от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021  года

Пермский филиал федерального государственного автономного образовательного учреждения высшего образования

«Национальный исследовательский университет   
«Высшая школа экономики»

**Факультет профессиональной переподготовки**

**Разработка программного модуля для управления задачами предприятия**

Проектная работа по направлению подготовки

«Объектно-ориентированное программирование»

   Слушателя Пьянкова С.Р.

         группы ООП-20

Руководитель проектной работы

Викентьева О.Л.

Пермь, 2021

**аннотация**

Проектная работа «Разработка программного модуля для управления задачами предприятия» посвящена процессу создания приложения для обеспечения управления задачами на предприятии.

В статье описывается подробный процесс, как создавалась информационная система управления задачами.

Рассматриваемая тема будет интересна специалистам по разработке приложений, анализу данных, проектированию систем.

Состоит из: 4 этапов, где подробно расписаны все действия, предпринятые для разработки приложения, приведены таблицы и рисунки для систематизации и визуализации информации, приложения с исходным кодом программы, руководством по использованию приложения.

Представлены: введение, аналитическая часть, проектирование, разработка приложения, тестирование, заключение, приложение.

**оглавление**

Введение 4

1 этап

Анализ задачи 5

Спецификация требований 6

2 этап

Модель вариантов использования 6

Документирование прецедентов 7

Модель анализа 10

Диаграмма последовательности 12

Проектирование базы данных 14

Проектирование интерфейса 15

3 этап

Разработка 20

4 этап

Тестирование 25

Заключение 26

Приложение 27

Руководство пользователя 32

**введение**

Проектная работа «Разработка программного модуля для управления задачами предприятия» создавалась для обеспечения предприятия инструментом для оперативного регулирования задач на предприятии. Из-за больших площадей и удаленности отделов друг от друга, очень трудно организовать работу персонала, невозможно решить эту проблему с помощью досок или списков на информационных билбордах.

Используя за основу локальную сеть предприятия, планируется без особых вложений средств в техническую сторону вопроса развернуть информационную систему для оперативного управления задачами.

В основе лежит приложение, в котором будут реализованы возможности по управлению задачами для персонала. Приложение выполнено на языке c# с использованием базы данных. Авторизации в приложении нет по причине того, что данная программа будет использоваться только в пределах предприятия и не несет в себе секретных данных.

Было принято решение воспользоваться языком программирования высокого уровня C#, инструментами windows forms, а также базой данных Microsoft SQL Server.

Оперативное регулирование задач на предприятии выполняется на бумажных носителях, требуется множество действий, чтобы все заинтересованные лица были уведомлены о существующих задачах, что вносит сложности в оперативность исполнения задач. Помимо уведомления сотрудников, архивирование и учет задач на бумажных носителях вносит определенные неудобства.

Главной проблемой при управлении задачами можно считать неудобную систему обмена, учета и архивирования задач, которая в данный момент используется на предприятии.

Объектом является процесс оперативного регулирования задачами на предприятии.

На первом этапе работы следует провести анализ процесса, разработать требования к информационной системе.

Вторым этапом будет проектирование информационной системы, в который войдут проектирование интерфейса программы, проектирование базы данных.

Третий этап — это разработка приложения.

На четвертом этапе будет проведено тестирование приложения.

**1 этап**

**анализ задачи**

Предприятие, где будет реализовываться программа – несколько цехов и складских помещений, удаленных друг от друга на большое расстояние.

При проведении интервью с заказчиком были выяснены следующие данные о предстоящей работе:

1. Пользователи программы – персонал предприятия с начальным знанием работы на персональном компьютере.
2. Персональные компьютеры на предприятии средней производительности.
3. Локальная сеть соединяет все компьютеры с сервером.
4. Сервер работает на операционной системе Windows.
5. Обслуживание локальной сети и работоспособности компьютеров ведется штатным системным администратором.
6. Предприятие обеспечено резервным источником питания, все компьютеры укомплектованы бесперебойными источниками питания.
7. Выход в сеть интернет контролируется файрволлом.
8. Языковая локализация при создании задач – Русский язык, Английский язык.
9. Количество прогнозируемых одновременных задач – 15 штук.

По информации, которую предоставил заказчик стала ясна система, которая в данный момент используется на предприятии. На ее основе была составлена диаграмма активности:

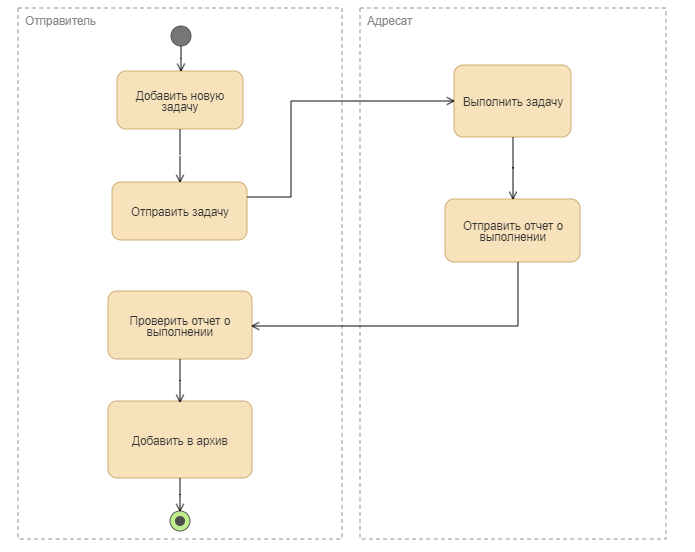


Рисунок 1. Диаграмма активности на предприятии

**спецификация требований**

Система должна:

1. Обеспечивать добавление новых задач
2. Изменять существующие задачи
3. Удаление задач
4. Поиск задач по адресату

**2 этап**

**проектирование**

**модель вариантов использования**

Таблица 1. Распределение требований по субъектам и прецедентам

|  |  |  |  |
| --- | --- | --- | --- |
| № | Требование | Субъект | Прецедент |
| 1 | Система должна обеспечивать возможность добавления новых задач. | Сотрудники предприятия | Создать новую задачу |
| 2 | Система должна обеспечивать возможность изменения существующих задач. | Сотрудник предприятия | Изменить существующую задачу |
| 3 | Система должна обеспечивать удаление существующей задачи. | Сотрудник предприятия | Удалить существующую задачу |
| 4 | Система должна обеспечить поиск по всем задачам с фильтрацией по адресату. | Сотрудник предприятия | Найти задачу по адресату |

На основе собранной информации и в ходе анализа задачи, были выявлены требования к будущей системе, на основе которых была создана диаграмма прецедентов:

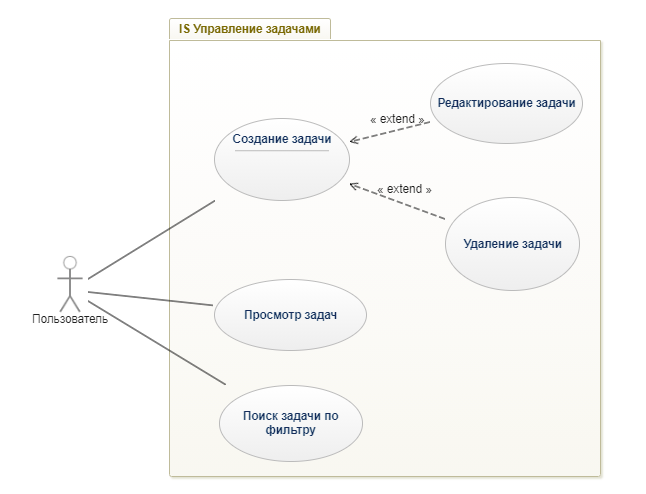


Рисунок 2. Диаграмма прецедентов для системы задач предприятия

**документирование прецедентов**

Дальнейшей работой стало документирование прецедентов. Оно было выполнено в виде нескольких таблиц:

Таблица 2. Прецедент добавления новой задачи

|  |  |
| --- | --- |
| Краткое описание | Прецедент дает возможность Пользователю создать задачу. Задача включает: id задачи, текст задачи, имя отправителя, имя адресата, дату создания. |
| Актеры | Пользователь |
| Предусловия | Необходимость в новой задаче |
| Основной поток | Пользователь выбирает вкладку «Создать задачу» в окне программы.  Система выводит форму «Создать задачу».  Пользователь вводит текст задачи, имя отправителя. Имя адресата.  Пользователь создает заказ нажатием кнопки «Создать».  Система создает новую задачу  Система сохраняет задачу в базе данных.  Система обновляет список задач в главном окне. |
| Альтернативные потоки | Пользователь не вводит всю необходимую информацию в поля ввода, система выводит сообщение о том, что не все поля заполнены. |
| Постусловия | Если прецедент был успешным, задача записывается в базу данных.  В противном случае состояние системы остается неизменным. |
| Точки расширения | Редактировать задачу  Удалить задачу |

Таблица 3. Прецедент изменения существующей задачи.

|  |  |
| --- | --- |
| Краткое описание | Прецедент дает возможность Пользователю создать задачу.  Изменение задачи включает: Новый текст задачи, новый адресат |
| Актеры | Пользователь |
| Предусловия | Необходимость в изменении существующей задачи |
| Основной поток | Пользователь выбирает задачу в списке всех задач в главном окне программы.  Пользователь выбирает вкладку «Изменить задачу» в окне программы.  Система выводит форму «Изменить задачу».  Пользователь вводит новый текст задачи, нового адресата.  Пользователь применяет изменения нажатием на кнопку «Применить изменения»  Система сохраняет изменения в базе данных.  Система обновляет список всех задач в главном окне. |
| Альтернативные потоки | Пользователь на выбирает задачу, которую нужно изменить, система, при нажатии на кнопку «Применить изменения», выводит окно с предупреждающим текстом «Не выбрана задача для изменения».  Пользователь не заполняет все поля для изменения задачи, система выводит сообщение что не все поля заполнены. |
| Постусловия | Если прецедент был успешным, измененная задача записывается в базу данных.  В противном случае состояние системы остается неизменным. |

Таблица 4. Прецедент удаления задачи.

|  |  |
| --- | --- |
| Краткое описание | Прецедент дает возможность Пользователю удалить заказ. |
| Актеры | Пользователь |
| Предусловия | Необходимость удаления задачи |
| Основной поток | Пользователь выбирает задачу, которую необходимо удалить из списка всех задач в главном окне программы.  Пользователь нажимает на кнопку «Удалить»  Система удаляет выбранную задачу из списка задач и удаляет задачу из базы данных. |
| Альтернативные потоки | Пользователь не выделяет необходимую задачу в списке всех задач в главном окне программы, система выводит окно с предупреждающим текстом «Не выбрана задача для удаления». |
| Постусловия | Если прецедент был успешным, задача удаляется из базы данных.  В противном случае состояние системы не изменяется. |

Таблица 5. Прецедент поиска задач по имени адресата.

|  |  |
| --- | --- |
| Краткое описание | Прецедент дает возможность Пользователю осуществить поиск всех задач по определенному адресату. |
| Актеры | Пользователь |
| Предусловия | Необходимость поиска задач, в которых заявлен один и тот же адресат. |
| Основной поток | Пользователь выбирает вкладку «Поиск по адресату».  Пользователь вводит имя адресата.  Система выводит все задачи, в которых упоминается этот адресат. |
| Альтернативные потоки | Пользователь не заполняет окно ввода имени адресата, система выводит сообщение, что не все поля заполнены. |
| Постусловия | Если прецедент был успешным, найденные задачи выводятся в окно выдачи.  В противном случае состояние системы не изменяется. |

**модель анализа**

Была построена модель анализа, которая включает в себя диаграммы деятельности, диаграмму последовательности модели анализа.

диаграммы деятельности (активности)

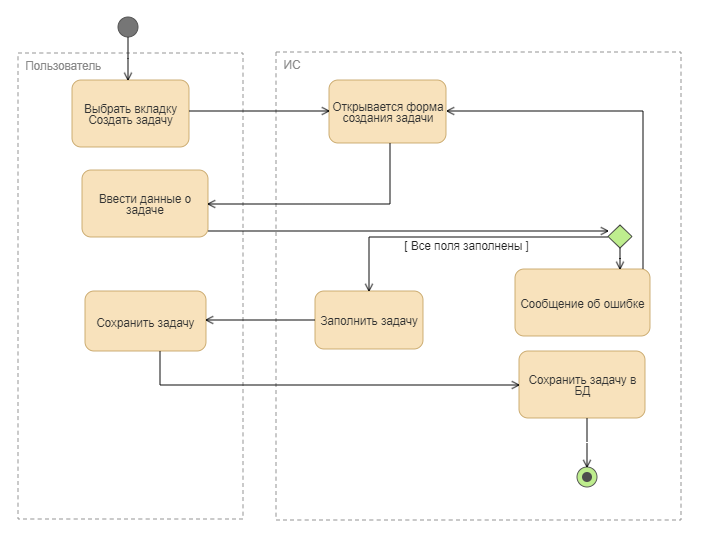


Рисунок 3. Диаграмма деятельности для прецедента «Создать новую задачу»

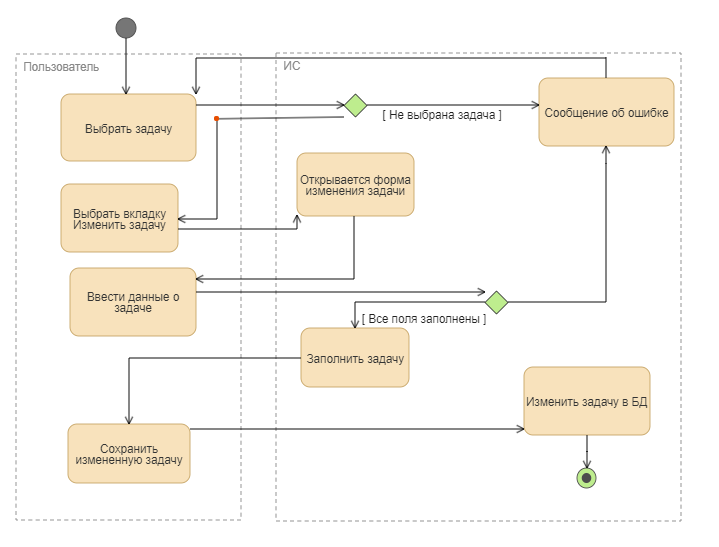


Рисунок 4. Диаграмма деятельности для прецедента «Изменить существующую задачу»

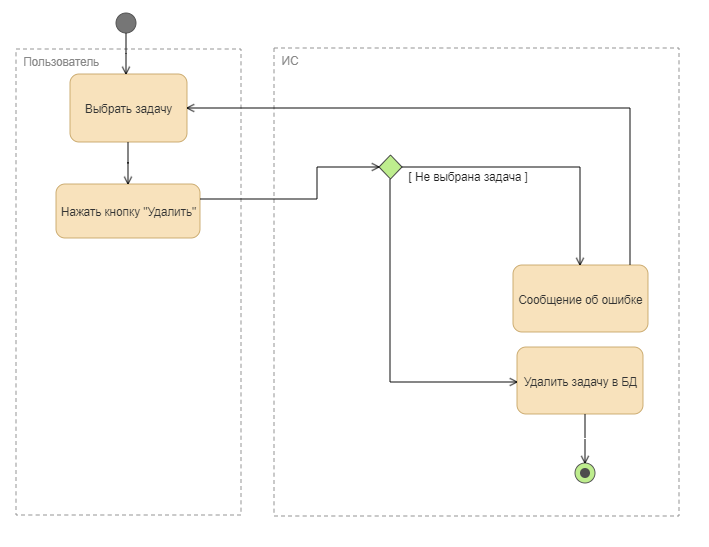


Рисунок 5. Диаграмма деятельности для прецедента «Удалить существующую задачу»

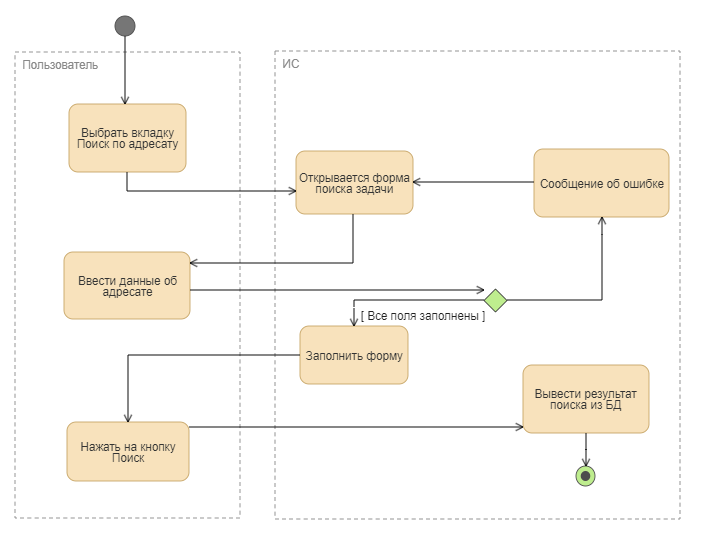


Рисунок 6. Диаграмма деятельности для прецедента «Поиск задач по адресату»

**диаграмма последовательности**

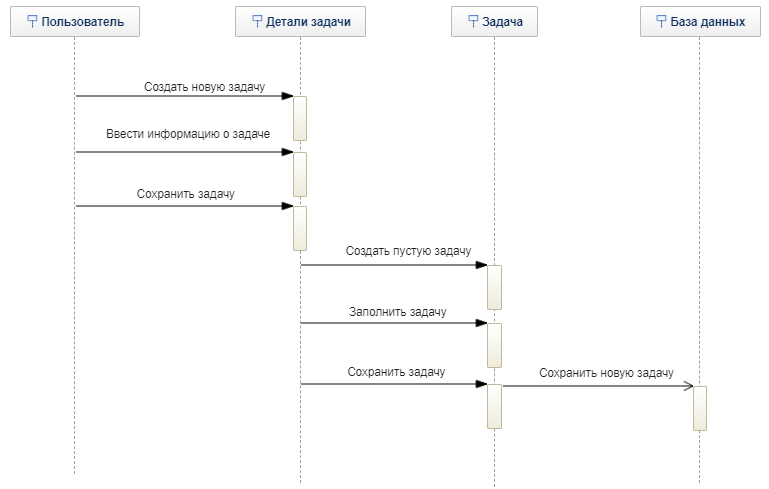


Рисунок 7. Диаграмма последовательности для прецедента «Создать новую задачу» на этапе анализа

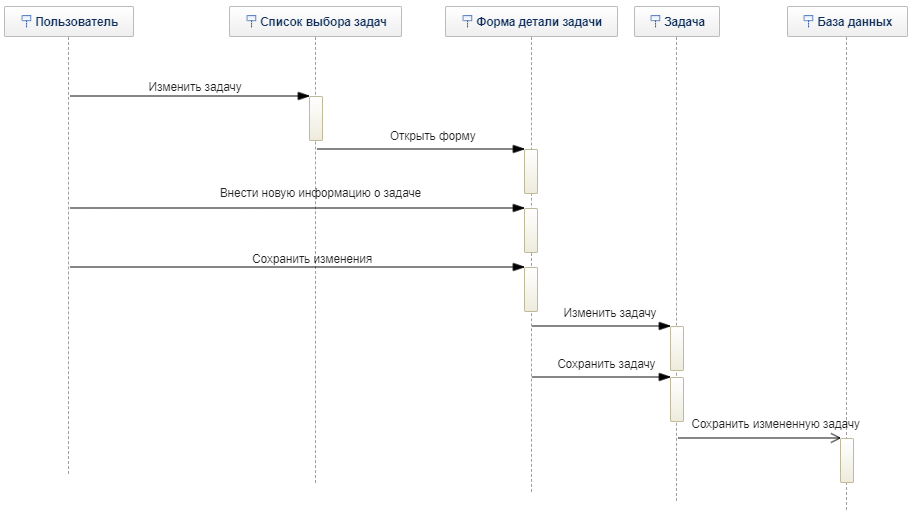


Рисунок 8. Диаграмма последовательности для прецедента «Изменить существующую задачу» на этапе анализа

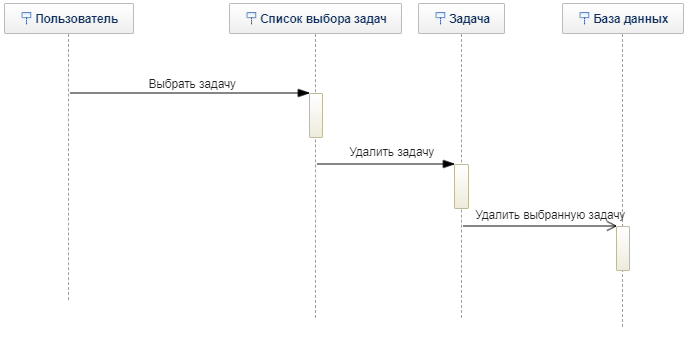


Рисунок 9. Диаграмма последовательности для прецедента «Удалить существующую задачу» на этапе анализа

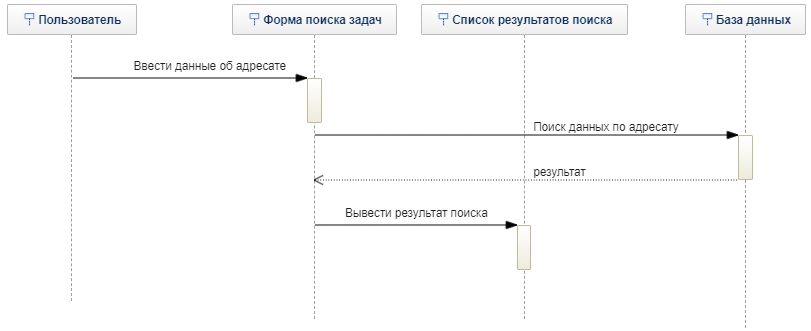


Рисунок 10. Диаграмма последовательности для прецедента «Поиск задач по адресату» на этапе анализа

**проектирование базы данных**

Главной стратегией в решении вопроса проектирование задачи стало создание отдельной базы данных, в которой будет храниться вся информация по задачам и приложения, которое будет с этой базой данных работать.

В базе данных будут присутствовать следующие поля:

1. Id задачи, тип int
2. Текст задачи, тип string
3. Автор, тип string
4. Адресат, тип string
5. Дата создания, тип datetime

Было решено создавать базу данных на платформе Microsoft SQL Server.

Id присваивается автоматически базой данных с шагом 1.

CreateDate (Дата создания) присваивается автоматически в момент добавления в базу данных. Часовой пояс присваивается такой же, как у сервера, на котором работает база данных.

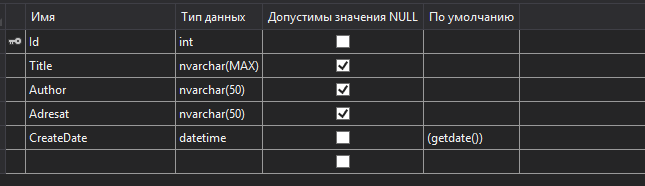


Рис. 11 Таблица Notes

**проектирование интерфейса**

Так как приложение будет создаваться в системе Windows Forms, было принято решение создавать не отдельные методы на каждый прецедент, а методы к каждому событию в приложении.

Например, при нажатии кнопки на главном окне программы, не будет вызываться определенное событие, а будет происходить цепочка запросов и действий.

Программа представляет из себя главное окно, кнопки функций и панель вкладок.

Для быстрого доступа к обновлению и удалению задач, на главном экране расположены соответствующие кнопки:

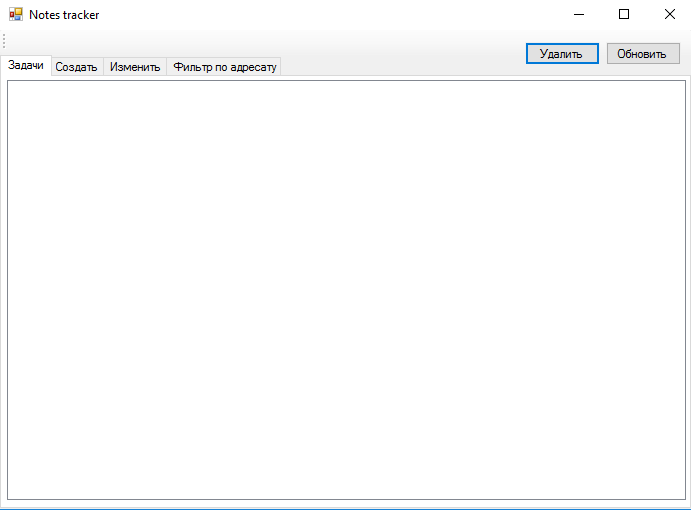


Рис. 12 Главное окно программы

Вкладка «Создать» выглядит следующим образом:

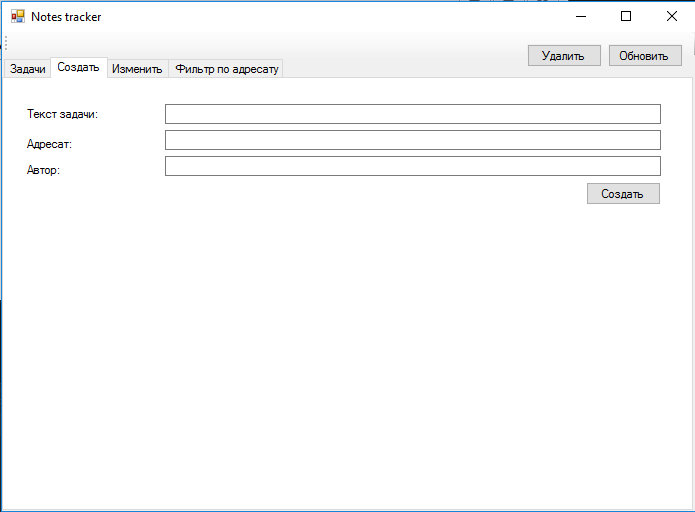


Рис 13. Вкладка «Создать»

Если пользователь не вводит в одно из полей необходимую информацию, появляется предупреждающая надпись:

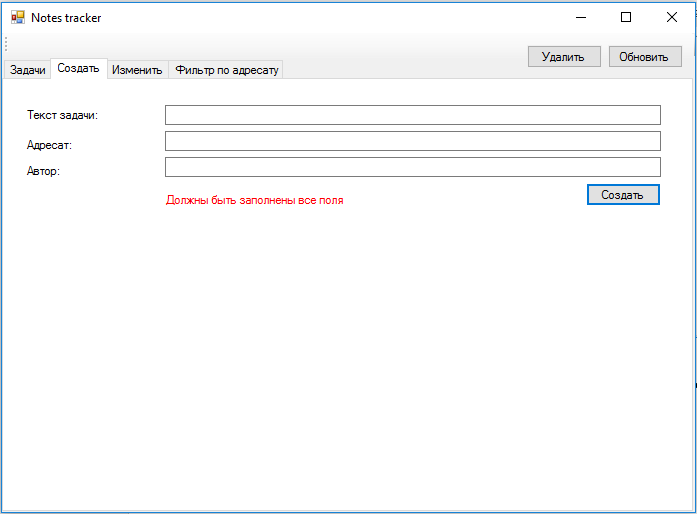


Рис 14. Предупреждение при неполном вводе.

Вкладка «Изменить» выглядит следующим образом:

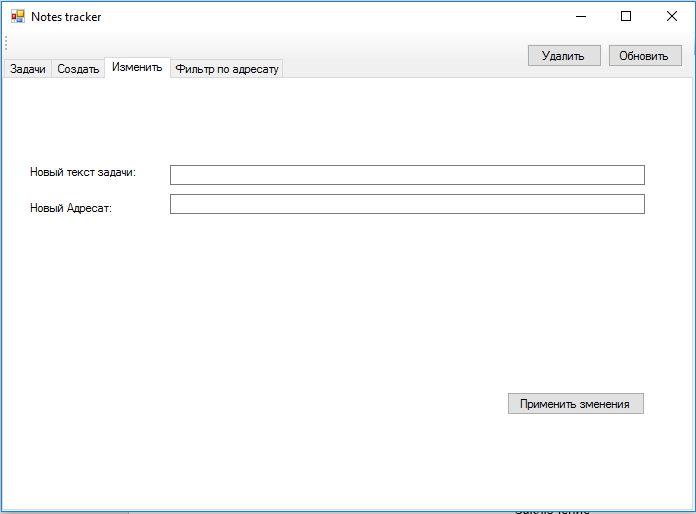


Рис 15. Вкладка «Изменить»

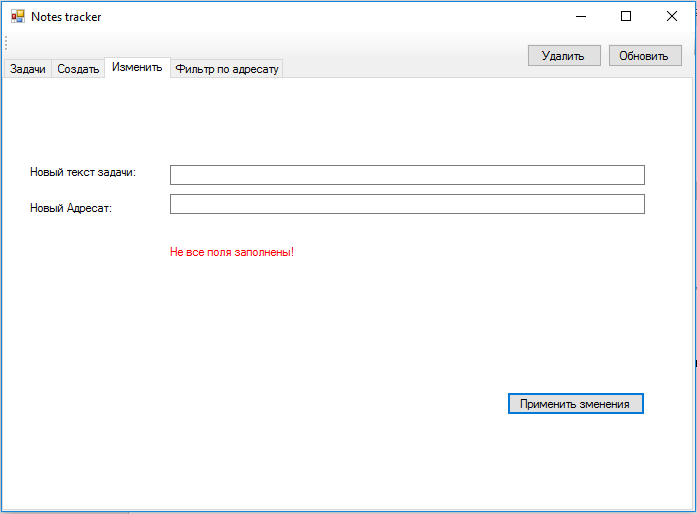
При неполном вводе данных, появляется предупреждение по аналогии с вкладкой «Создать»:

Рис 16. Предупреждение при неполном вводе.

Если пользователь не выбрал задачу, которую хочет изменить, появится предупреждающее окно:

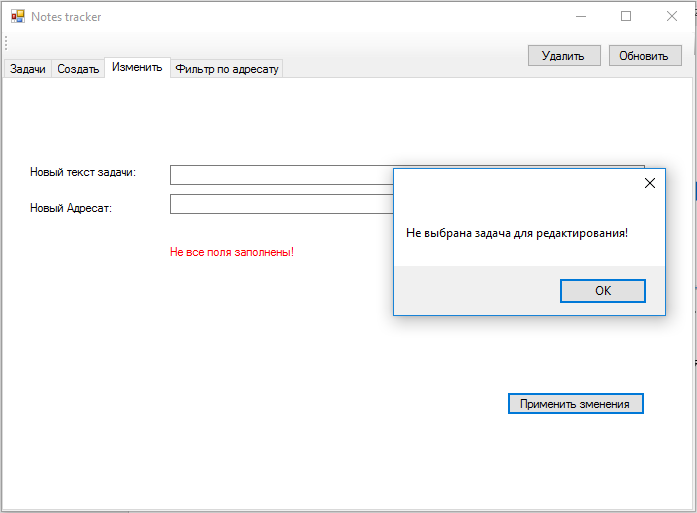


Рис.17 Предупреждающее окно при изменении задачи.

При удалении задачи, реакция аналогична, так же появится предупреждающее окно:

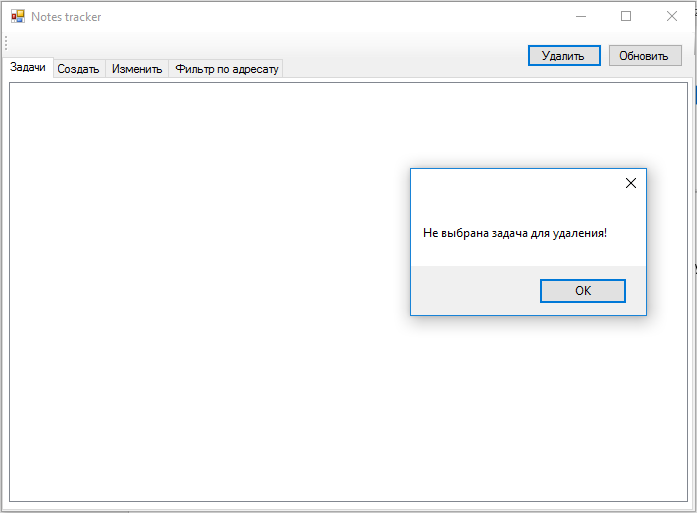


Рис 18. Предупреждающее окно при удалении задачи.

Вкладка «Фильтр по адресату» Выполнено следующим образом:

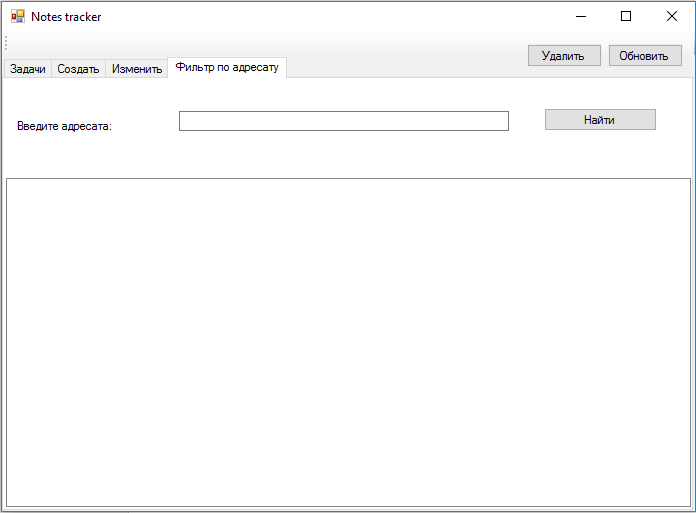


Рис 19. Вкладка «Фильтр по адресату»

**3 этап**

**разработка**

При реализации программной системы после проектирования интерфейса был решен вопрос по подключению базы данных к программе.

Была создана строка подключения, сформированы SQL запросы

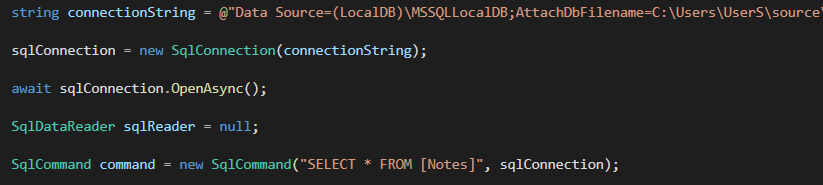


Рис 20. Подключение к базе данных

Был решен вопрос отключения от базы данных при закрытии приложения.

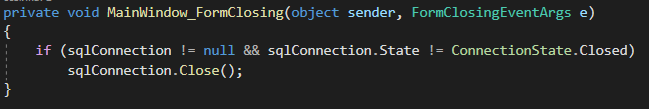


Рис 21. Отключение от базы данных

К каждой функции в программе были применены следующие шаги:

1. Подключение к базе данных
2. Формирование SQL запроса
3. Вывод информации на экран или изменение информации в базе данных
4. Отключение от базы данных

Все функции работают в асинхронном режиме, сделано это для того, чтобы быстродействие программы не зависело от работы с базой данных.

Каждая функция имеет проверку при вводе или выборе заметки.

Весь код реализован в одном файле: «LoginForm.cs»

**подключение к базе данных и вывод всех объектов в listbox.**

В первую очередь нужно узнать строку подключения, для этого в окне обозревателя серверов эта строка была скопирована из соответственного окна.

После чего создается переменная типа string, которой присваивается строка подключения, так же она переводится в формат Unicode.

Создается класс SqlConnection, у которого создается экземпляр класса, в качестве параметров которого в конструктор передается параметр с переменной, в которую мы записали строку подключения.

У класса SqlConnection есть метод .OpenAsync, мы вызовем его, но сделаем это в другом потоке.

Асинхронное открытие позволит не тормозить пользовательский интерфейс.

Для того чтобы метод стал асинхронным, мы добавим оператор async.

Таким образом мы открываем соединение с базой данных.

Далее следует получить содержимое базы данных. Для этого мы будем использовать SQL инструкцию – SELECT.

Для обращения к базе данных нам понадобится SqlDataReader, которому мы присваиваем значение null.

У SqlReader-а есть несколько очень важных методов, один из них это метод .Read, с помощью него мы сможем считать базу данных.

Нам нужно написать запрос для базы данных, для этого мы воспользуемся классом SqlCommand.

В качестве параметра в конструктор SqlCommand мы передаем 2 параметра: первый параметр- наша инструкция для базы данных, вторым параметром передаем наше соединение – SqlConnection.

Далее следует использовать структуру try/catch, в котором будет выводиться messagebox с ошибкой и ее источником.

После операции try/catch мы должны закрыть наш SqlReader дабы избежать ошибки.

В части try мы вызываем метод .ExecuteReaderAsync, после чего мы должны пройти по всей считанной информации. Для этого вызываем метод .ReadAsync

После чего начинаем заполнять наш listbox. У SqlReader-а есть свой индексатор, к которому мы обратимся. Составляем строку загрузки элементов для вывода в listox. Все данные приводим к типу string.

Далее, чтобы избежать утечки памяти, создадим событие для закрытия формы в которое передадим метод SqlConnection.Close.



Рис 22. Подключение к базе данных и вывод информации

**создание задачи**

В нашей базе данных id инкрементируется автоматически, поэтому за id следить не надо.

Используем обработчик события при нажатии на кнопку.

Теперь нужно писать только соответствующие команды.

Вызываем sqlcommand, для добавления мы используем SQL инструкцию INSERT INTO.

Обращаемся к свойствам .Parameters.AddWithWalue, куда мы передаем информацию из textbox-ов, в которые вводится информация.

Также нужно проверить на правильность ввода, если в любое из полей не вписаны значения, выводим в label текст ошибки. Делаем label невидимым, применяем к нему красный цвет, передаем в него значение в случае срабатывания сценария ошибки.

Нам остается асинхронно выполнить команду INSERT.

Далее для удобства необходимо обновить список всех задач, для этого возьмем метод обновления списка, который полностью копирует метод вывода списка задач и создадим обработчик кнопки «обновить». Таким образом, мы создали кнопку обновления списка задач и использовали его в функции обновления списка задач, после создания задачи, редактирования задачи или удаления задачи.

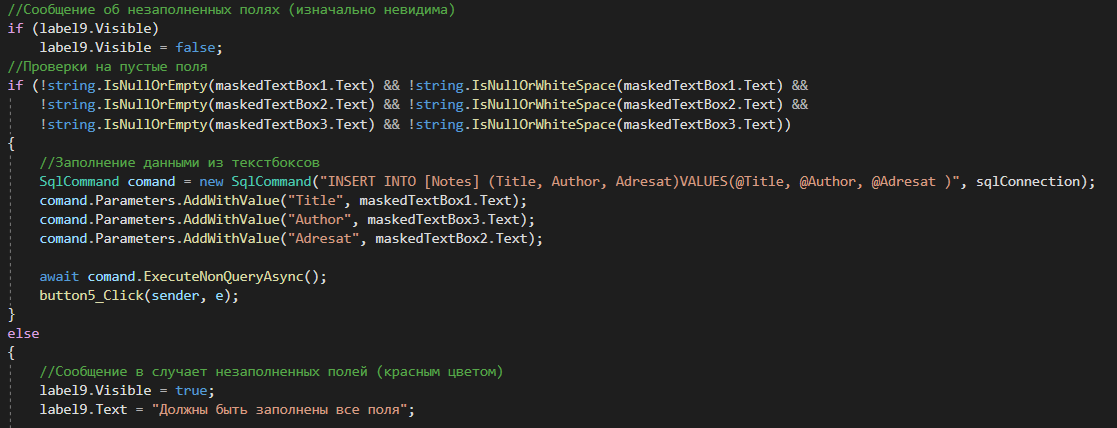


Рис 23. Создание новой задачи

**выбор элемента из списка**

Для решения момента выбора существующей задачи из списка задач, а именно его Id, мы воспользуемся методом SelectedItem, применим к нему регулярное выражение, которое оставит только Id и передадим этот Id в команду по Изменению или удалению задачи из базы данных.

Для проверки на то, что выделена строка, можно воспользоваться методом .SectedIndex.

Если строка не выделена, этот метод вернет значение минус 1.

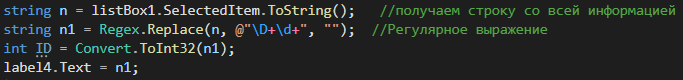


Рис 24. Выбор элемента из списка

**редактирование задачи**

Эта функция очень похожа на функцию создания задачи, только работа происходит с уже созданной задачей в базе данных, у которой мы меняем ее значения.

Используется SQL команда UPDATE.

Точно так же, как и в функции создания происходит проверка на заполненные строки и создается SQL запрос. После чего происходит вызов метода обновления списка.



Рис 25. Редактирование задачи

**фильтрация по адресату**

Данная функция основана на SQL запросе WHERE, в который мы подставляем текст, взятый из textbox-а. Для вывода информации используется listbox, который мы заполняем информацией по аналогии с функцией обновления или загрузки данных из базы данных.

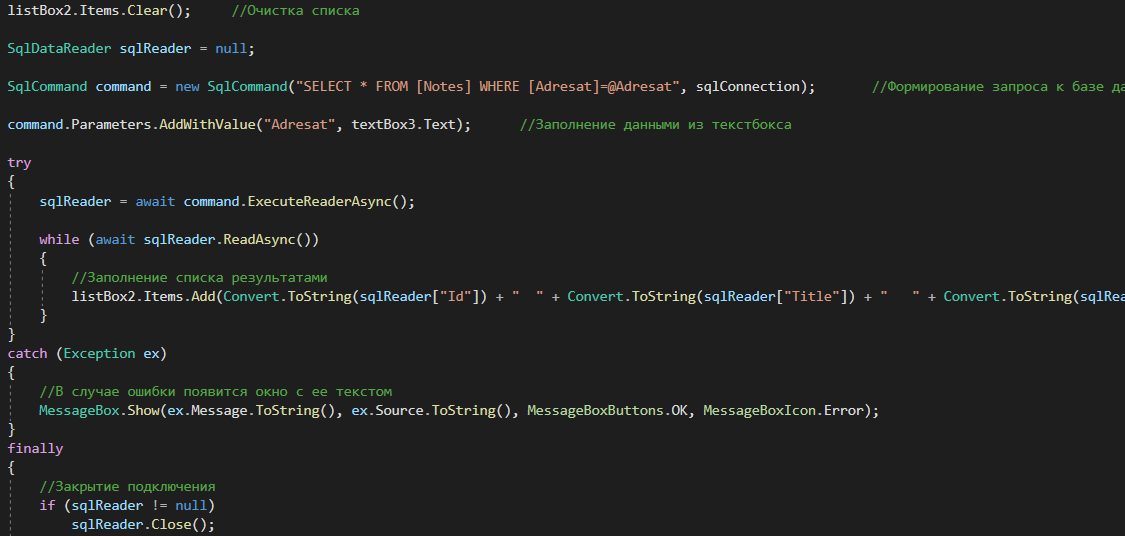


Рис 26. Фильтрация по адресату

**тестирование и отладка приложения**

При тестировании приложения были проверены все основные функции по принципу «Черного ящика»

Были проверены следующие возможности:

1. Пустой ввод
2. Единичный ввод
3. Множественный ввод
4. Выбор элемента таблицы
5. Не выбран элемент таблицы

Таблица 6. Тестирование приложения

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Исходные данные | Ожидаемый результат |  |
| Т1 | Пустой ввод | Сообщение «Не все поля заполнены» | + |
| Т2 | Число «1» | Число «1» | + |
| Т3 | Строка «Новая задача для тестирования» | Строка «Новая задача для тестирования» | + |
| Т4 | Число «12345» | Число «12345» | + |
| Т5 | Нет выбранной задачи | Сообщение об ошибке «Не выбрана задача для удаления» | + |
| Т6 | Выбрана задача | Id задачи получен | + |

Сообщения об ошибке и предупреждающие сообщения были проверены на информативность, если существует возможность в сообщении вывести причину ошибки, то она была сделана.

**Заключение**

Работа над проектом «Разработка программного модуля для управления задачами предприятия» велась по этапам.

Первым этапом стал анализ процесса и разработка требований. Выяснилось, что ранее на предприятии весь обмен задачами велся в бумажном виде, были определены недостатки такого метода и стратегия борьбы с ними. Так же были сформированы требования к приложению.

Вторым этапом стало проектирование информационной системы. Были спроектированы интерфейсы, описаны все действия, спроектирована база данных.

Третьим этапом была разработка приложения. Была создана функция подключения к базе данных, методы для реализации функционала, описаны все алгоритмы работы приложения.

Четвертый этап был тестированием приложения. По методу черного ящика были проверены все поля ввода, все реакции кнопок и прочий функционал на отказоустойчивость.

Если подвести итог:

Интуитивность системы благодаря windows forms получилась хорошая, обучение требуется короткое и не сложное.

Использование многопоточного режима работы функций программы увеличивает ее отзывчивость при работе с базой данных.

Результатом работы стала система по систематизации задач предприятия. Система выполняет свои главные функции и имеет возможность к дальнейшему развитию.

Были решены следующие задачи:

- Создание базы данных

- Создание пользовательского интерфейса

- Реализация основных функций системы (просмотр, создание, редактирование, поиск задачи по адресату, удаление)

Одним из вариантов развития системы может стать интеграция ее с одним из мессенджеров в сети интернет, к примеру возможно отсылать задачи сотрудникам напрямую по номеру телефона, чем сильно ускорить их выполнение.

Так же добавление нового функционала не составит труда по причине популярности платформы windows forms и простоте ее настройки.

В программу в дальнейшем можно внести дополнительный функционал, к примеру: статус каждой задачи, ее рейтинг среди всех задач, возможность архивирования завершенный или удаленных задач, вывод задачи на печать.

**приложение**

**исходный код программы**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.SqlClient;

using System.Text.RegularExpressions;

namespace NotesTracker

{

public partial class MainWindow : Form

{

SqlConnection sqlConnection;

public MainWindow()

{

InitializeComponent();

}

private async void MainWindow\_Load(object sender, EventArgs e)

{

string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\UserS\source\repos\NotesTracker\NotesTracker\Database.mdf;Integrated Security=True";

sqlConnection = new SqlConnection(connectionString);

await sqlConnection.OpenAsync();

SqlDataReader sqlReader = null;

SqlCommand command = new SqlCommand("SELECT \* FROM [Notes]", sqlConnection);

try

{

sqlReader = await command.ExecuteReaderAsync();

while (await sqlReader.ReadAsync())

{

listBox1.Items.Add(Convert.ToString(sqlReader["Id"]) + " " + Convert.ToString(sqlReader["Title"]) + " " + "Автор: " + Convert.ToString(sqlReader["Author"]) + " " + "Адресат: " + Convert.ToString(sqlReader["Adresat"]) + " " + Convert.ToString(sqlReader["CreateDate"]));

}

}

catch (Exception ex)

{

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

}

finally

{

if (sqlReader != null)

sqlReader.Close();

}

}

//Отключение от базы данных

private void MainWindow\_FormClosing(object sender, FormClosingEventArgs e)

{

if (sqlConnection != null && sqlConnection.State != ConnectionState.Closed)

sqlConnection.Close();

}

private async void button1\_Click(object sender, EventArgs e) //Создание новой задачи

{

//Сообщение об незаполненных полях (изначально невидима)

if (label9.Visible)

label9.Visible = false;

//Проверки на пустые поля

if (!string.IsNullOrEmpty(maskedTextBox1.Text) && !string.IsNullOrWhiteSpace(maskedTextBox1.Text) &&

!string.IsNullOrEmpty(maskedTextBox2.Text) && !string.IsNullOrWhiteSpace(maskedTextBox2.Text) &&

!string.IsNullOrEmpty(maskedTextBox3.Text) && !string.IsNullOrWhiteSpace(maskedTextBox3.Text))

{

//Заполнение данными из текстбоксов

SqlCommand comand = new SqlCommand("INSERT INTO [Notes] (Title, Author, Adresat)VALUES(@Title, @Author, @Adresat )", sqlConnection);

comand.Parameters.AddWithValue("Title", maskedTextBox1.Text);

comand.Parameters.AddWithValue("Author", maskedTextBox3.Text);

comand.Parameters.AddWithValue("Adresat", maskedTextBox2.Text);

await comand.ExecuteNonQueryAsync();

button5\_Click(sender, e);

}

else

{

//Сообщение в случает незаполненных полей (красным цветом)

label9.Visible = true;

label9.Text = "Должны быть заполнены все поля";

}

}

private async void button5\_Click(object sender, EventArgs e) //Обновление списка задач

{

listBox1.Items.Clear(); //Очистка списка

SqlDataReader sqlReader = null;

SqlCommand command = new SqlCommand("SELECT \* FROM [Notes]", sqlConnection); //Формирование запроса к базе данных

try

{

sqlReader = await command.ExecuteReaderAsync();

while (await sqlReader.ReadAsync())

{

//Заполнение списка данными

listBox1.Items.Add(Convert.ToString(sqlReader["Id"]) + " " + Convert.ToString(sqlReader["Title"]) + " " +"Автор: " + Convert.ToString(sqlReader["Author"]) + " " +"Адресат: " +Convert.ToString(sqlReader["Adresat"]) + " " + Convert.ToString(sqlReader["CreateDate"]));

}

}

catch (Exception ex)

{

//В случае ошибки появится окно с ее текстом

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

}

finally

{

if (sqlReader != null)

sqlReader.Close();

}

}

private async void button2\_Click(object sender, EventArgs e) //Изменение

{

//Проверка на то, что строка выбрана

if (listBox1.SelectedIndex == -1)

{

MessageBox.Show("Не выбрана задача для редактирования!");

return;

}

else

{

//Сообщение об незаполненных полях (изначально невидима)

if (label10.Visible)

label10.Visible = false;

//Проверки на пустые поля

if (!string.IsNullOrEmpty(textBox1.Text) && !string.IsNullOrWhiteSpace(textBox1.Text) &&

!string.IsNullOrEmpty(textBox2.Text) && !string.IsNullOrWhiteSpace(textBox2.Text))

{

string n = listBox1.SelectedItem.ToString(); //получаем строку со всей информацией

string n1 = Regex.Replace(n, @"\D+\d+", ""); //Регулярное выражение

int ID = Convert.ToInt32(n1);

label4.Text = n1;

SqlCommand command = new SqlCommand("UPDATE [Notes] SET [Title]=@Title, [Adresat]=@Adresat WHERE [Id]=@ID ", sqlConnection); //Формирование запроса к базе данных

//Заполнение данными из текстбоксов

command.Parameters.AddWithValue("ID", label4.Text);

command.Parameters.AddWithValue("Title", textBox1.Text);

command.Parameters.AddWithValue("Adresat", textBox2.Text);

await command.ExecuteNonQueryAsync();

//Обновление списка

button5\_Click(sender, e);

}

else

{

//Сообщение в случает незаполненных полей (красным цветом)

label10.Visible = true;

label10.Text = "Не все поля заполнены!";

}

}

}

private async void button3\_Click(object sender, EventArgs e) //Удаление

{

//Проверка на то, что строка выбрана

if (listBox1.SelectedIndex == -1)

{

MessageBox.Show("Не выбрана задача для удаления!");

return;

}

else

{

string n = listBox1.SelectedItem.ToString(); //получаем строку со всей информацией

string n1 = Regex.Replace(n, @"\D+\d+", ""); //Регулярное выражение

int ID = Convert.ToInt32(n1);

label4.Text = n1;

SqlCommand command = new SqlCommand("DELETE FROM [Notes] WHERE [Id]=@ID", sqlConnection);

command.Parameters.AddWithValue("ID", label4.Text);

await command.ExecuteNonQueryAsync();

button5\_Click(sender, e);

}

}

private void tabPage2\_Click(object sender, EventArgs e)

{

}

//Фильтр по адресату

private async void button4\_Click(object sender, EventArgs e) //Поиск по адресату

{

listBox2.Items.Clear(); //Очистка списка

SqlDataReader sqlReader = null;

SqlCommand command = new SqlCommand("SELECT \* FROM [Notes] WHERE [Adresat]=@Adresat", sqlConnection); //Формирование запроса к базе данных

command.Parameters.AddWithValue("Adresat", textBox3.Text); //Заполнение данными из текстбокса

try

{

sqlReader = await command.ExecuteReaderAsync();

while (await sqlReader.ReadAsync())

{

//Заполнение списка результатами

listBox2.Items.Add(Convert.ToString(sqlReader["Id"]) + " " + Convert.ToString(sqlReader["Title"]) + " " + Convert.ToString(sqlReader["Author"]) + " " + Convert.ToString(sqlReader["Adresat"]) + " " + Convert.ToString(sqlReader["CreateDate"]));

}

}

catch (Exception ex)

{

//В случае ошибки появится окно с ее текстом

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

}

finally

{

//Закрытие подключения

if (sqlReader != null)

sqlReader.Close();

}

}

}

}

**руководство пользователя**

**функция создания задачи**

Для создания задачи необходимо:

1. Нажать на вкладку «Создать»
2. Ввести текст задачи в строку ввода, обозначенную «Текст задачи»
3. Ввести Адресата в строку ввода, обозначенную «Адресат»
4. Ввести Автора в строку ввода, обозначенную «Автор»
5. Нажать на кнопку «Создать»

**функция изменения задачи**

Для изменения задачи необходимо:

1. Выделить задачу мышкой, которую необходимо изменить, на главном окне программы
2. Перейти на вкладку «Изменить»
3. Ввести новый тест в строку ввода, обозначенную «Новый текст задачи»
4. Ввести нового адресата в строку, обозначенную «Новый Адресат»
5. Нажать на кнопку «Применить изменения»

**функция удаления задачи**

Для удаления задачи необходимо:

1. Выделить задачу мышкой, которую необходимо удалить, на главном окне программы
2. Нажать на кнопку «Удалить»

**функция фильтрации (поиска) по адресату**

Для поиска по адресату необходимо:

1. Перейти на вкладку «Фильтр по адресату»
2. Ввести имя адресата в строку, обозначенную «Введите адресата»
3. Нажать на кнопку «Найти»

**возможные информационные сообщения:**

1. «Не выбрана задача для удаления» - В главном окне программы не выбрана кнопкой мышки задача, которую необходимо удалить.
2. «Не все поля заполнены» (выделяется красным цветом) – В одном или нескольких полях не вписаны необходимые данные.
3. Окно с ошибкой и ее расшифровка – проблема с подключением к базе данных.