

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

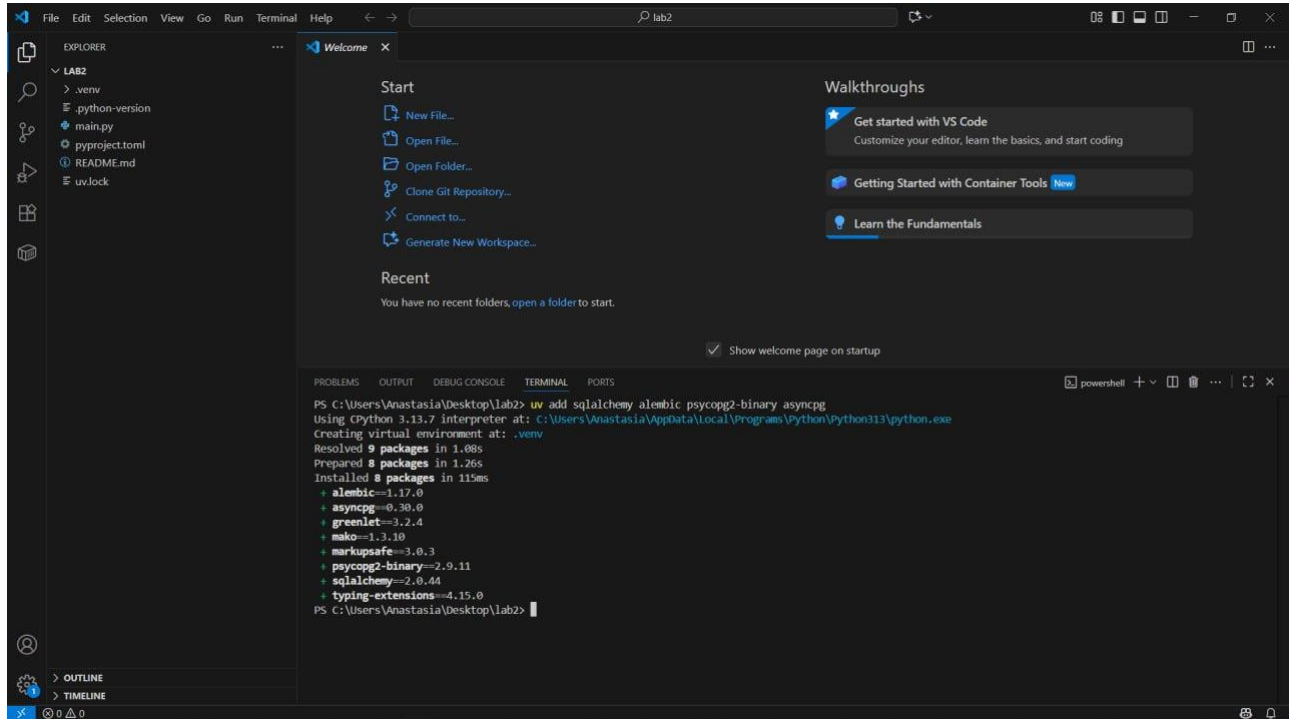
Структура проекта

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

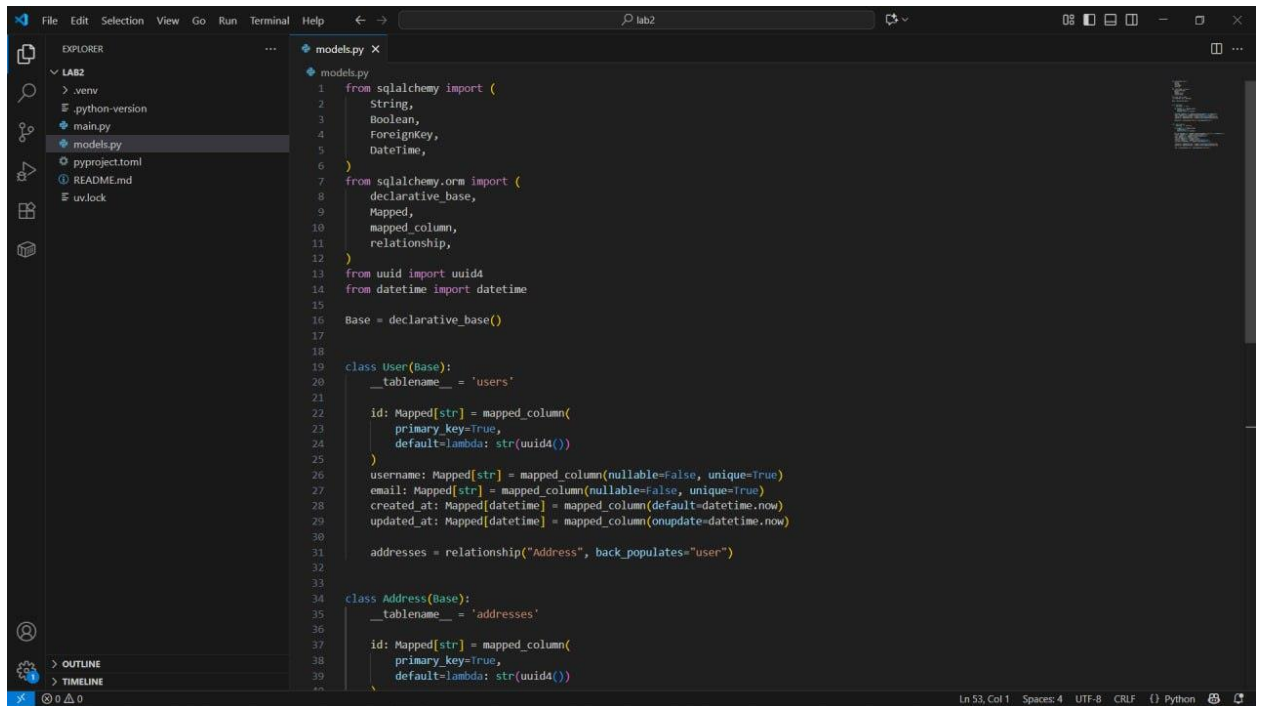


Создаем новый проект с помощью команды **uv init**

В терминале запускаем команду установки пакетов

- **uv add sqlalchemy alembic psycopg2-binary asyncpg**

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



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
```

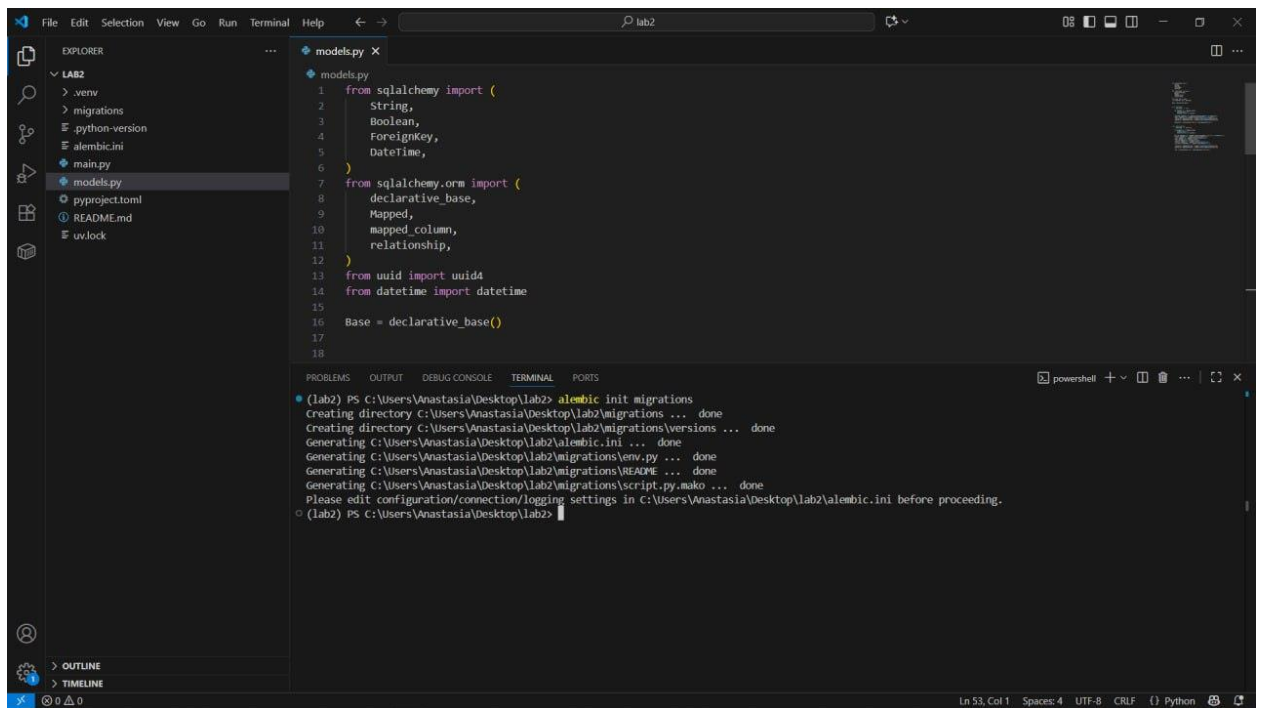
Импорты из SQLAlchemy ORM

- `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)

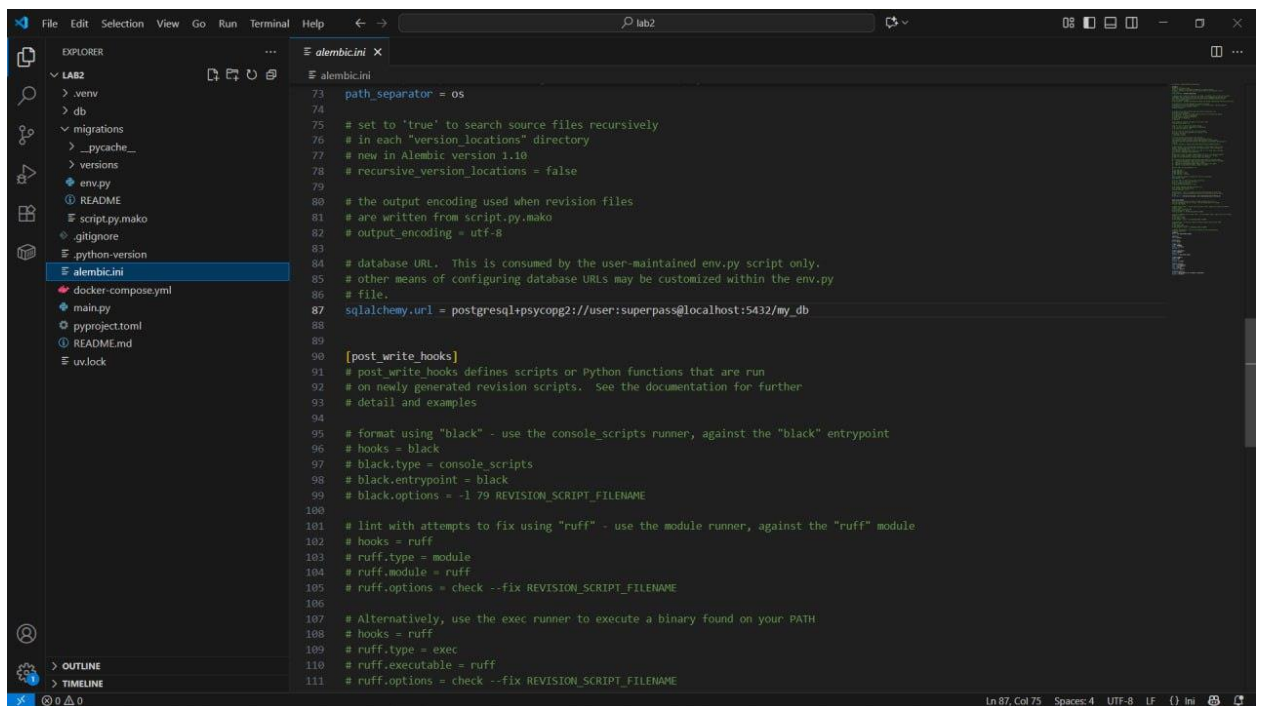
Объявили 2 класса: `User` и `Address` (определяем базовый класс с помощью `declarative_base()`)



Инициализация миграций Alembic

- `alembic init migrations`.

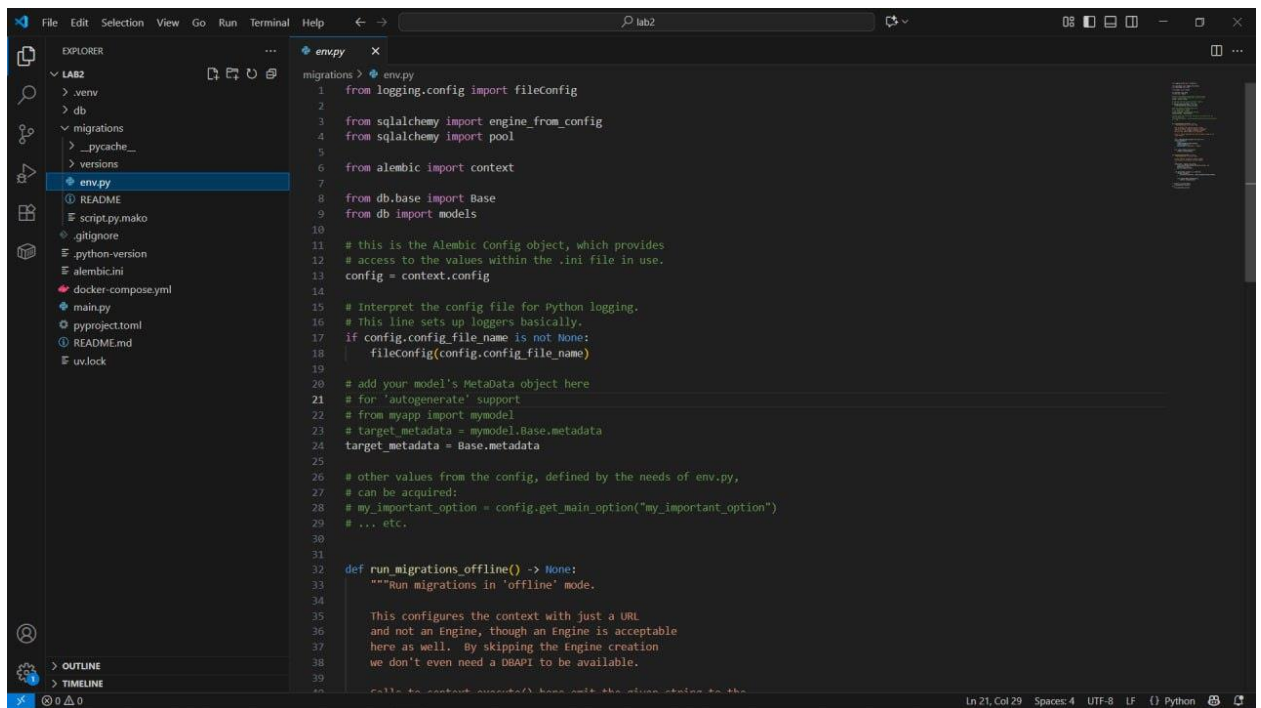
Создана папка `migrations` с необходимыми файлами.



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`

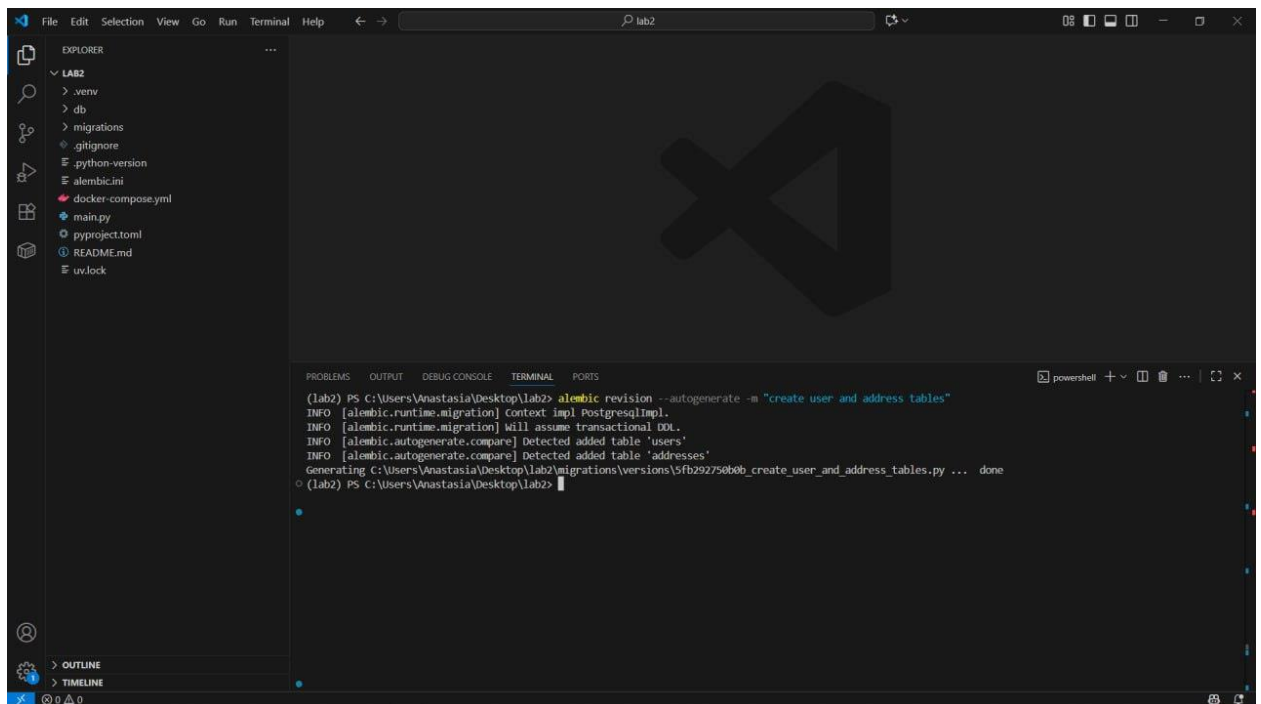


Импорты:

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

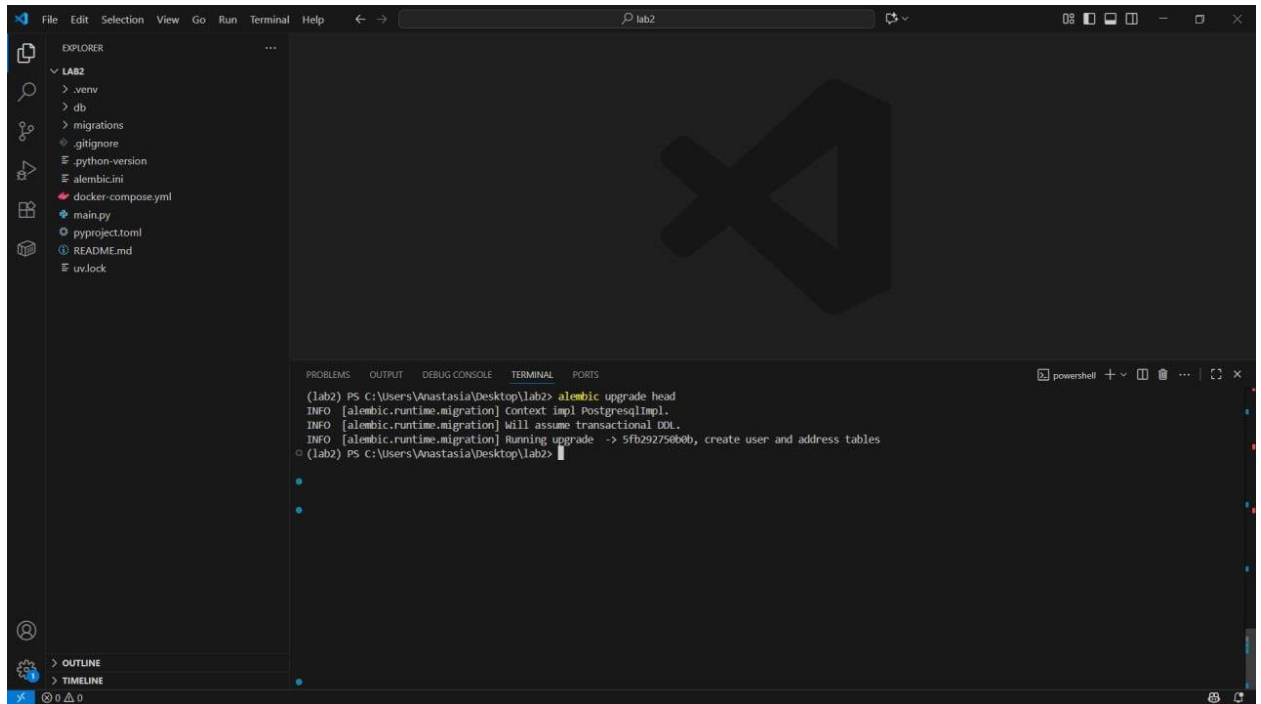
Установлены метаданные для миграций:

- `target_metadata = Base.metadata`



Создана первая миграция

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

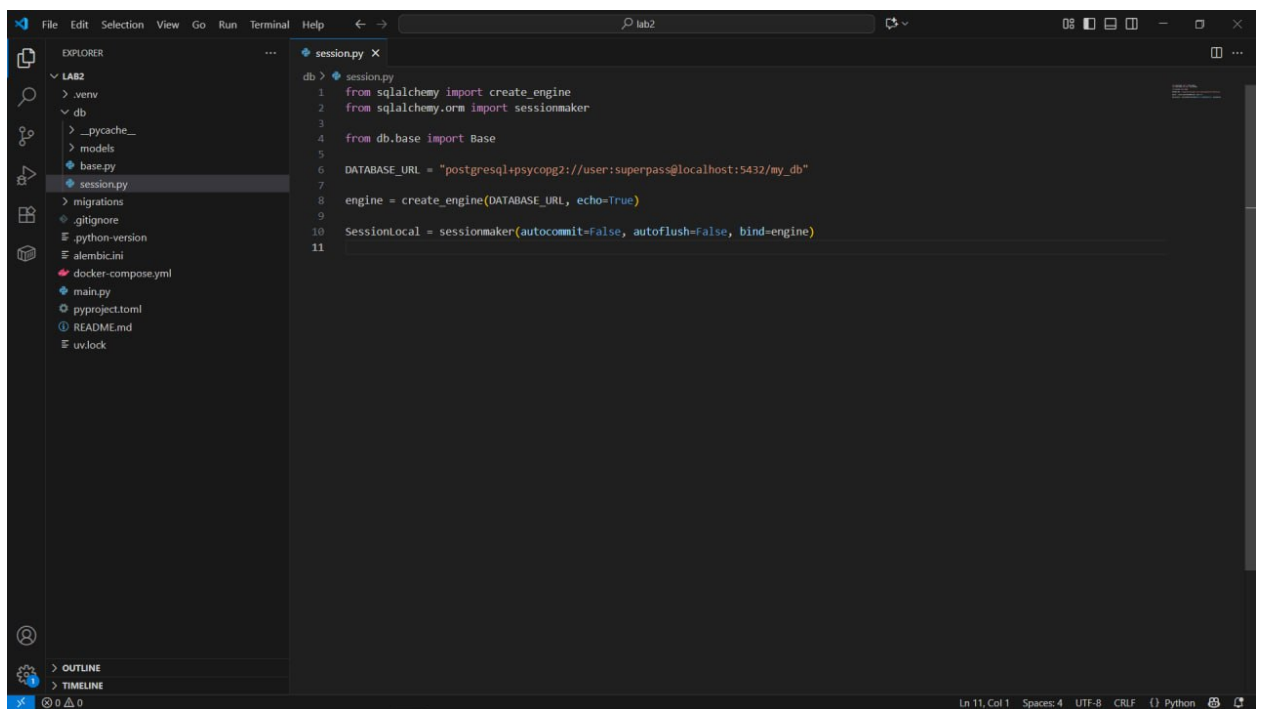


Миграция применена к БД:

- alembic upgrade head

Задача: наполнить БД данными

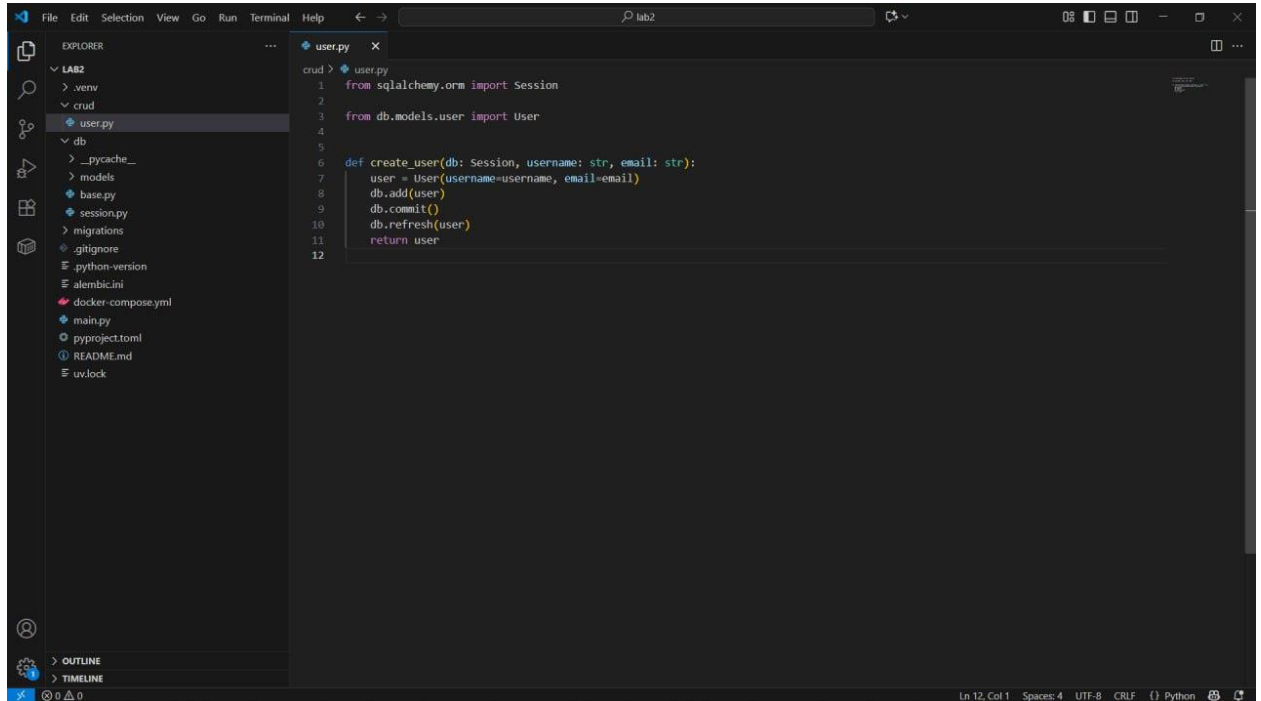
1) Настройка подключения к базе данных (db/ session.py)



- 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

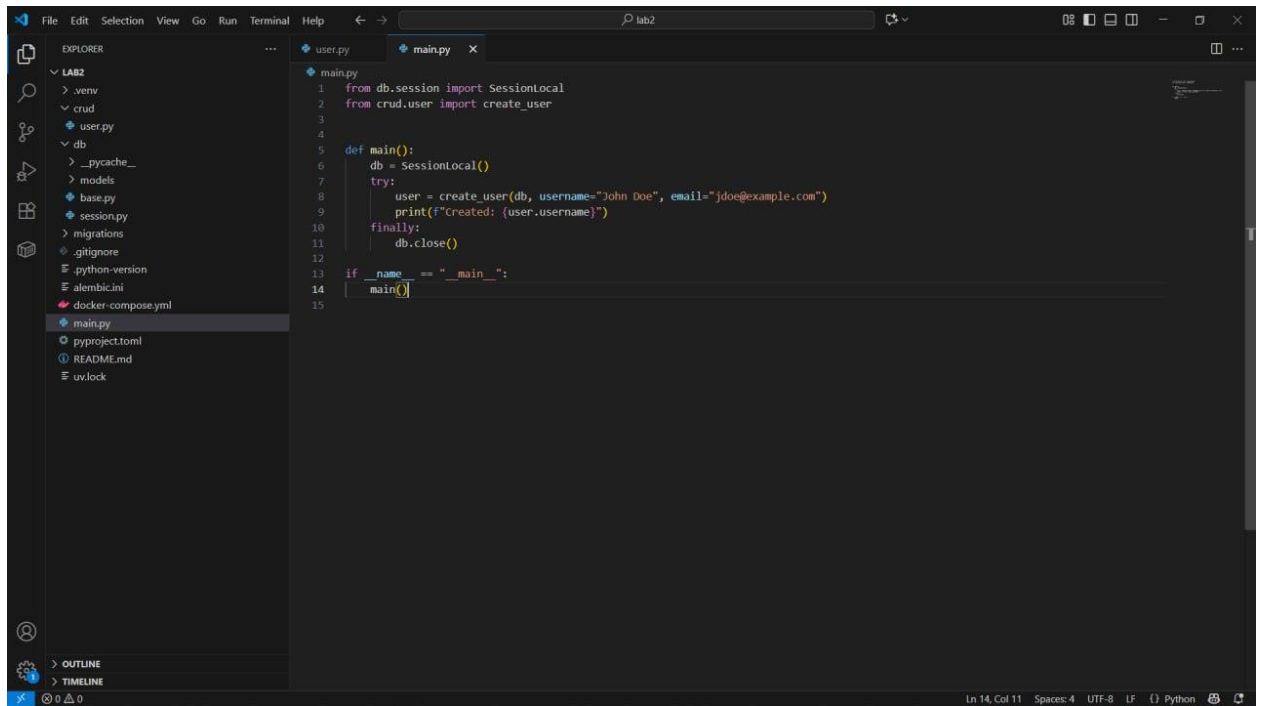
2) Функция создания пользователя (`crud/user.py`)



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;

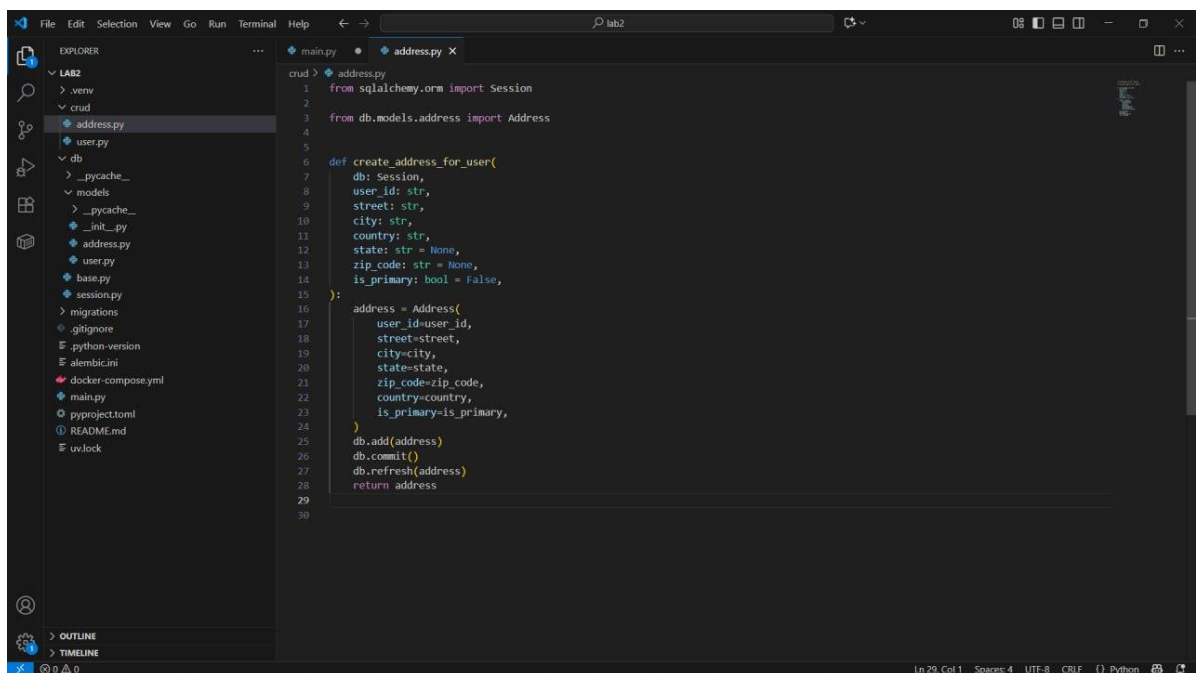
3) Файл `main.py`



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

```

10 def main():
11     db = SessionLocal()
12     try:
13         user = create_user(db, username="John Doe", email="jdoe@example.com")
14         print(f"Created: {user.username}")
15
16         address = create_address_for_user(
17             db,
18             user_id=user.id,
19             street="21 Wall St",
20             city="New York",
21             state="New York",
22             zip_code="10005",
23             country="USA",
24             is_primary=True,
25         )
26         print(f"Created address for {user.username}: {address.street}, {address.city}")
27
28         address = create_address_for_user(
29             db,
30             user_id=user.id,
31             street="132 Central Prk W",
32             city="New York",
33             state="New York",
34             zip_code="10023",
35             country="USA",
36             is_primary=False,
37         )
38         print(f"Created address for {user.username}: {address.street}, {address.city}")
39
40     stmt = select(User).options(selectinload(User.addresses))
41     users = db.scalars(stmt).all()
42
43     for user in users:
44         print(f"{user.username} ({user.email})")
45         for address in user.addresses:
46             print(f"{address.street}, {address.city}, {address.country}")
47
48 finally:

```

- 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.

```
def main():
    try:
        print(f"Created address for (user.username): {address.street}, {address.city}")

        stmt = select(User).options(selectinload(User.addresses))
        users = db.scalars(stmt).all()

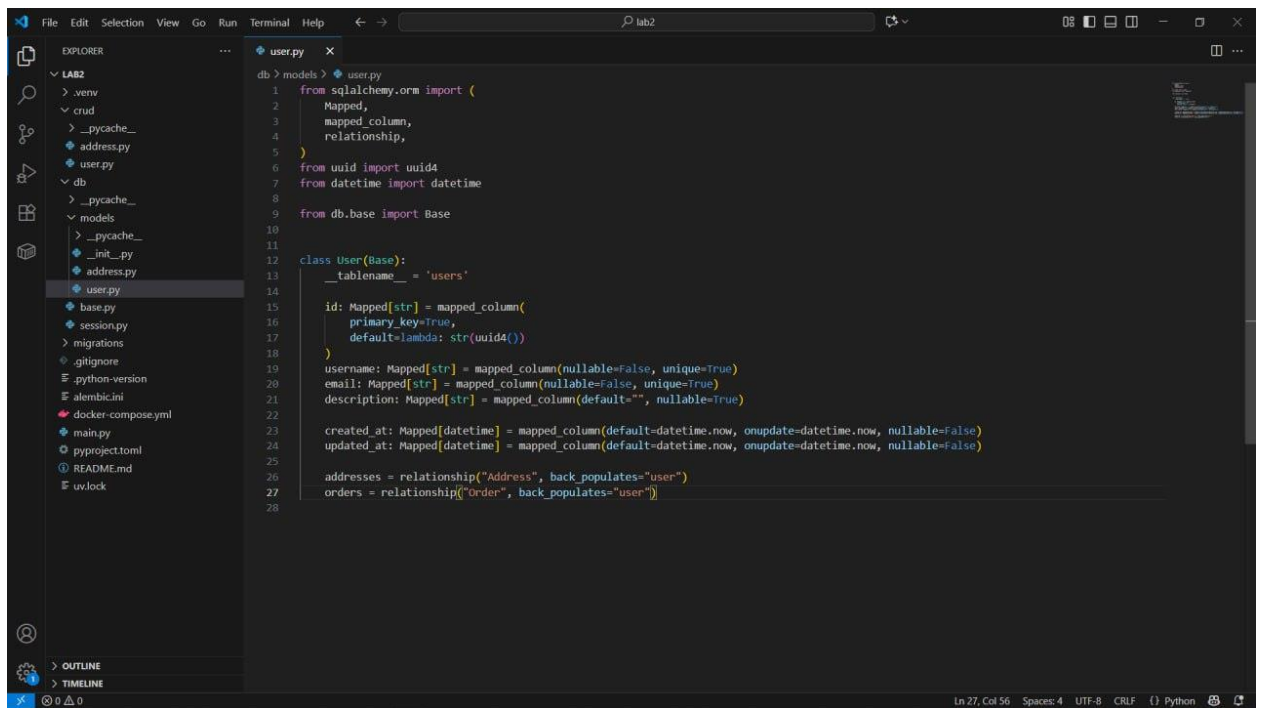
        for user in users:
            print(f"(user.username) {(user.email)}")
            for address in user.addresses:
                print(f"{address.street}, {address.city}, {address.country}")

    finally:
        # (lab2) PS C:\Users\Anastasia\Desktop\lab2> python .\main.py
        2025-10-20 01:36:06,913 INFO sqlalchemy.engine.Engine select pg_catalog.version()
        2025-10-20 01:36:06,913 INFO sqlalchemy.engine.Engine [raw sql] {}
        2025-10-20 01:36:06,919 INFO sqlalchemy.engine.Engine select current_schema()
        2025-10-20 01:36:06,919 INFO sqlalchemy.engine.Engine [raw sql] {}
        2025-10-20 01:36:06,924 INFO sqlalchemy.engine.Engine show standard_conforming_strings
        2025-10-20 01:36:06,925 INFO sqlalchemy.engine.Engine [raw sql] {}
        2025-10-20 01:36:06,930 INFO sqlalchemy.engine.Engine BEGIN (implicit)
        2025-10-20 01:36:06,950 INFO sqlalchemy.engine.Engine SELECT users.id, users.username, users.email, users.created_at, users.updated_at
        FROM users
        WHERE users.email = %(email_1)s
        Created: John Doe
        2025-10-20 01:36:06,960 INFO sqlalchemy.engine.Engine INSERT INTO addresses (id, user_id, street, city, state, zip_code, country, is_primary, created_at, update
        d_at) VALUES (%(id)s, %(user_id)s, %(street)s, %(city)s, %(state)s, %(zip_code)s, %(country)s, %(is_primary)s, %(created_at)s, %(updated_at)s)
        2025-10-20 01:36:06,960 INFO sqlalchemy.engine.Engine [generated in 0.000996s] {'id': '9677592d-e206-45dc-98cb-3a438576f965', 'user_id': '6488822d-4846-4a99-9892
        -5b128545469f', 'street': '21 Wall St', 'city': 'New York', 'state': 'New York', 'zip_code': '10005', 'country': 'USA', 'is_primary': True, 'created_at': dateti
        me.datetime(2025, 10, 20, 1, 36, 6, 959940), 'updated_at': datetime.datetime(2025, 10, 20, 1, 36, 6, 959940)}
        2025-10-20 01:36:06,973 INFO sqlalchemy.engine.Engine COMMIT
        2025-10-20 01:36:06,980 INFO sqlalchemy.engine.Engine BEGIN (implicit)
        2025-10-20 01:36:06,986 INFO sqlalchemy.engine.Engine SELECT addresses.id, addresses.user_id, addresses.street, addresses.city, addresses.state, addresses.zip_c
        ode, addresses.country, addresses.is_primary, addresses.created_at, addresses.updated_at
        FROM addresses
        WHERE addresses.id = %(pk_1)s
        2025-10-20 01:36:06,986 INFO sqlalchemy.engine.Engine [generated in 0.001045s] {'pk_1': '9677592d-e206-45dc-98cb-3a438576f965'}
```

```
FROM users
WHERE users.id = %(pk_1)s
2025-10-20 01:36:07,015 INFO sqlalchemy.engine.Engine [cached since 0.022055s ago] {'pk_1': '6488822d-4846-4a99-9892-5b128545469f'}
Created address for John Doe: 132 Central Prk W, New York
2025-10-20 01:36:07,021 INFO sqlalchemy.engine.Engine SELECT users.id, users.username, users.email, users.created_at, users.updated_at
2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.023335s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'}
2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS
users_created_at, users.updated_at AS users_updated_at
FROM users
WHERE users.id = %(pk_1)s
2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.023335s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'}
2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS
users_created_at, users.updated_at AS users_updated_at
FROM users
WHERE users.id = %(pk_1)s
2025-10-20 01:36:07,015 INFO sqlalchemy.engine.Engine [cached since 0.022055s ago] {'pk_1': '6488822d-4846-4a99-9892-5b128545469f'}
Created address for John Doe: 132 Central Prk W, New York
2025-10-20 01:36:07,021 INFO sqlalchemy.engine.Engine SELECT users.id, users.username, users.email, users.created_at, users.updated_at
FROM users
2025-10-20 01:36:07,022 INFO sqlalchemy.engine.Engine [generated in 0.000711s] {}
2025-10-20 01:36:07,027 INFO sqlalchemy.engine.Engine SELECT addresses.user_id AS addresses_user_id, addresses.id AS addresses_id, addresses.street AS addresses_street,
addresses.city AS addresses_city, addresses.state AS addresses_state, addresses.zip_code AS addresses_zip_code, addresses.country AS addresses_country, addresses.is_pr
imary AS addresses_is_primary, addresses.created_at AS addresses_created_at, addresses.updated_at AS addresses_updated_at
FROM addresses
WHERE addresses.user_id IN (%(primary_keys_1)s)
2025-10-20 01:36:07,029 INFO sqlalchemy.engine.Engine [generated in 0.001656s] {'primary_keys_1': '6488822d-4846-4a99-9892-5b128545469f'}
John Doe (jdoe@example.com)
21 Wall St, New York, USA
132 Central Prk W, New York, USA
2025-10-20 01:36:07,035 INFO sqlalchemy.engine.Engine ROLLBACK
```

Часть 3: Последующие работы с БД и миграции

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

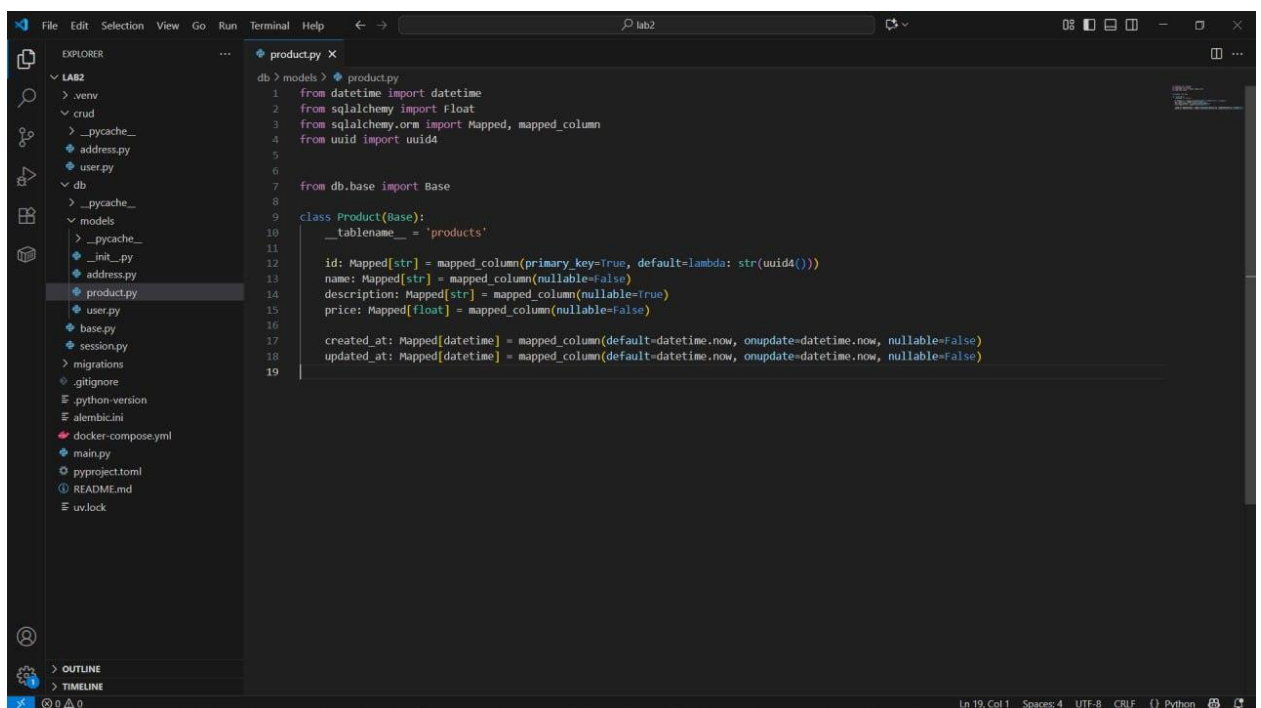


```
db > models > user.py
1 from sqlalchemy.orm import (
2     Mapped,
3     mapped_column,
4     relationship,
5 )
6 from uuid import uuid4
7 from datetime import datetime
8
9 from db.base import Base
10
11
12 class User(Base):
13     __tablename__ = 'users'
14
15     id: Mapped[str] = mapped_column(
16         primary_key=True,
17         default=lambda: str(uuid4())
18     )
19     username: Mapped[str] = mapped_column(nullable=False, unique=True)
20     email: Mapped[str] = mapped_column(nullable=False, unique=True)
21     description: Mapped[str] = mapped_column(default="", nullable=True)
22
23     created_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
24     updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
25
26     addresses = relationship("Address", back_populates="user")
27     orders = relationship("Order", back_populates="user")
28
```

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)

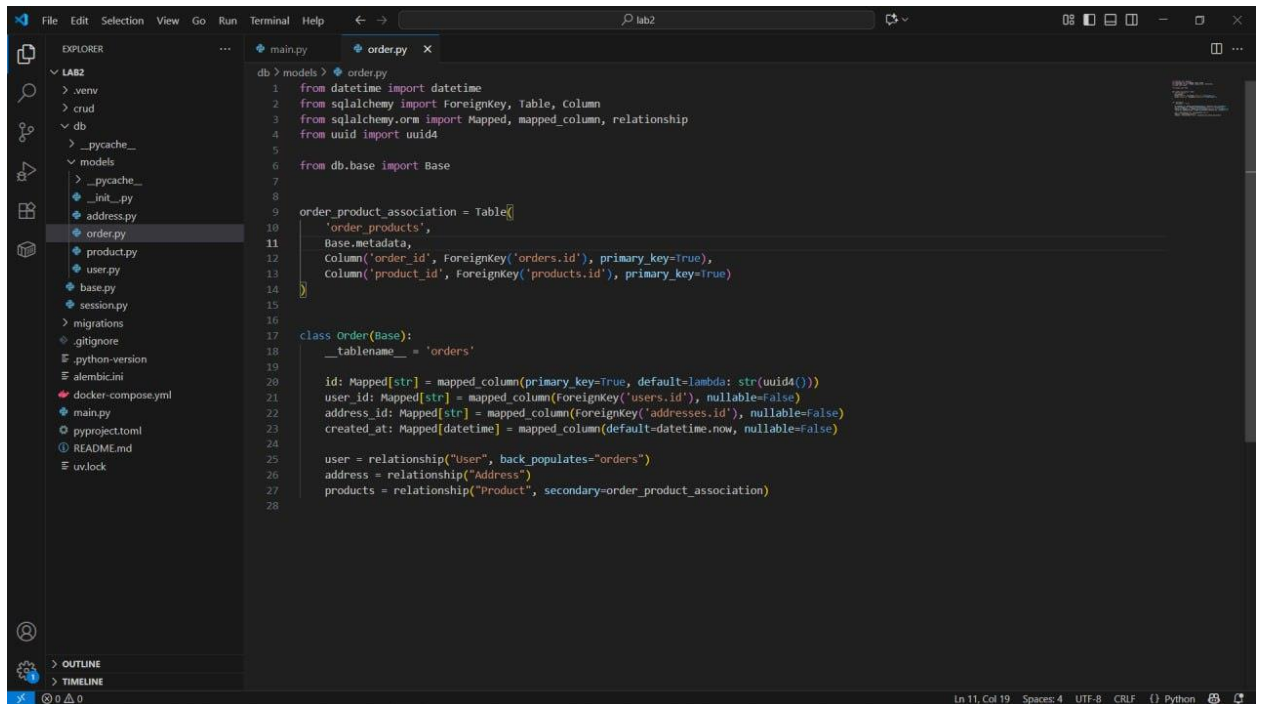


```
db > models > product.py
1 from datetime import datetime
2 from sqlalchemy import Float
3 from sqlalchemy.orm import Mapped, mapped_column
4 from uuid import uuid4
5
6
7 from db.base import Base
8
9 class Product(Base):
10     __tablename__ = 'products'
11
12     id: Mapped[str] = mapped_column(primary_key=True, default=lambda: str(uuid4()))
13     name: Mapped[str] = mapped_column(nullable=False)
14     description: Mapped[str] = mapped_column(nullable=True)
15     price: Mapped[float] = mapped_column(nullable=False)
16
17     created_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
18     updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
19
```

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)



```

1 from datetime import datetime
2 from sqlalchemy import ForeignKey, Table, Column
3 from sqlalchemy.orm import Mapped, mapped_column, relationship
4 from uuid import uuid4
5
6 from db.base import Base
7
8
9 order_product_association = Table(
10     'order_products',
11     Base.metadata,
12     Column('order_id', ForeignKey('orders.id'), primary_key=True),
13     Column('product_id', ForeignKey('products.id'), primary_key=True)
14 )
15
16
17 class Order(Base):
18     __tablename__ = 'orders'
19
20     id: Mapped[str] = mapped_column(primary_key=True, default=lambda: str(uuid4()))
21     user_id: Mapped[str] = mapped_column(ForeignKey('users.id'), nullable=False)
22     address_id: Mapped[str] = mapped_column(ForeignKey('addresses.id'), nullable=False)
23     created_at: Mapped[datetime] = mapped_column(default=datetime.now, nullable=False)
24
25     user = relationship("User", back_populates="orders")
26     address = relationship("Address")
27     products = relationship("Product", secondary=order_product_association)
28
  
```

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

Поля модели Order:

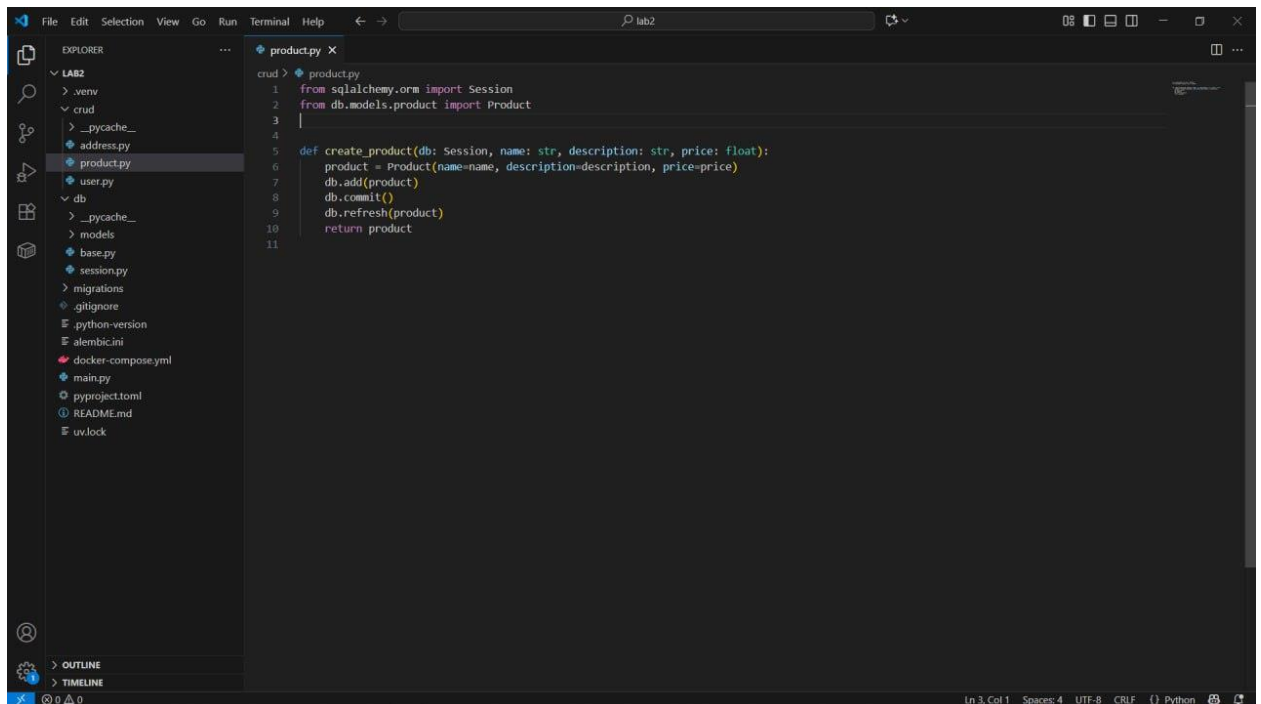
- id (первичный ключ, UUID)
- user_id (внешний ключ на users.id)
- address_id (внешний ключ на addresses.id)
- created_at (дата создания заказа)

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

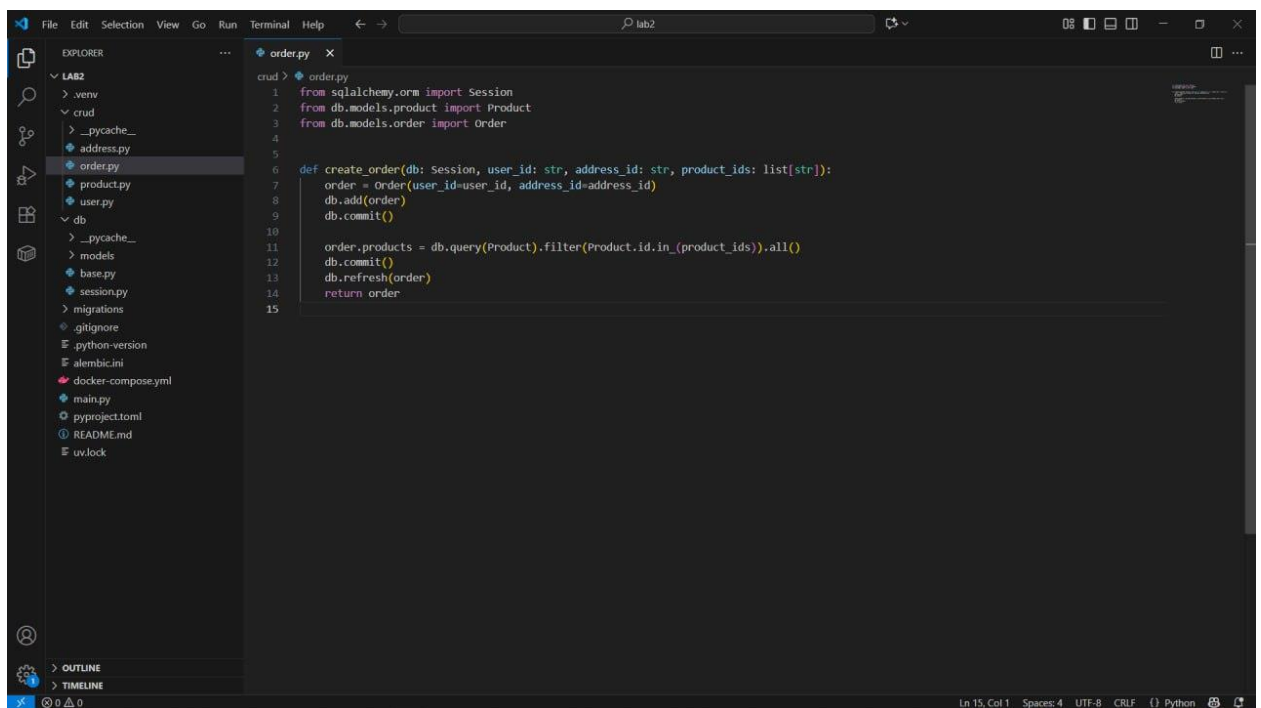


The screenshot shows a VS Code editor window with the file explorer on the left and the editor area on the right. The file explorer shows a project structure with a 'crud' directory containing 'product.py' and 'user.py'. The editor area shows the code for 'product.py'.

```
crud > product.py
1 from sqlalchemy.orm import Session
2 from db.models.product import Product
3
4
5 def create_product(db: Session, name: str, description: str, price: float):
6     product = Product(name=name, description=description, price=price)
7     db.add(product)
8     db.commit()
9     db.refresh(product)
10    return product
11
```

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

Файл order.py

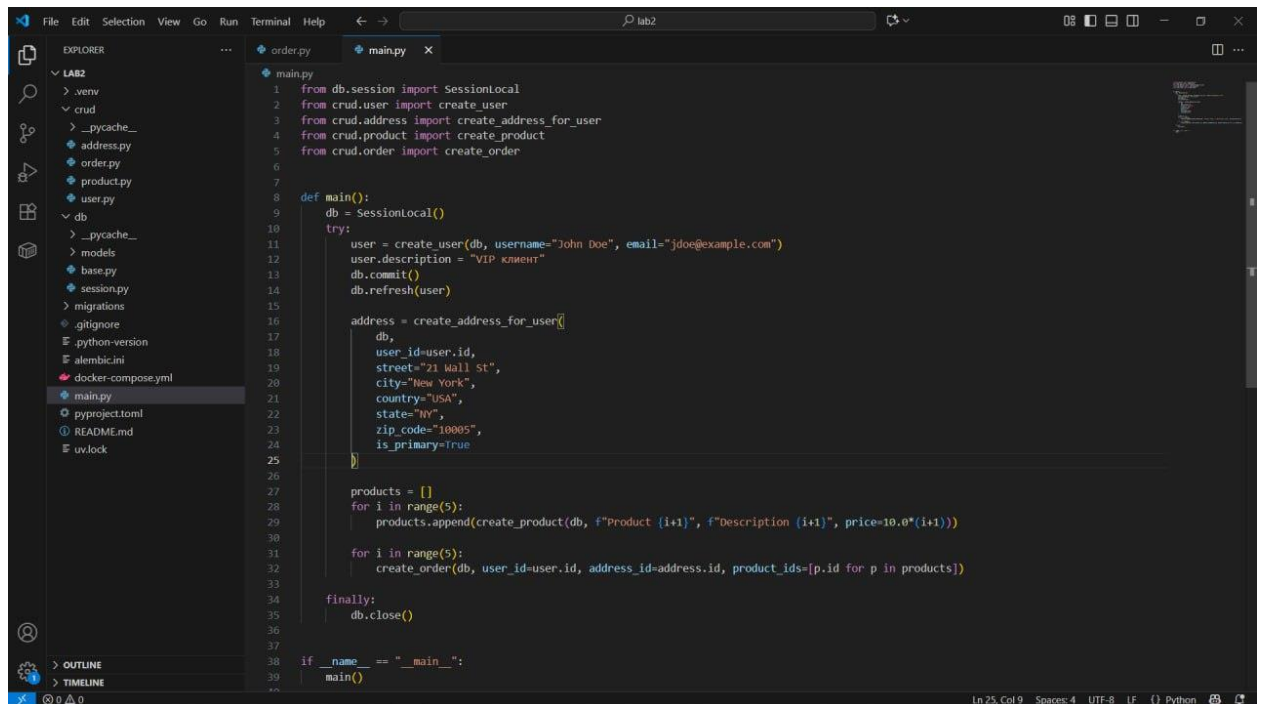


The screenshot shows a VS Code editor window with the file explorer on the left and the editor area on the right. The file explorer shows a project structure with a 'crud' directory containing 'product.py' and 'order.py'. The editor area shows the code for 'order.py'.

```
crud > order.py
1 from sqlalchemy.orm import Session
2 from db.models.product import Product
3 from db.models.order import Order
4
5
6 def create_order(db: Session, user_id: str, address_id: str, product_ids: list[str]):
7     order = Order(user_id=user_id, address_id=address_id)
8     db.add(order)
9     db.commit()
10
11     order.products = db.query(Product).filter(Product.id.in_(product_ids)).all()
12     db.commit()
13     db.refresh(order)
14     return order
15
```

- определена функция `create_order` для создания заказов.

Файл main.py – добавление тестовых данных



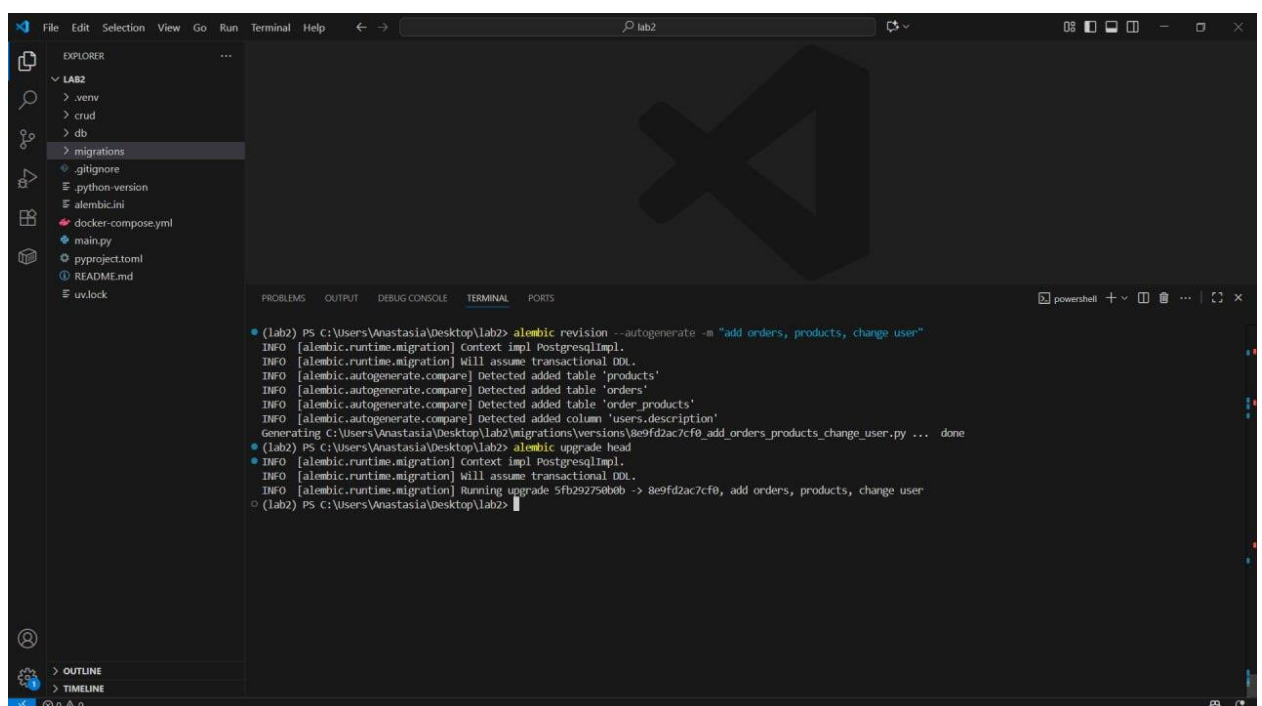
```
1 from db.session import SessionLocal
2 from crud.user import create_user
3 from crud.address import create_address_for_user
4 from crud.product import create_product
5 from crud.order import create_order
6
7
8 def main():
9     db = SessionLocal()
10    try:
11        user = create_user(db, username="John Doe", email="jdoe@example.com")
12        user.description = "VIP клиент"
13        db.commit()
14        db.refresh(user)
15
16        address = create_address_for_user(
17            db,
18            user_id=user.id,
19            street="21 Wall St",
20            city="New York",
21            country="USA",
22            state="NY",
23            zip_code="10005",
24            is_primary=True
25        )
26
27        products = []
28        for i in range(5):
29            products.append(create_product(db, f"Product {i+1}", f"Description {i+1}", price=10.0*(i+1)))
30
31        for i in range(5):
32            create_order(db, user_id=user.id, address_id=address.id, product_ids=[p.id for p in products])
33
34    finally:
35        db.close()
36
37
38 if __name__ == "__main__":
39     main()
```

- 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



```
(lab2) PS C:\Users\Anastasia\Desktop\lab2> alembic revision --autogenerate -m "add orders, products, change user"
INFO [alembic.runtime.migration] context impl PostgresqlImpl.
INFO [alembic.runtime.migration] will assume transactional DDL.
INFO [alembic.autogenerate.compare] Detected added table 'products'
INFO [alembic.autogenerate.compare] Detected added table 'orders'
INFO [alembic.autogenerate.compare] Detected added table 'order_products'
INFO [alembic.autogenerate.compare] Detected added column 'users.description'
Generating C:\Users\Anastasia\Desktop\lab2\migrations\versions\8e9fd2ac7cf0_add_orders_products_change_user.py ... done
(lab2) PS C:\Users\Anastasia\Desktop\lab2> alembic upgrade head
INFO [alembic.runtime.migration] context impl PostgresqlImpl.
INFO [alembic.runtime.migration] will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade 5fb292750b0b -> 8e9fd2ac7cf0, add orders, products, change user
(lab2) PS C:\Users\Anastasia\Desktop\lab2>
```

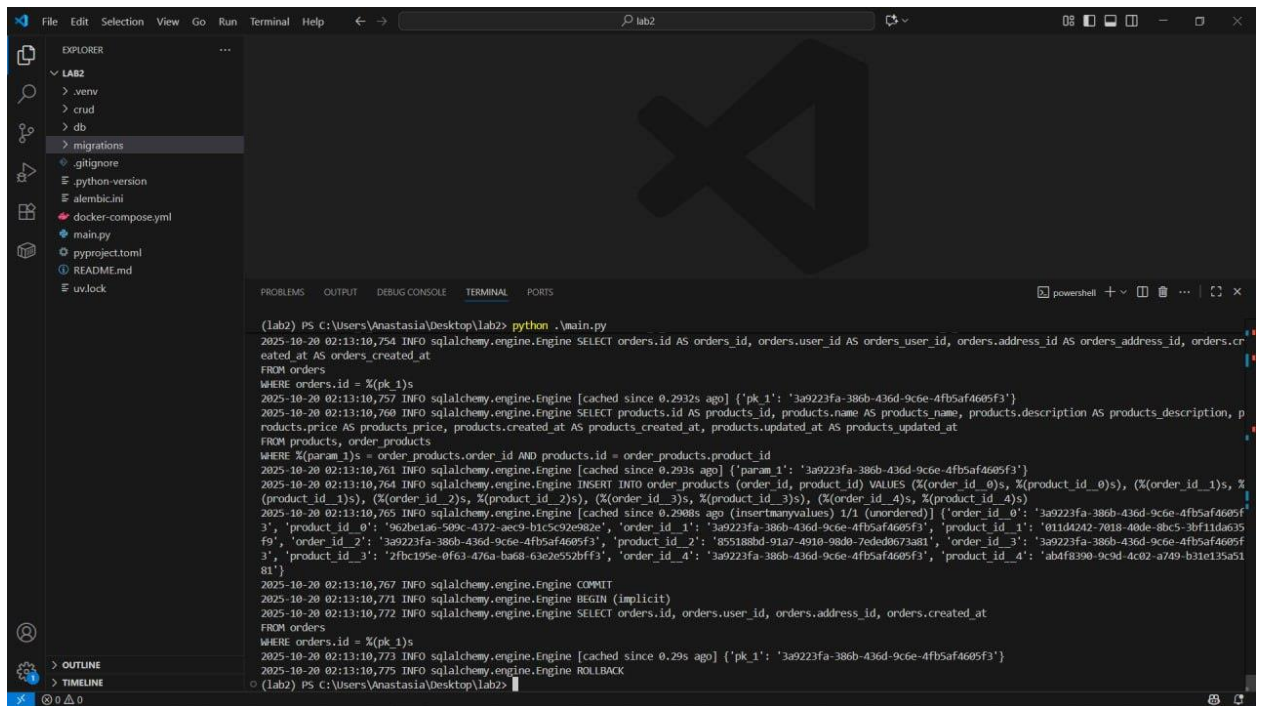
Создание миграций:

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

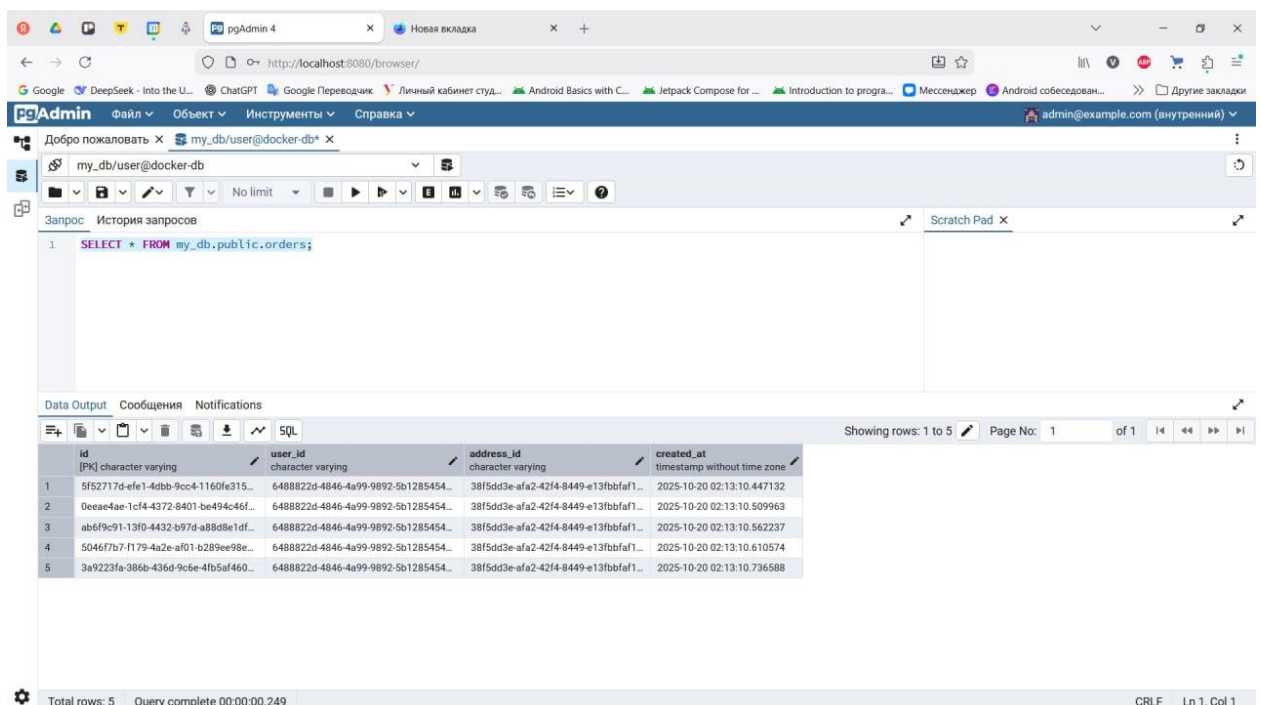
- обнаружено добавление таблиц products, orders и order_products.
- обнаружено добавленное поле users.description

Применение миграции к БД:

alembic upgrade head



```
(lab2) PS C:\Users\Anastasia\Desktop\lab2> python .\main.py
2025-10-20 02:13:10,754 INFO sqlalchemy.engine.Engine SELECT orders.id AS orders_id, orders.user_id AS orders_user_id, orders.address_id AS orders_address_id, orders.created_at AS orders_created_at
FROM orders
WHERE orders.id = %(pk_1)s
2025-10-20 02:13:10,757 INFO sqlalchemy.engine.Engine [cached since 0.2932s ago] {'pk_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3'}
2025-10-20 02:13:10,764 INFO sqlalchemy.engine.Engine INSERT INTO order_products (order_id, product_id) VALUES (%(order_id_0)s, %(product_id_0)s), (%(order_id_1)s, %(product_id_1)s), (%(order_id_2)s, %(product_id_2)s), (%(order_id_3)s, %(product_id_3)s), (%(order_id_4)s, %(product_id_4)s)
2025-10-20 02:13:10,765 INFO sqlalchemy.engine.Engine [cached since 0.2908s ago (insertmanyvalues) 1/1 (unordered)] {'order_id_0': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_0': '962be1a6-509c-4372-aec9-b1c5c92e982e', 'order_id_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_1': '011d4242-7018-40de-8bc5-3bf11da635f9', 'order_id_2': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_2': '855188bd-91a7-4910-98d0-7eded0673a81', 'order_id_3': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_3': '2fbc195e-0f63-476a-ba68-63e2e552bfff', 'order_id_4': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_4': 'ab4f8390-9c9d-4c02-a749-b31e135a5181'}
2025-10-20 02:13:10,767 INFO sqlalchemy.engine.Engine COMMIT
2025-10-20 02:13:10,771 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-20 02:13:10,772 INFO sqlalchemy.engine.Engine SELECT orders.id, orders.user_id, orders.address_id, orders.created_at
FROM orders
WHERE orders.id = %(pk_1)s
2025-10-20 02:13:10,773 INFO sqlalchemy.engine.Engine [cached since 0.29s ago] {'pk_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3'}
2025-10-20 02:13:10,775 INFO sqlalchemy.engine.Engine ROLLBACK
(lab2) PS C:\Users\Anastasia\Desktop\lab2>
```



	id [PK] character varying	user_id character varying	address_id character varying	created_at timestamp without time zone
1	5f52717d-efe1-4dbb-9cc4-1160fa315...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13fbbfaf1...	2025-10-20 02:13:10.447132
2	0eeae4ae-1ef4-4372-8401-be494c46f...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13fbbfaf1...	2025-10-20 02:13:10.509963
3	ab6f9c91-13f0-4432-b97d-a88d9e1df...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13fbbfaf1...	2025-10-20 02:13:10.562237
4	5046f7b7-f179-4a2e-af01-b289ee98e...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13fbbfaf1...	2025-10-20 02:13:10.610574
5	3a9223fa-386b-436d-9c6e-4fb5af460...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13fbbfaf1...	2025-10-20 02:13:10.736588

pgAdmin 4

Новая вкладка

http://localhost:8080/browser/

GoogleDeepSeek - Into the U...ChatGPTGoogle ПереводчикЛичный кабинет студ...Android Basics with C...Jetpack Compose for ...Introduction to progra...МессенджерAndroid собеседован...Другие закладки

pgAdminФайлОбъектИнструментыСправкаadmin@example.com (внутренний)

Добро пожаловатьmy_db/user@docker-db

ЗапросИстория запросовScratch Pad

```
1 SELECT * FROM my_db.public.products;
```

Data OutputСообщенияNotifications

Showing rows: 1 to 5Page 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-ba68-63e2e552bf3	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: 5Query complete 00:00:00.171CRLF Ln 1, Col 37

pgAdmin 4

Новая вкладка

http://localhost:8080/browser/

GoogleDeepSeek - Into the U...ChatGPTGoogle ПереводчикЛичный кабинет студ...Android Basics with C...Jetpack Compose for ...Introduction to progra...МессенджерAndroid собеседован...Другие закладки

pgAdminФайлОбъектИнструментыСправкаadmin@example.com (внутренний)

Добро пожаловатьmy_db/user@docker-db

ЗапросИстория запросовScratch Pad

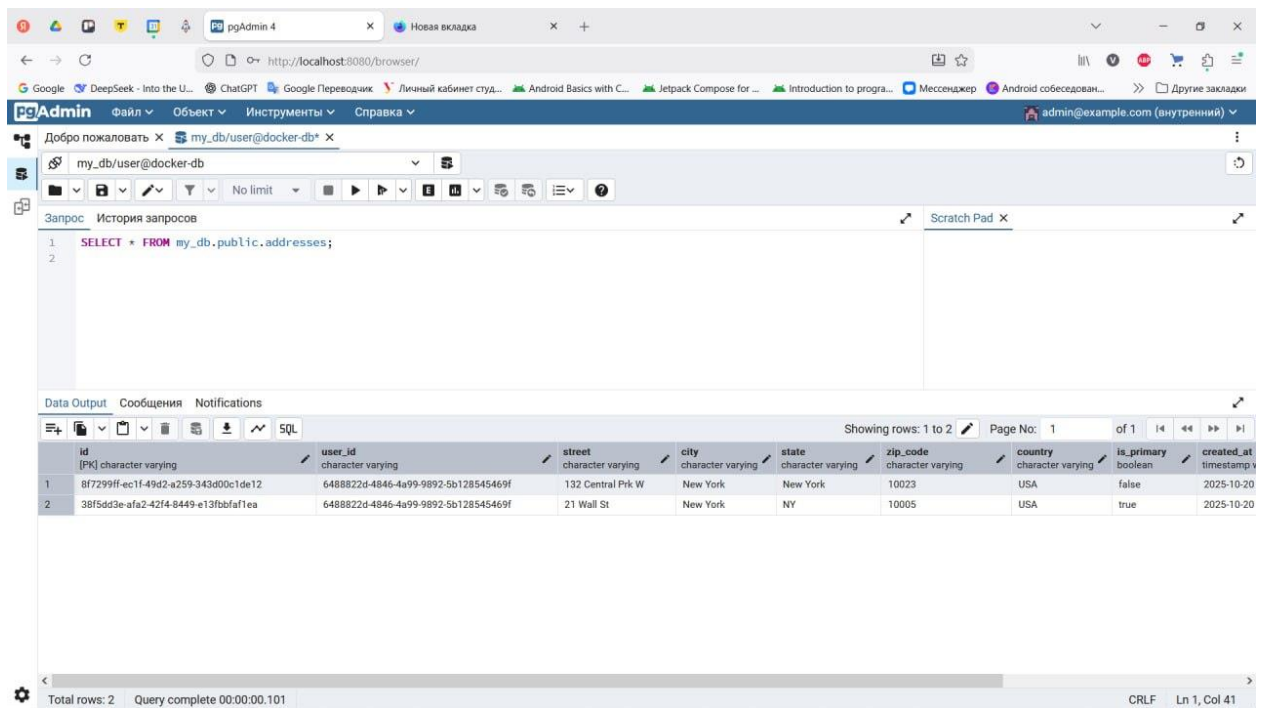
```
1 SELECT * FROM my_db.public.users;
```

Data OutputСообщенияNotifications

Showing rows: 1 to 1Page 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: 1Query complete 00:00:00.171CRLF 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`. Требуется 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.