# Список Python-файлов

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\main.py

import asyncio  
from sqlalchemy.orm import Session  
from app.bot.bot\_runner import main  
from app.core.database.helper import SessionLocal  
from app.core.models.city import City  
from app.core.services.city import create\_city  
from app.core.schemas.city import CityCreate  
  
def init\_db():  
 """Инициализация базы данных: добавление города Кокшетау, если таблица cities пуста."""  
 with SessionLocal() as session:  
 try:  
 if not session.query(City).first():  
 city\_data = CityCreate(name="Кокшетау")  
 create\_city(session, city\_data)  
 session.commit()  
 print("Город 'Кокшетау' добавлен в базу данных.")  
 else:  
 print("База данных уже содержит города, инициализация не требуется.")  
 except Exception as e:  
 print(f"Ошибка при инициализации базы данных: {e}")  
 session.rollback()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Сначала инициализируем базу данных  
 init\_db()  
 # Затем запускаем бота  
 asyncio.run(main())

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\env.py

from logging.config import fileConfig  
from sqlalchemy import engine\_from\_config, pool  
from alembic import context  
from app.core.config import DB\_URL  
from app.core.models.base import Base  
from app.core.models.user import User  
from app.core.models.association import user\_categories  
from app.core.models.category import Category  
from app.core.models.city import City  
from app.core.models.offer import Offer  
from app.core.models.order import Order  
from app.core.models.review import Review  
  
# Настраиваем логирование  
if context.config.config\_file\_name is not None:  
 fileConfig(context.config.config\_file\_name)  
  
# Указываем метаданные для автогенерации  
target\_metadata = Base.metadata  
  
def run\_migrations\_offline() -> None:  
 """Запуск миграций в оффлайн-режиме."""  
 url = DB\_URL # Используем DB\_URL из config.py  
 context.configure(  
 url=url,  
 target\_metadata=target\_metadata,  
 literal\_binds=True,  
 dialect\_opts={"paramstyle": "named"},  
 )  
 with context.begin\_transaction():  
 context.run\_migrations()  
  
def run\_migrations\_online() -> None:  
 """Запуск миграций в онлайн-режиме."""  
 connectable = engine\_from\_config(  
 {"sqlalchemy.url": DB\_URL}, # Используем DB\_URL напрямую  
 prefix="sqlalchemy.",  
 poolclass=pool.NullPool,  
 )  
 with connectable.connect() as connection:  
 context.configure(  
 connection=connection,  
 target\_metadata=target\_metadata,  
 )  
 with context.begin\_transaction():  
 context.run\_migrations()  
  
if context.is\_offline\_mode():  
 run\_migrations\_offline()  
else:  
 run\_migrations\_online() # Добавлен вызов функции с правильным отступом

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\02487aa00d87\_add\_start\_date\_to\_offer.py

"""Add start\_date to Offer  
  
Revision ID: 02487aa00d87  
Revises: ec634b458093  
Create Date: 2025-03-12 12:31:35.912972  
  
"""  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = '02487aa00d87'  
down\_revision: Union[str, None] = 'ec634b458093'  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.add\_column('offers', sa.Column('start\_date', sa.DateTime(), nullable=True))  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.drop\_column('offers', 'start\_date')  
 # ### end Alembic commands ###

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_15\_2145-4b861d6e9719\_add\_status\_to\_orders\_and\_remove\_is\_.py

"""add status to orders and remove is\_completed  
  
Revision ID: 4b861d6e9719  
Revises: 02487aa00d87  
Create Date: 2025-03-15 21:45:30.776605  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
# revision identifiers, used by Alembic.  
revision: str = "4b861d6e9719"  
down\_revision: Union[str, None] = "02487aa00d87"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
# Определение ENUM-типа с значениями в верхнем регистре  
orderstatus = sa.Enum("PENDING", "IN\_PROGRESS", "COMPLETED", "CANCELED", name="orderstatus")  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # Создание типа orderstatus перед добавлением колонки  
 orderstatus.create(op.get\_bind(), checkfirst=True)  
  
 # Добавление колонки status с типом orderstatus и значением по умолчанию  
 op.add\_column(  
 "orders",  
 sa.Column(  
 "status",  
 orderstatus,  
 nullable=False,  
 server\_default="PENDING" # Значение по умолчанию в верхнем регистре  
 ),  
 )  
  
 # Удаление колонки is\_completed  
 op.drop\_column("orders", "is\_completed")  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # Добавление обратно колонки is\_completed  
 op.add\_column(  
 "orders",  
 sa.Column(  
 "is\_completed", sa.BOOLEAN(), autoincrement=False, nullable=False  
 ),  
 )  
  
 # Удаление колонки status  
 op.drop\_column("orders", "status")  
  
 # Удаление типа orderstatus  
 orderstatus.drop(op.get\_bind(), checkfirst=True)

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_15\_2214-92fdddd0c03c\_create\_otder\_nullable\_true.py

"""Create otder nullable true  
  
Revision ID: 92fdddd0c03c  
Revises: 4b861d6e9719  
Create Date: 2025-03-15 22:14:44.580445  
  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
from sqlalchemy.dialects import postgresql  
  
# revision identifiers, used by Alembic.  
revision: str = "92fdddd0c03c"  
down\_revision: Union[str, None] = "4b861d6e9719"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.alter\_column(  
 "orders",  
 "status",  
 existing\_type=postgresql.ENUM(  
 "pending",  
 "in\_progress",  
 "completed",  
 "canceled",  
 name="orderstatus",  
 ),  
 nullable=True,  
 existing\_server\_default=sa.text("'pending'::orderstatus"),  
 )  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.alter\_column(  
 "orders",  
 "status",  
 existing\_type=postgresql.ENUM(  
 "pending",  
 "in\_progress",  
 "completed",  
 "canceled",  
 name="orderstatus",  
 ),  
 nullable=False,  
 existing\_server\_default=sa.text("'pending'::orderstatus"),  
 )  
 # ### end Alembic commands ###

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_15\_2254-b343e50c4c1a\_еще\_одна\_попытка\_перемиграции\_после\_.py

"""Еще одна попытка перемиграции после изменений  
  
Revision ID: b343e50c4c1a  
Revises: 92fdddd0c03c  
Create Date: 2025-03-15 22:54:41.125060  
  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = "b343e50c4c1a"  
down\_revision: Union[str, None] = "92fdddd0c03c"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 pass  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 pass  
 # ### end Alembic commands ###

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_15\_2327-0ed9e05ec864\_.py

"""empty message  
  
Revision ID: 0ed9e05ec864  
Revises: b343e50c4c1a  
Create Date: 2025-03-15 23:27:28.254177  
  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = "0ed9e05ec864"  
down\_revision: Union[str, None] = "b343e50c4c1a"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 pass  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 pass

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_15\_2328-423455fd3281\_изменены\_статусы\_на\_русский\_язык.py

"""Изменены статусы на русский язык  
  
Revision ID: 423455fd3281  
Revises: 0ed9e05ec864  
Create Date: 2025-03-15 23:28:31.488773  
  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = "423455fd3281"  
down\_revision: Union[str, None] = "0ed9e05ec864"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 pass  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 pass

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_16\_0152-7f9dd6a7c3de\_удалил\_всю\_api.py

"""Удалил всю API  
  
Revision ID: 7f9dd6a7c3de  
Revises: 423455fd3281  
Create Date: 2025-03-16 01:52:18.534663  
  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = "7f9dd6a7c3de"  
down\_revision: Union[str, None] = "423455fd3281"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 pass  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 pass  
 # ### end Alembic commands ###

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\2025\_03\_16\_0202-293a2255133d\_добавил\_исключение\_если\_в\_базе\_нет\_.py

"""Добавил исключение, если в базе нет городов  
  
Revision ID: 293a2255133d  
Revises: 7f9dd6a7c3de  
Create Date: 2025-03-16 02:02:26.614308  
  
"""  
  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = "293a2255133d"  
down\_revision: Union[str, None] = "7f9dd6a7c3de"  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 pass  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 pass  
 # ### end Alembic commands ###

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\alembic\versions\ec634b458093\_initial\_migration.py

"""Initial migration  
  
Revision ID: ec634b458093  
Revises:   
Create Date: 2025-03-10 07:46:03.885036  
  
"""  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = 'ec634b458093'  
down\_revision: Union[str, None] = None  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.create\_table('categories',  
 sa.Column('name', sa.String(), nullable=False),  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.PrimaryKeyConstraint('id'),  
 sa.UniqueConstraint('name')  
 )  
 op.create\_index(op.f('ix\_categories\_id'), 'categories', ['id'], unique=False)  
 op.create\_table('cities',  
 sa.Column('name', sa.String(), nullable=False),  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.PrimaryKeyConstraint('id'),  
 sa.UniqueConstraint('name')  
 )  
 op.create\_index(op.f('ix\_cities\_id'), 'cities', ['id'], unique=False)  
 op.create\_table('users',  
 sa.Column('telegram\_id', sa.Integer(), nullable=False),  
 sa.Column('name', sa.String(), nullable=False),  
 sa.Column('username', sa.String(), nullable=True),  
 sa.Column('is\_customer', sa.Boolean(), nullable=False),  
 sa.Column('is\_executor', sa.Boolean(), nullable=False),  
 sa.Column('is\_admin', sa.Boolean(), nullable=False),  
 sa.Column('city\_id', sa.Integer(), nullable=False),  
 sa.Column('rating', sa.Numeric(precision=2, scale=1), nullable=False),  
 sa.Column('completed\_orders', sa.Integer(), nullable=False),  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.ForeignKeyConstraint(['city\_id'], ['cities.id'], ondelete='CASCADE'),  
 sa.PrimaryKeyConstraint('id'),  
 sa.UniqueConstraint('telegram\_id'),  
 sa.UniqueConstraint('username')  
 )  
 op.create\_index(op.f('ix\_users\_id'), 'users', ['id'], unique=False)  
 op.create\_table('orders',  
 sa.Column('customer\_id', sa.Integer(), nullable=False),  
 sa.Column('executor\_id', sa.Integer(), nullable=True),  
 sa.Column('category\_id', sa.Integer(), nullable=False),  
 sa.Column('title', sa.String(), nullable=False),  
 sa.Column('description', sa.String(), nullable=True),  
 sa.Column('desired\_price', sa.Numeric(precision=10, scale=2), nullable=False),  
 sa.Column('due\_date', sa.DateTime(), nullable=False),  
 sa.Column('created\_at', sa.DateTime(), nullable=False),  
 sa.Column('is\_completed', sa.Boolean(), nullable=False),  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.ForeignKeyConstraint(['category\_id'], ['categories.id'], ondelete='CASCADE'),  
 sa.ForeignKeyConstraint(['customer\_id'], ['users.id'], ondelete='CASCADE'),  
 sa.ForeignKeyConstraint(['executor\_id'], ['users.id'], ondelete='SET NULL'),  
 sa.PrimaryKeyConstraint('id')  
 )  
 op.create\_index(op.f('ix\_orders\_id'), 'orders', ['id'], unique=False)  
 op.create\_table('user\_categories',  
 sa.Column('user\_id', sa.Integer(), nullable=False),  
 sa.Column('category\_id', sa.Integer(), nullable=False),  
 sa.ForeignKeyConstraint(['category\_id'], ['categories.id'], ondelete='CASCADE'),  
 sa.ForeignKeyConstraint(['user\_id'], ['users.id'], ondelete='CASCADE'),  
 sa.PrimaryKeyConstraint('user\_id', 'category\_id')  
 )  
 op.create\_table('offers',  
 sa.Column('order\_id', sa.Integer(), nullable=False),  
 sa.Column('executor\_id', sa.Integer(), nullable=False),  
 sa.Column('price', sa.Numeric(precision=10, scale=2), nullable=False),  
 sa.Column('estimated\_time', sa.Integer(), nullable=False),  
 sa.Column('status', sa.Enum('PENDING', 'ACCEPTED', 'REJECTED', name='offerstatus'), nullable=False),  
 sa.Column('created\_at', sa.DateTime(), nullable=False),  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.ForeignKeyConstraint(['executor\_id'], ['users.id'], ondelete='CASCADE'),  
 sa.ForeignKeyConstraint(['order\_id'], ['orders.id'], ondelete='CASCADE'),  
 sa.PrimaryKeyConstraint('id')  
 )  
 op.create\_index(op.f('ix\_offers\_id'), 'offers', ['id'], unique=False)  
 op.create\_table('reviews',  
 sa.Column('order\_id', sa.Integer(), nullable=False),  
 sa.Column('author\_id', sa.Integer(), nullable=False),  
 sa.Column('target\_id', sa.Integer(), nullable=False),  
 sa.Column('rating', sa.Integer(), nullable=False),  
 sa.Column('comment', sa.String(), nullable=True),  
 sa.Column('created\_at', sa.DateTime(), nullable=False),  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.ForeignKeyConstraint(['author\_id'], ['users.id'], ondelete='CASCADE'),  
 sa.ForeignKeyConstraint(['order\_id'], ['orders.id'], ondelete='CASCADE'),  
 sa.ForeignKeyConstraint(['target\_id'], ['users.id'], ondelete='CASCADE'),  
 sa.PrimaryKeyConstraint('id')  
 )  
 op.create\_index(op.f('ix\_reviews\_id'), 'reviews', ['id'], unique=False)  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.drop\_index(op.f('ix\_reviews\_id'), table\_name='reviews')  
 op.drop\_table('reviews')  
 op.drop\_index(op.f('ix\_offers\_id'), table\_name='offers')  
 op.drop\_table('offers')  
 op.drop\_table('user\_categories')  
 op.drop\_index(op.f('ix\_orders\_id'), table\_name='orders')  
 op.drop\_table('orders')  
 op.drop\_index(op.f('ix\_users\_id'), table\_name='users')  
 op.drop\_table('users')  
 op.drop\_index(op.f('ix\_cities\_id'), table\_name='cities')  
 op.drop\_table('cities')  
 op.drop\_index(op.f('ix\_categories\_id'), table\_name='categories')  
 op.drop\_table('categories')  
 # ### end Alembic commands ###

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\bot\_runner.py

from aiogram import Bot, Dispatcher  
from app.bot.config import BOT\_TOKEN, ADMIN\_TELEGRAM\_ID  
from app.bot.handlers import start\_router, create\_order\_router, switch\_role\_router, admin\_router, create\_offer\_router, manage\_offers\_router  
import asyncio  
import logging  
  
logging.basicConfig(level=logging.INFO)  
logger = logging.getLogger(\_\_name\_\_)  
  
if not BOT\_TOKEN:  
 raise ValueError("BOT\_TOKEN не задан")  
  
bot = Bot(token=BOT\_TOKEN)  
dp = Dispatcher()  
  
dp.include\_router(start\_router)  
dp.include\_router(create\_order\_router)  
dp.include\_router(switch\_role\_router)  
dp.include\_router(admin\_router)  
dp.include\_router(create\_offer\_router)  
dp.include\_router(manage\_offers\_router)  
  
async def main():  
 logger.info(f"Бот запущен с токеном: {BOT\_TOKEN[:10]}...")  
 try:  
 bot\_info = await bot.get\_me()  
 logger.info(f"Bot info: {bot\_info.username}, ID: {bot\_info.id}")  
 await dp.start\_polling(bot)  
 except Exception as e:  
 logger.error(f"Ошибка при запуске polling: {e}")  
 raise  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(main())

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\config.py

BOT\_TOKEN = "7706031286:AAFpxovmhtpGJv1As1TAbcZGK5J4HNEhVm4" # Ваш токен  
ADMIN\_TELEGRAM\_ID = 704342630 # Исправлено на ваш Telegram ID

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\admin.py

from aiogram import Router, F  
from aiogram.types import Message, InlineKeyboardMarkup, InlineKeyboardButton, CallbackQuery  
from aiogram.fsm.state import State, StatesGroup  
from aiogram.fsm.context import FSMContext  
from app.bot.config import ADMIN\_TELEGRAM\_ID  
from app.core.services.user import get\_users, delete\_user\_by\_id  
from app.core.services.order import get\_orders\_by\_user, delete\_order\_by\_id  
from app.core.services.city import create\_city, get\_all\_cities, update\_city\_by\_id, delete\_city\_by\_id  
from app.core.schemas.city import CityCreate, CityUpdate  
from app.core.models.user import User  
from app.core.models.order import Order  
from app.bot.handlers.utils import get\_db\_session, get\_user\_telegram\_id  
from .start import get\_main\_keyboard  
  
router = Router()  
  
class AdminPanel(StatesGroup):  
 delete\_user = State()  
 delete\_order = State()  
 add\_city = State()  
 edit\_city = State()  
 delete\_city = State()  
  
@router.message(F.text == "Админ-панель", lambda msg: msg.from\_user.id == ADMIN\_TELEGRAM\_ID)  
async def admin\_panel(message: Message):  
 keyboard = InlineKeyboardMarkup(inline\_keyboard=[  
 [InlineKeyboardButton(text="Список пользователей", callback\_data="list\_users")],  
 [InlineKeyboardButton(text="Список заказов", callback\_data="list\_orders")],  
 [InlineKeyboardButton(text="Удалить пользователя", callback\_data="delete\_user")],  
 [InlineKeyboardButton(text="Удалить заказ", callback\_data="delete\_order")],  
 [InlineKeyboardButton(text="Добавить город", callback\_data="add\_city")],  
 [InlineKeyboardButton(text="Изменить город", callback\_data="edit\_city")],  
 [InlineKeyboardButton(text="Удалить город", callback\_data="delete\_city")],  
 [InlineKeyboardButton(text="Назад", callback\_data="back")]  
 ])  
 await message.answer("Админ-панель:", reply\_markup=keyboard)  
  
@router.callback\_query(F.data == "list\_users")  
async def list\_users(callback: CallbackQuery):  
 session = next(get\_db\_session())  
 try:  
 users = get\_users(session)  
 if not users:  
 await callback.message.answer("Пользователей нет.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
 return  
 response = "Список пользователей:\n\n"  
 for user in users:  
 role = "Заказчик" if user.is\_customer else "Исполнитель" if user.is\_executor else "Не определена"  
 response += (  
 f"ID: {user.id}\n"  
 f"Telegram ID: {user.telegram\_id}\n"  
 f"Имя: {user.name}\n"  
 f"Роль: {role}\n"  
 f"Рейтинг: {user.rating}\n\n"  
 )  
 await callback.message.answer(response.strip(), reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except Exception as e:  
 await callback.message.answer(f"Ошибка загрузки пользователей: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
  
@router.callback\_query(F.data == "list\_orders")  
async def list\_orders(callback: CallbackQuery):  
 telegram\_id = callback.from\_user.id  
 session = next(get\_db\_session())  
 try:  
 orders = get\_orders\_by\_user(session, telegram\_id)  
 if not orders:  
 await callback.message.answer("Заказов нет.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
 return  
 response = "Список заказов:\n\n"  
 for order in orders:  
 status\_map = {  
 "В\_ожидании": "Ожидает",  
 "В\_прогрессе": "В процессе",  
 "Выполнен": "Завершён",  
 "Отменен": "Отменён"  
 }  
 status = status\_map.get(order.status, order.status)  
 due\_date = order.due\_date.strftime("%Y-%m-%d")  
 response += (  
 f"ID: {order.id}\n"  
 f"Название: {order.title}\n"  
 f"Статус: {status}\n"  
 f"Желаемая цена: {order.desired\_price} тенге\n"  
 f"Срок: {due\_date}\n\n"  
 )  
 await callback.message.answer(response.strip(), reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except Exception as e:  
 await callback.message.answer(f"Ошибка загрузки заказов: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
  
@router.callback\_query(F.data == "delete\_user")  
async def start\_delete\_user(callback: CallbackQuery, state: FSMContext):  
 await callback.message.answer("Введите ID пользователя для удаления:")  
 await state.set\_state(AdminPanel.delete\_user)  
 await callback.answer()  
  
@router.message(AdminPanel.delete\_user)  
async def process\_delete\_user(message: Message, state: FSMContext):  
 session = next(get\_db\_session())  
 try:  
 user\_id = int(message.text)  
 delete\_user\_by\_id(session, user\_id)  
 await message.answer(f"Пользователь с ID {user\_id} удалён.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректный ID (число).")  
 except Exception as e:  
 await message.answer(f"Ошибка удаления пользователя: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await state.clear()  
  
@router.callback\_query(F.data == "delete\_order")  
async def start\_delete\_order(callback: CallbackQuery, state: FSMContext):  
 await callback.message.answer("Введите ID заказа для удаления:")  
 await state.set\_state(AdminPanel.delete\_order)  
 await callback.answer()  
  
@router.message(AdminPanel.delete\_order)  
async def process\_delete\_order(message: Message, state: FSMContext):  
 session = next(get\_db\_session())  
 try:  
 order\_id = int(message.text)  
 delete\_order\_by\_id(session, order\_id)  
 await message.answer(f"Заказ с ID {order\_id} удалён.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректный ID (число).")  
 except Exception as e:  
 await message.answer(f"Ошибка удаления заказа: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await state.clear()  
  
@router.callback\_query(F.data == "add\_city")  
async def start\_add\_city(callback: CallbackQuery, state: FSMContext):  
 await callback.message.answer("Введите название нового города:")  
 await state.set\_state(AdminPanel.add\_city)  
 await callback.answer()  
  
@router.message(AdminPanel.add\_city)  
async def process\_add\_city(message: Message, state: FSMContext):  
 session = next(get\_db\_session())  
 city\_name = message.text.strip()  
 try:  
 city\_data = CityCreate(name=city\_name)  
 create\_city(session, city\_data)  
 await message.answer(f"Город '{city\_name}' успешно добавлен.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except Exception as e:  
 await message.answer(f"Ошибка добавления города: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await state.clear()  
  
@router.callback\_query(F.data == "edit\_city")  
async def start\_edit\_city(callback: CallbackQuery, state: FSMContext):  
 session = next(get\_db\_session())  
 try:  
 cities = get\_all\_cities(session)  
 if not cities:  
 await callback.message.answer("Городов нет.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
 return  
 response = "Список городов:\n\n"  
 for city in cities:  
 response += f"ID: {city.id} - {city.name}\n"  
 await callback.message.answer(response + "\nВведите ID города для изменения:")  
 await state.set\_state(AdminPanel.edit\_city)  
 except Exception as e:  
 await callback.message.answer(f"Ошибка загрузки городов: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
  
@router.message(AdminPanel.edit\_city)  
async def process\_edit\_city(message: Message, state: FSMContext):  
 try:  
 city\_id = int(message.text)  
 await state.update\_data(city\_id=city\_id)  
 await message.answer("Введите новое название города:")  
 await state.set\_state(AdminPanel.edit\_city)  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректный ID (число).")  
 await state.clear()  
  
@router.message(AdminPanel.edit\_city, lambda msg: "city\_id" in (state.get\_data(msg) or {}))  
async def process\_edit\_city\_name(message: Message, state: FSMContext):  
 session = next(get\_db\_session())  
 data = await state.get\_data()  
 city\_id = data["city\_id"]  
 new\_name = message.text.strip()  
 try:  
 city\_update = CityUpdate(name=new\_name)  
 update\_city\_by\_id(session, city\_update, city\_id)  
 await message.answer(f"Город с ID {city\_id} обновлён на '{new\_name}'.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except Exception as e:  
 await message.answer(f"Ошибка изменения города: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await state.clear()  
  
@router.callback\_query(F.data == "delete\_city")  
async def start\_delete\_city(callback: CallbackQuery, state: FSMContext):  
 session = next(get\_db\_session())  
 try:  
 cities = get\_all\_cities(session)  
 if not cities:  
 await callback.message.answer("Городов нет.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
 return  
 response = "Список городов:\n\n"  
 for city in cities:  
 response += f"ID: {city.id} - {city.name}\n"  
 await callback.message.answer(response + "\nВведите ID города для удаления:")  
 await state.set\_state(AdminPanel.delete\_city)  
 except Exception as e:  
 await callback.message.answer(f"Ошибка загрузки городов: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()  
  
@router.message(AdminPanel.delete\_city)  
async def process\_delete\_city(message: Message, state: FSMContext):  
 session = next(get\_db\_session())  
 try:  
 city\_id = int(message.text)  
 delete\_city\_by\_id(session, city\_id)  
 await message.answer(f"Город с ID {city\_id} удалён.", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректный ID (число).")  
 except Exception as e:  
 await message.answer(f"Ошибка удаления города: {e}", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await state.clear()  
  
@router.callback\_query(F.data == "back")  
async def back\_to\_main(callback: CallbackQuery):  
 await callback.message.answer("Главное меню:", reply\_markup=get\_main\_keyboard({"is\_admin": True}))  
 await callback.answer()

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\create\_offer.py

from aiogram import Router, F  
from aiogram.types import Message, ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup, InlineKeyboardButton, CallbackQuery  
from aiogram.fsm.state import State, StatesGroup  
from aiogram.fsm.context import FSMContext  
from app.core.services.offer import create\_offer  
from app.core.schemas.offer import OfferCreate  
from app.core.services.order import get\_available\_orders  
from app.core.models.user import User  
from app.bot.handlers.utils import get\_db\_session, get\_user\_telegram\_id  
from .start import get\_main\_keyboard  
from datetime import datetime  
  
router = Router()  
  
class CreateOffer(StatesGroup):  
 select\_order = State()  
 price = State()  
 estimated\_time = State()  
  
@router.message(F.text == "Создать предложение")  
async def start\_create\_offer(message: Message, state: FSMContext):  
 telegram\_id = get\_user\_telegram\_id(message)  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user or not user.is\_executor:  
 await message.answer("Только исполнители могут создавать предложения.", reply\_markup=get\_main\_keyboard())  
 return  
 orders = get\_available\_orders(session)  
 if not orders:  
 await message.answer("Нет доступных заказов для предложений.", reply\_markup=get\_main\_keyboard({"is\_executor": True}))  
 return  
 keyboard = InlineKeyboardMarkup(inline\_keyboard=[  
 [InlineKeyboardButton(text=f"ID {order.id} - {order.title}", callback\_data=f"offer\_order\_{order.id}")]  
 for order in orders  
 ] + [[InlineKeyboardButton(text="Отмена", callback\_data="cancel")]])  
 await message.answer("Выберите заказ для предложения:", reply\_markup=keyboard)  
 await state.set\_state(CreateOffer.select\_order)  
 except Exception as e:  
 await message.answer(f"Ошибка: {e}", reply\_markup=get\_main\_keyboard())  
  
@router.callback\_query(CreateOffer.select\_order, F.data.startswith("offer\_order\_"))  
async def process\_order\_selection(callback: CallbackQuery, state: FSMContext):  
 order\_id = int(callback.data.split("\_")[2])  
 await state.update\_data(order\_id=order\_id)  
 await callback.message.answer("Введите вашу цену (в тенге, например, 6000):", reply\_markup=ReplyKeyboardMarkup(  
 keyboard=[[KeyboardButton(text="Отмена")]],  
 resize\_keyboard=True  
 ))  
 await state.set\_state(CreateOffer.price)  
 await callback.answer()  
  
@router.callback\_query(CreateOffer.select\_order, F.data == "cancel")  
async def cancel\_offer\_creation(callback: CallbackQuery, state: FSMContext):  
 await state.clear()  
 await callback.message.answer("Создание предложения отменено.", reply\_markup=get\_main\_keyboard({"is\_executor": True}))  
 await callback.answer()  
  
@router.message(CreateOffer.price, F.text != "Отмена")  
async def process\_price(message: Message, state: FSMContext):  
 try:  
 price = float(message.text)  
 if price <= 0:  
 raise ValueError("Цена должна быть положительной")  
 await state.update\_data(price=price)  
 await message.answer("Введите оценочное время выполнения (в часах, например, 5):", reply\_markup=ReplyKeyboardMarkup(  
 keyboard=[[KeyboardButton(text="Отмена")]],  
 resize\_keyboard=True  
 ))  
 await state.set\_state(CreateOffer.estimated\_time)  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректную цену (число).")  
  
@router.message(CreateOffer.price, F.text == "Отмена")  
async def cancel\_offer\_creation(message: Message, state: FSMContext):  
 await state.clear()  
 await message.answer("Создание предложения отменено.", reply\_markup=get\_main\_keyboard({"is\_executor": True}))  
  
@router.message(CreateOffer.estimated\_time, F.text != "Отмена")  
async def process\_estimated\_time(message: Message, state: FSMContext):  
 try:  
 estimated\_time = int(message.text)  
 if estimated\_time <= 0:  
 raise ValueError("Время должно быть положительным")  
 telegram\_id = get\_user\_telegram\_id(message)  
 session = next(get\_db\_session())  
 data = await state.get\_data()  
 offer\_data = OfferCreate(  
 order\_id=data["order\_id"],  
 price=data["price"],  
 estimated\_time=estimated\_time,  
 start\_date=datetime.utcnow()  
 )  
 create\_offer(session, offer\_data, telegram\_id)  
 await message.answer("Предложение успешно создано!", reply\_markup=get\_main\_keyboard({"is\_executor": True}))  
 await state.clear()  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректное время (целое число).")  
 except Exception as e:  
 await message.answer(f"Ошибка создания предложения: {e}", reply\_markup=get\_main\_keyboard({"is\_executor": True}))  
 await state.clear()  
  
@router.message(CreateOffer.estimated\_time, F.text == "Отмена")  
async def cancel\_offer\_creation(message: Message, state: FSMContext):  
 await state.clear()  
 await message.answer("Создание предложения отменено.", reply\_markup=get\_main\_keyboard({"is\_executor": True}))

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\create\_order.py

from aiogram import Router, F  
from aiogram.fsm.state import State, StatesGroup  
from aiogram.fsm.context import FSMContext  
from aiogram.types import Message  
from app.core.services.order import create\_order  
from app.core.schemas.order import OrderCreate  
from app.core.models.user import User  
from app.bot.handlers.utils import get\_db\_session, get\_user\_telegram\_id  
from .start import get\_main\_keyboard  
from ..config import ADMIN\_TELEGRAM\_ID  
  
router = Router()  
  
class OrderCreation(StatesGroup):  
 title = State()  
 description = State()  
 price = State()  
 due\_date = State()  
  
@router.message(F.text == "Создать заказ")  
async def start\_order\_creation(message: Message, state: FSMContext):  
 telegram\_id = get\_user\_telegram\_id(message)  
 session = next(get\_db\_session())  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user or not user.is\_customer:  
 await message.answer("Только заказчики могут создавать заказы.", reply\_markup=get\_main\_keyboard())  
 return  
 await message.answer("Введите название заказа:")  
 await state.set\_state(OrderCreation.title)  
  
@router.message(OrderCreation.title)  
async def process\_title(message: Message, state: FSMContext):  
 await state.update\_data(title=message.text)  
 await message.answer("Введите описание заказа:")  
 await state.set\_state(OrderCreation.description)  
  
@router.message(OrderCreation.description)  
async def process\_description(message: Message, state: FSMContext):  
 await state.update\_data(description=message.text)  
 await message.answer("Введите желаемую цену (в тенге, например, 5000):")  
 await state.set\_state(OrderCreation.price)  
  
@router.message(OrderCreation.price)  
async def process\_price(message: Message, state: FSMContext):  
 try:  
 price = float(message.text)  
 if price <= 0:  
 raise ValueError("Цена должна быть положительной")  
 await state.update\_data(price=price)  
 await message.answer("Введите срок выполнения (например, 2025-03-20):")  
 await state.set\_state(OrderCreation.due\_date)  
 except ValueError:  
 await message.answer("Пожалуйста, введите корректную цену (число).")  
  
@router.message(OrderCreation.due\_date)  
async def process\_due\_date(message: Message, state: FSMContext):  
 data = await state.get\_data()  
 telegram\_id = get\_user\_telegram\_id(message)  
 session = next(get\_db\_session())  
 try:  
 due\_date = message.text # Предполагаем формат YYYY-MM-DD, можно добавить валидацию  
 order\_data = OrderCreate(  
 category\_id=1, # Захардкодим, можно добавить выбор категории  
 title=data["title"],  
 description=data["description"],  
 desired\_price=data["price"],  
 due\_date=due\_date  
 )  
 order = create\_order(session, order\_data, telegram\_id)  
 roles = {  
 "is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID,  
 "is\_executor": False,  
 "is\_customer": True  
 }  
 await message.answer(f"Заказ создан с ID: {order.id}", reply\_markup=get\_main\_keyboard(roles))  
 except Exception as e:  
 await message.answer(f"Ошибка создания заказа: {e}", reply\_markup=get\_main\_keyboard())  
 await state.clear()

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\manage\_offers.py

from aiogram import Router, F  
from aiogram.types import Message, InlineKeyboardMarkup, InlineKeyboardButton, CallbackQuery  
from aiogram.fsm.state import State, StatesGroup  
from aiogram.fsm.context import FSMContext  
from app.core.services.order import get\_orders\_by\_user, update\_order\_by\_id  
from app.core.services.offer import get\_offer\_by\_id, update\_offer\_by\_id  
from app.core.schemas.order import OrderUpdate  
from app.core.schemas.offer import OfferUpdate  
from app.core.models.user import User  
from app.bot.config import ADMIN\_TELEGRAM\_ID  
from app.bot.handlers.utils import get\_db\_session, get\_user\_telegram\_id  
from .start import get\_main\_keyboard  
  
router = Router()  
  
class ManageOffers(StatesGroup):  
 select\_order = State()  
  
@router.message(F.text == "Посмотреть предложения")  
async def start\_manage\_offers(message: Message, state: FSMContext):  
 telegram\_id = get\_user\_telegram\_id(message)  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user or not user.is\_customer:  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": user.is\_executor if user else False, "is\_customer": user.is\_customer if user else False}  
 await message.answer("Только заказчики могут просматривать предложения.", reply\_markup=get\_main\_keyboard(roles))  
 return  
 orders = get\_orders\_by\_user(session, telegram\_id)  
 if not orders:  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": user.is\_executor, "is\_customer": user.is\_customer}  
 await message.answer("У вас нет заказов.", reply\_markup=get\_main\_keyboard(roles))  
 return  
 keyboard = InlineKeyboardMarkup(inline\_keyboard=[  
 [InlineKeyboardButton(text=f"ID {order.id} - {order.title}", callback\_data=f"view\_offers\_{order.id}")]  
 for order in orders  
 ] + [[InlineKeyboardButton(text="Отмена", callback\_data="cancel")]])  
 await message.answer("Выберите заказ для просмотра предложений:", reply\_markup=keyboard)  
 await state.set\_state(ManageOffers.select\_order)  
 except Exception as e:  
 await message.answer(f"Ошибка: {e}", reply\_markup=get\_main\_keyboard())  
  
@router.callback\_query(ManageOffers.select\_order, F.data.startswith("view\_offers\_"))  
async def show\_offers(callback: CallbackQuery, state: FSMContext):  
 order\_id = int(callback.data.split("\_")[2])  
 telegram\_id = callback.from\_user.id  
 session = next(get\_db\_session())  
 try:  
 order = session.query(Order).filter(Order.id == order\_id, Order.customer\_id == telegram\_id).first()  
 if not order:  
 await callback.message.answer("Заказ не найден или не принадлежит вам.", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 await state.clear()  
 await callback.answer()  
 return  
 offers = order.offers  
 if not offers:  
 await callback.message.answer("По этому заказу нет предложений.", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 await state.clear()  
 await callback.answer()  
 return  
 response = f"Предложения по заказу ID {order\_id}:\n\n"  
 keyboard\_buttons = []  
 for offer in offers:  
 executor = offer.executor  
 start\_date = offer.start\_date.strftime("%Y-%m-%d") if offer.start\_date else "Не указано"  
 response += (  
 f"ID предложения: {offer.id}\n"  
 f"Исполнитель: {executor.name} (Рейтинг: {executor.rating})\n"  
 f"Цена: {offer.price} тенге\n"  
 f"Время: {offer.estimated\_time} часов\n"  
 f"Дата начала: {start\_date}\n"  
 f"Статус: {offer.status}\n\n"  
 )  
 if offer.status == "pending":  
 keyboard\_buttons.append([  
 InlineKeyboardButton(text=f"Принять {offer.id}", callback\_data=f"accept\_offer\_{offer.id}\_{order\_id}"),  
 InlineKeyboardButton(text=f"Отклонить {offer.id}", callback\_data=f"reject\_offer\_{offer.id}\_{order\_id}")  
 ])  
 keyboard\_buttons.append([InlineKeyboardButton(text="Назад", callback\_data="cancel")])  
 keyboard = InlineKeyboardMarkup(inline\_keyboard=keyboard\_buttons)  
 await callback.message.answer(response.strip(), reply\_markup=keyboard)  
 except Exception as e:  
 await callback.message.answer(f"Ошибка загрузки предложений: {e}", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 await state.clear()  
 await callback.answer()  
  
@router.callback\_query(F.data.startswith("accept\_offer\_"))  
async def accept\_offer(callback: CallbackQuery, state: FSMContext):  
 telegram\_id = callback.from\_user.id  
 offer\_id, order\_id = map(int, callback.data.split("\_")[2:4])  
 session = next(get\_db\_session())  
 try:  
 offer = get\_offer\_by\_id(session, offer\_id)  
 if offer.order\_id != order\_id or offer.status != "pending":  
 await callback.message.answer("Предложение недоступно для принятия.", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 return  
 order\_update = OrderUpdate(executor\_id=offer.executor\_id, status="В\_прогрессе")  
 update\_order\_by\_id(session, order\_update, order\_id)  
 offer\_update = OfferUpdate(status="accepted")  
 update\_offer\_by\_id(session, offer\_update, offer\_id)  
 executor = offer.executor  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": False, "is\_customer": True}  
 await callback.message.answer(  
 f"Предложение принято, исполнитель назначен!\nСвяжитесь с исполнителем: @{executor.username}",  
 reply\_markup=get\_main\_keyboard(roles)  
 )  
 await state.clear()  
 except Exception as e:  
 await callback.message.answer(f"Ошибка принятия предложения: {e}", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 await state.clear()  
 await callback.answer()  
  
@router.callback\_query(F.data.startswith("reject\_offer\_"))  
async def reject\_offer(callback: CallbackQuery, state: FSMContext):  
 telegram\_id = callback.from\_user.id  
 offer\_id, order\_id = map(int, callback.data.split("\_")[2:4])  
 session = next(get\_db\_session())  
 try:  
 offer = get\_offer\_by\_id(session, offer\_id)  
 if offer.order\_id != order\_id or offer.status != "pending":  
 await callback.message.answer("Предложение недоступно для отклонения.", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 return  
 offer\_update = OfferUpdate(status="rejected")  
 update\_offer\_by\_id(session, offer\_update, offer\_id)  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": False, "is\_customer": True}  
 await callback.message.answer("Предложение отклонено.", reply\_markup=get\_main\_keyboard(roles))  
 await state.clear()  
 except Exception as e:  
 await callback.message.answer(f"Ошибка отклонения предложения: {e}", reply\_markup=get\_main\_keyboard({"is\_customer": True}))  
 await state.clear()  
 await callback.answer()  
  
@router.callback\_query(ManageOffers.select\_order, F.data == "cancel")  
async def cancel\_manage\_offers(callback: CallbackQuery, state: FSMContext):  
 telegram\_id = callback.from\_user.id  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": user.is\_executor if user else False, "is\_customer": user.is\_customer if user else True}  
 except Exception:  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": False, "is\_customer": True}  
 await state.clear()  
 await callback.message.answer("Действие отменено.", reply\_markup=get\_main\_keyboard(roles))  
 await callback.answer()

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\start.py

import logging  
from aiogram import Router, F  
from aiogram.types import Message, ReplyKeyboardMarkup, KeyboardButton  
from app.core.services.user import get\_user\_by\_id, create\_user  
from app.core.schemas.user import UserCreate  
from app.core.models.user import User  
from app.bot.config import ADMIN\_TELEGRAM\_ID  
from app.bot.handlers.utils import get\_db\_session, get\_user\_telegram\_id  
  
# Настраиваем логирование  
logging.basicConfig(level=logging.INFO)  
logger = logging.getLogger(\_\_name\_\_)  
  
router = Router()  
  
  
def get\_main\_keyboard(roles: dict = None):  
 roles = roles or {}  
 buttons = [  
 [KeyboardButton(text="Профиль"), KeyboardButton(text="Создать заказ")],  
 [KeyboardButton(text="Список заказов"), KeyboardButton(text="Сменить роль")]  
 ]  
 if roles.get("is\_executor"):  
 buttons.append([KeyboardButton(text="Создать предложение")])  
 if roles.get("is\_customer"):  
 buttons.append([KeyboardButton(text="Посмотреть предложения")])  
 if roles.get("is\_admin"):  
 buttons.append([KeyboardButton(text="Админ-панель")])  
 return ReplyKeyboardMarkup(keyboard=buttons, resize\_keyboard=True, row\_width=2)  
  
  
@router.message(F.command == "start")  
async def cmd\_start(message: Message):  
 telegram\_id = get\_user\_telegram\_id(message)  
 logger.info(f"Получена команда /start от Telegram ID: {telegram\_id}")  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user:  
 logger.info(f"Пользователь с Telegram ID {telegram\_id} не найден, создаём нового.")  
 user\_data = UserCreate(  
 telegram\_id=telegram\_id,  
 name=message.from\_user.full\_name or "Без имени",  
 username=message.from\_user.username,  
 is\_customer=False,  
 is\_executor=False,  
 city\_id=1 # Предполагаем, что город с ID 1 (Кокшетау) существует  
 )  
 user = create\_user(session, user\_data)  
 logger.info(f"Создан пользователь: {user.id}, {user.name}")  
 await message.answer("Добро пожаловать! Вы зарегистрированы. Укажите роль через 'Сменить роль'.")  
 else:  
 logger.info(f"Пользователь найден: {user.id}, {user.name}")  
  
 role = "Заказчик" if user.is\_customer else "Исполнитель" if user.is\_executor else "Не определена"  
 city = user.city.name  
 text = f"Добро пожаловать!\nИмя: {user.name}\nРоль: {role}\nГород: {city}\nРейтинг: {user.rating}"  
 roles = {  
 "is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID,  
 "is\_executor": user.is\_executor,  
 "is\_customer": user.is\_customer  
 }  
 await message.answer(text, reply\_markup=get\_main\_keyboard(roles))  
 except Exception as e:  
 logger.error(f"Ошибка в /start: {e}")  
 await message.answer(f"Ошибка: {e}", reply\_markup=get\_main\_keyboard())  
 finally:  
 session.close()  
  
  
@router.message(F.text == "Профиль")  
async def show\_profile(message: Message):  
 telegram\_id = get\_user\_telegram\_id(message)  
 logger.info(f"Запрос профиля от Telegram ID: {telegram\_id}")  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user:  
 logger.info(f"Пользователь с Telegram ID {telegram\_id} не найден в профиле.")  
 await message.answer("Пользователь не найден. Используйте /start для регистрации.")  
 return  
 role = "Заказчик" if user.is\_customer else "Исполнитель" if user.is\_executor else "Не определена"  
 city = user.city.name  
 text = f"Имя: {user.name}\nРоль: {role}\nГород: {city}\nРейтинг: {user.rating}"  
 roles = {  
 "is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID,  
 "is\_executor": user.is\_executor,  
 "is\_customer": user.is\_customer  
 }  
 await message.answer(text, reply\_markup=get\_main\_keyboard(roles))  
 except Exception as e:  
 logger.error(f"Ошибка в профиле: {e}")  
 await message.answer(f"Ошибка: {e}", reply\_markup=get\_main\_keyboard())  
 finally:  
 session.close()

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\switch\_role.py

from aiogram import Router, F  
from aiogram.types import Message, InlineKeyboardMarkup, InlineKeyboardButton, CallbackQuery  
from app.core.services.user import get\_user\_by\_id, update\_user\_by\_id  
from app.core.schemas.user import UserUpdate  
from app.core.models.user import User # Добавлен импорт  
from app.bot.config import ADMIN\_TELEGRAM\_ID  
from app.bot.handlers.utils import get\_db\_session, get\_user\_telegram\_id  
from .start import get\_main\_keyboard  
  
router = Router()  
  
@router.message(F.text == "Сменить роль")  
async def switch\_role(message: Message):  
 telegram\_id = get\_user\_telegram\_id(message)  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user:  
 await message.answer("Пользователь не найден. Используйте /start для регистрации.", reply\_markup=get\_main\_keyboard())  
 return  
 current\_role = "Заказчик" if user.is\_customer else "Исполнитель" if user.is\_executor else "Не определена"  
 keyboard = InlineKeyboardMarkup(inline\_keyboard=[  
 [InlineKeyboardButton(text="Заказчик", callback\_data="role\_customer")],  
 [InlineKeyboardButton(text="Исполнитель", callback\_data="role\_executor")],  
 [InlineKeyboardButton(text="Назад", callback\_data="back")]  
 ])  
 await message.answer(f"Текущая роль: {current\_role}\nВыберите новую роль:", reply\_markup=keyboard)  
 except Exception as e:  
 await message.answer(f"Ошибка загрузки текущей роли: {e}", reply\_markup=get\_main\_keyboard())  
  
@router.callback\_query(F.data.startswith("role\_"))  
async def change\_role(callback: CallbackQuery):  
 telegram\_id = callback.from\_user.id  
 role = callback.data.split("\_")[1]  
 role\_name = "Заказчик" if role == "customer" else "Исполнитель"  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 if not user:  
 await callback.message.answer("Пользователь не найден.", reply\_markup=get\_main\_keyboard())  
 return  
 update\_data = UserUpdate(  
 is\_customer=role == "customer",  
 is\_executor=role == "executor"  
 )  
 updated\_user = update\_user\_by\_id(session, update\_data, user.id)  
 roles = {  
 "is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID,  
 "is\_executor": updated\_user.is\_executor,  
 "is\_customer": updated\_user.is\_customer  
 }  
 await callback.message.answer(f"Роль успешно изменена на: {role\_name}", reply\_markup=get\_main\_keyboard(roles))  
 except Exception as e:  
 await callback.message.answer(f"Ошибка смены роли: {e}", reply\_markup=get\_main\_keyboard())  
 await callback.answer()  
  
@router.callback\_query(F.data == "back")  
async def back\_to\_main(callback: CallbackQuery):  
 telegram\_id = callback.from\_user.id  
 session = next(get\_db\_session())  
 try:  
 user = session.query(User).filter(User.telegram\_id == telegram\_id).first()  
 roles = {  
 "is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID,  
 "is\_executor": user.is\_executor if user else False,  
 "is\_customer": user.is\_customer if user else False  
 }  
 except Exception:  
 roles = {"is\_admin": telegram\_id == ADMIN\_TELEGRAM\_ID, "is\_executor": False, "is\_customer": False}  
 await callback.message.answer("Главное меню:", reply\_markup=get\_main\_keyboard(roles))  
 await callback.answer()

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\utils.py

from sqlalchemy.orm import Session  
from app.core.database.helper import SessionLocal  
from aiogram.types import Message  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
def get\_db\_session() -> Session:  
 with SessionLocal() as session:  
 yield session  
  
def get\_user\_telegram\_id(message: Message) -> int:  
 return message.from\_user.id

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\bot\handlers\\_\_init\_\_.py

from .start import router as start\_router  
from .create\_order import router as create\_order\_router  
from .switch\_role import router as switch\_role\_router  
from .admin import router as admin\_router  
from .create\_offer import router as create\_offer\_router  
from .manage\_offers import router as manage\_offers\_router # Новый роутер

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\config.py

import os  
  
DB\_URL = os.getenv("DB\_URL")  
DB\_ECHO = False

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\init\_db.py

from app.core.database.helper import engine  
from app.core.models import Base  
Base.metadata.create\_all(bind=engine)  
# если база пустая то запустить отдельно

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\database\helper.py

from sqlalchemy import create\_engine  
from sqlalchemy.orm import sessionmaker  
from app.core.config import DB\_URL, DB\_ECHO  
  
engine = create\_engine(url=DB\_URL, echo=DB\_ECHO)  
SessionLocal = sessionmaker(bind=engine)  
  
def get\_session():  
 with SessionLocal() as session:  
 yield session

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\association.py

from sqlalchemy import Table, Column, Integer, ForeignKey  
from app.core.models.base import Base  
  
user\_categories = Table(  
 "user\_categories",  
 Base.metadata,  
 Column("user\_id", Integer, ForeignKey("users.id", ondelete="CASCADE"), primary\_key=True), # ID пользователя  
 Column("category\_id", Integer, ForeignKey("categories.id", ondelete="CASCADE"), primary\_key=True), # ID категории  
) # Таблица связи многие-ко-многим между пользователями и категориями

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\base.py

from sqlalchemy.orm import DeclarativeBase, Mapped, mapped\_column  
  
class Base(DeclarativeBase):  
 """Базовый класс для всех моделей базы данных."""  
 id: Mapped[int] = mapped\_column(primary\_key=True, index=True)

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\category.py

from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from app.core.models.base import Base  
  
class Category(Base):  
 """Модель категории услуг."""  
 \_\_tablename\_\_ = "categories"  
  
 name: Mapped[str] = mapped\_column(unique=True, nullable=False) # Название категории (уникальное)  
 users: Mapped[list["User"]] = relationship(  
 "User", secondary="user\_categories", back\_populates="categories" # Пользователи, связанные с категорией  
 )

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\city.py

from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from app.core.models.base import Base  
  
class City(Base):  
 """Модель города."""  
 \_\_tablename\_\_ = "cities"  
  
 name: Mapped[str] = mapped\_column(unique=True, nullable=False) # Название города (уникальное)  
 users: Mapped[list["User"]] = relationship("User", back\_populates="city") # Пользователи, привязанные к городу

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\offer.py

from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from datetime import datetime  
from app.core.models.base import Base  
from decimal import Decimal  
from sqlalchemy import Enum, ForeignKey, Numeric  
import enum  
  
class OfferStatus(str, enum.Enum):  
 PENDING = "pending" # Ожидает  
 ACCEPTED = "accepted" # Принято  
 REJECTED = "rejected" # Отклонено  
  
class Offer(Base):  
 """Модель предложения."""  
 \_\_tablename\_\_ = "offers"  
  
 order\_id: Mapped[int] = mapped\_column(ForeignKey("orders.id", ondelete="CASCADE"), nullable=False) # ID заказа  
 executor\_id: Mapped[int] = mapped\_column(ForeignKey("users.id", ondelete="CASCADE"), nullable=False) # ID исполнителя  
 price: Mapped[Decimal] = mapped\_column(Numeric(10, 2), nullable=False) # Цена предложения  
 estimated\_time: Mapped[int] = mapped\_column(nullable=False) # Оценочное время выполнения (в часах)  
 status: Mapped[OfferStatus] = mapped\_column(Enum(OfferStatus), default=OfferStatus.PENDING, nullable=False) # Статус предложения  
 created\_at: Mapped[datetime] = mapped\_column(default=datetime.utcnow, nullable=False) # Дата создания  
 start\_date: Mapped[datetime | None] = mapped\_column(nullable=True) # Добавлено поле для даты начала  
  
 # Связи с другими моделями  
 order: Mapped["Order"] = relationship("Order", back\_populates="offers") # Заказ  
 executor: Mapped["User"] = relationship("User", back\_populates="offers") # Исполнитель

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\order.py

from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from datetime import datetime  
from app.core.models.base import Base  
from decimal import Decimal  
from sqlalchemy import ForeignKey, Numeric, Enum  
import enum  
  
class OrderStatus(str, enum.Enum):  
 PENDING = "В\_ожидании" # Приводим к верхнему регистру  
 IN\_PROGRESS = "В\_прогрессе"  
 COMPLETED = "Выполнен"  
 CANCELED = "Отменен"  
  
class Order(Base):  
 """Модель заказа."""  
 \_\_tablename\_\_ = "orders"  
  
 customer\_id: Mapped[int] = mapped\_column(ForeignKey("users.id", ondelete="CASCADE"), nullable=False) # ID заказчика  
 executor\_id: Mapped[int | None] = mapped\_column(ForeignKey("users.id", ondelete="SET NULL"), nullable=True) # ID исполнителя (опционально)  
 category\_id: Mapped[int] = mapped\_column(ForeignKey("categories.id", ondelete="CASCADE"), nullable=False) # ID категории  
 title: Mapped[str] = mapped\_column(nullable=False) # Название заказа  
 description: Mapped[str | None] = mapped\_column(nullable=True) # Описание заказа (опционально)  
 desired\_price: Mapped[Decimal] = mapped\_column(Numeric(10, 2), nullable=False) # Желаемая цена  
 due\_date: Mapped[datetime] = mapped\_column(nullable=False) # Срок выполнения  
 created\_at: Mapped[datetime] = mapped\_column(default=datetime.utcnow, nullable=False) # Дата создания  
 status: Mapped[OrderStatus] = mapped\_column(Enum(OrderStatus, name="orderstatus"), default=OrderStatus.PENDING,  
 nullable=True)  
 # Связи с другими моделями  
 customer: Mapped["User"] = relationship("User", foreign\_keys="Order.customer\_id", back\_populates="orders\_created") # Заказчик  
 executor: Mapped["User"] = relationship("User", foreign\_keys="Order.executor\_id", back\_populates="orders\_executed") # Исполнитель  
 category: Mapped["Category"] = relationship("Category") # Категория  
 offers: Mapped[list["Offer"]] = relationship("Offer", back\_populates="order") # Предложения по заказу  
 review: Mapped["Review"] = relationship("Review", back\_populates="order", uselist=False) # Отзыв по заказу

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\review.py

from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from datetime import datetime  
from app.core.models.base import Base  
from sqlalchemy import ForeignKey, CheckConstraint  
  
class Review(Base):  
 """Модель отзыва."""  
 \_\_tablename\_\_ = "reviews"  
  
 order\_id: Mapped[int] = mapped\_column(ForeignKey("orders.id", ondelete="CASCADE"), nullable=False) # ID заказа  
 author\_id: Mapped[int] = mapped\_column(ForeignKey("users.id", ondelete="CASCADE"), nullable=False) # ID автора отзыва  
 target\_id: Mapped[int] = mapped\_column(ForeignKey("users.id", ondelete="CASCADE"), nullable=False) # ID получателя отзыва  
 rating: Mapped[int] = mapped\_column(CheckConstraint("rating BETWEEN 1 AND 5"), nullable=False) # Рейтинг (от 1 до 5)  
 comment: Mapped[str | None] = mapped\_column(nullable=True) # Комментарий (опционально)  
 created\_at: Mapped[datetime] = mapped\_column(default=datetime.utcnow, nullable=False) # Дата создания  
  
 # Связи с другими моделями  
 order: Mapped["Order"] = relationship("Order", back\_populates="review") # Заказ  
 author: Mapped["User"] = relationship("User", foreign\_keys="Review.author\_id", back\_populates="reviews\_written") # Автор  
 target: Mapped["User"] = relationship("User", foreign\_keys="Review.target\_id", back\_populates="reviews\_received") # Получатель

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\user.py

from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from app.core.models.base import Base  
from decimal import Decimal  
from sqlalchemy import ForeignKey, Numeric, CheckConstraint  
  
class User(Base):  
 """Модель пользователя."""  
 \_\_tablename\_\_ = "users"  
  
 telegram\_id: Mapped[int] = mapped\_column(unique=True, nullable=False) # Уникальный Telegram ID  
 name: Mapped[str] = mapped\_column(nullable=False) # Имя пользователя  
 username: Mapped[str | None] = mapped\_column(unique=True, nullable=True) # Уникальное имя в Telegram (опционально)  
 is\_customer: Mapped[bool] = mapped\_column(default=False) # Является ли заказчиком  
 is\_executor: Mapped[bool] = mapped\_column(default=False) # Является ли исполнителем  
 is\_admin: Mapped[bool] = mapped\_column(default=False) # Является ли администратором  
 city\_id: Mapped[int] = mapped\_column(ForeignKey("cities.id", ondelete="CASCADE"), nullable=False) # ID города  
 rating: Mapped[Decimal] = mapped\_column(Numeric(2, 1), default=0.0, nullable=False) # Рейтинг пользователя  
 completed\_orders: Mapped[int] = mapped\_column(default=0, nullable=False) # Количество завершенных заказов  
  
 # Связи с другими моделями  
 city: Mapped["City"] = relationship("City", back\_populates="users") # Связь с городом  
 categories: Mapped[list["Category"]] = relationship(  
 "Category", secondary="user\_categories", back\_populates="users" # Связь с категориями через таблицу user\_categories  
 )  
 orders\_created: Mapped[list["Order"]] = relationship(  
 "Order", foreign\_keys="Order.customer\_id", back\_populates="customer" # Заказы, созданные пользователем  
 )  
 orders\_executed: Mapped[list["Order"]] = relationship(  
 "Order", foreign\_keys="Order.executor\_id", back\_populates="executor" # Заказы, выполненные пользователем  
 )  
 offers: Mapped[list["Offer"]] = relationship("Offer", back\_populates="executor") # Предложения пользователя  
 reviews\_received: Mapped[list["Review"]] = relationship(  
 "Review", foreign\_keys="Review.target\_id", back\_populates="target" # Полученные отзывы  
 )  
 reviews\_written: Mapped[list["Review"]] = relationship(  
 "Review", foreign\_keys="Review.author\_id", back\_populates="author" # Написанные отзывы  
 )  
  
 \_\_table\_args\_\_ = (  
 CheckConstraint("NOT (is\_customer AND is\_executor)", name="check\_role\_exclusivity"), # Проверка: нельзя быть заказчиком и исполнителем одновременно  
 )

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\models\\_\_init\_\_.py

from .base import Base  
from .user import User  
from .city import City  
from .category import Category  
from .association import user\_categories  
from .order import Order  
from .offer import Offer  
from .review import Review

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\base.py

from pydantic import BaseModel  
  
class BaseSchema(BaseModel):  
 model\_config = {  
 "from\_attributes": True,  
 "populate\_by\_name": True,  
 # Убираем alias\_generator=camelize  
 }

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\category.py

from app.core.schemas.base import BaseSchema  
from typing import Optional  
  
  
class CategoryRead(BaseSchema):  
 id: int  
 name: str  
  
  
class CategoryCreate(BaseSchema):  
 name: str  
  
 model\_config = {"str\_strip\_whitespace": True}  
  
  
class CategoryUpdate(BaseSchema):  
 name: Optional[str] = None  
  
 model\_config = {"str\_strip\_whitespace": True}

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\city.py

from app.core.schemas.base import BaseSchema  
from typing import Optional  
  
  
class CityRead(BaseSchema):  
 id: int  
 name: str  
  
  
class CityCreate(BaseSchema):  
 name: str  
  
 model\_config = {"str\_strip\_whitespace": True}  
  
  
class CityUpdate(BaseSchema):  
 name: Optional[str] = None  
  
 model\_config = {"str\_strip\_whitespace": True}

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\offer.py

from typing import Optional  
from datetime import datetime  
from app.core.schemas.base import BaseSchema  
import enum  
  
class OfferStatus(str, enum.Enum):  
 PENDING = "pending"  
 ACCEPTED = "accepted"  
 REJECTED = "rejected"  
  
class OfferRead(BaseSchema):  
 id: int  
 order\_id: int  
 executor\_id: int  
 price: float  
 estimated\_time: int  
 status: OfferStatus  
 created\_at: datetime  
 start\_date: Optional[datetime] # Добавлено  
  
class OfferCreate(BaseSchema):  
 order\_id: int  
 price: float  
 estimated\_time: int  
 start\_date: Optional[datetime] # Добавлено  
  
 model\_config = {"str\_strip\_whitespace": True}  
  
class OfferUpdate(BaseSchema):  
 price: Optional[float] = None  
 estimated\_time: Optional[int] = None  
 status: Optional[OfferStatus] = None  
 start\_date: Optional[datetime] = None # Добавлено

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\order.py

from typing import Optional  
from datetime import datetime  
from app.core.schemas.base import BaseSchema  
import enum  
  
class OrderStatus(str, enum.Enum):  
 PENDING = "В\_ожидании" # Приводим к верхнему регистру  
 IN\_PROGRESS = "В\_прогрессе"  
 COMPLETED = "Выполнен"  
 CANCELED = "Отменен"  
  
class OrderRead(BaseSchema):  
 id: int  
 customer\_id: int  
 executor\_id: Optional[int]  
 category\_id: int  
 title: str  
 description: Optional[str]  
 desired\_price: float  
 due\_date: datetime  
 created\_at: datetime  
 status: OrderStatus  
  
class OrderCreate(BaseSchema):  
 category\_id: int  
 title: str  
 description: Optional[str] = None  
 desired\_price: float  
 due\_date: datetime  
  
 model\_config = {"str\_strip\_whitespace": True}  
  
class OrderUpdate(BaseSchema):  
 executor\_id: Optional[int] = None  
 status: Optional[OrderStatus] = None

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\review.py

from typing import Optional  
from datetime import datetime  
from app.core.schemas.base import BaseSchema  
  
class ReviewRead(BaseSchema):  
 id: int  
 order\_id: int  
 author\_id: int  
 target\_id: int  
 rating: int  
 comment: Optional[str]  
 created\_at: datetime  
  
class ReviewCreate(BaseSchema):  
 order\_id: int  
 target\_id: int  
 rating: int  
 comment: Optional[str] = None  
  
 model\_config = {"str\_strip\_whitespace": True}  
  
class ReviewUpdate(BaseSchema):  
 rating: Optional[int] = None  
 comment: Optional[str] = None  
  
 model\_config = {"str\_strip\_whitespace": True}

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\schemas\user.py

from typing import Optional, List  
from app.core.schemas.base import BaseSchema  
  
class UserRead(BaseSchema):  
 id: int  
 telegram\_id: int  
 name: str  
 username: Optional[str]  
 is\_customer: bool  
 is\_executor: bool  
 is\_admin: bool  
 city\_id: int  
 rating: float  
 completed\_orders: int  
  
class UserCreate(BaseSchema):  
 telegram\_id: int  
 name: str  
 username: Optional[str]  
 is\_customer: bool = False  
 is\_executor: bool = False  
 city\_id: int  
 category\_ids: Optional[List[int]] = None  
  
 model\_config = {"str\_strip\_whitespace": True}  
  
class UserUpdate(BaseSchema):  
 name: Optional[str] = None  
 username: Optional[str] = None  
 is\_customer: Optional[bool] = None  
 is\_executor: Optional[bool] = None  
 city\_id: Optional[int] = None  
 category\_ids: Optional[List[int]] = None  
  
 model\_config = {"str\_strip\_whitespace": True}

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\services\category.py

from fastapi.exceptions import HTTPException  
from sqlalchemy import select  
from sqlalchemy.orm import Session  
from sqlalchemy.exc import SQLAlchemyError, IntegrityError  
from app.core.models.category import Category  
from app.core.schemas.category import CategoryCreate, CategoryUpdate  
  
def create\_category(session: Session, data: CategoryCreate) -> Category:  
 """Создать новую категорию."""  
 category = Category(\*\*data.model\_dump())  
 session.add(category)  
 try:  
 session.commit()  
 session.refresh(category)  
 except IntegrityError:  
 session.rollback()  
 raise HTTPException(status\_code=400, detail="Категория с таким названием уже существует")  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при создании категории: {e}")  
 return category  
  
def get\_category\_by\_id(session: Session, id: int) -> Category:  
 """Получить категорию по ID."""  
 category = session.get(Category, id)  
 if not category:  
 raise HTTPException(status\_code=404, detail="Категория не найдена")  
 return category  
  
def get\_all\_categories(session: Session) -> list[Category]:  
 """Получить список всех категорий."""  
 return session.scalars(select(Category)).all()  
  
def update\_category\_by\_id(session: Session, data: CategoryUpdate, id: int) -> Category:  
 """Обновить данные категории по ID."""  
 category = get\_category\_by\_id(session, id)  
 update\_data = data.model\_dump(exclude\_unset=True)  
 for key, value in update\_data.items():  
 setattr(category, key, value)  
 try:  
 session.commit()  
 session.refresh(category)  
 except IntegrityError:  
 session.rollback()  
 raise HTTPException(status\_code=400, detail="Категория с таким названием уже существует")  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при обновлении категории: {e}")  
 return category  
  
def delete\_category\_by\_id(session: Session, id: int):  
 """Удалить категорию по ID."""  
 category = get\_category\_by\_id(session, id)  
 session.delete(category)  
 try:  
 session.commit()  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при удалении категории: {e}")

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\services\city.py

from fastapi.exceptions import HTTPException  
from sqlalchemy import select  
from sqlalchemy.orm import Session  
from sqlalchemy.exc import SQLAlchemyError, IntegrityError  
from app.core.models.city import City  
from app.core.schemas.city import CityCreate, CityUpdate  
  
def create\_city(session: Session, data: CityCreate) -> City:  
 """Создать новый город."""  
 city = City(\*\*data.model\_dump())  
 session.add(city)  
 try:  
 session.commit()  
 session.refresh(city)  
 except IntegrityError:  
 session.rollback()  
 raise HTTPException(status\_code=400, detail="Город с таким названием уже существует")  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при создании города: {e}")  
 return city  
  
def get\_city\_by\_id(session: Session, id: int) -> City:  
 """Получить город по ID."""  
 city = session.get(City, id)  
 if not city:  
 raise HTTPException(status\_code=404, detail="Город не найден")  
 return city  
  
def get\_all\_cities(session: Session) -> list[City]:  
 """Получить список всех городов."""  
 return session.scalars(select(City)).all()  
  
def update\_city\_by\_id(session: Session, data: CityUpdate, id: int) -> City:  
 """Обновить данные города по ID."""  
 city = get\_city\_by\_id(session, id)  
 update\_data = data.model\_dump(exclude\_unset=True)  
 for key, value in update\_data.items():  
 setattr(city, key, value)  
 try:  
 session.commit()  
 session.refresh(city)  
 except IntegrityError:  
 session.rollback()  
 raise HTTPException(status\_code=400, detail="Город с таким названием уже существует")  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при обновлении города: {e}")  
 return city  
  
def delete\_city\_by\_id(session: Session, id: int):  
 """Удалить город по ID."""  
 city = get\_city\_by\_id(session, id)  
 session.delete(city)  
 try:  
 session.commit()  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при удалении города: {e}")

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\services\offer.py

from fastapi.exceptions import HTTPException  
from sqlalchemy import select  
from sqlalchemy.orm import Session  
from sqlalchemy.exc import SQLAlchemyError  
from app.core.models.offer import Offer  
from app.core.models.order import Order  
from app.core.schemas.offer import OfferCreate, OfferUpdate  
  
def create\_offer(session: Session, data: OfferCreate, executor\_id: int) -> Offer:  
 """Создать новое предложение."""  
 order = session.get(Order, data.order\_id)  
 if not order:  
 raise HTTPException(status\_code=400, detail="Заказ не найден")  
 executor = session.get(User, executor\_id)  
 if order.customer\_id == executor\_id and not executor.is\_admin:  
 raise HTTPException(status\_code=400, detail="Самопредложение запрещено для не-администраторов")  
 offer\_data = data.model\_dump()  
 offer = Offer(\*\*offer\_data, executor\_id=executor\_id)  
 session.add(offer)  
 try:  
 session.commit()  
 session.refresh(offer)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при создании предложения: {e}")  
 return offer  
  
def get\_offer\_by\_id(session: Session, id: int) -> Offer:  
 """Получить предложение по ID."""  
 offer = session.get(Offer, id)  
 if not offer:  
 raise HTTPException(status\_code=404, detail="Предложение не найдено")  
 return offer  
  
def get\_offers\_by\_user(session: Session, user\_id: int) -> list[Offer]:  
 """Получить список предложений пользователя."""  
 stmt = select(Offer).where(Offer.executor\_id == user\_id)  
 return session.scalars(stmt).all()  
  
def update\_offer\_by\_id(session: Session, data: OfferUpdate, id: int) -> Offer:  
 """Обновить данные предложения по ID."""  
 offer = get\_offer\_by\_id(session, id)  
 update\_data = data.model\_dump(exclude\_unset=True)  
 for key, value in update\_data.items():  
 setattr(offer, key, value)  
 try:  
 session.commit()  
 session.refresh(offer)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при обновлении предложения: {e}")  
 return offer  
  
def delete\_offer\_by\_id(session: Session, id: int):  
 """Удалить предложение по ID."""  
 offer = get\_offer\_by\_id(session, id)  
 session.delete(offer)  
 try:  
 session.commit()  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при удалении предложения: {e}")

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\services\order.py

from fastapi.exceptions import HTTPException  
from sqlalchemy import select  
from sqlalchemy.orm import Session  
from sqlalchemy.exc import SQLAlchemyError  
from app.core.models.order import Order  
from app.core.schemas.order import OrderCreate, OrderUpdate  
from app.core.services.category import get\_category\_by\_id  
  
def create\_order(session: Session, data: OrderCreate, customer\_id: int) -> Order:  
 """Создать новый заказ."""  
 get\_category\_by\_id(session, data.category\_id) # Проверка существования категории  
 order\_data = data.model\_dump()  
 order = Order(\*\*order\_data, customer\_id=customer\_id)  
 session.add(order)  
 try:  
 session.commit()  
 session.refresh(order)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при создании заказа: {e}")  
 return order  
  
def get\_order\_by\_id(session: Session, id: int) -> Order:  
 """Получить заказ по ID."""  
 order = session.get(Order, id)  
 if not order:  
 raise HTTPException(status\_code=404, detail="Заказ не найден")  
 return order  
  
def get\_orders\_by\_user(session: Session, user\_id: int) -> list[Order]:  
 """Получить список заказов пользователя."""  
 stmt = select(Order).where((Order.customer\_id == user\_id) | (Order.executor\_id == user\_id))  
 return session.scalars(stmt).all()  
  
def get\_available\_orders(session: Session) -> list[Order]:  
 """Получить список всех заказов со статусом 'В\_ожидании'."""  
 stmt = select(Order).where(Order.status == "В\_ожидании", Order.executor\_id.is\_(None))  
 return session.scalars(stmt).all()  
  
def update\_order\_by\_id(session: Session, data: OrderUpdate, id: int) -> Order:  
 """Обновить данные заказа по ID."""  
 order = get\_order\_by\_id(session, id)  
 update\_data = data.model\_dump(exclude\_unset=True)  
 for key, value in update\_data.items():  
 setattr(order, key, value)  
 try:  
 session.commit()  
 session.refresh(order)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при обновлении заказа: {e}")  
 return order  
  
def delete\_order\_by\_id(session: Session, id: int):  
 """Удалить заказ по ID."""  
 order = get\_order\_by\_id(session, id)  
 session.delete(order)  
 try:  
 session.commit()  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при удалении заказа: {e}")

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\services\review.py

from fastapi.exceptions import HTTPException  
from sqlalchemy import select  
from sqlalchemy.orm import Session  
from sqlalchemy.exc import SQLAlchemyError  
from app.core.models.review import Review  
from app.core.models.order import Order  
from app.core.schemas.review import ReviewCreate, ReviewUpdate  
  
def create\_review(session: Session, data: ReviewCreate, author\_id: int) -> Review:  
 """Создать новый отзыв."""  
 order = session.get(Order, data.order\_id)  
 if not order or order.customer\_id != author\_id or order.status != "completed":  
 raise HTTPException(status\_code=400, detail="Недопустимый или незавершенный заказ")  
 review\_data = data.model\_dump()  
 review = Review(\*\*review\_data, author\_id=author\_id)  
 session.add(review)  
 try:  
 session.commit()  
 session.refresh(review)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при создании отзыва: {e}")  
 return review  
  
def get\_review\_by\_id(session: Session, id: int) -> Review:  
 """Получить отзыв по ID."""  
 review = session.get(Review, id)  
 if not review:  
 raise HTTPException(status\_code=404, detail="Отзыв не найден")  
 return review  
  
def get\_reviews\_by\_user(session: Session, user\_id: int) -> list[Review]:  
 """Получить список отзывов пользователя."""  
 stmt = select(Review).where(Review.author\_id == user\_id)  
 return session.scalars(stmt).all()  
  
def update\_review\_by\_id(session: Session, data: ReviewUpdate, id: int) -> Review:  
 """Обновить данные отзыва по ID."""  
 review = get\_review\_by\_id(session, id)  
 update\_data = data.model\_dump(exclude\_unset=True)  
 for key, value in update\_data.items():  
 setattr(review, key, value)  
 try:  
 session.commit()  
 session.refresh(review)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при обновлении отзыва: {e}")  
 return review  
  
def delete\_review\_by\_id(session: Session, id: int):  
 """Удалить отзыв по ID."""  
 review = get\_review\_by\_id(session, id)  
 session.delete(review)  
 try:  
 session.commit()  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при удалении отзыва: {e}")

## Файл: C:/Users/User/PycharmProjects/telegram\_service/app\core\services\user.py

from fastapi.exceptions import HTTPException  
from sqlalchemy import select  
from sqlalchemy.orm import Session  
from sqlalchemy.exc import SQLAlchemyError, IntegrityError  
from app.core.models.user import User  
from app.core.models.category import Category  
from app.core.schemas.user import UserCreate, UserUpdate  
from app.core.services.city import get\_city\_by\_id  
  
def create\_user(session: Session, data: UserCreate) -> User:  
 """Создать нового пользователя."""  
 from app.api.depends.user import ADMIN\_TELEGRAM\_ID # Импортируем здесь  
 get\_city\_by\_id(session, data.city\_id) # Проверка существования города  
 user\_data = data.model\_dump(exclude={"category\_ids"}) # Исключаем category\_ids из данных  
 # Устанавливаем is\_admin=True, если telegram\_id совпадает с ADMIN\_TELEGRAM\_ID  
 if user\_data["telegram\_id"] == ADMIN\_TELEGRAM\_ID:  
 user\_data["is\_admin"] = True  
 user = User(\*\*user\_data)  
 if data.category\_ids: # Если указаны категории  
 categories = session.query(Category).filter(Category.id.in\_(data.category\_ids)).all()  
 if len(categories) != len(data.category\_ids):  
 raise HTTPException(status\_code=404, detail="Одна или несколько категорий не найдены")  
 user.categories = categories  
 session.add(user)  
 try:  
 session.commit()  
 session.refresh(user)  
 except IntegrityError:  
 session.rollback()  
 raise HTTPException(status\_code=400, detail="Пользователь с таким telegram\_id или username уже существует")  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при создании пользователя: {e}")  
 return user  
  
def get\_user\_by\_id(session: Session, id: int) -> User:  
 """Получить пользователя по ID."""  
 user = session.get(User, id)  
 if not user:  
 raise HTTPException(status\_code=404, detail="Пользователь не найден")  
 return user  
  
def get\_users(session: Session) -> list[User]:  
 """Получить список всех пользователей."""  
 return session.scalars(select(User)).all()  
  
def update\_user\_by\_id(session: Session, data: UserUpdate, id: int) -> User:  
 """Обновить данные пользователя по ID."""  
 from app.api.depends.user import ADMIN\_TELEGRAM\_ID # Импортируем здесь  
 user = get\_user\_by\_id(session, id)  
 update\_data = data.model\_dump(exclude\_unset=True, exclude\_none=True)  
 if "city\_id" in update\_data:  
 get\_city\_by\_id(session, data.city\_id) # Проверка существования города  
 if "category\_ids" in update\_data and data.category\_ids is not None:  
 categories = session.query(Category).filter(Category.id.in\_(data.category\_ids)).all()  
 if len(categories) != len(data.category\_ids):  
 raise HTTPException(status\_code=404, detail="Одна или несколько категорий не найдены")  
 user.categories = categories  
 del update\_data["category\_ids"]  
 for key, value in update\_data.items():  
 setattr(user, key, value)  
 # Синхронизируем is\_admin с ADMIN\_TELEGRAM\_ID  
 if user.telegram\_id == ADMIN\_TELEGRAM\_ID:  
 user.is\_admin = True  
 try:  
 session.commit()  
 session.refresh(user)  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при обновлении пользователя: {e}")  
 return user  
  
def delete\_user\_by\_id(session: Session, id: int):  
 """Удалить пользователя по ID."""  
 user = get\_user\_by\_id(session, id)  
 session.delete(user)  
 try:  
 session.commit()  
 except SQLAlchemyError as e:  
 session.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка при удалении пользователя: {e}")