Задание

Чтобы немного скрасить жизнь людей на самоизоляции, вы решаете открыть интернет-магазин по доставке конфет "Сласти от всех напастей".

Ваша задача — разработать на python REST API сервис, который позволит нанимать курьеров на работу, принимать заказы и оптимально распределять заказы между курьерами, попутно считая их рейтинг и заработок.

Сервис необходимо развернуть на предоставленной виртуальной машине на 0.0.0.0:8080.

Описание обработчиков REST API

Подробная схема всех обработчиков описана в `openapi.yaml`

1: POST /couriers

Для загрузки списка курьеров в систему, запланирован описанный ниже интерфейс.

Обработчик принимает на вход в формате `json` список с данными о курьерах и графиком их работы.
Курьеры работают только в заранее определенных районах,
а так же различаются по типу: пеший, велокурьер и курьер на автомобиле. От типа курьера
зависит его грузоподъемность — 10 кг, 15 кг и 50 кг соответственно.
Районы задаются целыми положительными числами. График работы задается списком строк формата `HH:MM-HH:MM`.

Пример запроса:

```
POST /couriers
{
    "data": [
        {
            "courier_id": 1,
            "courier_type": "foot",
            "regions": [1, 12, 22],
            "working_hours": ["11:35-14:05", "09:00-11:00"]
        },
        {
            "courier_id": 2,
            "courier_type": "bike",
            "regions": [22],
            "working_hours": ["09:00-18:00"]
        },
        {
            "courier_id": 3,
            "courier_type": "car",
            "regions": [12, 22, 23, 33],
            "working_hours": []
        },
        ...
    ]
}
```

Поле	Тип	Описание
courier_id	Целое положительное число	Уникальный идентификатор курьера, положительное число. Идентификаторы уникальны в пределах всего сервиса.
courier_type	Строка	Тип курьера. Возможные значения: `foot` — пеший курьер `bike` — велокурьер `car` — курьер на автомобиле
regions	Массив целых положительных чисел	Список идентификаторов районов, в которых работает курьер.
working_hours	Массив строк	График работы курьера. Формат строки `HH:MM-HH:MM`. Есть гарантия на то, что промежутки, переданные тестирующей системой, не будут пересекаться.

Все поля обязательны.

В случае, если в наборе есть неописанные поля или какие-либо из полей отсутствуют — следует вернуть ошибку `HTTP 400 Bad Request` и список id, которые не удалось провалидировать.

```
HTTP 400 Bad Request
{
    "validation_error": {
        "couriers": [{"id": 2}, {"id": 3}]
    }
}
```

В случае успеха — вернуть ответ `HTTP 201 Created` и списком импортированных id.

```
HTTP 201 Created
{
    "couriers": [{"id": 1}, {"id": 2}, {"id": 3}]
}
```

2: PATCH /couriers/$courier_id

Позволяет изменить информацию о курьере. Принимает `json` и любые поля из списка: `courier_type`, `regions`, `working_hours`.
При редактировании следует учесть случаи, когда меняется график и уменьшается грузоподъемность и появляются заказы,
которые курьер уже не сможет развести — такие заказы должны сниматься и быть доступными для выдачи другим курьерам.

```
PATCH /couriers/2
{
    "regions": [11, 33, 2]
}
```

В случае, если передано неописанное поле — вернуть `HTTP 400 Bad Request`.

В случае успеха — `HTTP 200 OK` и актуальную информацию о редактируемом курьере.

```
HTTP 200 OK
{
    "courier_id": 2,
    "courier_type": "foot",
    "regions": [11, 33, 2],
    "working_hours": ["09:00-18:00"]
}
```

3: POST /orders

Принимает на вход в формате `json` список с данными о заказах. Заказы характеризуются весом, районом и временем доставки.

Пример запроса:

```
POST /orders
{
    "data": [
        {
            "order_id": 1,
            "weight": 0.23,
            "region": 12,
            "delivery_hours": ["09:00-18:00"]
        },
        {
            "order_id": 2,
            "weight": 15,
            "region": 1,
            "delivery_hours": ["09:00-18:00"]
        },
        {
            "order_id": 3,
            "weight": 0.01,
            "region": 22,
            "delivery_hours": ["09:00-12:00", "16:00-21:30"]
        },
        ...
    ]
}
```

Поле	Тип	Описание
order_id	Целое положительное число	id заказа, уникален в пределах всех заказов сервиса.
weight	Положительное число с плавающей точкой	Вес заказа в кг. 2 значащих разряда после запятой. Значения меньше 0.01 и больше 50 считать невалидными.
region	Целое положительное число	Район доставки заказа.
delivery_hours	Массив строк	Промежутки, в которые клиенту удобно принять заказ. Формат строки `HH:MM-HH:MM`. Есть гарантия на то, что промежутки, переданные тестирующей системой, не будут пересекаться.

В случае, если в наборе есть неописанные поля или какие-либо из полей отсутствуют — следует вернуть ошибку `HTTP 400 Bad Request`.

Пример ответа:

```
HTTP 400 Bad Request
{
    "validation_error": {
        "orders": [{"id": 2}, {"id": 3}]
    }
}
```

В случае успеха — `HTTP 201 Created` и список импортированных id.

Пример:

```
HTTP 201 Created
{
    "orders": [{"id": 1}, {"id": 2}, {"id": 3}]
}
```

4: POST /orders/assign

Принимает id курьера и назначает максимальное количество заказов, подходящих по весу, району и графику работы. Обработчик должен быть идемпотентным. Заказы, выданные одному курьеру, не должны быть доступны для выдачи другому.
Если `/orders/assign` вызывается после того, как курьер уже доставил какие-то ранее выданные заказы, то такие заказы нужно убрать из ответа, а `assign_time` должен остаться тем же. В `assign_time` должно содержаться время успешного назначения заказов.

В случае, когда не удалось найти подходящих заказов, нужно вернуть пустой список; `assign_time` возвращать не нужно.

Пример запроса:

```
POST /orders/assign
{
    "courier_id": 2
}
```

Пример ответа:

```
HTTP 200 OK
{
    "orders": [{"id": 1}, {"id": 2}],
    "assign_time": "2021-01-10T09:32:14.42Z"
}
```

В случае, если передан идентификатор несуществующего курьера, следует вернуть ошибку `HTTP 400 Bad Request`.

5: POST /orders/complete

Принимает 3 параметра: id курьера, id заказа и время выполнения заказа, отмечает заказ выполненным.
В случае, если заказ не найден, был назначен на другого курьера или не назначен вовсе, следует вернуть ошибку `HTTP 400 Bad Request`.
В случае успеха — `HTTP 200 OK` и идентификатор завершенного заказа.

Обработчик должен быть идемпотентным.

Пример запроса:

```
POST /orders/complete
{
    "courier_id": 2,
    "order_id": 33,
    "complete_time": "2021-01-10T10:33:01.42Z"
}
```

Пример ответа:

```
HTTP 200 OK
{
    "order_id": 33
}
```

6: GET /couriers/$courier_id

Возвращает информацию о курьере и дополнительную статистику: рейтинг и заработок.

Пример ответа:

```
GET /couriers/2
{
    "courier_id": 2,
    "courier_type": "foot",
    "regions": [11, 33, 2],
    "working_hours": ["09:00-18:00"],
    "rating": 4.93,
    "earnings": 10000
}
```

Рейтинг рассчитывается следующим образом:

`(60*60 - min(t, 60*60))/(60*60) * 5`

где `t` - минимальное из средних времен доставки по районам (в секундах),
`t = min(td[1], td[2], ..., td[n])`

`td[i]` - среднее время доставки заказов по району `i` (в секундах).

Время доставки одного заказа определяется как разница между временем окончания этого заказа и временем окончания предыдущего заказа (или временем назначения заказов, если вычисляется время для первого заказа).

Если курьер не сделал ни одного заказа, то рассчитывать и возвращать рейтинг не нужно.

Заработок рассчитывается следующим образом:

`sum = N * 500 * C`,

где `N` — это количество развозов (1 развоз — это все заказы, полученные из `/orders/assign`),

`C` — коэффициент, зависящий от типа курьера (пеший — 2, велокурьер — 5, авто — 9).

На что обратить внимание

- Валидация входных данных
- Все даты должны быть в формате, удовлетворяющем ISO 8601 и RFC 3339
- Статусы HTTP ответов
- Структура `json` на входе и выходе
- Типы данных (строки, числа)
- URL без trailing slash

Как производится оценка задания

Мы проверяем решения роботом и вручную.
Задание считается выполненным, если в REST API реализованы и проходят валидацию
пять первых обработчиков (последовательно, не более 1 запроса в один момент времени).

Также оценивается:

- Наличие реализованного обработчика `6: GET /couriers/$courier_id`
- Наличие структуры с подробным описанием ошибок каждого некорректного поля, пришедшего в запросе
- Явно описанные внешние python-библиотеки (зависимости)
- Наличие тестов
- Наличие файла `README` в корне репозитория с инструкциями по установке, развертыванию и запуску сервиса и тестов
- Автоматическое возобновление работы REST API после перезагрузки виртуальной машины
- Возможность обработки нескольких запросов сервисом одновременно