

**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ (МИИТ)**

**ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ УПРАВЛЕНИЯ**

**Курсовая работа**

**на тему**

**«Разработка программного обеспечения конвертации печатных форм»**

**по дисциплине** «**Методы программирования**»

**Выполнил:** ст. гр. ТКИ-342

Никулин Д.В.

Ситало Р.В.

**Проверил:** доцент,к.т.н.

Сафронов А.И.

**Москва – 2024 г**

Оглавление

[Введение 3](#_Toc166603944)

[Цель работы 4](#_Toc166603945)

[Содержательная часть 5](#_Toc166603946)

[Диаграммы классов, входящих в состав решения 9](#_Toc166603947)

[Полная сеть Петри 10](#_Toc166603948)

[Работа программы 11](#_Toc166603949)

[Код программы 16](#_Toc166603950)

[ChartForm 16](#_Toc166603951)

[MainForm 17](#_Toc166603952)

[DB 23](#_Toc166603953)

[Eextrc 25](#_Toc166603954)

[Wextrc 28](#_Toc166603955)

[Вывод 36](#_Toc166603956)

# Введение

В данной работе рассмотрена разработка программного обеспечения, способного обрабатывать печатные формы установленного формата, содержащие рапорта на замену занятий по болезни. Программа позволит пользователю выполнить предпросмотр табличного расписания, его конвертацию в различные форматы, например, формат Excel, Word. Программа будет выводить график замен.

# Цель работы

Целью данной работы является разработка программного обеспечения, способного обрабатывать формы с расписанием, содержащимся в файлах .docx, уметь конвертировать их в .xlsx, .docx. по файлу .docx программа должна уметь воссоздать файл .xls, а также построить график на столбчатой диаграмме общее количество часов, заменённых преподавателями по болезни.

# Содержательная часть

В главной форме MainForm программы eWExtractor реализовано приложение для извлечения данных из документов и их обработки. Вот как работает программа:

1. **Инициализация компонентов и базы данных**: В конструкторе MainForm() происходит инициализация всех компонентов интерфейса пользователя, включая панель для перетаскивания файлов, кнопки для загрузки файла, отображения справки, сохранения данных в форматах Word и Excel, добавления данных в базу данных SQLite и отображения диаграммы. Также создается экземпляр DatabaseManager для работы с базой данных SQLite.
2. **Обработка событий перетаскивания файла**: Два метода DragDropPanel\_DragEnter и DragDropPanel\_DragDrop отвечают за обработку событий перетаскивания файла на панель. После перетаскивания файла выполняется его обработка в зависимости от формата (Word или Excel).
3. **Обработка событий загрузки файла**: Метод BtnLoadFile\_Click вызывается при нажатии кнопки "Загрузить файл". Пользователю предоставляется возможность выбрать файл для загрузки, после чего он обрабатывается.
4. **Отображение извлеченных данных**: Метод DisplayExtractedData отображает извлеченные данные из документа в двух метках MainLab и AppLab, соответствующих основной и дополнительной информации соответственно.
5. **Сохранение данных в форматах Word и Excel**: Методы BtnSaveToWord\_Click и BtnSaveToExcel\_Click сохраняют данные в форматах Word и Excel соответственно, используя извлеченные данные и путь к исходному файлу.
6. **Отображение справки**: Метод BtnHelp\_Click отображает окно со справкой по использованию программы.
7. **Добавление данных в базу данных SQLite**: Метод BtnAddToSQLite\_Click добавляет извлеченные данные в базу данных SQLite через экземпляр DatabaseManager.
8. **Отображение диаграммы**: Методы BtnShowChart\_Click и ShowReplacementChart отображают диаграмму замен преподавателей в новом окне ChartForm.

Для извлечения данных из документов Word в приложении использовался пакет Microsoft.Office.Interop.Word. Вот какие данные были извлечены из документов Word и как работает класс WordExtractor:

1. **Метод GetExtractedMainText**: Этот метод извлекает основные данные из документа Word. Он вызывает метод ExtractTextFromWordDocument для извлечения текста из документа и метод ExtractDataFromMainText для извлечения конкретных данных, таких как подписи и информация о болезни.
2. **Метод GetExtractedAddText**: Этот метод извлекает дополнительные данные из документа Word. Он также вызывает метод ExtractTextFromWordDocument для извлечения текста из документа и метод ExtractDataFromAddText для извлечения информации о заменах преподавателей.
3. **Метод SaveDataToWord**: Этот метод сохраняет данные в документ Word. Он использует шаблонный файл Temple.docx и заполняет его извлеченными данными с помощью метода FillTemplate.
4. **Метод ExtractTextFromWordDocument**: Этот метод открывает документ Word и извлекает текст из него, используя Microsoft.Office.Interop.Word.Application.
5. **Методы ExtractDataFromMainText и ExtractDataFromAddText**: Эти методы извлекают конкретные данные из текста документа Word, такие как подписи, даты и информацию о замене преподавателей, используя регулярные выражения.
6. **Метод FillTemplate**: Этот метод заполняет шаблонный файл документа Word данными извлеченными из исходного документа. Он также заменяет плейсхолдеры в документе на соответствующие значения.
7. **Дополнительные методы**: В классе также присутствуют дополнительные методы для работы с текстом, такие как InsertTextInDocument для вставки текста в документ и FindAndReplace для поиска и замены текста.

Класс DatabaseManager отвечает за управление базой данных SQLite. Давай разберем, что он делает:

1. **Конструктор DatabaseManager**: Принимает путь к файлу базы данных SQLite и инициализирует соединение с базой данных, а также создает необходимые таблицы, если они еще не существуют.
2. **Метод InitializeDatabase**: Создает две таблицы в базе данных: MainData для хранения основных данных и RepData для хранения дополнительных данных о замене преподавателей.
3. **Метод InsertMainData**: Вставляет основные данные в таблицу MainData, такие как подпись, должность, полное имя, даты и лист нетрудоспособности. Затем сохраняет id вставленной записи для использования в методе InsertRepData.
4. **Метод InsertRepData**: Вставляет дополнительные данные о замене преподавателей в таблицу RepData. Использует сохраненный mainDataId, чтобы связать дополнительные данные с основной записью.
5. **Метод ExecuteNonQuery**: Выполняет SQL-запрос, который не возвращает результат.
6. **Метод GetReplacementCount**: Возвращает количество записей в таблице RepData для определенного преподавателя.

Этот класс обеспечивает взаимодействие с базой данных SQLite для хранения данных, извлеченных из документов Word.

Этот функционал программы обеспечивает удобное извлечение, обработку и сохранение данных из различных типов документов, а также их анализ с помощью диаграммы.

# Диаграммы классов, входящих в состав решения

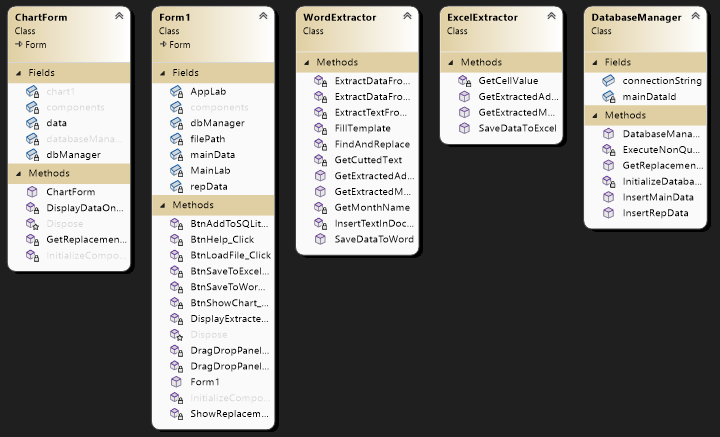


Рисунок 1 – Диаграмма классов

# Полная сеть Петри

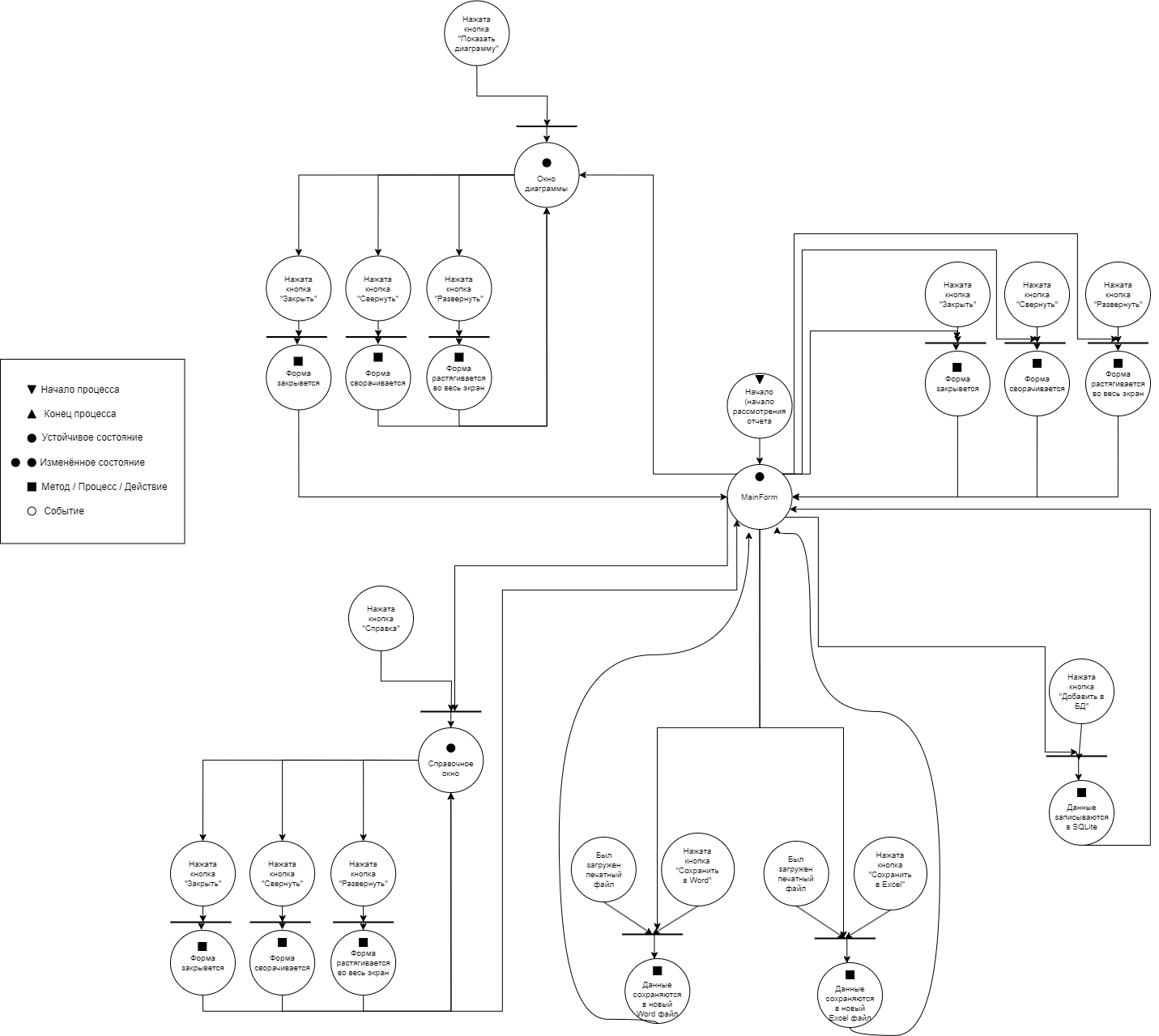


Рисунок 2 – Полная сеть Петри

# Работа программы

На рисунке 4 представлена главная форма программы с несколькими элементами управления.

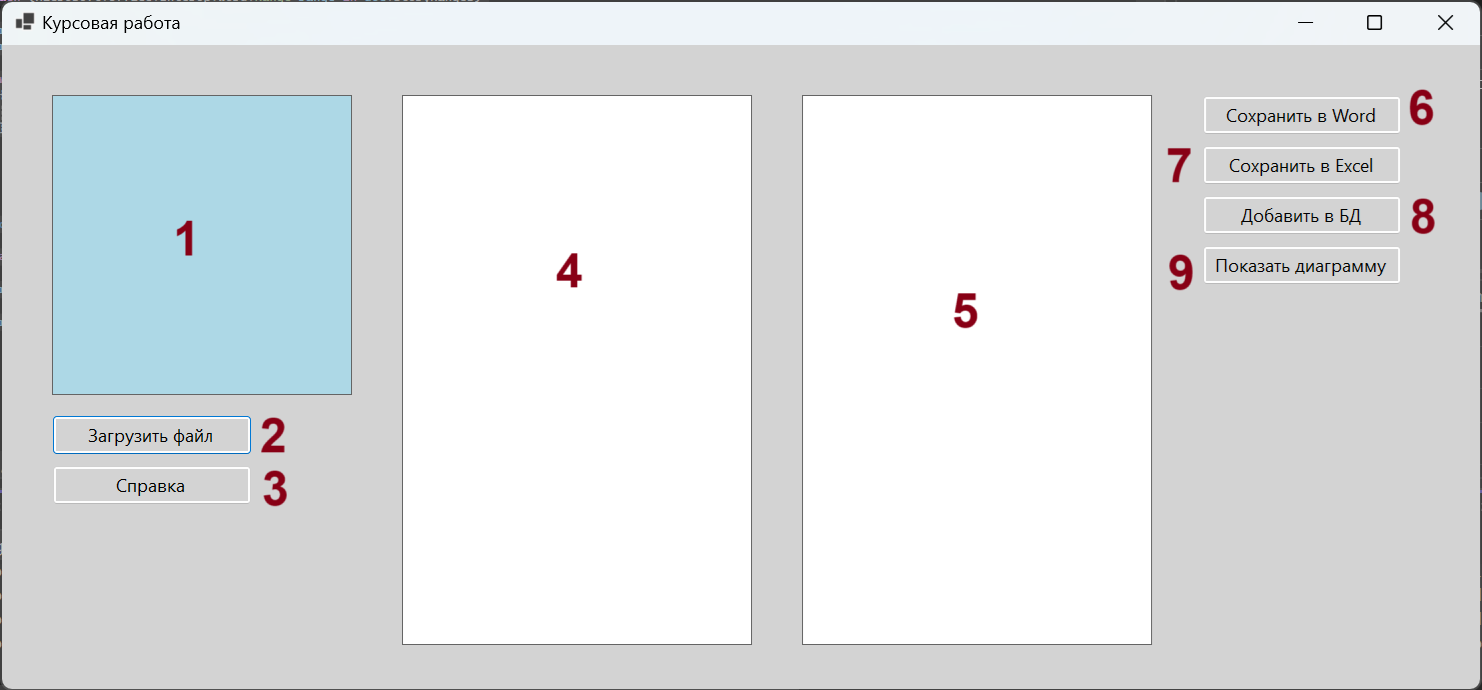


Рисунок 4 – Окно программы

Под номером 1 на рисунке 4 представлен элемент Drag and Drop, который позволяет пользователю перетаскивать файлы в приложение для загрузки.

Под номером 2 на рисунке 4 находится кнопка "Загрузить файл", которая предоставляет альтернативный способ загрузки файлов в приложение, отличный от функциональности Drag and Drop. Под номером 3 на рисунке 4 находится кнопка "Справка". Содержимое справки, представленное на рисунке 5, включает описание использования приложения, включая инструкции по загрузке файлов, просмотру данных и сохранению в различных форматах

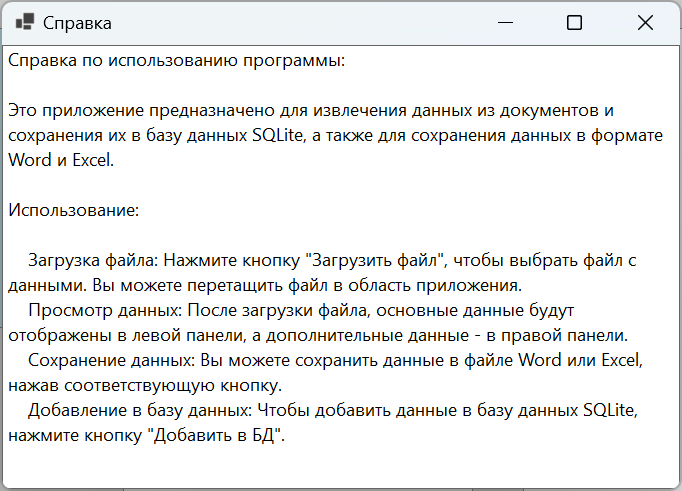


Рисунок 5 – Справка

В label под номером 4 выводится информация о преподавателе, которого заменяют, а в label под номером 5 выводится информация о преподавателе, который заменяет другого. Пример работы представлен на рисунке 6.

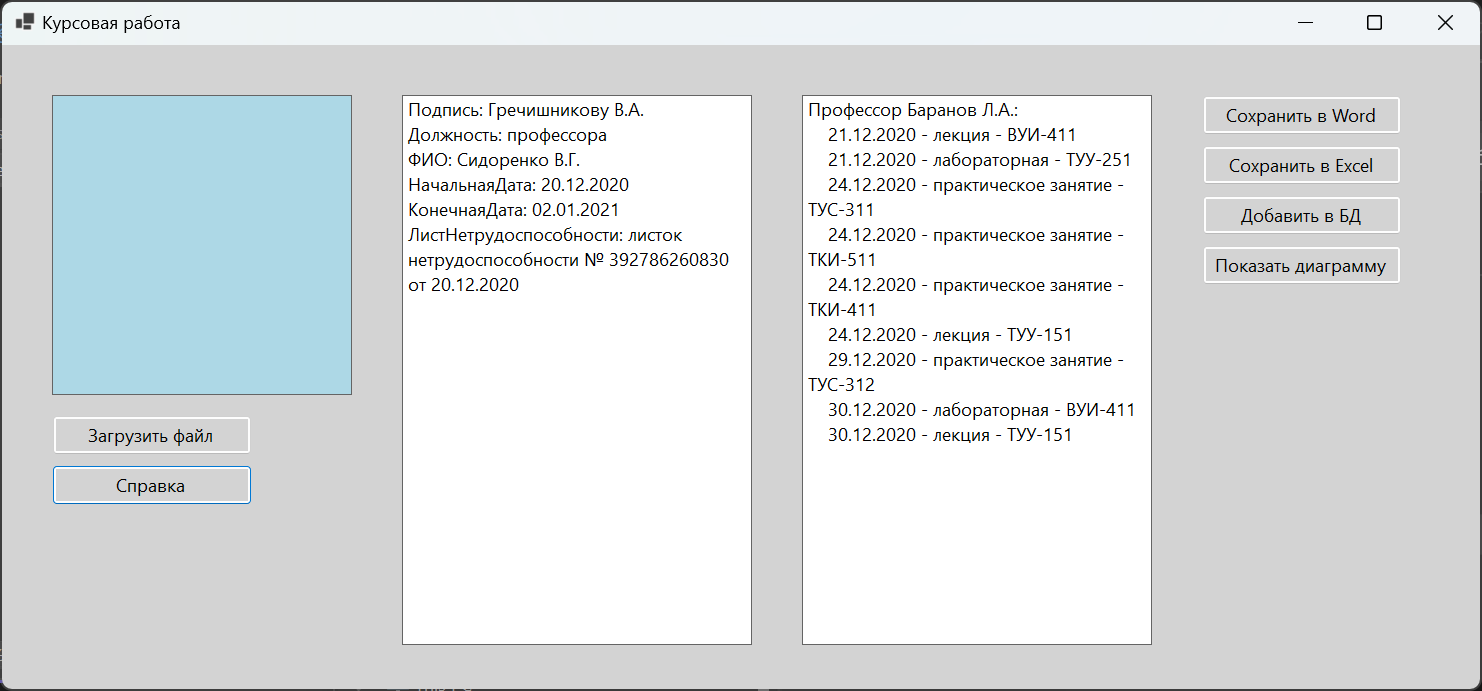


Рисунок 6 – Результат загрузки word/excel файла

Под номерами 6 и 7 рисунка 4 находятся кнопки "Сохранить в Word" и "Сохранить в Excel" соответственно. Они предназначены для сохранения информации, которая была извлечена и отображена в метках под номерами 4 и 5, в соответствующих форматах файлов. Эти кнопки позволяют пользователям сохранить данные в удобном для них формате, чтобы обеспечить их доступность и возможность дальнейшего использования. Пример работы этих функций можно увидеть на рисунках 7 и 8.



Рисунок 7 – Содержимое word файла

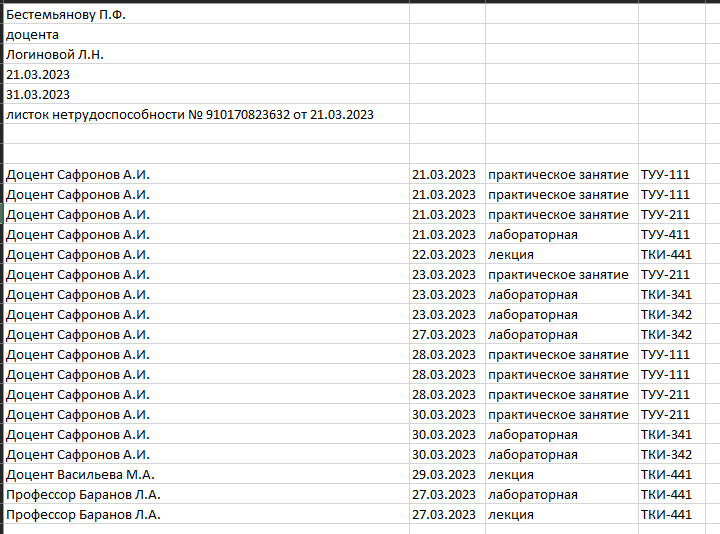


Рисунок 8 – Содержимое excel файла

Под номером 8 находится функция записи данных в базу данных SQLite для последующего использования в создании диаграммы. Этот процесс включает сохранение данных из меток под номерами 4 и 5 в базу данных, чтобы затем можно было создать диаграмму на основе этих данных. Структура двух таблиц базы данных представлена на рисунках 9 и 10.

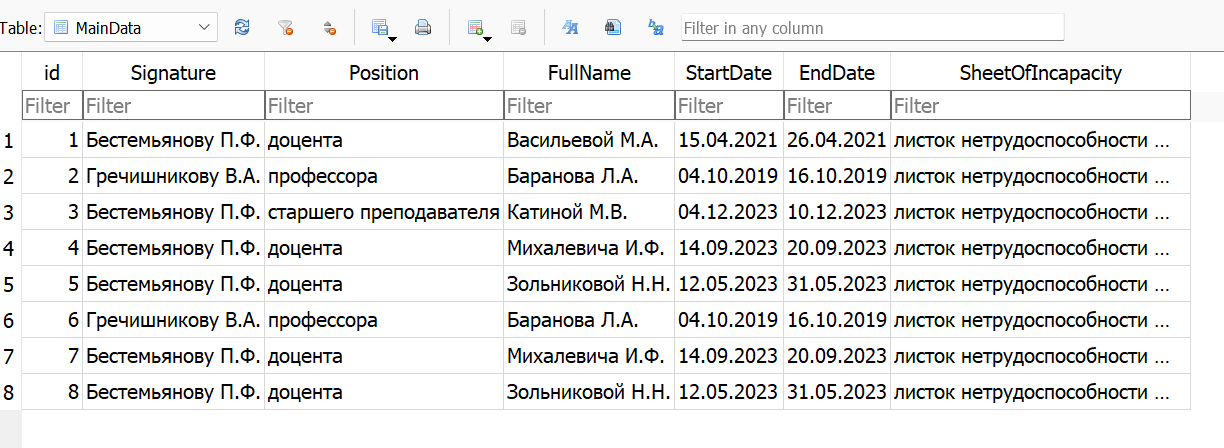


Рисунок 9 – Таблица MainData

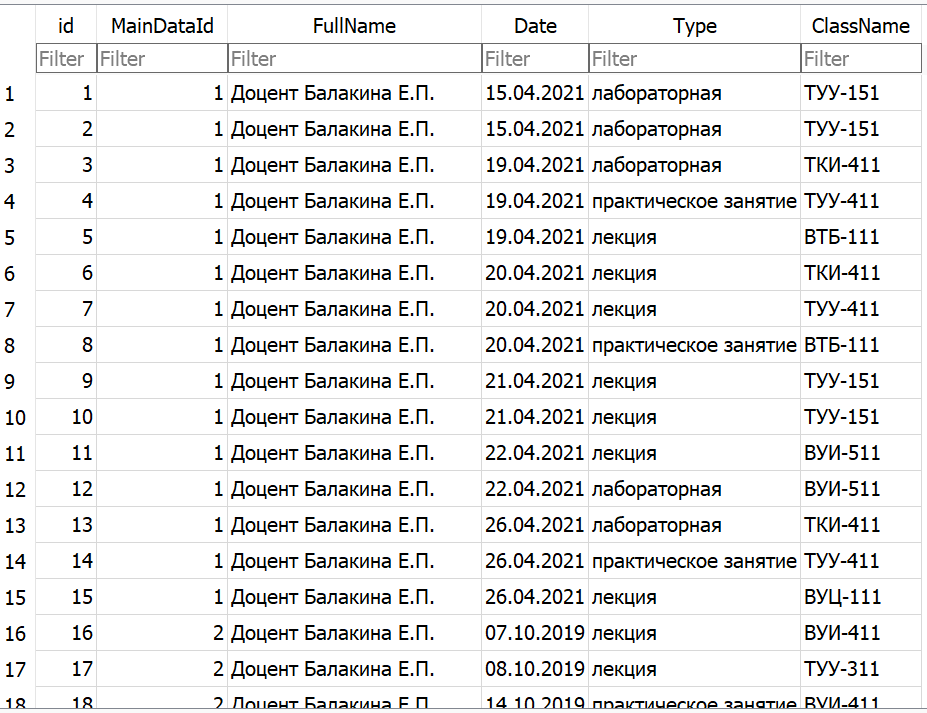


Рисунок 10 – Таблица RepData

Кнопка "Показать диаграмму" открывает новое окно и отображает в нем диаграмму, созданную на основе данных из базы данных. Рисунок номер 11 демонстрирует это новое окно с отображенной диаграммой.

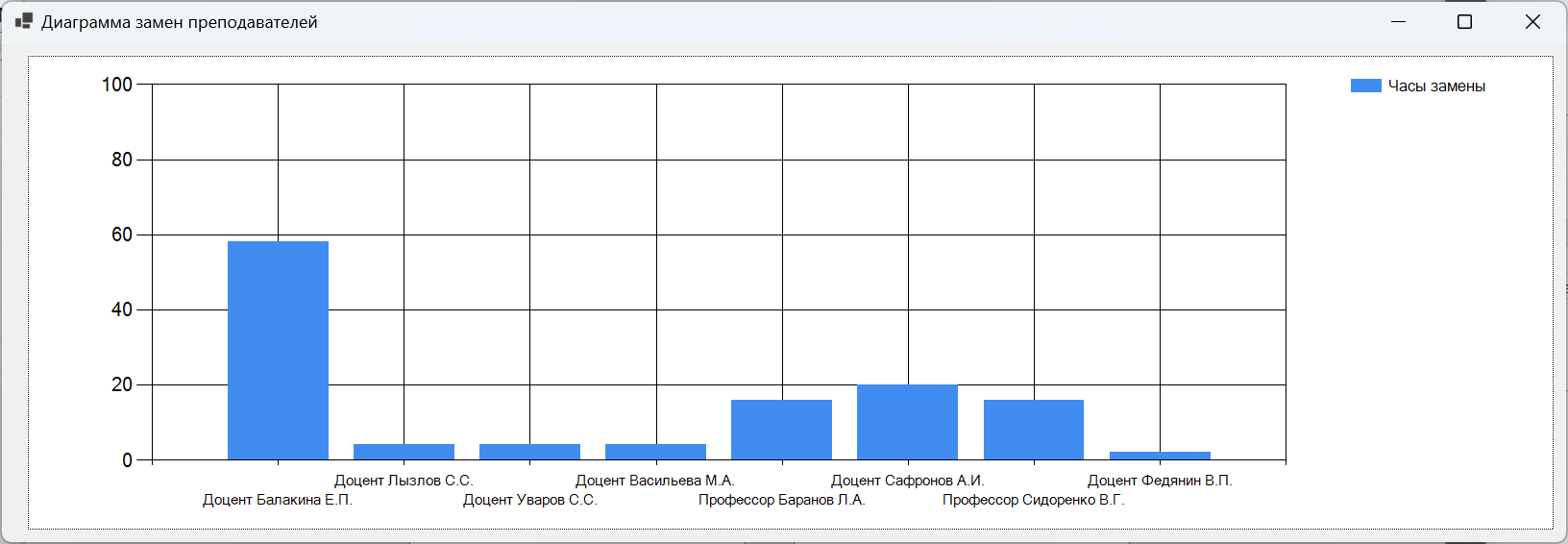
****

Рисунок 11 – Диагармма замен преподавателей

# Код программы

## ChartForm

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace eWExtractor

{

public partial class ChartForm : Form

{

DatabaseManager dbManager = new DatabaseManager("MyDatabase.sqlite");

Dictionary<string, int> data;

public ChartForm()

{

InitializeComponent();

data = GetReplacementInfo();

// Вызываем метод для отображения данных на графике

DisplayDataOnChart();

}

private void DisplayDataOnChart()

{

// Очищаем график перед добавлением новых данных

chart1.Series.Clear();

// Создаем новую серию для графика

Series series = new Series("Часы замены");

series.ChartType = SeriesChartType.Column; // Устанавливаем тип диаграммы на столбчатую

// Добавляем данные из словаря на график

int i = 0;

foreach (var kvp in data)

{

// Добавляем точку данных в серию с уникальным значением X

series.Points.Add(new DataPoint(i, kvp.Value) { AxisLabel = kvp.Key });

i++; // Увеличиваем значение i для следующей точки данных

}

// Добавляем серию на график

chart1.Series.Add(series);

// Настраиваем ось Y для автоматического масштабирования

chart1.ChartAreas[0].AxisY.Minimum = 0;

chart1.ChartAreas[0].AxisY.Maximum = 100;

}

private Dictionary<string, int> GetReplacementInfo()

{

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

// Подключаемся к базе данных SQLite

using (SQLiteConnection connection = new SQLiteConnection(dbManager.connectionString))

{

connection.Open();

// Получаем уникальные значения FullName из таблицы RepData

string query = "SELECT DISTINCT FullName FROM RepData";

using (SQLiteCommand command = new SQLiteCommand(query, connection))

{

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

string fullName = reader.GetString(0);

// Подсчитываем количество записей для каждого преподавателя в таблице RepData

int count = dbManager.GetReplacementCount(connection, fullName);

// Умножаем количество записей на 2

int replacementHours = count \* 2;

// Добавляем информацию в словарь

replacementInfo.Add(fullName, replacementHours);

}

}

}

}

return replacementInfo;

}

}

}

## MainForm

namespace eWExtractor

{

public partial class MainForm : Form

{

private string filePath;

private Dictionary<string, string> mainData;

private Dictionary<string, List<(string, string, string)>> repData;

private Label MainLab; // Объявление label1 как поле класса

private Label AppLab; // Объявление label2 как поле класса

private DatabaseManager dbManager;

public MainForm()

{

InitializeComponent();

dbManager = new DatabaseManager("MyDatabase.sqlite");

this.Text = "Курсовая работа";

this.Size = new System.Drawing.Size(1500, 700);

this.BackColor = System.Drawing.Color.LightGray;

// Создание и настройка панели для перетаскивания файлов

Panel dragDropPanel = new Panel();

dragDropPanel.Location = new System.Drawing.Point(50, 50);

dragDropPanel.Size = new System.Drawing.Size(300, 300);

dragDropPanel.BackColor = System.Drawing.Color.LightBlue;

dragDropPanel.BorderStyle = BorderStyle.FixedSingle;

dragDropPanel.AllowDrop = true;

dragDropPanel.DragEnter += DragDropPanel\_DragEnter;

dragDropPanel.DragDrop += DragDropPanel\_DragDrop;

this.Controls.Add(dragDropPanel);

// Создание и настройка кнопки для загрузки файла

Button btnLoadFile = new Button();

btnLoadFile.Text = "Загрузить файл";

btnLoadFile.Location = new System.Drawing.Point(50, 370);

btnLoadFile.Size = new System.Drawing.Size(200, 40);

btnLoadFile.Click += BtnLoadFile\_Click;

this.Controls.Add(btnLoadFile);

// Создаем кнопку для справки

Button btnHelp = new Button();

btnHelp.Text = "Справка";

btnHelp.Location = new System.Drawing.Point(50, 420);

btnHelp.Size = new System.Drawing.Size(200, 40);

// Привязываем обработчик события Click к кнопке

btnHelp.Click += BtnHelp\_Click;

// Добавляем кнопку на форму

this.Controls.Add(btnHelp);

this.MainLab = new Label(); // Присвоение label1 полю класса

this.MainLab.AutoSize = false;

this.MainLab.TextAlign = ContentAlignment.TopLeft;

this.MainLab.AutoEllipsis = true;

this.MainLab.Location = new System.Drawing.Point(400, 50);

this.MainLab.Size = new System.Drawing.Size(350, 550); // Изменение расположения и размера

this.MainLab.BackColor = System.Drawing.Color.White;

this.MainLab.BorderStyle = BorderStyle.FixedSingle;

this.Controls.Add(this.MainLab);

this.AppLab = new Label(); // Присвоение label2 полю класса

this.AppLab.AutoSize = false;

this.AppLab.TextAlign = ContentAlignment.TopLeft;

this.AppLab.AutoEllipsis = true;

this.AppLab.Location = new System.Drawing.Point(800, 50); // Перемещение label2 вправо

this.AppLab.Size = new System.Drawing.Size(350, 550);

this.AppLab.BackColor = System.Drawing.Color.White;

this.AppLab.BorderStyle = BorderStyle.FixedSingle;

this.Controls.Add(this.AppLab);

// Кнопка для сохранения в Word

Button btnSaveToWord = new Button();

btnSaveToWord.Text = "Сохранить в Word";

btnSaveToWord.Location = new System.Drawing.Point(50, 420); // Расположение кнопки

btnSaveToWord.Size = new System.Drawing.Size(150, 30); // Размер кнопки

btnSaveToWord.Location = new System.Drawing.Point(1200, 50); // Перемещение label2 вправо

btnSaveToWord.Size = new System.Drawing.Size(200, 40);

btnSaveToWord.Click += BtnSaveToWord\_Click; // Обработчик события нажатия на кнопку

this.Controls.Add(btnSaveToWord);

// Кнопка для сохранения в Excel

Button btnSaveToExcel = new Button();

btnSaveToExcel.Text = "Сохранить в Excel";

btnSaveToExcel.Location = new System.Drawing.Point(50, 420); // Расположение кнопки

btnSaveToExcel.Size = new System.Drawing.Size(150, 30); // Размер кнопки

btnSaveToExcel.Location = new System.Drawing.Point(1200, 100); // Перемещение label2 вправо

btnSaveToExcel.Size = new System.Drawing.Size(200, 40);

btnSaveToExcel.Click += BtnSaveToExcel\_Click; // Обработчик события нажатия на кнопку

this.Controls.Add(btnSaveToExcel);

// Кнопка для добавления в SQLite

Button btnAddToSQLite = new Button();

btnAddToSQLite.Text = "Добавить в БД";

btnAddToSQLite.Size = new System.Drawing.Size(150, 30); // Размер кнопки

btnAddToSQLite.Location = new System.Drawing.Point(1200, 150); // Перемещение label2 вправо

btnAddToSQLite.Size = new System.Drawing.Size(200, 40);

btnAddToSQLite.Click += BtnAddToSQLite\_Click; // Обработчик события нажатия на кнопку

this.Controls.Add(btnAddToSQLite);

// Кнопка для отображения диаграммы

Button btnShowChart = new Button();

btnShowChart.Text = "Показать диаграмму";

btnShowChart.Size = new System.Drawing.Size(200, 40); // Размер кнопки

btnShowChart.Location = new System.Drawing.Point(1200, 200); // Перемещение label2 вправо

btnShowChart.Click += BtnShowChart\_Click;

this.Controls.Add(btnShowChart);

}

// Обработчик события перетаскивания файлов на панель

private void DragDropPanel\_DragEnter(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

e.Effect = DragDropEffects.Copy;

else

e.Effect = DragDropEffects.None;

}

// Обработчик события перетаскивания файла на панель

private void DragDropPanel\_DragDrop(object sender, DragEventArgs e)

{

string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

if (files.Length > 0)

{

// Предполагаем, что перетаскивается только один файл

filePath = files[0];

// Проверка расширения файла

string extension = Path.GetExtension(filePath);

if (extension != null)

{

if (extension.Equals(".docx"))

{

mainData = WordExtractor.GetExtractedMainText(filePath);

repData = WordExtractor.GetExtractedAddText(filePath);

}

else if (extension.Equals(".xlsx"))

{

mainData = ExcelExtractor.GetExtractedMainData(filePath);

repData = ExcelExtractor.GetExtractedAddData(filePath);

}

else

{

MessageBox.Show("Неподдерживаемый формат файла. Поддерживаются только файлы .docx и .xlsx.", "Неподдерживаемый файл", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DisplayExtractedData();

}

}

}

// Обработчик события нажатия кнопки для загрузки файла

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

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Файлы Word (\*.docx)|\*.docx|Файлы Excel (\*.xlsx)|\*.xlsx|Все файлы (\*.\*)|\*.\*";

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

{

filePath = openFileDialog.FileName;

// Проверка расширения файла

string extension = Path.GetExtension(filePath);

if (extension != null)

{

if (extension.Equals(".docx"))

{

mainData = WordExtractor.GetExtractedMainText(filePath);

repData = WordExtractor.GetExtractedAddText(filePath);

}

else if (extension.Equals(".xlsx"))

{

mainData = ExcelExtractor.GetExtractedMainData(filePath);

repData = ExcelExtractor.GetExtractedAddData(filePath);

}

else

{

MessageBox.Show("Неподдерживаемый формат файла. Поддерживаются только файлы .docx и .xlsx.", "Неподдерживаемый файл", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DisplayExtractedData();

}

}

}

private void DisplayExtractedData()

{

// Проверяем, что label1 и label2 были созданы

if (MainLab != null && AppLab != null)

{

// Отображаем данные из extractedData в label1

MainLab.Text = ""; // Очищаем label1 перед добавлением новых данных

foreach (var kvp in mainData)

{

MainLab.Text += $"{kvp.Key}: {kvp.Value}\r\n";

}

// Отображаем данные из extractedData2 в label2

AppLab.Text = ""; // Очищаем label2 перед добавлением новых данных

// Выводим данные из extractedData2 в label2

foreach (var kvp in repData)

{

AppLab.Text += $"{kvp.Key}:\r\n"; // Выводим ключ

// Выводим данные о заменяющих преподавателях

foreach (var lessonData in kvp.Value)

{

AppLab.Text += $" {lessonData.Item1} - {lessonData.Item2} - {lessonData.Item3}\r\n";

}

}

}

}

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

{

WordExtractor.SaveDataToWord(filePath, mainData, repData);

MessageBox.Show("Данные успешно сохранены в файле Word.", "Сохранение завершено", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

// Вызываем функцию сохранения в Excel и передаем ей путь к файлу, основные данные и данные о замене

ExcelExtractor.SaveDataToExcel(filePath, mainData, repData);

// Показываем сообщение об успешном сохранении

MessageBox.Show("Данные успешно сохранены в файле Excel.", "Сохранение завершено", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

// Создаем новое окно с описанием

Form helpForm = new Form();

helpForm.Text = "Справка";

helpForm.Size = new System.Drawing.Size(700, 500);

// Создаем метку для отображения содержания

Label helpLabel = new Label();

helpLabel.AutoSize = false;

helpLabel.Dock = DockStyle.Fill;

helpLabel.TextAlign = ContentAlignment.TopLeft;

helpLabel.BackColor = Color.White;

helpLabel.BorderStyle = BorderStyle.FixedSingle;

helpLabel.Text = "Справка по использованию программы:\r\n\r\nЭто приложение предназначено для извлечения данных из документов и сохранения их в базу данных SQLite, а также для сохранения данных в формате Word и Excel.\r\n\r\nИспользование:\r\n\r\n Загрузка файла: Нажмите кнопку \"Загрузить файл\", чтобы выбрать файл с данными. Вы можете перетащить файл в область приложения.\r\n Просмотр данных: После загрузки файла, основные данные будут отображены в левой панели, а дополнительные данные - в правой панели.\r\n Сохранение данных: Вы можете сохранить данные в файле Word или Excel, нажав соответствующую кнопку.\r\n Добавление в базу данных: Чтобы добавить данные в базу данных SQLite, нажмите кнопку \"Добавить в БД\".";

// Добавляем метку на форму

helpForm.Controls.Add(helpLabel);

// Отображаем окно

helpForm.ShowDialog();

}

////////////SQLite/////////////

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

{

dbManager.InsertMainData(mainData);

dbManager.InsertRepData(repData);

}

////////////SQLite/////////////

////////////Charting/////////////

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

{

ShowReplacementChart();

}

private void ShowReplacementChart()

{

// Создаем экземпляр формы Chart

ChartForm chartForm = new ChartForm();

// Устанавливаем текст формы

chartForm.Text = "Диаграмма замен преподавателей";

// Отображаем окно с диаграммой

chartForm.ShowDialog();

}

////////////Charting/////////////

}

}

## DB

using System;

using System.Collections.Generic;

using System.Data.SQLite;

public class DatabaseManager

{

public string connectionString;

private int mainDataId = -1;

public DatabaseManager(string dbFilePath)

{

connectionString = $"Data Source={dbFilePath};Version=3;";

InitializeDatabase();

}

private void InitializeDatabase()

{

using (SQLiteConnection connection = new SQLiteConnection(connectionString))

{

connection.Open();

// Создаем таблицу для mainData

string createMainDataTableQuery = @"CREATE TABLE IF NOT EXISTS MainData (

id INTEGER PRIMARY KEY AUTOINCREMENT,

Signature TEXT,

Position TEXT,

FullName TEXT,

StartDate TEXT,

EndDate TEXT,

SheetOfIncapacity TEXT);";

ExecuteNonQuery(connection, createMainDataTableQuery);

// Создаем таблицу для repData

string createRepDataTableQuery = @"CREATE TABLE IF NOT EXISTS RepData (

id INTEGER PRIMARY KEY AUTOINCREMENT,

MainDataId INTEGER,

FullName TEXT,

Date TEXT,

Type TEXT,

ClassName TEXT,

FOREIGN KEY (MainDataId) REFERENCES MainData(id));";

ExecuteNonQuery(connection, createRepDataTableQuery);

}

}

public void InsertMainData(Dictionary<string, string> mainData)

{

using (SQLiteConnection connection = new SQLiteConnection(connectionString))

{

connection.Open();

string insertQuery = @"INSERT INTO MainData (Signature, Position, FullName, StartDate, EndDate, SheetOfIncapacity)

VALUES (@Signature, @Position, @FullName, @StartDate, @EndDate, @SheetOfIncapacity);";

SQLiteCommand command = new SQLiteCommand(insertQuery, connection);

command.Parameters.AddWithValue("@Signature", mainData["Подпись"]);

command.Parameters.AddWithValue("@Position", mainData["Должность"]);

command.Parameters.AddWithValue("@FullName", mainData["ФИО"]);

command.Parameters.AddWithValue("@StartDate", mainData["НачальнаяДата"]);

command.Parameters.AddWithValue("@EndDate", mainData["КонечнаяДата"]);

command.Parameters.AddWithValue("@SheetOfIncapacity", mainData["ЛистНетрудоспособности"]);

command.ExecuteNonQuery();

// Получаем id после вставки

insertQuery = "SELECT last\_insert\_rowid()";

command = new SQLiteCommand(insertQuery, connection);

mainDataId = Convert.ToInt32(command.ExecuteScalar());

}

}

public void InsertRepData(Dictionary<string, List<(string, string, string)>> repData)

{

using (SQLiteConnection connection = new SQLiteConnection(connectionString))

{

connection.Open();

foreach (var kvp in repData)

{

string teacherName = kvp.Key;

foreach (var lessonData in kvp.Value)

{

string insertQuery = @"INSERT INTO RepData (MainDataId,FullName, Date, Type, ClassName)

VALUES (@MainDataId, @FullName, @Date, @Type, @ClassName);";

SQLiteCommand command = new SQLiteCommand(insertQuery, connection);

command.Parameters.AddWithValue("@MainDataId", mainDataId); // Используем сохраненное значение id

command.Parameters.AddWithValue("@FullName", teacherName);

command.Parameters.AddWithValue("@Date", lessonData.Item1);

command.Parameters.AddWithValue("@Type", lessonData.Item2);

command.Parameters.AddWithValue("@ClassName", lessonData.Item3);

command.ExecuteNonQuery();

}

}

}

}

private void ExecuteNonQuery(SQLiteConnection connection, string query)

{

using (SQLiteCommand command = new SQLiteCommand(query, connection))

{

command.ExecuteNonQuery();

}

}

public int GetReplacementCount(SQLiteConnection connection, string fullName)

{

string query = "SELECT COUNT(\*) FROM RepData WHERE FullName = @FullName";

using (SQLiteCommand command = new SQLiteCommand(query, connection))

{

command.Parameters.AddWithValue("@FullName", fullName);

object result = command.ExecuteScalar();

return Convert.ToInt32(result);

}

}

}

## Eextrc

using System;

using System.Collections.Generic;

using System.IO;

using Microsoft.Office.Interop.Excel;

using ExcelApp = Microsoft.Office.Interop.Excel.Application;

namespace eWExtractor

{

public class ExcelExtractor

{

// Получение основных данных из файла Excel

public static Dictionary<string, string> GetExtractedMainData(string filePath)

{

Dictionary<string, string> extractedData = new Dictionary<string, string>();

// Создание экземпляра приложения Excel

ExcelApp excelApp = new ExcelApp();

// Открытие файла Excel

Workbook workbook = excelApp.Workbooks.Open(filePath);

// Получение первого листа рабочей книги

Microsoft.Office.Interop.Excel.Worksheet worksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Sheets[1];

// Чтение данных из необходимых ячеек и добавление их в словарь

extractedData["Подпись"] = GetCellValue(worksheet.Cells[1, 1]);

extractedData["Должность"] = GetCellValue(worksheet.Cells[2, 1]);

extractedData["ФИО"] = GetCellValue(worksheet.Cells[3, 1]);

extractedData["НачальнаяДата"] = GetCellValue(worksheet.Cells[4, 1]);

extractedData["КонечнаяДата"] = GetCellValue(worksheet.Cells[5, 1]);

extractedData["ЛистНетрудоспособности"] = GetCellValue(worksheet.Cells[6, 1]);

// Закрытие книги Excel и приложения Excel

workbook.Close(false);

excelApp.Quit();

return extractedData;

}

// Получение дополнительных данных из файла Excel

public static Dictionary<string, List<(string, string, string)>> GetExtractedAddData(string filePath)

{

Dictionary<string, List<(string, string, string)>> extractedData = new Dictionary<string, List<(string, string, string)>>();

// Создание экземпляра приложения Excel

ExcelApp excelApp = new ExcelApp();

// Открытие файла Excel

Workbook workbook = excelApp.Workbooks.Open(filePath);

// Получение второго листа рабочей книги

Worksheet worksheet = null;

if (workbook.Sheets.Count >= 2)

{

worksheet = (Worksheet)workbook.Sheets[2];

}

else

{

// Обработка ситуации, когда в рабочей книге нет второго листа

// Например, можно выбросить исключение или записать сообщение в лог

Console.WriteLine("В рабочей книге отсутствует второй лист.");

}

// Чтение данных из необходимых ячеек и добавление их в словарь

int rowCount = worksheet.UsedRange.Rows.Count;

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

{

string teacherName = GetCellValue(worksheet.Cells[i, 1]);

string date = GetCellValue(worksheet.Cells[i, 2]);

string lessonType = GetCellValue(worksheet.Cells[i, 3]);

string group = GetCellValue(worksheet.Cells[i, 4]);

if (!extractedData.ContainsKey(teacherName))

{

extractedData[teacherName] = new List<(string, string, string)>();

}

extractedData[teacherName].Add((date, lessonType, group));

}

// Закрытие книги Excel и приложения Excel

workbook.Close(false);

excelApp.Quit();

return extractedData;

}

// Вспомогательный метод для обработки значений ячеек и возврата строки

private static string GetCellValue(object cellValue)

{

return cellValue != null ? cellValue.ToString() : string.Empty;

}

// Сохранение данных в файл Excel

public static void SaveDataToExcel(string filePath, Dictionary<string, string> mainData, Dictionary<string, List<(string, string, string)>> repData)

{

// Создание нового экземпляра приложения Excel

Microsoft.Office.Interop.Excel.Application excelApp = new Microsoft.Office.Interop.Excel.Application();

// Открытие новой рабочей книги Excel

Microsoft.Office.Interop.Excel.Workbook workbook = excelApp.Workbooks.Add();

// Получение первого листа рабочей книги Excel

Microsoft.Office.Interop.Excel.Worksheet worksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Sheets[1];

// Запись основных данных в ячейки первого листа

int row = 1;

foreach (var kvp in mainData)

{

((Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row, 1]).Value = kvp.Value;

row++;

}

// Запись дополнительных данных в ячейки первого листа под основными данными

row += 2; // Пропуск двух строк для разделения

foreach (var kvp in repData)

{

string teacherName = kvp.Key;

foreach (var data in kvp.Value)

{

((Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row, 1]).Value = teacherName;

((Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row, 2]).Value = data.Item1;

((Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row, 3]).Value = data.Item2;

((Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row, 4]).Value = data.Item3;

row++;

}

}

// Изменение пути к файлу для изменения расширения на .xlsx

string newFilePath = Path.ChangeExtension(filePath, "xlsx");

// Сохранение рабочей книги Excel с измененным путем к файлу

workbook.SaveAs(newFilePath);

// Закрытие рабочей книги Excel и приложения Excel

workbook.Close();

excelApp.Quit();

}

}

}

## Wextrc

using Microsoft.Office.Interop.Word;

using System.Globalization;

using System.Text;

using System.Text.RegularExpressions;

using WordApp = Microsoft.Office.Interop.Word.Application;

namespace eWExtractor

{

public class WordExtractor

{

public static Dictionary<string, string> GetExtractedMainText(string filePath)

{

string extractedText = ExtractTextFromWordDocument(filePath);

var extractedGeneralData = ExtractDataFromMainText(extractedText);

return extractedGeneralData;

}

public static Dictionary<string, List<(string, string, string)>> GetExtractedAddText(string filePath)

{

string extractedText = ExtractTextFromWordDocument(filePath);

string cuttedText = GetCuttedText(extractedText);

// Выводим сообщение о завершении извлечения текста

var extractedReplacementData = ExtractDataFromAddText(cuttedText);

return extractedReplacementData;

}

public static void SaveDataToWord(string filePath, Dictionary<string, string> mainData, Dictionary<string, List<(string, string, string)>> repData)

{

// Выводим сообщение о начале процесса

try

{

// Сохраняем текст в файл

string templateFilePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Temple.docx");

string outputFilePath = Path.Combine(Path.GetDirectoryName(filePath), Path.GetFileNameWithoutExtension(filePath) + "\_Extracted.docx");

FillTemplate(templateFilePath, outputFilePath, mainData, repData);

// Выводим сообщение о завершении процесса и пути к сохраненному файлу

MessageBox.Show($"Процесс завершен. Извлеченный текст сохранен в файле:\n{outputFilePath}", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

// Выводим сообщение об ошибке, если что-то пошло не так

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

}

}

static string GetCuttedText(string extractedText)

{

int startIndex = extractedText.IndexOf("нагрузки:");

int endIndex = extractedText.IndexOf("УиЗИ");

return extractedText.Substring(startIndex, endIndex - startIndex).Trim();

}

static string ExtractTextFromWordDocument(string filePath)

{

// Create an instance of Word Application

Microsoft.Office.Interop.Word.Application wordApp = new Microsoft.Office.Interop.Word.Application();

// Define missing object to use as optional parameters

object missing = System.Reflection.Missing.Value;

// Open the document

Document doc = wordApp.Documents.Open(filePath, ref missing, ref missing, ref missing,

ref missing, ref missing, ref missing, ref missing,

ref missing, ref missing, ref missing, ref missing,

ref missing, ref missing, ref missing, ref missing);

// Read the text from the document

string text = "";

foreach (Paragraph paragraph in doc.Paragraphs)

{

text += paragraph.Range.Text + Environment.NewLine;

}

// Close the document and application

doc.Close();

wordApp.Quit();

return text;

}

static Dictionary<string, string> ExtractDataFromMainText(string extractedText)

{

Dictionary<string, string> data = new Dictionary<string, string>();

// Кому на подпись документ

if (extractedText.Contains("Гречишникову В.А."))

{

/\*data["MP"] = "Зам. директора ИТТСУ^pпо учебной работе,^pд.т.н., профессору^pГречишникову В.А.".TrimStart();\*/

data["Подпись"] = "Гречишникову В.А.";

}

else if (extractedText.Contains("Бестемьянову П.Ф."))

{

/\*data["На подпись"] = "Директору ИТТСУ^pБестемьянову П.Ф.".TrimStart();\*/

data["Подпись"] = "Бестемьянову П.Ф.";

}

// Кто заболел

string pattern = @"(?<Должность>(?:доцент[а]\*|профессор[а]\*|старшего преподавателя))\s+(?<Фамилия>\p{Lu}\p{L}+)(?:\s+(?<Инициалы>\p{Lu}\.\p{Lu}\.)?)?";

Match match = Regex.Match(extractedText, pattern, RegexOptions.IgnoreCase);

if (match.Success)

{

string position = match.Groups["Должность"].Value.Trim();

data["Должность"] = position;

string lastName = match.Groups["Фамилия"].Value.Trim();

string initials = match.Groups["Инициалы"].Value.Trim();

if (!string.IsNullOrEmpty(initials))

{

data["ФИО"] = $"{lastName} {initials}";

}

else

{

data["ФИО"] = lastName;

}

}

// Даты болезни

pattern = @"с\s+(?:период\s+)?(?<НачальнаяДата>\d{1,2}\s+\p{L}+\s+\d{4}|\d{2}\.\d{2}\.\d{4})\s+(?:по|–|-)\s+(?<КонечнаяДата>\d{1,2}\s+\p{L}+\s+\d{4}|\d{2}\.\d{2}\.\d{4})\s+г\.";

match = Regex.Match(extractedText, pattern, RegexOptions.IgnoreCase);

if (match.Success)

{

data["НачальнаяДата"] = match.Groups["НачальнаяДата"].Value.Trim();

data["КонечнаяДата"] = match.Groups["КонечнаяДата"].Value.Trim();

}

// Содержимое листка нетрудоспособности

pattern = @"\(([^)]\*)\)";

match = Regex.Match(extractedText, pattern);

if (match.Success)

{

string sickLeaveNote = match.Groups[1].Value.Trim();

data["ЛистНетрудоспособности"] = sickLeaveNote;

}

return data;

}

static Dictionary<string, List<(string Date, string LessonType, string Group)>> ExtractDataFromAddText(string extractedText)

{

var data = new Dictionary<string, List<(string Date, string LessonType, string Group)>>();

// Разделение текста по строкам

string[] lines = extractedText.Split(new[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

string currentFullName = null;

string dateOfReplacement = null; // Дата замены занятия

foreach (string line in lines)

{

// Если строка содержит "Доцент" или "Профессор", то это новый преподаватель

if (line.Contains("Доцент") || line.Contains("Профессор") || line.Contains("Старший преподаватель"))

{

currentFullName = line.Trim();

data[currentFullName] = new List<(string Date, string LessonType, string Group)>();

}

// Если строка содержит дату и информацию о замене занятия

else if (line.Contains(","))

{

string[] parts = line.Split(',');

if (parts.Length == 2)

{

string[] details = parts[1].Split('=');

if (details.Length == 2)

{

string[] lessonDetails = details[0].Trim().Split();

if (lessonDetails.Length >= 4)

{

dateOfReplacement = parts[0].Trim(); // Записываем дату замены занятия

string lessonType = "неизвестно"; // По умолчанию, если тип занятия не определен

// Проверяем тип занятия

foreach (string detail in lessonDetails)

{

if (detail.Contains("лекция"))

{

lessonType = "лекция";

break;

}

else if (detail.Contains("лабораторная") || detail.Contains("лабораторные"))

{

lessonType = "лабораторная";

break;

}

else if (detail.Contains("практическое"))

{

lessonType = "практическое занятие";

break;

}

else if (detail.Contains("экзамен"))

{

lessonType = "экзамен";

break;

}

else if (detail.Contains("консультация"))

{

lessonType = "консультация";

break;

}

}

string group = "";

// Извлекаем группу из строки lessonDetails

for (int i = 3; i < lessonDetails.Length; i++)

{

if (lessonDetails[i].Contains("-")) // Проверяем, содержит ли элемент "-"

{

group = lessonDetails[i]; // Если содержит, присваиваем это значение переменной group

break; // Заканчиваем цикл после нахождения группы

}

}

if (!string.IsNullOrEmpty(currentFullName) && data.ContainsKey(currentFullName))

{

// Добавляем данные в список текущего преподавателя

data[currentFullName].Add((dateOfReplacement, lessonType, group));

}

}

}

}

}

// Если строка содержит "Итого:"

else if (line.Contains("Итого:"))

{

break; // Останавливаем парсинг после обработки всех данных

}

}

return data;

}

static void FillTemplate(string templateFilePath, string outputFilePath, Dictionary<string, string> extractedGeneralData, Dictionary<string, List<(string Date, string LessonType, string Group)>> extractedReplacementData)

{

WordApp wordApp = new WordApp();

Document doc = wordApp.Documents.Open(templateFilePath);

// Заменяем плейсхолдер для общей информации

foreach (var kvp in extractedGeneralData)

{

string key = kvp.Key;

string value = kvp.Value;

// Проверяем, является ли ключ "Подпись"

if (key == "Подпись")

{

// Проверяем значение и заменяем в соответствии с именем

if (value == "Гречишникову В.А.")

{

value = "Зам. директора ИТТСУ^pпо учебной работе,^pд.т.н., профессору^pГречишникову В.А.".TrimStart();

}

else if (value == "Бестембчянову П.Ф.")

{

value = "Директору ИТТСУ^pБестемьянову П.Ф.".TrimStart();

}

}

// Заменяем значение в документе

FindAndReplace(wordApp, doc, $"[{key}]", value);

}

// Находим плейсхолдер и устанавливаем выравнивание по правому краю

Microsoft.Office.Interop.Word.Range range = doc.Content;

range.Find.ClearFormatting();

range.Find.Text = "[repData]";

/\*while (range.Find.Execute())

{

string currentText = range.Text.Trim();

range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphLeft;

}\*/

StringBuilder allReplacementData = new StringBuilder();

foreach (var kvp in extractedReplacementData)

{

string teacherName = kvp.Key;

// Добавляем имя преподавателя

allReplacementData.AppendLine(teacherName.PadRight(50));

int workHours = 0;

string workMonth = "";

foreach (var lessonData in kvp.Value)

{

// Выводим данные о замененных занятиях выровненными по правому краю с отступом

allReplacementData.AppendLine($"\t\t{lessonData.Date}, {lessonData.LessonType} в группе {lessonData.Group} = 2 часа");

workHours += 2;

workMonth = GetMonthName(lessonData.Date);

}

allReplacementData.AppendLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_".PadLeft(105));

allReplacementData.AppendLine($"{workMonth} - {workHours} ч.".PadLeft(120));

}

// Заменяем плейсхолдер [ExtractDataFromText2] на собранные данные о замененных занятиях

InsertTextInDocument(wordApp, doc, "[repData]", allReplacementData.ToString());

doc.SaveAs(outputFilePath);

doc.Close();

wordApp.Quit();

}

static string GetMonthName(string date)

{

DateTime dateTime = DateTime.ParseExact(date, "dd.MM.yyyy", CultureInfo.InvariantCulture);

string monthName = dateTime.ToString("MMMM", CultureInfo.GetCultureInfo("ru-RU"));

return char.ToUpper(monthName[0]) + monthName.Substring(1);

}

static void InsertTextInDocument(WordApp wordApp, Document doc, string placeholder, string text)

{

foreach (Microsoft.Office.Interop.Word.Range range in doc.StoryRanges)

{

range.Find.ClearFormatting();

range.Find.Text = placeholder;

while (range.Find.Execute())

{

range.Text = text;

}

}

}

static void FindAndReplace(WordApp wordApp, Document doc, string findText, string replaceText)

{

foreach (Microsoft.Office.Interop.Word.Range range in doc.StoryRanges)

{

range.Find.ClearFormatting();

range.Find.Execute(FindText: findText, ReplaceWith: replaceText, Replace: WdReplace.wdReplaceAll);

}

}

}

}

# Вывод

В ходе написания данной работы были получены навыки работы с экранными формами Windows посредством языка программирования C#. Освоены основные библиотеки, взаимодействие классов, элементы объектно‑ориентированного программирования. Результатом работы стал также приобретённый навык построения схем Петри процессов.

Основным результатом работы стала программа, способная обрабатывать печатные формы с расписанием, конвертировать их в другие форматы, производить предварительный просмотр. Программа способна строить графики для анализа данных, собранных в результате её эксплуатации.