|  |  |  |
| --- | --- | --- |
| Защищено:  Гапанюк Ю. Е.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |  | Демонстрация:  Гапанюк Ю. Е.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

**Отчет по Домашнему заданию №1 по курсу**

**«Парадигмы и конструкции языков программирования»**

#### "Написание телеграмм-бота с конечным автоматом на языке Python"

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: | Агапова Анна Денисовна |
| студент группы  ИУ5Ц-52Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Гапанюк Ю.Е. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

Москва, МГТУ 2024

**СОДЕРЖАНИЕ**

[Практическое задание 3](#_Toc182337810)

[Листинг программы 4](#_Toc182337811)

[База данных SQLite 8](#_Toc182337812)

[Результат работы программы 9](#_Toc182337813)

[Регистрация 9](#_Toc182337814)

[Заказ пиццы 9](#_Toc182337815)

## Практическое задание

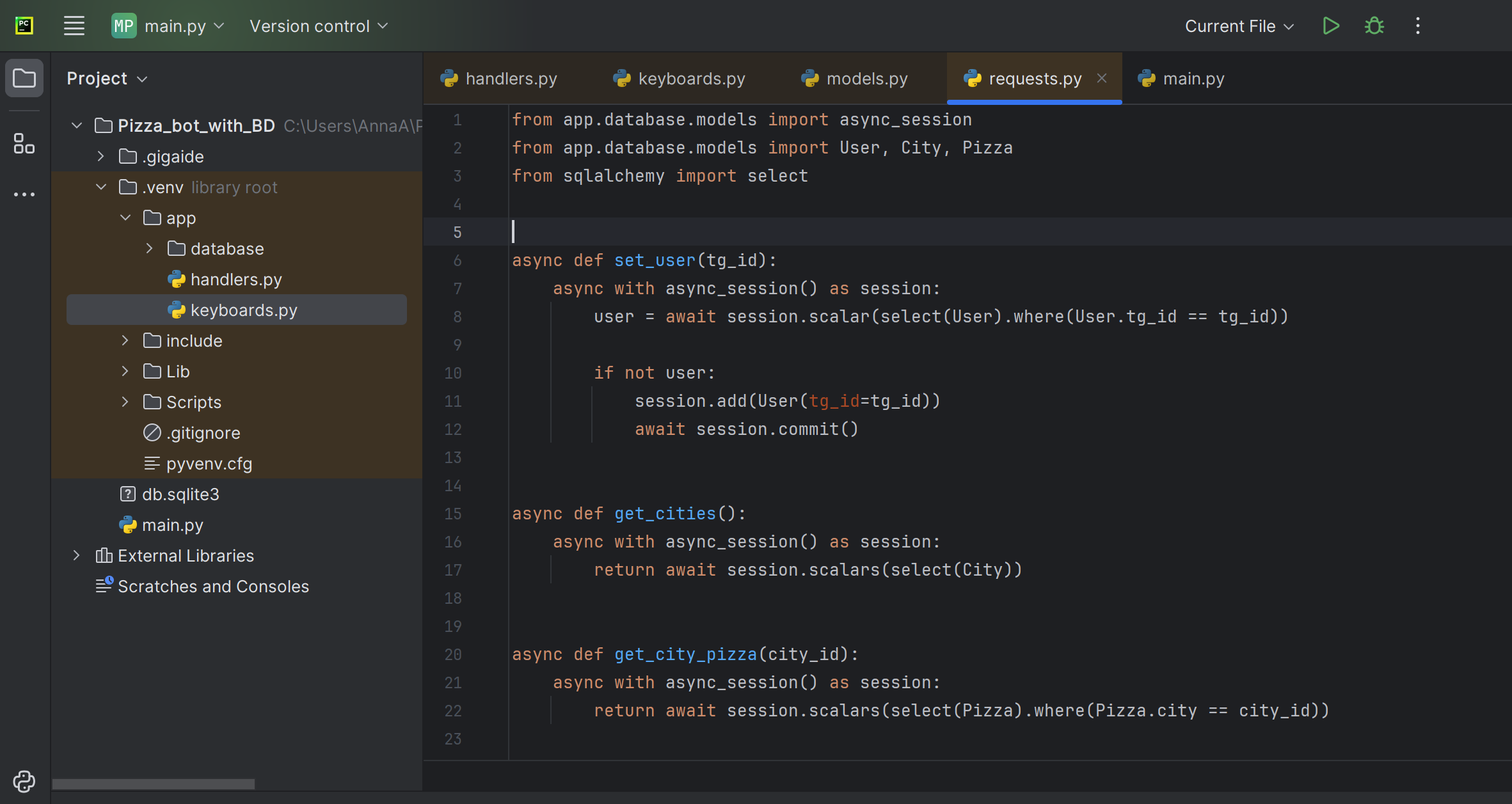
Разработайте бота для Telegram. Бот должен реализовывать конечный автомат из трех состояний.

В качестве темы для телеграмм-бота я решила взять свою АИС «Круглосуточная пицца на дом» по Моделям данных.

Для демонстрации работы с библиотекой fsm из aiogram я решила сделать регистрацию пользователя. Состояния: ввод ФИО, ввод адреса, ввод номера телефона

## Листинг программы

**Организация файлов в проекте**



main.py

import asyncio  
from aiogram import Bot, Dispatcher  
  
from app.handlers import router  
from app.database.models import async\_main  
  
async def main():  
 await async\_main()  
 bot = Bot(token='7655047518:AAGxqI4m2Fr07UNYGiaURGoi9baGjyH4KUk')  
 dp = Dispatcher()  
 dp.include\_router(router)  
 await dp.start\_polling(bot)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 try:  
 asyncio.run(main())  
 except KeyboardInterrupt:  
 print('Бот выключен')

handlers.py

from aiogram import F, Router  
from aiogram.types import Message, CallbackQuery  
from aiogram.filters import CommandStart  
  
from aiogram.fsm.state import State, StatesGroup  
from aiogram.fsm.context import FSMContext  
  
import app.keyboards as kb  
import app.database.requests as rq  
  
router = Router()  
  
class Register(StatesGroup):  
 name = State()  
 adress = State()  
 number = State()  
  
@router.message(CommandStart())  
async def cmd\_start(message: Message, state: FSMContext):  
 await rq.set\_user(message.from\_user.id)  
 await message.answer('Добро пожаловать в чат-бот для АИС! \n '  
 'Зарегистрируйтесь, пожалуйста', reply\_markup=kb.main)  
 await state.set\_state(Register.name)  
 await message.answer('Введите своё ФИО')  
  
@router.message(Register.name)  
async def register\_name(message: Message, state: FSMContext):  
 await state.update\_data(name=message.text)  
 await state.set\_state(Register.adress)  
 await message.answer('Введите свой адрес')  
  
@router.message(Register.adress)  
async def register\_adress(message: Message, state: FSMContext):  
 await state.update\_data(adress=message.text)  
 await state.set\_state(Register.number)  
 await message.answer('Отправьте номер телефона, чтобы курьер мог связатьсяс Вами ', reply\_markup=kb.contact)  
  
@router.message(Register.number, F.contact)  
async def register\_number(message: Message, state: FSMContext):  
 await state.update\_data(number=message.contact.phone\_number)  
 data = await state.get\_data()  
 await message.answer(f'Проверьте свои данные\nВаше имя: {data["name"]}\nВаш адрес: '  
 f'{data["adress"]}\nНомер телефона: {data["number"]}', reply\_markup=kb.main)  
 await state.clear()  
  
@router.message(F.text == 'Пиццерия в Вашем городе')  
async def catalog(message: Message):  
 await message.answer('Выберите Ваш город', reply\_markup=await kb.cities())  
  
@router.message(F.text == 'Меню')  
async def menu(message: Message):  
 await message.answer('Сначала выберите свой город', reply\_markup=await kb.cities())  
  
@router.message(F.text == 'О проекте')  
async def about(message: Message):  
 await message.answer('В нашей пиццерии вы можете заказать пиццу, которую вам не привезут!\n'  
 'Это просто чат-бот :)\n'  
 'И вообще пицца - это не полезно.\n'  
 'А вот учить иностранные языки - полезно!\n'  
 'Поэтому сейчас, когда мы уже добились Вашего внимания, '  
 'представляем наш новый бот-ПЕРЕВОДЧИК!\n'  
 'ссылка на чат-бот: t.me/Dream\_translator\_bot')  
  
@router.message(F.text == 'Контакты')  
async def contacts(message: Message):  
 await message.answer('Наши контакты\n\n'  
 'Тел.: +7(495)324-96-26\n'  
 'Адрес: ул. Госпитальная, 10')  
  
@router.callback\_query(F.data.startswith('city\_'))  
async def city(callback: CallbackQuery):  
 await callback.answer('Вы выбрали город')  
 await callback.message.answer('Выберите параметры заказа',  
 reply\_markup=await kb.pizzas(callback.data.split('\_')[1]))  
  
  
@router.callback\_query(F.data.startswith('pizza\_'))  
async def pizza(callback: CallbackQuery):  
 order\_data = await rq.get\_pizza(callback.data.split('\_')[1])  
 await callback.answer('Вы выбрали пиццу')  
 await callback.message.answer(f'Пицца: {order\_data.pizza\_name}\n'  
 f'Размер: {order\_data.pizza\_size}\nЦена: {order\_data.price} руб\n'  
 f'Состав: {order\_data.description}\nЗаказ Вам доставит '  
 f'{order\_data.courier}\nТелефон для связи с курьером: {order\_data.tel}')  
  
 #Дата: {order\_data.date}\n

app/models.py

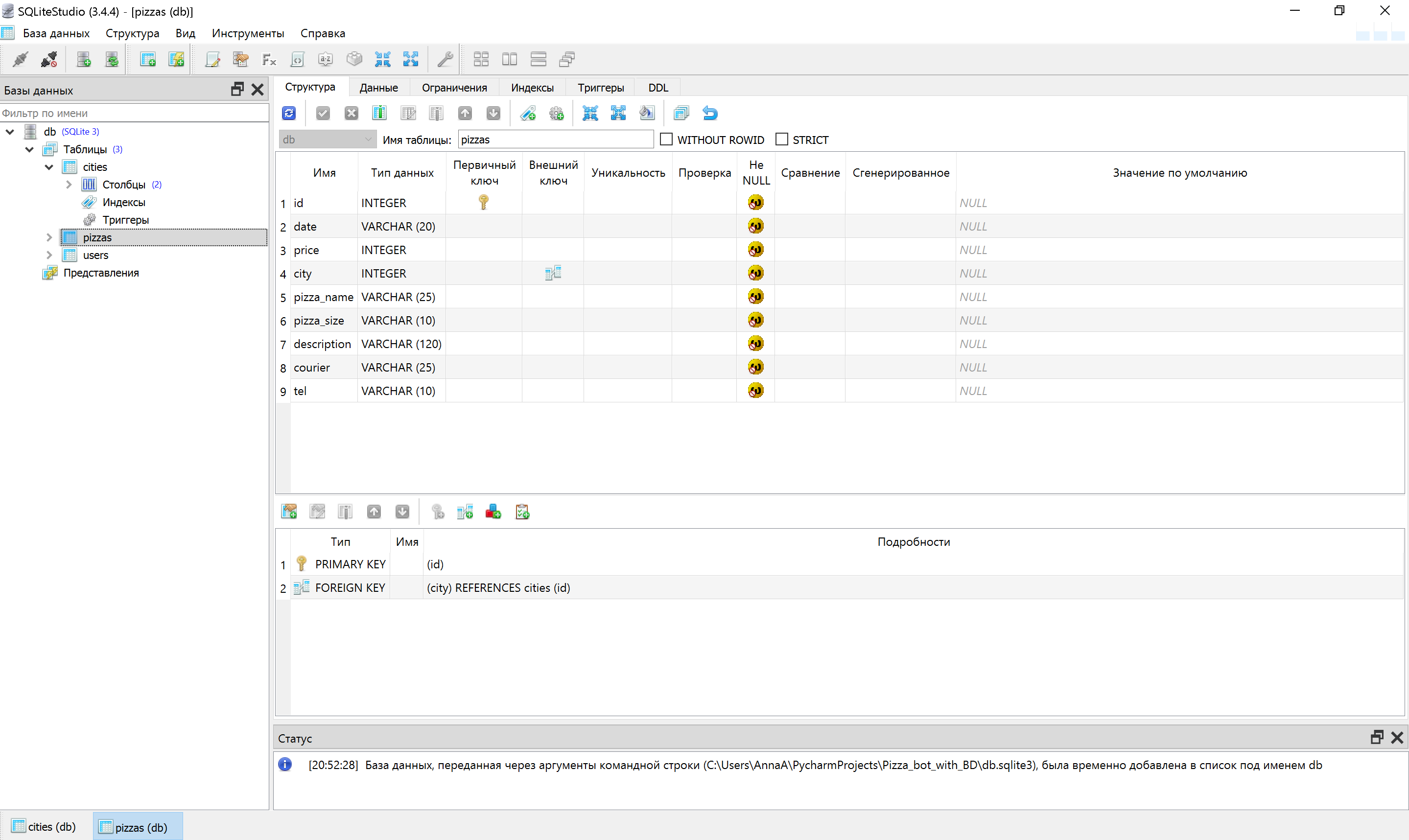
from sqlalchemy import BigInteger, String, ForeignKey  
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped\_column  
from sqlalchemy.ext.asyncio import AsyncAttrs, async\_sessionmaker, create\_async\_engine  
  
engine = create\_async\_engine(url='sqlite+aiosqlite:///db.sqlite3')  
  
async\_session = async\_sessionmaker(engine)  
  
  
class Base(AsyncAttrs, DeclarativeBase):  
 pass  
  
class User(Base):  
 \_\_tablename\_\_ = 'users'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 tg\_id = mapped\_column(BigInteger)  
  
  
class City(Base):  
 \_\_tablename\_\_ = 'cities'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 name: Mapped[str] = mapped\_column(String(25))  
  
class Pizza(Base):  
 \_\_tablename\_\_ = 'pizzas'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 date: Mapped[str] = mapped\_column(String(20)) # тип данных Дата datetime.date() ?  
 price: Mapped[int] = mapped\_column()  
 city: Mapped[int] = mapped\_column(ForeignKey('cities.id'))  
 pizza\_name: Mapped[str] = mapped\_column(String(25))  
 pizza\_size: Mapped[str] = mapped\_column(String(10))  
 description: Mapped[str] = mapped\_column(String(120))  
 courier: Mapped[str] = mapped\_column(String(25))  
 tel: Mapped[str] = mapped\_column(String(10))  
  
async def async\_main():  
 async with engine.begin() as conn:  
 await conn.run\_sync(Base.metadata.create\_all)

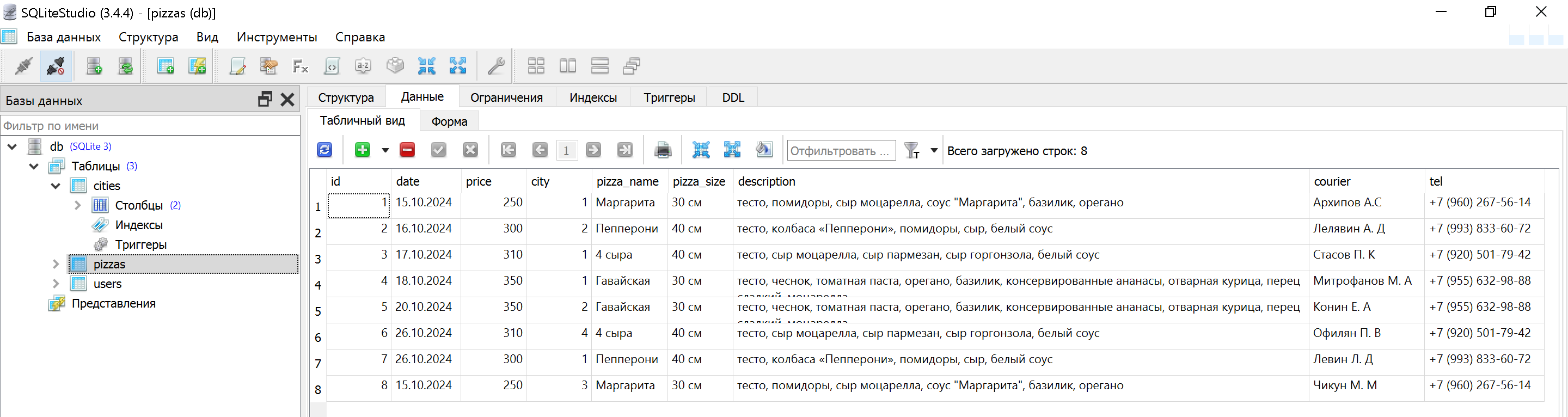
app/requests.py

from app.database.models import async\_session  
from app.database.models import User, City, Pizza  
from sqlalchemy import select  
  
  
async def set\_user(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if not user:  
 session.add(User(tg\_id=tg\_id))  
 await session.commit()  
  
  
async def get\_cities():   
 async with async\_session() as session:  
 return await session.scalars(select(City))  
  
  
async def get\_city\_pizza(city\_id):   
 async with async\_session() as session:  
 return await session.scalars(select(Pizza).where(Pizza.city == city\_id))  
  
  
async def get\_pizza(pizza\_id):  
 async with async\_session() as session:  
 return await session.scalar(select(Pizza).where(Pizza.id == pizza\_id))

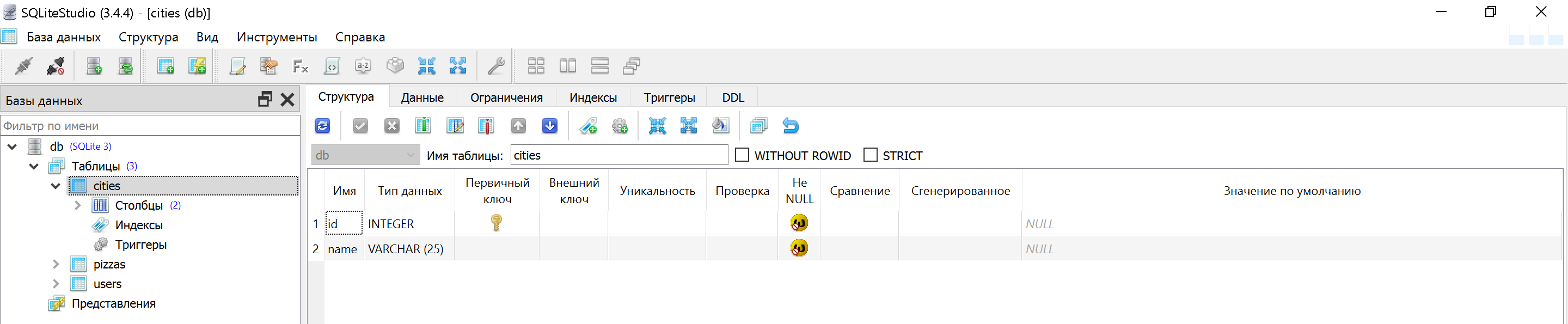
## База данных SQLite

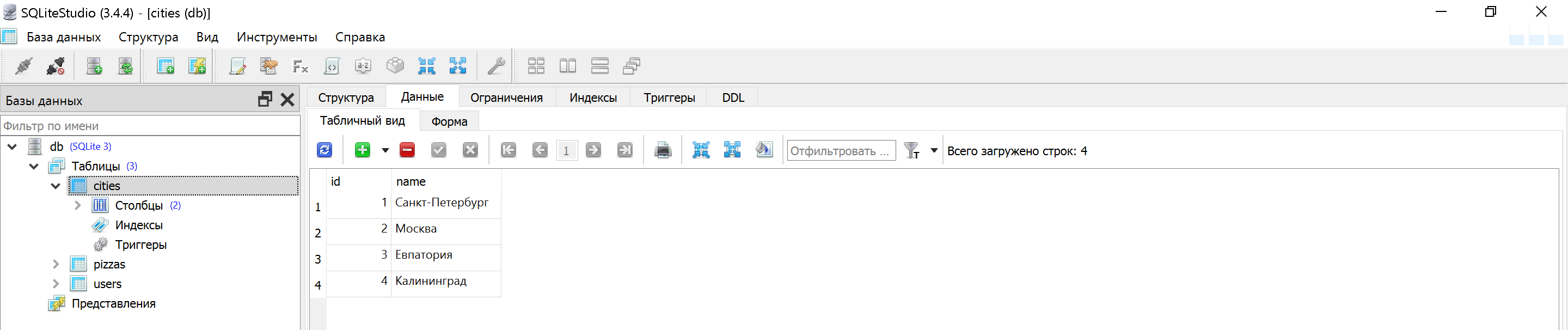
Пиццы





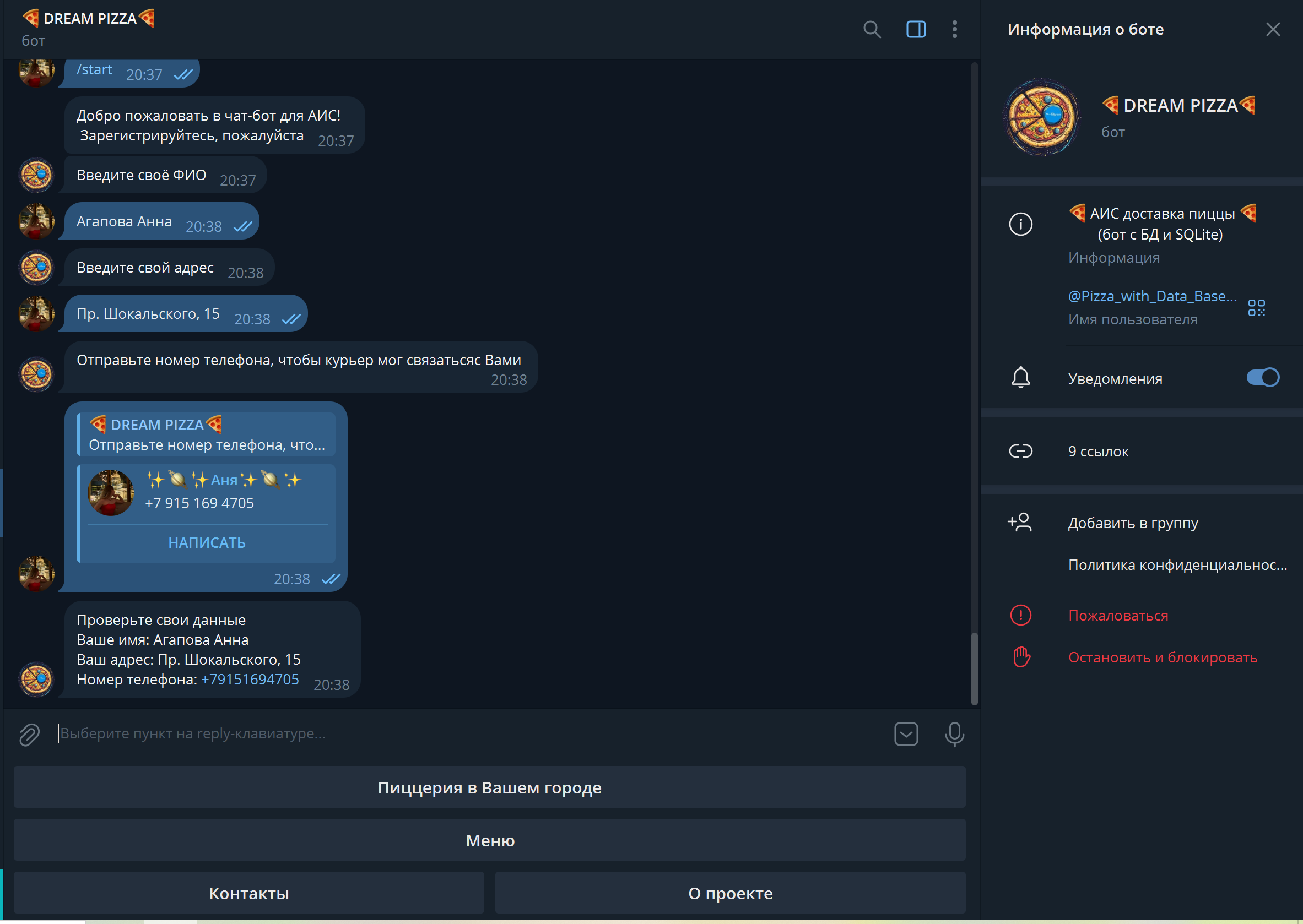
Города





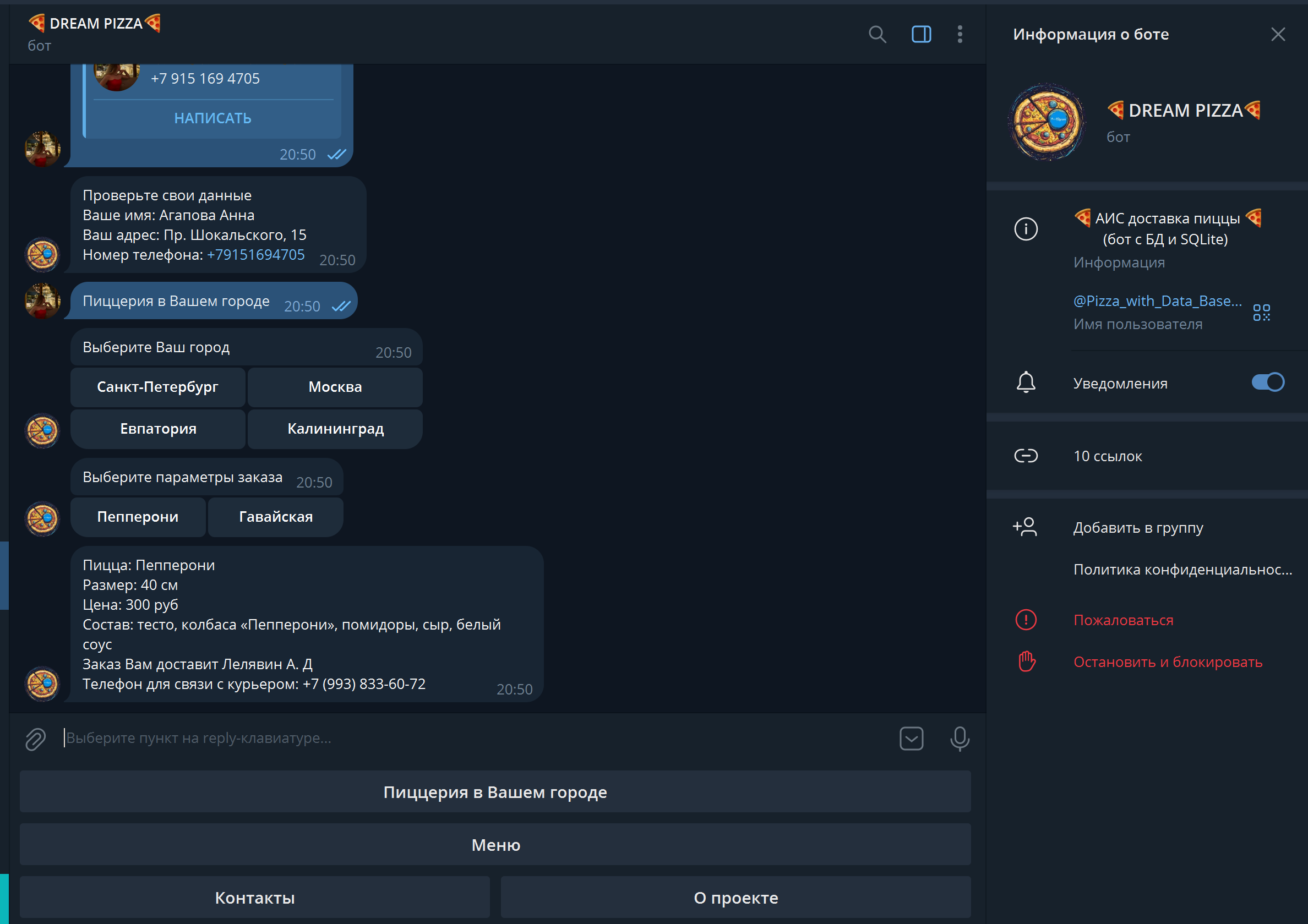
## Результат работы программы

### Регистрация



### Заказ пиццы

Сначала нужно выбрать свой город



Проверка работы reply-клавиатуры

