Схема взаимодействия

```
| -> main.py (root manager)
| --> driver.py (прослойка подклчюения к БД)
| --> dispatcher.py (обработка запросов от пользователя и маршрутизация их по
проекту)
| ---> schemas.json (набор схем подключаемых таблиц)
```

Константы

`driver/DATABASE_PATH` - путь к базе данных (относительный или абсолютный)

`dispatcher/METHODS` - набор методов, доступных в API

Методы

`dispatcher/Dispatcher`

`Dispathcer` - класс обработки пользовательских запросов и загрузки схемы таблиц БД.

`Dispatcher.dispatch(QueryParam)` - обработчик `query` строки `http` запроса пользователя. Возвращает ответ в формате JSON вида: `{"response": response}`, где `response` - результат выполнения операции в БД.

`driver/DB`

`DB` - статический класс, организующий в себе хранение методов доступа к базе данных.

`DB.create(QueryParam)` - метод создания записей в таблице базы данных, согласно переданным параметрам и схеме таблицы. Возвращает словарь ответа выполнения запроса вида: `{"response": "ok!"}`, в случае успешного выполнения.

`DB.read(QueryParam)` - метод чтения записей из таблицы базы данных согласно переданным параметрам и схеме таблицы. Возвращает список ответа выполнения запроса вида: `list: [[list1], [list2], ...]`, в случае успешного выполнения.

`DB.update(QueryParam)` - метод изменения записей из таблицы базы данных согласно переданным параметрам и схеме таблицы. Возвращает словарь ответа выполнения запроса вида: `{"response": "ok!"}`, в случае успешного выполнения.

`DB.delete(QueryParam)` - метод удаления записей в таблице базы данных, согласно переданным параметрам и схеме таблицы. Возвращает словарь ответа выполнения запроса вида: `{"response": "ok!"}`, в случае успешного выполнения.

`schemas methodology`

Методология "схем" содержится в `Dispatcher` классе и хранит в себе описательную характеристику таблиц из БД. Класс `Dispatcher` при вызове своего внутреннего метода `dispatch()` выгружает из файла `schemas.json` текущий набор "схем" таблиц в БД, и при обращении к определенной таблицы из параметра запроса `table` ищет эту таблицу в списке "схем" и сопоставляет переданные параметры с набором полей из "схемы".

Прим.:

1. *Выполним запрос к API (далее - req):* `http://host:8000/api/request?table=users&method=read&id=1&username=&uniq_key=&role=`
2. *После попадания в точку входа в API, запрос отправляется в главный модуль программы и обрабатывается соответствующим методом*

   `req -> main.py:query_request(Request: request)`
3. *После чтения строки запроса, параметры отправляются в метод* `dispatch()` *класса* `Dispatcher`

   `req -> dispatcher/Dispatcher.dispatch(req)`
4. *Объект класса* `Dispatcher` *выгружает схему таблиц из файла* `schemas.json` *и проверяет в нем наличие таблицы из параметра* `req[table]`. *Если такая таблица существует, то проверяется соответствие переданных аргументов поля таблицы*

   `if req[params] == schemas[req[table]] -> driver/DB`