ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А. Н. Тихонова  
Департамент электронной инженерии

**Курс: Научно-исследовательский семинар "Искусственный интеллект в инженерном образовании"**

**ОТЧЁТ**

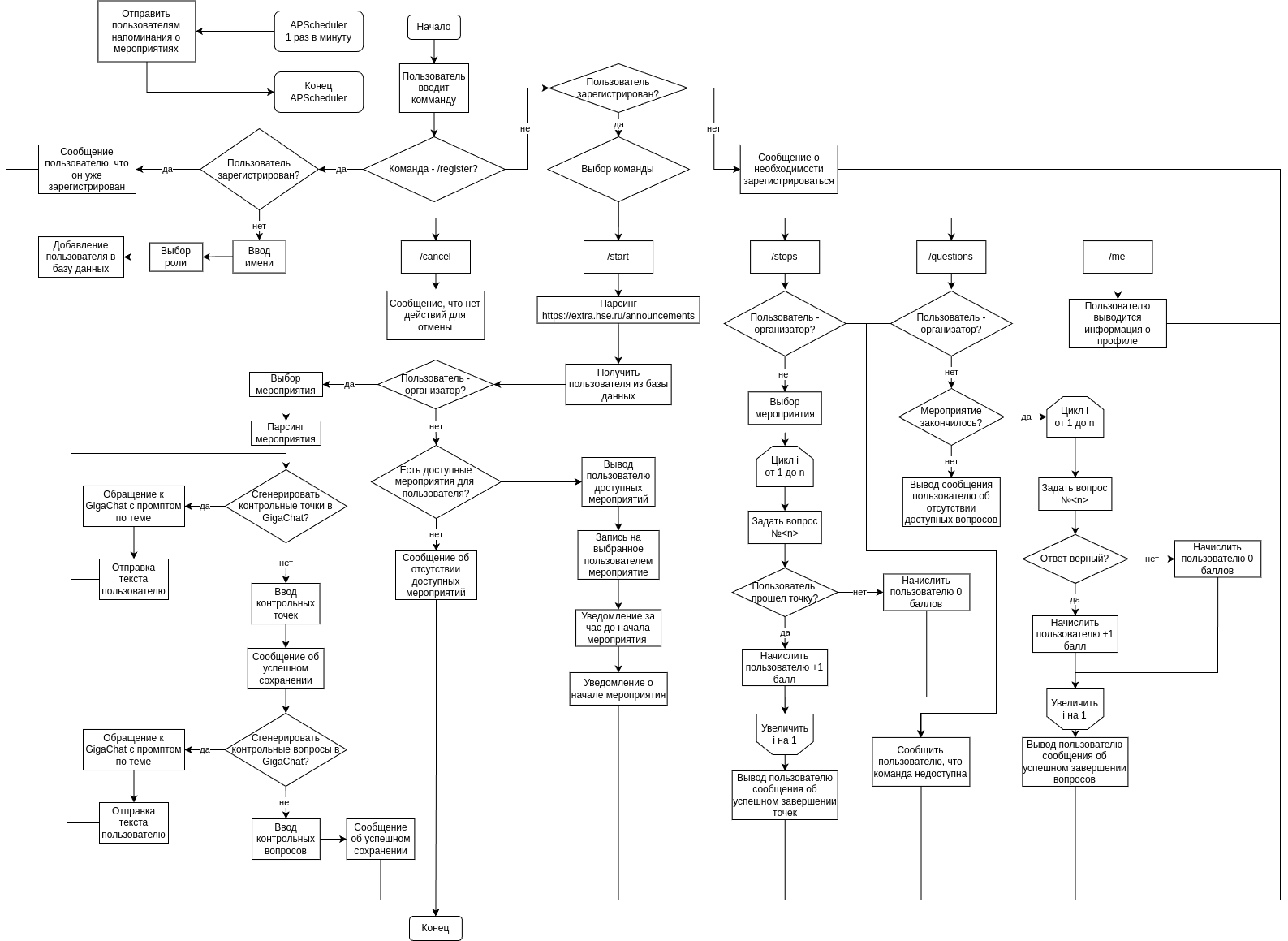
**по проекту**

Тема: «Telegram-бот для геймификации посещения мероприятий с функцией начисления баллов за посещение мероприятий»

**Студенты:** Иксанова Алина Рашитовна  
Быцко Вячеслав Михайлович  
Курбетьева Арина Сергеевна  
**Группа:** БИТ233  
**Дата сдачи:** 16.01.25  
**Руководитель:** Варнавский Александр Николаевич

**МОСКВА 2024**

**Блок-схема**

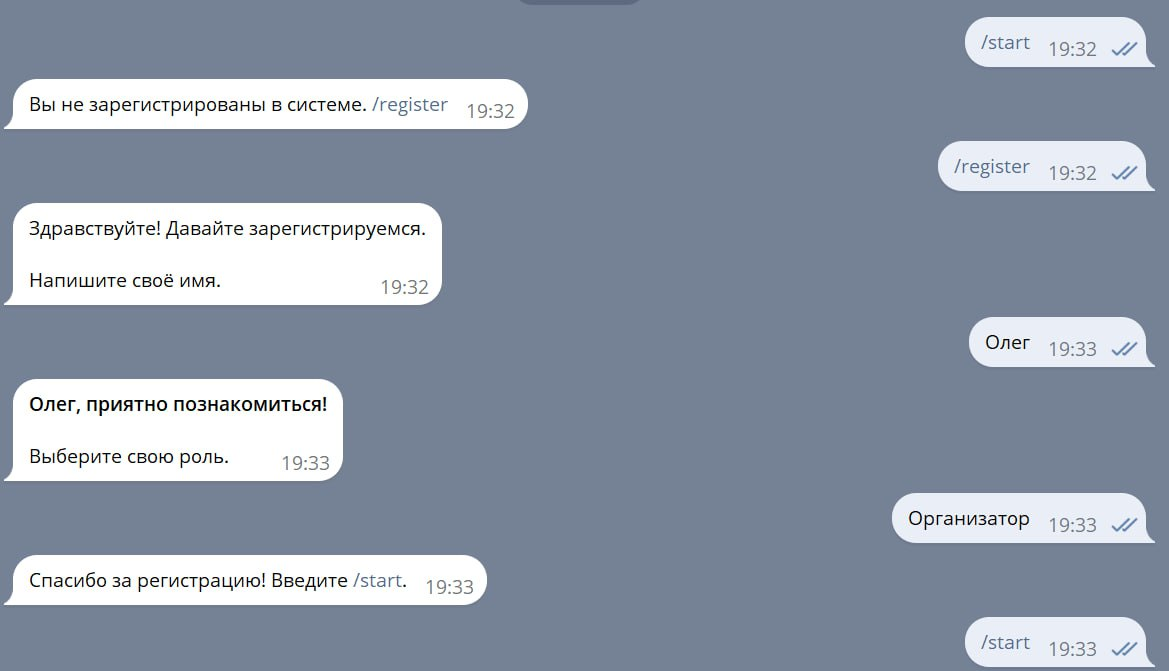
****

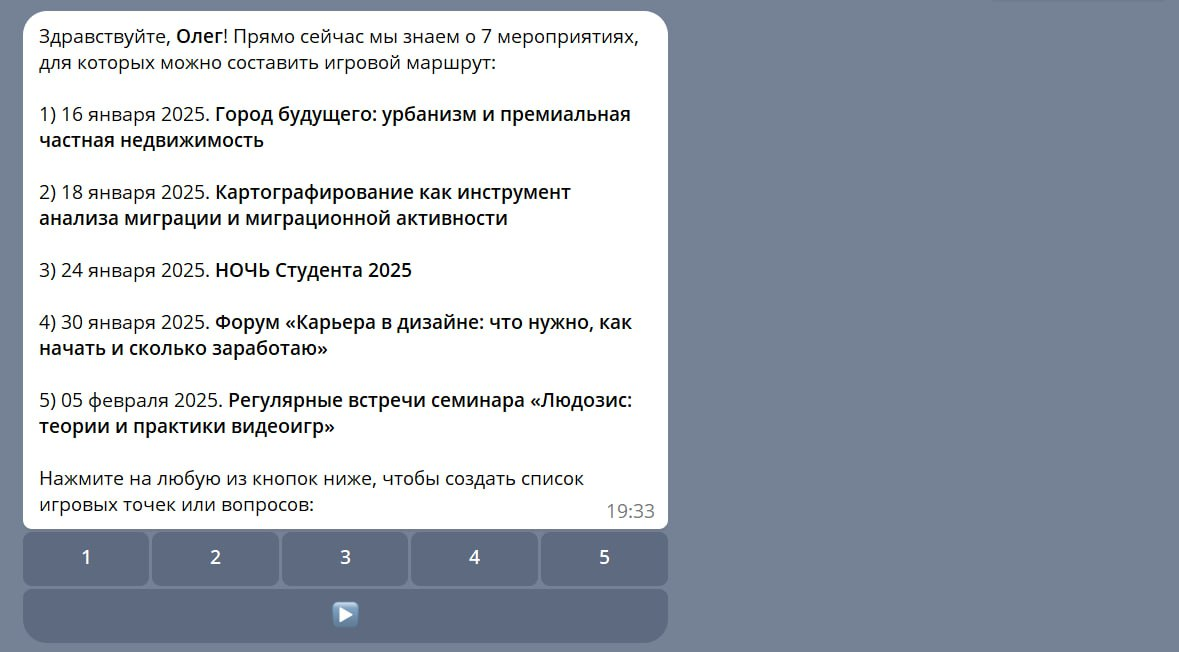
Эта же фотография по ссылке в максимальном качестве:

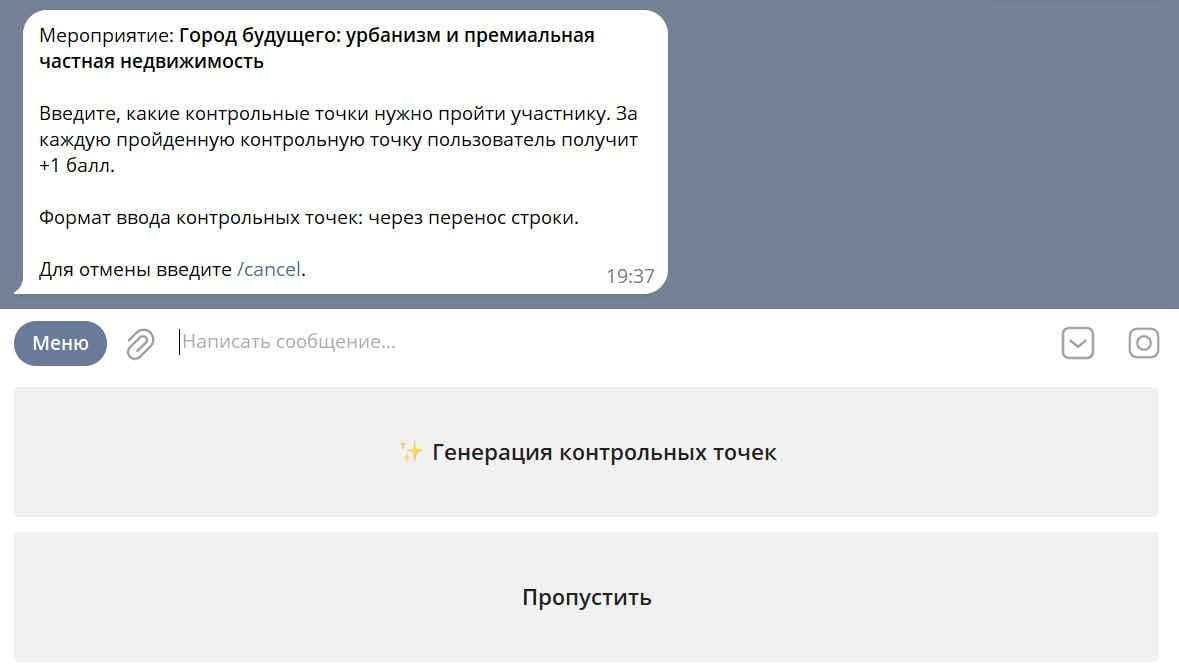
<https://raw.githubusercontent.com/vyacheslavbytsko/HSE-Extra-Events-Bot/refs/heads/main/%D0%B1%D0%BB%D0%BE%D0%BA-%D1%81%D1%85%D0%B5%D0%BC%D0%B0.png>

**Скриншоты работы**

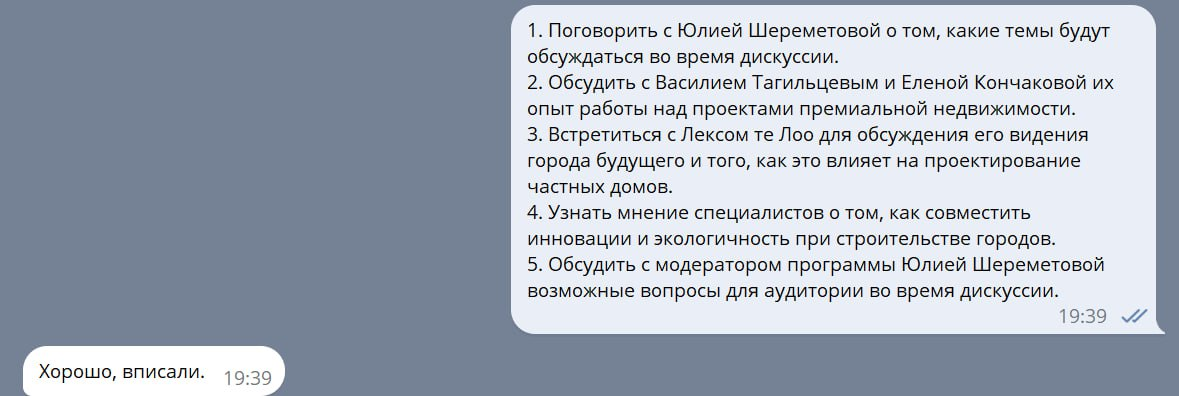
Пользователь 1 – организатор мероприятий:

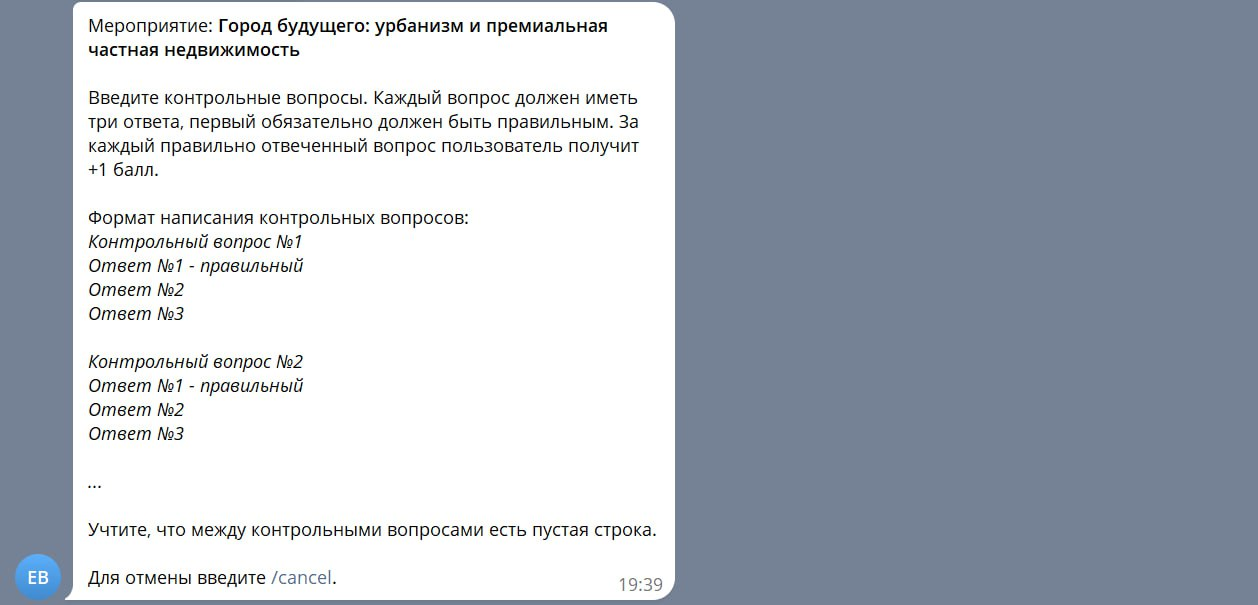


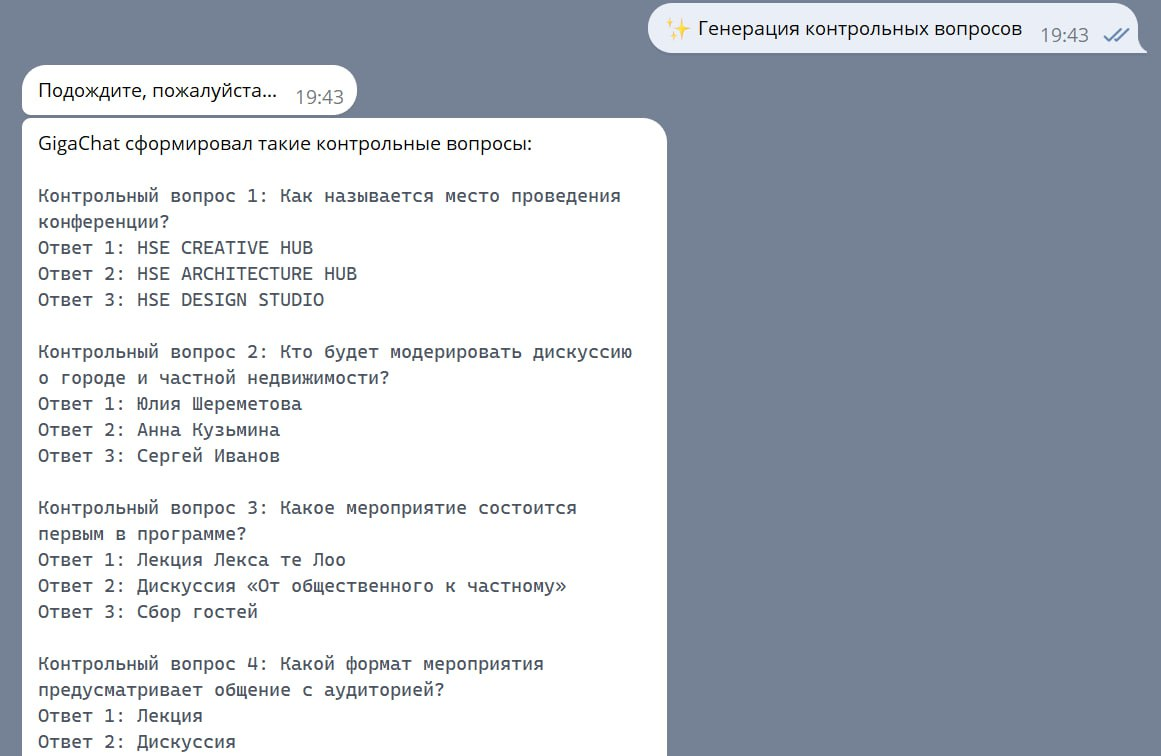


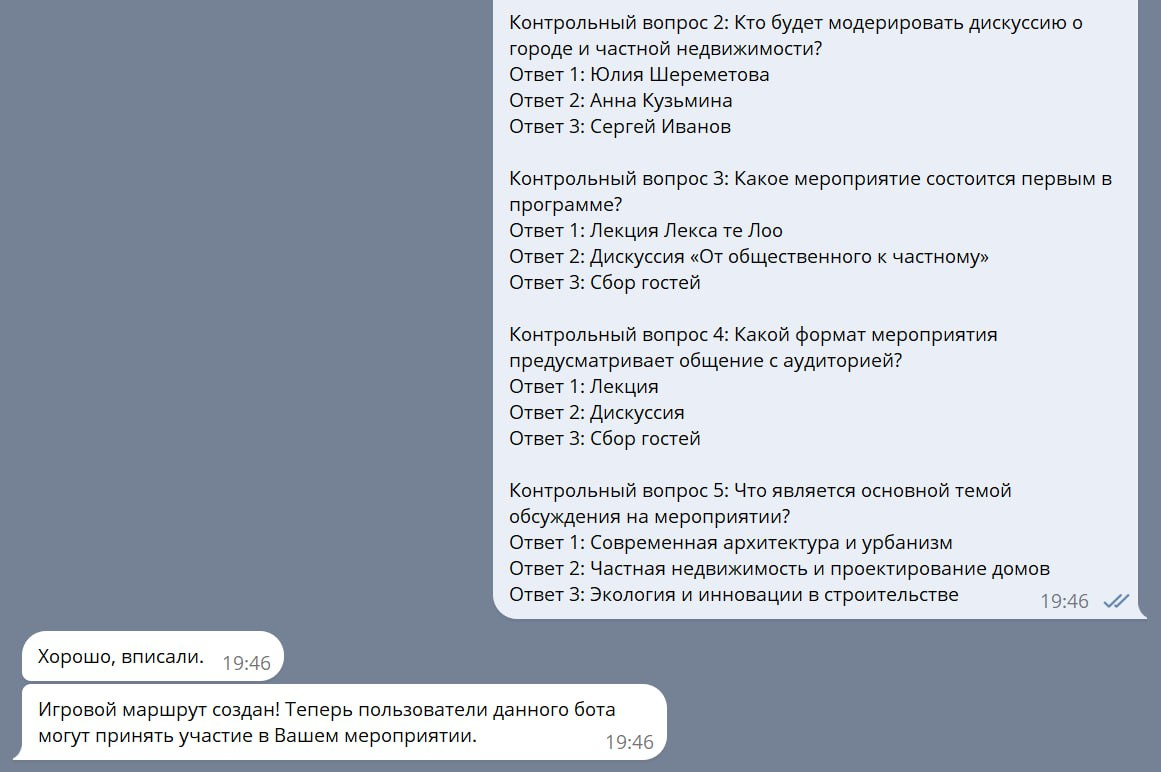




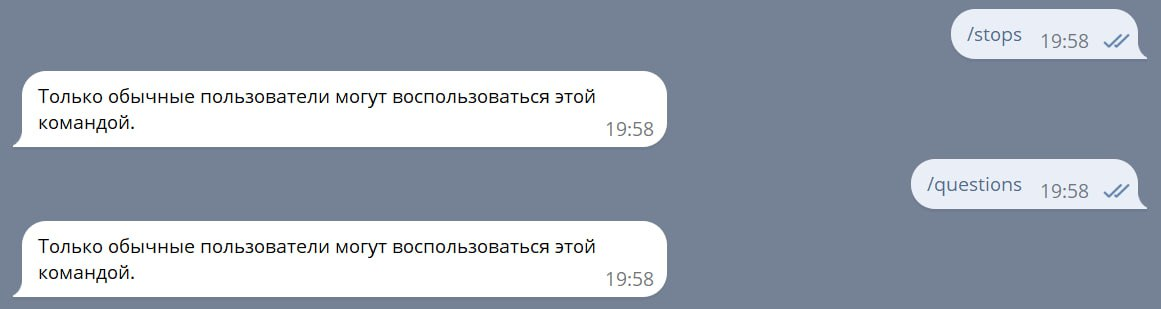




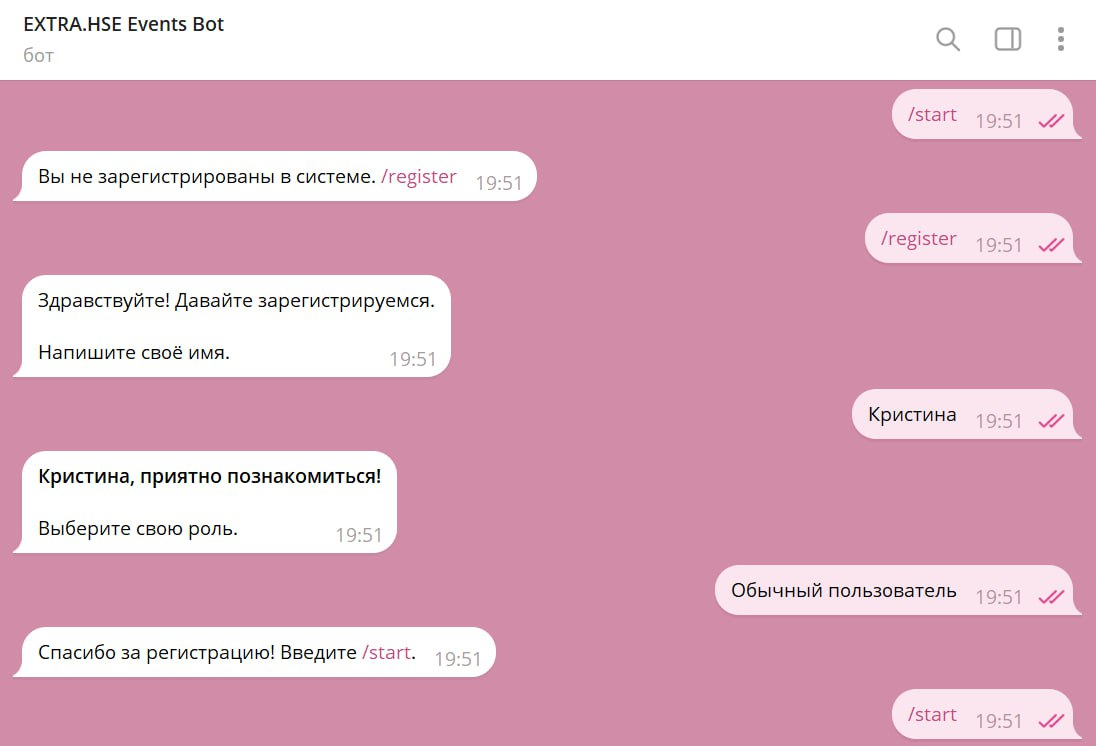


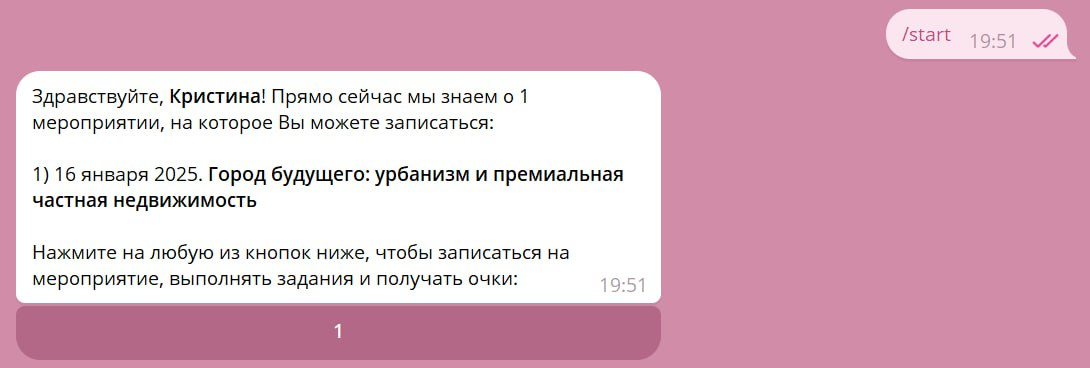


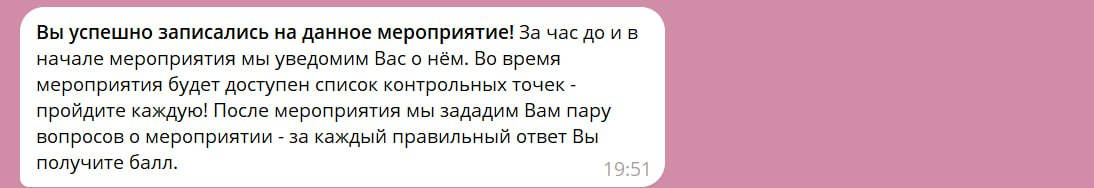


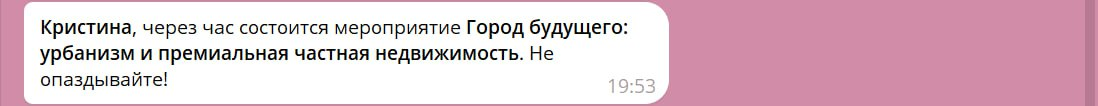


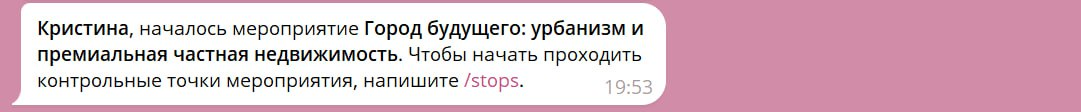
Пользователь 2 – участник мероприятий:

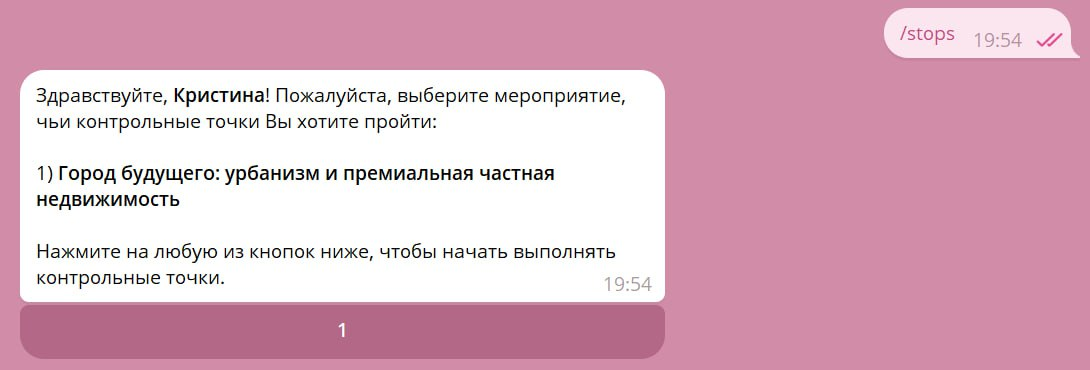


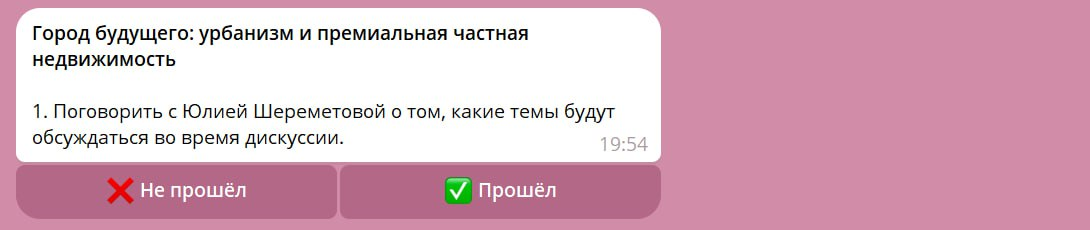




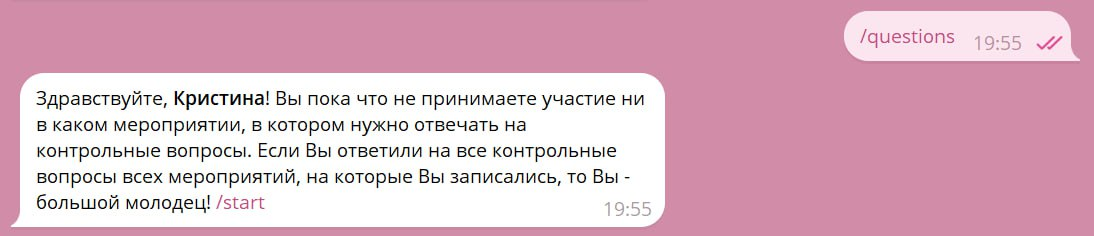


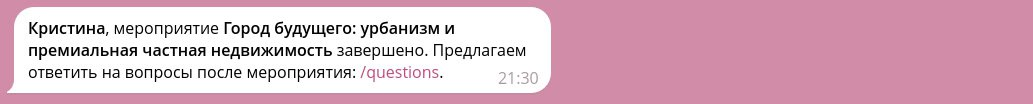


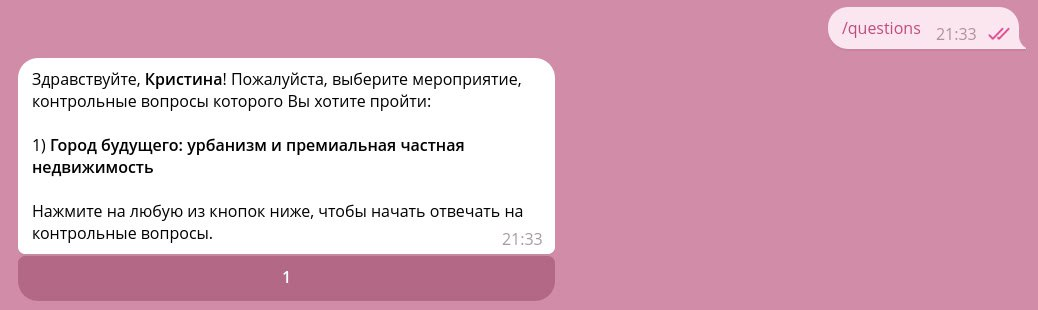


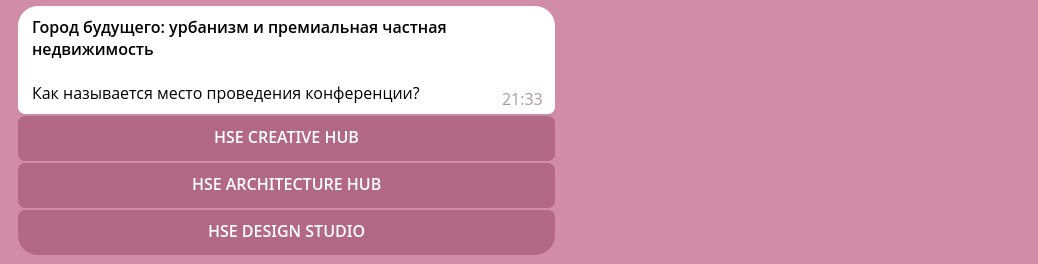


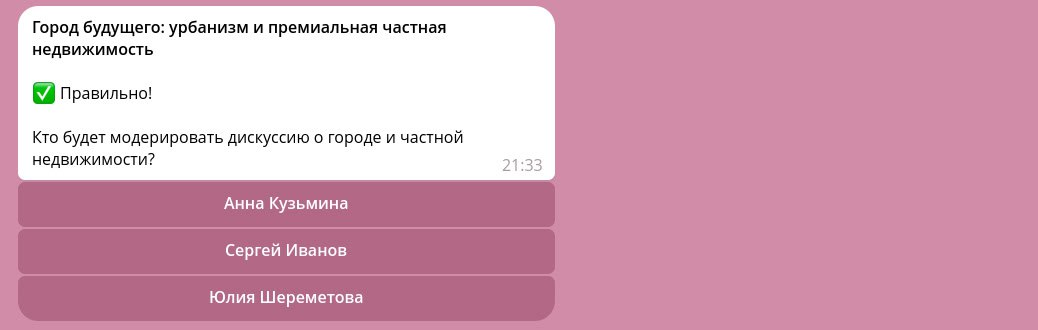


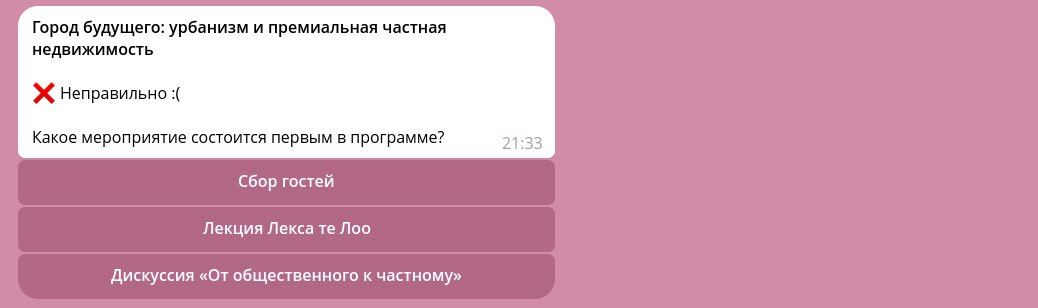


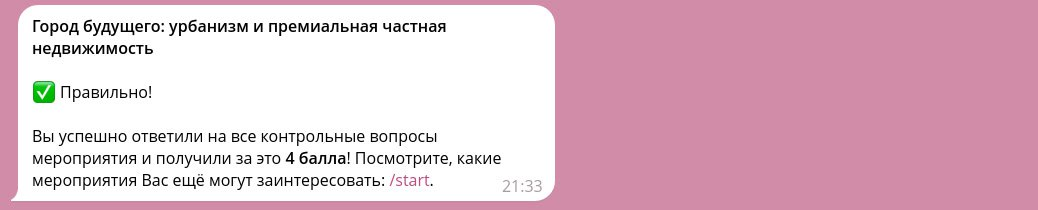


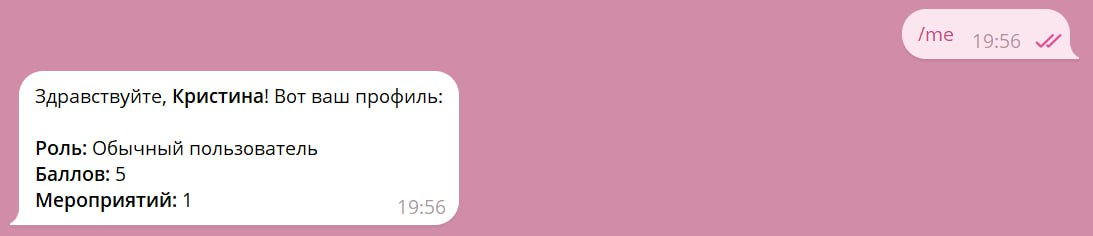












**Программный код**

**main.py**

import asyncio

import datetime

import locale

import logging

import random

import sys

from typing import Callable, Dict, Any, Awaitable

import pytz

from aiogram import Bot, Dispatcher, html, F, BaseMiddleware

from aiogram.client.default import DefaultBotProperties

from aiogram.enums import ParseMode

from aiogram.filters import CommandStart, Command

from aiogram.fsm.context import FSMContext

from aiogram.fsm.state import StatesGroup, State

from aiogram.fsm.storage.memory import MemoryStorage

from aiogram.types import Message, CallbackQuery, BotCommand, KeyboardButton, ReplyKeyboardRemove, TelegramObject, \

InlineKeyboardButton

from aiogram.utils.keyboard import InlineKeyboardBuilder, ReplyKeyboardBuilder

from apscheduler.schedulers.asyncio import AsyncIOScheduler

from sqlalchemy import select

from ai import get\_stops\_from\_gigachat, get\_questions\_from\_gigachat

from browser import get\_event\_from\_internet, get\_rough\_events\_from\_internet, driver

from classes import CreateEventGameCallback, JoinEventGameCallback, EventsMessageChangePageCallback, \

EventInfoCallback, EventGameStopCallback, EventsGamesStopsChangePageCallback, \

EventsGamesQuestionsChangePageCallback, EventGameQuestionCallback

from db import async\_session, db\_add\_event\_game\_to\_user, create\_tables, db\_get\_user, db\_add\_user, DBUser, DBEventGame, \

db\_add\_event\_game, DBUserEventGame, db\_get\_event\_game

from misc import declension

from tokens import tg\_token

locale.setlocale(locale.LC\_TIME, "ru\_RU.UTF-8")

moscow\_tz = pytz.timezone("Europe/Moscow")

bot = Bot(token=tg\_token, default=DefaultBotProperties(parse\_mode=ParseMode.HTML))

dp = Dispatcher(storage=MemoryStorage())

csv\_file\_datetime = datetime.datetime.now()

csv\_file\_name = f"log-{str(csv\_file\_datetime)}.csv"

class EventGameCreationStates(StatesGroup):

stops = State()

questions = State()

class RegistrationStates(StatesGroup):

full\_name = State()

role = State()

class RegistrationMiddleware(BaseMiddleware):

async def \_\_call\_\_(

self,

handler: Callable[[TelegramObject, Dict[str, Any]], Awaitable[Any]],

event: TelegramObject,

data: Dict[str, Any]

) -> Any:

user = data["event\_from\_user"]

state = data["state"]

with open(csv\_file\_name, "a") as csv\_file:

csv\_file.write(str(event.update\_id)+",")

csv\_file.write(str(data["event\_from\_user"].id)+",")

csv\_file.write("\""+str(datetime.datetime.now())+"\",")

if event.message:

csv\_file.write("message,")

csv\_file.write(event.message.text+"\n")

elif event.callback\_query:

csv\_file.write("callback\_query,")

csv\_file.write(event.callback\_query.data+"\n")

else:

csv\_file.write("unknown,None\n")

async with async\_session() as session:

db\_user = await db\_get\_user(session, user.id)

if db\_user is None:

if event.message.text == "/register" or (

await state.get\_state() in [RegistrationStates.full\_name, RegistrationStates.role]):

result = await handler(event, data)

return result

else:

await bot.send\_message(data["event\_context"].chat.id, "Вы не зарегистрированы в системе. /register")

else:

if not event.message.text.startswith("/register"):

data["db\_user"] = db\_user

result = await handler(event, data)

return result

else:

await bot.send\_message(data["event\_context"].chat.id, "Вы уже зарегистрированы в системе. /start")

@dp.message(Command("register"))

async def command\_register\_handler(message: Message, state: FSMContext):

await message.answer(f"Здравствуйте! Давайте зарегистрируемся.\n\nНапишите своё имя.")

await state.set\_state(RegistrationStates.full\_name)

button\_user = "Обычный пользователь"

button\_organisator = "Организатор"

@dp.message(RegistrationStates.full\_name)

async def registration\_full\_name\_handler(message: Message, state: FSMContext):

await state.update\_data({"full\_name": message.text})

reply\_keyboard\_builder = ReplyKeyboardBuilder()

reply\_keyboard\_builder.row(KeyboardButton(text=button\_user))

reply\_keyboard\_builder.row(KeyboardButton(text=button\_organisator))

await message.answer(f"{html.bold(message.text + ", приятно познакомиться!")}\n\nВыберите свою роль.",

reply\_markup=reply\_keyboard\_builder.as\_markup())

await state.set\_state(RegistrationStates.role)

@dp.message(RegistrationStates.role)

async def registration\_role\_handler(message: Message, state: FSMContext):

role = None

if message.text == button\_user:

role = "user"

elif message.text == button\_organisator:

role = "organisator"

if role is None:

await message.answer("Неправильная роль. Введите роль ещё раз.")

else:

async with async\_session() as session:

await db\_add\_user(session=session,

user\_id=message.from\_user.id,

user\_full\_name=(await state.get\_data())["full\_name"],

user\_role=role)

await state.clear()

await message.answer("Спасибо за регистрацию! Введите /start.", reply\_markup=ReplyKeyboardRemove())

@dp.message(Command("me"))

async def command\_me\_handler(message: Message, db\_user: DBUser):

await message.answer(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Вот ваш профиль:\n\n"

f"{html.bold("Роль:")} {button\_user if db\_user.role == "user" else button\_organisator}\n"

f"{f"{html.bold("Баллов:")} {db\_user.points}" if db\_user.role == "user" else ""}\n"

f"{f"{html.bold("Мероприятий:")} {len(await db\_user.awaitable\_attrs.event\_games)}" if db\_user.role == "user" else ""}"

)

@dp.message(CommandStart())

async def command\_start\_handler(message: Message, db\_user: DBUser) -> None:

sent\_message = await message.answer(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Сейчас мы получаем список мероприятий EXTRA.HSE, пожалуйста, подождите...")

await edit\_events\_message(sent\_message=sent\_message, db\_user=db\_user, page\_index=0)

async def edit\_events\_message(sent\_message: Message, db\_user: DBUser,

page\_index: int):

rough\_events = await get\_rough\_events\_from\_internet(with\_games=(db\_user.role == "user"))

if db\_user.role == "user":

db\_user\_event\_games = await db\_user.awaitable\_attrs.event\_games

db\_user\_event\_games\_ids = [

db\_user\_event\_game.event\_id for db\_user\_event\_game in db\_user\_event\_games]

rough\_events = list(filter(lambda rough\_event: rough\_event.id not in db\_user\_event\_games\_ids, rough\_events))

if len(rough\_events) == 0:

await sent\_message.edit\_text(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Пока что мероприятий, {"на которые Вы можете записаться" if db\_user.role == "user" else "для которых можно составить игровой маршрут"}, нет."

)

return

rough\_events\_limited = rough\_events[page\_index \* 5:page\_index \* 5 + 5]

max\_pages = ((len(rough\_events) - 1) // 5 + 1) - 1

inline\_keyboard\_builder = InlineKeyboardBuilder()

for i in range(len(rough\_events\_limited)):

inline\_keyboard\_builder.button(text=f"{(page\_index \* 5) + i + 1}",

callback\_data=EventInfoCallback(

event\_id=rough\_events\_limited[i].id,

from\_page=page\_index))

if page\_index != 0:

inline\_keyboard\_builder.button(text="◀️",

callback\_data=EventsMessageChangePageCallback(

page=max(page\_index - 1, 0)))

if page\_index != max\_pages:

inline\_keyboard\_builder.button(text="▶️️",

callback\_data=EventsMessageChangePageCallback(

page=min(page\_index + 1, max\_pages)))

inline\_keyboard\_builder.adjust(len(rough\_events\_limited), int(page\_index != 0) + int(page\_index != max\_pages))

text\_meropriyatiye = declension(len(rough\_events), "мероприятиях", "мероприятии", "мероприятиях")

text\_na\_kotoriye = declension(len(rough\_events), "которые", "которое", "которые")

text\_dlya\_kototrykh = declension(len(rough\_events), "которых", "которого", "которых")

await sent\_message.edit\_text(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! "

f"Прямо сейчас мы знаем о {len(rough\_events)} {text\_meropriyatiye}, {f"на {text\_na\_kotoriye} Вы можете записаться" if db\_user.role == "user" else f"для {text\_dlya\_kototrykh} можно составить игровой маршрут"}:\n\n"

f"{"\n\n".join([f"{rough\_events.index(rough\_event) + 1}) " +

rough\_event.date.strftime("%d %B %Y") + ". " +

html.bold(rough\_event.title)

for rough\_event in rough\_events\_limited])}\n\n"

f"Нажмите на любую из кнопок ниже, чтобы " +

(f"записаться на мероприятие, "

f"выполнять задания и получать очки:"

if db\_user.role == "user" else

f"создать список игровых точек или вопросов:"),

reply\_markup=inline\_keyboard\_builder.as\_markup())

@dp.callback\_query(EventsMessageChangePageCallback.filter())

async def events\_message\_change\_page\_callback(query: CallbackQuery, callback\_data: EventsMessageChangePageCallback,

db\_user: DBUser):

await edit\_events\_message(

sent\_message=query.message,

db\_user=db\_user,

page\_index=callback\_data.page)

await query.answer()

@dp.callback\_query(EventInfoCallback.filter())

async def events\_message\_event\_info\_callback(query: CallbackQuery, callback\_data: EventInfoCallback, db\_user: DBUser):

event = get\_event\_from\_internet(callback\_data.event\_id)

inline\_keyboard\_builder = InlineKeyboardBuilder()

if db\_user.role == "organisator":

inline\_keyboard\_builder.button(text="Создать игровой маршрут",

callback\_data=CreateEventGameCallback(

event\_id=event.id))

else:

inline\_keyboard\_builder.button(text="Я хочу участвовать",

callback\_data=JoinEventGameCallback(

event\_id=event.id))

inline\_keyboard\_builder.button(text="↩️",

callback\_data=EventsMessageChangePageCallback(

page=callback\_data.from\_page))

inline\_keyboard\_builder.adjust(1, 1)

await query.message.edit\_text(

f"🔸 {html.bold(event.title)}. {event.rating}\n\n"

f"{event.description}\n\n"

f"{html.bold("Где:")} {event.address}\n"

f"{html.bold("Начало:")} {event.ical.get("DTSTART").dt.astimezone(moscow\_tz).strftime("%d %B %Y %H:%M")}\n"

f"{html.bold("Конец:")} {event.ical.get("DTEND").dt.astimezone(moscow\_tz).strftime("%d %B %Y %H:%M")}\n"

f"{html.bold(html.link("Ссылка", f"https://extra.hse.ru/announcements/{event.id}.html"))}",

reply\_markup=inline\_keyboard\_builder.as\_markup()

)

await query.answer()

@dp.callback\_query(CreateEventGameCallback.filter())

async def create\_event\_game\_callback(query: CallbackQuery, callback\_data: CreateEventGameCallback, state: FSMContext):

event = get\_event\_from\_internet(callback\_data.event\_id)

reply\_keyboard\_builder = ReplyKeyboardBuilder()

reply\_keyboard\_builder.row(KeyboardButton(text="✨ Генерация контрольных точек"))

await bot.send\_message(query.message.chat.id,

f"Мероприятие: {html.bold(event.title)}\n\nВведите, какие контрольные точки нужно пройти участнику. За каждую пройденную контрольную точку пользователь получит +1 балл.\n\nФормат ввода контрольных точек: через перенос строки.\n\nДля отмены введите /cancel.",

reply\_markup=reply\_keyboard\_builder.as\_markup())

await state.set\_state(EventGameCreationStates.stops)

await state.update\_data({"event": event})

await query.answer()

await query.message.delete()

@dp.message(Command("cancel"), EventGameCreationStates.stops)

async def create\_event\_game\_cancel\_stops(message: Message, state: FSMContext):

await message.answer("Отменили запись контрольных точек и создание маршрута мероприятия. /start",

reply\_markup=ReplyKeyboardRemove())

await state.clear()

@dp.message(F.text, EventGameCreationStates.stops)

async def create\_event\_game\_stops\_handler(message: Message, state: FSMContext):

event = (await state.get\_data())["event"]

# for further bot messages

reply\_keyboard\_builder = ReplyKeyboardBuilder()

reply\_keyboard\_builder.row(KeyboardButton(text="✨ Генерация контрольных вопросов"))

if message.text == "✨ Генерация контрольных точек":

await message.answer("Подождите, пожалуйста...")

gigachat\_answer = ""

i = 0

while len(gigachat\_answer.split("\n")) not in [6, 7] and i < 3:

gigachat\_answer = get\_stops\_from\_gigachat(event.title, event.description)

i += 1

if i >= 3:

await message.answer(f"GigaChat не смог сформировать контрольные точки. Повторите попытку.")

else:

await message.answer(

f"GigaChat сформировал такие контрольные точки:\n\n{html.code(gigachat\_answer.replace("Контрольные точки:", "").strip())}\n\nВы можете взять этот ответ за основу. Вы также можете запросить у GigaChat сформировать контрольные точки ещё раз - нажмите на кнопку ниже. Как только Вы придумаете контрольные точки, отправьте их нам в ответном сообщении.")

else:

await state.update\_data({"stops": message.text.split("\n")})

await message.answer("Хорошо, вписали.", reply\_markup=ReplyKeyboardRemove())

await message.answer(

f"Мероприятие: {html.bold(event.title)}\n\nВведите контрольные вопросы. Каждый вопрос должен иметь три ответа, первый обязательно должен быть правильным. За каждый правильно отвеченный вопрос пользователь получит +1 балл.\n\nФормат написания контрольных вопросов:\n{html.italic("Контрольный вопрос №1\nОтвет №1 - правильный\nОтвет №2\nОтвет №3\n\nКонтрольный вопрос №2\nОтвет №1 - правильный\nОтвет №2\nОтвет №3\n\n...")}\n\nУчтите, что между контрольными вопросами есть пустая строка. \n\nДля отмены введите /cancel.",

reply\_markup=reply\_keyboard\_builder.as\_markup())

await state.set\_state(EventGameCreationStates.questions)

@dp.message(Command("cancel"), EventGameCreationStates.questions)

async def create\_event\_game\_cancel\_questions(message: Message, state: FSMContext):

await message.answer("Отменили запись контрольных вопросов и создание игрового маршрута мероприятия. /start",

reply\_markup=ReplyKeyboardRemove())

await state.clear()

@dp.message(F.text, EventGameCreationStates.questions)

async def create\_event\_game\_questions\_handler(message: Message, state: FSMContext):

event = (await state.get\_data())["event"]

if message.text == "✨ Генерация контрольных вопросов":

await message.answer("Подождите, пожалуйста...")

gigachat\_answer = ""

i = 0

while len(gigachat\_answer.split("\n")) != 24 and i < 3:

gigachat\_answer = get\_questions\_from\_gigachat(event.title, event.description)

i += 1

if i >= 3:

await message.answer(f"GigaChat не смог сформировать контрольные вопросы. Повторите попытку.")

else:

await message.answer(

f"GigaChat сформировал такие контрольные вопросы:\n\n{html.code(gigachat\_answer)}\n\nВы можете взять этот ответ за основу. Вы также можете запросить у GigaChat сформировать контрольные вопросы ещё раз - нажмите на кнопку ниже. Как только Вы придумаете контрольные вопросы, отправьте их нам в ответном сообщении.")

else:

# first of all, check

questions\_and\_answers = message.text.split("\n\n")

if not all(len(question\_and\_answers.split("\n")) == 4 for question\_and\_answers in questions\_and\_answers):

await message.answer("Проверьте правильность введённых данных. У каждого вопроса по три ответа.")

return

questions = [question\_and\_answers.split("\n") for question\_and\_answers in questions\_and\_answers]

for i in range(len(questions)):

if questions[i][0].startswith("Контрольный вопрос"): questions[i][0] = ":".join(

questions[i][0].split(":")[1:]).strip()

for j in range(1, 3 + 1):

if questions[i][j].startswith("Ответ"): questions[i][j] = ":".join(

questions[i][j].split(":")[1:]).strip()

await state.update\_data({"questions": questions})

await message.answer("Хорошо, вписали.", reply\_markup=ReplyKeyboardRemove())

async with async\_session() as session:

db\_event\_game = DBEventGame()

db\_event\_game.event\_id = event.id

db\_event\_game.event\_title = event.title

db\_event\_game.stops = (await state.get\_data())["stops"]

db\_event\_game.questions = (await state.get\_data())["questions"]

db\_event\_game.start = event.ical.get("DTSTART").dt.astimezone(moscow\_tz)

db\_event\_game.end = event.ical.get("DTEND").dt.astimezone(moscow\_tz)

await db\_add\_event\_game(session, db\_event\_game)

await message.answer(

"Игровой маршрут создан! Теперь пользователи данного бота могут принять участие в Вашем мероприятии.",

reply\_markup=ReplyKeyboardRemove())

await state.clear()

@dp.callback\_query(JoinEventGameCallback.filter())

async def join\_event\_game\_callback(query: CallbackQuery, callback\_data: JoinEventGameCallback, db\_user: DBUser):

async with async\_session() as session:

await db\_add\_event\_game\_to\_user(

session=session,

user\_id=db\_user.user\_id,

event\_id=callback\_data.event\_id)

await query.message.edit\_text(f"{html.bold("Вы успешно записались на данное мероприятие!")} "

"За час до и в начале мероприятия мы уведомим Вас о нём. "

"Во время мероприятия будет доступен список контрольных "

"точек - пройдите каждую! После мероприятия мы зададим Вам "

"пару вопросов о мероприятии - за каждый правильный ответ Вы получите "

"балл.")

await query.answer()

async def send\_notifications():

async with async\_session() as session:

should\_commit = False

db\_event\_games = await session.execute(select(DBEventGame))

for db\_event\_game in db\_event\_games.scalars():

notify\_about\_pre\_start = datetime.datetime.now(tz=moscow\_tz) >= moscow\_tz.localize(db\_event\_game.start) - datetime.timedelta(hours=1)

notify\_about\_start = datetime.datetime.now(tz=moscow\_tz) >= moscow\_tz.localize(db\_event\_game.start)

notify\_about\_end = datetime.datetime.now(tz=moscow\_tz) >= moscow\_tz.localize(db\_event\_game.end)

db\_users: list[DBUser] = await db\_event\_game.awaitable\_attrs.users

for db\_user in db\_users:

db\_user\_event\_game = await session.get(DBUserEventGame, (db\_user.user\_id, db\_event\_game.event\_id))

if notify\_about\_pre\_start and not db\_user\_event\_game.pre\_start\_notified:

await bot.send\_message(

chat\_id=db\_user.user\_id,

text=f"{html.bold(db\_user.full\_name)}, через час состоится мероприятие {html.bold(db\_event\_game.event\_title)}. Не опаздывайте!")

db\_user\_event\_game.pre\_start\_notified = True

should\_commit = True

if notify\_about\_start and not db\_user\_event\_game.start\_notified:

await bot.send\_message(

chat\_id=db\_user.user\_id,

text=f"{html.bold(db\_user.full\_name)}, началось мероприятие {html.bold(db\_event\_game.event\_title)}. Чтобы начать проходить контрольные точки мероприятия, напишите /stops.")

db\_user\_event\_game.start\_notified = True

should\_commit = True

if notify\_about\_end and not db\_user\_event\_game.end\_notified:

await bot.send\_message(

chat\_id=db\_user.user\_id,

text=f"{html.bold(db\_user.full\_name)}, мероприятие {html.bold(db\_event\_game.event\_title)} завершено. Предлагаем ответить на вопросы после мероприятия: /questions.")

db\_user\_event\_game.end\_notified = True

should\_commit = True

if should\_commit:

await session.commit()

@dp.message(Command("stops"))

async def stops\_command\_handler(message: Message, state: FSMContext, db\_user: DBUser):

if db\_user.role != "user":

await message.answer("Только обычные пользователи могут воспользоваться этой командой.")

return

sent\_message = await message.answer(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Сейчас мы получаем список мероприятий EXTRA.HSE, пожалуйста, подождите...")

await edit\_stops\_events\_message(sent\_message=sent\_message, db\_user=db\_user, page\_index=0)

async def edit\_stops\_events\_message(sent\_message: Message, db\_user: DBUser, page\_index: int):

async with async\_session() as session:

db\_event\_games: list[DBEventGame] = await db\_user.awaitable\_attrs.event\_games

db\_event\_games\_filtered: list[DBEventGame] = []

for db\_event\_game in db\_event\_games:

if (moscow\_tz.localize(db\_event\_game.end) + datetime.timedelta(days=10) >=

datetime.datetime.now(tz=moscow\_tz) >=

moscow\_tz.localize(db\_event\_game.start)

and

not (await session.get(DBUserEventGame, (db\_user.user\_id, db\_event\_game.event\_id))).stops\_done):

db\_event\_games\_filtered.append(db\_event\_game)

db\_event\_games = db\_event\_games\_filtered

if len(db\_event\_games) == 0:

await sent\_message.edit\_text(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Вы пока что не принимаете участие ни в каком мероприятии, в котором нужно проходить контрольные точки. Если Вы прошли все контрольные точки всех мероприятий, на которые Вы записались, то Вы - большой молодец! /start"

)

return

db\_event\_games\_limited = db\_event\_games[page\_index\*5:page\_index\*5+5]

max\_pages = ((len(db\_event\_games) - 1) // 5 + 1) - 1

inline\_keyboard\_builder = InlineKeyboardBuilder()

for i in range(len(db\_event\_games\_limited)):

inline\_keyboard\_builder.button(text=f"{(page\_index \* 5) + i + 1}",

callback\_data=EventGameStopCallback(

event\_id=db\_event\_games\_limited[i].event\_id,

stop\_index=0,

points=0))

if page\_index != 0:

inline\_keyboard\_builder.button(text="◀️",

callback\_data=EventsGamesStopsChangePageCallback(

page=max(page\_index - 1, 0)))

if page\_index != max\_pages:

inline\_keyboard\_builder.button(text="▶️️",

callback\_data=EventsGamesStopsChangePageCallback(

page=min(page\_index + 1, max\_pages)))

inline\_keyboard\_builder.adjust(len(db\_event\_games\_limited), int(page\_index != 0) + int(page\_index != max\_pages))

await sent\_message.edit\_text(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! "

f"Пожалуйста, выберите мероприятие, чьи контрольные точки Вы хотите пройти:\n\n"

f"{"\n\n".join([f"{db\_event\_games.index(db\_event\_game) + 1}) " +

html.bold(db\_event\_game.event\_title)

for db\_event\_game in db\_event\_games\_limited])}\n\n"

f"Нажмите на любую из кнопок ниже, чтобы начать выполнять контрольные точки.",

reply\_markup=inline\_keyboard\_builder.as\_markup())

@dp.callback\_query(EventsGamesStopsChangePageCallback.filter())

async def events\_games\_stops\_change\_page\_callback(query: CallbackQuery, callback\_data: EventsGamesStopsChangePageCallback,

db\_user: DBUser):

await edit\_stops\_events\_message(

sent\_message=query.message,

db\_user=db\_user,

page\_index=callback\_data.page)

await query.answer()

@dp.callback\_query(EventGameStopCallback.filter())

async def event\_game\_stop\_callback(query: CallbackQuery, callback\_data: EventGameStopCallback, db\_user: DBUser):

async with async\_session() as session:

db\_event\_game = await db\_get\_event\_game(session, callback\_data.event\_id)

stops: list[str] = db\_event\_game.stops

stop\_index = callback\_data.stop\_index

if callback\_data.stop\_index >= len(stops):

ball = declension(callback\_data.points, "баллов", "балл", "балла")

recent\_db\_user = await session.get(DBUser, db\_user.user\_id)

recent\_db\_user.points = recent\_db\_user.points + callback\_data.points

db\_user\_event\_game = await session.get(DBUserEventGame, (db\_user.user\_id, db\_event\_game.event\_id))

db\_user\_event\_game.stops\_done = True

await session.commit()

await query.message.edit\_text(

f"{html.bold(db\_event\_game.event\_title)}\n\n"

f"Вы успешно прошли контрольные точки мероприятия и получили за это {html.bold(f"{callback\_data.points} {ball}")}! После завершения мероприятия не забудьте ответить на вопросы: /questions.")

else:

inline\_keyboard\_builder = InlineKeyboardBuilder()

inline\_keyboard\_builder.button(text="❌ Не прошёл", callback\_data=EventGameStopCallback(

event\_id=callback\_data.event\_id, stop\_index=callback\_data.stop\_index+1, points=callback\_data.points))

inline\_keyboard\_builder.button(text="✅ Прошёл", callback\_data=EventGameStopCallback(

event\_id=callback\_data.event\_id, stop\_index=callback\_data.stop\_index+1, points=callback\_data.points+1))

await query.message.edit\_text(

f"{html.bold(db\_event\_game.event\_title)}\n\n"

f"{stops[stop\_index]}", reply\_markup=inline\_keyboard\_builder.as\_markup())

@dp.message(Command("questions"))

async def questions\_command\_handler(message: Message, state: FSMContext, db\_user: DBUser):

if db\_user.role != "user":

await message.answer("Только обычные пользователи могут воспользоваться этой командой.")

return

sent\_message = await message.answer(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Сейчас мы получаем список мероприятий EXTRA.HSE, пожалуйста, подождите...")

await edit\_questions\_events\_message(sent\_message=sent\_message, db\_user=db\_user, page\_index=0)

async def edit\_questions\_events\_message(sent\_message: Message, db\_user: DBUser, page\_index: int):

async with async\_session() as session:

db\_event\_games: list[DBEventGame] = await db\_user.awaitable\_attrs.event\_games

db\_event\_games\_filtered: list[DBEventGame] = []

for db\_event\_game in db\_event\_games:

if (moscow\_tz.localize(db\_event\_game.end) + datetime.timedelta(days=10) >=

datetime.datetime.now(tz=moscow\_tz) >=

moscow\_tz.localize(db\_event\_game.end)

and

not (await session.get(DBUserEventGame, (db\_user.user\_id, db\_event\_game.event\_id))).questions\_done):

db\_event\_games\_filtered.append(db\_event\_game)

db\_event\_games = db\_event\_games\_filtered

if len(db\_event\_games) == 0:

await sent\_message.edit\_text(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! Вы пока что не принимаете участие ни в каком мероприятии, в котором нужно отвечать на контрольные вопросы. Если Вы ответили на все контрольные вопросы всех мероприятий, на которые Вы записались, то Вы - большой молодец! /start"

)

return

db\_event\_games\_limited = db\_event\_games[page\_index\*5:page\_index\*5+5]

max\_pages = ((len(db\_event\_games) - 1) // 5 + 1) - 1

inline\_keyboard\_builder = InlineKeyboardBuilder()

for i in range(len(db\_event\_games\_limited)):

inline\_keyboard\_builder.button(text=f"{(page\_index \* 5) + i + 1}",

callback\_data=EventGameQuestionCallback(

event\_id=db\_event\_games\_limited[i].event\_id,

last\_answer\_was\_right=True,

question\_index=0,

points=0))

if page\_index != 0:

inline\_keyboard\_builder.button(text="◀️",

callback\_data=EventsGamesQuestionsChangePageCallback(

page=max(page\_index - 1, 0)))

if page\_index != max\_pages:

inline\_keyboard\_builder.button(text="▶️️",

callback\_data=EventsGamesQuestionsChangePageCallback(

page=min(page\_index + 1, max\_pages)))

inline\_keyboard\_builder.adjust(len(db\_event\_games\_limited), int(page\_index != 0) + int(page\_index != max\_pages))

await sent\_message.edit\_text(

f"Здравствуйте, {html.bold(db\_user.full\_name)}! "

f"Пожалуйста, выберите мероприятие, контрольные вопросы которого Вы хотите пройти:\n\n"

f"{"\n\n".join([f"{db\_event\_games.index(db\_event\_game) + 1}) " +

html.bold(db\_event\_game.event\_title)

for db\_event\_game in db\_event\_games\_limited])}\n\n"

f"Нажмите на любую из кнопок ниже, чтобы начать отвечать на контрольные вопросы.",

reply\_markup=inline\_keyboard\_builder.as\_markup())

@dp.callback\_query(EventsGamesQuestionsChangePageCallback.filter())

async def events\_games\_questions\_change\_page\_callback(query: CallbackQuery, callback\_data: EventsGamesQuestionsChangePageCallback,

db\_user: DBUser):

await edit\_questions\_events\_message(

sent\_message=query.message,

db\_user=db\_user,

page\_index=callback\_data.page)

await query.answer()

@dp.callback\_query(EventGameQuestionCallback.filter())

async def event\_game\_stop\_callback(query: CallbackQuery, callback\_data: EventGameQuestionCallback, db\_user: DBUser):

async with async\_session() as session:

db\_event\_game = await db\_get\_event\_game(session, callback\_data.event\_id)

questions: list[list[str]] = db\_event\_game.questions

question\_index = callback\_data.question\_index

if question\_index != 0:

await query.answer(f"✅ Правильно!\n\n" if callback\_data.last\_answer\_was\_right else "❌ Неправильно :(\n\n")

if callback\_data.question\_index >= len(questions):

ball = declension(callback\_data.points, "баллов", "балл", "балла")

recent\_db\_user = await session.get(DBUser, db\_user.user\_id)

recent\_db\_user.points = recent\_db\_user.points + callback\_data.points

db\_user\_event\_game = await session.get(DBUserEventGame, (db\_user.user\_id, db\_event\_game.event\_id))

db\_user\_event\_game.questions\_done = True

await session.commit()

await query.message.edit\_text(

f"{html.bold(db\_event\_game.event\_title)}\n\n"

f"{"✅ Правильно!\n\n" if callback\_data.last\_answer\_was\_right else "❌ Неправильно :(\n\n"}"

f"Вы успешно ответили на все контрольные вопросы мероприятия и получили за это {html.bold(f"{callback\_data.points} {ball}")}! Посмотрите, какие мероприятия Вас ещё могут заинтересовать: /start.")

else:

question = questions[question\_index]

inline\_keyboard\_builder = InlineKeyboardBuilder()

buttons = [

InlineKeyboardButton(text=question[1], callback\_data=EventGameQuestionCallback(

event\_id=callback\_data.event\_id, question\_index=callback\_data.question\_index+1,

last\_answer\_was\_right=True, points=callback\_data.points+1

).pack()),

InlineKeyboardButton(text=question[2], callback\_data=EventGameQuestionCallback(

event\_id=callback\_data.event\_id, question\_index=callback\_data.question\_index+1,

last\_answer\_was\_right=False, points=callback\_data.points

).pack()),

InlineKeyboardButton(text=question[3], callback\_data=EventGameQuestionCallback(

event\_id=callback\_data.event\_id, question\_index=callback\_data.question\_index+1,

last\_answer\_was\_right=False, points=callback\_data.points

).pack()),

]

random.shuffle(buttons)

for button in buttons:

inline\_keyboard\_builder.add(button)

inline\_keyboard\_builder.adjust(1, 1, 1)

await query.message.edit\_text(

f"{html.bold(db\_event\_game.event\_title)}\n\n"

f"{("✅ Правильно!\n\n" if callback\_data.last\_answer\_was\_right else "❌ Неправильно :(\n\n") if callback\_data.question\_index != 0 else ""}"

f"{question[0]}", reply\_markup=inline\_keyboard\_builder.as\_markup())

@dp.message(F.text)

async def message\_handler(message: Message):

await message.answer("Что-то я Вас не понял. Выберите команду из меню или пропишите /start.")

async def start\_bot():

commands = [

BotCommand(command='start', description='Старт'),

BotCommand(command='cancel', description='Отмена действия'),

BotCommand(command='me', description='Посмотреть профиль'),

BotCommand(command='stops', description="Начать прохождение контрольных точек"),

BotCommand(command='questions', description="Начать отвечать на вопросы")

]

await bot.set\_my\_commands(commands)

with open(csv\_file\_name, "w") as csv\_file:

csv\_file.write("update\_id,user\_id,datetime,event\_type,data\n")

async def main() -> None:

await create\_tables()

dp.startup.register(start\_bot)

scheduler = AsyncIOScheduler(timezone='Europe/Moscow')

job = scheduler.add\_job(send\_notifications, 'interval', minutes=1)

scheduler.start()

dp.update.outer\_middleware(RegistrationMiddleware())

await dp.start\_polling(bot)

await bot.session.close()

scheduler.remove\_job(job.id)

driver.quit()

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

logging.basicConfig(level=logging.INFO, stream=sys.stdout)

asyncio.run(main())

**misc.py**

import json

from sqlalchemy import types

class DBJSON(types.TypeDecorator):

impl = types.String

cache\_ok = True

def process\_bind\_param(self, value, dialect):

return json.dumps(value)

def process\_result\_value(self, value, dialect):

return json.loads(value)

# число, "0 друзей", "1 друг", "2 друга"

def declension(n, form\_0, form\_1, form\_2):

units = n % 10

tens = (n // 10) % 10

if tens == 1:

return form\_0

if units in [0, 5, 6, 7, 8, 9]:

return form\_0

if units == 1:

return form\_1

if units in [2, 3, 4]:

return form\_2

**tokens.py**

import os

from dotenv import load\_dotenv

load\_dotenv()

tg\_token = os.getenv('TG\_TOKEN')

gigachat\_token = os.getenv("GIGACHAT\_TOKEN")

**db.py**

import datetime

from sqlalchemy import BigInteger, ForeignKey, String, Integer, Boolean, DateTime

from sqlalchemy.exc import SQLAlchemyError

from sqlalchemy.ext.asyncio import AsyncSession, create\_async\_engine, async\_sessionmaker, AsyncAttrs

from sqlalchemy.orm import Mapped, mapped\_column, relationship, DeclarativeBase

from misc import DBJSON

class Base(AsyncAttrs, DeclarativeBase):

pass

engine = create\_async\_engine("sqlite+aiosqlite:///db.db")

async\_session = async\_sessionmaker(engine, class\_=AsyncSession, expire\_on\_commit=False)

async def create\_tables():

async with engine.begin() as conn:

await conn.run\_sync(Base.metadata.create\_all)

class DBUser(Base):

\_\_tablename\_\_ = "Users"

user\_id: Mapped[int] = mapped\_column(BigInteger, primary\_key=True)

full\_name: Mapped[str] = mapped\_column(String)

role: Mapped[str] = mapped\_column(String)

points: Mapped[int] = mapped\_column(Integer)

event\_games = relationship("DBEventGame", secondary="UserEventGames", back\_populates="users")

class DBEventGame(Base):

\_\_tablename\_\_ = "EventGames"

event\_id: Mapped[str] = mapped\_column(String, primary\_key=True)

event\_title: Mapped[str] = mapped\_column(String)

stops: Mapped[DBJSON] = mapped\_column(DBJSON)

questions: Mapped[DBJSON] = mapped\_column(DBJSON)

start: Mapped[datetime.datetime] = mapped\_column(DateTime)

end: Mapped[datetime.datetime] = mapped\_column(DateTime)

users = relationship("DBUser", secondary="UserEventGames", back\_populates="event\_games")

class DBUserEventGame(Base):

\_\_tablename\_\_ = "UserEventGames"

user\_id: Mapped[int] = mapped\_column(BigInteger, ForeignKey('Users.user\_id'), primary\_key=True)

event\_id: Mapped[str] = mapped\_column(String, ForeignKey('EventGames.event\_id'), primary\_key=True)

pre\_start\_notified: Mapped[bool] = mapped\_column(Boolean)

start\_notified: Mapped[bool] = mapped\_column(Boolean)

end\_notified: Mapped[bool] = mapped\_column(Boolean)

stops\_done: Mapped[bool] = mapped\_column(Boolean)

questions\_done: Mapped[bool] = mapped\_column(Boolean)

async def db\_add\_user(session: AsyncSession, user\_id: int, user\_full\_name: str, user\_role: str):

db\_user = DBUser()

db\_user.user\_id = user\_id

db\_user.full\_name = user\_full\_name

db\_user.role = user\_role

db\_user.points = 0

session.add(db\_user)

await session.commit()

async def db\_add\_event\_game(session: AsyncSession, db\_event\_game: DBEventGame):

session.add(db\_event\_game)

await session.commit()

async def db\_add\_event\_game\_to\_user(session: AsyncSession, user\_id: int, event\_id: str):

db\_user\_event\_game = DBUserEventGame()

db\_user\_event\_game.user\_id = user\_id

db\_user\_event\_game.event\_id = event\_id

db\_user\_event\_game.pre\_start\_notified = False

db\_user\_event\_game.start\_notified = False

db\_user\_event\_game.end\_notified = False

db\_user\_event\_game.stops\_done = False

db\_user\_event\_game.questions\_done = False

session.add(db\_user\_event\_game)

await session.commit()

async def db\_get\_user(session: AsyncSession, user\_id: int) -> DBUser | None:

try:

db\_user = await session.get(DBUser, user\_id)

if not db\_user:

return None

return await session.get(DBUser, user\_id)

except SQLAlchemyError:

return None

async def db\_get\_event\_game(session: AsyncSession, event\_id: str) -> DBEventGame | None:

try:

db\_event\_game = await session.get(DBEventGame, event\_id)

if not db\_event\_game:

return None

return await session.get(DBEventGame, event\_id)

except SQLAlchemyError:

return None

**classes.py**

import datetime

from aiogram.filters.callback\_data import CallbackData

from icalendar.cal import Calendar

class RoughEvent:

id: str

title: str

date: datetime.datetime

class Event:

id: str

title: str

rating: str

ical: Calendar

description: str

address: str

class EventsMessageChangePageCallback(CallbackData, prefix="events\_message\_page"):

page: int

class EventInfoCallback(CallbackData, prefix="event\_info"):

event\_id: str

from\_page: int

class JoinEventGameCallback(CallbackData, prefix="join\_event\_game"):

event\_id: str

class CreateEventGameCallback(CallbackData, prefix="create\_event\_game"):

event\_id: str

class EventGameStopCallback(CallbackData, prefix="event\_game\_stop"):

event\_id: str

stop\_index: int

points: int

class EventsGamesStopsChangePageCallback(CallbackData, prefix="events\_games\_stops\_page"):

page: int

class EventsGamesQuestionsChangePageCallback(CallbackData, prefix="events\_games\_qs\_page"):

page: int

class EventGameQuestionCallback(CallbackData, prefix="event\_game\_q"):

event\_id: str

question\_index: int

last\_answer\_was\_right: bool

points: int

**browser.py**

from urllib.request import urlopen

import dateparser

import icalendar

from selenium import webdriver

from selenium.webdriver.common.by import By

from classes import Event, RoughEvent

from db import async\_session, db\_get\_event\_game

options = webdriver.ChromeOptions()

options.add\_argument('--headless')

options.add\_argument('--no-sandbox')

options.add\_argument('--disable-dev-shm-usage')

driver = webdriver.Chrome(options=options)

def get\_event\_from\_internet(event\_id: str) -> Event:

driver.get(f"https://extra.hse.ru/announcements/{event\_id}.html")

event\_html = driver.find\_element(by=By.CLASS\_NAME, value="post")

event = Event()

event.id = event\_id

event.title = event\_html.find\_element(by=By.CLASS\_NAME, value="post\_single").text

event.rating = event\_html.find\_element(by=By.CLASS\_NAME, value="rating-round").text

event.ical = \

icalendar.Calendar.from\_ical(urlopen(f"https://extra.hse.ru/events/ics/{event\_id}.ics").read()).walk("VEVENT")[

0]

event.description = event\_html.find\_element(by=By.CLASS\_NAME, value="post\_\_text").text.replace(

"Добавить в календарь", "").strip()

event.address = event\_html.find\_elements(by=By.CLASS\_NAME, value="articleMetaItem")[1].find\_element(

by=By.CLASS\_NAME, value="articleMetaItem\_\_content").text

return event

# rough - грубые, то есть не совсем полная информация

async def get\_rough\_events\_from\_internet(with\_games: bool):

async with async\_session() as session:

driver.get("https://extra.hse.ru/news/announcements/")

driver.implicitly\_wait(0.5)

events\_html = driver.find\_elements(by=By.CLASS\_NAME, value="b-events")

events = []

for event\_html in events\_html:

event = RoughEvent()

event.id = event\_html.find\_element(by=By.CLASS\_NAME, value="b-events\_\_body\_title").find\_element(by=By.TAG\_NAME,

value="a").get\_attribute(

name="href").split("/")[-1].split(".")[0]

event.title = event\_html.find\_element(by=By.CLASS\_NAME, value="b-events\_\_body\_title").find\_element(

by=By.TAG\_NAME, value="a").text

event\_date\_text = event\_html.find\_element(by=By.CLASS\_NAME, value="b-events\_\_title").find\_element(

by=By.CLASS\_NAME, value="title").text

event.date = dateparser.parse(

event\_date\_text.split(",")[0] if event\_date\_text[0].isalpha() else event\_date\_text).replace(hour=0,

minute=0,

second=0,

microsecond=0)

db\_event\_game = await db\_get\_event\_game(session=session, event\_id=event.id)

if (db\_event\_game is not None) == with\_games:

events.append(event)

return events

**ai.py**

from langchain\_community.chat\_models import GigaChat

from langchain\_core.messages import SystemMessage, HumanMessage

from tokens import gigachat\_token

giga = GigaChat(credentials=gigachat\_token,

model='GigaChat:latest',

verify\_ssl\_certs=False

)

def get\_questions\_from\_gigachat(title: str, description: str):

system\_message = SystemMessage(content='Когда пользователь отправит тебе название и описание мероприятия, '

'ты должен будешь составить 5 контрольных вопросов. '

'Формат каждого контрольного вопроса: '

'\"Контрольный вопрос\\nОтвет 1\\nОтвет 2\\nОтвет 3\". '

'Учти, что самый первый ответ должен быть правильным, '

'а остальные - нет. Пять вопросов. '

'Учти, что эти вопросы будут задаваться после прохождения '

'мероприятия. Будь креативным.')

response = giga([

system\_message,

HumanMessage(f"Название: {title}, описание: {description}")

])

print(response.content)

return response.content

def get\_stops\_from\_gigachat(title: str, description: str):

system\_message = SystemMessage(content='Когда пользователь отправит тебе название и описание мероприятия, '

'ты должен будешь составить 5 контрольных точек этого мероприятия. '

'Это может быть поговорить с каким-либо экспертом либо что-то подобное. '

'Формат каждой контрольный точки: '

'\"Контрольная точка\". Пять контрольных точек.'

'Каждая контрольная точка - с новой строки.'

'Будь креативным. Нумеруй контрольные точки.')

response = giga([

system\_message,

HumanMessage(f"Название: {title}, описание: {description}")

])

print(response.content)

return response.content

**requirements.txt**

aiofiles==24.1.0

aiogram==3.17.0

aiohappyeyeballs==2.4.4

aiohttp==3.11.11

aiosignal==1.3.2

aiosqlite==0.20.0

annotated-types==0.7.0

anyio==4.8.0

APScheduler==3.11.0

attrs==24.3.0

certifi==2024.12.14

charset-normalizer==3.4.1

dataclasses-json==0.6.7

dateparser==1.2.0

frozenlist==1.5.0

gigachat==0.1.37.post1

greenlet==3.1.1

h11==0.14.0

httpcore==1.0.7

httpx==0.27.2

httpx-sse==0.4.0

icalendar==6.1.0

idna==3.10

jsonpatch==1.33

jsonpointer==3.0.0

langchain==0.3.14

langchain-community==0.3.14

langchain-core==0.3.29

langchain-text-splitters==0.3.5

langsmith==0.2.10

magic-filter==1.0.12

marshmallow==3.25.1

multidict==6.1.0

mypy-extensions==1.0.0

nest-asyncio==1.6.0

numpy==2.2.1

orjson==3.10.14

outcome==1.3.0.post0

packaging==24.2

propcache==0.2.1

pydantic==2.10.5

pydantic-settings==2.7.1

pydantic\_core==2.27.2

PySocks==1.7.1

python-dateutil==2.9.0.post0

python-dotenv==1.0.1

pytz==2024.2

PyYAML==6.0.2

regex==2024.11.6

requests==2.32.3

requests-toolbelt==1.0.0

selenium==4.27.1

six==1.17.0

sniffio==1.3.1

sortedcontainers==2.4.0

SQLAlchemy==2.0.37

tenacity==9.0.0

trio==0.28.0

trio-websocket==0.11.1

typing-inspect==0.9.0

typing\_extensions==4.12.2

tzdata==2024.2

tzlocal==5.2

urllib3==2.3.0

websocket-client==1.8.0

wsproto==1.2.0

yarl==1.18.3

**Дополнительная информация**

Код на GitHub: <https://github.com/vyacheslavbytsko/HSE-Extra-Events-Bot>

Юзернейм в Telegram: <https://t.me/hse_extra_events_bot>

Время доступности: 18.01.2025-19.01.2025 оба дня 24 часа в сутки