**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования «Казанский национальный исследовательский

технический университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИАЭП\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование института (факультета), филиала)

\_\_\_\_\_\_\_\_\_Автоматики и управления\_\_\_\_\_\_\_\_

(наименование кафедры)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_27.03.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр и наименование направления подготовки (специальности))

ОТЧЕТ

на тему «

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Телеграмм бот КАИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

ОБУЧАЮЩИЙСЯ Шарафиев Т.Э. 3239

*(инициалы, фамилия) (личная подпись)*

РУКОВОДИТЕЛЬ \_\_\_\_\_\_Маликов А.И.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(ученая степень, звание, инициалы, фамилия) (личная подпись)*

Казань 2024

\_\_\_

Оглавление

1. Введение

2. Описание файлов

2.1. Файл `main.py`

2.2. Файл `weather.py`

2.3. Файл `m\_schedule.py`

2.4. Файл `generate\_table.py`

3. Работа с базой данных

4. Исходный код

5. Заключение

Введение

Этот проект представляет собой Telegram-бота для студентов, который предоставляет информацию о погоде и расписании занятий. Проект состоит из нескольких файлов на языке Python, каждый из которых отвечает за определенную функциональность.

Описание файлов

2.1. Файл `main.py`

Основной файл проекта, который инициализирует бота и обрабатывает команды и сообщения от пользователей.

Функции:

1. update\_bot()

- Описание: Функция для обновления бота и отправки уведомлений пользователям.

- Аргументы: нет.

- Возвращаемое значение: нет.

2. daily\_schedule()

- Описание: Заглушка для функции, которая может быть использована для расписания задач.

- Аргументы: нет.

- Возвращаемое значение: нет.

3. url(message)

- Описание: Обработчик команды `/start`. Отправляет приветственное сообщение и запрашивает номер учебной группы.

- Аргументы: message - сообщение от пользователя.

- Возвращаемое значение: нет.

4. get\_text\_messages(message)

- Описание: Обработчик текстовых сообщений. Реагирует на команды "Погода", "Расписание" и "Изменить номер группы".

- Аргументы: message - сообщение от пользователя.

- Возвращаемое значение: нет.

5. get\_variable\_num\_group(message)

- Описание: Обработчик ввода номера группы, если ранее он не был указан.

- Аргументы: message - сообщение от пользователя.

- Возвращаемое значение: нет.

6. get\_num\_group(message)

- Описание: Обработчик ввода номера группы при регистрации.

- Аргументы: message - сообщение от пользователя.

- Возвращаемое значение: нет.

7. callback\_inline(call)

- Описание: Обработчик inline-кнопок для выбора дня недели.

- Аргументы: call - вызов от inline-кнопки.

- Возвращаемое значение: нет.

8. call\_table\_without\_bd(call, group)

- Описание: Вспомогательная функция для отображения таблицы расписания без изменения БД.

- Аргументы: call - вызов от inline-кнопки, group - номер группы.

- Возвращаемое значение: нет.

2.2. Файл `weather.py`

Файл содержит класс для получения и обработки данных о погоде.

Классы и функции:

1. Класс my\_weather

- Описание: Класс для получения и обработки данных о погоде.

Методы:

- \_\_init\_\_(self, city\_name="kazan", cords\_city={"lat": 55.7887, "lon": 49.1221})

- Описание: Инициализирует объект с названием города и его координатами.

- Аргументы: city\_name - название города, cords\_city - координаты города.

- Возвращаемое значение: нет.

- weather\_description(self)

- Описание: Возвращает описание погоды в виде словаря.

- Аргументы: нет.

- Возвращаемое значение: словарь с описанием погоды.

- get\_weather\_by\_cords(self, lang="ru")

- Описание: Получает данные о погоде по координатам города.

- Аргументы: lang - язык данных (по умолчанию русский).

- Возвращаемое значение: JSON-ответ от API.

- get\_weather\_by\_name(self, lang="ru")

- Описание: Получает данные о погоде по названию города.

- Аргументы: lang - язык данных (по умолчанию русский).

- Возвращаемое значение: JSON-ответ от API.

- gent(self, word, gent)

- Описание: Склоняет название города в нужный падеж.

- Аргументы: word - слово для склонения, gent - нужный падеж.

- Возвращаемое значение: склоненное слово.

- formated\_print(self)

- Описание: Форматирует и возвращает данные о погоде.

- Аргументы: нет.

- Возвращаемое значение: строка с форматированными данными о погоде.

2.3. Файл m\_schedule.py

Файл для работы с расписанием занятий.

Функции:

1. \_schedule

- Описание: Функция для работы с расписанием занятий.

- Аргументы: номер группы.

- Возвращаемое значение: данные о расписании.

def convert\_schedule(self): # конвертируем расписание в свой формат

schedule = self.get\_schedule()

converted\_schedule = {}

for i in [1, 2, 3, 4, 5, 6]:

converted\_schedule[self.days\_in\_week[i]] = []

for \_class in schedule[f"{i}"]:

converted\_schedule[self.days\_in\_week[i]].append(

{"Время": " ".join(\_class["dayTime"].split()), "Дата": " ".join(\_class["dayDate"].split()),

"Дисциплина": " ".join(\_class["disciplName"].split()),

"Вид занятия": " ".join(\_class["disciplType"].split()),

"Аудитория": " ".join(\_class["audNum"].split()), "Здание": " ".join(\_class["buildNum"].split()),

"Преподаватель": " ".join(\_class["prepodName"].split()),

"Кафедра": " ".join(\_class["orgUnitName"].split())})

return converted\_schedule

2.4. Файл generate\_table.py

Файл для генерации таблицы расписания занятий.

Функции:

1. schedule\_day\_img

- Описание: Класс для создания изображения расписания занятий.

- Аргументы: день недели, номер группы.

- Возвращаемое значение: изображение расписания.

2.5. Файл generate\_table.py

Файл содержит классы и функции для генерации изображений с расписанием занятий.

Функции:

1. class schedule\_day\_img

- Описание: Класс для создания изображения с расписанием занятий на определенный день.

Методы:

- \_\_init\_\_(self, day, group)

- Описание: Инициализирует объект с днем недели и номером группы.

- Аргументы: day - день недели, group - номер группы.

- Возвращаемое значение: нет.

- create\_table\_schedule(self)

- Описание: Создает изображение с расписанием занятий.

- Аргументы: нет.

- Возвращаемое значение: изображение с расписанием занятий.

3. Работа с базой данных

Проект использует базу данных SQLite для хранения информации о пользователях и их учебных группах. Работа с базой данных осуществляется через модуль `sqlite3`.

\*\*Структура базы данных\*\*

База данных содержит одну таблицу `user\_group`, в которой хранятся следующие данные:

- `user\_id`: уникальный идентификатор пользователя Telegram.

- `num\_group`: номер учебной группы пользователя.

- `group\_status`: статус группы пользователя, который может быть `stable` или `variable`.

**Операции с базой данных**

1. Получение списка пользователей для обновления бота

python

Копировать код

def update\_bot():

a = cursor.execute("""SELECT user\_id FROM user\_group""").fetchall()

user\_id\_list = list(map(lambda x: x[0], a))

for id in user\_id\_list:

bot.send\_message(id, "Бот обновился!\n"

"Пожалуйста, напишите '/start'")

1. Проверка наличия пользователя в базе данных

python

Копировать код

info = cursor.execute('SELECT \* FROM user\_group WHERE user\_id=?', (message.from\_user.id,))

if info.fetchone() is None:

bot.send\_message(message.from\_user.id,

"Привет!\n"

"Для персонализированной помощи мне нужен номер твоей учебной группы в КАИ.\n"

"Пожалуйста, отправь мне свой номер группы, чтобы я мог предоставить тебе наилучшую поддержку и актуальную информацию.\n"

"Спасибо! 🎓", reply\_markup=markup\_start)

bot.register\_next\_step\_handler(message, get\_num\_group)

else:

bot.send\_message(message.from\_user.id, "И снова привет!")

1. Обновление номера группы

python

Копировать код

def get\_variable\_num\_group(message: types.Message):

if \_schedule(message.text).is\_group\_schedule():

cursor.execute("""UPDATE user\_group SET num\_group = ? WHERE user\_id = ?""",

(message.text, message.from\_user.id))

con.commit()

...

1. Добавление новой записи или обновление существующей

python

Копировать код

def get\_num\_group(message: types.Message):

...

if not \_schedule(message.text).get\_groupid():

bot.send\_message(message.from\_user.id,

"⚠️ Ой! Кажется, произошла ошибка. Пожалуйста, убедитесь, что введенный номер учебной группы"

" корректен, и попробуйте еще раз.")

...

else:

bot.send\_message(message.from\_user.id,

"🌟 Отлично! Спасибо за предоставленную информацию. Твоя учебная группа успешно записана.")

info = cursor.execute('SELECT \* FROM user\_group WHERE user\_id=?', (message.from\_user.id,))

if info.fetchone() is None:

inf = (message.from\_user.id, message.text, "stable")

cursor.execute("INSERT INTO user\_group (user\_id, num\_group, group\_status) VALUES (?, ?, ?)", inf)

con.commit()

else:

cursor.execute("""UPDATE user\_group SET num\_group = ? WHERE user\_id = ?""",

(message.text, message.from\_user.id))

cursor.execute("""UPDATE user\_group SET group\_status = ? WHERE user\_id = ?""",

("stable", message.from\_user.id))

Weather.py

import pymorphy2

import requests

class my\_weather:

# Конструктор класса, который инициализирует начальные параметры

def \_\_init\_\_(self, city\_name="kazan", cords\_city={"lat": 55.7887, "lon": 49.1221}):

self.openweathermap\_token = "afc12bf08d7666e923e8349647fa72a1" # Токен для доступа к API OpenWeatherMap

self.cords\_city = cords\_city # Координаты города

self.city\_name = city\_name # Название города

# Метод для получения описания погоды в виде словаря

def weather\_description(self):

weather = self.get\_weather\_by\_name() # Получение данных о погоде по названию города

return {

"description": weather["weather"][0]["description"].capitalize(), # Описание погоды

"temp": round(weather["main"]["temp"]), # Температура

"feels\_like": round(weather["main"]["feels\_like"]), # Ощущаемая температура

"city": weather["name"] # Название города

}

# Метод для получения данных о погоде по координатам города

def get\_weather\_by\_cords(self, lang="ru"):

lat = self.cords\_city["lat"]

lon = self.cords\_city["lon"]

req\_api = requests.get(

f"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={self.openweathermap\_token}&lang={lang}"

)

return req\_api.json() # Возвращает JSON ответ

# Метод для получения данных о погоде по названию города

def get\_weather\_by\_name(self, lang="ru"):

req\_api = requests.get(

"http://api.openweathermap.org/data/2.5/weather",

params={'q': f"{self.city\_name}", 'units': 'metric', 'lang': 'ru', 'APPID': self.openweathermap\_token}

)

return req\_api.json() # Возвращает JSON ответ

# Метод для склонения названия города в нужный падеж

def gent(self, word, gent):

morph = pymorphy2.MorphAnalyzer()

butyavka = morph.parse(f'{word}')[0]

gent = butyavka.inflect({f'{gent}'})

return gent.word.capitalize() # Возвращает слово в нужном падеже с заглавной буквы

# Метод для форматированного вывода данных о погоде

def formated\_print(self):

weather\_description = self.weather\_description()

return f'Погода в {self.gent(weather\_description["city"], "loct")}:\n' \

f'{weather\_description["description"]}\n' \

f'Температура: {weather\_description["temp"]}°C, ощущается как: {weather\_description["feels\_like"]}°C'

# Тестирование класса my\_weather

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

weather = my\_weather()

print(weather.formated\_print())

generate\_table.py

from PIL import Image, ImageDraw, ImageFont

import textwrap

from m\_schedule import \_schedule

class schedule\_day\_img:

def \_\_init\_\_(self, day=1, group="3239", my\_schedule=[]):

self.schedule\_ = \_schedule(group)

if not my\_schedule:

self.my\_schedule = self.schedule\_.convert\_schedule\_by\_day(day)

else:

self.my\_schedule = my\_schedule

def get\_size(self, text,

font=ImageFont.truetype("assets/Inter/Inter-Regular.ttf", 14)): # возвращает размера текста в пикселях

left, top, right, bottom = font.getbbox(f"{text}")

width = right - left

height = bottom - top

return list(map(lambda x: x + 20, [width, 20]))

def get\_line\_break(self, text, syms): # функция переноса строки

r = ""

for line in textwrap.wrap(text, width=syms):

r += line + "\n"

return r

def get\_vid(self, text):

d = {"лек": "Лекция", "л.р.": "Лабораторная работа", "пр": "Практика"}

return d[text]

def create\_cell(self, \_class): # генерация блока фото одной пары

cell = Image.new('RGB', (1100, 300), (21, 38, 52))

draw = ImageDraw.Draw(cell)

font = ImageFont.truetype("assets/Inter/Inter-Regular.ttf", size=94)

text\_time = \_class["Время"]

textsize = draw.textbbox((0, 0), text\_time, font=font)

textarea = (240, 70)

draw.text((40 + (textarea[0] - textsize[2]) / 2, 70 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_time, (0, 135, 200), font=font, align='center', anchor='lt')

font = ImageFont.truetype("assets/Inter/Inter-Regular.ttf", size=34)

text\_bild = f"Здание: {\_class['Здание']}"

if "ОЛИМП" in text\_bild:

text\_bild = "Здание: ОЛИМП"

text\_bild = text\_bild

textsize = draw.textbbox((0, 0), text\_bild, font=font)

textarea = (175, 35)

draw.text((70 + (textarea[0] - textsize[2]) / 2, 165 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_bild, (240, 240, 240), font=font, align='center', anchor='lt')

text\_aud = f"Ауд: {\_class['Аудитория']}"

if "ОЛИМП" in text\_aud:

text\_aud = "-----"

text\_aud = text\_aud

textsize = draw.textbbox((0, 0), text\_aud, font=font)

textarea = (137, 35)

draw.text((90 + (textarea[0] - textsize[2]) / 2, 205 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_aud, (240, 240, 240), font=font, align='center', anchor='lt')

font = ImageFont.truetype("assets/Inter/Inter-Regular.ttf", size=36)

text\_discip = f"{\_class['Дисциплина']}"

text\_discip = self.get\_line\_break(text\_discip, 33)

textsize = draw.textbbox((0, 0), text\_discip, font=font)

textarea = (670, 85)

draw.text((345, 45 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_discip, (0, 135, 200), font=font, align='left')

font = ImageFont.truetype("assets/Inter/Inter-Bold.ttf", size=36)

text\_vid = f"{\_class['Вид занятия']}"

text\_vid = self.get\_vid(text\_vid)

textsize = draw.textbbox((0, 0), text\_vid, font=font)

textarea = (695, 35)

draw.text((345, 145 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_vid, (240, 240, 240), font=font, align='left', anchor='lt')

font = ImageFont.truetype("assets/Inter/Inter-Bold.ttf", size=34)

text\_prepod = f"{\_class['Преподаватель']}"

textsize = draw.textbbox((0, 0), text\_prepod, font=font)

textarea = (695, 35)

draw.text((345, 185 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_prepod, (240, 240, 240), font=font, align='left', anchor='lt')

font = ImageFont.truetype("assets/Inter/Inter-Regular.ttf", size=36)

text\_date = f"{\_class['Дата']}"

if len(text\_date) > 30:

font = ImageFont.truetype("assets/Inter/Inter-Regular.ttf", size=24)

textsize = draw.textbbox((0, 0), text\_date, font=font)

textarea = (695, 35)

draw.text((345, 225 + (textarea[1] - textsize[3] + textsize[1]) / 2),

text\_date, (0, 135, 200), font=font, align='left', anchor='lt')

cell.save("test\_cell.png")

return cell

def create\_table\_schedule(self): # формирует фото расписания из блоков фото

table\_size = [1120, len(self.my\_schedule) \* 330]

table = Image.new('RGB', table\_size, (14, 26, 38))

k = 0

for \_class in self.my\_schedule:

cell = self.create\_cell(\_class)

table.paste(cell, (10, 15 + k \* 330))

k += 1

return table

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

sch = \_schedule().get\_schedule\_today()

table = schedule\_day\_img(day=1, group=3238, my\_schedule=sch)

table.create\_table\_schedule().show()

m\_schedule.py

import requests

import json

from tabulate import tabulate

from datetime import datetime

class \_schedule:

def \_\_init\_\_(self, group="3239"):

self.group = group

self.url\_group = f"https://kai.ru/raspisanie?p\_p\_id=pubStudentSchedule\_WAR\_publicStudentSchedule10&p\_p\_lifecycle=2&p\_p\_state=normal&p\_p\_mode=view&p\_p\_resource\_id=getGroupsURL&p\_p\_cacheability=cacheLevelPage&p\_p\_col\_id=column-1&p\_p\_col\_count=1&query={self.group}"

self.url\_rasp = "https://kai.ru/raspisanie?p\_p\_id=pubStudentSchedule\_WAR\_publicStudentSchedule10&p\_p\_lifecycle=2&p\_p\_state=normal&p\_p\_mode=view&p\_p\_resource\_id=schedule&p\_p\_cacheability=cacheLevelPage&p\_p\_col\_id=column-1&p\_p\_col\_count=1"

self.days\_in\_week = {1: "Понедельник", 2: "Вторник", 3: "Среда", 4: "Четверг", 5: "Пятница", 6: "Суббота",

7: "Воскресенье"}

self.days\_in\_week\_num = {"Понедельник": 1, "Вторник": 2, "Среда": 3, "Четверг": 4, "Пятница": 5, "Суббота": 6,

"Воскресенье": 7}

def get\_groupid(self): # получаем id группы с сервера kai

return requests.get(self.url\_group).json()

def get\_schedule(self): # получаем расписание с сервера kai

return requests.post(self.url\_rasp, params=self.from\_r\_group\_to\_rasp()).json()

def from\_r\_group\_to\_rasp(self): # получаем расписание с сервера kai

r\_group = self.get\_groupid()

return {"groupId": r\_group[0]["id"], "programForm": r\_group[0]["forma"]}

def unzip\_schedule(self): # распаковывываем расписание

schedule = self.get\_schedule()

res = ""

rasp = [["Время", "Дата", "Дисциплина", "Вид занятия", "Аудитория", "Здание", "Преподаватель", "Кафедра"]]

for i in [1, 2, 3, 4, 5, 6]:

res += f"{self.days\_in\_week[i]}\n"

for \_class in schedule[f"{i}"]:

rasp.append([\_class["dayTime"], \_class["dayDate"], \_class["disciplName"], \_class["disciplType"],

\_class["audNum"],

\_class["buildNum"], \_class["prepodName"], \_class["orgUnitName"]])

res += str(tabulate(rasp)) + "\n"

rasp = [["Время", "Дата", "Дисциплина", "Вид занятия", "Аудитория", "Здание", "Преподаватель", "Кафедра"]]

return res

def unzip\_schedule\_by\_day(self, day="Понедельник"): # распаковывываем расписание в определенный день

schedule = self.get\_schedule()

rasp = [["Время", "Дата", "Дисциплина", "Вид занятия", "Аудитория", "Здание", "Преподаватель", "Кафедра"]]

for \_class in schedule[f'{self.days\_in\_week\_num[day]}']:

rasp.append([\_class["dayTime"], \_class["dayDate"], \_class["disciplName"], \_class["disciplType"],

\_class["audNum"],

\_class["buildNum"], \_class["prepodName"], \_class["orgUnitName"]])

return tabulate(rasp)

def convert\_schedule(self): # конвертируем расписание в свой формат

schedule = self.get\_schedule()

converted\_schedule = {}

for i in [1, 2, 3, 4, 5, 6]:

converted\_schedule[self.days\_in\_week[i]] = []

for \_class in schedule[f"{i}"]:

converted\_schedule[self.days\_in\_week[i]].append(

{"Время": " ".join(\_class["dayTime"].split()), "Дата": " ".join(\_class["dayDate"].split()),

"Дисциплина": " ".join(\_class["disciplName"].split()),

"Вид занятия": " ".join(\_class["disciplType"].split()),

"Аудитория": " ".join(\_class["audNum"].split()), "Здание": " ".join(\_class["buildNum"].split()),

"Преподаватель": " ".join(\_class["prepodName"].split()),

"Кафедра": " ".join(\_class["orgUnitName"].split())})

return converted\_schedule

def convert\_schedule\_by\_day(self, day=1): # конвертируем расписание в свой формат в определенный день

return self.convert\_schedule()[self.days\_in\_week[day]]

def from\_dict\_to\_list(self, \_dict):

sch = []

for day in \_dict:

a = [["Время", "Дата", "Дисциплина", "Вид занятия", "Аудитория", "Здание", "Преподаватель", "Кафедра"]]

for \_class in \_dict[day]:

a.append([\_class["Время"], \_class["Дата"], \_class["Дисциплина"], \_class["Вид занятия"],

\_class["Аудитория"],

\_class["Здание"], \_class["Преподаватель"], \_class["Кафедра"]])

sch.append(a)

return sch

def from\_dict\_to\_list\_day(self, \_dict, day):

a = [["Время", "Дата", "Дисциплина", "Вид занятия", "Аудитория", "Здание", "Преподаватель", "Кафедра"]]

for \_class in \_dict[day]:

a.append([\_class["Время"], \_class["Дата"], \_class["Дисциплина"], \_class["Вид занятия"],

\_class["Аудитория"],

\_class["Здание"], \_class["Преподаватель"], \_class["Кафедра"]])

return a

def formated\_json(self, \_json): # форматированный вывод в json формате

return json.dumps(\_json, indent=4, ensure\_ascii=False)

def is\_group\_schedule(self):

return True if self.get\_groupid() and self.get\_schedule() else False

def get\_date(self):

now = datetime.now() # текущие дата и время

return datetime.isoweekday(now)

def get\_week\_num(self):

week\_number = datetime.today().isocalendar()[1]

return ["чет", "неч"][week\_number % 2]

def get\_schedule\_today(self): # возвращает расписание в сегодняшний день

schedule\_day = []

day = self.get\_date()

week\_number = self.get\_week\_num()

schedule = self.convert\_schedule\_by\_day(day)

for \_class in schedule:

if week\_number in \_class["Дата"] or \_class["Дата"] == "":

schedule\_day.append(\_class)

return schedule\_day

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

sch = \_schedule(3239)

print(sch.get\_schedule\_today())

main.py

import telebot

from telebot import types

from weather import my\_weather

from m\_schedule import \_schedule

from generate\_table import schedule\_day\_img

from io import BytesIO

import sqlite3

import schedule

# Токен для доступа к боту

telebot\_token = "7182476955:AAE6eNJpU\_Mj-cBmPf3JkmltvJqmXYDacSg"

bot = telebot.TeleBot(telebot\_token)

# Подключение к базе данных

con = sqlite3.connect("kai\_bot\_id.db", check\_same\_thread=False)

cursor = con.cursor()

# Функция для обновления бота и отправки сообщения пользователям

def update\_bot():

a = cursor.execute("""SELECT user\_id FROM user\_group""").fetchall()

user\_id\_list = list(map(lambda x: x[0], a))

for id in user\_id\_list:

bot.send\_message(id, "Бот обновился!\nПожалуйста, напишите '/start'")

update\_bot()

def daily\_schedule():

pass

# Обработчик команды /start

@bot.message\_handler(commands=['start'])

def url(message):

# Создание клавиатуры с кнопками

markup\_start = types.ReplyKeyboardMarkup(resize\_keyboard=True)

weather\_btn = types.KeyboardButton(text='🌤️Погода')

schedule\_btn = types.KeyboardButton(text='📆Расписание')

group\_btn = types.KeyboardButton(text='👥Изменить номер группы')

markup\_start.add(weather\_btn, schedule\_btn, group\_btn)

# Проверка, зарегистрирован ли пользователь

info = cursor.execute('SELECT \* FROM user\_group WHERE user\_id=?', (message.from\_user.id,))

if info.fetchone() is None:

# Если пользователь не зарегистрирован, запросить номер группы

bot.send\_message(message.from\_user.id,

"Привет!\nДля персонализированной помощи мне нужен номер твоей учебной группы в КАИ.\n"

"Пожалуйста, отправь мне свой номер группы, чтобы я мог предоставить тебе наилучшую поддержку и актуальную информацию.\n"

"Спасибо! 🎓", reply\_markup=markup\_start)

bot.register\_next\_step\_handler(message, get\_num\_group)

else:

bot.send\_message(message.from\_user.id, "И снова привет!")

# Обработчик текстовых сообщений

@bot.message\_handler(content\_types=['text'])

def get\_text\_messages(message):

if message.text == '🌤️Погода':

weather = my\_weather()

bot.send\_message(message.from\_user.id, text=weather.formated\_print())

if message.text == '📆Расписание':

days\_keyboard = types.InlineKeyboardMarkup()

pon\_button = types.InlineKeyboardButton(text="Понедельник", callback\_data="day\_pon")

vt\_button = types.InlineKeyboardButton(text="Вторник", callback\_data="day\_vt")

sr\_button = types.InlineKeyboardButton(text="Среда", callback\_data="day\_sr")

ch\_button = types.InlineKeyboardButton(text="Чертверг", callback\_data="day\_ch")

pt\_button = types.InlineKeyboardButton(text="Пятница", callback\_data="day\_pt")

sub\_button = types.InlineKeyboardButton(text="Суббота", callback\_data="day\_sub")

days\_keyboard.add(pon\_button, vt\_button, sr\_button, ch\_button, pt\_button, sub\_button)

if cursor.execute("""SELECT num\_group FROM user\_group where user\_id = ?""",

(message.from\_user.id,)).fetchone()[0] != "-":

bot.send\_message(message.from\_user.id, "Выбери день недели", reply\_markup=days\_keyboard)

else:

bot.send\_message(message.from\_user.id, "Укажи номер группы")

bot.register\_next\_step\_handler(message, get\_variable\_num\_group)

if message.text == "👥Изменить номер группы":

bot.send\_message(message.from\_user.id,

"🔄 Конечно! Если у тебя изменился номер учебной группы или есть необходимость его обновить, просто отправь мне новый номер группы")

bot.register\_next\_step\_handler(message, get\_num\_group)

# Обработчик ввода номера группы, если ранее он не был указан

def get\_variable\_num\_group(message: types.Message):

if \_schedule(message.text).is\_group\_schedule():

cursor.execute("""UPDATE user\_group SET num\_group = ? WHERE user\_id = ?""",

(message.text, message.from\_user.id))

con.commit()

days\_keyboard = types.InlineKeyboardMarkup()

pon\_button = types.InlineKeyboardButton(text="Понедельник", callback\_data="day\_pon")

vt\_button = types.InlineKeyboardButton(text="Вторник", callback\_data="day\_vt")

sr\_button = types.InlineKeyboardButton(text="Среда", callback\_data="day\_sr")

ch\_button = types.InlineKeyboardButton(text="Чертверг", callback\_data="day\_ch")

pt\_button = types.InlineKeyboardButton(text="Пятница", callback\_data="day\_pt")

sub\_button = types.InlineKeyboardButton(text="Суббота", callback\_data="day\_sub")

days\_keyboard.add(pon\_button, vt\_button, sr\_button, ch\_button, pt\_button, sub\_button)

bot.send\_message(message.from\_user.id, "Выбери день недели", reply\_markup=days\_keyboard)

# Обработчик ввода номера группы при регистрации

def get\_num\_group(message: types.Message):

if message.text != "-":

if not \_schedule(message.text).get\_groupid():

bot.send\_message(message.from\_user.id,

"⚠️ Ой! Кажется, произошла ошибка. Пожалуйста, убедитесь, что введенный номер учебной группы корректен, и попробуйте еще раз.")

bot.send\_message(message.from\_user.id,

'🤖 Если не хочешь указывать номер учебной группы прямо сейчас, не беспокойся! Просто напиши "-"')

bot.register\_next\_step\_handler(message, get\_num\_group)

else:

bot.send\_message(message.from\_user.id,

"🌟 Отлично! Спасибо за предоставленную информацию. Твоя учебная группа успешно записана.")

info = cursor.execute('SELECT \* FROM user\_group WHERE user\_id=?', (message.from\_user.id,))

if info.fetchone() is None:

inf = (message.from\_user.id, message.text, "stable")

cursor.execute("INSERT INTO user\_group (user\_id, num\_group, group\_status) VALUES (?, ?, ?)", inf)

con.commit()

else:

cursor.execute("""UPDATE user\_group SET num\_group = ? WHERE user\_id = ?""",

(message.text, message.from\_user.id))

cursor.execute("""UPDATE user\_group SET group\_status = ? WHERE user\_id = ?""",

("stable", message.from\_user.id))

con.commit()

else:

bot.send\_message(message.from\_user.id,

"👌 Нет проблем! Если когда-то захочешь указать номер учебной группы или у тебя возникнут вопросы, не стесняйся обращаться.")

info = cursor.execute('SELECT \* FROM user\_group WHERE user\_id=?', (message.from\_user.id,))

if info.fetchone() is None:

inf = (message.from\_user.id, message.text, "variable")

cursor.execute("INSERT INTO user\_group (user\_id, num\_group, group\_status) VALUES (?, ?, ?)", inf)

con.commit()

else:

cursor.execute("""UPDATE user\_group SET num\_group = ? WHERE user\_id = ?""",

(message.text, message.from\_user.id))

cursor.execute("""UPDATE user\_group SET group\_status = ? WHERE user\_id = ?""",

("variable", message.from\_user.id))

con.commit()

# Обработчик inline-кнопок для выбора дня недели

@bot.callback\_query\_handler(func=lambda call: True)

def callback\_inline(call):

if call.message:

if call.data in ["day\_pon", "day\_vt", "day\_sr", "day\_ch", "day\_pt", "day\_sub"]:

group = (cursor.execute("""SELECT num\_group FROM user\_group where user\_id = ?""",

(call.from\_user.id,)).fetchone())[0]

if \_schedule(group).is\_group\_schedule():

days = {"day\_pon": 1, "day\_vt": 2, "day\_sr": 3, "day\_ch": 4, "day\_pt": 5, "day\_sub": 6}

day = days[call.data]

table = schedule\_day\_img(day=day, group=group)

img = table.create\_table\_schedule()

bio = BytesIO()

bio.name = 'image.jpeg'

img.save(bio, 'JPEG')

bio.seek(0)

bot.send\_photo(call.message.chat.id, photo=bio)

if (cursor.execute("""SELECT group\_status FROM user\_group where user\_id = ?""",

(call.from\_user.id,)).fetchone())[0] == "variable":

cursor.execute("""UPDATE user\_group SET num\_group = ? WHERE user\_id = ?""",

("-", call.from\_user.id))

con.commit()

else:

bot.send\_message(call.from\_user.id, "Занятий нет")

elif call.inline\_message\_id:

if call.data == "test":

bot.edit\_message\_text(inline\_message\_id=call.inline\_message\_id, text="Бдыщь")

# Вспомогательная функция для отображения таблицы расписания без изменения БД

def call\_table\_without\_bd(call, group):

if call.message:

if call.data in ["day\_pon", "day\_vt", "day\_sr", "day\_ch", "day\_pt", "day\_sub"]:

days = {"day\_pon": 1, "day\_vt": 2, "day\_sr": 3, "day\_ch": 4, "day\_pt": 5, "day\_sub": 6}

day = days[call.data]

table = schedule\_day\_img(day=day, group=group)

img = table.create\_table\_schedule()

bio = BytesIO()

bio.name = 'image.jpeg'

img.save(bio, 'JPEG')

bio.seek(0)

bot.send\_photo(call.message.chat.id, photo=bio)

# Запуск бота

bot.polling(none\_stop=True, interval=0)

Заключение

Проект представляет собой полезного Telegram-бота для студентов, который предоставляет актуальную информацию о погоде и расписании занятий. Каждый файл и функция в проекте выполняют определенную роль, обеспечивая корректное функционирование бота. В будущем проект можно расширить, добавив новые функции и улучшив существующие.