МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА САПР

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: Создание бота для Telegram

Студент гр. 4351;4354 Вихорева В.Е.
Нем А.

Преподаватель Кулагин М.В.

Санкт-Петербург
2025

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Нем А.
Студентка Вихорева В.Е.

Группы 4354, 4351

Тема работы: Создание бота для Telegram

Исходные данные:

Язык программирования: Python

Фреймворк: pyTelegramBotAPI (TeleBot)

Среда разработки: PyCharm / VS Code / любая среда с поддержкой Python

Используемые библиотеки: telebot (pyTelegramBotAPI), requests, datetime, typing

Содержание пояснительной записки:

Титульный лист, задание на курсовую работу, аннотация, содержание, введение, теоретическая часть, практическая реализация, тестирование, заключение, список использованных источников, приложения.

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 30.09.2025

Дата сдачи реферата: 22.12.2025

Дата защиты реферата: .12.2025

Студент Нем А.
Студентка Вихорева В.Е.

Преподаватель Кулагин М.В.

АННОТАЦИЯ

Курсовая работа посвящена созданию Telegram-бота для предоставления расписания занятий студентам СПбГЭТУ «ЛЭТИ». В ходе работы на языке Java с использованием фреймворка Spring Boot и библиотеки TelegramBots была разработана программа, которая интегрируется с официальным API университета для получения актуальных данных. Бот позволяет пользователям по номеру группы получать расписание на текущий день, неделю, завтрашний день, а также узнавать ближайшее занятие. Для удобного взаимодействия реализован интерфейс с reply-клавиатурой и inline-кнопками. Приложение было развернуто в локальной среде, протестировано на корректность работы с API и обработку пользовательских команд. В итоге был создан стабильно работающий бот, который студенты университета могут использовать для быстрого доступа к расписанию.

SUMMARY

The graduate thesis is dedicated to the development of a Telegram bot for providing class schedules to students of SPbETU "LETI". Within the scope of the work, a program was implemented in Java using the Spring Boot framework and the TelegramBots library. The bot integrates with the university's official API to obtain up-to-date schedules and provides the ability to request the next class, as well as schedules for a day, week, or the next day based on a specified group number. Interactive control elements were implemented: a reply keyboard and inline buttons for convenient interaction. The application was deployed in a local environment and tested for correct operation with the API and processing of user commands. As a result of the thesis work, a stable, functional bot was developed, which can be used by students of the university.

Оглавление

ВВЕДЕНИЕ

Цель работы: создание Telegram-бота, который автоматически предоставляет студентам СПбГЭТУ «ЛЭТИ» актуальное расписание занятий. Для этого бот будет подключен к официальным источникам данных университета и оснащен удобным интерфейсом для выполнения запросов.

Задачи:

1. Изучение требований и проектирование структуры бота.
2. Подключение к официальному API университета (digital.etu.ru) для доступа к актуальному расписанию.
3. Реализация функций для обработки запросов: нахождение ближайшего занятия, показ расписания на определённую дату, на завтра и на учебную неделю с учётом её чётности.
4. Создание удобного интерфейса с использованием меню-клавиатуры и интерактивных кнопок.
5. Проверка стабильности работы бота и точности обработки пользовательских команд.

Методы решения задач:

1. Построение модульной архитектуры бота с использованием фреймворка telebot.
2. Обработка и преобразование данных, получаемых от внешнего API университета в формате JSON.
3. Реализация алгоритмов для определения типа учебной недели (чётной/нечётной) как для текущей даты, так и для заданной пользователем.
4. Разработка алгоритма для автоматического поиска и предоставления информации о ближайшем по времени занятии.

1. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

1.1. Диаграмма вариантов использования

Для системы «Расписание ЛЭТИ» был определен один основной актор – Пользователь (студент ЛЭТИ). Система реализует следующие варианты использования (Use Cases):

- **Запуск бота:** Получение приветственного сообщения и отображение главного меню с командами.
- **Получение ближайшего занятия:** Запрос информации о следующей паре для указанной учебной группы.
- **Получение расписания на завтра:** Запрос расписания на следующий учебный день для выбранной группы.
- **Получение расписания на день:** Запрос расписания на конкретный день недели (от Понедельника до Воскресенья) с возможностью указания типа недели (чётная или нечётная).
- **Получение расписания на неделю:** Запрос полного расписания на учебную неделю с указанием её типа (чётная/нечётная).

Диаграмма вариантов использования представлена на рисунке 1.

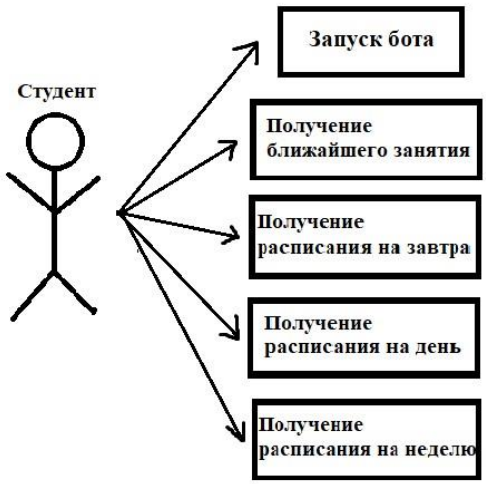

Рисунок 1 – Диаграмма вариантов использования Telegram-бота

1.2. Диаграмма классов объектной модели предметной области

Диаграмма классов на языке UML служит для визуализации объектно-ориентированной модели системы. Для проекта Telegram-бота «Расписание ЛЭТИ» была разработана следующая структура классов:

BotHandler — центральный класс-контроллер, который принимает и обрабатывает все сообщения от пользователей через Telegram API, управляет диалоговыми состояниями, отображает интерфейс с клавиатурами и кнопками, маршрутизирует команды к соответствующим сервисам и координирует общую логику работы приложения, обеспечивая связь между пользователем и бизнес-логикой.

ScheduleService — сервисный слой, отвечающий за всю бизнес-логику работы с расписанием: осуществляет интеграцию с внешним API университета для получения актуальных данных, реализует алгоритмы определения типа недели, поиска ближайших занятий и фильтрации по дням, обеспечивает кэширование данных для повышения производительности и отказоустойчивости, а также форматирует сырые данные в читаемые текстовые ответы для пользователя.

UserState — легковесный класс-модель, который хранит текущее состояние диалога для каждого отдельного пользователя, включая этап взаимодействия, выбранное действие, тип недели и день, обеспечивая поддержку многошаговых сценариев и персонализацию работы системы для каждого чата.

Взаимосвязи между классами:

1. Композиция (BotHandler → ScheduleService):
    - BotHandler создает и владеет экземпляром ScheduleService
    - Жизненный цикл сервиса привязан к контроллеру
2. Ассоциация (BotHandler → UserState):
    - Контроллер использует коллекцию объектов UserState
    - Каждый пользователь имеет свое уникальное состояние

3. Зависимость (BotHandler ← TeleBot):
    - Внешняя библиотека передается через зависимость
    - Обеспечивает возможность тестирования и замены компонентов

Диаграмма классов представлена на рисунке 2.

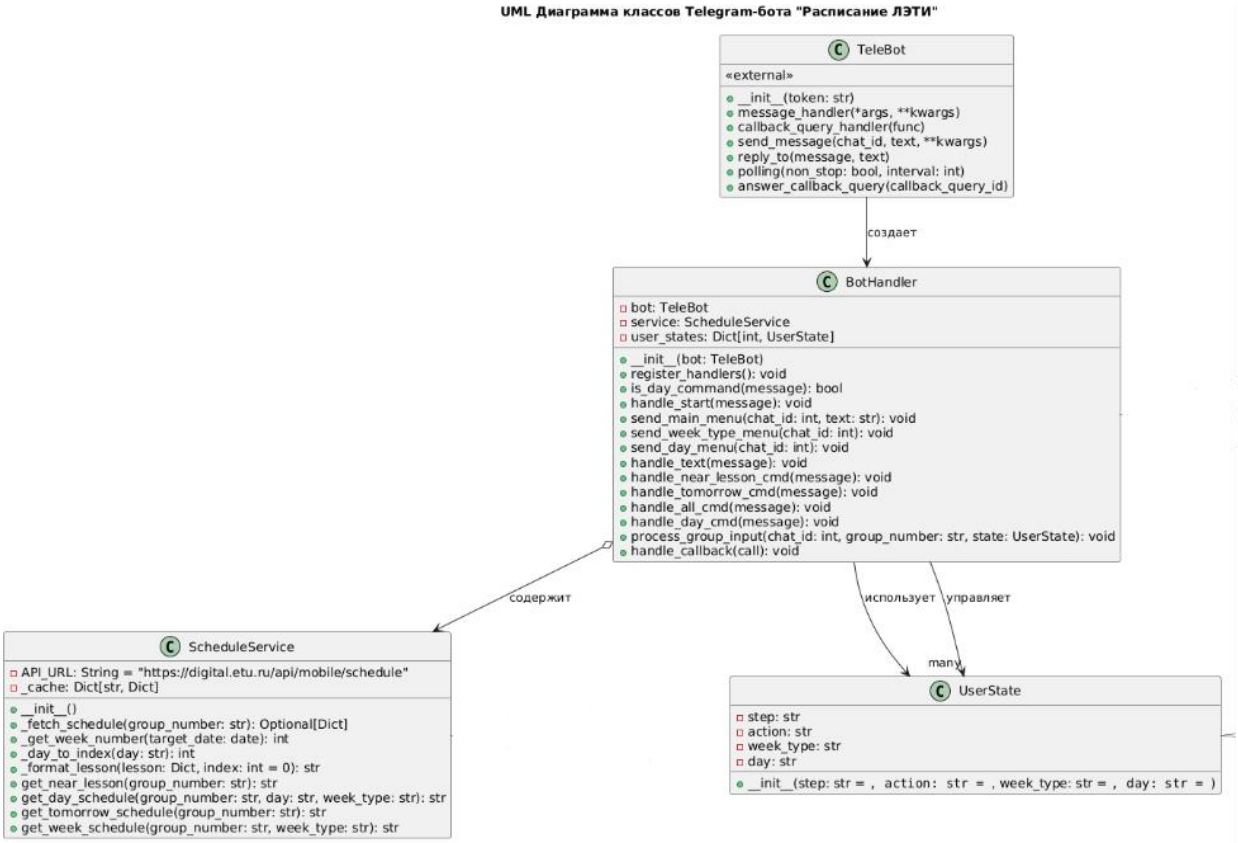

Рисунок 2 – Диаграмма классов

1.3. Спецификация классов

Класс ScheduleBot

Поле	Описание
API_URL	Базовый URL для API расписания университета: https://digital.etu.ru/api/mobile/schedule
_cache	Внутренний словарь для кэширования расписаний по номерам групп

Таблица 1 – Поля класса ScheduleBot

Метод	Описание
__init__()	Конструктор класса, инициализирует кэш расписаний
_fetch_schedule(group_number)	Получает расписание из API университета или кэша. Возвращает None при ошибке
_get_week_number(target_date)	Определяет тип недели по дате: 1 - нечётная, 2 - чётная
_day_to_index(day)	Преобразует название дня недели в индекс (0-6)
_format_lesson(lesson, index)	Форматирует информацию об одном занятии в читаемый текст с эмодзи
get_near_lesson(group_number)	Находит и возвращает ближайшее занятие для указанной группы
get_day_schedule(group_number, day, week_type)	Возвращает расписание на конкретный день для указанного типа недели
get_tomorrow_schedule(group_number)	Возвращает расписание на следующий учебный день
get_week_schedule(group_number, week_type)	Возвращает полное расписание на всю неделю указанного типа

Таблица 2 – Методы класса ScheduleBot

Класс UserState

Поле	Описание
step	Текущий этап диалога: "", "awaiting_group", "week_type", " day_selection"
action	Выбранное действие пользователя: "", "near_lesson", "tomorrow", " day", "week"
week_type	Тип учебной недели: "", "odd" (нечётная), "even" (чётная)
day	Выбранный день недели на английском (например: "monday", "tuesday")

Таблица 3 – Поля класса UserState

Метод	Описание
__init__(step, action, week_type, day)	Конструктор, инициализирует состояние пользователя с заданными параметрами

Таблица 4 – Методы класса UserState

Класс BotHandler

Поле	Описание
bot	Объект Telegram-бота из библиотеки pyTelegramBotAPI
service	Объект сервиса для работы с расписанием университета
user_states	Словарь, хранящий состояние каждого пользователя по ключу chat_id

Таблица 5 – Поля класса BotHandler

Метод	Описание
__init__(bot)	Конструктор, принимает объект Telegram-бота и создает сервис расписания
register_handlers()	Регистрирует все обработчики сообщений для различных команд
is_day_command(message)	Проверяет, является ли сообщение командой для получения расписания на день
handle_start(message)	Обрабатывает команду /start, сбрасывает состояние и показывает главное меню
send_main_menu(chat_id, text)	Отправляет главное меню с reply-клавиатурой основных действий
send_week_type_menu(chat_id)	Отправляет меню выбора типа недели (чётная/нечётная)
send_day_menu(chat_id)	Отправляет inline-клавиатуру для выбора дня недели
handle_text(message)	Основной обработчик текстовых сообщений, управляет диалогом по состояниям
handle_near_lesson_cmd(message)	Обрабатывает текстовую команду /near_lesson <номер_группы>
handle_tomorrow_cmd(message)	Обрабатывает текстовую команду /tommorow <номер_группы>
handle_all_cmd(message)	Обрабатывает текстовую команду /all <odd/even> <номер_группы>
handle_day_cmd(message)	Обрабатывает команду вида <день> <odd/even> <номер_группы>
process_group_input(chat_id, group_number, state)	Обрабатывает введенный номер группы на основе текущего состояния пользователя
handle_callback(call)	Обрабатывает нажатия на inline-кнопки (выбор дня недели)

Таблица 6 – Методы класса BotHandler

Функции в main.py

Функция	Описание
callback_handler(call)	Глобальный обработчик callback-запросов от inline-кнопок
Блок if __name__ == '__main__':	Точка входа программы, запускает бота в режиме polling

Таблица 7 – Функции в main

Переменная	Описание
TOKEN	Токен для аутентификации бота в Telegram API
bot	Основной объект бота, инициализированный с токеном
handler	Обработчик команд, связанный с объектом бота

Таблица 8 – Глобальные переменные в main

2. РЕАЛИЗАЦИЯ СИСТЕМЫ

2.1. Код классов объектной модели

Ниже представлены основные классы объектной модели Telegram-бота. Эти классы образуют ядро системы, обеспечивая получение расписания, обработку команд и взаимодействие с пользователем через интерфейс мессенджера.

Класс ScheduleService

```
import requests
from datetime import datetime, timedelta, date
from typing import Optional, Dict, Any, List

class ScheduleService:
    API_URL = "https://digital.etu.ru/api/mobile/schedule"

    def __init__(self):
        self._cache: Dict[str, Dict] = {}

    def _fetch_schedule(self, group_number: str) -> Optional[Dict[str, Any]]:
        # ... (полный метод)
        pass

    @staticmethod
    def _get_week_number(target_date: date) -> int:
        # ... (полный метод)
        pass

    @staticmethod
    def _day_to_index(day: str) -> int:
        # ... (полный метод)
        pass

    @staticmethod
    def _format_lesson(lesson: Dict[str, Any], index: int = 0) -> str:
        # ... (полный метод)
        pass

    def get_near_lesson(self, group_number: str) -> str:
        # ... (полный метод)
        pass
```

```
    def get_day_schedule(self, group_number: str, day: str, week_type: str) ->
str:
        # ... (полный метод)
        pass

    def get_tomorrow_schedule(self, group_number: str) -> str:
        # ... (полный метод)
        pass

    def get_week_schedule(self, group_number: str, week_type: str) -> str:
        # ... (полный метод)
        pass
```

Класс UserState

```
class UserState:
    def __init__(self, step: str = "", action: str = "", week_type: str = "", day: str
= ""):
        self.step = step          # awaiting_group, week_type, day_selection
        self.action = action      # near_lesson, tomorrow, day, week
        self.week_type = week_type
        self.day = day
```

Класс BotHandler

```
from telebot import TeleBot, types
from schedule_service import ScheduleService
from typing import Dict

class BotHandler:
    def __init__(self, bot: TeleBot):
        self.bot = bot
        self.service = ScheduleService()
        self.user_states: Dict[int, UserState] = {}
```

```
def register_handlers(self):
    # ... (полный метод)
    pass
def is_day_command(self, message):
    # ... (полный метод)
    pass
def handle_start(self, message):
    # ... (полный метод)
    pass
def send_main_menu(self, chat_id, text):
    # ... (полный метод)
    pass
def send_week_type_menu(self, chat_id):
    # ... (полный метод)
    pass
def send_day_menu(self, chat_id):
    # ... (полный метод)
    pass
def handle_text(self, message):
    # ... (полный метод)
    pass
def handle_near_lesson_cmd(self, message):
    # ... (полный метод)
    pass
def handle_tomorrow_cmd(self, message):
    # ... (полный метод)
    pass
def handle_all_cmd(self, message):
    # ... (полный метод)
    pass
```

```
def handle_day_cmd(self, message):
    # ... (полный метод)
    pass
def process_group_input(self, chat_id, group_number, state):
    # ... (полный метод)
    pass
def handle_callback(self, call):
    # ... (полный метод)
    pass
```

2.2. Описание интерфейса пользователя программы

На рисунке 3 виден стартовый экран бота после команды /start. Бот приветствует пользователя и предлагает ввести номер группы. Пользователь запросил расписание на завтра для группы 4351. Выводится время начала и конца пары, какой предмет, аудиторию и инициалы преподавателя.

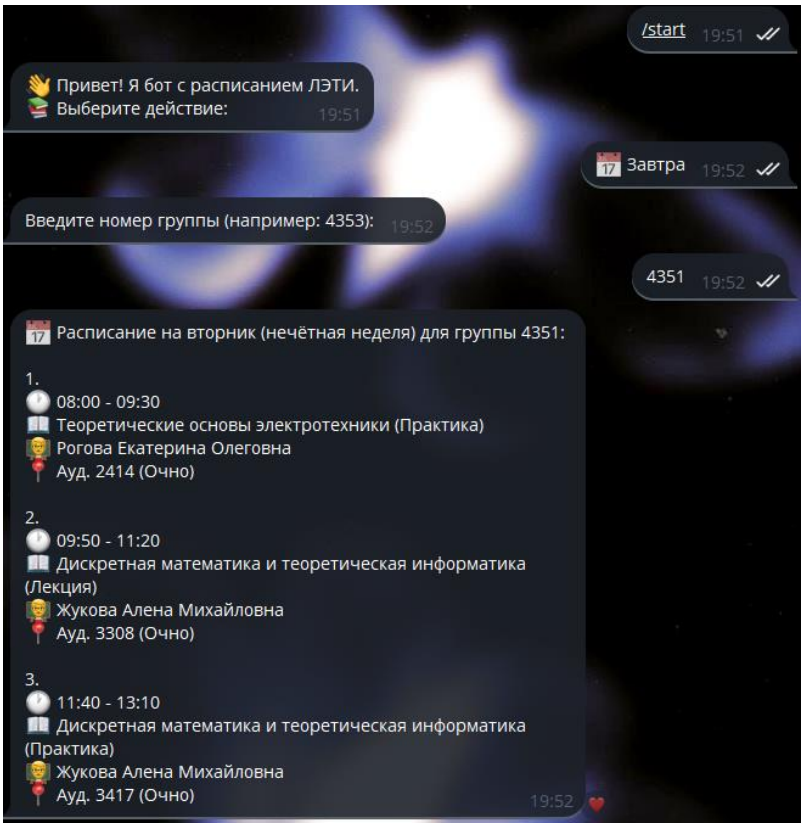

Рисунок 3. Пример работы программы

По запросу пользователя бот выводит расписание на ближайшую пару для группы 4351. Видно начало списка с занятиями, форматирование включает разделители между парами для разных типов информации (Рисунок 4).

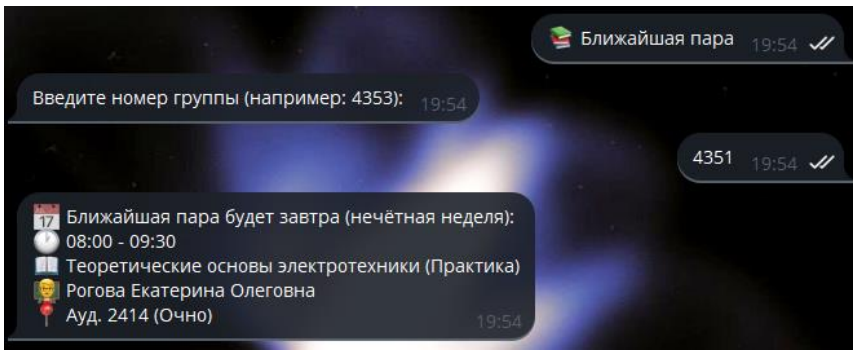

Рисунок 4. Пример работы программы

Пример интерфейса выбора дня недели через кнопки. Бот запрашивает тип недели (четная/нечетная) и день недели для просмотра расписания. Пользователь выбирает день из предложенных кнопок. После выбора четверг и ввода группы 4354 бот показывает расписание на этот день. Внизу в начале сообщения с расписанием пар отображаются текущие параметры выбора: тип недели и номер группы (Рисунок 5-7).

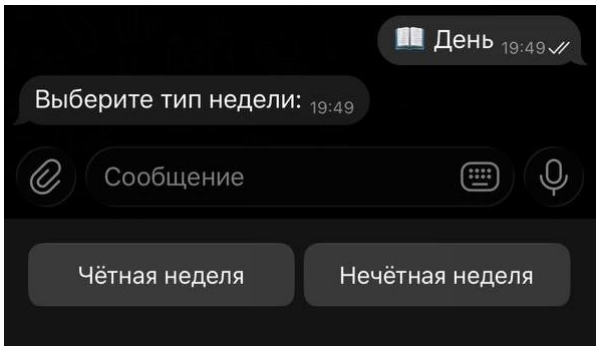

Рисунок 5. Пример работы программы

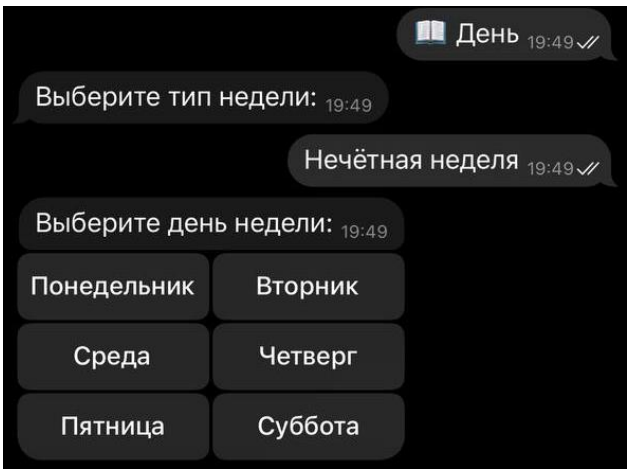

Рисунок 6. Пример работы программы

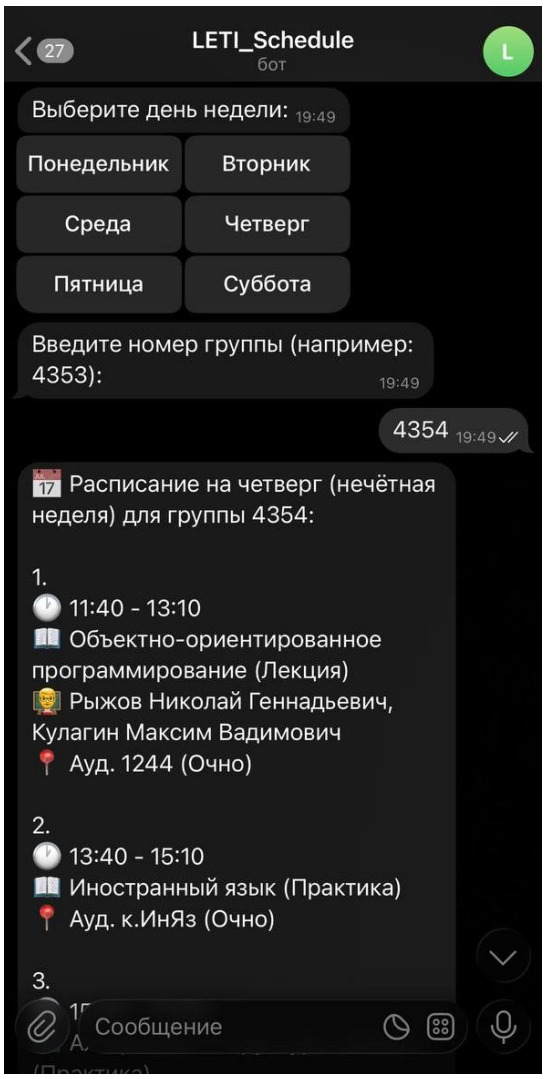

Рисунок 7. Пример работы программы

ЗАКЛЮЧЕНИЕ

Созданный Telegram-бот успешно решает поставленную задачу: он автоматически предоставляет студентам ЛЭТИ актуальное расписание занятий. Решение интегрировано с официальным API университета, что обеспечивает достоверность и оперативность получаемой информации. Все запланированные функции реализованы в полном объеме: пользователи могут запросить ближайшее занятие, расписание на конкретный день, на следующий день и на всю неделю с учетом четности. Интерфейс бота, основанный на reply-клавиатуре и inline-кнопках, удобен и интуитивно понятен, что упрощает взаимодействие с системой. Проведенное тестирование показало стабильную работу приложения, включая корректную обработку ошибочных и нестандартных запросов. Таким образом, разработано эффективное и надежное решение для быстрого получения учебного расписания.

Ссылка на репозиторий: https://github.com/1iper/OOP-kursach

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный API расписания ЛЭТИ // Портал цифровых сервисов СПбГЭТУ «ЛЭТИ». URL: https://digital.etu.ru/ (дата обращения: 19.12.2025).

2. Документация Telegram Bot API // Telegram API. URL: https://core.telegram.org/bots/api (дата обращения: 19.12.2025).

3. ГОСТ 7.32-2017-1. Межгосударственный стандарт. ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ. Структура и правила оформления.

4. TelegramBots: библиотека для создания ботов// GitHub. URL: https://github.com/rubenlagus/TelegramBots (дата обращения: 19.12.2025).

5. Требования к оформлению учебных работ в СПбГЭТУ «ЛЭТИ» // Официальный сайт университета. URL: https://etu.ru/ru/obrazovanie/trebovaniya-k-oformleniyu-rabot (дата обращения: 19.12.2025).

ПРИЛОЖЕНИЕ А

schedule_service.py

```
# bot_handler.py
from telebot import TeleBot, types
from schedule_service import ScheduleService
from typing import Dict

class UserState:
    def __init__(self, step: str = "", action: str = "", week_type: str = "",
day: str = ""):
        self.step = step          # awaiting_group, week_type, day_selection
        self.action = action      # near_lesson, tomorrow, day, week
        self.week_type = week_type
        self.day = day

class BotHandler:
    def __init__(self, bot: TeleBot):
        self.bot = bot
        self.service = ScheduleService()
        self.user_states: Dict[int, UserState] = {}

    def register_handlers(self):
        self.bot.message_handler(commands=['start'])(self.handle_start)

self.bot.message_handler(commands=['near_lesson'])(self.handle_near_lesso
n_cmd)

self.bot.message_handler(commands=['tommorow'])(self.handle_tomorrow_
cmd)

self.bot.message_handler(commands=['all'])(self.handle_all_cmd)

self.bot.message_handler(func=self.is_day_command)(self.handle_day_cmd
)
        self.bot.message_handler(func=lambda m:
True)(self.handle_text)

    def is_day_command(self, message):
        if not message.text:
            return False
        parts = message.text.split()
        if len(parts) < 3:
            return False
```

```
            day = parts[0].lower()
            return day in ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"]

        def handle_start(self, message):
            self.user_states.pop(message.chat.id, None)
            self.send_main_menu(message.chat.id, "👋 Привет! Я бот с расписанием ЛЭТИ.\n📚 Выберите действие:")

        def send_main_menu(self, chat_id, text):
            markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
            markup.row("📚 Ближайшая пара", "📅 Завтра")
            markup.row("📖 День", "🗓 Неделя")
            self.bot.send_message(chat_id, text, reply_markup=markup)

        def send_week_type_menu(self, chat_id):
            markup = types.ReplyKeyboardMarkup(resize_keyboard=True, one_time_keyboard=True)
            markup.row("Чётная неделя", "Нечётная неделя")
            self.bot.send_message(chat_id, "Выберите тип недели:", reply_markup=markup)

        def send_day_menu(self, chat_id):
            markup = types.InlineKeyboardMarkup(row_width=2)
            days = [
                ("Понедельник", "monday"),
                ("Вторник", "tuesday"),
                ("Среда", "wednesday"),
                ("Четверг", "thursday"),
                ("Пятница", "friday"),
                ("Суббота", "saturday")
            ]
            buttons = [types.InlineKeyboardButton(text=name, callback_data=f"day_{eng}") for name, eng in days]
            markup.add(*buttons)
            self.bot.send_message(chat_id, "Выберите день недели:", reply_markup=markup)

        def handle_text(self, message):
            chat_id = message.chat.id
            text = message.text.strip()

            state = self.user_states.get(chat_id)
```

```
        if state and state.step == "awaiting_group":
            self.process_group_input(chat_id, text, state)
            return

        if state and state.step == "week_type":
            if "нечётная" in text.lower():
                state.week_type = "odd"
                self.user_states[chat_id] = state
                if state.action == "day":
                    state.step = "day_selection"
                    self.send_day_menu(chat_id)
                elif state.action == "week":
                    state.step = "awaiting_group"
                    self.bot.send_message(chat_id, "Введите номер группы (например: 4353):")
            elif "чётная" in text.lower():
                state.week_type = "even"
                self.user_states[chat_id] = state
                if state.action == "day":
                    state.step = "day_selection"
                    self.send_day_menu(chat_id)
                elif state.action == "week":
                    state.step = "awaiting_group"
                    self.bot.send_message(chat_id, "Введите номер группы (например: 4353):")
            else:
                self.send_week_type_menu(chat_id)
            return

        # обработка кнопок главного меню
        if text == "📚 Ближайшая пара":
            self.user_states[chat_id] = UserState(step="awaiting_group", action="near_lesson")
            self.bot.send_message(chat_id, "Введите номер группы (например: 4353):")
        elif text == "📆 Завтра":
            self.user_states[chat_id] = UserState(step="awaiting_group", action="tomorrow")
            self.bot.send_message(chat_id, "Введите номер группы (например: 4353):")
        elif text == "📖 День":
            self.user_states[chat_id] = UserState(step="week_type", action="day")
```

```
                self.send_week_type_menu(chat_id)
            elif text == "📅 Неделя":
                self.user_states[chat_id] = UserState(step="week_type",
action="week")
                self.send_week_type_menu(chat_id)
            else:
                self.send_main_menu(chat_id, "❌ Неизвестная команда.
Выберите действие:")

        def handle_near_lesson_cmd(self, message):
            parts = message.text.split()
            if len(parts) < 2:
                self.bot.reply_to(message, "Использование: /near_lesson
<номер_группы>")
                return
            group = parts[1]
            response = self.service.get_near_lesson(group)
            self.bot.send_message(message.chat.id, response)

        def handle_tomorrow_cmd(self, message):
            parts = message.text.split()
            if len(parts) < 2:
                self.bot.reply_to(message, "Использование: /tommorow
<номер_группы>")
                return
            group = parts[1]
            response = self.service.get_tomorrow_schedule(group)
            self.bot.send_message(message.chat.id, response)

        def handle_all_cmd(self, message):
            parts = message.text.split()
            if len(parts) < 3:
                self.bot.reply_to(message, "Использование: /all <odd|even>
<номер_группы>")
                return
            week_type = parts[1].lower()
            group = parts[2]
            if week_type not in ("odd", "even"):
                self.bot.reply_to(message, "Неделя должна быть 'odd' или
'even'")
                return
            response = self.service.get_week_schedule(group, week_type)
            self.bot.send_message(message.chat.id, response)
```

```
        def handle_day_cmd(self, message):
            parts = message.text.split()
            day = parts[0].lower()
            week_type = parts[1].lower()
            group = parts[2]
            if week_type not in ("odd", "even"):
                self.bot.reply_to(message, "Неделя должна быть 'odd' или 'even'")
                return
            response = self.service.get_day_schedule(group, day, week_type)
            self.bot.send_message(message.chat.id, response)

        def process_group_input(self, chat_id, group_number, state):
            try:
                if state.action == "near_lesson":
                    resp = self.service.get_near_lesson(group_number)
                elif state.action == "tomorrow":
                    resp = self.service.get_tomorrow_schedule(group_number)
                elif state.action == "day":
                    resp = self.service.get_day_schedule(group_number, state.day, state.week_type)
                elif state.action == "week":
                    resp = self.service.get_week_schedule(group_number, state.week_type)
                else:
                    resp = "❌ Неизвестное действие."
            except Exception as e:
                resp = f"⚠️ Ошибка: {e}"
            self.user_states.pop(chat_id, None)
            self.send_main_menu(chat_id, resp)

        def handle_callback(self, call):
            chat_id = call.message.chat.id
            data = call.data
            if data.startswith("day_"):
                day = data[4:]
                state = self.user_states.get(chat_id)
                if state and state.step == "day_selection":
                    state.day = day
                    state.step = "awaiting_group"
                    self.bot.send_message(chat_id, "Введите номер группы (например: 4353):")
            self.bot.answer_callback_query(call.id)
```

ПРИЛОЖЕНИЕ Б

bot_handler.py

```
# schedule_service.py
import requests
from datetime import datetime, timedelta, date
from typing import Optional, Dict, Any, List

class ScheduleService:
    API_URL = "https://digital.etu.ru/api/mobile/schedule"

    def __init__(self):
        self._cache: Dict[str, Dict] = {}

    def _fetch_schedule(self, group_number: str) -> Optional[Dict[str, Any]]:
        if group_number in self._cache:
            return self._cache[group_number]

        try:
            response = requests.get(f"{self.API_URL}?groupNumber={group_number}", timeout=10)
            response.raise_for_status()
            data = response.json()
            if group_number not in data:
                return None
            schedule = data[group_number]
            self._cache[group_number] = schedule
            return schedule
        except Exception as e:
            print(f"Ошибка при получении расписания: {e}")
            return None

    @staticmethod
    def _get_week_number(target_date: date) -> int:
        """Возвращает 1 для нечётной, 2 для чётной недели."""
        if target_date.month >= 9:
            start_year = target_date.year
        else:
            start_year = target_date.year - 1
        start_academic = date(start_year, 9, 1)
        days = (target_date - start_academic).days
        weeks = days // 7
```

```
            return 1 if weeks % 2 == 0 else 2

        @staticmethod
        def _day_to_index(day: str) -> int:
            days = {
                "monday": 0, "tuesday": 1, "wednesday": 2,
                "thursday": 3, "friday": 4, "saturday": 5, "sunday": 6
            }
            return days.get(day.lower(), -1)

        @staticmethod
        def _format_lesson(lesson: Dict[str, Any], index: int = 0) -> str:
            def ru_type(t: str) -> str:
                return {"Лек": "Лекция", "Пр": "Практика", "Лаб": "Лабораторная"}.get(t, t)

            def ru_form(f: str) -> str:
                return {"standard": "Очно", "online": "Онлайн", "distant": "Дистанционно"}.get(f, f)

            lines = []
            if index > 0:
                lines.append(f"{index}.")
            lines.append(f"🕐 {lesson.get('start_time', '')} - {lesson.get('end_time', '')}")
            name = lesson.get("name", "")
            subj_type = lesson.get("subjectType")
            if subj_type:
                name += f" ({ru_type(subj_type)})"
            lines.append(f"📖 {name}")
            teachers = [t for t in [lesson.get("teacher"), lesson.get("second_teacher")] if t]
            if teachers:
                lines.append(f"👨‍🏫 {', '.join(teachers)}")
            room = lesson.get("room")
            form = lesson.get("form")
            loc = ""
            if room:
                loc = f"📍 Ауд. {room}"
                if form:
                    loc += f" ({ru_form(form)})"
            elif form:
                loc = f"📍 {ru_form(form)}"
```

```
            if loc:
                lines.append(loc)
            url = lesson.get("url")
            if url:
                lines.append(f"📎 {url}")
            return "\n".join(lines)

        def get_near_lesson(self, group_number: str) -> str:
            schedule = self._fetch_schedule(group_number)
            if not schedule:
                return f"❌ Группа {group_number} не найдена."

            now = datetime.now()
            today = now.date()
            current_time = now.time()
            today_index = today.weekday()
            week_num = self._get_week_number(today)

            day_data = schedule["days"].get(str(today_index))
            if day_data and day_data.get("lessons"):
                today_lessons = [l for l in day_data["lessons"] if l.get("week")
== str(week_num)]
                for lesson in today_lessons:
                    start = datetime.strptime(lesson["start_time"],
"%H:%M").time()
                    end = datetime.strptime(lesson["end_time"],
"%H:%M").time()
                    if start <= current_time <= end:
                        return "Сейчас идёт:\n" + self._format_lesson(lesson)
                for lesson in today_lessons:
                    start = datetime.strptime(lesson["start_time"],
"%H:%M").time()
                    if start > current_time:
                        return "Ближайшая пара сегодня:\n" +
self._format_lesson(lesson)

            for i in range(1, 8):
                future_date = today + timedelta(days=i)
                if future_date.weekday() == 6:
                    continue
                w_num = self._get_week_number(future_date)
                idx = future_date.weekday()
                day_schedule = schedule["days"].get(str(idx))
                if not day_schedule or not day_schedule.get("lessons"):
```

```
                continue
            for lesson in day_schedule["lessons"]:
                if lesson.get("week") == str(w_num):
                    day_names = ["понедельник", "вторник", "среда", "четверг", "пятница", "суббота"]
                    prefix = "завтра" if i == 1 else day_names[idx]
                    return f"📅 Ближайшая пара будет {prefix} ({'нечётная' if w_num == 1 else 'чётная'} неделя):\n" + self._format_lesson(lesson)

        return "📭 Ближайших пар не найдено."

    def get_day_schedule(self, group_number: str, day: str, week_type: str) -> str:
        schedule = self._fetch_schedule(group_number)
        if not schedule:
            return f"❌ Группа {group_number} не найдена."

        day_idx = self._day_to_index(day)
        if day_idx == -1:
            return "❌ Неверный день недели."

        target_week = "1" if week_type == "odd" else "2"
        day_data = schedule["days"].get(str(day_idx))
        if not day_data or not day_data.get("lessons"):
            return f"📭 В этот день пар нет для группы {group_number}."

        lessons = [l for l in day_data["lessons"] if l.get("week") == target_week]
        if not lessons:
            return f"📭 На {'нечётную' if target_week == '1' else 'чётную'} неделю в этот день пар нет."

        day_names = ["понедельник", "вторник", "среда", "четверг", "пятница", "суббота", "воскресенье"]
        result = f"📅 Расписание на {day_names[day_idx]} ({'нечётная' if target_week == '1' else 'чётная'} неделя) для группы {group_number}:\n\n"
        for i, lesson in enumerate(lessons, 1):
            result += self._format_lesson(lesson, i) + "\n\n"
        return result.strip()

    def get_tomorrow_schedule(self, group_number: str) -> str:
```

```
            tomorrow = datetime.now().date() + timedelta(days=1)
            if tomorrow.weekday() == 6:
                tomorrow += timedelta(days=1)
            w_num = self._get_week_number(tomorrow)
            day_idx = tomorrow.weekday()
            day_names = ["monday", "tuesday", "wednesday", "thursday",
"friday", "saturday", "sunday"]
            return                          self.get_day_schedule(group_number,
day_names[day_idx], "odd" if w_num == 1 else "even")

        def get_week_schedule(self, group_number: str, week_type: str) ->
str:
            schedule = self._fetch_schedule(group_number)
            if not schedule:
                return f"❌ Группа {group_number} не найдена."

            target_week = "1" if week_type == "odd" else "2"
            day_names = ["понедельник", "вторник", "среда", "четверг",
"пятница", "суббота"]
            result = f"📅 Расписание на {'нечётную' if target_week == '1'
else 'чётную'} неделю для группы {group_number}:\n\n"
            has_any = False

            for i in range(6):
                day_data = schedule["days"].get(str(i))
                if not day_data or not day_data.get("lessons"):
                    continue
                lessons = [l for l in day_data["lessons"] if l.get("week") ==
target_week]
                if not lessons:
                    continue
                has_any = True
                result += f"--- {day_names[i].capitalize()} ---\n"
                for j, lesson in enumerate(lessons, 1):
                    result += self._format_lesson(lesson, j) + "\n\n"

            if not has_any:
                return f"📭 На {'нечётной' if target_week == '1' else 'чётной'}
неделе пар нет для группы {group_number}."
            return result.strip()
```

ПРИЛОЖЕНИЕ В

main.py

```
# main.py
import telebot
from bot_handler import BotHandler

TOKEN = '8490919762:AAFH6CBEy6xE160WWiQ6UjegB_RZBzxeYXM'  # ← замените на свой

bot = telebot.TeleBot(TOKEN)
handler = BotHandler(bot)
handler.register_handlers()

@bot.callback_query_handler(func=lambda call: True)
def callback_handler(call):
    handler.handle_callback(call)

if __name__ == '__main__':
    print("Бот запущен...")
    bot.polling(non_stop=True, interval=0)
```