Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное   
образовательное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

От проблемы к решению

Пояснительная записка к хакатон по учебной дисциплине «Основы программирования: Алгоритмические языки и программирование»

по специальности 09.03.01 Информатика и вычислительная техника  
ПТИ.ЛР 4093 004.002ПЗ

|  |  |
| --- | --- |
|  | Руководитель  / И. Ю. Кулаков  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |
|  | Студент группы 4093  / М.М.Нематжанов  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |

Содержание

[Введение 3](#_Toc196263865)

[1 Постановка задачи 4](#_Toc196263866)

[1.1 Обоснование необходимости разработки 4](#_Toc196263867)

[1.2 Технико-математическое описание задачи 4](#_Toc196263868)

[1.3 Требования к программе 4](#_Toc196263869)

[2 Теоретическая часть 5](#_Toc196263870)

[2.1 Используемые технологии 5](#_Toc196263871)

[2.2 Архитектура решения 5](#_Toc196263872)

[3 Практическая часть 6](#_Toc196263873)

[3.1 Основные функции 6](#_Toc196263874)

[3.2 Примеры использования 6](#_Toc196263875)

[3.3 Ограничения текущей реализации 6](#_Toc196263876)

[Заключение 7](#_Toc196263877)

[Список литературы 8](#_Toc196263878)

[Приложения А 9](#_Toc196263879)

[Приложение Б 11](#_Toc196263880)

Введение

Целью данного проекта является разработка Telegram бота для автоматизации процесса управления расписанием занятий в учебных заведениях. Бот позволяет администраторам факультетов и преподавателям вносить данные о группах, редактировать расписание, удалять устаревшие записи и экспортировать данные в удобном формате. Решение направлено на сокращение времени, затрачиваемого на ручное составление расписания, и минимизацию человеческих ошибок.

Целевая аудитория:

- администраторы учебных заведений

- преподаватели.

1 Постановка задачи

1.1 Обоснование необходимости разработки

Ручное составление расписания требует значительных временных затрат и подвержено ошибкам. Telegram бот предоставляет интуитивно понятный интерфейс для быстрого внесения изменений, автоматизирует хранение данных и обеспечивает мгновенный доступ к актуальной информации.

1.2 Технико-математическое описание задачи

- хранение данных: SQLite;

- логика бота: Python, библиотека python-telegram-bot;

- экспорт данных: Excel.

1.3 Требования к программе

- Поддержка добавления групп с учетом факультета и курса;

- Редактирование расписания по дням недели и временным слотам;

- Экспорт данных в Excel;

- Совместимость с Telegram на всех платформах.

2 Теоретическая часть

2.1 Используемые технологии

- python: Основной язык разработки;

- SQLite: Для хранения данных о группах и расписании;

- python-telegram-bot: Фреймворк для создания Telegram ботов.

2.2 Архитектура решения

База данных:

- таблица groups: факультет, курс, название группы;

- таблица schedule\_entries: расписание с привязкой к группе, дню и времени.

Состояния бота:

- выбор факультета, курса, группы;

- редактирование расписания;

- экспорт данных.

3 Практическая часть

3.1 Основные функции

Добавление групп:

- пользователь выбирает факультет, курс и вводит название группы;

- данные сохраняются в таблицу groups.

Управление расписанием:

- для выбранной группы можно добавить, изменить или удалить занятия по дням недели;

- временные слоты: с 6:00 до 21:00.

Экспорт в Excel:

- Расписание экспортируется в формате .xlsx с сортировкой по группам и времени.

3.2 Примеры использования

Добавление занятия:  
Пользователь выбирает группу, день недели и вводит данные в формате:

10:00 - Математика

12:00 – Физика

Экспорт расписания:  
Бот генерирует Excel-файл с расписанием на выбранный день.

3.3 Ограничения текущей реализации

Не учтены требования ТЗ кейса 2:

- Время перемещения между корпусами.

Заключение

Разработанный Telegram бот решает базовые задачи управления расписанием, но требует доработки для полного соответствия техническому заданию кейса 2.

Рекомендации по улучшению:

- добавить проверку на пересечение занятий в разных корпусах;

- внедрить уведомления о изменениях в расписании.

Список литературы

1. Документация библиотеки [python telegram bot](https://python-telegram-bot.org).

2. Официальная документация [SQlite](https://sqlite.org).

3. Учебные материалы с сайта [itproger](https://itproger.com/course/telegram-bot).

Приложения А

(обязательное)

 Рисунок А.1 — Выбор факультета

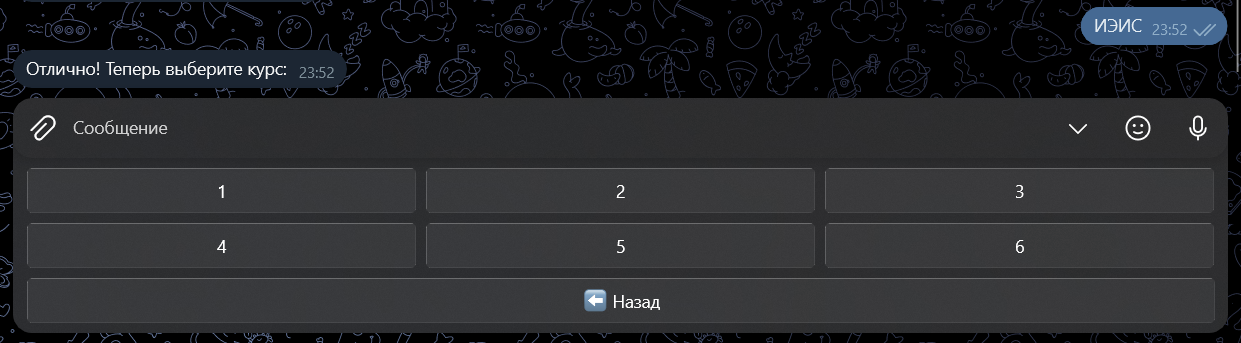
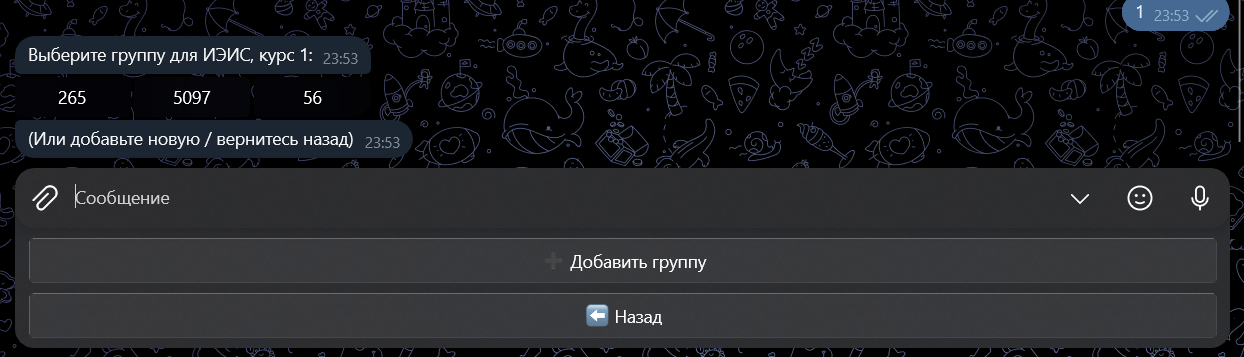
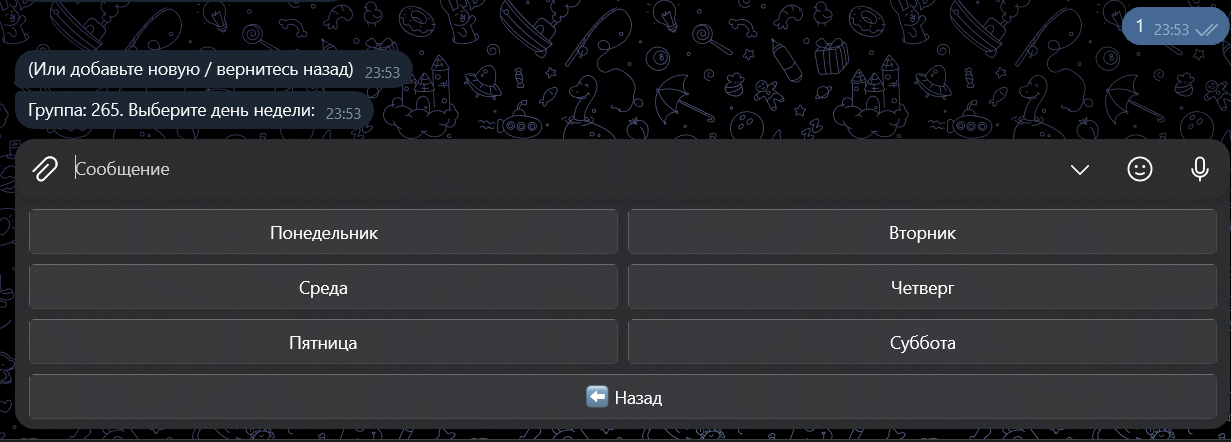


Рисунок А.2 — Выбор курса

 Рисунок А.3 — Выбор группы или добавление группы

 Рисунок А.3 — Выбор дня недели

# Приложение Б (обязательное)

Листинг Д.1 — Код файла case2.py

import logging

import sqlite3

import os

import pandas as pd

from telegram import Update, ReplyKeyboardMarkup, InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboardRemove

from telegram.ext import (

    Application,

    CommandHandler,

    ConversationHandler,

    CallbackQueryHandler,

    MessageHandler,

    filters,

    ContextTypes,

)

import datetime

logging.basicConfig(

    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO

)

logger = logging.getLogger(\_\_name\_\_)

TELEGRAM\_BOT\_TOKEN = "7580331808:AAHxGMkM7ypuJAqmzwt-J1u-\_XqX\_R5Oepw"

DB\_NAME = "schedule\_bot\_v2.db"

FACULTIES = ["ИЭИС", "ИЦЭУС", "ПИ", "ИБХИ", "ИГУМ", "ИМО", "ИЮР", "ИПТ", "ПТИ"]

COURSES = ["1", "2", "3", "4", "5", "6"]

DAYS\_OF\_WEEK = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота"]

TIME\_SLOTS = [f"с {h}:00 до {h+1}:00" for h in range(6, 21)]

(SELECT\_FACULTY, SELECT\_COURSE, SELECT\_GROUP, ADD\_GROUP\_PROMPT,

 SELECT\_DAY, ENTER\_SCHEDULE, POST\_SAVE\_OPTIONS, EXPORT\_ASK\_DAY) = range(8)

(CALLBACK\_FACULTY, CALLBACK\_COURSE, CALLBACK\_GROUP, CALLBACK\_DAY) = ("FACULTY", "COURSE", "GROUP", "DAY")

def init\_db():

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS groups (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            faculty TEXT NOT NULL,

            course INTEGER NOT NULL,

            group\_name TEXT NOT NULL,

            UNIQUE(faculty, course, group\_name)

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS schedule\_entries (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            faculty TEXT NOT NULL,

            course INTEGER NOT NULL,

            group\_name TEXT NOT NULL,

            day\_of\_week TEXT NOT NULL,

            time\_slot TEXT NOT NULL,

            subject TEXT NOT NULL

        )

    """)

    conn.commit()

    conn.close()

    logger.info(f"База данных {DB\_NAME} инициализирована.")

def add\_group\_db(faculty: str, course: int, group\_name: str) -> bool:

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    try:

        cursor.execute(

            "INSERT INTO groups (faculty, course, group\_name) VALUES (?, ?, ?)",

            (faculty, course, group\_name)

        )

        conn.commit()

        logger.info(f"Добавлена группа: {faculty}, Курс {course}, {group\_name}")

        return True

    except sqlite3.IntegrityError:

        logger.warning(f"Попытка добавить существующую группу: {faculty}, Курс {course}, {group\_name}")

        return False

    finally:

        conn.close()

def get\_groups\_db(faculty: str, course: int) -> list:

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    cursor.execute(

        "SELECT group\_name FROM groups WHERE faculty = ? AND course = ? ORDER BY group\_name",

        (faculty, course)

    )

    groups = [row[0] for row in cursor.fetchall()]

    conn.close()

    return groups

def save\_schedule\_entry\_db(faculty: str, course: int, group\_name: str, day: str, time\_slot: str, subject: str):

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    try:

        cursor.execute(

            "INSERT INTO schedule\_entries (faculty, course, group\_name, day\_of\_week, time\_slot, subject) VALUES (?, ?, ?, ?, ?, ?)",

            (faculty, course, group\_name, day, time\_slot, subject)

        )

        conn.commit()

        logger.info(f"Сохранена запись: {faculty}, К{course}, {group\_name}, {day}, {time\_slot}, {subject}")

    except Exception as e:

        logger.error(f"Ошибка сохранения записи в БД: {e}")

    finally:

        conn.close()

def delete\_schedule\_for\_day\_db(faculty: str, course: int, group\_name: str, day: str):

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    cursor.execute(

        "DELETE FROM schedule\_entries WHERE faculty = ? AND course = ? AND group\_name = ? AND day\_of\_week = ?",

        (faculty, course, group\_name, day)

    )

    conn.commit()

    conn.close()

    logger.info(f"Удалены записи для {faculty}, К{course}, {group\_name}, {day}")

def get\_schedule\_data\_db(faculty: str = None, course: int = None, group\_name: str = None, day: str = None) -> list:

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    query = "SELECT faculty, course, group\_name, day\_of\_week, time\_slot, subject FROM schedule\_entries"

    conditions = []

    params = []

    if faculty:

        conditions.append("faculty = ?")

        params.append(faculty)

    if course is not None:

        conditions.append("course = ?")

        params.append(course)

    if group\_name:

        conditions.append("group\_name = ?")

        params.append(group\_name)

    if day:

        conditions.append("day\_of\_week = ?")

        params.append(day)

    if conditions:

        query += " WHERE " + " AND ".join(conditions)

    query += " ORDER BY faculty, course, group\_name, day\_of\_week, time\_slot"

    cursor.execute(query, params)

    data = cursor.fetchall()

    conn.close()

    return data

def create\_reply\_keyboard(buttons: list, columns: int, one\_time: bool = True, add\_back: bool = False, add\_add\_group: bool = False, custom\_buttons: list = None) -> ReplyKeyboardMarkup:

    keyboard = []

    row = []

    for i, button\_text in enumerate(buttons):

        row.append(button\_text)

        if (i + 1) % columns == 0 or i == len(buttons) - 1:

            keyboard.append(row)

            row = []

    if add\_add\_group:

         keyboard.append(["➕ Добавить группу"])

    if custom\_buttons:

        for btn\_row in custom\_buttons:

             keyboard.append(btn\_row)

    if add\_back:

        keyboard.append(["⬅️ Назад"])

    return ReplyKeyboardMarkup(keyboard, resize\_keyboard=True, one\_time\_keyboard=one\_time)

def create\_inline\_keyboard(buttons: list, columns: int) -> InlineKeyboardMarkup:

    keyboard = []

    row = []

    for i, button\_text in enumerate(buttons):

        row.append(InlineKeyboardButton(button\_text, callback\_data=button\_text))

        if (i + 1) % columns == 0 or i == len(buttons) - 1:

            keyboard.append(row)

            row = []

    return InlineKeyboardMarkup(keyboard)

def time\_slot\_to\_start\_time(time\_slot: str) -> str:

    parts = time\_slot.split()

    if len(parts) >= 2:

        return parts[1]

    return "?"

async def start(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    user = update.effective\_user

    context.user\_data.clear()

    logger.info(f"Пользователь {user.username} ({user.id}) запустил бота.")

    reply\_markup = create\_reply\_keyboard(FACULTIES, columns=3)

    await update.message.reply\_text(

        f"Привет, {user.first\_name}! 👋\n"

        "Я помогу тебе внести данные для расписания.\n\n"

        "Выберите институт/факультет:",

        reply\_markup=reply\_markup,

    )

    return SELECT\_FACULTY

async def select\_faculty(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    faculty = update.message.text

    if faculty not in FACULTIES:

        await update.message.reply\_text("Пожалуйста, выберите факультет из предложенных кнопок.")

        return SELECT\_FACULTY

    context.user\_data[CALLBACK\_FACULTY] = faculty

    logger.info(f"Пользователь {update.effective\_user.id} выбрал факультет: {faculty}")

    reply\_markup = create\_reply\_keyboard(COURSES, columns=3, add\_back=True)

    await update.message.reply\_text("Отлично! Теперь выберите курс:", reply\_markup=reply\_markup)

    return SELECT\_COURSE

async def select\_course(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    course\_text = update.message.text

    if course\_text not in COURSES:

         await update.message.reply\_text("Пожалуйста, выберите курс из предложенных кнопок.")

         return SELECT\_COURSE

    course = int(course\_text)

    context.user\_data[CALLBACK\_COURSE] = course

    faculty = context.user\_data[CALLBACK\_FACULTY]

    logger.info(f"Пользователь {update.effective\_user.id} выбрал курс: {course} для факультета {faculty}")

    await send\_group\_selection(update, context)

    return SELECT\_GROUP

async def send\_group\_selection(update: Update, context: ContextTypes.DEFAULT\_TYPE, message\_id\_to\_edit: int = None):

    faculty = context.user\_data[CALLBACK\_FACULTY]

    course = context.user\_data[CALLBACK\_COURSE]

    groups = get\_groups\_db(faculty, course)

    reply\_markup\_main = create\_reply\_keyboard([], columns=1, add\_back=True, add\_add\_group=True)

    if groups:

        inline\_markup = create\_inline\_keyboard(groups, columns=3)

        message\_text = f"Выберите группу для {faculty}, курс {course}:"

        if message\_id\_to\_edit:

             try:

                  await context.bot.edit\_message\_text(

                       chat\_id=update.effective\_chat.id,

                       message\_id=message\_id\_to\_edit,

                       text=message\_text,

                       reply\_markup=inline\_markup

                  )

                  await update.message.reply\_text("(Или добавьте новую / вернитесь назад)", reply\_markup=reply\_markup\_main)

             except Exception as e:

                  logger.error(f"Не удалось отредактировать сообщение выбора группы: {e}. Отправляю новое.")

                  await update.message.reply\_text(message\_text, reply\_markup=inline\_markup)

                  await update.message.reply\_text("(Или добавьте новую / вернитесь назад)", reply\_markup=reply\_markup\_main)

        else:

             await update.message.reply\_text(message\_text, reply\_markup=inline\_markup)

             await update.message.reply\_text("(Или добавьте новую / вернитесь назад)", reply\_markup=reply\_markup\_main)

    else:

        message\_text = f"Для {faculty}, курс {course} пока нет добавленных групп."

        await update.message.reply\_text(message\_text, reply\_markup=reply\_markup\_main)

async def select\_group\_inline(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    query = update.callback\_query

    await query.answer()

    group\_name = query.data

    context.user\_data[CALLBACK\_GROUP] = group\_name

    faculty = context.user\_data[CALLBACK\_FACULTY]

    course = context.user\_data[CALLBACK\_COURSE]

    logger.info(f"Пользователь {query.from\_user.id} выбрал группу: {group\_name} ({faculty}, {course})")

    await query.delete\_message()

    reply\_markup = create\_reply\_keyboard(DAYS\_OF\_WEEK, columns=2, add\_back=True)

    await query.message.reply\_text(

        f"Группа: {group\_name}. Выберите день недели:",

        reply\_markup=reply\_markup

    )

    return SELECT\_DAY

async def prompt\_add\_group(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    logger.info(f"Пользователь {update.effective\_user.id} нажал 'Добавить группу'.")

    try:

        context.user\_data['group\_select\_message\_id'] = update.message.message\_id - 1

    except:

        context.user\_data['group\_select\_message\_id'] = None

    await update.message.reply\_text("Введите название новой группы:", reply\_markup=ReplyKeyboardMarkup([["⬅️ Отмена добавления"]], resize\_keyboard=True))

    return ADD\_GROUP\_PROMPT

async def add\_group\_handler(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    new\_group\_name = update.message.text.strip()

    user = update.effective\_user

    if not new\_group\_name or len(new\_group\_name) > 50:

        await update.message.reply\_text("Название группы не может быть пустым или слишком длинным. Попробуйте еще раз или нажмите 'Отмена добавления'.")

        return ADD\_GROUP\_PROMPT

    faculty = context.user\_data[CALLBACK\_FACULTY]

    course = context.user\_data[CALLBACK\_COURSE]

    if add\_group\_db(faculty, course, new\_group\_name):

        logger.info(f"Пользователь {user.id} успешно добавил группу: {new\_group\_name}")

        await update.message.reply\_text(f"Группа '{new\_group\_name}' успешно добавлена!", reply\_markup=ReplyKeyboardRemove())

    else:

        logger.warning(f"Пользователь {user.id} пытался добавить существующую группу: {new\_group\_name}")

        await update.message.reply\_text(f"Группа '{new\_group\_name}' уже существует для этого курса и факультета.", reply\_markup=ReplyKeyboardRemove())

    message\_id\_to\_update = context.user\_data.pop('group\_select\_message\_id', None)

    await send\_group\_selection(update, context, message\_id\_to\_edit=message\_id\_to\_update)

    return SELECT\_GROUP

async def cancel\_add\_group(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    logger.info(f"Пользователь {update.effective\_user.id} отменил добавление группы.")

    await update.message.reply\_text("Добавление группы отменено.", reply\_markup=ReplyKeyboardRemove())

    message\_id\_to\_update = context.user\_data.pop('group\_select\_message\_id', None)

    await send\_group\_selection(update, context, message\_id\_to\_edit=message\_id\_to\_update)

    return SELECT\_GROUP

async def select\_day(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    day = update.message.text

    if day not in DAYS\_OF\_WEEK:

         await update.message.reply\_text("Пожалуйста, выберите день недели из предложенных кнопок.")

         return SELECT\_DAY

    context.user\_data[CALLBACK\_DAY] = day

    faculty = context.user\_data[CALLBACK\_FACULTY]

    course = context.user\_data[CALLBACK\_COURSE]

    group\_name = context.user\_data[CALLBACK\_GROUP]

    logger.info(f"Пользователь {update.effective\_user.id} выбрал день: {day} для группы {group\_name}")

    schedule\_data = get\_schedule\_data\_db(faculty=faculty, course=course, group\_name=group\_name, day=day)

    if schedule\_data:

        schedule\_text = f"Текущее расписание для {day}:\n"

        for entry in schedule\_data:

            time\_slot = entry[4]

            subject = entry[5]

            start\_time = time\_slot\_to\_start\_time(time\_slot)

            schedule\_text += f"{start\_time} - {subject}\n"

        message = schedule\_text + "\nВведите новое расписание или 'нет' для удаления."

    else:

        message = f"Расписание для {day} отсутствует. Введите расписание или 'нет'."

    await update.message.reply\_text(

        message,

        reply\_markup=ReplyKeyboardMarkup([["⬅️ Назад"]], resize\_keyboard=True)

    )

    return ENTER\_SCHEDULE

async def enter\_schedule(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    text = update.message.text.strip()

    faculty = context.user\_data[CALLBACK\_FACULTY]

    course = context.user\_data[CALLBACK\_COURSE]

    group\_name = context.user\_data[CALLBACK\_GROUP]

    day = context.user\_data[CALLBACK\_DAY]

    if text.lower() == 'нет':

        delete\_schedule\_for\_day\_db(faculty, course, group\_name, day)

        await update.message.reply\_text(f"Расписание для {day} удалено.")

    else:

        delete\_schedule\_for\_day\_db(faculty, course, group\_name, day)

        lines = text.split('\n')

        for line in lines:

            line = line.strip()

            if line:

                parts = line.split('-', 1)

                if len(parts) == 2:

                    time\_str = parts[0].strip()

                    subject = parts[1].strip()

                    try:

                        hour = int(time\_str.split(':')[0])

                        time\_slot = f"с {hour}:00 до {hour+1}:00"

                        if time\_slot in TIME\_SLOTS:

                            save\_schedule\_entry\_db(faculty, course, group\_name, day, time\_slot, subject)

                        else:

                            await update.message.reply\_text(f"Неверное время: {time\_str}. Пропускаю.")

                    except ValueError:

                        await update.message.reply\_text(f"Неверный формат времени: {time\_str}. Пропускаю.")

                else:

                    await update.message.reply\_text(f"Неверный формат строки: {line}. Пропускаю.")

        await update.message.reply\_text(f"Расписание для {day} сохранено.")

    final\_options\_keyboard = [

        ["➕ Добавить еще запись"], [" EЯ Выбрать другую группу"],

        ["📊 Вывести расписание дня"], ["✅ Завершить"]

    ]

    reply\_markup = ReplyKeyboardMarkup(final\_options\_keyboard, resize\_keyboard=True, one\_time\_keyboard=False)

    await update.message.reply\_text("Что делаем дальше?", reply\_markup=reply\_markup)

    return POST\_SAVE\_OPTIONS

async def add\_another\_entry(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    group\_name = context.user\_data.get(CALLBACK\_GROUP, "текущая группа")

    logger.info(f"Пользователь {update.effective\_user.id} решил добавить еще запись для группы {group\_name}.")

    context.user\_data.pop(CALLBACK\_DAY, None)

    reply\_markup = create\_reply\_keyboard(DAYS\_OF\_WEEK, columns=2, add\_back=True)

    await update.message.reply\_text(

        f"Добавляем еще запись для группы {group\_name}.\nВыберите день недели:",

        reply\_markup=reply\_markup

    )

    return SELECT\_DAY

async def go\_to\_group\_selection(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    logger.info(f"Пользователь {update.effective\_user.id} решил выбрать другую группу.")

    context.user\_data.pop(CALLBACK\_GROUP, None)

    context.user\_data.pop(CALLBACK\_DAY, None)

    await send\_group\_selection(update, context)

    return SELECT\_GROUP

async def prompt\_export\_day(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    logger.info(f"Пользователь {update.effective\_user.id} запросил экспорт за день.")

    reply\_markup = create\_reply\_keyboard(DAYS\_OF\_WEEK, columns=2, add\_back=True)

    await update.message.reply\_text("Выберите день недели для экспорта расписания:", reply\_markup=reply\_markup)

    return EXPORT\_ASK\_DAY

async def export\_day\_schedule(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    day\_to\_export = update.message.text

    user = update.effective\_user

    if day\_to\_export not in DAYS\_OF\_WEEK:

        await update.message.reply\_text("Пожалуйста, выберите день из кнопок.")

        return EXPORT\_ASK\_DAY

    logger.info(f"Пользователь {user.id} экспортирует расписание за {day\_to\_export}.")

    schedule\_data = get\_schedule\_data\_db(day\_filter=day\_to\_export)

    if not schedule\_data:

        await update.message.reply\_text(f"Нет записей расписания для '{day\_to\_export}'.")

        final\_options\_keyboard = [

            ["➕ Добавить еще запись"], [" EЯ Выбрать другую группу"],

            ["📊 Вывести расписание дня"], ["✅ Завершить"]

        ]

        reply\_markup = ReplyKeyboardMarkup(final\_options\_keyboard, resize\_keyboard=True, one\_time\_keyboard=False)

        await update.message.reply\_text("Что делаем дальше?", reply\_markup=reply\_markup)

        return POST\_SAVE\_OPTIONS

    try:

        df = pd.DataFrame(schedule\_data, columns=['Факультет', 'Курс', 'Группа', 'День', 'Время', 'Предмет'])

        df\_sorted = df.sort\_values(by=['Группа', 'Время'])

        current\_date = datetime.datetime.now().strftime("%Y-%m-%d")

        excel\_filename = f"schedule\_{day\_to\_export}\_{user.id}\_{current\_date}.xlsx"

        df\_sorted.to\_excel(excel\_filename, index=False, engine='openpyxl')

        logger.info(f"Excel файл {excel\_filename} создан.")

        await update.message.reply\_document(

            document=open(excel\_filename, 'rb'),

            filename=f"Расписание\_{day\_to\_export}.xlsx",

            caption=f"Вот расписание для '{day\_to\_export}' в формате Excel."

        )

        os.remove(excel\_filename)

        logger.info(f"Excel файл {excel\_filename} удален.")

    except Exception as e:

        logger.error(f"Ошибка при создании или отправке Excel файла для дня {day\_to\_export}: {e}")

        await update.message.reply\_text("Произошла ошибка при создании Excel файла. Попробуйте позже.")

    final\_options\_keyboard = [

        ["➕ Добавить еще запись"], [" EЯ Выбрать другую группу"],

        ["📊 Вывести расписание дня"], ["✅ Завершить"]

    ]

    reply\_markup = ReplyKeyboardMarkup(final\_options\_keyboard, resize\_keyboard=True, one\_time\_keyboard=False)

    await update.message.reply\_text("Что делаем дальше?", reply\_markup=reply\_markup)

    return POST\_SAVE\_OPTIONS

async def back\_handler(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    user = update.effective\_user

    logger.info(f"Пользователь {user.id} нажал Назад.")

    current\_data = context.user\_data

    if CALLBACK\_DAY in current\_data:

        del current\_data[CALLBACK\_DAY]

        logger.debug("Возврат к выбору группы")

        await send\_group\_selection(update, context)

        return SELECT\_GROUP

    elif CALLBACK\_GROUP in current\_data:

        del current\_data[CALLBACK\_GROUP]

        logger.debug("Возврат к выбору курса")

        reply\_markup = create\_reply\_keyboard(COURSES, columns=3, add\_back=True)

        await update.message.reply\_text("Выберите курс:", reply\_markup=reply\_markup)

        return SELECT\_COURSE

    elif CALLBACK\_COURSE in current\_data:

        del current\_data[CALLBACK\_COURSE]

        logger.debug("Возврат к выбору факультета")

        reply\_markup = create\_reply\_keyboard(FACULTIES, columns=3)

        await update.message.reply\_text("Выберите институт/факультет:", reply\_markup=reply\_markup)

        return SELECT\_FACULTY

    elif CALLBACK\_FACULTY in current\_data:

        del current\_data[CALLBACK\_FACULTY]

        logger.debug("Возврат в начало")

        return await start(update, context)

    else:

        logger.debug("Некуда возвращаться, переход в начало")

        return await start(update, context)

async def cancel(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    user = update.effective\_user

    logger.info(f"Пользователь {user.id} отменил диалог командой /cancel.")

    context.user\_data.clear()

    await update.message.reply\_text(

        "Действие отменено. Чтобы начать заново, введите /start.",

        reply\_markup=ReplyKeyboardRemove()

    )

    return ConversationHandler.END

async def done(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    user = update.effective\_user

    logger.info(f"Пользователь {user.id} завершил ввод расписания.")

    context.user\_data.clear()

    await update.message.reply\_text(

        "Отлично! Ввод данных завершен.\nЧтобы начать заново, введите /start.",

        reply\_markup=ReplyKeyboardRemove()

    )

    return ConversationHandler.END

def main() -> None:

    init\_db()

    application = Application.builder().token(TELEGRAM\_BOT\_TOKEN).build()

    conv\_handler = ConversationHandler(

        entry\_points=[CommandHandler("start", start)],

        states={

            SELECT\_FACULTY: [MessageHandler(filters.TEXT & ~filters.COMMAND, select\_faculty)],

            SELECT\_COURSE: [

                MessageHandler(filters.Regex("^⬅️ Назад$"), back\_handler),

                MessageHandler(filters.TEXT & ~filters.COMMAND, select\_course),

            ],

            SELECT\_GROUP: [

                MessageHandler(filters.Regex("^⬅️ Назад$"), back\_handler),

                MessageHandler(filters.Regex("^➕ Добавить группу$"), prompt\_add\_group),

                CallbackQueryHandler(select\_group\_inline),

                MessageHandler(filters.TEXT & ~filters.COMMAND, lambda update, context: update.message.reply\_text("Используйте кнопки для выбора или добавления группы.")),

            ],

            ADD\_GROUP\_PROMPT: [

                MessageHandler(filters.Regex("^⬅️ Отмена добавления$"), cancel\_add\_group),

                MessageHandler(filters.TEXT & ~filters.COMMAND, add\_group\_handler),

            ],

            SELECT\_DAY: [

                 MessageHandler(filters.Regex("^⬅️ Назад$"), back\_handler),

                 MessageHandler(filters.TEXT & ~filters.COMMAND, select\_day),

            ],

            ENTER\_SCHEDULE: [

                MessageHandler(filters.Regex("^⬅️ Назад$"), back\_handler),

                MessageHandler(filters.TEXT & ~filters.COMMAND, enter\_schedule),

            ],

            POST\_SAVE\_OPTIONS: [

                MessageHandler(filters.Regex("^➕ Добавить еще запись$"), add\_another\_entry),

                MessageHandler(filters.Regex("^ EЯ Выбрать другую группу$"), go\_to\_group\_selection),

                MessageHandler(filters.Regex("^📊 Вывести расписание дня$"), prompt\_export\_day),

                MessageHandler(filters.Regex("^✅ Завершить$"), done),

                MessageHandler(filters.TEXT & ~filters.COMMAND, lambda update, context: update.message.reply\_text("Используйте предложенные кнопки.")),

            ],

            EXPORT\_ASK\_DAY: [

                MessageHandler(filters.Regex("^⬅️ Назад$"), lambda update, context: POST\_SAVE\_OPTIONS),

                MessageHandler(filters.TEXT & ~filters.COMMAND, export\_day\_schedule),

            ]

        },

        fallbacks=[CommandHandler("cancel", cancel)],

    )

    application.add\_handler(conv\_handler)

    application.add\_handler(CommandHandler("start", start))

    logger.info("Запуск бота (v3)...")

    application.run\_polling()

    logger.info("Бот остановлен.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()