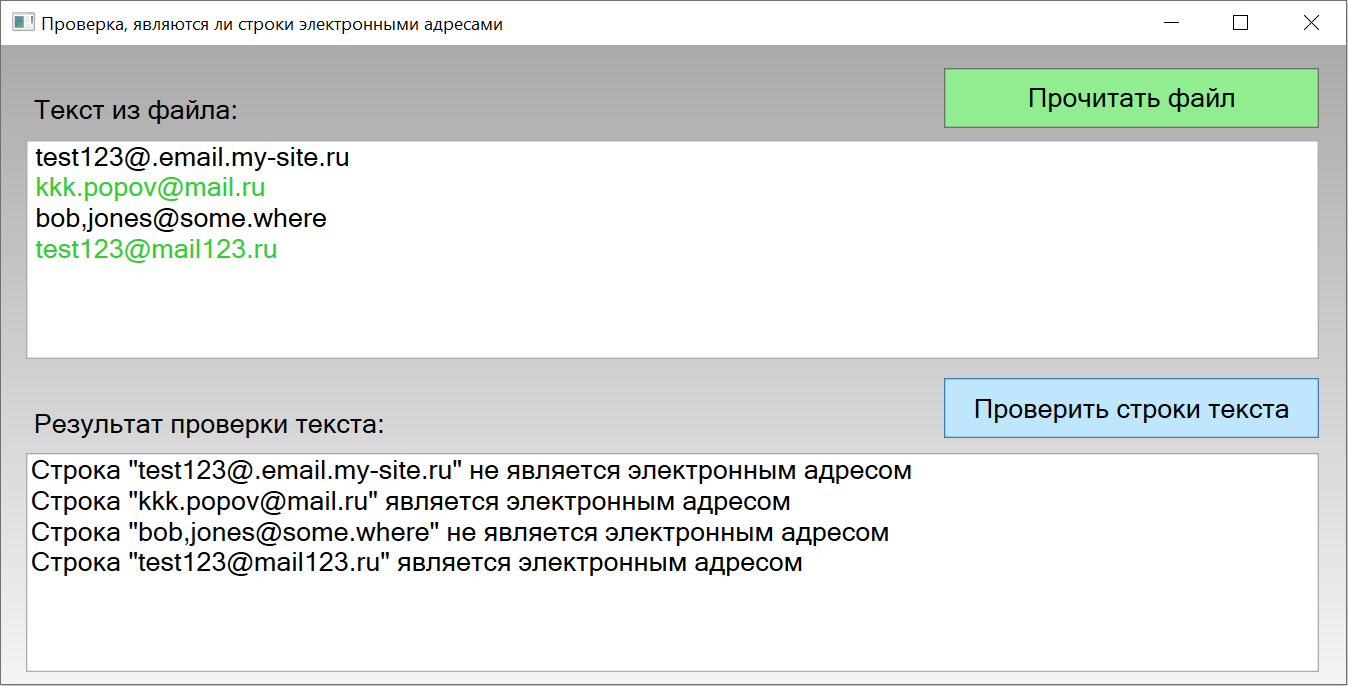




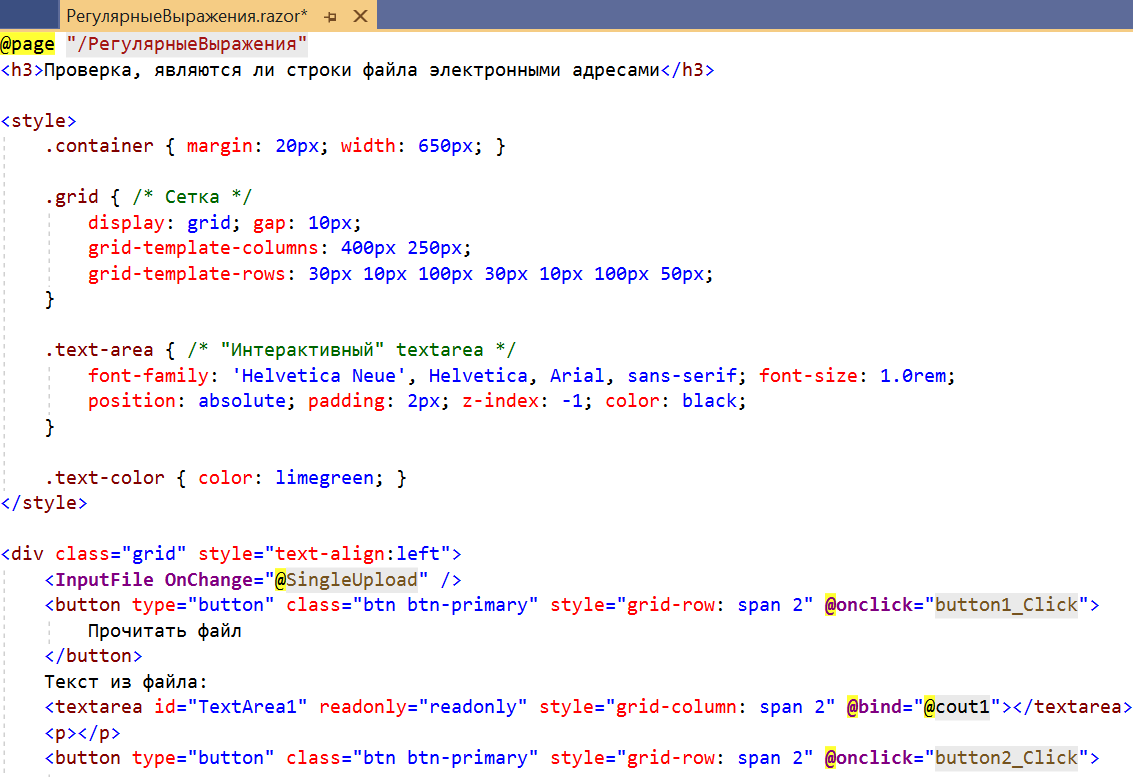
Скрины отлаженной программы:



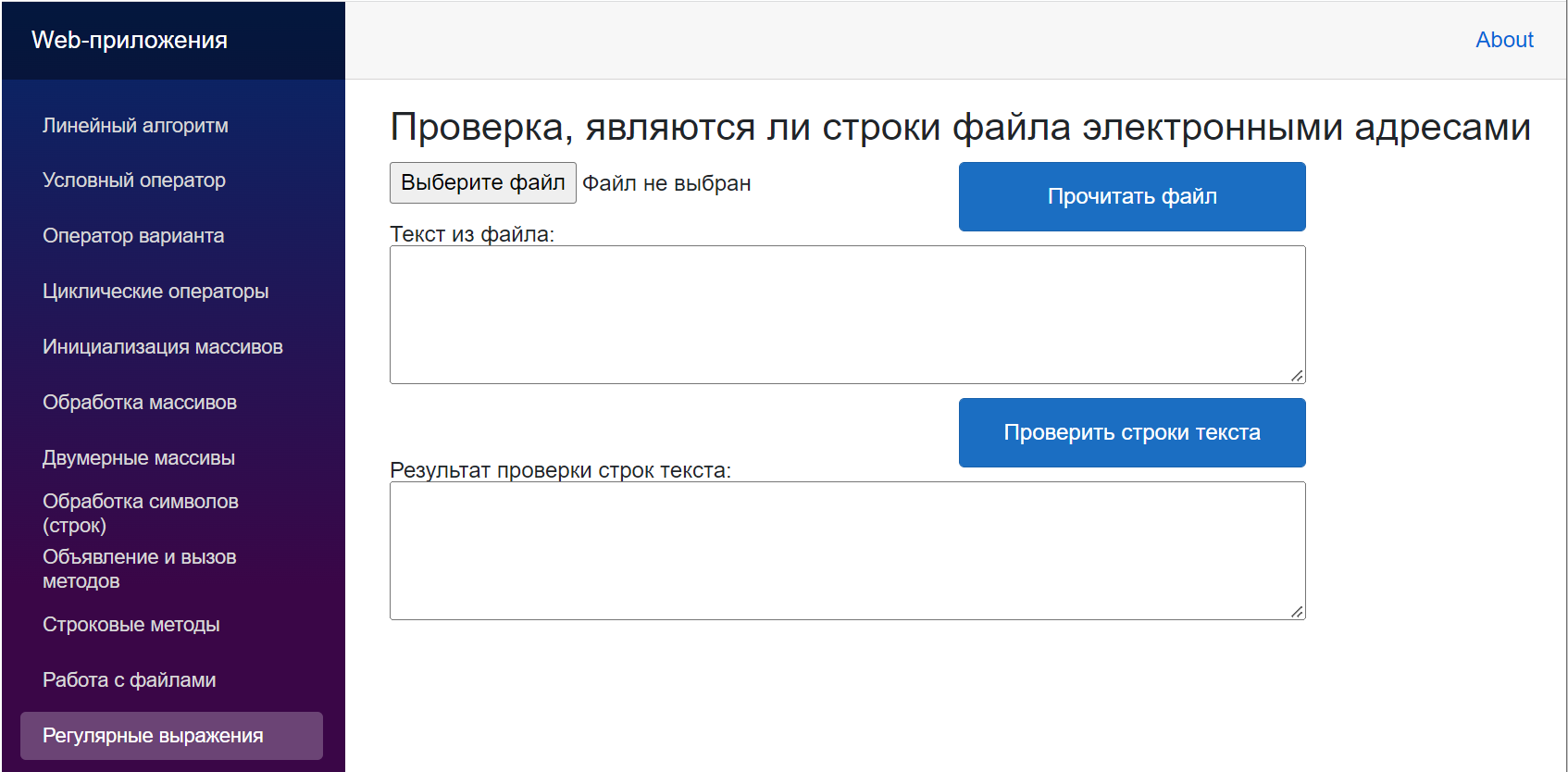




в) Интерфейс приложения WebAssembly Blazor (.NET 5.0):



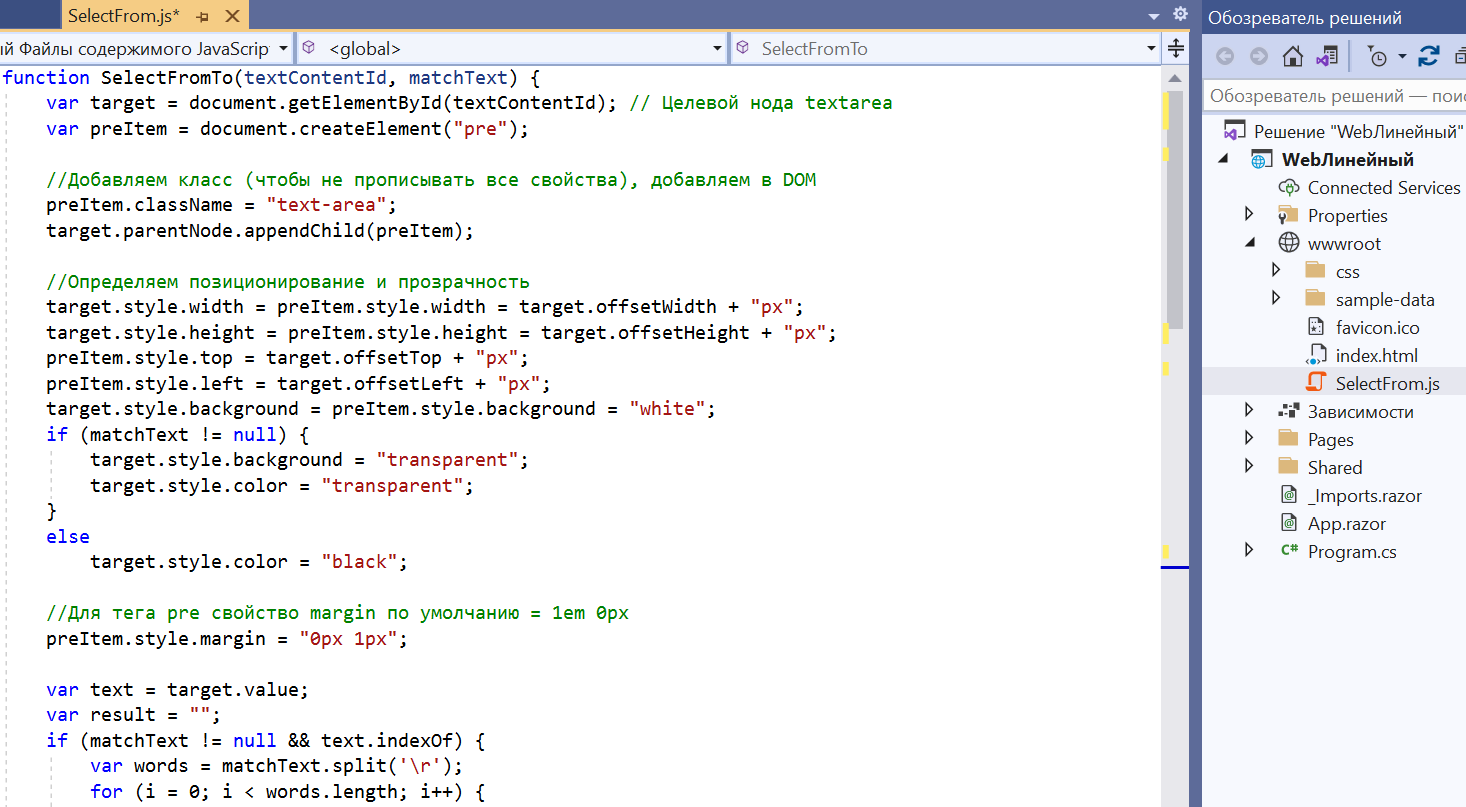


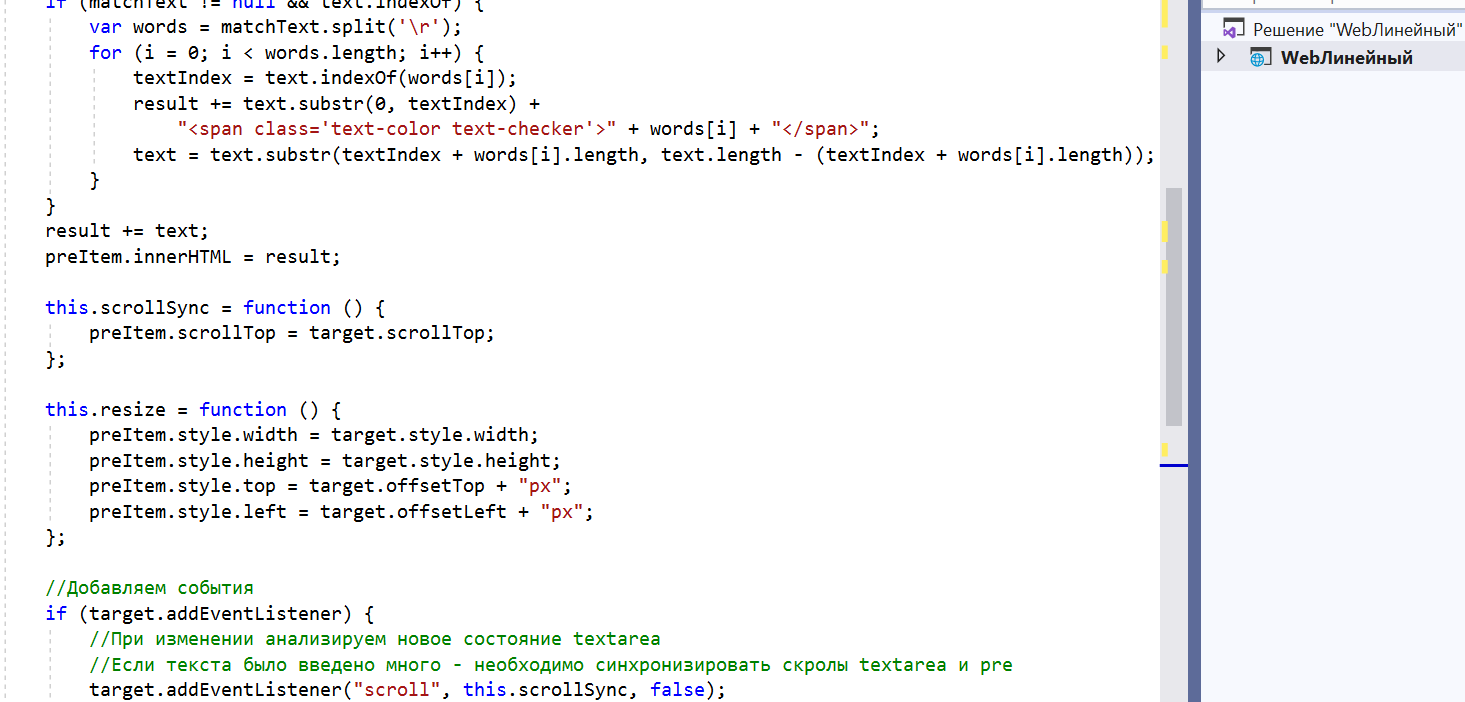


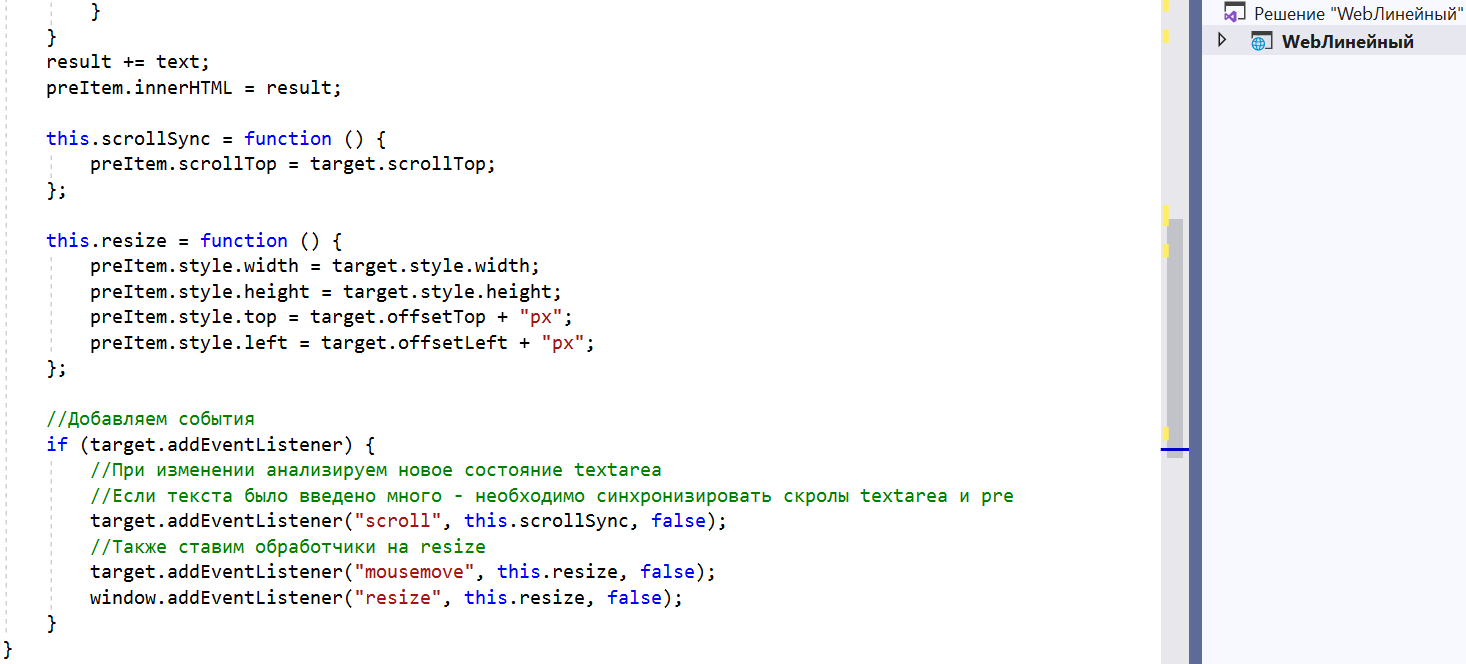
Коды приложения:

В Blazor не существует способа добавить в текстовое поле textarea поддержку тегов для цветного выделения фрагментов текста в зависимости от определенных условий. Одним из вариантов решения этой задачи является использование «многослойности». Для ее реализации в папкеwwwroot вашего проекта создадим файл с именем SelectFrom.js. Этот файл будет содержать функцию JavaScript, которую будем вызывать из кода C#. Эта функция принимает два аргумента: имя текстового поля и строку с текстовыми фрагментами, которые необходимо в этом текстовом поле выделить цветом.

В этой функции с помощью z-index и абсолютного позиционирования поместим блок pre за необходимым нам textarea. В элементе pre настроим шрифт, аналогичный textarea, и также зададим css-свойства, которые нам позволят зеркально повторять текст блока textarea. Также создадим класс, который будет после каждого изменения содержимого textarea синхронизировать данные, осуществлять поиск данных, необходимых для выделения.





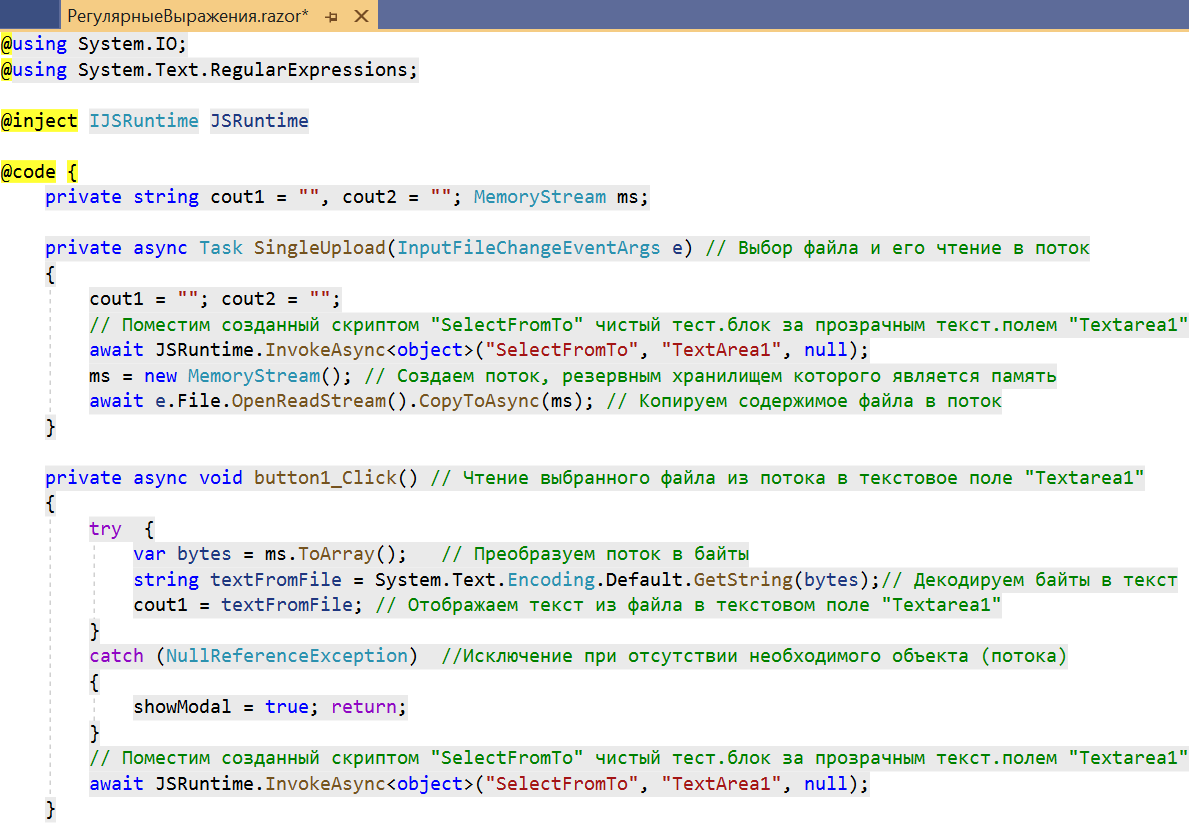


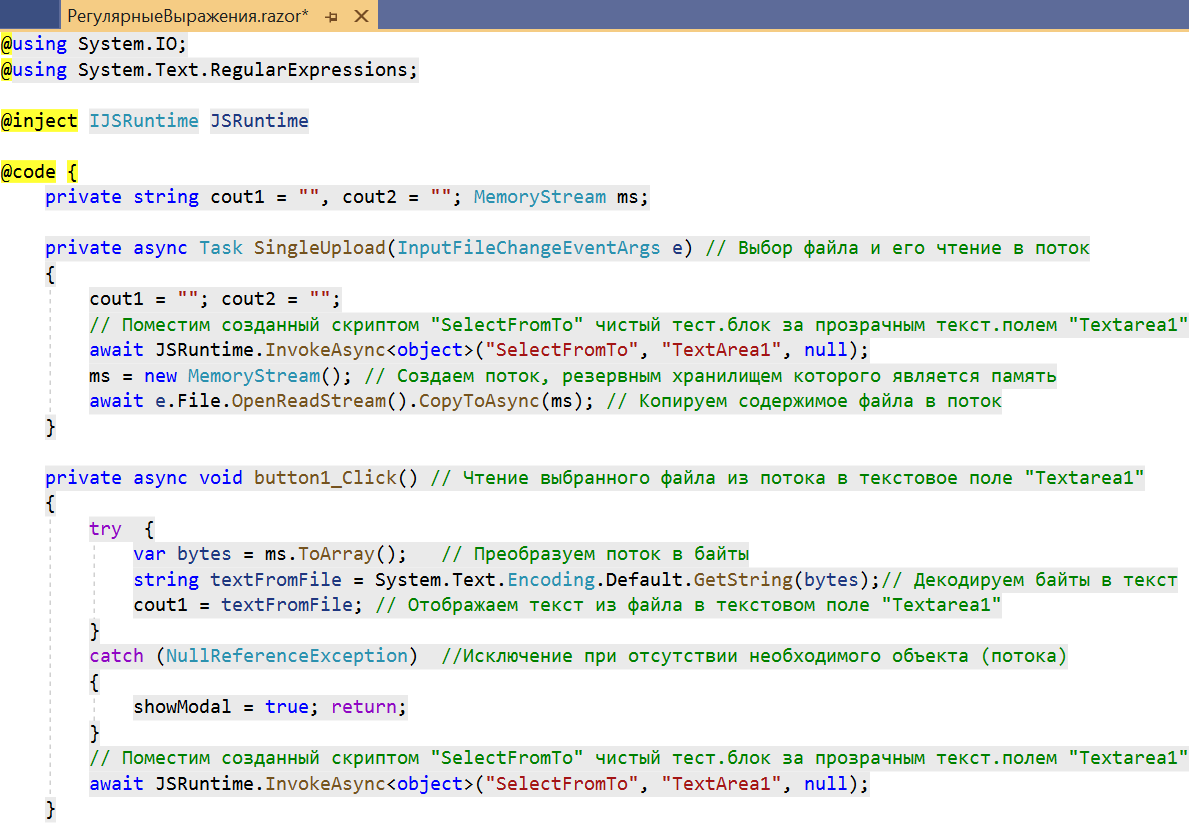
Загрузим скрипт в wwwroot/index.html непосредственно перед закрывающим тегом body </body>.

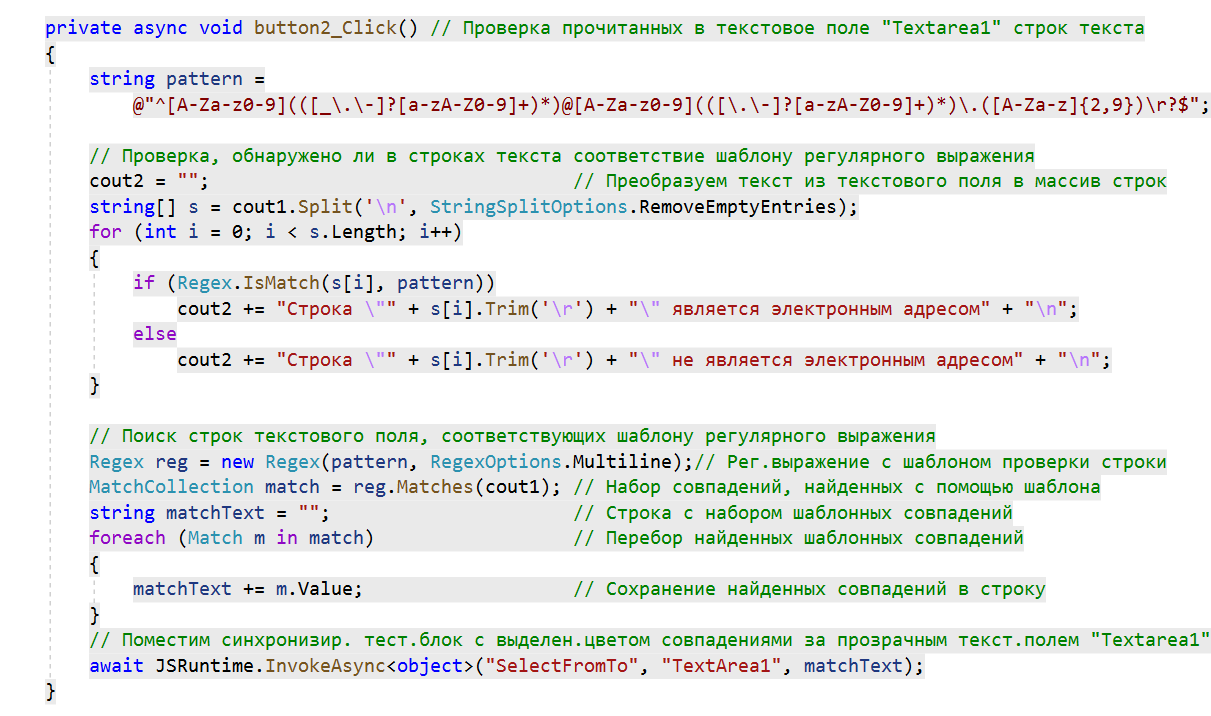


Доступ к файлам и тестовому блоку, поддерживающему цветное выделение текста и позиционированному с прозрачным текстовым полем, осуществляется в Blazor асинхронно, поэтому методы, реализующие соответствующий доступ создаются как async-методы.

Чтобы использовать код JavaScript в Blazor, необходимо в компонент Razor, из которого выполняется код JavaScript, внедрить экземпляр объекта IJSRuntime.

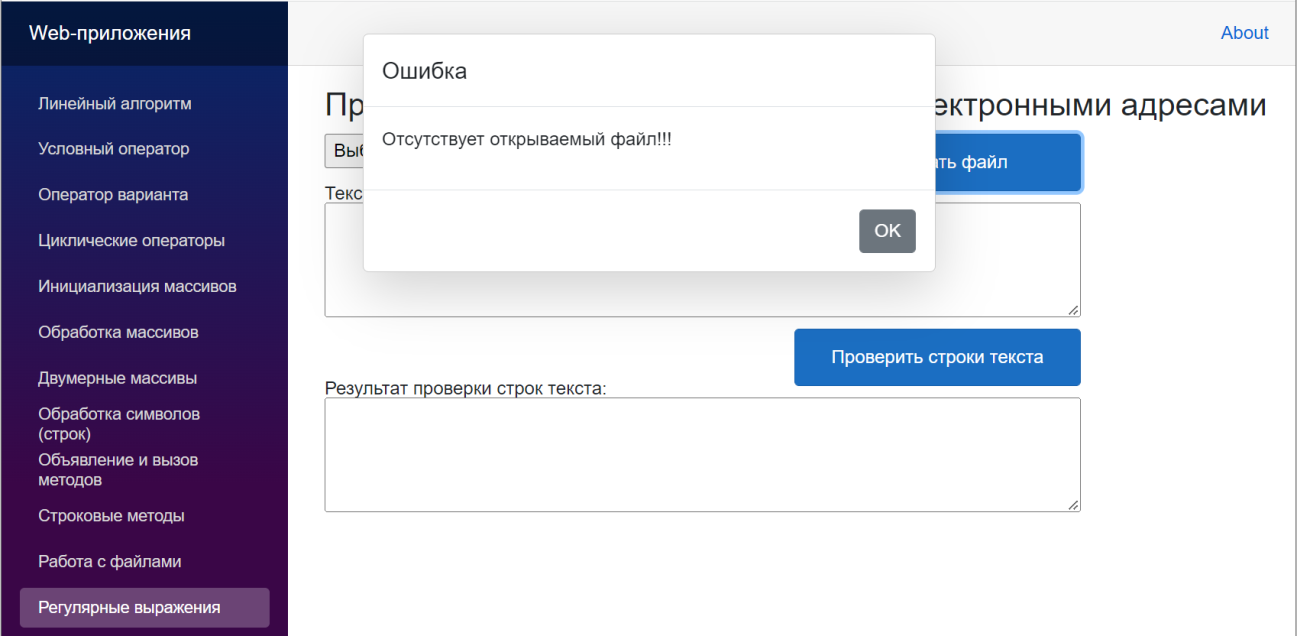


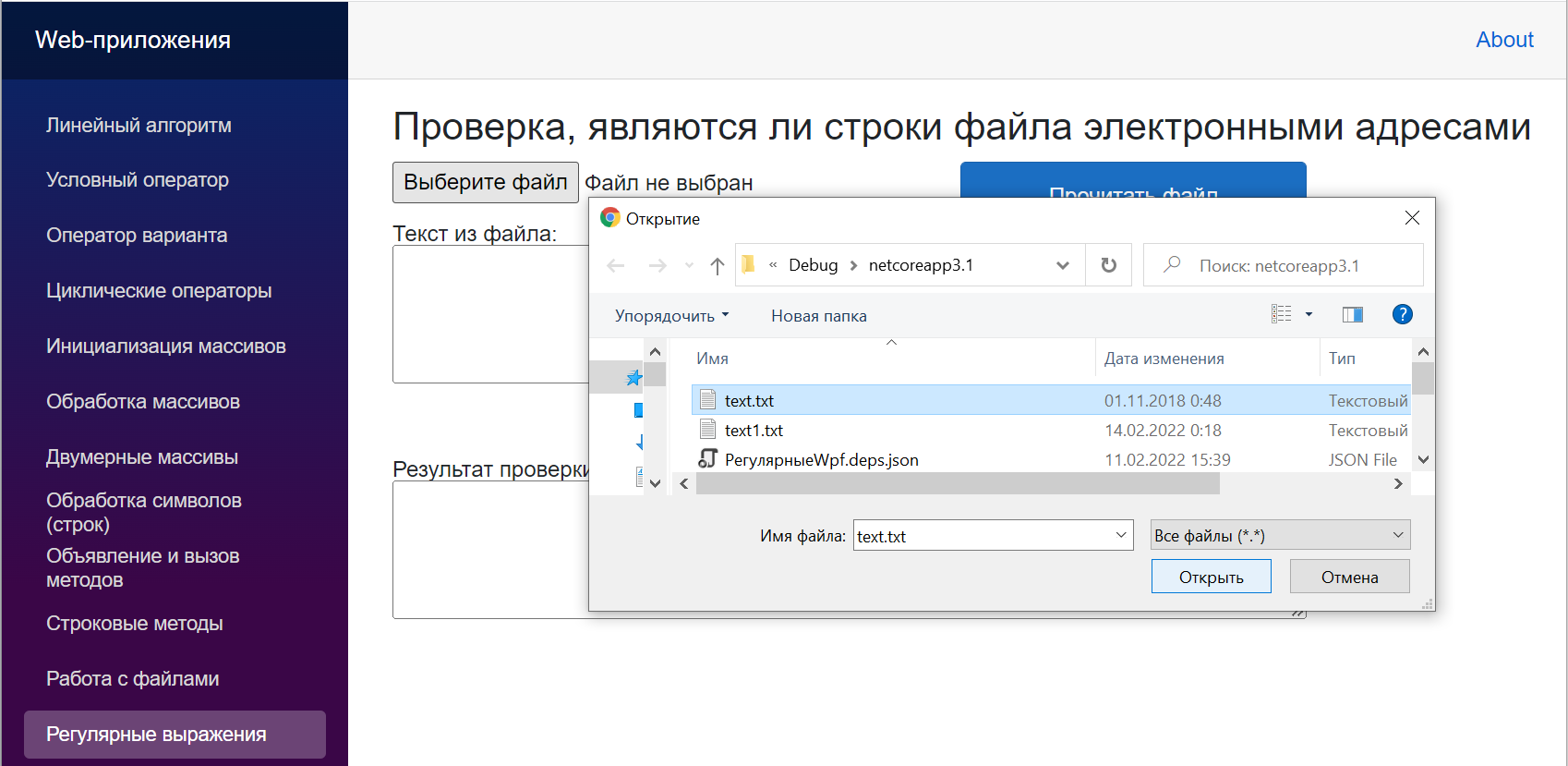


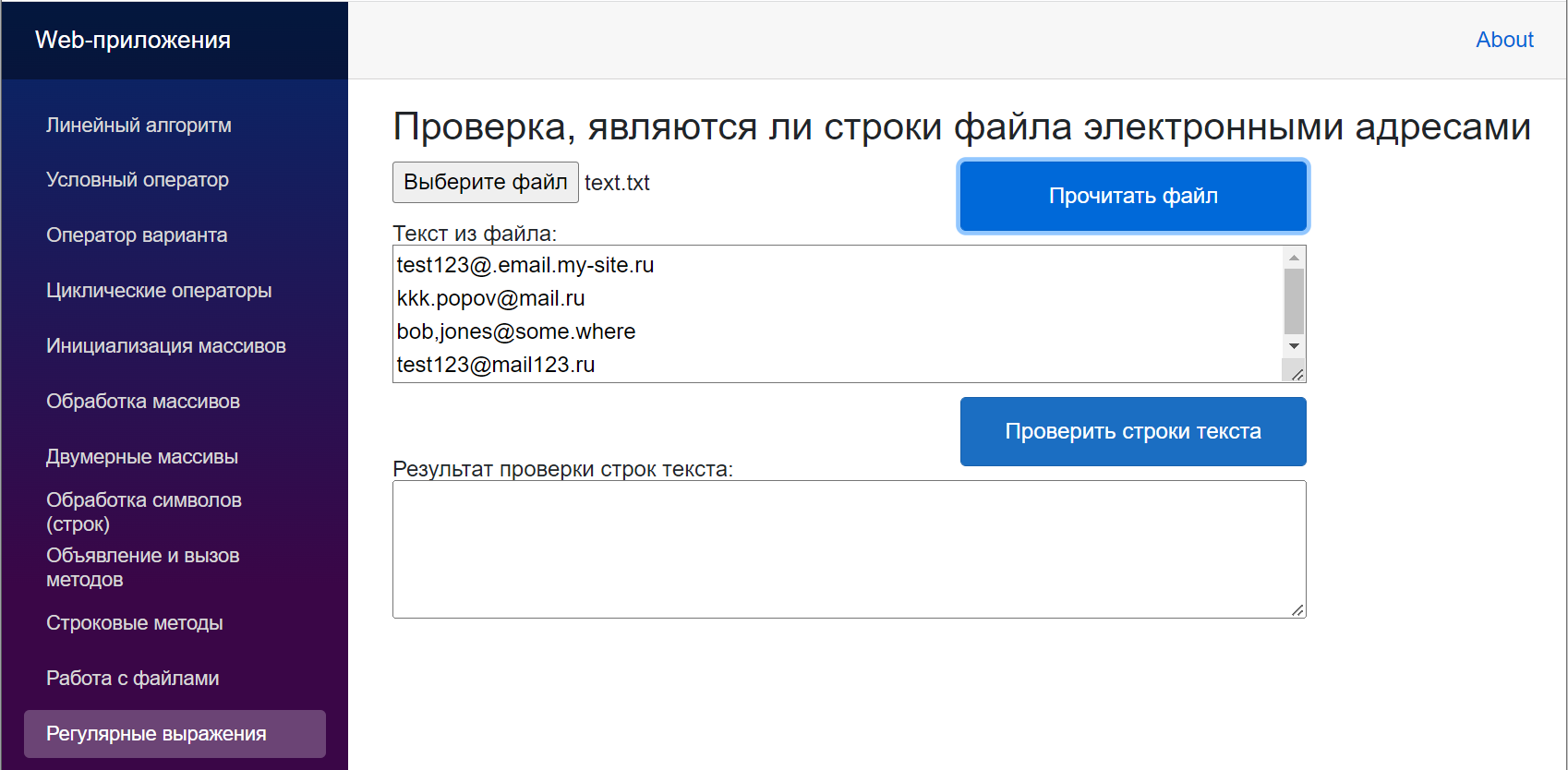


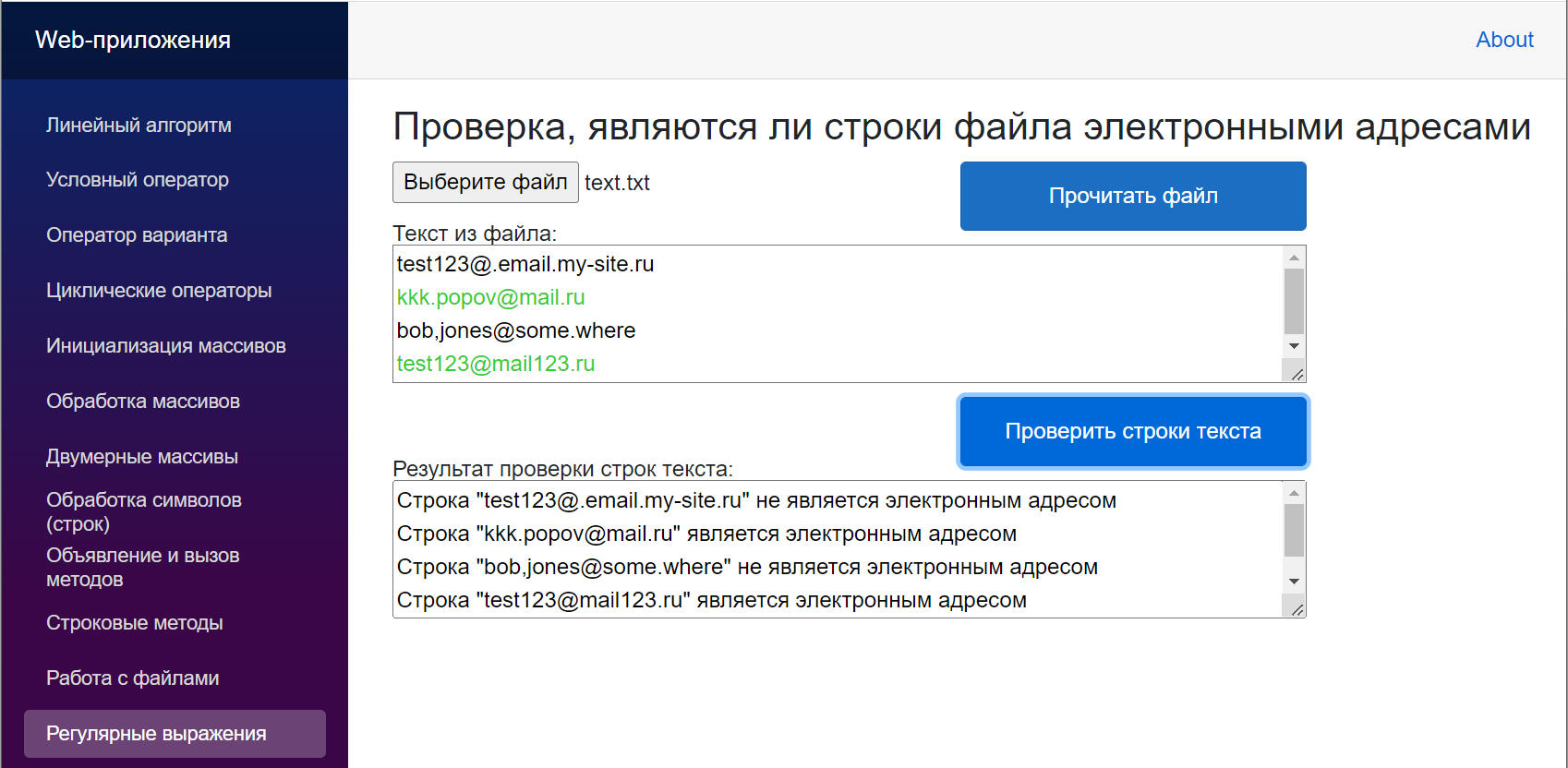


Скрины отлаженной программы:









**4 Порядок выполнения работы**

4.1 Изучить теоретические сведения и задание к работе.

4.2 В соответствии с вариантом задания создать форму приложения Windows Forms (WPF, WebAssembly Blazor).

4.3 В соответствии с вариантом задания разработать и отладить коды приложения Windows Forms (WPF, WebAssembly Blazor), применяя регулярные выражения для поиска соответствий в строках и используя исключения, выбрасываемые, если форматы или значения входных данных не соответствуют спецификациям параметров обрабатывающих их методов.

**5 Содержание отчета**

5.1 Название работы.

5.2 Цель работы.

5.3 Программные коды и скрины работы разработанного приложения Windows Forms в соответствии с вариантом задания.

**6 Варианты заданий**

6.1 Дан текстовый файл, состоящий из нескольких строк. Проверить является ли каждая из них фамилией и инициалами, написанными в формате «Фамилия И.О.». Выдать сообщение о результате проверки.

6.2 Дан текстовый файл, состоящий из нескольких строк. Найти в нем слово «мир». Вывести это слово и количество встречающихся экземпляров.

6.3 Дан текстовый файл. Найти в нем слово «авто» и однокоренные с ним слова, и выделить их цветом.

6.4 Дан текстовый файл, состоящий из нескольких строк. Проверить, начинается ли он с введенного слова. Выдать сообщение о результате проверки.

6.5 Дан текстовый файл. Найти в нем слова, заканчивающиеся на букву «а», и вывести эти слова.

6.6 Дан текстовый файл, состоящий из нескольких строк. Проверить, заканчивается ли он введенным словом. Выдать сообщение о результате проверки.

6.7 Дан текстовый файл, состоящий из нескольких строк, и слово. Вывести строки, в которых встречается это слово, и заканчивающиеся символом «?».

6.8 Дан текстовый файл, состоящий из нескольких строк. Найти в нем введенное слово, организуя поиск с конца в начало, и выделить его цветом.

6.9 Дан текстовый файл, состоящий из нескольких строк. Проверить, является ли каждая из них номером телефона. Выдать сообщение о результате проверки.

6.10 Дан текстовый файл. Найти в нем введенное слово, написанное прописными буквами, и выделить его цветом.

6.11 Дан текстовый файл и слово. Вывести все слова из текста, не абсолютно совпадающие с введенным словом.

6.12 Дан текстовый файл, состоящий из нескольких строк. Проверить является ли каждая из них IP-адресом. Выдать сообщение о результате проверки.

6.13 Дан текстовый файл, состоящий из нескольких строк. Найти в нем слово «куб». Вывести строки, в которых оно встречается, и заканчивающиеся символом «!».

6.14 Дан текстовый файл. Найти в нем слова, содержащие введенные буквы, и вывести их количество.

6.15 Дан текстовый файл, состоящий из нескольких строк. Проверить является ли каждая из них датой, написанной в формате «ДД.ММ.ГГГГ». Выдать сообщение о результате проверки.

6.16 Дан текстовый файл. Найти в нем слова, написанные с заглавной буквы, и вывести эти слова.

6.17 Дан текстовый файл, состоящий из нескольких строк. Найти в нем все слова заданной длины и выделить их цветом.

**7 Используемая литература**

7.1 Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул Технология разработки программного обеспечения. Форум, Инфра-М, 2009

7.2 Эндрю Троелсен Язык программирования С# 2010 и платформа .NET 4.0, М., ВИЛЬЯМС, 2011

7.3 А. Марченко C#. Введение в программирование, М, Вильямс, 2009

7.4 http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx.