Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)
Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 805 «Математическая кибернетика»

Отчёт по курсовому проекту по курсу «Базы данных»

Тема: «Каталог автомобилей» (Python+PostgreSQL)

Подготовили:
студенты группы М8О-303Б-21
Сайфуллин И.
Своеволин И.
Ковалев А.

Проверила:
Кузнецова С.В.

Москва, 2023

Техническое задание

Необходимо разработать прикладное программное обеспечение, которое обеспечит работу с базой данных автомобилей. Главными элементами базы данных являются автомобили, для каждого из автомобилей хранится информация о его комплектации (производитель, марка, модель), пробеге, цене, цвете и др. роль.

Более подробное представление БД можно посмотреть на ER диаграмме.

Постановка задачи

Разработать прикладное программное обеспечение, представляющее собой телеграмм бота , предоставляющий удобный интерфейс для лиц заинтересованных в покупке или продаже автомобиля

Функциональность

Продавец:

- Добавление автомобиля на продажу с указанием:
  - пробега
  - цвета
  - типа коробки передач
  - мощности
  - вида топлива
  - объем двигателя
  - модели
  - марки
  - цены
  - номера владельца
  - прикрепление фотографии
  - состояния

Покупатель:

- Поиск автомобиля по фильтрам
- Получения информации по авто с возможностью связаться с владельцем
- Просмотр фото авто

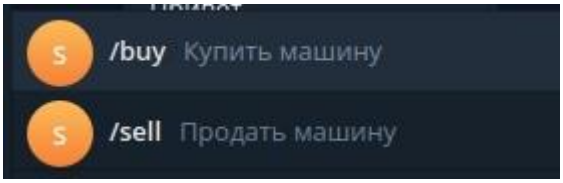
Обоснование выбора СУБД

Мы выбрали Python в качестве основного ЯП по причинам:
1) Объектной ориентированности языка для работы с асинхронной ORM (async sqlalchemy orm)
2) Читабельности кода благодаря выделению блоков кода отступами, поддержка type hinting для подсказки типов данных в Python, которые помогают разработчикам лучше понимать и контролировать типы данных, используемые в коде
3) Высокой скорости разработки
4) Гибкости разработки благодаря динамической типизации и поддержки множества библиотек
5) Возможности использовать асинхронный конкурентный код благодаря поддержке asyncio

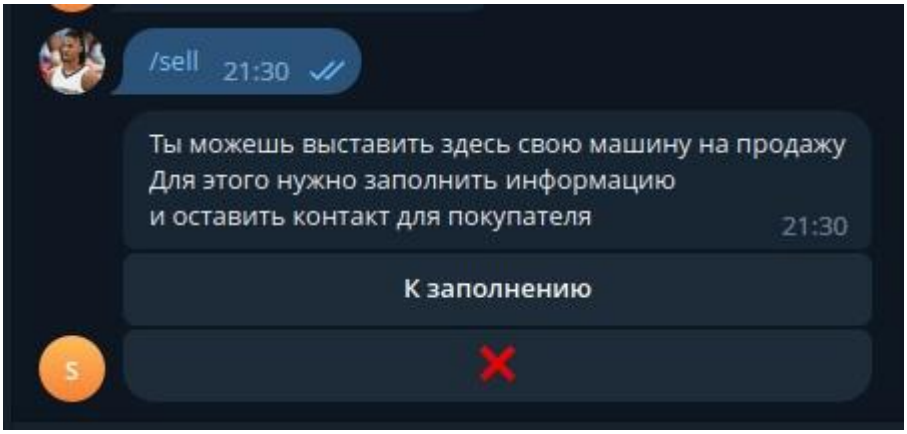
PostgreSQL больше подходит для ситуаций, когда:
1) важна возможность создания селектов (вложенных подзапросов);
2) нужна возможность создания сложных команд SQL (за счет соответствия SQL-стандартам ANSI);
3) важна целостность данных;
4) требуется поддержка MVCC для предоставления одновременного доступа к базе данных большому количеству пользователей на чтение и запись;
(MVCC - (Multiversion Concurrency Control, Многоверсионное управление конкурентным доступом). Это означает, что каждый SQL-оператор видит снимок данных (версию базы данных) на определённый момент времени, вне зависимости от текущего состояния данных. Это защищает операторы от несогласованности данных, возможной, если другие конкурирующие транзакции внесут изменения в те же строки данных, и обеспечивает тем самым изоляцию транзакций для каждого сеанса баз данных. MVCC, отходя от методик блокирования, принятых в традиционных СУБД, снижает уровень конфликтов блокировок и таким образом обеспечивает более высокую производительность в многопользовательской среде.)
5) предполагается выполнение сложных процедур и расширение БД;

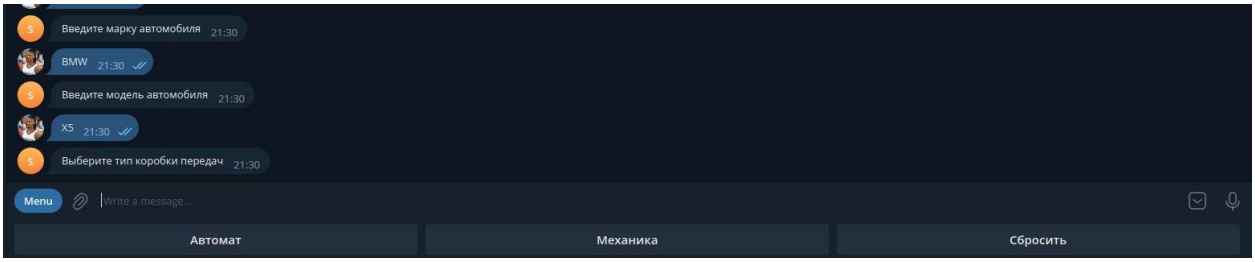
Интерфейс

Основные возможности бота:

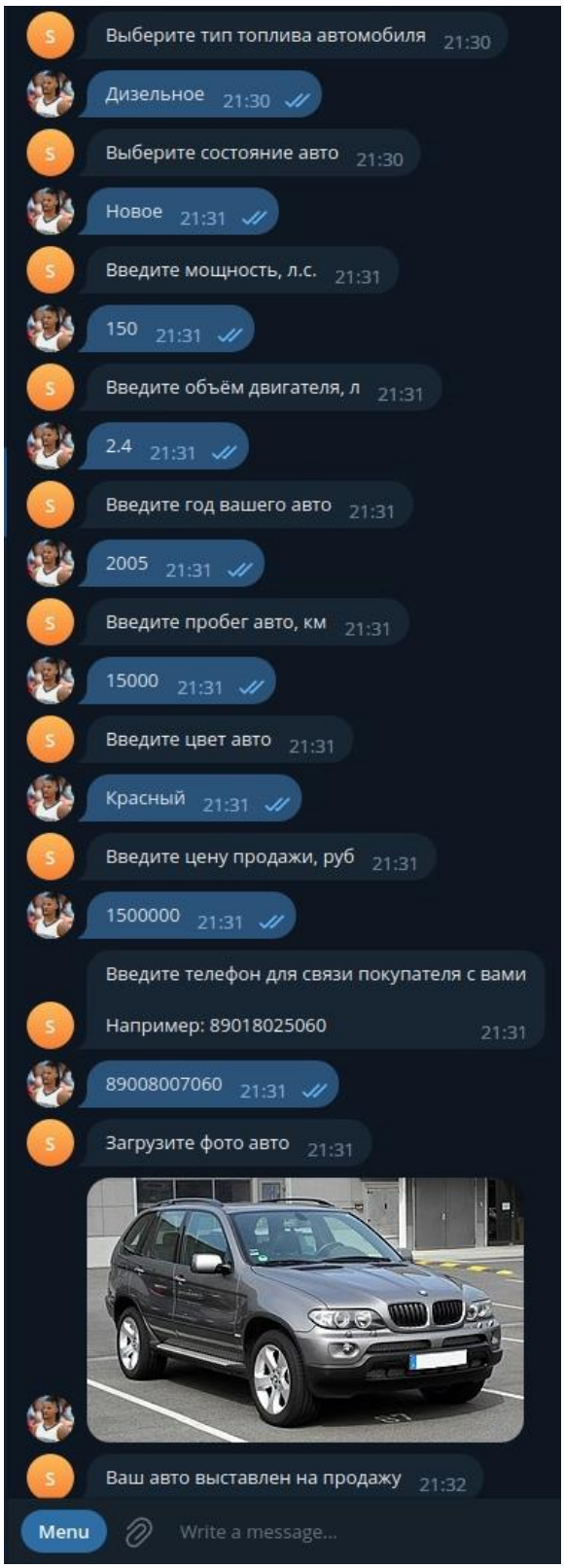


/sell - команда для клиента заинтересованного в продаже автомобиля, по нажатию будет предложено заполнить анкету с информацией об авто и владельце

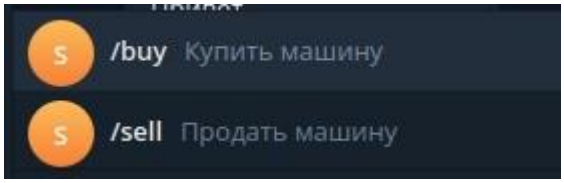




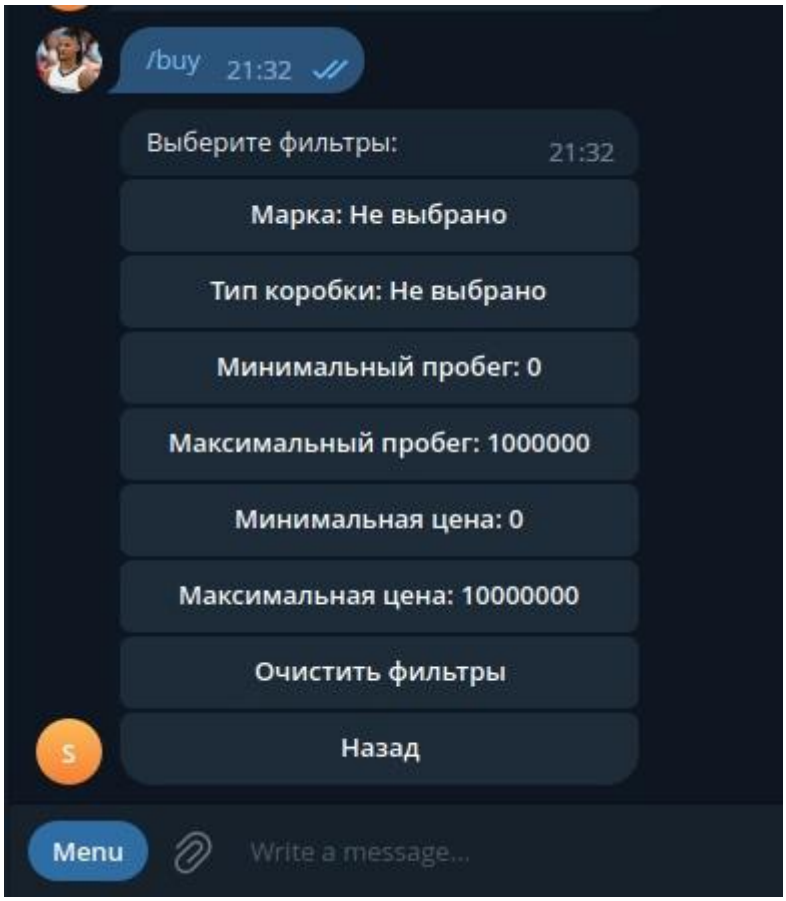
Во время заполнения всплывают подсказки автозаполнения некоторых полей (на примере выше появляются варианты Автомат/Механика)



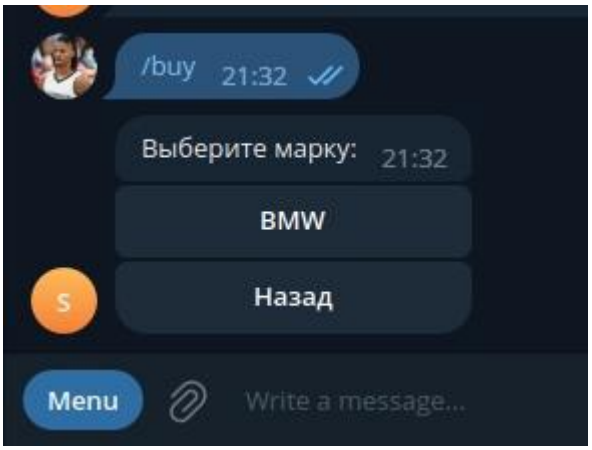
После успешного заполнения всех полей автомобиль будет доступен в каталоге покупки у всех пользователей



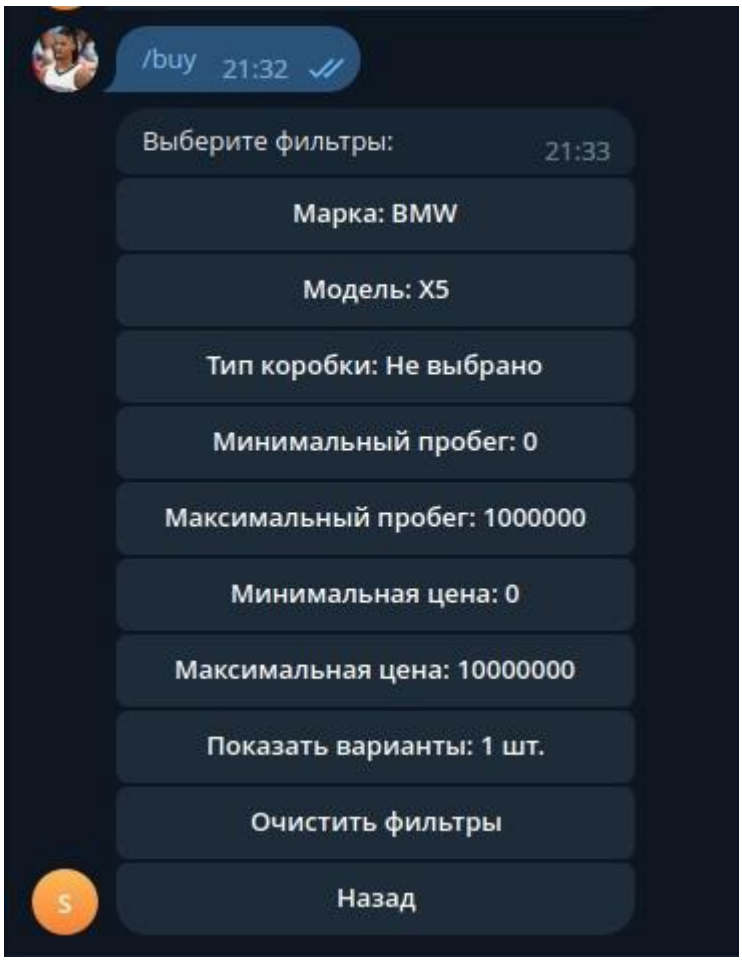
/buy - команда для клиента заинтересованного в покупке автомобиля, по нажатию будет открыто меню с возможностью подбора автомобиля по фильтрам



По нажатию на кнопки будет предложено ввести свои предпочтения Например, по нажатию на кнопку “Марка” выпадет список всех марок автомобилей находящихся на продаже

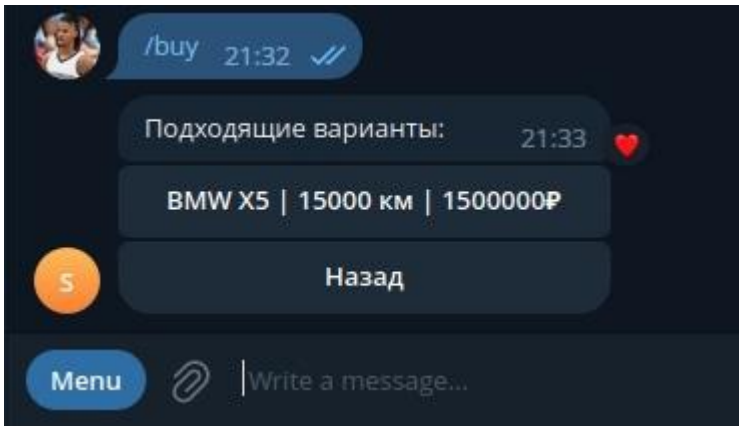


После выбора нужной марки клиент попадает в основное меню покупки с выбранным фильтром марки, аналогично и для модели:



После выбора фильтров добавляется кнопка “Показать варианты: N”, где N - количество машин подходящих под фильтры

По нажатию на эту кнопку выпадет меню с краткой информациях об автомобилях подходящих под фильтры:



После выбора нужного и нажатия бот отправит фото авто с главной информацией:


/buy 21:32
Марка: BMW
Модель: X5
Коробка: Автомат
Пробег: 15000
Цена: 1500000₽
Номер владельца: 89008007060
21:33
S
Назад

Используемые технологии:

SQLAlchemy ORM – программное обеспечение с открытым исходным кодом. Используется для работы с базами данных через язык запросов SQL. Использует для работы технологии ORM (Object Relational Mapping). С ее помощью удается связать базы данных с концепциями объектно-ориентированных языков разработки.

Alembic - это инструмент для работы с миграциями для SQLAlchemy

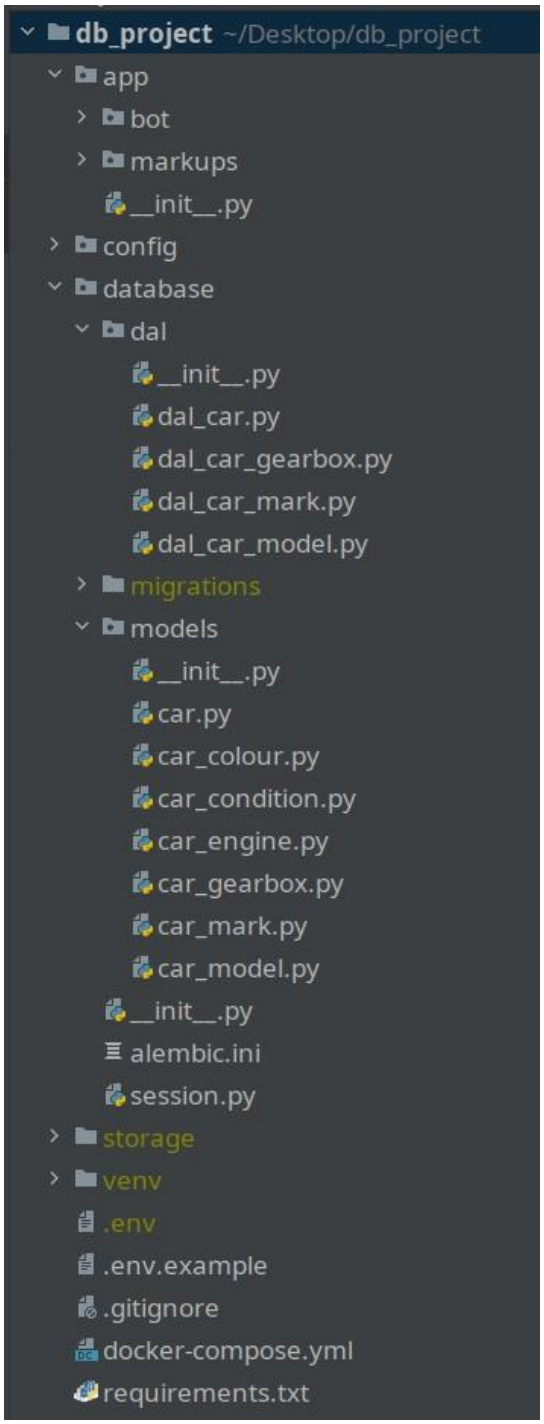
Миграции - это способ распространять изменения, которые вы внесены в модели (добавление поля, удаление модели и т. д.), в схему вашей базы данных.

pytelegrambotapi - Python библиотека для работы с Telegram API

docker-compose - средство для определения и запуска приложений Docker с несколькими контейнерами

Docker – это программная платформа для быстрой разработки, тестирования и развертывания приложений. Docker упаковывает ПО в стандартизованные блоки, которые называются контейнерами. Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Благодаря Docker можно быстро развертывать и масштабировать приложения в любой среде и сохранять уверенность в том, что код будет работать.

Структура проекта:



requirements.txt - файл с необходимыми зависимостями для запуска проекта
docker-compose.yml - файл конфигурации для запуска контейнеров, таких как pgAdmin и postgresql
storage - директория для хранения медиафайлов
models - директория с моделями БД

```python
from sqlalchemy import Integer, ForeignKey, String, SmallInteger
from sqlalchemy.orm import mapped_column, relationship, Mapped
from database.models.car_colour import CarColour
from database.models.car_condition import CarCondition
from database.models.car_model import CarModel
from database.models.car_engine import CarEngine
from database.models.car_gearbox import CarGearbox
from database.models import Base

 Svoevolin
class Car(Base):
    __tablename__ = "car"

    id_car = mapped_column(Integer, primary_key=True, autoincrement=True)
    id_colour = mapped_column(ForeignKey('car_colour.id_colour'), nullable=False)
    id_condition = mapped_column(ForeignKey('car_condition.id_condition'), nullable=False)
    id_model = mapped_column(ForeignKey('car_model.id_model'), nullable=False)
    id_engine = mapped_column(ForeignKey('car_engine.id_engine'), nullable=False)
    id_gearbox = mapped_column(ForeignKey('car_gearbox.id_gearbox'), nullable=False)
    year = mapped_column(SmallInteger, nullable=False)
    mileage = mapped_column(Integer, nullable=False)
    price = mapped_column(Integer, nullable=False)
    owner_mobile = mapped_column(String, nullable=False)
    dir_photo = mapped_column(String, nullable=False)

    colour: Mapped["CarColour"] = relationship(back_populates="cars")
    condition: Mapped["CarCondition"] = relationship(back_populates="cars")
    model: Mapped["CarModel"] = relationship(back_populates="cars")
    engine: Mapped["CarEngine"] = relationship(back_populates="cars")
    gearbox: Mapped["CarGearbox"] = relationship(back_populates="cars")

```

```python
from typing import List
from sqlalchemy import String, Integer
from sqlalchemy.orm import mapped_column, relationship, Mapped
from database.models import Base


 Svoevolin
class CarColour(Base):
    __tablename__ = "car_colour"

    id_colour = mapped_column(Integer, primary_key=True, autoincrement=True)
    name_colour = mapped_column(String(20), nullable=False)

    cars: Mapped[List["Car"]] = relationship(back_populates="colour")

```

```python
from typing import List
from sqlalchemy import String, Integer, Double
from sqlalchemy.orm import mapped_column, relationship, Mapped
from database.models import Base


 Svoevolin
class CarEngine(Base):
    __tablename__ = "car_engine"

    id_engine = mapped_column(Integer, primary_key=True, autoincrement=True)
    fuel_type = mapped_column(String(20), nullable=False)
    hoursepower = mapped_column(Integer, nullable=False)
    caparcity = mapped_column(String(4), nullable=False)

    cars: Mapped[List["Car"]] = relationship(back_populates="engine")

```

```python
from typing import List
from sqlalchemy import String, Integer, SmallInteger
from sqlalchemy.orm import mapped_column, Mapped, relationship
from database.models import Base


Svoevolin
class CarGearbox(Base):
    __tablename__ = "car_gearbox"

    id_gearbox = mapped_column(Integer, primary_key=True, autoincrement=True)
    gear_type = mapped_column(String(20), nullable=False)

    cars: Mapped[List["Car"]] = relationship(back_populates="gearbox")

```

```python
from typing import List
from sqlalchemy import Integer, String
from sqlalchemy.orm import mapped_column, relationship, Mapped
from database.models import Base


Svoevolin
class CarMark(Base):
    __tablename__ = "car_mark"

    id_mark = mapped_column(Integer, primary_key=True, autoincrement=True)
    name_mark = mapped_column(String(20), nullable=False)

    models: Mapped[List["CarModel"]] = relationship(back_populates="mark")

```

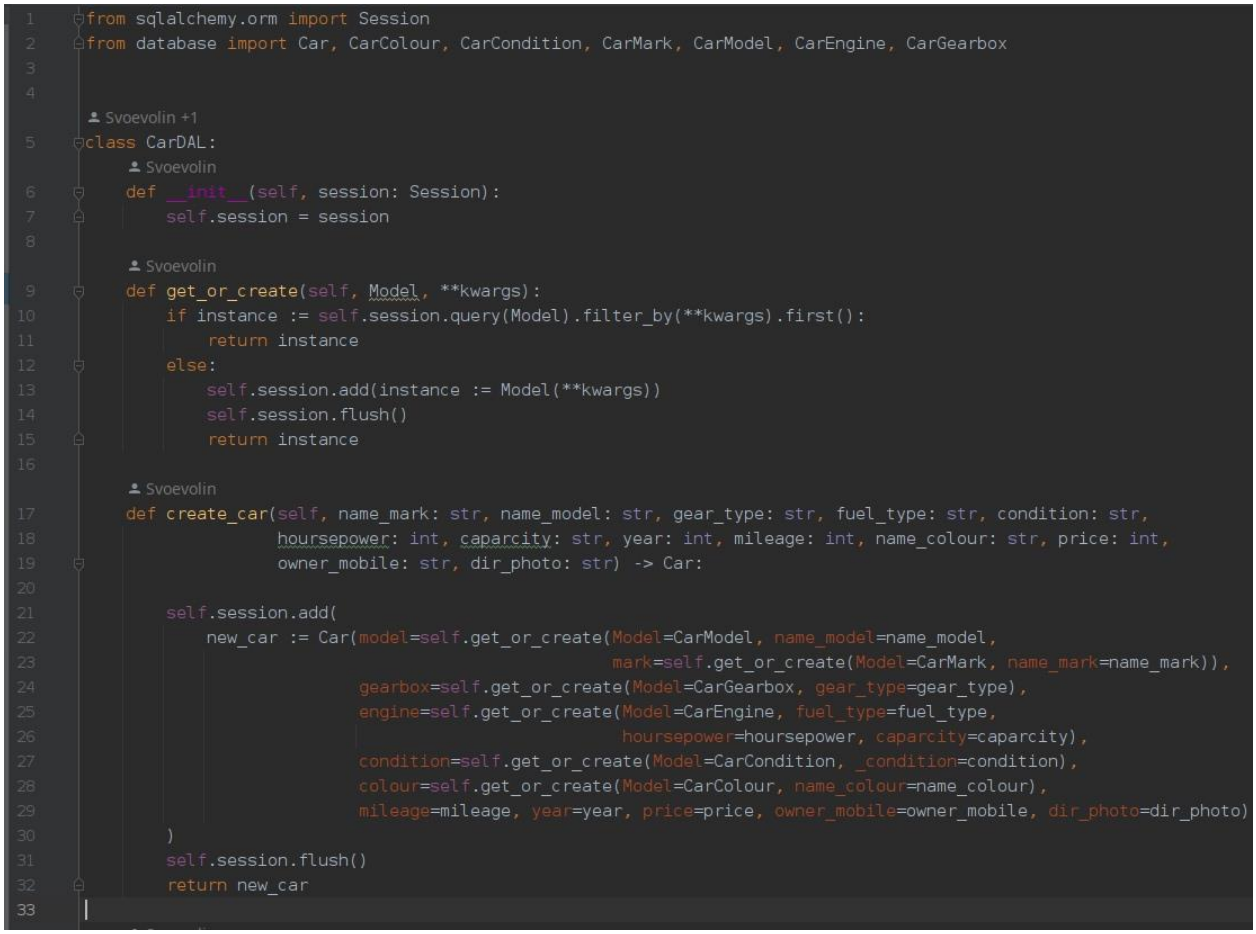
```python
from typing import List
from sqlalchemy import Integer, String, ForeignKey
from sqlalchemy.orm import mapped_column, relationship, Mapped
from database.models import Base


Svoevolin
class CarModel(Base):
    __tablename__ = "car_model"

    id_model = mapped_column(Integer, primary_key=True, autoincrement=True)
    id_mark = mapped_column(ForeignKey('car_mark.id_mark'), nullable=False)
    name_model = mapped_column(String(20), nullable=False)

    mark: Mapped[List["CarMark"]] = relationship(back_populates="models")
    cars: Mapped[List["Car"]] = relationship(back_populates="model")

```

session - модуль содержащий класс Sessionmaker, подключающийся к базе данных и манипулирующий транзакциями БД

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from config.config import Config, load_config

config: Config = load_config()

engine = create_engine(
    url=f"postgresql:"
        f"//{config.database.user}"
        f":{config.database.password}"
        f"@{config.database.host}"
        f":{config.database.port}"
        f"/{config.database.name}",
    echo=False
)

Session = sessionmaker(bind=engine, expire_on_commit=False)
```

migrations - автоматически созданная Alembic’ом директория (Alembic - инструмент для работы с миграциями для SQLAlchemy)

dal - Data Access Layer - слой доступа к данным (содержит методы/запросы к БД)

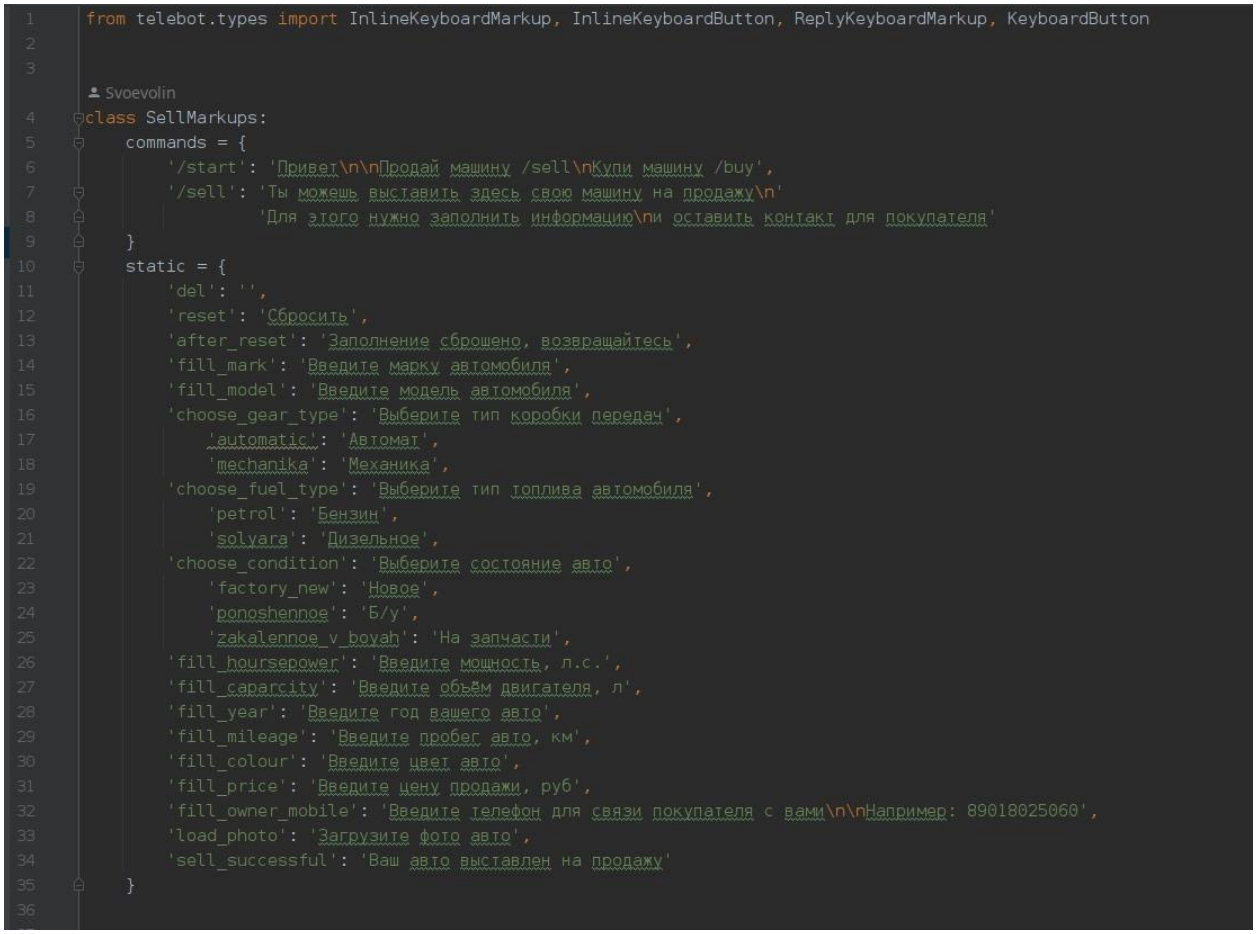


```python
from sqlalchemy.orm import Session
from database import Car, CarColour, CarCondition, CarMark, CarModel, CarEngine, CarGearbox


Svoevolin +1
class CarDAL:
    Svoevolin
    def __init__(self, session: Session):
        self.session = session

    Svoevolin
    def get_or_create(self, Model, **kwargs):
        if instance := self.session.query(Model).filter_by(**kwargs).first():
            return instance
        else:
            self.session.add(instance := Model(**kwargs))
            self.session.flush()
            return instance

    Svoevolin
    def create_car(self, name_mark: str, name_model: str, gear_type: str, fuel_type: str, condition: str,
                   hoursepower: int, caparcity: str, year: int, mileage: int, name_colour: str, price: int,
                   owner_mobile: str, dir_photo: str) -> Car:

        self.session.add(
            new_car := Car(model=self.get_or_create(Model=CarModel, name_model=name_model,
                                                    mark=self.get_or_create(Model=CarMark, name_mark=name_mark)),
                           gearbox=self.get_or_create(Model=CarGearbox, gear_type=gear_type),
                           engine=self.get_or_create(Model=CarEngine, fuel_type=fuel_type,
                                                     hoursepower=hoursepower, caparcity=caparcity),
                           condition=self.get_or_create(Model=CarCondition, _condition=condition),
                           colour=self.get_or_create(Model=CarColour, name_colour=name_colour),
                           mileage=mileage, year=year, price=price, owner_mobile=owner_mobile, dir_photo=dir_photo)
        )
        self.session.flush()
        return new_car
```

get_or_create(signature) - метод, обеспечивающий наличие только единственного экземпляра с указанными полями (адаптация Singleton)
create_car(data) - метод создающий запись о новой машине в БД, при этом все связные таблицы заполняются новыми данными, если нужные не были созданы ранее (используется метод Singleton get_or_create(signature))

config - директория содержащая конфигурационные данные проекта

markups - директория с текстовыми данными сообщений в боте, а также с описанием меню для взаимодействия



```python
from telebot.types import InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboardMarkup, KeyboardButton


Svoevolin
class SellMarkups:
    commands = {
        '/start': 'Привет\n\nПродай машину /sell\nКупи машину /buy',
        '/sell': 'Ты можешь выставить здесь свою машину на продажу\n'
                 'Для этого нужно заполнить информацию\nи оставить контакt для покупателя'
    }
    static = {
        'del': '',
        'reset': 'Сбросить',
        'after_reset': 'Заполнение сброшено, возвращайтесь',
        'fill_mark': 'Введите марку автомобиля',
        'fill_model': 'Введите модель автомобиля',
        'choose_gear_type': 'Выберите тип коробки передач',
            'automatic': 'Автомат',
            'mechanika': 'Механика',
        'choose_fuel_type': 'Выберите тип топлива автомобиля',
            'petrol': 'Бензин',
            'solyara': 'Дизельное',
        'choose_condition': 'Выберите состояние авто',
            'factory_new': 'Новое',
            'ponoshennoe': 'Б/у',
            'zakalennoe_v_boyah': 'На запчасти',
        'fill_hoursepower': 'Введите мощность, л.с.',
        'fill_caparcity': 'Введите объём двигателя, л',
        'fill_year': 'Введите год вашего авто',
        'fill_mileage': 'Введите пробег авто, км',
        'fill_colour': 'Введите цвет авто',
        'fill_price': 'Введите цену продажи, руб',
        'fill_owner_mobile': 'Введите телефон для связи покупателя с вами\n\nНапример: 89018025060',
        'load_photo': 'Загрузите фото авто',
        'sell_successful': 'Ваш авто выставлен на продажу'
    }

```

bot - основная директория содержащая всю логику работы интерфейса

Подготовили:
Своеволин И.
Сайфуллин И.
Ковалев А.
Среда разработки: Python 3.11

Вывод

Данный проект помог нам освоить принципы создания интерфейса для базы данных, а также:
Освоить сервис удобного и быстрого развёртывания проектов docker и docker compose.
Взаимодействие с базой данных происходило с помощью SQLAlchemy ORM, его использование повысило читаемость кода и позволило работать с данными в парадигме ООП.

Использование миграций с помощью Alembic позволило изменять структуру моделей и связей между ними прямо во время разработки клиента, что повысило удобство и скорость написания кода.
Pytelegrambotapi - библиотека для работы с Telegram Bot API, которая использовалась в проекте - дала полное понимание работы API телеграмма

Все перечисленные инструменты используются в продакшн разработке и были полноценно освоены на этом проекте.

Получившийся сервис при некоторых доработках может стать полноценным законченным продуктом, имеющим спрос в коммерции, на рынке.