Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Лабораторная работа №2
Работа с SQLAlchemy и alembic

Студент группы РИМ – 150950: _____ Вальнева А.Д.

Екатеринбург 2025

Цель работы

Освоить принципы работы с библиотеками SQLAlchemy и Alembic для создания и управления реляционными базами данных на Python, изучить механизмы миграции базы данных.

Задачи

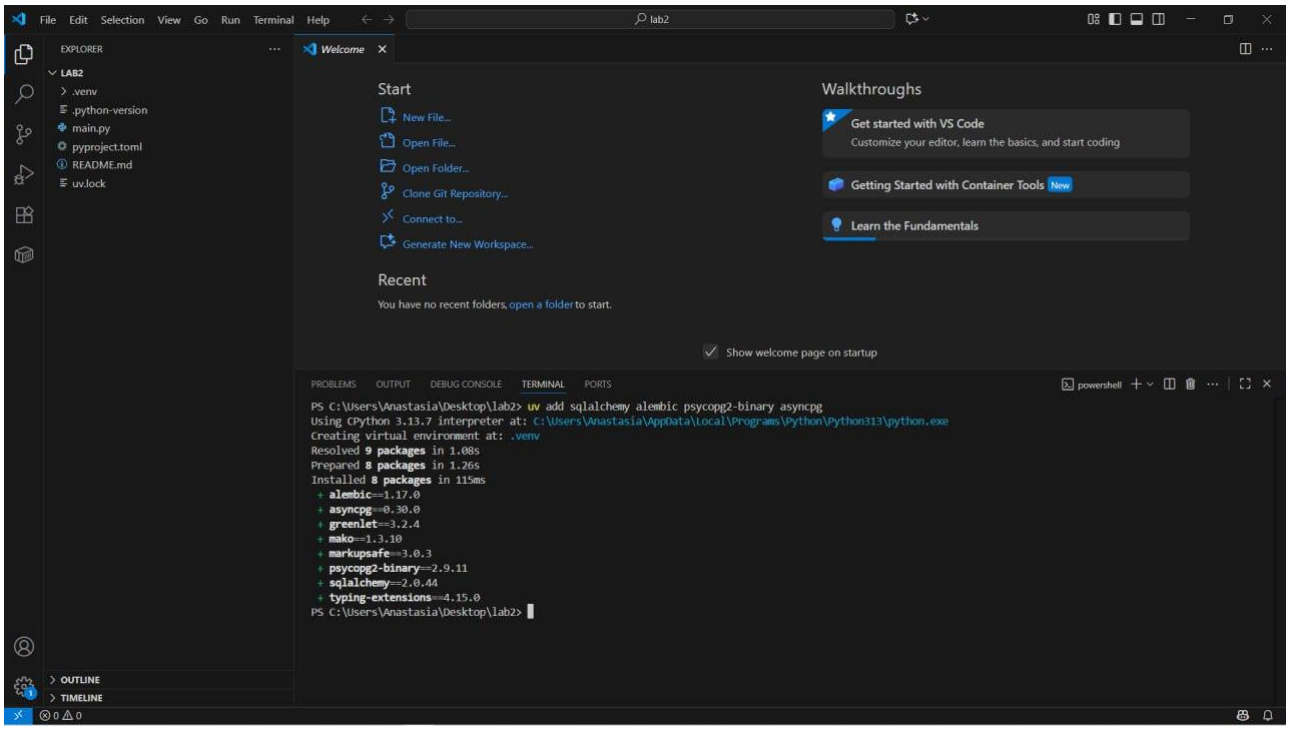
1. Подготовить окружение для работы с базой данных, установив необходимые библиотеки.
2. Проектирование и реализация ORM-моделей.
3. Работа с системой миграций Alembic.
4. Применение миграций и управление версиями БД
5. Создание CRUD-операций и тестовых данных

Структура проекта

```
Lab2/
├── alembic.ini              # Конфигурация Alembic
├── migrations/              # Миграции Alembic
│   ├── versions/
│   ├── env.py
│
├── db/                      # Папка с моделями и базовым классом
│   ├── models/
│   │   ├── user.py          # Модель пользователя
│   │   ├── address.py       # Модель адреса
│   │   ├── order.py         # Модель заказа
│   │   └── product.py       # Модель продукта
│   │
│   ├── base.py              # Базовый класс моделей
│   └── session.py           # Настройки сессии БД
│
├── crud/
│   └── user.py              # Ф-ии для работы с пользователями
│   └── address.py           # Файл для работы с адресами
│   └── order.py
│   └── product.py
│
├── main.py                  # Главный скрипт
```

Часть 1: Инициализация БД

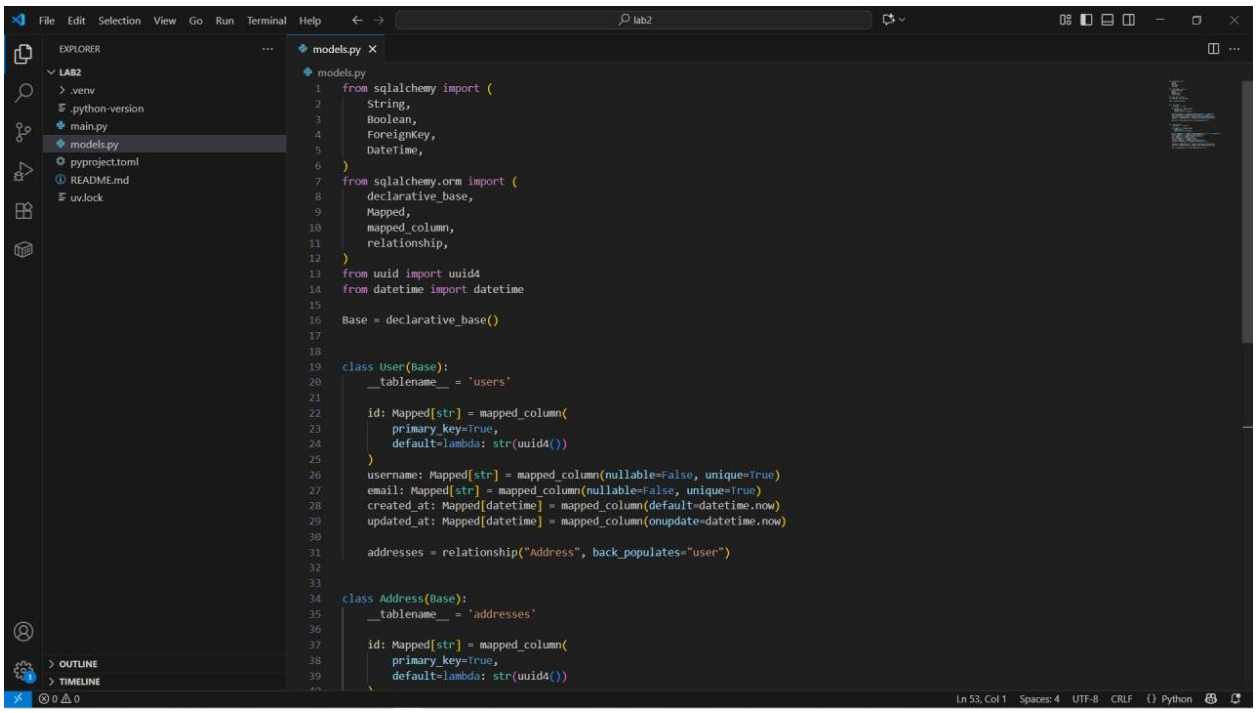
Задача: подготовить окружение для работы с базой данных, установив необходимые библиотеки



Создаем новый проект с помощью команды **uv init**
В терминале запускаем команду установки пакетов

- **uv add sqlalchemy alebic psycorg- binary asyncpg**

Установлено 8 пакетов в виртуальное окружение (файл uv.lock обновлен; создана папка .venv для изолированного окружения).




```python
from sqlalchemy import (
    String,
    Boolean,
    ForeignKey,
    DateTime,
)
from sqlalchemy.orm import (
    declarative_base,
    Mapped,
    mapped_column,
    relationship,
)
from uuid import uuid4
from datetime import datetime

Base = declarative_base()


class User(Base):
    __tablename__ = 'users'

    id: Mapped[str] = mapped_column(
        primary_key=True,
        default=lambda: str(uuid4())
    )
    username: Mapped[str] = mapped_column(nullable=False, unique=True)
    email: Mapped[str] = mapped_column(nullable=False, unique=True)
    created_at: Mapped[datetime] = mapped_column(default=datetime.now)
    updated_at: Mapped[datetime] = mapped_column(onupdate=datetime.now)

    addresses = relationship("Address", back_populates="user")


class Address(Base):
    __tablename__ = 'addresses'

    id: Mapped[str] = mapped_column(
        primary_key=True,
        default=lambda: str(uuid4())
    )
```

Задача: определить структуру таблиц БД через ORM-модели для пользователя и адреса.

Создали файл models.py

Импорты:

from sqlalchemy import String, DateTime, ForeignKey
from sqlalchemy.orm import declarative_base, Mapped, mapped_column, relationship
from uuid import uuid4
from datetime import datetime

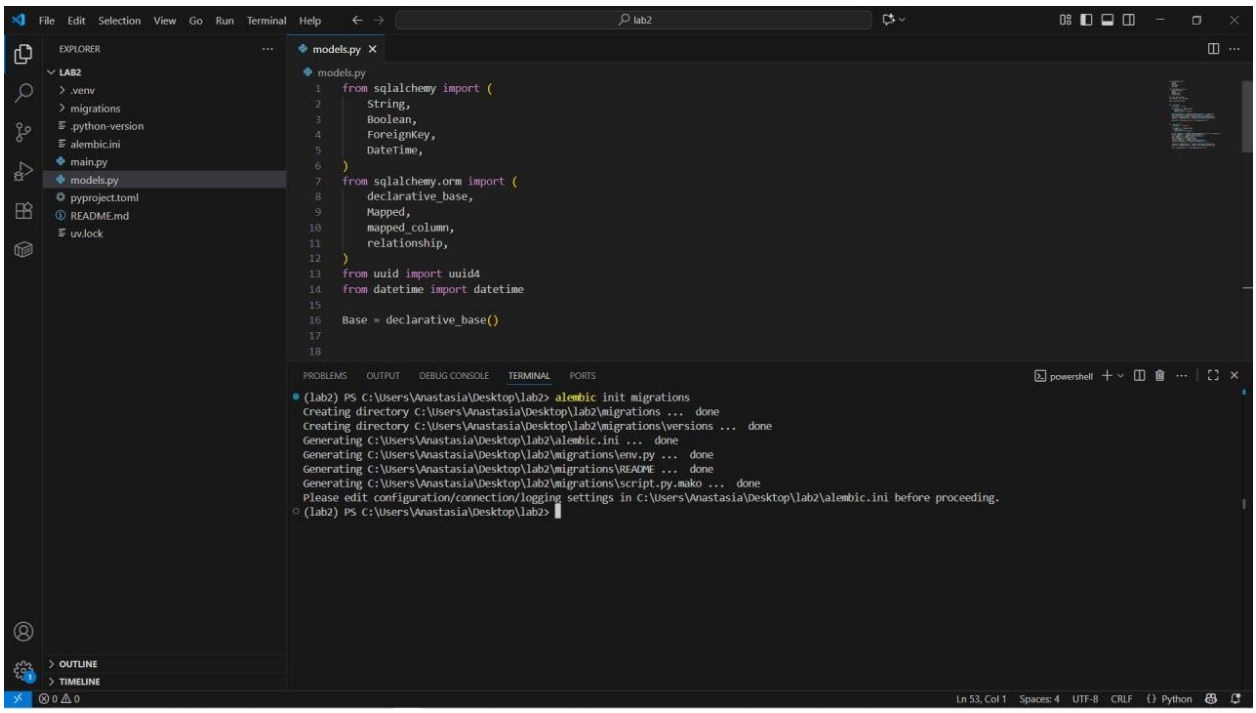
Импорты из SQLAlchemy ORM

- declarative_base (фабрика для создания базового класса моделей)
- Mapped и mapped_column (для аннотаций полей)
- Relationship (определение связей между моделями (отношения))

Стандартные импорты Python

- uuid4 (для генерации идентификаторов)
- datetime из datetime (работа с датой и временем)

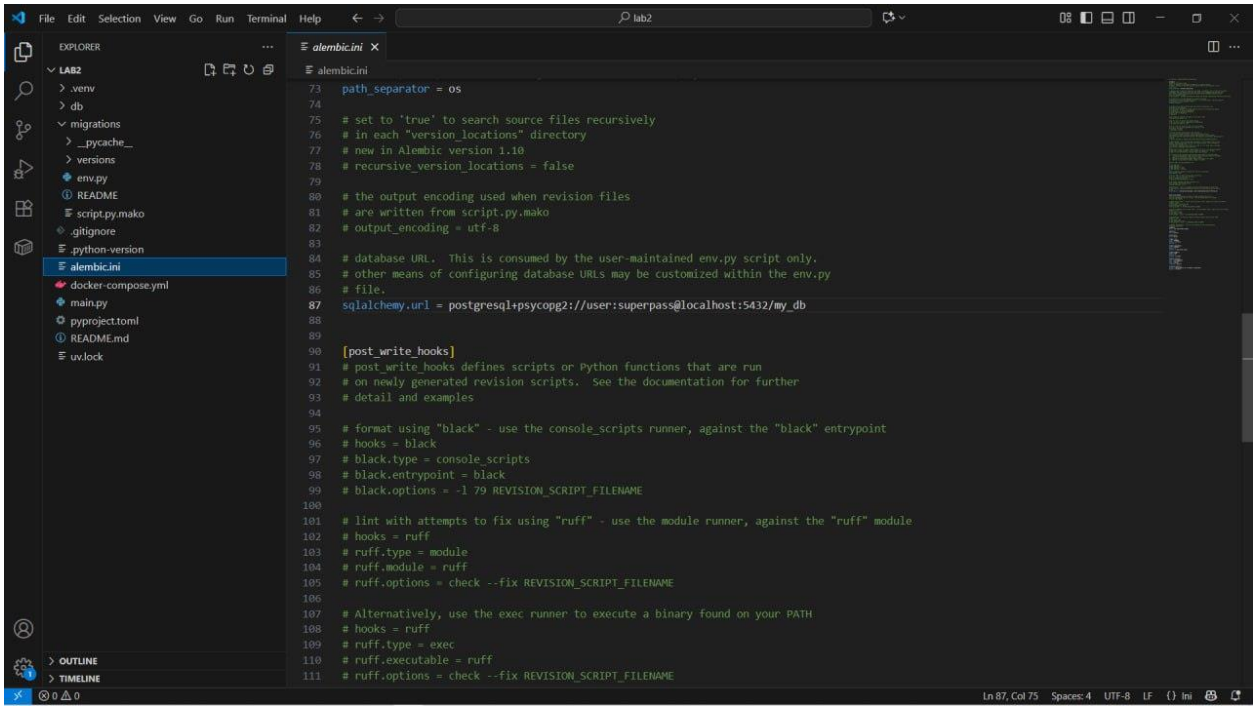
Объявили 2 класса: User и Address (определяем базовый класс с помощью declarative_base())



Инициализация миграций Alembic

- **alembic init migrations.**

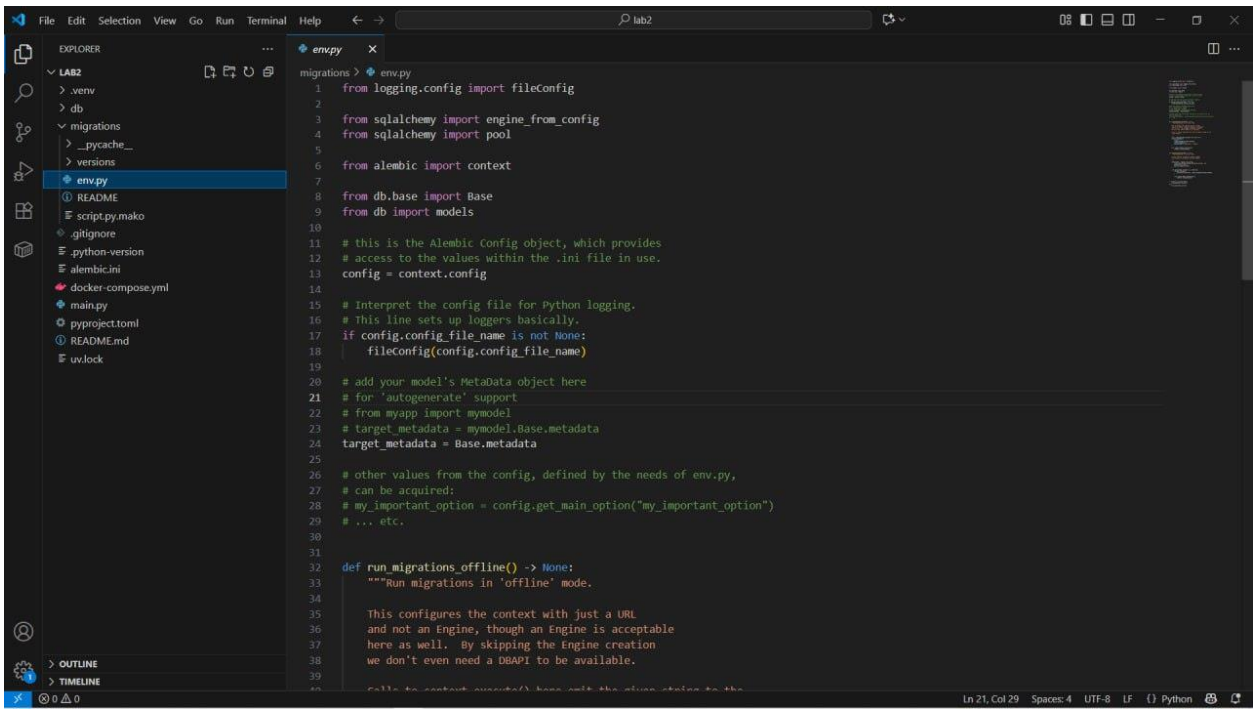
Создана папка migrations с необходимыми файлами.



С помощью docker compose развернута БД Postgres из лабораторной работы 1.

В файле alembic.ini в корне проекта меняем строку подключения к БД:

- sqlalchemy.url = postgresql+psycopg2://user:superpass@localhost:5432/my_db

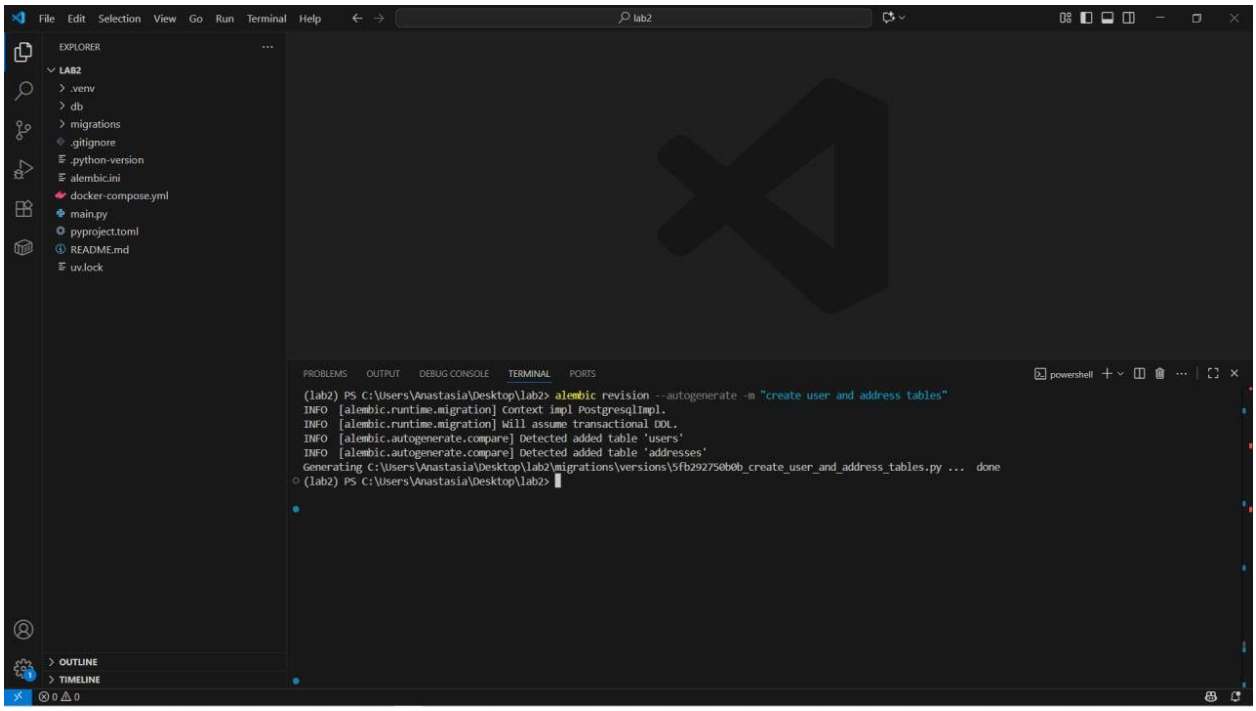


Импорты:

from db.base import Base # Для метаданных
from db import models # Для обнаружения всех моделей

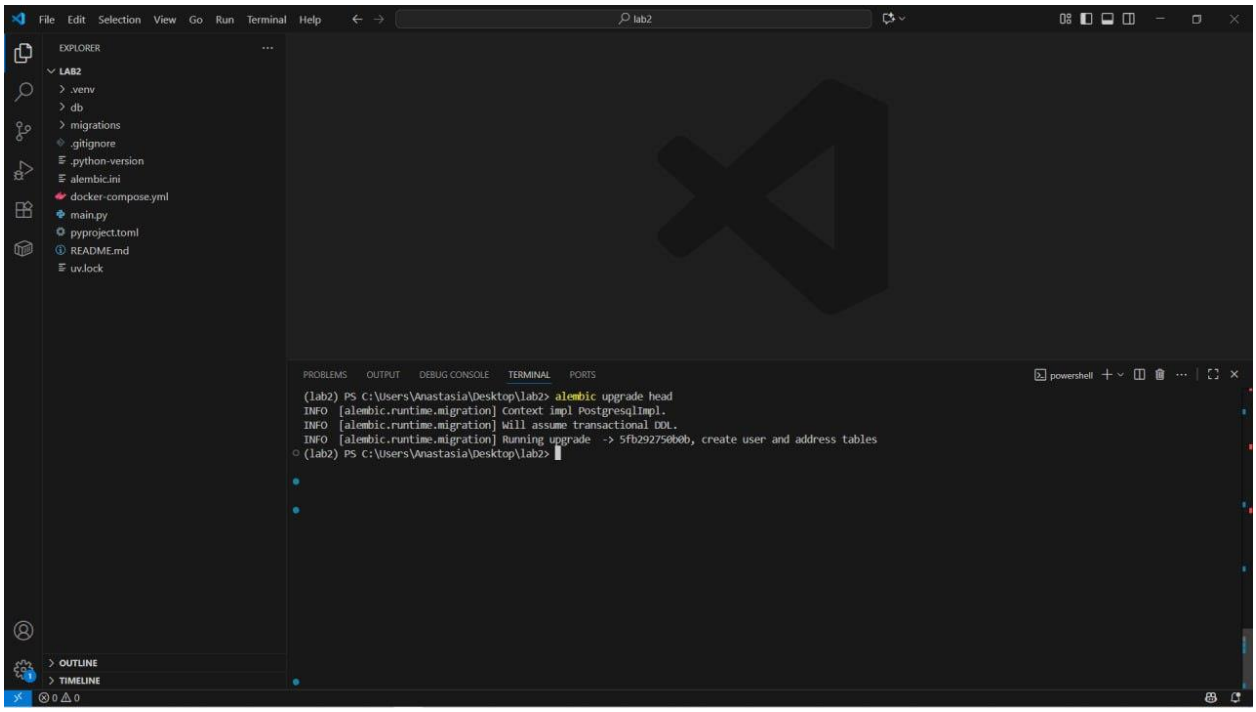
Установлены метаданные для миграций:

- target_metadata = Base.metadata



Создана первая миграция

- alembic revision --autogenerate -m "create user and address tables"

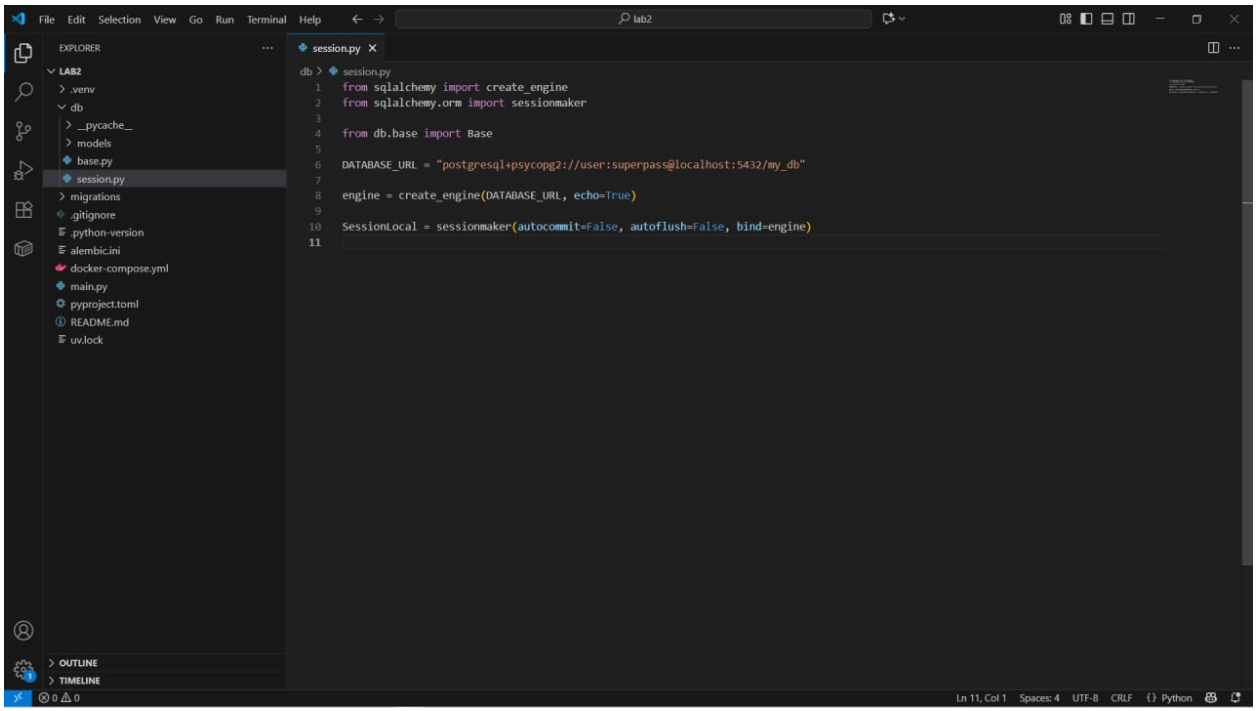


Миграция применена к БД:

- alembic upgrade head

Задача: наполнить БД данными

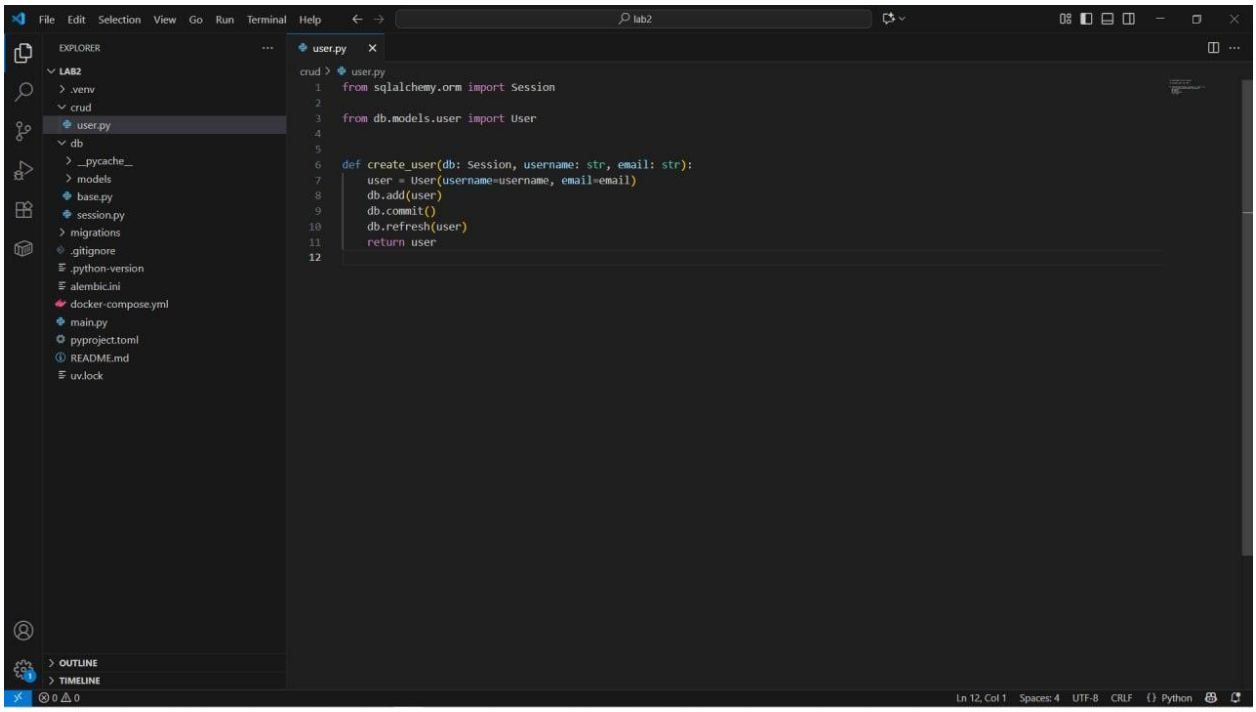
1) Настройка подключения к базе данных (db/ session.py)



- В session.py задается строка подключения к базе данных (DATABASE_URL)

- Создается фабрика подключений
    - engine = create_engine(DATABASE_URL, echo=True)
- SessionLocal - фабрика сессий для работы с БД

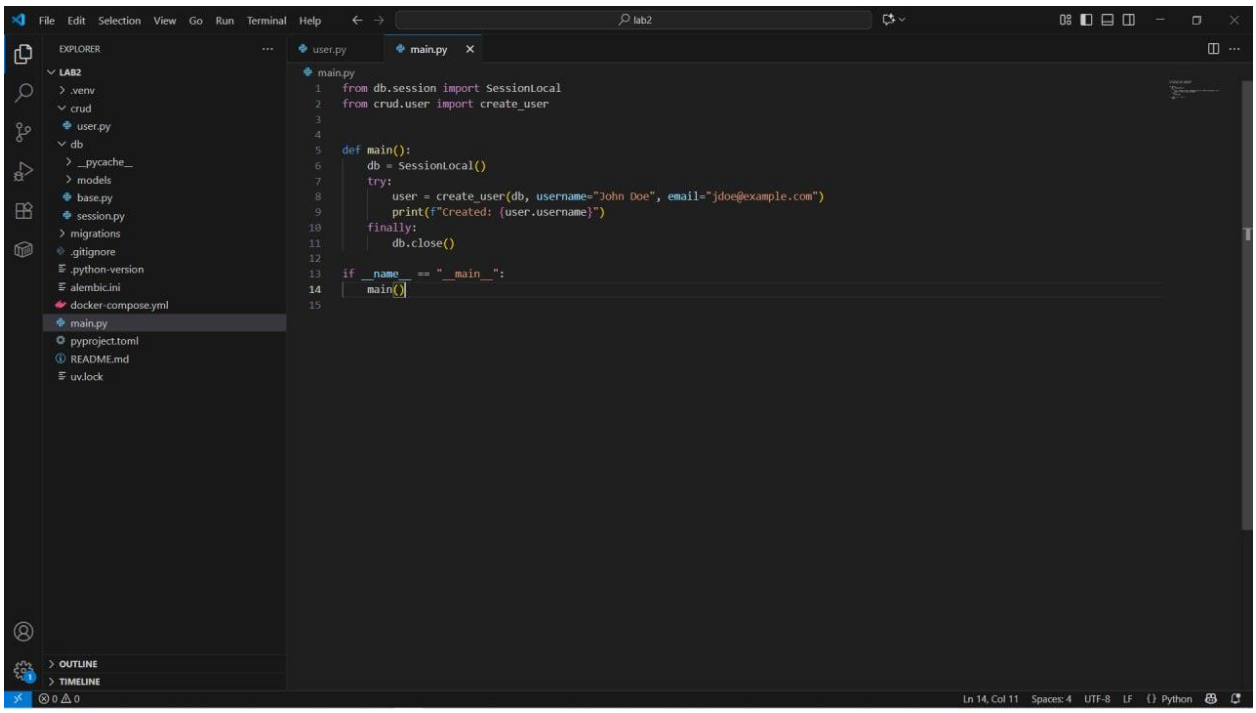
2) Функция создания пользователя (crud/user.py)



Функция create_user:

- принимает сессию базы данных, имя пользователя и email;
- создает объект пользователя, добавляет его в сессию (db.add(user)), сохраняет в БД (db.commit()) и обновляет объект из БД (db.refresh(user)) ;
- возвращает созданного пользователя;

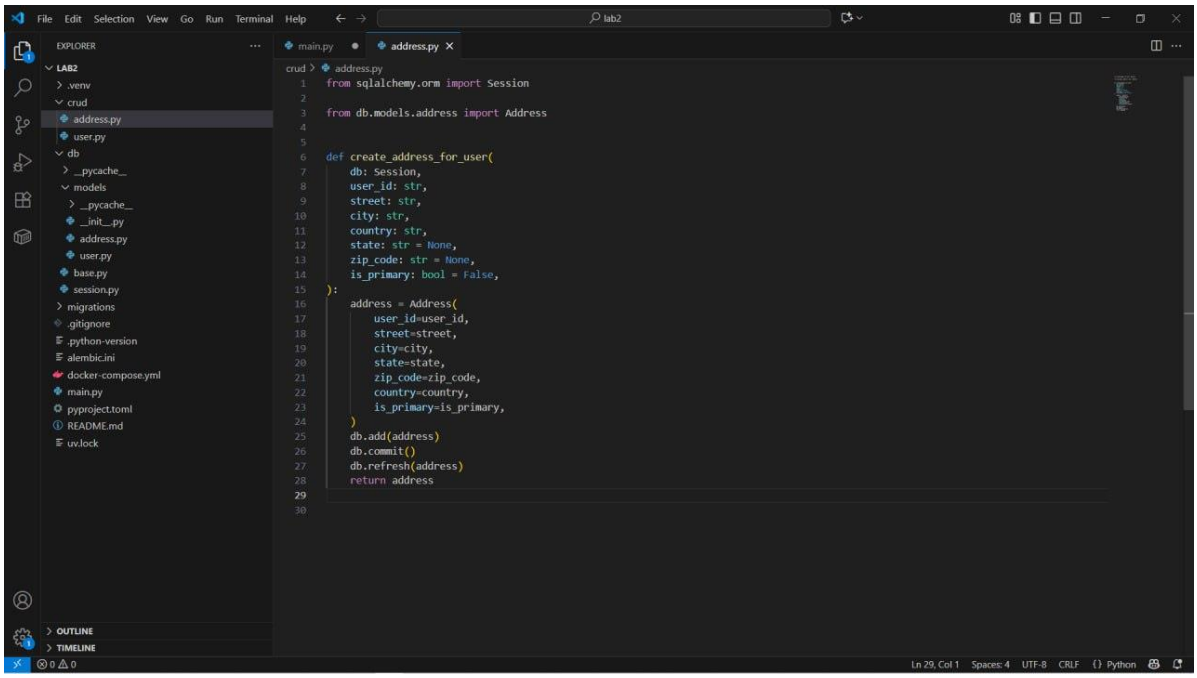
3) Файл main.py



Функция main():

- Создает сессию БД;
- В блоке try/finally гарантирует закрытие сессии;
- Создает одного тестового пользователя;
- Выводит результат;

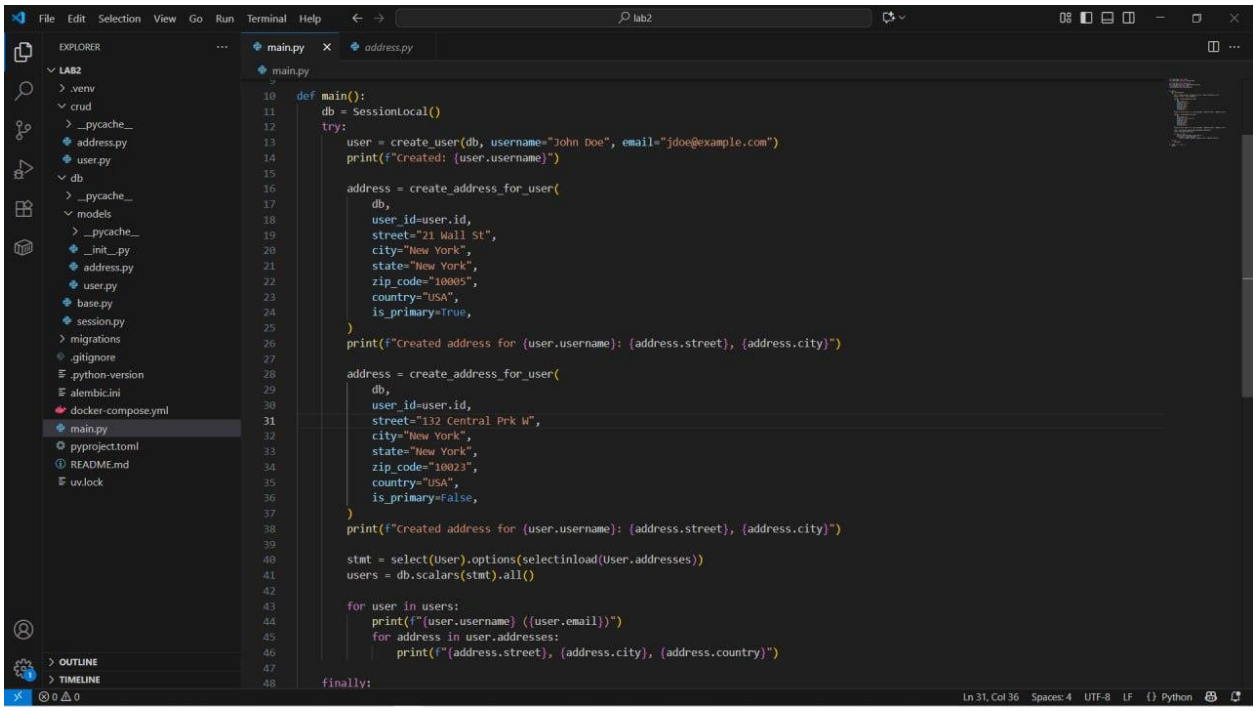
4) Функция создания адреса (crud/address.py)



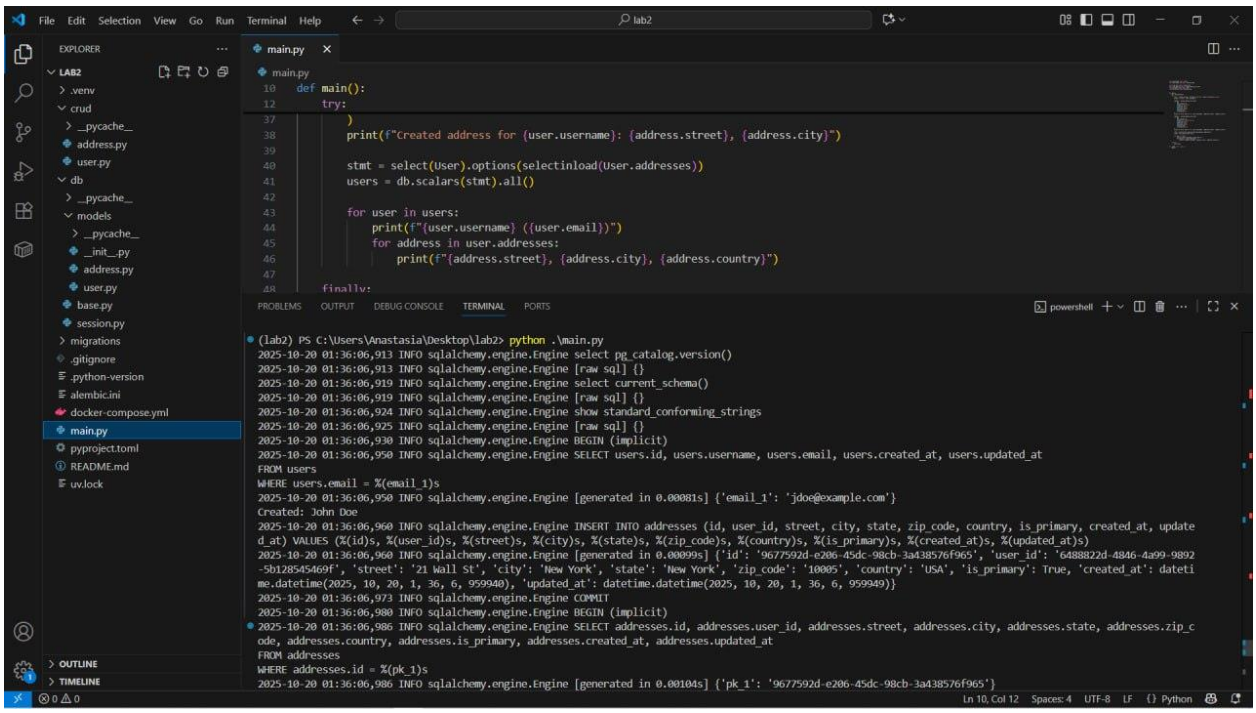
Функция (create_address_for_user) для работы с адресами:

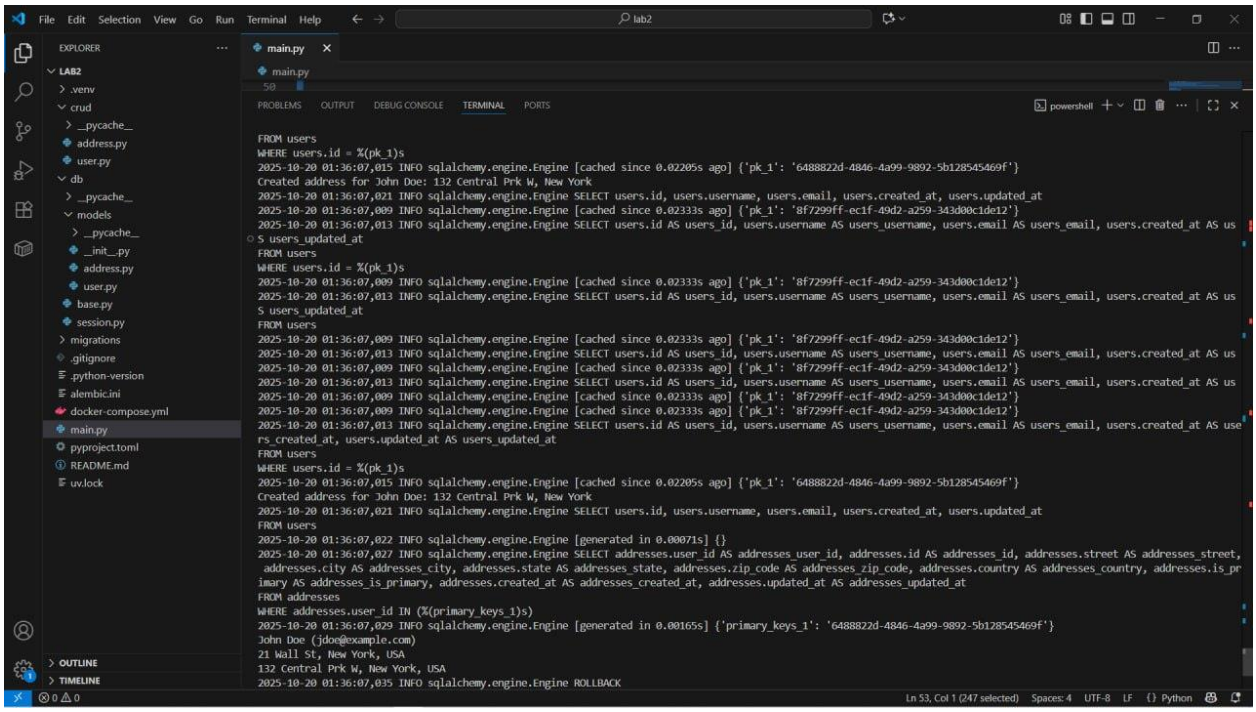
- Принимает все необходимые параметры адреса (сессию БД, user_id, улицу, город, страну, штат, почтовый код, флаг is_primary)
- Создает новый адрес для пользователя с теми же параметрами
- Сохраняет адрес в базу данных
- Возвращает созданный объект адреса

Часть 2: Запрос связанных данных



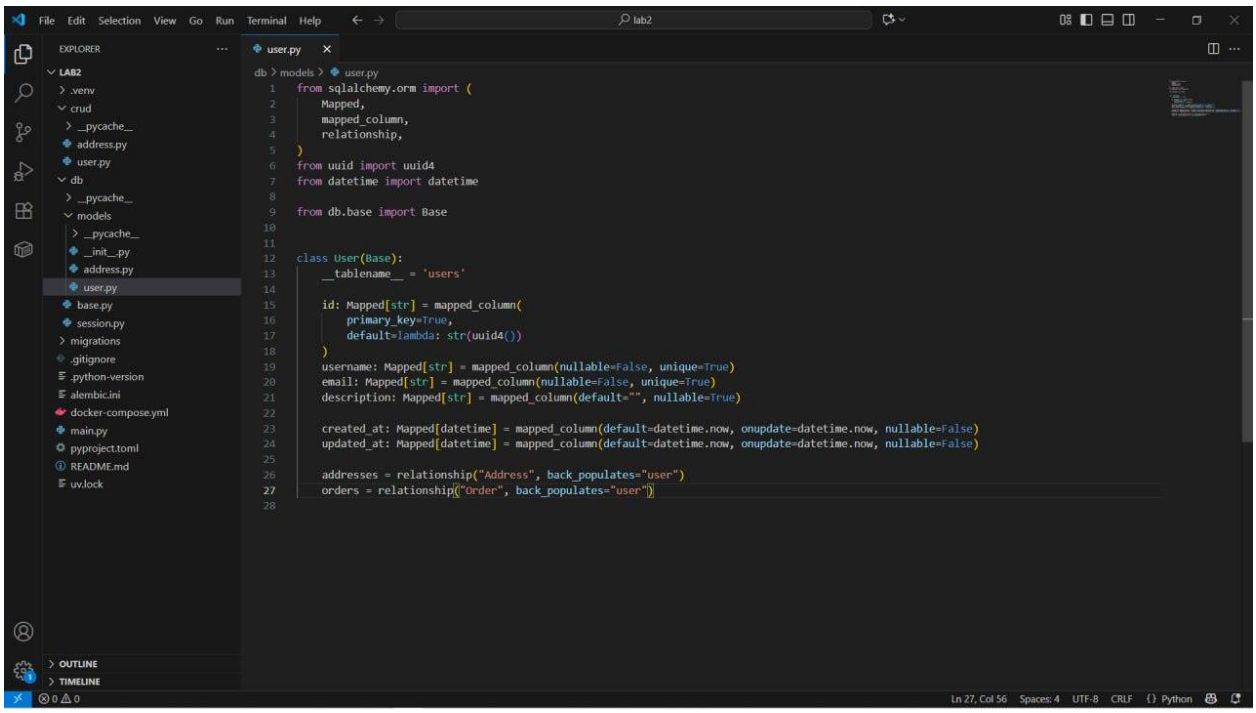
- Создание сессии БД: db = SessionLocal()
- Добавление пользователя: create_user(db, username=..., email=...)
- Добавление двух адресов для предварительно созданного пользователя: create_address_for_user(db, user_id=..., street=..., city=..., ...).
- После наполнения выполняем запрос на выборку всех пользователей с адресами (чтобы убедиться, что данные добавлены) и выводим их.





Часть 3: Последующие работы с БД и миграции

1) Модификация модели User (models/user.py)

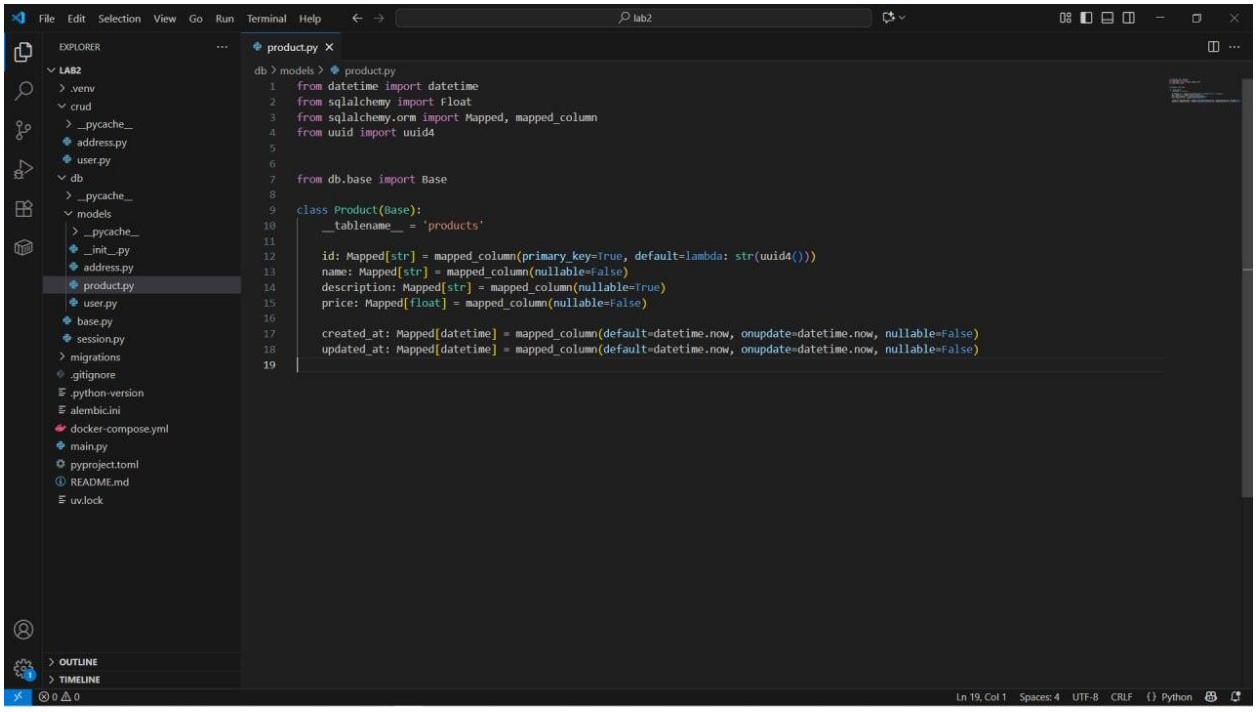


Добавлено новое строковое поле description:

- description: Mapped[str] = mapped_column(default="", nullable=True)

    - Тип: Mapped[str]
    - Значение по умолчанию: пустая строка ("").
    - Разрешено значение NULL (nullable=True)

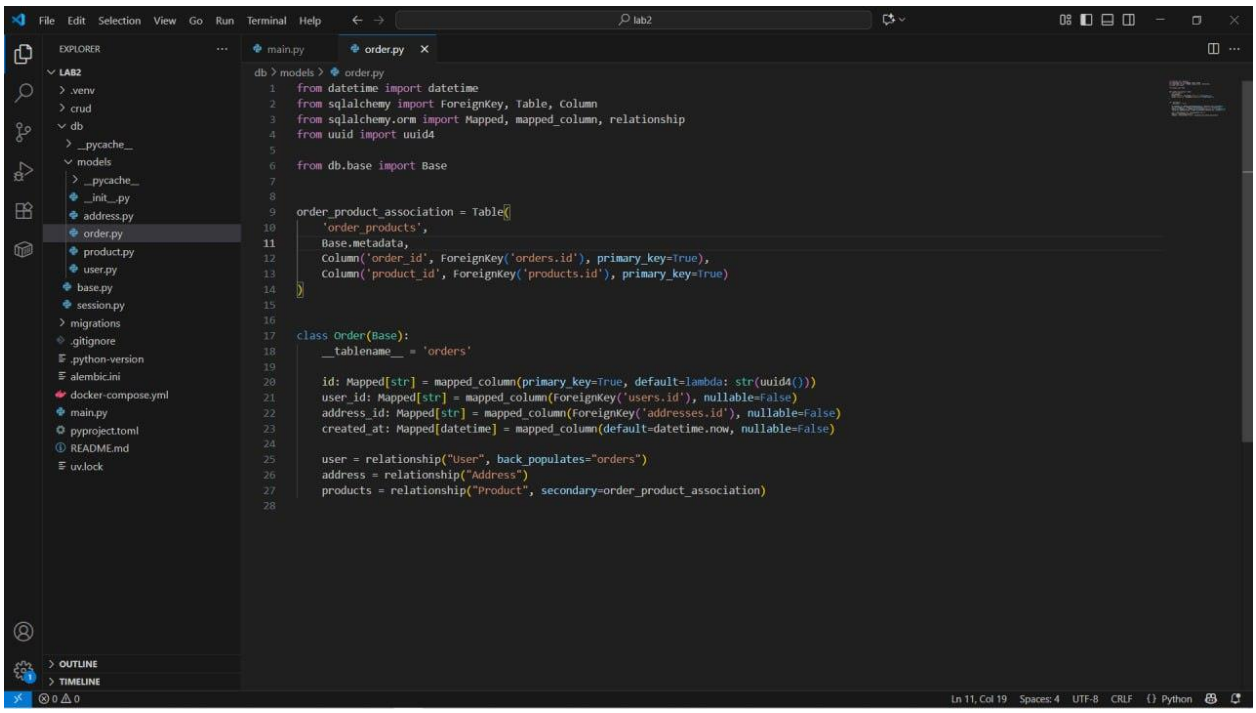
2) Создание модели Product (models/product.py)



Создана модель Product со следующими полями:

- id (первичный ключ, UUID)
- name (название продукта, не NULL)
- description (описание, может быть NULL)
- price (цена, не NULL)
- created_at и updated_at (даты создания и обновления)

3) Создание модели Order и ассоциативной таблицы (models/order.py)



Модель Order: связывает пользователя (user_id), адрес доставки (address_id) и продукты **через ассоциативную таблицу order_products.**

Поля модели Order:

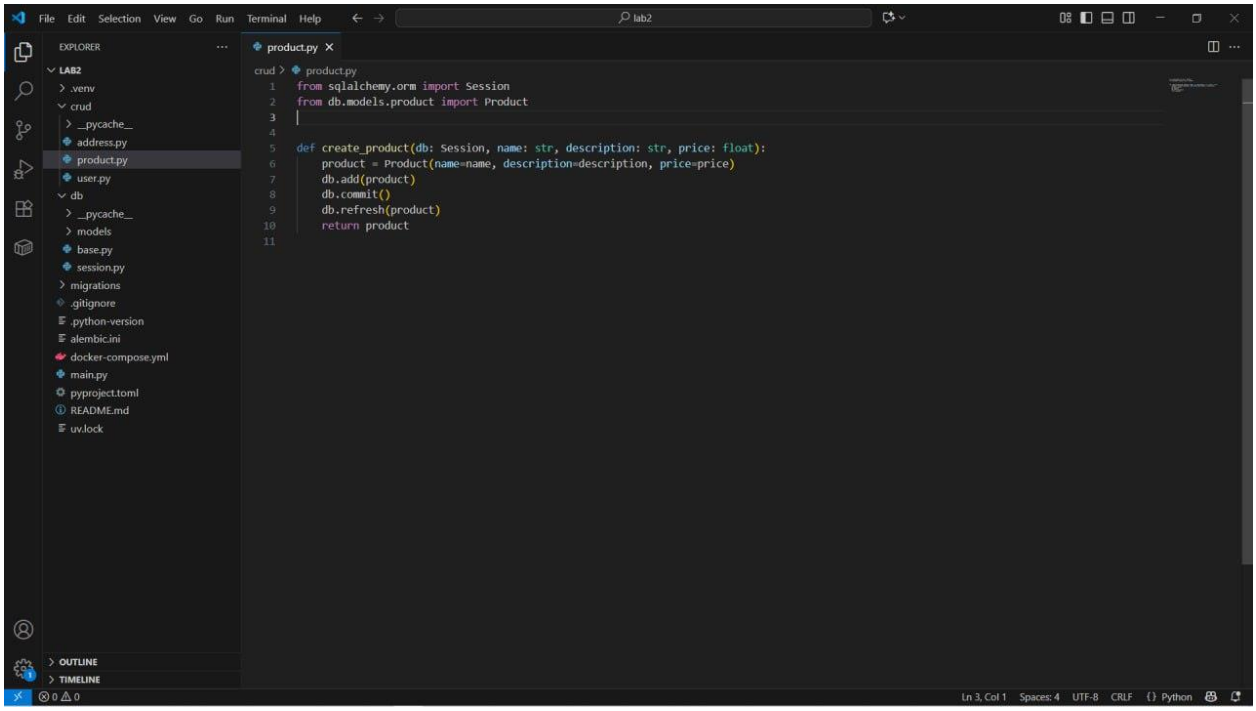
- id (первичный ключ, UUID)
- user_id (внешний ключ на users.id)
- address_id (внешний ключ на addresses.id)
- created_at (дата создания заказа)

Ассоциативная таблица order_products: реализует связь **«многие-ко-многим»** между заказами и продуктами.

Связи между моделями

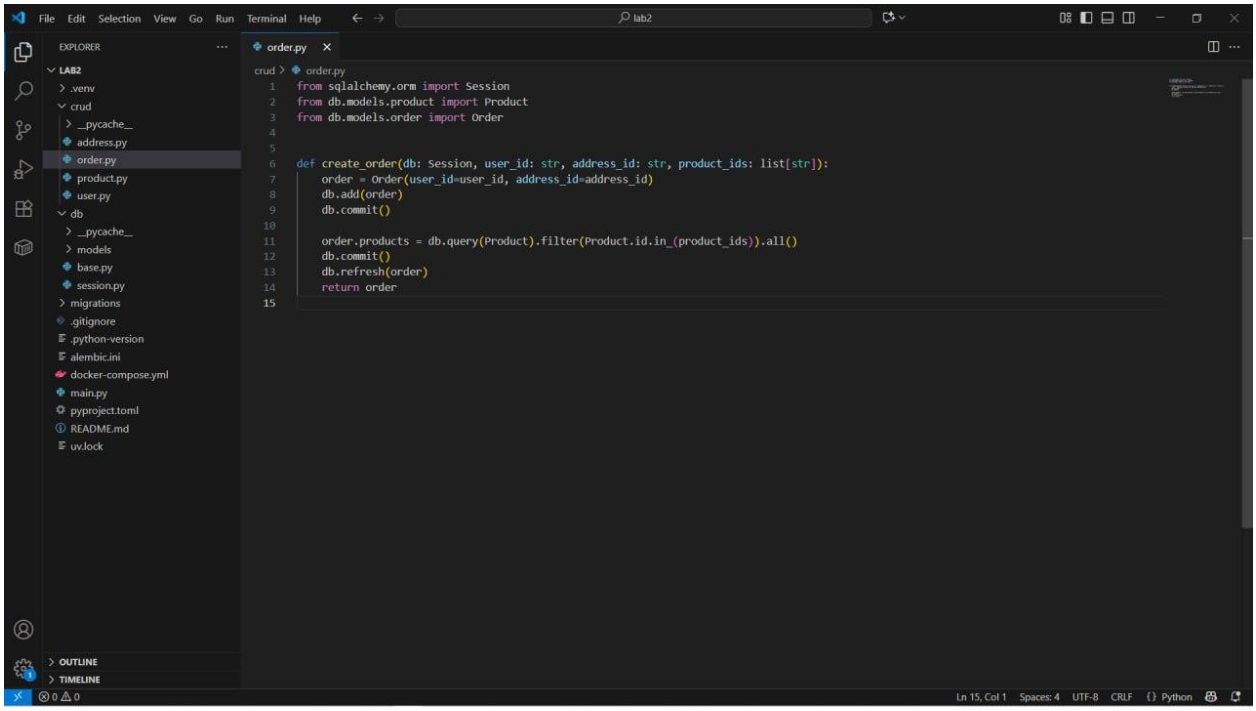
- User → Order: Один-ко-многим (users.orders).
- Order → Product: Многие-ко-многим через order_products.
- Order → Address: Многие-к-одному (один заказ привязан к одному адресу).

Файл product.py



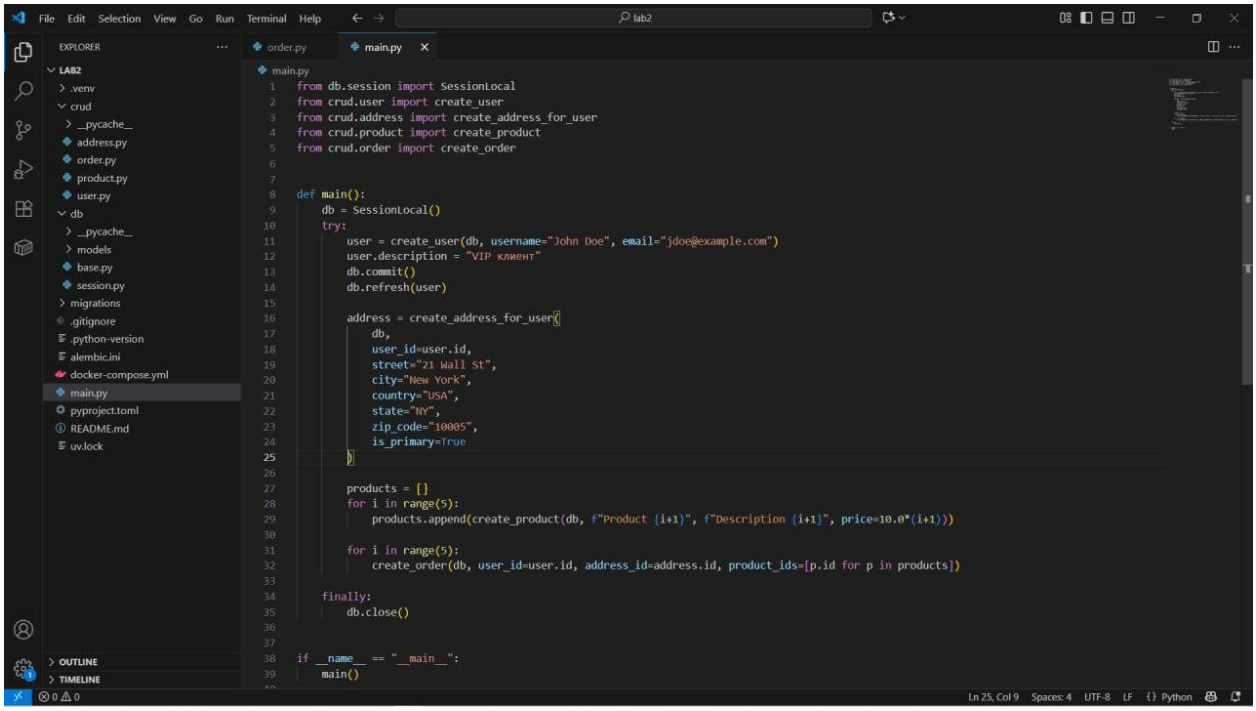
- определена функция create_product для добавления новых продуктов в базу данных.

Файл order.py



- определена функция create_order для создания заказов.

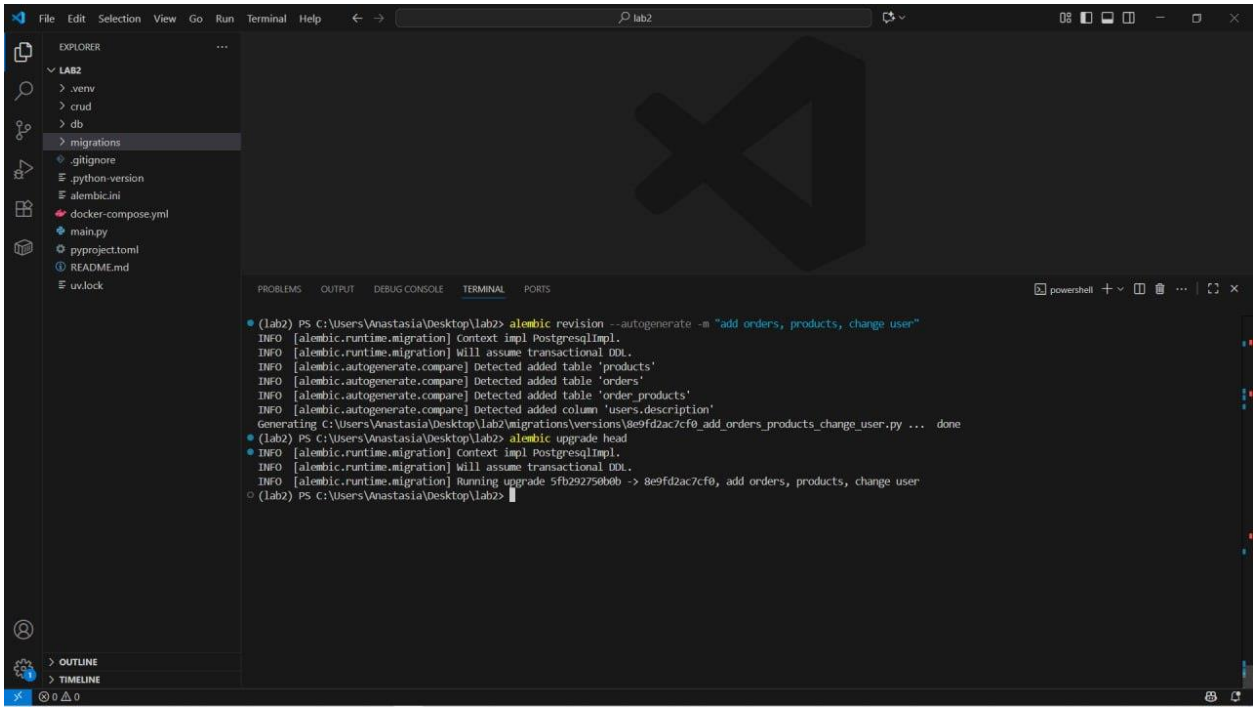
Файл main.py – добавление тестовых данных



- Добавлено 5 записей в таблицу products
- Добавлено 5 заказов в таблицу orders

Каждый заказ привязан к существующему пользователю и адресу.
В ассоциативную таблицу добавлены связи между заказами и продуктами.

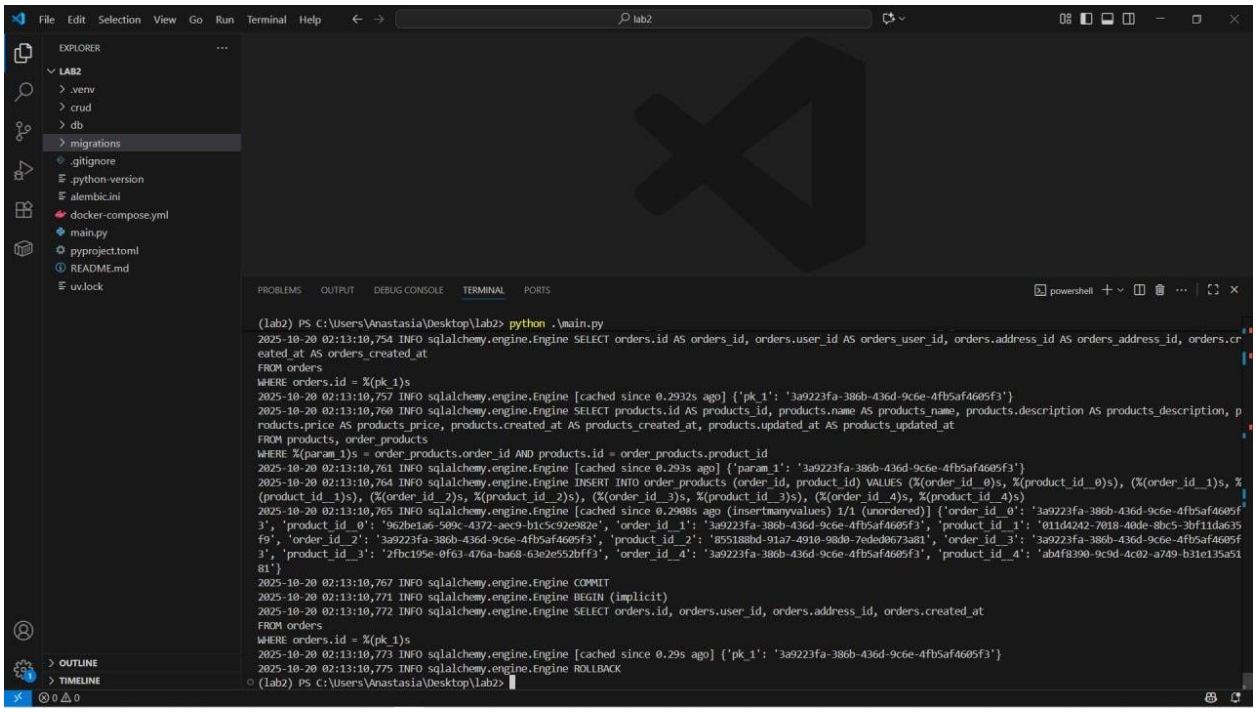
Задача: Проведение миграций

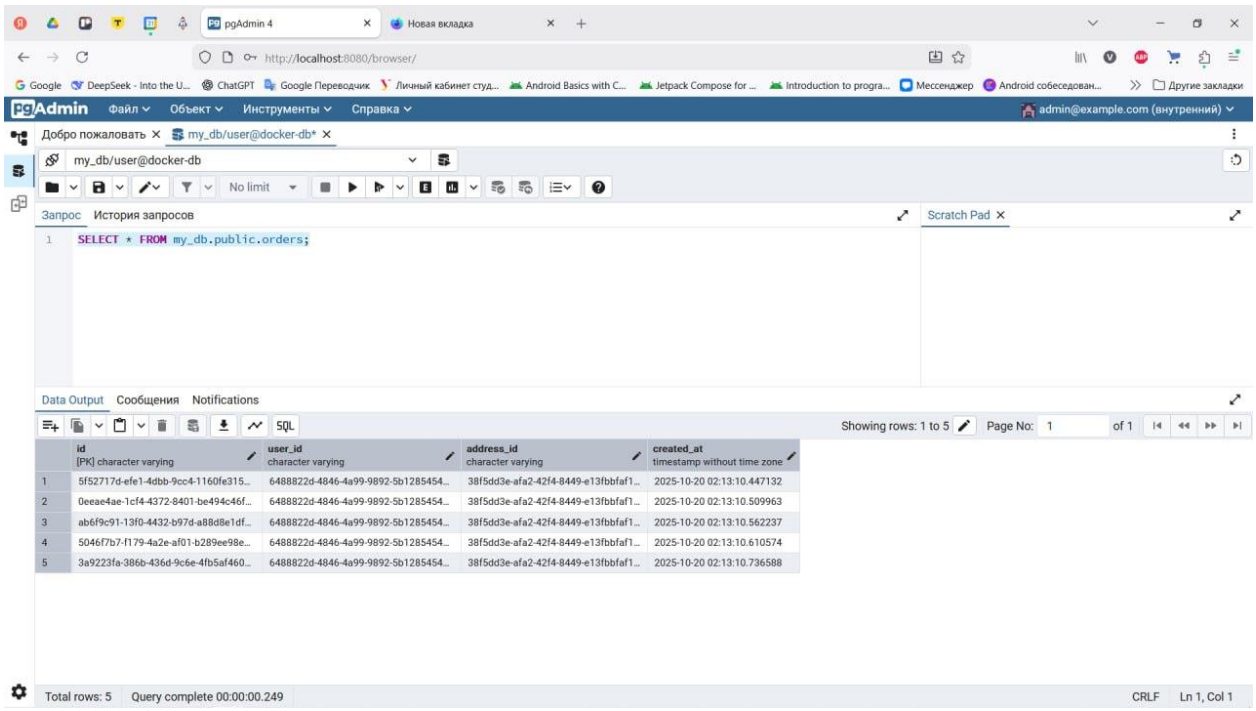


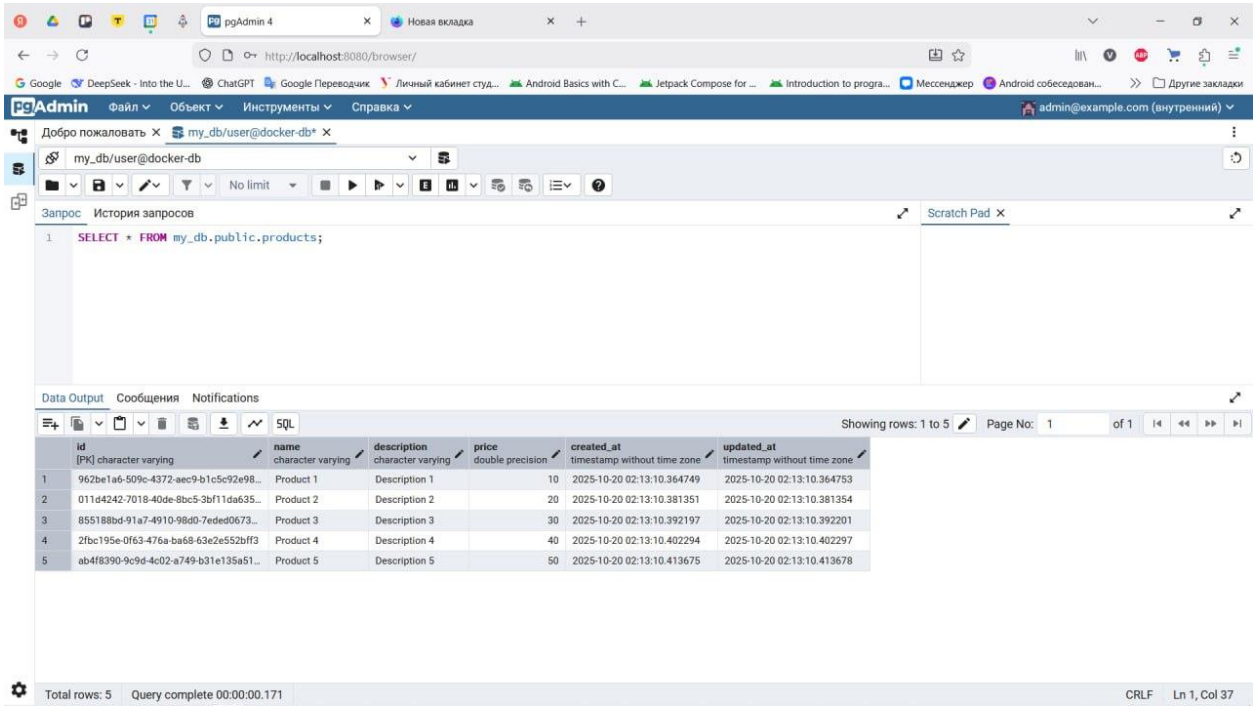
Создание миграций:
alembic revision --autogenerate -m "add orders, products, change user"

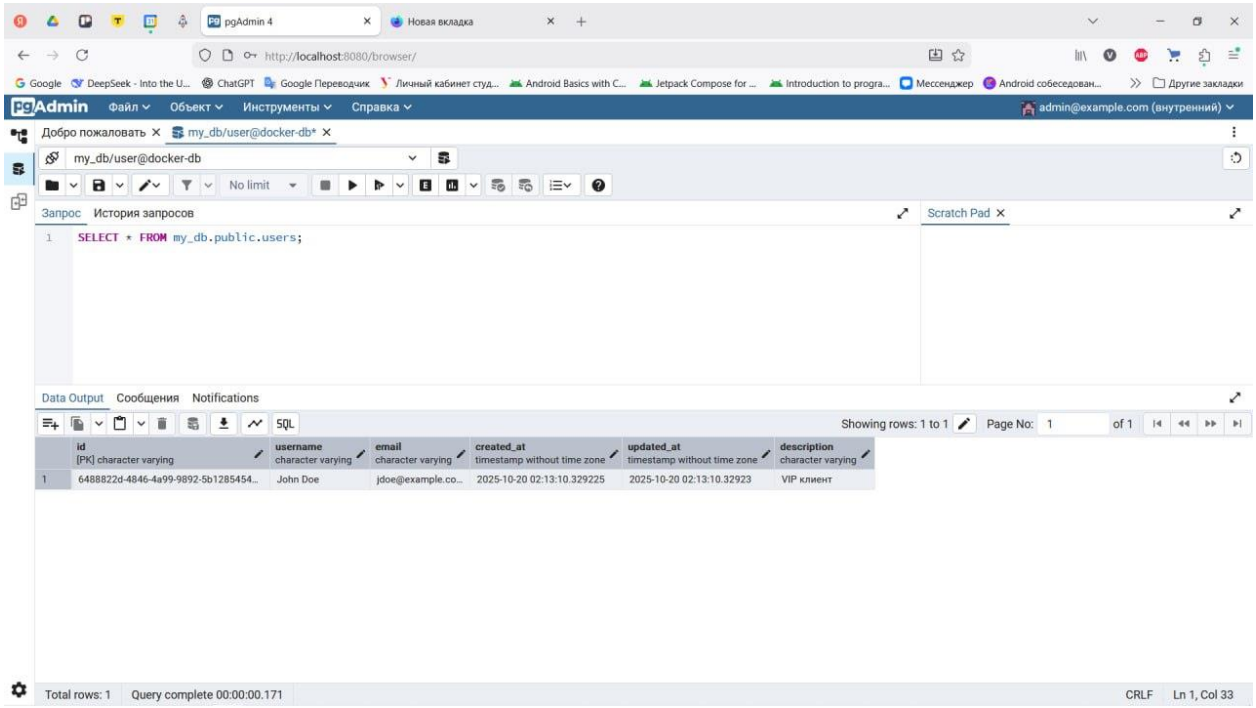
- обнаружено добавление таблиц products, orders и order_products.
- обнаружено добавленное поле users.description

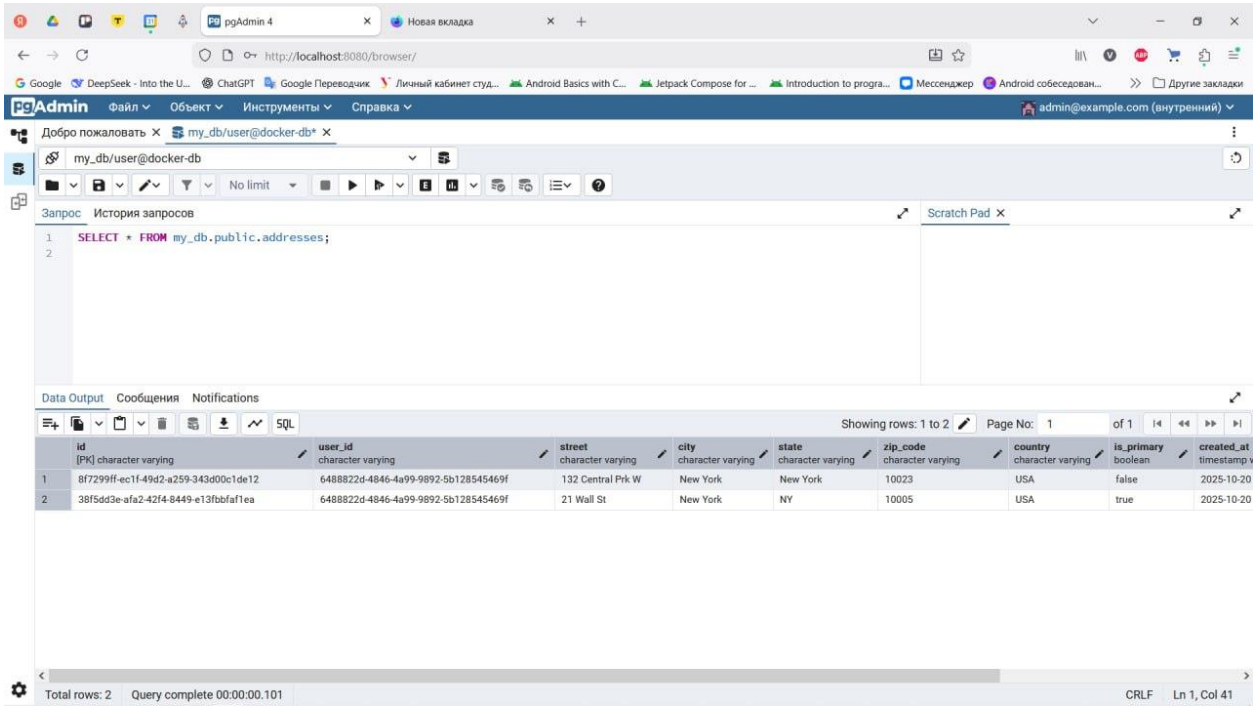
Применение миграции к БД:
alembic upgrade head






pgAdmin 4
Новая вкладка
http://localhost:8080/browser/
PgAdmin
Файл
Объект
Инструменты
Справка
admin@example.com (внутренний)
Добро пожаловать
my_db/user@docker-db*
my_db/user@docker-db
No limit
Запрос
История запросов
Scratch Pad
SELECT * FROM my_db.public.products;
Data Output
Сообщения
Notifications
SQL
Showing rows: 1 to 5
Page No: 1
of 1
id [PK] character varying
name character varying
description character varying
price double precision
created_at timestamp without time zone
updated_at timestamp without time zone
1 962be1a6-509c-4372-aec9-b1c5c92e98... Product 1 Description 1 10 2025-10-20 02:13:10.364749 2025-10-20 02:13:10.364753
2 011d4242-7018-40de-8bc5-3bf11da635... Product 2 Description 2 20 2025-10-20 02:13:10.381351 2025-10-20 02:13:10.381354
3 855188bd-91a7-4910-98d0-7eded0673... Product 3 Description 3 30 2025-10-20 02:13:10.392197 2025-10-20 02:13:10.392201
4 2fbc195e-0f63-476a-ba68-63e2e552bff3 Product 4 Description 4 40 2025-10-20 02:13:10.402294 2025-10-20 02:13:10.402297
5 ab4f8390-9c9d-4c02-a749-b31e135a51... Product 5 Description 5 50 2025-10-20 02:13:10.413675 2025-10-20 02:13:10.413678
Total rows: 5
Query complete 00:00:00.171
CRLF
Ln 1, Col 37


pgAdmin 4
Новая вкладка
http://localhost:8080/browser/
PgAdmin
Файл
Объект
Инструменты
Справка
admin@example.com (внутренний)
Добро пожаловать
my_db/user@docker-db*
my_db/user@docker-db
No limit
Запрос
История запросов
Scratch Pad
SELECT * FROM my_db.public.users;
Data Output
Сообщения
Notifications
SQL
Showing rows: 1 to 1
Page No: 1
of 1
id [PK] character varying
username character varying
email character varying
created_at timestamp without time zone
updated_at timestamp without time zone
description character varying
1 6488822d-4846-4a99-9892-5b1285454... John Doe jdoe@example.co... 2025-10-20 02:13:10.329225 2025-10-20 02:13:10.32923 VIP клиент
Total rows: 1
Query complete 00:00:00.171
CRLF
Ln 1, Col 33



Ответы на вопросы

1. **Какие есть подходы маппинга в SQLAlchemy? Когда следует использовать каждый подход?**

SQLAlchemy предлагает два основных подхода к маппингу:

Декларативный маппинг (использован в лабораторной работе) - современный подход, где модели определяются как классы с аннотациями типов. Позволяет компактно описывать структуру таблиц непосредственно в классах Python с использованием Mapped и mapped_column.

Императивный (классический) маппинг - более традиционный подход, где таблицы и классы определяются отдельно, затем связываются через mapper. Требует больше кода, но дает полный контроль над процессом маппинга.

Когда использовать:

- Декларативный – более современный подход, подходит для новых проектов, когда нужна быстрота разработки (быстро описать модели с минимальным кодом) и читаемость кода (аннотации типов + минимизация boilerplate-кода)
- Императивный - при работе с унаследованными схемами (когда БД уже существует, и ее структура фиксирована; когда нужно маппить существующие сложные таблицы; требуется точное соответствие с существующей схемой) или когда требуется тонкая настройка маппинга, нестандартные типы данных и кастомные преобразования

2. **Как Alembic отслеживает текущую версию базы данных?**

Alembic хранит текущую версию в специальной таблице (по умолчанию alembic_version) в самой базе данных. В таблице лежит version_num — идентификатор последней применённой миграции.

При выполнении команд alembic upgrade или alembic downgrade система:

- Читает текущую версию из таблицы alembic_version
- Определяет какие миграции нужно применить для достижения целевого состояния
- Последовательно выполняет миграционные скрипты
- Обновляет запись в таблице alembic_version

3. **Какие типы связей между таблицами вы реализовали в данной работе?**

В работе реализованы 3 типа реляционных связей:

- Один-ко-многим (One-to-Many) - связь между пользователем и заказами, где один пользователь может иметь несколько заказов, но каждый заказ принадлежит только одному пользователю.

- Многие-к-одному (Many-to-One) - обратная сторона предыдущей связи, а также связь между заказом и адресом доставки, где несколько заказов могут быть привязаны к одному адресу.
- Многие-ко-многим (Many-to-Many) - реализована через ассоциативную таблицу order_products для связи заказов и продуктов. Один заказ может содержать несколько продуктов, и один продукт может присутствовать в нескольких заказах.

4. **Что такое миграция базы данных и почему она важна?**

Миграция БД — это управляемый процесс изменения схемы базы данных, который включает модификацию таблиц, столбцов, индексов и связей между таблицами.

Важность миграций обусловлена:

- Безопасностью изменений - миграции позволяют применять изменения к структуре БД без потери данных и с возможностью отката
- Версионным контролем - каждая миграция имеет уникальный идентификатор, что позволяет точно отслеживать историю изменений схемы БД
- Совместной разработкой - команды разработчиков могут работать независимо, создавая миграции, которые затем объединяются
- Воспроизводимостью - одинаковое состояние БД может быть достигнуто на разных окружениях (разработка, тестирование, продакшен)

5. **Как обрабатываются отношения многие-ко-многим в SQLAlchemy?**

В SQLAlchemy отношения многие-ко-многим реализуются через ассоциативные таблицы:

- Создание ассоциативной таблицы - промежуточной таблицы, содержащей только

внешние ключи на связываемые таблицы.
- Настройка relationship - в основной модели указывается связь через параметр secondary

6. Каков порядок действий при возникновении конфликта версий в Alembic?

1) **Анализ ситуации** - определить какие именно миграции конфликтуют и в чем заключается конфликт (изменение одних и тех же таблиц, столбцов и т.д.)
2) **Создание merge-миграции** - выполнить команду alembic merge для создания миграции, объединяющей изменения из конфликтующих версий
3) **Ручное разрешение конфликтов** - если автоматическое объединение невозможно, отредактировать миграционные скрипты вручную, устранив противоречия
4) **Тестирование** - проверить корректность merge-миграции на тестовой базе данных перед применением к основной
5) **Коммуникация в команде** - уведомить других разработчиков о разрешении конфликта и предоставить обновленные миграции
6) **Профилактика** - установить правила работы в команде, минимизирующие вероятность конфликтов (например, согласование изменений схемы БД перед созданием миграций)

Вывод:

В ходе выполнения лабораторной работы были освоены практические навыки работы с SQLAlchemy ORM и системой миграций Alembic**,** в том числе:

- создание и модификация ORM-моделей (описание структуры БД на уровне Python-классов, добавление новых полей (как поле description для пользователя) и настройка связей между таблицами.
- работа со связями через ассоциативные таблицы (реализована связь один-ко-многим (пользователь-заказы) и многие-ко-многим (заказы-продукты)
- работа с системой миграций (создание миграционных скриптов, отслеживание изменений в моделях и применение их к БД)

Работа показала, что использование ORM и системы миграций значительно упрощает разработку и сопровождение приложений с базами данных. Миграции позволяют безопасно изменять структуру БД при активной разработке, а SQLAlchemy избавляет от необходимости писать сложные SQL-запросы вручную.