Задание (Python + PostgreSQL)

Для решения задач можно использовать любой набор инструментов, модулей и фремворков.

Цель задания: написать сервис, который будет слушать входящие запросы по HTTP, преобразовывать их в запрос к соответствующей функции Postgres, выполнять запрос и возвращать ответ клиенту.

Скиллы: Python, Postgres, regexp, строки, работа с json в Python и Postgres.

1. Web-сервис

Написать сервис, который будет слушать входящие запросы по HTTP, преобразовывать их в запрос к соответствующей функции Postgres (по схеме трансляции, приведённой ниже), выполнять запрос и возвращать ответ клиенту.

Как плюс: ограничить максимальное количество одновременных коннектов к БД.
Как два плюса: добавить prometheus метрики на вызовы (количество вызовов, длительность выполнения).

Настройки соединения с сервером Postgres читать из config файла:

- port - (int) порт, на котором слушать запросы
- endpoint - (string) название API
- host - (string) hostname, где установлен Postgres
- user - (string) имя пользователя Postgres
- password - (string) пароль пользователя Postgres
- schema - (string) схема в Postgres

Трансляция запроса в вызов Postgres функции

Формат запроса к сервису:

`HTTP_METHOD server:port/endpoint/vversion[/object/id ...]]/destination/[id]` , где

- `HTTP_METHOD` - одно из: GET, POST, PUT, DELETE
- `server` - сервер, где запущен веб-сервис
- `port` - порт
- `endpoint` - значение из config-файла
- `version` - номер версии API, число
- `/object/id` - *необязательный* повторяющийся параметр, определяющий путь в иерархии объектов
- `/destination/` - конечный объект
- `id` - id конечного объекта. Обязателен для методов PUT, DELETE, не указывается для POST. Для GET -- если указан, то возвращает элемент с данным id, если не указан, возвращает полный список элементов.

Правила трансляции

запрос в Postgres = select * from `схема`.[`object1`[`_object2`]...]`_destination_method`([`id1`[, `id2`]... ,] `id`[, `params`])

В зависимости от HTTP метода к имени функции добавляется суффикс `method`:

- для GET - `get`
- для POST - `ins`
- для PUT - `upd`
- для DELETE - `del`

В случае, если идентификатор объекта не указан, соответствующий элемент `id` в запросе должен быть равен нулю, на примерах:

- для запроса GET `http://localhost:80/api/v1/user/`**12**`/comment/`**34**
  - запрос в Postgres должен выглядеть так: `select * from test.user_comment_get(`**12, 34**`)` (комментарий с id=32 пользователя с id=12)
- для запроса GET `http://localhost:80/api/v1/user/`**12**`/comment/`
  - запрос в Postgres должен выглядеть так: `select * from test.user_comment_get(`**12, 0**`)` (все комментарии пользователя 12)
- для запроса GET `http://localhost:80/api/v1/user/comment/`
  - запрос в Postgres должен выглядеть так: `select * from test.user_comment_get(`**0, 0**`)` (все комментарии всех пользователей)

Для POST и PUT методов в теле запроса принимается JSON, который передаётся в Postgres в качестве параметра `params`.

Все методы должны возвращать результат работы соответствующей Postgres функции с ContentType = 'application/json'

2. PostgreSQL часть

Реализовать на стороне Postgres'а функции для работы с объектами

- user : просмотр, добавление, редактирование, удаление (см. пример в sample.sql)
- comment : просмотр, редактирование, удаление по id
- user/XX/comment : просмотр комментариев пользователя XX, добавление комментария от пользователя XX

3. Примеры

- **GET localhost:80/api/v1/user/34452**
  - Транслируется в: **select * from test.user_get(34452)**
  - Физический смысл: Получить данные по пользователю 34452
  - Ответ сервиса: `{"id":34452, "name":"Vasya", "email":"vasya@google"}`
- **GET localhost:80/api/v1/comment/456**
  - Транслируется в: **select * from test.comment_get(456)**
  - Физический смысл: Получить комментарий с ID 456
  - Ответ сервиса: `{"id":456, "id_user":34452, "txt":"My comment"}`
- **GET localhost:80/api/v1/user/34452/comment/**

    - Транслируется в: **select * from test.user_comment_get(34452, 0)**
    - Физический смысл: Получить все комментарии пользователя 34452
    - Ответ сервиса: `[{"id":456, "id_user":34452, "txt":"My comment"},{"id":460, "id_user":34452, "txt":"Foo!"}]`
- **GET localhost:80/api/v1/user/34452/comment/456**
    - Транслируется в: **select * from test.user_comment_get(34452, 456)**
    - Физический смысл: Получить комментарий с ID 456 от пользователя 34452
    - Ответ сервиса: `{"id":456, "id_user":34452, "txt":"My comment"}`
- **POST localhost:80/api/v1/user/34452/comment/**
  body: `{"txt":"foo"}`
    - Транслируется в: **select * from test.user_comment_ins(34452, '{"txt":"foo"}')**
    - Физический смысл: Создать комментарий от пользователя 34452
    - Ответ сервиса: `{"id": 470}`
- **PUT localhost:80/api/v1/comment/460**
  body: `{"txt":"bar"}`
    - Транслируется в: **select * from test.comment_upd(34452, 460, '{"txt":"bar"}')**
    - Физический смысл: Изменить комментарий с ID 460
    - Ответ сервиса: `{"id":460}`
- **DELETE localhost:80/api/v1/comment/460**
    - Транслируется в: **select * from test.user_comment_del(34452, 460)**
    - Физический смысл: Удалить комментарий с ID 460
    - Ответ сервиса: `{"id":460}`