**Отчет  
по лабораторной работе №2. Проектирование и реализация приложения с многоуровневой архитектурой.**

Выполнил  
студент группы 23-КМ-ПР1,  
института КСиИБ,  
Харченко Анатолий Сергеевич

**Цель работы**

Освоить принцип разделения приложения на уровни и организовать взаимодействие между уровнями.

**Задание на лабораторную работу**

1) Создать приложение, используя многоуровневый подход к по строению архитектуры.

2) Составить тесты для каждого из уровней приложения, проверяющие корректность их работы.

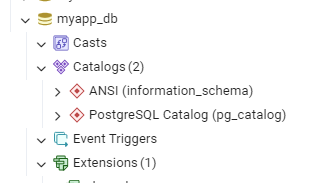
3) Запустить тестирование и проверить результаты работы.

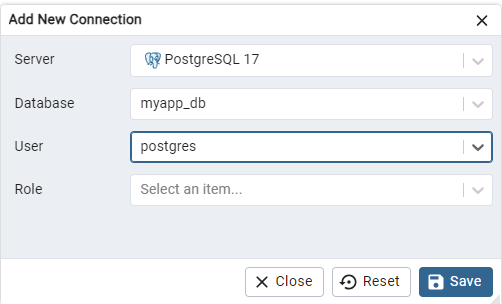
4) Составить отчет.

**Разработка**

Для данной лабораторной работы были использованы postgresSQL + Python + VS Code.

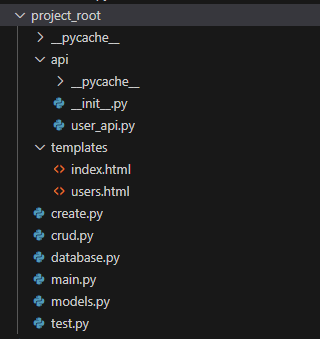
Был скачан postgresSQL, установлен и настроен. Создана база данных:



Далее создано подключение:  


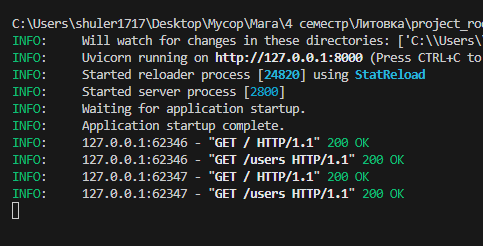
В VS Code в свою очередь были докачаны необходимые библиотеки, для работы с базой данных, а также использованием API в будущем.

Приложение было разработано многоуровневое:

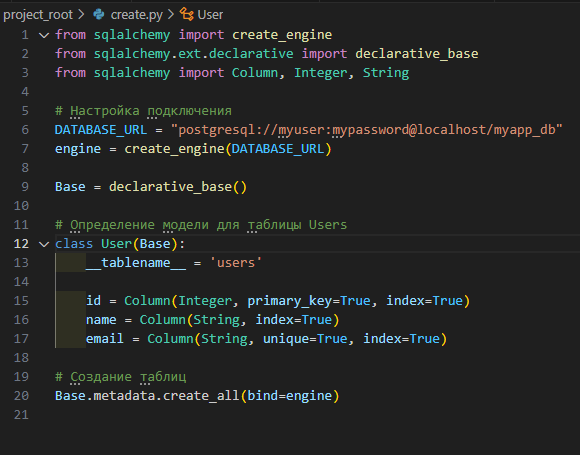


Где верхние уровни задаются для самого приложения и основных модулей, а нижние для подключения к API. В нижних уровнях указаны зависимости от верхних уровней.

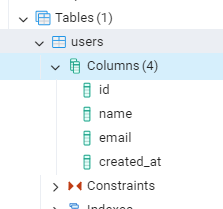
Запускаем корневой терминал, вводим команду «uvicorn main:app --reload» для запуска приложения:



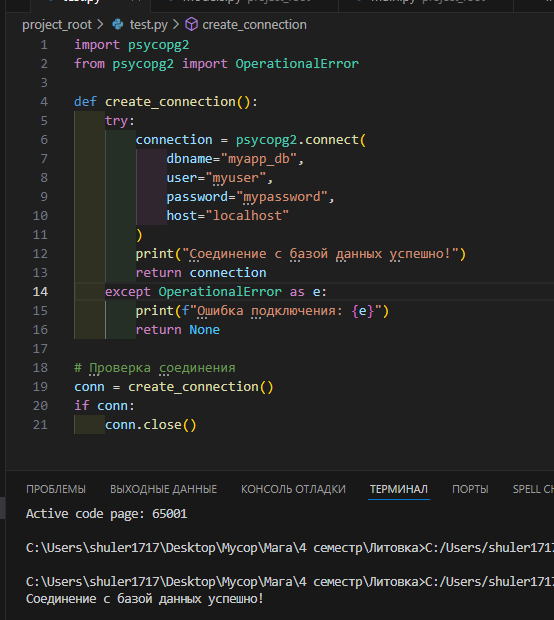
Видим, что программа успешно работает. Создаем в базу таблицу с помощью нашей программы:



Проверяем в БД:



База данных успешно создана. Проверяем подключение с помощью тестового модуля:



Код программы

Модуль main  
from fastapi import FastAPI, Depends

from fastapi.responses import HTMLResponse

from fastapi.templating import Jinja2Templates

from sqlalchemy.orm import Session

from starlette.requests import Request

from database import SessionLocal, engine

from models import Base, User

import crud

# Создаем все таблицы в базе данных

Base.metadata.create\_all(bind=engine)

app = FastAPI()

# Инициализация шаблонов Jinja2

templates = Jinja2Templates(directory="templates")

# Функция для получения сессии базы данных

def get\_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()

# Маршрут для главной страницы

@app.get("/", response\_class=HTMLResponse)

async def read\_root(request: Request):

    return templates.TemplateResponse("index.html", {"request": request})

# Маршрут для списка пользователей

@app.get("/users", response\_class=HTMLResponse)

async def list\_users(request: Request, db: Session = Depends(get\_db)):

    users = crud.get\_users(db)

    return templates.TemplateResponse("users.html", {"request": request, "users": users})

models.py - Определение моделей данных

from sqlalchemy import Column, Integer, String

from database import Base

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, index=True)

email = Column(String, unique=True, index=True)

# database.py - Настройка базы данных

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

#SQLALCHEMY\_DATABASE\_URL = "postgresql://myuser:mypassword@localhost/myapp\_db"postgres

SQLALCHEMY\_DATABASE\_URL = "postgresql://postgres:1234@localhost/myapp\_db"

engine = create\_engine(SQLALCHEMY\_DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

# crud.py - Логика работы с БД

from sqlalchemy.orm import Session

from models import User

def get\_users(db: Session):

    return db.query(User).all()

def get\_user(db: Session, user\_id: int):

    return db.query(User).filter(User.id == user\_id).first()

def create\_user(db: Session, name: str, email: str):

    db\_user = User(name=name, email=email)

    db.add(db\_user)

    db.commit()

    db.refresh(db\_user)

    return db\_user

Создание таблицы

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy import Column, Integer, String

# Настройка подключения

DATABASE\_URL = "postgresql://myuser:mypassword@localhost/myapp\_db"

engine = create\_engine(DATABASE\_URL)

Base = declarative\_base()

# Определение модели для таблицы Users

class User(Base):

    \_\_tablename\_\_ = 'users'

    id = Column(Integer, primary\_key=True, index=True)

    name = Column(String, index=True)

    email = Column(String, unique=True, index=True)

# Создание таблиц

Base.metadata.create\_all(bind=engine)