

# Пример проекта

## Структура проекта

Директория / файл	Описание
alembic/	Настройки alembic
conf/	Настройки окружений.
conf/settings	Файлы основных настроек.
db/	Описание структуры базы данных. initializer.py - Инициализация базы данных, метаданных
db/tablesdefinition	Файлы описания структур таблиц и методов взаимодействия с данными.
docker/	Настройки контейнера
docker/data	Данные БД
docker/docker-entrypoint-initdb.d	Скрипты инициализации БД main.sql - Файл скрипта инициализации
docker/docker-compose.yml	Compose файл
src/	Дополнительные модули
main.py	Точка входа
error.log	Файл лога.

## Предварительная настройка

Для работы примера необходимо установить docker.

Клонировать проекта с репозитория

```
git clone https://gitverse.ru/bobrobot/alembictemplate.git
```

Перейти в директорию проекта, создать виртуальное окружение и активировать

```
cd alembictemplate
python3 -m venv env
source env/bin/activate
```

Установить дополнительные модули

```
pip install -r requirements.txt
```

Перейти в директорию docker и в файле docker-compose.yml настроить пути, имя БД, логин и пароль к новой базе данных.

```
services:
  postgres:
    image: postgres:latest
    environment:
      POSTGRES_DB: "learnsqlalchemy"
      POSTGRES_USER: "learner"
      POSTGRES_PASSWORD: "StrongPassword123"
      PGDATA: "/home/sergey/projects/alembictemplate/docker/data/pgdata"
    volumes:
      - ../docker-entrypoint-initdb.d
      - mydata:/home/sergey/projects/alembictemplate/docker/data
    ports:
      - "5430:5432"
  volumes:
    mydata:
```

В файле docker-entrypoint-initdb.d/main.sql изменить имя БД, логин и пароль.

```
CREATE DATABASE learnsqlalchemy;
CREATE USER learner WITH PASSWORD 'StrongPassword123';
ALTER ROLE learner WITH PASSWORD 'StrongPassword123';
GRANT ALL PRIVILEGES ON DATABASE learnsqlalchemy to learner;
```

В директории docker запустить контейнер БД в фоновом режиме.

```
docker compose up -d
```

Для остановки контейнера:

```
docker compose stop
```

Сейчас, запустив контейнер, при помощи консольного клиента psql можно проверить соединение с базой данных для пользователя.

```
psql -d learnsqlalchemy -U learner -W -h 127.0.0.1 -p 5430
```

Для работы с настройками в формате json используется библиотека src/libsettings.py

[Описание библиотеки](#) Создать папку src, скопировать из проекта библиотеку libsettings.py

## Настройки системы

Файлы основных настроек расположены в conf/settings/ Файл base.py

```
'''Loading settings to project'''
import os
from src.libsettings import Jsettings

settingspath = os.path.join('conf', 'settings', 'settings.json')
schemapath = os.path.join('conf', 'settings', 'schema.json')
# dev settings
mysettings = Jsettings(settingsfname= settingspath,
                        schemafname=schemapath)
mysettings.load_settings()
```

Файл schema.json

```
{
  "type": "object",
  "properties": {
    "db_username": {"type": "string"},
    "db_password": {"type": "string"},
    "db_host": {"type": "string"},
    "db_port": {"type": "string"},
    "db_name": {"type": "string"}
  },
  "required": ["db_username", "db_password", "db_host",
               "db_port", "db_name"]
}
```

Файл settings.json

```
{
  "db_username": "learner",
  "db_password": "StrongPassword123",
  "db_host": "127.0.0.1",
  "db_port": "5430",
  "db_name": "learnsqllalchemy"
```

```
}
```

Файл `base.py` проверяет схему и создает объект настроек `mysettings` из файла `settings.json`. Для получения объекта настроек нужно импортировать объект `mysettings` в нужном модуле. В данный момент присутствуют настройки базы данных с префиксом `db_*`

Если такая усложненная система управления настройками покажется излишней, возможно использовать экспорт настроек напрямую из файла.

### Инициализация базы данных

Создаем папку `db`, в ней создаем файл `initializer.py`

```
"""Db classes and initialization"""
from sqlalchemy import create_engine
from sqlalchemy.orm import declarative_base
from conf.settings.base import mysettings

#load engine settings
engine = create_engine(
    "postgresql+psycopg2://{db_username}:{db_password}@{db_host}:{db_port}/{db_name}".format(
        db_username=mysettings.db_username,
        db_password=mysettings.db_password,
        db_host=mysettings.db_host,
        db_port=mysettings.db_port,
        db_name=mysettings.db_name
    ),
    echo=True)
Base = declarative_base()
```

Здесь только создается `engine` для подключения к БД и класс `Base`. При настройке структуры таблиц данный файл будет обновлен, сейчас нужен только класс `Base`.

### Настройка системы версионирования базы данных alembic

Инициализируем `alembic` в корне проекта.

```
alembic init alembic
```

В корне проекта будет создан файл `alembic.ini`, будет создана папка `alembic` с файлами инициализации. В большинстве инструкций параметры подключения задаются в файле `alembic.ini` однако, для доступа к настройкам из единой точки будет использоваться способ установки параметров в файле `env.py`. Поэтому в файле `alembic.ini` переменная `sqlalchemy.url`

должна быть закомментирована. Часть файла alembic.ini

```
#sqlalchemy.url = driver://user:pass@localhost/dbname
```

В файле env.py

- импортируем путь

```
import os
import sys

sys.path.append(os.getcwd())
```

Импортируем настройки, создаем строку соединения и создаем закомментированный ранее в файле alembic.ini параметр sqlalchemy.url

```
from conf.settings.base import mysettings

connstring =
"postgresql+psycopg2://{db_username}:{db_password}@{db_host}:{db_port}/{db_name}".format(
    db_username=mysettings.db_username,
    db_password=mysettings.db_password,
    db_host=mysettings.db_host,
    db_port=mysettings.db_port,
    db_name=mysettings.db_name
)
config.set_main_option(name="sqlalchemy.url", value=connstring)
```

Импортируем db.initializer и создаем метаданные

```
import db.initializer

target_metadata = db.initializer.Base.metadata
```

Остальные параметры оставляем неизменными. Результирующий файл настроек окружения alembic env.py:

```
from logging.config import fileConfig
import os
import sys

from sqlalchemy import engine_from_config
```

```

from sqlalchemy import pool

from alembic import context

from conf.settings.base import mysettings

sys.path.append(os.getcwd())

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
if config.config_file_name is not None:
    fileConfig(config.config_file_name)

connstring =
"postgresql+psycopg2://{db_username}:{db_password}@{db_host}:{db_port}/{db_name}".format(
    db_username=mysettings.db_username,
    db_password=mysettings.db_password,
    db_host=mysettings.db_host,
    db_port=mysettings.db_port,
    db_name=mysettings.db_name
)
config.set_main_option(name="sqlalchemy.url", value=connstring)

# add your model's MetaData object here
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata
import db.initializer

target_metadata = db.initializer.Base.metadata

# other values from the config, defined by the needs of env.py,
# can be acquired:
# my_important_option = config.get_main_option("my_important_option")
# ... etc.

```

```
def run_migrations_offline() -> None:
```

```
    """Run migrations in 'offline' mode.
```

This configures the context with just a URL  
and not an Engine, though an Engine is acceptable  
here as well. By skipping the Engine creation  
we don't even need a DBAPI to be available.

Calls to context.execute() here emit the given string to the  
script output.

```
    """
```

```
    url = config.get_main_option("sqlalchemy.url")
    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:
```

```
    """Run migrations in 'online' mode.
```

In this scenario we need to create an Engine  
and associate a connection with the context.

```
    """
```

```
    connectable = engine_from_config(
        config.get_section(config.config_ini_section, {}),
        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()

```

### Обновление конфигурации таблиц

Для проверки создаем первую пустую миграцию. После ее выполнения создастся таблица `alembic_version`

```
alembic revision -m "Empty Init"
```

В данный момент фактического соединения с БД не было. В папке `versions` сформируется файл вида `<id>_empty_init.py`

После выполнения команды

```
alembic upgrade head
```

в таблице `alembic_version` появится одна запись - идентификатор текущей версии базы данных.

Сейчас в папке `db` создаем папку `tablesdefinition`. В ней будем хранить файлы описаний таблиц и методы для работы с таблицами. Создадим файл `userprofile.py`

```

"""Definition tables of userprofile"""
import logging
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import sessionmaker
import db.initializer
from db.initializer import engine

def create_userprofile_class(Curbase):
    class Userprofile(Curbase):
        """Class Userprofile definition"""

```



```

__tablename__ = 'userprofile'

user_id = Column(Integer(), primary_key=True)
username = Column(String(15), nullable=False, unique=True)
password = Column(String(255), nullable=False)
email = Column(String(255))
name = Column(String(100))
second_name = Column(String(100))
photo = Column(String(255))
balance = Column(Integer())

return Userprofile

def create_one_userprofile(username, password, email="", name="",
                           second_name="", photo="", balance=0):
    """ Create one userprofile """
    try:
        with engine.connect():
            Session = sessionmaker(bind=engine)
            with Session() as sess:
                upelem = db.initializer.userprofile(username=username, password=password,
                                                    email=email, name=name, second_name=second_name,
                                                    photo=photo, balance=balance)

                sess.add(upelem)
                sess.commit()
    except Exception as e:
        logging.error(e)

```

И в файле `initializer.py` после инициализации переменной `Base` добавим раздел инициализации описания таблицы

```

#===== Creation classes definitions
=====
# === Import Userprofile class ===
from db.tablesdefinition.userprofile import create_userprofile_class
userprofile = create_userprofile_class(Base)
#
=====
=====

```

Теперь после выполнения команды

```
alembic revision --autogenerate -m "Added userprofile model"
```

будет автоматически сгенерирован файл миграции, и после

```
alembic upgrade head
```

создастся таблица userprofile.

P.s. В точке входа необходимо полностью импортировать initializer иначе будет ошибка, пример:

```
'''Main learning module'''
import logging

from db import initializer
from db.tablesdefinition.userprofile import create_one_userprofile

logging.basicConfig(level=logging.INFO,
                    filename='error.log',
                    format="%(levelname)s %(message)s")

if __name__ == '__main__':
    create_one_userprofile(username = 'first6',
                           password = 'first',
                           balance = 1)
```

---

Revision #12

Created 23 October 2024 08:39:59 by Admin

Updated 20 November 2024 16:12:05 by Admin