Flask-sqlalchemy-blueprint

Что нужно?

Установим всё необходимое

```
pip install flask Flask-SQLAlchemy flask-blueprint SQLAlchemy-serializer
```

Предварительная подготвка

Необходимые модели и ORM

Подготовим подключение к базе данных, создадим папки data и db в папке data будут находится описание моделей (model_name.py) их перечень (__all_models.py), логика подключения к базе данных () и заготовка для REST-сервиса (REST_service.py).

db_session.py

```
import sqlalchemy as sa
from sqlalchemy.log import echo_property
import sqlalchemy.orm as orm
from sqlalchemy.orm import Session
import sqlalchemy.ext.declarative as dec


SqlAchemyBase = dec.declarative_base()

__factory = None


def global_init(db_file):
    global __factory

    if __factory:
        return

    if not db_file or not db_file.strip():
        raise Exception('Wrong database-file')

    conn_str = f'sqlite:///{db_file.strip()}?check_same_thread=False'
    print(f'{conn_str} connected')

    engine = sa.create_engine(conn_str, echo=False)
    __factory = orm.sessionmaker(bind=engine)

    from . import __all_models

    SqlAchemyBase.metadata.create_all(engine)


def create_session() -> Session:
    global __factory
    return __factory()
```

Первая модель

works.py

```
import datetime
import sqlalchemy
from sqlalchemy import sql
from .db_session import SqlAchemyBase
from sqlalchemy_serializer import SerializerMixin


class Work(SqlAchemyBase, SerializerMixin):
    __tablename__ = 'works'

    id = sqlalchemy.Column(sqlalchemy.Integer,
                           primary_key=True,
                           autoincrement=True)
    name = sqlalchemy.Column(sqlalchemy.String, nullable=True)
    about = sqlalchemy.Column(sqlalchemy.String, nullable=True)
    created_date = sqlalchemy.Column(sqlalchemy.DateTime,
                                     default=datetime.datetime.now)
```

Обращаем внимание на **SerializerMixin** - класс-примесь который используется для расширения функционала класса.

Для работы с базой данных добавим Works в __all_models.py

```
from . import works
```

Blueprint

Чтобы не смешивать код обработчиков API для новостей с кодом основного приложения, мы воспользуемся механизмом разделения приложения Flask на независимые модули или Blueprint. Как правило, blueprint — логически выделяемый набор обработчиков адресов, который можно вынести в отдельный файл или модуль. Blueprint работает аналогично объекту приложения Flask, но в действительности он не является приложением. Обычно это лишь эскиз для сборки или расширения приложения

REST_service.py

```
import flask
from flask import jsonify
from . import db_session
from .works import Work


blueprint = flask.Blueprint(
    'REST_service',
    __name__,
    template_folder='templates'
)


@blueprint.route('/api/works')
def get_news():
    db_sess = db_session.create_session()
    works = db_sess.query(Work).all()
    return jsonify(
        {
            'works':
                [item.to_dict(only=('name', 'about'))
                 for item in works]
        }
    )
```

зарегистрируем полученный код в основном приложении app.py

```
import json
from flask import Flask
from data import db_session, REST_service


app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret_pass123'


def main():
    db_session.global_init("db/works.db")
    app.register_blueprint(REST_service.blueprint)
    app.run()


if __name__ == '__main__':
    main()
```

В результате запуска основного приложения, по адресу

должны увидеть следующее:

```
{"works":[]}
```

напоследок добавим обработчик добавления работ в виде функции задекорированной с помощью blueprint

```
@blueprint.route('/api/works', methods=['POST'])
def create_news():
    if not request.json:
        return jsonify({'error': 'Empty request'})
    elif not all(key in request.json for key in
                 ['name', 'about']):
        return jsonify({'error': 'Bad request'})
    db_sess = db_session.create_session()
    work = Work(
        name=request.json['name'],
        about=request.json['about']
    )
    db_sess.add(work)
    db_sess.commit()
    return jsonify({'success': 'OK'})
```

и проведём небольшие тесты с помощью бразера и библиотеки requests

from flask import json from requests import post

```
print(post("http://localhost:5000/api/works",
           json={"name": "lab",
                 "about": "About"}
           ).json())
```