МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

(наименование института, филиала)

Кафедра КБ-2 «Информационно-аналитические системы кибербезопасности»

(наименование кафедры)

КУРСОВАЯ РАБОТА

(указать вид работы)

по дисциплине «РАЗРАБОТКА ПРОГРАММНЫХ СРЕДСТВ ПОИСКА И АНАЛИЗА КИБЕРУГРОЗ»

(указать дисциплину в соответствии с учебным планом)

Тема курсовой работы: Разработка программного средства аудита нормативных документов по защите информации

Исполнитель курсовой работы:

Студент 4 **курса, учебной группы** БББО-01-21
Мысливец Леонид Владимирович

(фамилия, имя и отчество)

Руководитель курсовой работы:

Новиков Е. И.	доцент кафедры, к.т.н., доцент	
(Фамилия И.О.)	*(Должность, звание, ученая степень)*	*(Подпись руководителя)*

Рецензент (при наличии):

(Фамилия И.О.)	*(Должность, звание, ученая степень)*	*(Подпись рецензента)*

Работа представлена к защите:

«___» 2024 г. _____ (Новиков Е. И., доцент)
Подпись *Расшифровка, должность*

Допущен к защите:

«___ » 2024 г. _____ (Новиков Е. И., доцент)
Подпись *Расшифровка, должность*

Москва – 2024

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

(наименование института, филиала)

Кафедра КБ-2 «Информационно-аналитические системы кибербезопасности»

(наименование кафедры)

УТВЕРЖДАЮ
Заведующий кафедрой КБ-2

_____/Трубиенко О.В./
« » сентября 2024 г.

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

(указать вид работы)

по дисциплине «РАЗРАБОТКА ПРОГРАММНЫХ СРЕДСТВ ПОИСКА И АНАЛИЗА КИБЕРУГРОЗ»

(указать дисциплину в соответствии с учебным планом)

Студенту 4 **курса, учебной группы** ББББО-01-21
Мысливец Леонид Владимирович

(фамилия, имя и отчество)

1. Тема: Разработка программного средства аудита нормативных документов по защите информации
2. Исходные данные: Электронные нормативные документы по защите информации
3. Перечень вопросов, подлежащих разработке, и обязательного графического материала: Анализ программных средств аудита текстовых документов

Срок представления к защите курсовой работы: «___» декабря 2024 г.

Задание на курсовую работу выдал:
«___» сентября 2024 г. _____ (Новиков Е. И.)
Подпись *(Фамилия И.О.)*

Задание на курсовуюработу получил:
«___» сентября 2024 г. _____ (Мысливец Л.В.)
Подпись *(Фамилия И.О.)*

Оглавление

Введение

Актуальность задачи курсовой работы заключается в необходимости автоматизации процесса извлечения данных из различных нормативных документов по защите информации, в форматах DOCX, PDF и HTML, для их дальнейшего анализа. Задача программного анализа документов актуальна в условиях большого объема данных, требующих систематической обработки для извлечения ключевой информации, которая затем может быть использована для аналитических целей. Решение этой задачи необходимо для создания эффективных инструментов, которые помогут автоматизировать рутинные процессы обработки данных, сэкономив время и усилия специалистов.

В рамках курсовой работы будет реализован алгоритм, позволяющий извлекать данные из различных форматов документов, включая текстовую информацию, даты и другие данные, с целью их дальнейшего использования в аналитических системах.

Основная часть

Раздел 1. Анализ предметной области

Задача, поставленная в курсовой работе, заключается в разработке алгоритмов для извлечения информации из нормативных документов, представленных в форматах DOCX, PDF и HTML. Существующие методы решения задачи включают использование различных библиотек для анализа документов, таких как:

– «python-docx» для работы с DOCX файлами;

– «PyMuPDF (fitz)» для извлечения текста из PDF;

– «BeautifulSoup» для анализа HTML.

Использование этих инструментов позволяет автоматически извлекать необходимые данные, такие как даты и текстовые разделы, без необходимости вручную обрабатывать каждый документ.

В рамках работы принято решение использовать методы анализа с применением регулярных выражений и библиотек, способных работать с различными форматами документов. Это решение обосновано высоким качеством извлечения данных, простотой интеграции с другими системами и возможностью масштабирования.

Исходные данные, которые используются для программы, включают:

– Документы в формате DOCX с таблицами, содержащими ключевую информацию;

– PDF файлы с текстовыми разделами, из которых необходимо извлечь информацию о платежах и статьях;

– HTML файлы, содержащие данные о датах подписания документов.

Задача состоит в извлечении даты и других значимых данных из этих файлов для последующего их использования в расчетах и анализе.

Математическая постановка задачи заключается в использовании регулярных выражений для извлечения строк, соответствующих определенному шаблону (например, даты в формате “дд.мм.гггг”). Алгоритм анализа

реализуется через последовательность операций поиска, замены и анализа текста.

Раздел 2. Описание программы анализа документов

В этой работе для анализа документов выбран ряд методов, обеспечивающих высокую точность извлечения данных:

– Для работы с DOCX использована библиотека python-docx, которая позволяет извлекать текст из таблиц и других элементов документа;

– Для извлечения текста из PDF использована библиотека PyMuPDF (fitz), которая позволяет читать текст с каждой страницы документа;

– Для обработки HTML файлов использована библиотека BeautifulSoup, которая предоставляет удобные инструменты для навигации по элементам HTML-документа.

Алгоритмы реализуются с использованием регулярных выражений для поиска дат и текстовых разделов в документах. Основной идеей алгоритма является поиск определенных шаблонов в тексте и извлечение информации с помощью регулярных выражений и манипуляций с объектами документов.

Взаимодействие Пользователя с Программой

Пользователь начинает работу с программой, загружая документы в форматах DOCX, PDF и HTML, которые содержат нормативную информацию. Программа обеспечивает удобный и интуитивно понятный интерфейс (рисунок 1), позволяя пользователю легко выбрать и загрузить файлы для анализа.

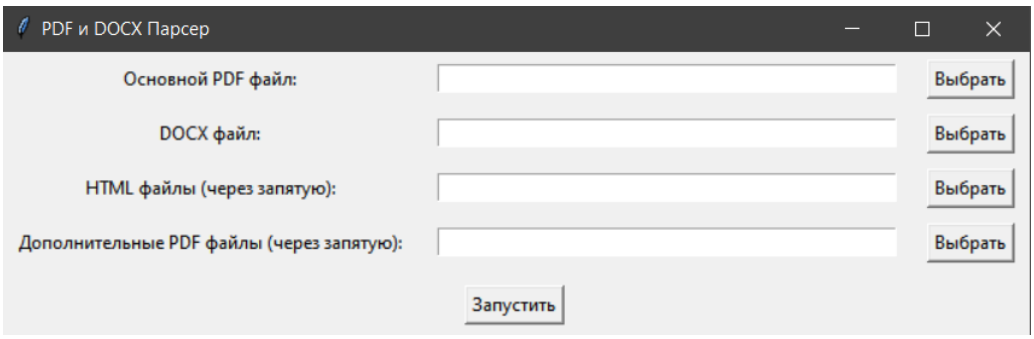

Рисунок 1 - Окно программы

Загрузка Документов: Пользователь выбирает файлы с документацией для анализа. Программа поддерживает различные форматы, включая DOCX, PDF и HTML. Окно загрузки файлов показано на рисунке 2

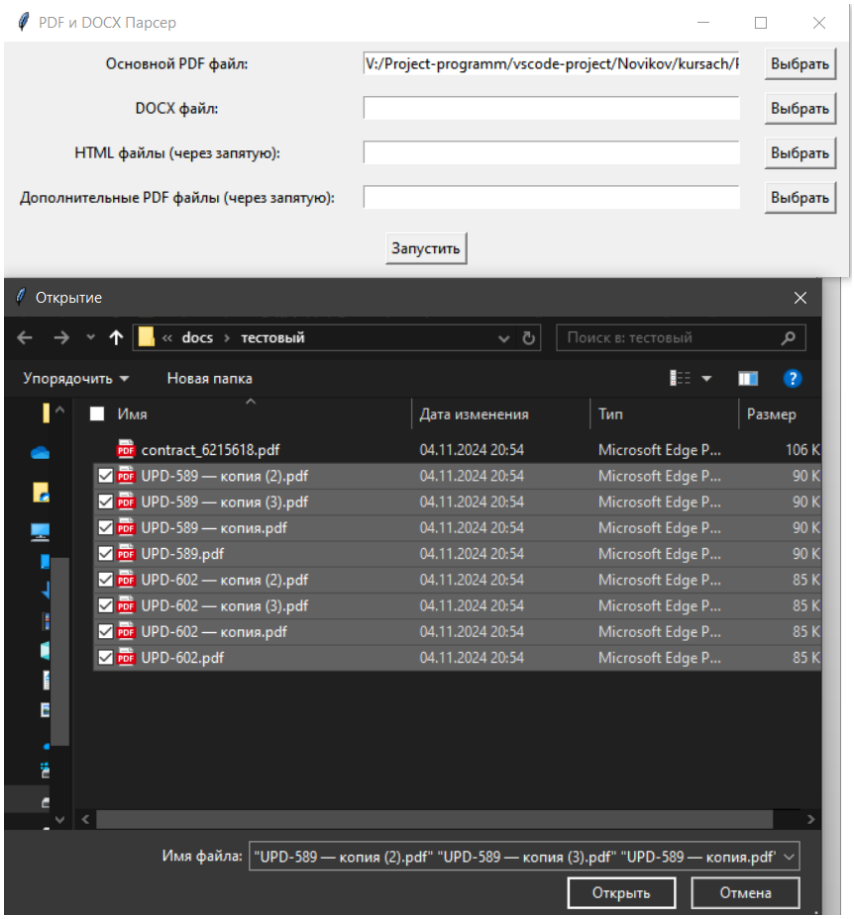

Рисунок 2 - Загрузка файлов

Запуск Анализа: после загрузки документов пользователь запускает процесс анализа. Программа автоматически извлекает ключевую информацию из каждого файла.

Основные Этапы Анализа

Программа выполняет следующие основные этапы анализа:

1. Извлечение Текста: все текстовые данные из загруженных файлов извлекаются с помощью вышеописанных методов и сохраняются для дальнейшей обработки.

2. Анализ Текста: извлеченный текст анализируется с использованием регулярных выражений для поиска ключевых данных, таких как даты, статьи и платежи. Пример использования регулярных выражений:

```
import re
def extract_dates(text):
    date_pattern = r'\b\d{2}\.\d{2}\.\d{4}\b'
    dates = re.findall(date_pattern, text)
    return dates
```

3. Сохранение и Отображение Результатов**:** Результаты анализа отображаются пользователю в удобном виде (рисунок 3). Пользователь может просмотреть, сохранить и использовать извлеченную информацию для дальнейших расчетов и анализа.

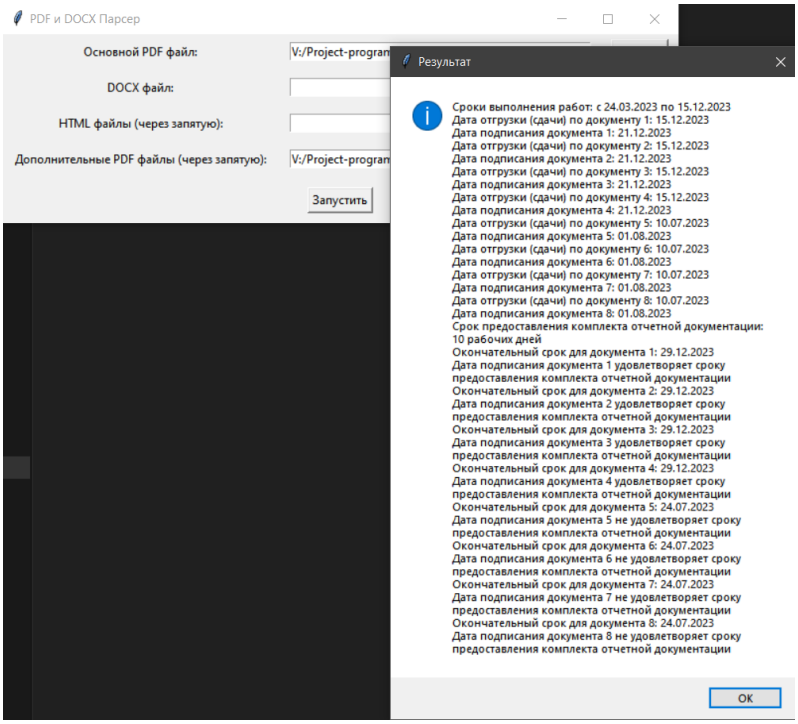

Рисунок 3 - Вывод программы (прошедшие и не прошедшие проверки документы)

Раздел 3. Реализация алгоритмов

Реализация алгоритмов заключается в следующем:

Подготовка исходных данных

1) Для DOCX файлов анализируются таблицы на наличие ключевых слов и извлекаются соответствующие данные.

Обработка DOCX Файлов:

Библиотека «python-docx»: позволяет извлекать текст и таблицы из DOCX файлов.

Метод:

```
import docx
def extract_text_from_docx(file_path):
    doc = docx.Document(file_path)
    full_text = []
    for para in doc.paragraphs:
        full_text.append(para.text)
    return '\n'.join(full_text)
```

2) Для PDF файлов извлекается весь текст с каждой страницы, затем ищутся необходимые разделы.

Обработка PDF Файлов:

Библиотека «PyMuPDF (fitz)»: используется для извлечения текста из PDF файлов.

Метод:

```
import fitz  # PyMuPDF
def extract_text_from_pdf(file_path):
    pdf_document = fitz.open(file_path)
    text = ""
    for page_num in range(len(pdf_document)):
        page = pdf_document.load_page(page_num)
        text += page.get_text()
    return text
```

3) Для HTML файлов анализируются таблицы на предмет наличия дат и другой информации.

Обработка HTML Файлов:

Библиотека BeautifulSoup: применяется для анализа HTML и извлечения данных.

Метод:

```
from bs4 import BeautifulSoup
def extract_text_from_html(file_path):
    with open(file_path, "r", encoding="utf-8") as file:
        soup = BeautifulSoup(file, "html.parser")
        return soup.get_text()
```

Реализация алгоритмов:

– Алгоритмы для извлечения данных из DOCX, PDF и HTML файлов реализуются с использованием библиотек, описанных выше;

– В случае с DOCX осуществляется поиск в таблицах и извлечение текста из определенных ячеек;

– В PDF используется поиск по текстовым разделам с помощью регулярных выражений для выявления ключевой информации;

– В HTML выполняется парсинг таблиц для извлечения дат подписания документов по меткам HTML.

Алгоритмы анализа тестируются на реальных документах для проверки точности извлечения информации и корректности работы с различными форматами файлов.

Раздел 4. Оценивание качества алгоритмов

Оценка качества алгоритмов

Для анализа документов качество алгоритмов чаще всего оценивается по следующим критериям:

– Корректность извлечения данных — насколько точно алгоритм находит нужные данные (например, даты, текстовые фрагменты). Здесь важно измерить, насколько результаты анализа соответствуют реальному содержимому документов;

– Производительность — время выполнения алгоритма, особенно если необходимо обрабатывать большие объемы данных. Это важно для применения алгоритмов в реальных условиях;

– Устойчивость к ошибкам — способность программы правильно работать с некорректными или неполными данными, например, с пропущенными элементами, пустыми ячейками или необычным форматированием текста.

Методы тестирования

– Сравнение с эталонными результата. Для оценки точности алгоритма сравнивались результаты программы с заранее подготовленными вручную данными;

– Тестирование на различных форматах. Алгоритм должен быть протестирован на различных типах документов (например, .docx, .pdf, .html), чтобы удостовериться в его универсальности.

Раздел 5. Применение алгоритмов

Алгоритмы программы были применены для анализа нескольких типов нормативных документов, таких как отчеты, контракты и прочие нормативные документы, чтобы извлечь важную информацию, включая даты подписания, данные о платежах и другие ключевые моменты. Результаты использовались для дальнейших расчетов и анализа сроков и периодичности платежей.

Заключение

В ходе выполнения курсовой работы были разработаны и реализованы алгоритмы для анализа документов в форматах DOCX, PDF и HTML. Алгоритмы успешно извлекают ключевую информацию из документов, включая даты, текстовые разделы и другие значимые данные. Применение этих алгоритмов в реальных задачах позволяет значительно упростить процесс обработки документов и повысить его эффективность.

Список используемых источников

1. Официальная документация Python. – URL: https://docs.python.org/ (Дата обращения 11.12.2024).

2. Официальная документация библиотеки python-docx. – URL: https://python-docx.readthedocs.io/ (Дата обращения 11.12.2024).

3. Официальная документация библиотеки PyMuPDF. – URL: https://pymupdf.readthedocs.io/ (Дата обращения 11.12.2024).

4. Официальная документация библиотеки BeautifulSoup. – URL: https://www.crummy.com/software/BeautifulSoup/ (Дата обращения 12.12.2024).

Приложение А

Main.py:

```
import parse_pdf
import parse_docx
from datetime import datetime
import sys

def main():
    if len(sys.argv) < 4:
        print("Usage: main.py <main_pdf_path> <docx_path> <html_paths>
<additional_pdf_paths>")
        return

    # Ввод пути к основному PDF файлу
    main_pdf_path = sys.argv[1]

    # Ввод пути к DOCX файлу
    docx_path = sys.argv[2]

    # Ввод путей к HTML файлам
    html_paths = sys.argv[3].split(',')

    # Ввод путей к дополнительным PDF файлам для поиска даты отгрузки и даты подписания
    additional_pdf_paths = sys.argv[4].split(',')

    # Парсинг основного и дополнительных PDF файлов
    main_dates, additional_dates, article_4_number =
parse_pdf.parse_multiple_pdfs(main_pdf_path, additional_pdf_paths)
    if main_dates and len(additional_dates) == len(additional_pdf_paths) * 2:
        print(f"Сроки выполнения работ: с {main_dates[0]} по {main_dates[1]}")
        for i in range(len(additional_pdf_paths)):
            print(f"Дата отгрузки (сдачи) по документу {i + 1}: {additional_dates[i *
2]}")
            print(f"Дата подписания документа {i + 1}: {additional_dates[i * 2 + 1]}")
        if article_4_number:
            print(f"Срок предоставления комплекта отчетной документации:
{article_4_number} рабочих дней")

            # Расчет срока предоставления комплекта отчетной документации
            shipping_dates = [additional_dates[i * 2] for i in
range(len(additional_pdf_paths))]
            for index, shipping_date in enumerate(shipping_dates):
                shipping_date_dt = datetime.strptime(shipping_date, '%d.%m.%Y')
                deadline_date_dt = parse_pdf.add_business_days(shipping_date_dt,
article_4_number)
                deadline_date_str = deadline_date_dt.strftime('%d.%m.%Y')
                print(f"Окончательный срок для документа {index + 1}:
{deadline_date_str}")

                signed_date = additional_dates[index * 2 + 1]
                signed_date_dt = datetime.strptime(signed_date, '%d.%m.%Y')
```

```
                if signed_date_dt <= deadline_date_dt:
                    print(f"Дата подписания документа {index + 1} удовлетворяет сроку
предоставления комплекта отчетной документации")
                else:
                    print(f"Дата подписания документа {index + 1} не удовлетворяет сроку
предоставления комплекта отчетной документации")
    else:
        print("Даты не найдены в PDF. Открытие .docx для поиска...")

        # Парсинг DOCX и PDF файлов
        dates_from_docx, payment_period, article_4_number =
parse_docx.parse_docx_and_pdf(docx_path, main_pdf_path)
        if dates_from_docx:
            if len(dates_from_docx) == 2:
                print(f"Сроки выполнения работ: с {dates_from_docx[0]} по
{dates_from_docx[1]}")
                if payment_period:
                    print(f"Период оплаты: {', '.join(payment_period)}")
                if article_4_number:
                    print(f"Срок предоставления комплекта отчетной документации:
{article_4_number} рабочих дней")

                    # Парсинг HTML файлов и расчет срока предоставления комплекта
отчетной документации
                    html_dates = parse_docx.parse_multiple_html(html_paths)
                    if html_dates:
                        for i, html_date in enumerate(html_dates):
                            print(f"Дата подписания документа {i + 1}: {html_date}")
                        deadline_dates =
parse_docx.calculate_deadline_dates(article_4_number, html_dates)
                        print(f"Расчетные даты дедлайнов: {deadline_dates}")

                        for i, html_date in enumerate(html_dates):
                            html_date_dt = datetime.strptime(html_date, '%d.%м.%Y')
                            deadline_date_dt = datetime.strptime(deadline_dates[i],
'%d.%м.%Y')
                            if html_date_dt <= deadline_date_dt:
                                print(f"Дата подписания {html_date} удовлетворяет
дедлайну {deadline_dates[i]}")
                            else:
                                print(f"Дата подписания {html_date} НЕ удовлетворяет
дедлайну {deadline_dates[i]}")
            elif len(dates_from_docx) == 1:
                print(f"Сроки выполнения работ: {dates_from_docx[0]}")
                if article_4_number:
                    print(f"Срок предоставления комплекта отчетной документации:
{article_4_number} рабочих дней")
            else:
                print("Не удалось найти даты в таблице DOCX.")
        else:
            print("Не удалось найти даты в DOCX документе.")

if __name__ == "__main__":
```

```
    # sys.argv = [
    #     "main.py",
    #     "docs/example-1/contract_6215618.pdf",
    #     "docs/example-2/T3 178_2023.docx",
    #     "docs/example-2/UPD-13.html,docs/example-2/UPD-23.html",
    #     "docs/example-1/UPD-589.pdf,docs/example-1/UPD-602.pdf"
    # ]
    main()
```

start.py:

```
import tkinter as tk
from tkinter import filedialog, messagebox
import subprocess

def run_main(main_pdf_path, docx_path, html_paths, additional_pdf_paths):
    args = [
        "python", "main.py",
        main_pdf_path,
        docx_path,
        ",".join(html_paths),
        ",".join(additional_pdf_paths)
    ]
    result = subprocess.run(args, capture_output=True, text=True)

    if result.returncode == 0:
        messagebox.showinfo("Результат", result.stdout)
    else:
        messagebox.showerror("Ошибка", f"Произошла ошибка:\n{result.stderr}")

def select_file(entry):
    file_path = filedialog.askopenfilename()
    if file_path:
        entry.delete(0, tk.END)
        entry.insert(0, file_path)

def select_files(entry):
    file_paths = filedialog.askopenfilenames()
    if file_paths:
        entry.delete(0, tk.END)
        entry.insert(0, ",".join(file_paths))

def create_gui():
    root = tk.Tk()
    root.title("PDF и DOCX Парсер")

    tk.Label(root, text="Основной PDF файл:").grid(row=0, column=0, padx=10, pady=5)
    main_pdf_entry = tk.Entry(root, width=50)
    main_pdf_entry.grid(row=0, column=1, padx=10, pady=5)
    tk.Button(root, text="Выбрать", command=lambda:
select_file(main_pdf_entry)).grid(row=0, column=2, padx=10, pady=5)

    tk.Label(root, text="DOCX файл:").grid(row=1, column=0, padx=10, pady=5)
```

```
    docx_entry = tk.Entry(root, width=50)
    docx_entry.grid(row=1, column=1, padx=10, pady=5)
    tk.Button(root, text="Выбрать", command=lambda: select_file(docx_entry)).grid(row=1,
column=2, padx=10, pady=5)

    tk.Label(root, text="HTML файлы (через запятую):").grid(row=2, column=0, padx=10,
pady=5)
    html_entry = tk.Entry(root, width=50)
    html_entry.grid(row=2, column=1, padx=10, pady=5)
    tk.Button(root, text="Выбрать", command=lambda:
select_files(html_entry)).grid(row=2, column=2, padx=10, pady=5)

    tk.Label(root, text="Дополнительные PDF файлы (через запятую):").grid(row=3,
column=0, padx=10, pady=5)
    additional_pdf_entry = tk.Entry(root, width=50)
    additional_pdf_entry.grid(row=3, column=1, padx=10, pady=5)
    tk.Button(root, text="Выбрать", command=lambda:
select_files(additional_pdf_entry)).grid(row=3, column=2, padx=10, pady=5)

    tk.Button(root, text="Запустить", command=lambda: run_main(
        main_pdf_entry.get(),
        docx_entry.get(),
        html_entry.get().split(","),
        additional_pdf_entry.get().split(",")
    )).grid(row=4, column=0, columnspan=3, pady=10)

    root.mainloop()

if __name__ == "__main__":
    create_gui()
```

parse_docx.py:

```
import re
import docx
import fitz  # PyMuPDF
from bs4 import BeautifulSoup
from datetime import datetime, timedelta
import pandas as pd

def extract_data_from_table(docx_path):
    doc = docx.Document(docx_path)
    for table in doc.tables:
        first_cell_text = table.cell(0, 0).text
        if "УСЛУГИ ПО ПРЕДОСТАВЛЕНИЮ ДОСТУПА К ПРОГРАММНОМУ ПРОДУКТУ" in
first_cell_text:
            right_cell_text = table.cell(3, 3).text
            dates = find_dates_in_section(right_cell_text)
            if dates:
                return dates
    return None

def find_dates_in_section(section):
    section = section.replace('\n', ' ')
```

```
    date_pattern = re.compile(r'\b\d{2}\.\d{2}\.\d{4}\b')
    dates = date_pattern.findall(section)
    return dates

def extract_text_from_pdf(pdf_path):
    document = fitz.open(pdf_path)
    text = ""
    for page_num in range(len(document)):
        page = document.load_page(page_num)
        text += page.get_text()
    return text

def find_section(text, start_pattern, end_pattern):
    pattern = re.compile(rf'({start_pattern}.*?){end_pattern}', re.DOTALL)
    match = pattern.search(text)
    if match:
        return match.group(0)
    return None

def find_payment_period(text):
    sections_to_check = ['2\\.6\\.2', '2\\.7\\.2']
    keywords = ['по факту', 'ежемесячно', 'ежеквартально']
    found_keywords = set()
    for section in sections_to_check:
        section_text = find_section(text, section, '\\n\\s*\\n')
        if section_text:
            for keyword in keywords:
                if keyword in section_text:
                    found_keywords.add(keyword)
    return found_keywords

def find_article_4_number(text):
    section = find_section(text, '4\\.1', '\\n\\s*\\н')
    if section:
        section = section.replace('4.1', '', 1)  # Убираем '4.1' один раз из начала
        match = re.search(r'\b(\d+)\b', section)
        if match:
            return int(match.group(1))  # Преобразуем в число
    return None

def extract_date_from_html(html_path):
    with open(html_path, 'r', encoding='utf-8') as file:
        soup = BeautifulSoup(file, 'html.parser')
        tables = soup.find_all('table')
        for table in tables:
            cell = table.find('td', string="Дата и время подписания ")
            if cell:
                rows = table.find_all('tr')
                if len(rows) >= 3:
                    columns = rows[2].find_all('td')
                    if len(columns) >= 4:
                        date_cell = columns[3]
                        # Оставляем только первые 10 символов
```

```
                        return date_cell.text[:10]
    return None

def parse_multiple_html(html_paths):
    html_dates = []
    for path in html_paths:
        date = extract_date_from_html(path)
        if date:
            html_dates.append(date)
    return html_dates

def get_first_working_day(year, month):
    # Получаем первый день месяца
    first_day = datetime(year, month, 1)
    # Проверяем, является ли первый день месяца рабочим днем
    while first_day.weekday() > 4:  # 0 - понедельник, 4 - пятница
        first_day += timedelta(days=1)
    return first_day

def add_business_days(start_date, business_days):
    current_date = start_date
    days_added = 0
    while days_added < business_days-1:
        current_date += timedelta(days=1)
        if current_date.weekday() < 5:  # 0 - понедельник, 4 - пятница
            days_added += 1
    return current_date

def calculate_deadline_dates(article_4_number, html_dates):
    deadlines = []
    for date_str in html_dates:
        year = int(date_str[-4:])
        month = int(date_str[3:5])
        first_working_day = get_first_working_day(year, month)
        deadline_date = add_business_days(first_working_day, article_4_number)
        deadlines.append(deadline_date.strftime('%d.%m.%Y'))
    return deadlines

def parse_docx_and_pdf(docx_path, pdf_path):
    docx_dates = extract_data_from_table(docx_path)
    if docx_dates:
        pdf_text = extract_text_from_pdf(pdf_path)
        payment_period = find_payment_period(pdf_text)
        article_4_number = find_article_4_number(pdf_text)
        return docx_dates, payment_period, article_4_number
    return None, None, None
```

parse_pdf.py:

```
import fitz  # PyMuPDF
import re
from datetime import datetime, timedelta
```

```
def extract_text_from_pdf(pdf_path):
    document = fitz.open(pdf_path)
    text = ""
    for page_num in range(len(document)):
        page = document.load_page(page_num)
        text += page.get_text()
    return text

def find_section(text, start_pattern, end_pattern):
    pattern = re.compile(rf'({start_pattern}.*?){end_pattern}', re.DOTALL)
    match = pattern.search(text)
    if match:
        return match.group(0)
    return None

def find_dates_in_section(section):
    date_pattern = re.compile(r'\b\d{2}\.\d{2}\.\d{4}\b')
    dates = date_pattern.findall(section)
    if len(dates) >= 2:
        return dates[:2]
    return None

def parse_pdf(pdf_path):
    start_pattern = '3\\.1'
    end_pattern = '\\n\\s*\\n'
    extracted_text = extract_text_from_pdf(pdf_path)
    section = find_section(extracted_text, start_pattern, end_pattern)
    if section:
        dates = find_dates_in_section(section)
        if dates:
            return dates
    return None

def find_shipping_date(text):
    pattern = re.compile(r'Дата отгрузки \(сдачи\)\s*\n\s*(\d{2}\s+[А-Яа-
я]+\s+\d{4}\s+г\.)')
    match = pattern.search(text)
    if match:
        date_str = match.group(1)
        return convert_date_to_dd_mm_yyyy(date_str)
    return None

def find_signed_date(text):
    pattern = re.compile(r'ДОКУМЕНТ ПОДПИСАН\nЭЛЕКТРОННОЙ ПОДПИСЬЮ\nДОКУМЕНТ
ПОДПИСАН\nЭЛЕКТРОННОЙ ПОДПИСЬЮ\n\s*(\d{2}\.\d{2}\.\d{4})')
    match = pattern.search(text)
    if match:
        return match.group(1)
    return None

def convert_date_to_dd_mm_yyyy(date_str):
    months = {
        "января": "01", "февраля": "02", "марта": "03", "апреля": "04",
```

```
        "мая": "05", "июня": "06", "июля": "07", "августа": "08",
        "сентября": "09", "октября": "10", "ноября": "11", "декабря": "12"
    }
    parts = date_str.split()
    if len(parts) != 4:
        raise ValueError(f"Неправильный формат даты: {date_str}")
    day, month, year, _ = parts
    return f"{day.zfill(2)}.{months[month]}.{year[:4]}"

def find_article_4_number(text):
    section = find_section(text, '4\\.1', '\\n\\s*\\n')
    if section:
        section = section.replace('4.1', '', 1)  # Убираем '4.1' один раз из начала
        match = re.search(r'\b(\d+)\b', section)
        if match:
            return int(match.group(1))  # Преобразуем в число
    return None

def parse_additional_pdfs(additional_pdf_paths):
    all_dates = []
    for path in additional_pdf_paths:
        text = extract_text_from_pdf(path)

        shipping_date = find_shipping_date(text)
        if shipping_date:
            all_dates.append(shipping_date)

        signed_date = find_signed_date(text)
        if signed_date:
            all_dates.append(signed_date)

    return all_dates

def add_business_days(start_date, num_days):
    current_date = start_date
    while num_days > 0:
        current_date += timedelta(days=1)
        if current_date.weekday() < 5:  # Понедельник-пятница
            num_days -= 1
    return current_date

def parse_multiple_pdfs(main_pdf_path, additional_pdf_paths):
    main_dates = parse_pdf(main_pdf_path)
    additional_dates = parse_additional_pdfs(additional_pdf_paths)
    article_4_number = find_article_4_number(extract_text_from_pdf(main_pdf_path))
    return main_dates, additional_dates, article_4_number
```