Задание

Интернет-магазин подарков хочет запустить акцию в разных регионах. Чтобы стратегия продаж была эффективной, необходимо произвести анализ рынка.

У магазина есть поставщик, регулярно присылающий выгрузки данных с информацией о жителях. Проанализировав их, можно выявить спрос на подарки в разных городах у жителей разных возрастных групп по месяцам.

Ваша задача - разработать на python REST API сервис, который сохраняет переданные ему наборы данных (выгрузки от поставщика) с жителями, позволяет их просматривать, редактировать информацию об отдельных жителях, а также производить анализ возрастов жителей по городам и анализировать спрос на подарки в разных месяцах для указанного набора данных.

Должна быть реализована возможность загрузить несколько независимых наборов данных с разными идентификаторами, независимо друг от друга изменять и анализировать их.

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

Описание обработчиков REST API

1: POST /imports

Принимает на вход набор с данными о жителях в формате `json` и сохраняет его с уникальным идентификатором `import_id`.

В наборе данных для каждого жителя должны присутствовать все поля, значения не могут быть `null`, порядок полей не важен:

Поле	Тип	Значение
citizen_id[1]	целое число	Уникальный идентификатор жителя
town	строка	Название города. Непустая строка, содержащая хотя бы 1 букву или цифру.

street	строка	Название улицы. Непустая строка, содержащая хотя бы 1 букву или цифру.
building	строка	Номер дома, корпус и строение. Непустая строка, содержащая хотя бы 1 букву или цифру.
apartment	целое число	Номер квартиры.
name	строка	Имя, фамилия, отчетство (если есть). Непустая строка, состоящая из 2 (или 3х если есть отчество) слов, разделенных пробелом. Минимальная длина каждого слова - 1 буква.
birth_date[2]	строка	Дата рождения в формате ДД.ММ.ГГГГ
gender	строка	Значения `male`, `female`
relatives[3]	список из целых чисел	Ближайшие родственники, уникальные значения существующих `citizen_id` жителей из этой же выгрузки.

(1) Поставщик предупредил, что в разных выгрузках `citizen_id` не уникален и может повторяться у разных жителей, не закладывайтесь на то, что `citizen_id` будут уникальны между выгрузками от поставщика.

(2) помимо проверки на формат ДД.ММ.ГГГГ - дата должна быть существующей (`31.02.2019` - не является валидной датой).
Проверить, что дата является валидной можно с помощью `datetime.date`.

(3) Родственные связи двусторонние (если у жителя #1 в родственниках указан житель #2, то и у жителя #2 должен быть родственник #1).
Родственные связи `relatives` актуальны только в рамках одной выгрузки.
В одной выгрузке будет не более 1000 родственных связей (2000 связанных жителей).

```
POST /imports
{
  "citizens": [
    {
      "citizen_id": 1,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
```

```
      "apartment": 7,
      "name": "Иванов Иван Иванович",
      "birth_date": "26.12.1986",
      "gender": "male",
      "relatives": [2]  // id родственников
    },
    {
      "citizen_id": 2,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
      "apartment": 7,
      "name": "Иванов Сергей Иванович",
      "birth_date": "17.04.1997",
      "gender": "male",
      "relatives": [1]  // id родственников
    },
    {
      "citizen_id": 3,
      "town": "Керчь",
      "street": "Иосифа Бродского",
      "building": "2",
      "apartment": 11,
      "name": "Романова Мария Леонидовна",
      "birth_date": "23.11.1986",
      "gender": "female",
      "relatives": []
    },
    ...
  ]
}
```

В случае успеха возвращается ответ с HTTP статусом `201 Created` и идентификатором импорта:

```
HTTP 201
{
  "data": {
    "import_id": 1
  }
}
```

2: PATCH /imports/$import_id/citizens/$citizen_id

Изменяет информацию о жителе в указанном наборе данных.

На вход подается JSON в котором можно указать любые данные о жителе (`name` , `gender` , `birth_date` , `relatives` , `town` , `street` ,

`building`, `apartment`), кроме `citizen_id`.

В запросе должно быть указано хотя бы одно поле, значения не могут быть `null`.

Если в запросе указано поле `relatives` - изменение родственных связей должно быть двусторонним.

Например, есть два брата - Ивановы Иван (`citizen_id=1`) и Сергей (`citizen_id=2`). Мария Леонидовна (`citizen_id=3`) вышла замуж за Ивана, стала ему ближайшей родственницей и переехала в Москву:

```
PATCH /imports/1/citizens/3
{
    "name": "Иванова Мария Леонидовна",
    "town": "Москва",
    "street": "Льва Толстого",
    "building": "16к7стр5",
    "apartment": 7,
    "relatives": [1]
}
```

В результате этого запроса данные о жителях должны прийти в следующее состояние:

- Житель 1: relatives = [2, 3] (житель #2 брат, житель #3 супруга)
- Житель 2: relatives = [1] (житель #1 брат)
- Житель 3: relatives = [1] (житель #1 супруг)

Возвращается актуальная информация об указанном жителе:

```
HTTP 200
{
  "data": {
      "citizen_id": 3,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
      "apartment": 7,
      "name": "Иванова Мария Леонидовна",
      "birth_date": "23.11.1986",
      "gender": "female",
      "relatives": [1]
  }
}
```

Если девушка разведется с супругом, необходимо будет выполнить следующий запрос:

```
PATCH /imports/1/citizens/3
{
    "relatives": []
}
```

В результате этого запроса данные о жителях должны прийти в следующее состояние:

- Житель 1: relatives = [2] (житель #2 брат)
- Житель 2: relatives = [1] (житель #1 брат)
- Житель 3: relatives = []

И вернется актуальная информация об указанном жителе:

```
HTTP 200
{
  "data": {
      "citizen_id": 3,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
      "apartment": 7,
      "name": "Иванова Мария Леонидовна",
      "birth_date": "23.11.1986",
      "gender": "female",
      "relatives": []
  }
}
```

3: GET /imports/$import_id/citizens

Возвращает список всех жителей для указанного набора данных.

```
HTTP 200
{
  "data": [
    {
      "citizen_id": 1,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
      "apartment": 7,
      "name": "Иванов Иван Иванович",
      "birth_date": "26.12.1986",
      "gender": "male",
```

```
      "relatives": [2,3]  // id родственников
    },
    {
      "citizen_id": 2,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
      "apartment": 7,
      "name": "Иванов Сергей Иванович",
      "birth_date": "17.04.1997",
      "gender": "male",
      "relatives": [1]  // id родственников
    },
    {
      "citizen_id": 3,
      "town": "Москва",
      "street": "Льва Толстого",
      "building": "16к7стр5",
      "apartment": 7,
      "name": "Иванова Мария Леонидовна",
      "birth_date": "23.11.1986",
      "gender": "female",
      "relatives": [1]
    },
    ...
  ]
}
```

4: GET /imports/$import_id/citizens/birthdays

Возвращает жителей и количество подарков, которые они будут покупать своим ближайшим родственникам (1-го порядка), сгруппированных по месяцам из указанного набора данных.

Ключом должен быть месяц (нумерация должна начинаться с единицы, `"1"` - январь, `"2"` - февраль и т.п.).

Если в импорте в каком-либо месяце нет ни одного жителя с днями рождения ближайших родственников, значением такого ключа должен быть пустой список.

```
HTTP 200
{
  "data": {
    "1": [],
    "2": [],
    "3": [],
    "4": [{
      "citizen_id": 1,
      "presents": 1,
```

```
    }],
    "5": [],
    "6": [],
    "7": [],
    "8": [],
    "9": [],
    "10": [],
    "11": [{
      "citizen_id": 1,
      "presents": 1
    }],
    "12": [
      {
        "citizen_id": 2,
        "presents": 1
      },
      {
        "citizen_id": 3,
        "presents": 1
      }
    ]
  }
}
```

5: GET /imports/$import_id/towns/stat/percentile/age

Возвращает статистику по городам для указанного набора данных в разрезе возраста (полных лет) жителей: p50, p75, p99, где число - это значение перцентиля.

```
HTTP 200
{
  "data": [
    {
      "town": "Москва",
      "p50": 20,
      "p75": 45,
      "p99": 100
    },
    {
      "town": "Санкт-Петербург",
      "p50": 17,
      "p75": 35,
      "p99": 80
    }
  ]
}
```

Что означает:

- `"p50": 20,` - 50% жителей меньше 20 лет
- `"p75": 45,` - 75% жителей меньше 45 лет

На что обратить внимание

Для прохождения проверки обратите внимание на следующее:

- Статусы HTTP ответов
- Структура json на входе и выходе
- Типы данных (строки, числа)
- Формат даты
- Таймаут на вызов каждого обработчика - 10 секунд при количестве жителей 10,000
- URL без trailing slash
- Реализация перцентилей должна соответствовать numpy.percentile с `interpolation='linear'`

Как производится оценка задания

Мы проверяем решения роботом и вручную.

Задание считается выполненным, если в REST API реализованы и проходят валидацию три первых обработчика (последовательно, не более 1 запроса в один момент времени).

Также учитывается:

- Наличие реализованных обработчиков:
  - 4: `GET /imports/$import_id/citizens/birthdays`
  - 5: `GET /imports/$import_id/towns/stat/percentile/age`
- Наличие валидации входных данных (на некорректные входные данные сервис отвечает HTTP статусом `400 Bad Request`).
  Тело ответа в этом случае не будет проверяться роботом (важен только статус), изменений в базе данных происходить не должно;
- Наличие файла `README` в корне репозитория с инструкциями по установке, развертыванию и запуску тестов;
- Явно описанные внешние python-библиотеки (зависимости);
- Наличие тестов;
- Автоматическое возобновление работы REST API после перезагрузки виртуальной машины;
- Возможность обработки нескольких запросов сервисом одновременно.

Правки

Мы стараемся дополнять задание примерами по мере получения вопросов. В этом разделе мы описали основные дополнения к заданию, чтобы Вы могли быстро сориентироваться.

1. В структурах запросов и ответов поле `appartement` изменено на `apartment`.
   При проверке оба варианта считаются правильными, наш робот попробует оба варианта.

2. Указано, что поле `relatives` может содержать только уникальные идентификаторы существующих горожан. Родственные связи должны быть двусторонними (если у жителя #1 в поле `relatives` есть родственник #2, то и у жителя #2 в поле relatives должен быть родственник #1. В противном случае необходимо возвращать ответ 400: Bad Request

3. Добавлен пример обновления родственных связей для обработчика `PATCH /imports/$import_id/citizens/$citizen_id`. В случае, если обработчику передано поле `relatives` - необходимо также обновить поля `relatives` у ближайших родственников.
   В случае ответа 400: Bad Request мы ожидаем что состояние базы данных не изменилось.

4. Добавлено подробное описание типов данных для обработчика `POST /imports`, указано что порядок полей внутри объекта json не важен.

5. Указано, что решение считается успешным, если сервис реализует три первых обработчика и способен отвечать на запросы по порядку (1 запрос в один момент времеини). Сервисы, которые способны обрабатывать больше 1 запроса одновременно оцениваются выше.

6. Помимо формата даты необходимо проверять, что дата существующая (например, 31 февраля - не валидная дата).

7. В одной выгрузке, которая приходит в обработчик `POST /imports` будет не более 1000 родственных связей (2000 связанных жителей).

8. Добавлены уточнения по валидации входных данных, пример запроса для удаления родственников в обработчике PATCH.

Остались вопросы?

Мы постарались ответить на наиболее популярные вопросы в трансляции на Youtube, посмотреть которую можно здесь.

Если вы не смогли найти ответ на ваш вопрос - напишите нам ответом на письмо с заданием и мы обязательно поможем :)