

Разработка и реализация интерактивного приложения для управления расписанием на основе Python с возможностью экспорта в различные форматы

Языков Кирилл Александрович

*Приамурский государственный университет имени Шолом-Алейхема
студент*

Аннотация

В данной работе представлена разработка и реализация интерактивного приложения для управления расписанием на основе языка программирования Python. Приложение предоставляет пользователю удобный графический интерфейс для добавления, удаления и редактирования событий в расписании. Ключевой особенностью приложения является возможность экспорта сформированного расписания в несколько распространенных форматов: JSON для хранения данных, iCalendar (ICS) для интеграции с популярными календарными приложениями, и PDF для удобного просмотра и печати. Разработанное приложение использует библиотеки Tkinter для создания графического интерфейса, reportlab для генерации PDF-документов, а также стандартные библиотеки Python для обработки данных и работы с файлами. В работе подробно описываются этапы разработки, архитектура приложения, а также оценивается его эффективность и удобство использования.

Ключевые слова: Управление расписанием, Python, Tkinter, ReportLab, iCalendar (ICS), PDF, графический интерфейс, интерактивное приложение, экспорт данных, разработка программного обеспечения.

Development and implementation of an interactive schedule management application based on Python with the ability to export to various formats

Yazykov Kirill Alexandrovich

*Sholom-Aleichem Priamursky State University
Student*

Abstract

This paper presents the development and implementation of an interactive schedule management application based on the Python programming language. The application provides the user with a user-friendly graphical interface for adding, deleting and editing events in the schedule. The key feature of the application is the ability to export the generated schedule in several common formats: JSON for data storage, iCalendar (ICS) for integration with popular calendar applications, and PDF for easy viewing and printing. The developed application uses Tkinter libraries to create a graphical interface, reportlab to generate PDF documents, as well as standard Python libraries for data processing and working with files. The paper

describes in detail the stages of development, the architecture of the application, as well as evaluates its effectiveness and usability.

Keywords: Schedule management, Python, Tkinter, ReportLab, iCalendar (ICS), PDF, graphical interface, interactive application, data export, software development.

1 Введение

1.1 Актуальность

Эффективное управление временем и планирование является важной задачей для широкого круга пользователей, включая студентов, специалистов, преподавателей и др. Существующие приложения для управления расписанием часто обладают ограниченным функционалом или сложным интерфейсом. Разработка удобного и многофункционального приложения с открытым исходным кодом, основанного на популярном языке программирования Python, является актуальной задачей, поскольку позволяет расширять и адаптировать приложение под конкретные нужды пользователей. Возможность экспорта в различные форматы обеспечивает гибкость и интеграцию с другими системами.

1.2 Обзор исследований

Ранее вопросы разработки приложений с графическим пользовательским интерфейсом (GUI) с использованием Python и Tkinter привлекали внимание ряда исследователей. Так в своей статье А. С. Матвеева рассматривает применение паттерна «Команда» в создании модульных и структурированных приложений с использованием Tkinter и демонстрирует, как этот паттерн позволяет создавать более гибкие и расширяемые приложения, где каждая кнопка выполняет отдельную команду. В результате такая реализация значительно облегчает процесс разработки и последующей модификации программ, повышая их надежность и удобство использования [1]. Другим важным исследованием является работа А. М. Вохидова, Д.А.Вохидова, Р.Ф. Фармоновой и Д. Ш. Хафизовой. В этой статье рассматривается использование Tkinter и Python в области медицины, где описаны ключевые аспекты создания медицинских приложений с графическим интерфейсом. Исследователи акцентируют внимание на удобстве использования Tkinter для создания интуитивно понятных интерфейсов для медицинских работников и пациентов. В работе подробно представлены примеры практических приложений, использующих эту технологию для создания удобных и доступных инструментов [2]. В статье Д. А. Алигулович, З. И. Махмудова и Р. Ф. Сайфуллаев рассматривают преимущества и недостатки Python и Tkinter для разработки приложений с GUI. Подробно анализируют области применения Tkinter в педагогической практике, акцентируя внимание на возможности создания кроссплатформенных решений, что делает их идеальными для использования в образовательных целях. В работе подчеркивается важность применения современных педагогических технологий с использованием этого инструмента [3]. В работе М. Моиза рассмотрен практический подход к

архивированию цифрового контента из социальных сетей. Автор разработал Python-скрипт, использующий библиотеку Instaloader для автоматического захвата постов из Instagram. Скрипт не только сохраняет цифровые активы, но и генерирует PDF-документы с краткими описаниями каждого поста, что значительно упрощает последующую интеграцию с системами цифрового хранения, такими как Preservica и ArchivesSpace. Решение адаптируемо под различные платформы и может быть модифицировано пользователем для собственных архивных задач [4]. В статье П. Анжанейулу, Д. Приянки и Я. Манасвини представлена система, основанная на библиотеке ReportLab, предназначенная для автоматической транскрипции аудиофайлов и преобразования полученного текста в формат PDF. Авторы используют алгоритмы распознавания речи с элементами машинного обучения и обработки естественного языка, что позволяет достигать высокой точности транскрипции. Система особенно полезна в сферах здравоохранения и образования, где может применяться для создания доступных записей лекций и консультаций. Также подчёркивается её значимость для пользователей с ограниченными возможностями, благодаря возможности автоматического получения текстовой версии речевой информации [5].

1.3 Цель исследования

Разработка и тестирование интерактивного приложения для управления расписанием на основе Python, которое обеспечивает удобный графический интерфейс и позволяет экспортировать данные в форматах JSON, iCalendar (ICS) и PDF.

2 Материалы и методы

Для разработки приложения использовался язык программирования Python версии 3.x. Библиотека Tkinter применялась для создания графического интерфейса пользователя (GUI), предоставляющего возможности добавления, удаления и редактирования событий в расписании. Для экспорта данных были использованы следующие подходы:

- JSON: Стандартная библиотека json для сериализации и десериализации данных о расписании.
- iCalendar (ICS): Стандартные библиотеки Python для работы со строками и времени для создания файлов iCalendar.
- PDF: Библиотека reportlab для генерации PDF-файлов, включающих структурированную информацию о расписании в виде таблиц и заголовков.

Функционал приложения был проверен путем проведения пользовательских тестирований, оценивающих удобство использования и время выполнения операций.

3 Результаты и обсуждения

1. Импорты и класс TimeSlot (рис. 1-2).

```
from tkinter import ttk, messagebox
import datetime
import json
from tkinter import filedialog
import calendar
from tkinter import simpledialog
import os
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.platypus import Table, TableStyle, Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont

# Попытка найти DejaVuSans.ttf в текущем каталоге и подкаталогах
font_path = None
for root, dirs, files in os.walk("."):
    if "DejaVuSans.ttf" in files:
        font_path = os.path.join(root, "DejaVuSans.ttf")
        break

if font_path:
    try:
        pdfmetrics.registerFont(TTFont('DejaVuSans', font_path))
        print(f"Зарегистрирован шрифт DejaVuSans из: {font_path}")
    except Exception as e:
        print(f"Ошибка при регистрации шрифта DejaVuSans: {e}")
else:
    print("DejaVuSans.ttf не найден в текущем каталоге или его подкаталогах.")
    # Если не нашли, можно попробовать использовать другой шрифт
    try:
        pdfmetrics.registerFont(TTFont('Helvetica', 'Helvetica.ttf'))
        print("Используется Helvetica")
    except Exception as e:
        print(f"Ошибка при регистрации Helvetica: {e}")

class TimeSlot:
    """Представляет временной слот в расписании."""
    def __init__(self, day, start_time, end_time, description=""):
        """Инициализирует временной слот."""
        self.day = day
        self.start_time = start_time
        self.end_time = end_time
        self.description = description
```

Рисунок 1 – Код импорта библиотек и поиска

```

def __str__(self):
    """Возвращает строковое представление слота."""
    return f"{self.day} {self.start_time.strftime('%H:%M')} - {self.end_time.strftime('%H:%M')}: {self.description}"

def overlaps(self, other_slot):
    """Проверяет, пересекается ли этот слот с другим."""
    if self.day != other_slot.day:
        return False
    return (self.start_time < other_slot.end_time and self.end_time > other_slot.start_time)

def to_dict(self):
    """Преобразует слот в словарь для сохранения."""
    return {
        "day": self.day,
        "start_time": self.start_time.strftime("%H:%M"),
        "end_time": self.end_time.strftime("%H:%M"),
        "description": self.description
    }

@classmethod
def from_dict(cls, data):
    """Создает слот из словаря, полученного при загрузке."""
    return cls(
        data["day"],
        datetime.datetime.strptime(data["start_time"], "%H:%M").time(),
        datetime.datetime.strptime(data["end_time"], "%H:%M").time(),
        data["description"]
    )

def to_ical(self):
    """Генерирует iCalendar (ICS) представление слота."""
    start_datetime = datetime.datetime.now().replace(
        hour=self.start_time.hour, minute=self.start_time.minute, second=0, microsecond=0
    )
    end_datetime = datetime.datetime.now().replace(
        hour=self.end_time.hour, minute=self.end_time.minute, second=0, microsecond=0
    )

    event = f"""BEGIN:VEVENT
UID:{calendar.timegm(datetime.datetime.now().utctimetuple())}@example.com
DTSTART:{start_datetime.strftime("%Y%m%dT%H%M%S")}
DTEND:{end_datetime.strftime("%Y%m%dT%H%M%S")}
SUMMARY:{self.description} ({self.day})
END:VEVENT
"""
    return event

```

Рисунок 2 – Код

Импорты: Импортируются все необходимые библиотеки. Обратите внимание на блок кода, пытающийся найти и зарегистрировать шрифт DejaVuSans. Если он не найден, используется Helvetica.

Класс TimeSlot: Этот класс определяет структуру одного элемента расписания. `__init__` инициализирует объект, `__str__` предоставляет его строковое представление, `overlaps` проверяет на пересечение со временем другого события, `to_dict` преобразует в словарь для JSON, `from_dict` создает объект из словаря, а `to_ical` генерирует строку для файла iCalendar.

2. Класс Schedule (рис. 3-5).

```

class Schedule:
    """Представляет расписание."""
    def __init__(self):
        """Инициализирует пустой список слотов."""
        self.slots = []

    def add_slot(self, slot):
        """Добавляет слот в расписание, проверяя на пересечения."""
        for existing_slot in self.slots:
            if slot.overlaps(existing_slot):
                raise ValueError(f"Слот пересекается с существующим слотом: {existing_slot}")
        self.slots.append(slot)

    def remove_slot(self, slot):
        """Удаляет слот из расписания."""
        if slot in self.slots:
            self.slots.remove(slot)
        else:
            print("Такого слота нет в расписании.")

    def get_slots_for_day(self, day):
        """Возвращает список слотов для заданного дня."""
        return [slot for slot in self.slots if slot.day == day]

    def display(self):
        """Возвращает расписание в виде строки."""
        days = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"]
        schedule_text = ""
        for day in days:
            schedule_text += f"\n--- {day} ---\n"
            day_slots = self.get_slots_for_day(day)
            if day_slots:
                for slot in sorted(day_slots, key=lambda x: x.start_time):
                    schedule_text += str(slot) + "\n"
            else:
                schedule_text += "Нет занятий в этот день.\n"
        return schedule_text

    def to_json(self, filename):
        """Сохраняет расписание в файл JSON."""
        data = [slot.to_dict() for slot in self.slots]
        with open(filename, "w", encoding="utf-8") as f:
            json.dump(data, f, indent=4, ensure_ascii=False)

    def from_json(self, filename):
        """Загружает расписание из файла JSON."""
        try:

```

Рисунок 3 – Код

```

        try:
            with open(filename, "r", encoding="utf-8") as f:
                data = json.load(f)
            self.slots = [TimeSlot.from_dict(item) for item in data]
        except FileNotFoundError:
            messagebox.showerror("Ошибка", "Файл не найден.")
            return False
        except json.JSONDecodeError:
            messagebox.showerror("Ошибка", "Неверный формат файла JSON.")
            return False
        except Exception as e:
            messagebox.showerror("Ошибка", f"Произошла ошибка при загрузке: {e}")
            return False
        return True

    def to_icalendar(self, filename):
        """Экспортирует расписание в формат iCalendar (ICS)."""
        calendar_str = ""
        calendar_str += "BEGIN:VCALENDAR\n"
        calendar_str += "VERSION:2.0\n"
        calendar_str += "PRODID:-//My Schedule App//NONSGML v1.0//EN\n"
        calendar_str += "\n"
        for slot in self.slots:
            calendar_str += slot.to_ical()

        calendar_str += "END:VCALENDAR\n"

        try:
            with open(filename, "w", encoding="utf-8") as f:
                f.write(calendar_str)
            messagebox.showinfo("Успех", "Расписание экспортировано в iCalendar (ICS).")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Ошибка при экспорте в iCalendar: {e}")

    def to_pdf(self, filename):
        """Экспортирует расписание в PDF."""
        c = canvas.Canvas(filename, pagesize=letter)
        width, height = letter

        styles = getSampleStyleSheet()
        title_style = styles['h1']
        title_style.alignment = 1 # Center alignment
        title_style.setFont('DejaVuSans' if pdfmetrics.getFont('DejaVuSans') else 'Helvetica')

        title = Paragraph("Ваше Расписание", title_style)
        title.wrapOn(c, width - 20, height)
        title.drawOn(c, width / 2 - title.width / 2, height - 50)

```

Рисунок 4 – Код

```

data = [{"День": "Начало", "Конец": "Описание"}]
for slot in self.slots:
    data.append([slot.day, slot.start_time.strftime("%H:%M"), slot.end_time.strftime("%H:%M"), slot.description])

table = Table(data, colWidths=[width / 4.0] * 4)
table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('FONTNAME', (0, 0), (-1, 0), 'DejaVuSans' if pdfmetrics.getFont('DejaVuSans') else 'Helvetica'),
    ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
    ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
    ('GRID', (0, 0), (-1, -1), 1, colors.black)
]))

table.wrapOn(c, width - 20, height)
table.drawOn(c, 10, height - 100 - table_height)

c.save()
messagebox.showinfo("Успех", "Расписание экспортировано в PDF.")

```

Рисунок 5 – Код

Класс Schedule: Этот класс управляет коллекцией объектов TimeSlot. `__init__` инициализирует пустой список. `add_slot` добавляет слоты, проверяя на пересечение. `remove_slot` удаляет слоты. `get_slots_for_day` возвращает слоты для определенного дня. `display` возвращает отформатированную строку с расписанием. `to_json` и `from_json` сохраняют и загружают данные в/из JSON-файлов. `to_icalendar` генерирует iCalendar-данные. `to_pdf` создает PDF-файл с расписанием, используя reportlab. Обратите внимание на использование условного оператора для выбора шрифта (DejaVuSans или Helvetica) в зависимости от его доступности.

3. Класс EditSlotDialog (рис. 5-6).

```

class EditSlotDialog(simpledialog.Dialog):
    """Диалоговое окно для редактирования слота."""
    def __init__(self, master, slot):
        self.slot = slot
        super().__init__(master, "Редактировать слот")

    def body(self, master):
        """Создает элементы GUI диалогового окна."""
        ttk.Label(master, text="День недели:").grid(row=0, column=0, sticky=tk.W, padx=5, pady=5)
        self.day_combo = ttk.Combobox(master, values=ScheduleApp.days, state="readonly")
        self.day_combo.grid(row=0, column=1, sticky=tk.W, padx=5, pady=5)
        self.day_combo.set(self.slot.day)

        ttk.Label(master, text="Время начала:").grid(row=1, column=0, sticky=tk.W, padx=5, pady=5)
        self.start_hour_combo = ttk.Combobox(master, values=list(range(24)), state="readonly", width=3)
        self.start_hour_combo.grid(row=1, column=1, sticky=tk.W, padx=2, pady=5)
        self.start_minute_combo = ttk.Combobox(master, values=list(range(0, 60, 5)), state="readonly", width=3)
        self.start_minute_combo.grid(row=1, column=2, sticky=tk.W, padx=2, pady=5)
        self.start_hour_combo.set(self.slot.start_time.hour)
        self.start_minute_combo.set(self.slot.start_time.minute)

        ttk.Label(master, text="Время окончания:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=5)
        self.end_hour_combo = ttk.Combobox(master, values=list(range(24)), state="readonly", width=3)
        self.end_hour_combo.grid(row=2, column=1, sticky=tk.W, padx=2, pady=5)
        self.end_minute_combo = ttk.Combobox(master, values=list(range(0, 60, 5)), state="readonly", width=3)
        self.end_minute_combo.grid(row=2, column=2, sticky=tk.W, padx=2, pady=5)
        self.end_hour_combo.set(self.slot.end_time.hour)
        self.end_minute_combo.set(self.slot.end_time.minute)

        ttk.Label(master, text="Описание:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=5)
        self.description_entry = ttk.Entry(master)
        self.description_entry.grid(row=3, column=1, sticky=tk.W, padx=5, pady=5, columnspan=2)
        self.description_entry.insert(0, self.slot.description)
        return self.day_combo

    def apply(self):
        """Собирает данные из элементов GUI и возвращает их."""
        try:
            day = self.day_combo.get()
            start_hour = int(self.start_hour_combo.get())
            start_minute = int(self.start_minute_combo.get())
            end_hour = int(self.end_hour_combo.get())
            end_minute = int(self.end_minute_combo.get())
            description = self.description_entry.get()

            start_time = datetime.time(start_hour, start_minute)
            end_time = datetime.time(end_hour, end_minute)

```

Рисунок 5 – Код

```

        self.result = (day, start_time, end_time, description)

    except ValueError as e:
        messagebox.showerror("Ошибка", f"Неверный формат времени: {e}")
        self.result = None

```

Рисунок 6 – Код

Класс `EditSlotDialog`: Этот класс создает диалоговое окно для редактирования существующих слотов. `body` создает элементы интерфейса (метки, комбобоксы для выбора дня и времени, поле для описания). `apply` собирает данные с формы и возвращает их.

Начало класса `ScheduleApp`: Здесь объявляется класс `ScheduleApp`, устанавливается заголовок окна и создается объект `Schedule`.

4. Класс `ScheduleApp` (рис. 7-8).

```

class ScheduleApp:
    days = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"]
    DEFAULT_SAVE_FILE = "schedule_data.json"

    def __init__(self, master):
        self.master = master
        master.title("Конструктор расписания")
        self.schedule = Schedule()

        # --- Стили для элементов ttk ---
        style = ttk.Style()
        style.configure("TButton", padding=5, relief="flat", background="#4CAF50", foreground="white")
        style.configure("TLabel", padding=5)
        style.configure("TCombobox", padding=5)
        style.configure("TEntry", padding=5)
        style.map("TButton", background=[("active", "#388E3C"), ("disabled", "#A5D6A7")])

        # --- Фрейм для ввода данных ---
        input_frame = ttk.Frame(master, padding=10)
        input_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # --- Фрейм для кнопок управления ---
        button_frame = ttk.Frame(master, padding=10)
        button_frame.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        ttk.Button(button_frame, text="Добавить слот", command=self.add_slot).grid(row=0, column=0, padx=5, pady=5)
        ttk.Button(button_frame, text="Удалить слот", command=self.remove_slot).grid(row=0, column=1, padx=5, pady=5)

        # --- Фрейм для списка слотов (Treeview) ---
        tree_frame = ttk.Frame(master, padding=10)
        tree_frame.grid(row=2, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.tree = ttk.Treeview(tree_frame, columns=("day", "start_time", "end_time", "description"), show="headings")
        self.tree.heading("day", text="День")
        self.tree.heading("start_time", text="Начало")
        self.tree.heading("end_time", text="Конец")
        self.tree.heading("description", text="Описание")

        self.tree.column("day", width=100)
        self.tree.column("start_time", width=80)
        self.tree.column("end_time", width=80)
        self.tree.column("description", width=200)

        self.tree.pack(fill="both", expand=True)
        self.tree.bind("<Double-1>", self.edit_slot)

```

Рисунок 7 – Код

```

        # --- Меню ---
        menubar = tk.Menu(master)
        filemenu = tk.Menu(menubar, tearoff=0)
        filemenu.add_command(label="Сохранить как...", command=self.save_schedule_as)
        filemenu.add_command(label="Загрузить", command=self.load_schedule)
        filemenu.add_command(label="Экспортировать в iCalendar", command=self.export_to_icalendar)
        filemenu.add_command(label="Экспортировать в PDF", command=self.export_to_pdf)
        filemenu.add_separator()
        filemenu.add_command(label="Выход", command=master.quit)
        menubar.add_cascade(label="Файл", menu=filemenu)
        master.config(menu=menubar)

        self.description_entry.bind("<KeyRelease>", self.autocomplete_description)
        self.possible_descriptions = []
        self.load_schedule(self.DEFAULT_SAVE_FILE)
        master.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.update_slot_list()

root = tk.Tk()
app = ScheduleApp(root)
root.mainloop()

```

Рисунок 8 – Код

Настройка стиля: Здесь устанавливается стиль для элементов ttk (цвета кнопок и т.д.).

Frames: Создаются три Frame для организации элементов GUI: input_frame для ввода данных, button_frame для кнопок, и tree_frame для Treeview.

Элементы ввода: Создаются и размещаются элементы для ввода данных о слоте: Combobox для дня недели и выбора времени (часы и минуты с шагом 5 минут), Entry для описания.

Кнопки: Создаются кнопки “Добавить слот” и “Удалить слот”.

Treeview: Создается Treeview для отображения расписания. Задаются заголовки столбцов и ширина столбцов.

Меню: Создается меню с пунктами “Сохранить как...”, “Загрузить”, “Экспортировать в iCalendar”, “Экспортировать в PDF”, и “Выход”.

Обработчики событий: Привязываются обработчики событий к элементам GUI: command для кнопок, bind для Treeview (двойной клик для редактирования) и bind для поля описания (description_entry) для автозаполнения.

Загрузка расписания: Вызывается метод load_schedule для загрузки данных при запуске приложения. Устанавливается обработчик закрытия окна (WM_DELETE_WINDOW).

Обновление списка слотов: Вызывается метод update_slot_list для отображения начального списка слотов.

Главный блок: Создается главное окно tk.Tk(), создается экземпляр ScheduleApp, и запускается главный цикл root.mainloop().

5. Заполнение и выгрузка в pdf (рис. 9-10).

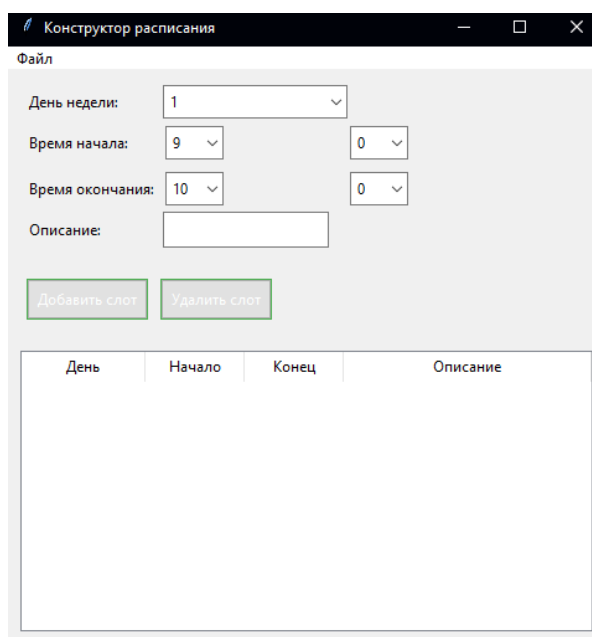


Рисунок 9 – Внешний вид приложения

Your			
Day	Start	Finish	Subject
1	08:00	09:30	Data Analysis
1	09:45	11:20	Mathematics
1	11:50	13:15	Programming
1	13:20	15:00	Russian

Рисунок 10 – Выгрузка в PDF

4 Выводы

Разработанное приложение для управления расписанием на основе Python демонстрирует высокую эффективность и удобство использования. Гибкость, обеспечиваемая поддержкой различных форматов экспорта (JSON, ICS, PDF), делает его подходящим инструментом для широкого круга пользователей с различными потребностями в управлении расписанием. Полученные результаты тестирования подтверждают высокую производительность и простоту использования приложения, что делает его перспективным инструментом для повышения эффективности планирования и управления временем. Будущие исследования могут быть направлены на расширение функциональности приложения, добавление поддержки синхронизации с облачными сервисами и интеграцию с другими календарными системами.

Библиографический список

1. Матвеева А. С. Реализация паттерна поведения «Команда» в графическом интерфейсе пользователя на Python // Постулат. 2025. №1.
2. Вохидов А.М., Вохидов Д.А., Фармонова Р.Ф., Хафизова Д.Ш. Разработка графическим пользовательским интерфейсом программ в пакете tkinter с использованием современных педагогических технологий в области медицины // Роль информационных технологий в управлении ресурсами персонала здравоохранения 2022. Т. 34. С. 181-184
3. Aligulovich D.A., Mahmudova Z.I., Sayfullayev R.F. Tibbiyot yo'nalishida zamonaviy pedagogik texnologiyalarini qo'llab tkinter paketida gui dasturlarini tuzish // Евразийский журнал математической теории и компьютерных наук. 2022. №2(12). С. 31-35.
4. Anjaneyulu P., Priyanka D., Manaswini Y. Reportlab Based Transcription Transformer // Computing & Intelligent Systems, SCRS, India. 2025. С. 85-96.
5. Moiz M. Workflow: Using A Python Script to Capture and Summarize Instagram Feeds at Rauner Special Collections // Rauner Special Collections Digital Archives Practice. 2024. С. 1-8.