Highload Cup 2018 (Task)

Добро пожаловать, участник Highload Cup 2018! Внимательно прочитай правила и ТЗ на задачу, изложенные ниже. Если что-то непонятно, не получается или появились идеи, как все это улучшить, ты всегда можешь написать на cups@corp.mail.ru (mailto:cups@corp.mail.ru) или в Telegram (https://t.me/highloadcup). Удачи в нашем конкурсе!

Легенда соревнования

В альтернативной реальности человечество решило создать и запустить глобальную систему по поиску "вторых половинок". Такая система призвана уменьшить количество одиноких людей в мире и способствовать созданию крепких семей.

Участникам соревнования Highload Cup 2018 предлагается выступить в роли инженера, которому заказали создание прототипа подобной системы. Прототип должен как можно быстрее выдавать правильные ответы на запросы сторонних сервисов, которые делают что-то с ответами (например, отображают пользователям в красивых интерфейсах). По сути, он должен служить для внешних гипотетических сервисов функциональным API.

1. Правила участия в конкурсе

Для того чтобы начать, %username%, расчехли свою любимую IDE и скачай архив с тестовыми данными в формате JSON с сайта проведения конкурса https://highloadcup.ru (https://highloadcup.ru). Тебе необходимо сначала создать, а затем и развернуть производительный сервер приложения, который будет реализовывать необходимое Web API к этим данным. Обрати внимание, что решения будут приниматься в виде docker-контейнеров. Что это такое - читай ниже.

Ты можешь использовать любые веб-технологии, которые сможешь найти или придумать. Выбери свой собственный язык программирования и фреймворк. Это может быть C++, Java + Tomcat, Python + Django, Ruby + RoR, GoLang, JavaScript + NodeJs, Haskell или что-то еще, на твоё усмотрение. Также и для хранения данных: MySQL, PostgreSQL, Redis, MongoDB, кэши - up-to-you! Обрати внимание, **в конкурсе оценивается не только кол-во верных ответов на запросы, но и скорость работы сервера** - выбирай осторожно!

Сначала обкатай своё решение локально на тестовых данных. А когда будешь готов, собери из него docker-контейнер и залей его в систему проведения конкурса. Соответствующие команды написаны на странице с задачей. После заливки контейнера, на той же странице появится запись о принятом решении и о постановке его в очередь на предварительный обстрел.

Особенность 1. Если одно и то же решение заливается дважды без изменений, то система не будет отправлять его на обстрел. В каждом решении должны быть, пусть минимальные, отличия от предыдущих.

Будет происходить следующее:

1. Решение отправится на тестирующую машину с процессором Intel Core i7. Решению будут выделены 4 ядра по 2.4 GHz, 2 GB оперативной памяти и 10 GB жёсткого диска.
2. Решение будет запущено как docker-контейнер (docker run). В случае возникновения ошибок запуска, они будут показаны на странице сайта с логом обстрела.
3. После запуска контейнера в папке /tmp/data будет доступен файл data.zip с архивированными "боевыми" данными (примерно 10 MB данных для предварительного и 1 GB для полного обстрела). Обратите внимание, что каталог /tmp/data доступен только для чтения, поэтому решение должно загружать архив в ОЗУ для обработки. В самом архиве будут лежать файлы с названиями вида "accounts_<номер файла>.json". Внутри таких файлов - валидные данные в формате JSON.
   Пример структуры приведён ниже:

```
{"accounts": [
    {
        "id": 10003,
        "fname": "Мария",
        "email": "ewheten@icloud.com",
        "interests": [
            "Красное вино",
            "Стейк",
            "Вкусно поесть"
        ],
        "status": "свободны",
        "premium": {
            "start": 1533321770,
            "finish": 1533321770
        },
        "sex": "f",
        "phone": "8(985)4076805",
        "likes": [
            {
                "ts": 1476378752,
                "id": 41803
            },
            ...
        ],
        "birth": 870172195,
        "city": "Испляндия",
        "country": "Кроноштадт",
        "joined": 1450137600
    },
    ... // дальше ещё много аккаунтов
]
```

4. У решения есть фиксированное время до начала обстрела, чтобы залить эти данные в собственную базу данных и подготовить их к обработке (1 минута для предварительного и 10 минут для полного обстрела).
5. По истечении этого времени начинается обстрел запросами из указанных в разделе API. длительность обстрела составляет 90 секунд для предварительного и 9 минут для

полного рейтингового обстрела. Важно - сервер должен слушать 80-й порт, чтобы обстрел прошел успешно! Запросы идут с заголовком Host: accounts.com по протоколу HTTP/1.1 с переиспользуемыми соединениями (keep-alive). Сетевые потери полностью отсутствуют.

6. Результаты и ошибки обстрела ты увидишь на сайте, на странице с деталями решения в секциях "Обстрел" и "Результаты" соответственно.

Особенность 2. При замеченных попытках хакерских атак на сервера проведения конкурса Highload Cup 2018, участнику выдаётся бан, а результаты обстрела не засчитываются.

Обратите внимание! Предварительный обстрел запускается автоматически и нужен для тестирования решений на малой нагрузке. По такому обстрелу показываются результаты в виде графиков, но не считается рейтинг. Для участия в рейтинге, необходимо вручную запустить рейтинговый обстрел, который проводится в гораздо более хардкорных условиях. Количество рейтинговых обстрелов ограничено, 4 запусками в 12 часов.

Результаты рейтинговых обстрелов всех участников будут сводиться в таблицу на сайте. Лучшие из лучших получат призы!

2. Описание предметной области

Как в тестовых, так и в "боевых" данных имеются записи об одной сущности: Account. Она описывает всю известную информацию о пользователе - его имя, контакты, интересы, выявленные симпатии к другим пользователям. Гарантируется корректность предоставляемых данных в соответствии с указанными далее типами и ограничениями.

В одной записи `Account` (Профиль) имеются следующие личные данные:

- **id** - уникальный внешний идентификатор пользователя. Устанавливается тестирующей системой и используется затем, для проверки ответов сервера. Тип - 32-разрядное целое число.
- **email** - адрес электронной почты пользователя. Тип - unicode-строка длиной до 100 символов. Гарантируется уникальность.
- **fname и sname** - имя и фамилия соответственно. Тип - unicode-строки длиной до 50 символов. Поля опциональны и могут отсутствовать в конкретной записи.
- **phone** - номер мобильного телефона. Тип - unicode-строка длиной до 16 символов. Поле является опциональным, но для указанных значений гарантируется уникальность. Заполняется довольно редко.
- **sex** - unicode-строка "m" означает мужской пол, а "f" - женский.
- **birth** - дата рождения, записанная как число секунд от начала UNIX-эпохи по UTC (другими словами - это timestamp). Ограничено снизу 01.01.1950 и сверху 01.01.2005-ым.
- **country** - страна проживания. Тип - unicode-строка длиной до 50 символов. Поле опционально.

- **city** - город проживания. Тип - unicode-строка длиной до 50 символов. Поле опционально и указывается редко. Каждый город расположен в определённой стране.

Особенность 3. Все данные сгенерированы случайным образом и не имеют отношения к реальным людям, контактам или местам, даже если случились совпадения. Код генератора данных не использует сторонних решений, кроме импортов модулей random, datetime, calendar и string из стандартной библиотеки Python.

Также в одной записи Account есть поля специфичные для системы поиска "второй половинки":

- **joined** - дата регистрации в системе. Тип - timestamp с ограничениями: снизу 01.01.2011, сверху 01.01.2018.
- **status** - текущий статус пользователя в системе. Тип - одна строка из следующих вариантов: "свободны", "заняты", "всё сложно". Не обращайте внимание на странные окончания :)
- **interests** - интересы пользователя в обычной жизни. Тип - массив unicode-строк, возможно пустой. Строки не превышают по длине 100 символов.
- **premium** - начало и конец премиального периода в системе (когда пользователям очень хотелось найти "вторую половинку" и они делали денежный вклад). В json это поле представлено вложенным объектом с полями start и finish, где записаны timestamp-ы с нижней границей 01.01.2018.
- **likes** - массив известных симпатий пользователя, возможно пустой. Все симпатии идут вразнобой и каждая представляет собой объект из следующих полей:
  - **id** - идентификатор другого аккаунта, к которому симпатия. Аккаунт по id в исходных данных всегда существует. В данных может быть несколько лайков с одним и тем же id.
  - **ts** - время, то есть timestamp, когда симпатия была записана в систему.

3. Описание необходимого API

API - это схемы http-запросов, которые должен обслуживать разработанный участником сервер. URL-ы строятся в соответствии с парадигмой REST. В угловых скобках указаны части URL, которые могут и будут меняться от запроса к запросу.

Во всех ответах от сервера учитываются заголовки Content-Type, Content-Length, Connection.

Особенность 4. Все примеры в дальнейшем красиво и аккуратно отформатированы для более лёгкого восприятия. В ответах сервера форматирование не учитывается. Кириллица и спец. символы в URL кодируются python-функцией urlencode().

Запросы выборки данных (GET):

1. Получение списка пользователей: **/accounts/filter/**

Данный метод API планируется использовать для поиска пользователей по заранее известным или желаемым полям. К примеру, кому-то захотелось посмотреть всех людей определённого возраста и пола, кто живёт в определённом городе.

В теле ответа ожидается структура `{"accounts": [ ... ]}` с пользователями, данные которых соответствуют указанным в GET-параметрах ограничениям. Для каждой подошедшей записи аккаунта не нужно передавать все известные о ней данные, а только поля id, email и те, что были использованы в запросе.

20.12.2018: в этом запросе теперь не нужно выводить данные по **interests** и **likes**. Так сделано, чтобы уменьшить кол-во данных после обстрела, которые хранятся на серверах.

Пользователи в результате должны быть отсортированы по убыванию значений в поле id. Количество выбираемых записей ограничено обязательным GET-параметром limit.

Остальные GET-параметры формируются как `<поле>_<предикат>` . У разных полей могут использоваться только определённые фильтрующие предикаты, которые перечислены в таблице ниже. В данном запросе действие нескольких параметров складывается, то есть сначала фильтрация по одному, затем фильтрация результата по второму и т. д.

#	Название поля	Возможные предикаты с расшифровкой
1	sex	`eq` - соответствие конкретному полу - "m" или "f";
2	email	`domain` - выбрать всех, чьи email-ы имеют указанный домен; `lt` - выбрать всех, чьи email-ы лексикографически раньше; `gt` - то же, но лексикографически позже;
3	status	`eq` - соответствие конкретному статусу; `neq` - выбрать всех, чей статус не равен указанному;
4	fname	`eq` - соответствие конкретному имени; `any` - соответствие любому имени из перечисленных через запятую; `null` - выбрать всех, у кого указано имя (если 0) или не указано (если 1);
5	sname	`eq` - соответствие конкретной фамилии; `starts` - выбрать всех, чьи фамилии начинаются с переданного префикса; `null` - выбрать всех, у кого указана фамилия (если 0) или не указана (если 1);

6	phone	`code` - выбрать всех, у кого в телефоне конкретный код (три цифры в скобках); `null` - аналогично остальным полям;
7	country	`eq` - всех, кто живёт в конкретной стране; `null` - аналогично;
8	city	`eq` - всех, кто живёт в конкретном городе; `any` - в любом из перечисленных через запятую городов; `null` - аналогично;
9	birth	`lt` - выбрать всех, кто родился до указанной даты; `gt` - после указанной даты; `year` - кто родился в указанном году;
10	interests	`contains` - выбрать всех, у кого есть все перечисленные интересы; `any` - выбрать всех, у кого есть любой из перечисленных интересов;
11	likes	`contains` - выбрать всех, кто лайкал всех перечисленных пользователей (в значении - перечисленные через запятые id);
12	premium	`now` - все у кого есть премиум на текущую дату; `null` - аналогично остальным;

Конечно, мы не генерируем наборы этих параметров абсолютным рандомом. Разрешены только определённые сочетания и у каждого поля есть вероятность включения его в запрос. Сочетания и вероятности выбраны не просто так - попробуйте использовать эти знания в свою пользу.

Пример запроса и корректного ответа на него:

```
GET: /accounts/filter/?status_neq=всё+сложно&birth_lt=643972596&country_eq=Индляндия&limit=5&query_id=110
```

```
{
    "accounts": [
        {
            "email": "monnorakodehrenod@list.ru",
            "country": "Индляндия",
            "id": 99270,
            "status": "заняты",
            "birth": 581863572
        },{
            "email": "erwirarhadmemeddifde@yahoo.com",
            "country": "Индляндия",
            "id": 98881,
            "status": "свободны",
            "birth": 640015608
        },{
            "email": "rupewseor@rambler.ru",
            "country": "Индляндия",
            "id": 98828,
            "status": "заняты",
            "birth": 604256977
        },{
            "email": "fiotnefaersohhev@inbox.ru",
            "country": "Индляндия",
            "id": 98804,
            "status": "свободны",
            "birth": 596799123
        },{
            "email": "geslasereshedot@yahoo.com",
            "country": "Индляндия",
            "id": 98718,
            "status": "свободны",
            "birth": 640919302
        }
    ]
}
```


Особенность 5. В случае неизвестного поля или неразрешённого предиката, в ответе ожидается код 400 с пустым телом. Во всех остальных случаях ожидается ответ 200, даже если ни одного пользователя не нашлось.

Особенность 6. Во всех запросах API присутствует технический GET-параметр `query_id`. Решение должно просто игнорировать этот параметр, так как он не требует никаких действий.

2. Разбиение пользователей по группам: **/accounts/group/**

   Данный метод API планируется использовать для создания отчётов о работе системы. Поля, по которым производится группировка переданы в GET-параметре keys через запятую. Они не так многочисленны, как в запросе на фильтрацию пользователей. Полей для группировки всего пять - `sex, status, interests, country, city`.

Перед тем как выполнять группировку необходимо выполнить выборку как в предыдущем запросе, но по конкретным значениям, а не по предикатам. К примеру, если в GET-параметрах указано `country=Алания`, значит группировка выполняется только по пользователям из этой страны. Выборка может идти по любому полю, но значение в нём будет только одно (для likes будет только один id, для interests только одна строка, для birth и joined - будет одно число - год).

В теле ответа ожидается структура `{"groups": [ ... ]}` со списком групп. В каждой группе обязательно должны быть ключи, по которым производилась группировка с соответствующими конкретными значениями. Для каждой выявленной группы нужно подсчитать сколько пользователей в неё попало и записать в результате по ключу `count`. То есть, агрегирующая функция для этого запроса только одна и это подсчёт.

В результате нужно вернуть не все выявленные группы, а только N самых крупных или N самых мелких. Число N задаётся GET-параметром `limit=N`, а возвращать ли сначала крупные или сначала мелкие - GET-параметром `order=-1` или `order=1` соответственно. В ответе могут получиться группы с одинаковым count и это может создать проблемы на этапе валидации ответов. Пожалуйста, сортируйте такие группы между собой по значениям других полей в порядке, заданном order.

Обратите внимание, что N не превышает 50. Возможно этот факт поможет с оптимизацией :)

Пример запроса и корректного ответа на него:

```
GET: /accounts/group/?birth=1998&limit=4&order=-1&keys=country
```

(вернуть 4 страны, где больше всего пользователей с годом рождения 1998)

```
{"groups": [
    {"country": "Малатрис", "count": 8745},
    {"country": "Алания", "count": 4390},
    {"country": "Финляндия", "count": 2100},
    {"country": "Гератрис", "count": 547}
]}
```

Особенность 7. При появлении в запросе неожиданных полей группировки или неизвестных GET-параметров в ответе ожидается код 400 с пустым телом ответа.

3. Рекомендации по совместимости: **/accounts/<id>/recommend/**

Данный запрос используется для поиска "второй половинки" по указанным пользовательским данным. В запросе передаётся id пользователя, для которого ищутся те, кто лучше всего совместимы по статусу, возрасту и интересам. Решение должно проверять совместимость только с противоположным полом (мы не против секс-меньшинств и осуждаем дискриминацию, просто так получилось :)). Если в GET-запросе передана страна или город с ключами country и city соответственно, то нужно искать только среди живущих в указанном месте.

В ответе ожидается код 200 и структура `{"accounts": [ ... ]}` либо код 404 , если пользователя с искомым id не обнаружено в хранимых данных. По ключу "accounts" должны быть N пользователей, сортированных по убыванию их совместимости с обозначенным id. Число N задаётся в запросе GET-параметром limit и не бывает больше 20.

Совместимость определяется как функция от двух пользователей: `compatibility = f (me, somebody)` . Функция строится самими участниками, но так, чтобы соответствовать следующим правилам:

1. Наибольший вклад в совместимость даёт наличие статуса "свободны". Те кто "всё сложно" идут во вторую очередь, а "занятые" в третью и последнюю (очень вероятно их вообще не будет в ответе).
2. Далее идёт совместимость по интересам. Чем больше совпавших интересов у пользователей, тем более они совместимы.
3. Третий по значению параметр - различие в возрасте. Чем больше разница, тем меньше совместимость.
4. Те, у кого активирован премиум-аккаунт, пропихиваются в самый верх, вперёд обычных пользователей. Если таких несколько, то они сортируются по совместимости между собой.
5. Если общих интересов нет, то стоит считать пользователей абсолютно несовместимыми с `compatibility = 0` .

В итоговом списке необходимо выводить только следующие поля: `id, email, status, fname, sname, birth, premium, interests` . Если в ответе оказались одинаково совместимые пользователи (одни и те же status, interests, birth), то выводить их по возрастанию `id`

20.12.2018: в этом запросе теперь не нужно выводить данные по **interests**. Сделано по той же причине, что и в запросах /filter/.

Пример запроса и корректного ответа на него:

```
GET: /accounts/89528/recommend/?country=Индция&limit=8&query_id=151
```

(вернуть 8 самых совместимых с пользователем id=89528 в стране "Индция")

```
{
    "accounts":  [
        {
            "email": "heernetletem@me.com",
            "premium": {"finish": 1546029018.0, "start": 1530304218},
            "status": "свободны",
            "sname": "Данашевен",
            "fname": "Анатолий",
            "id": 35473,
            "birth": 926357446
        },{
            "email": "teicfiwidadsuna@inbox.com",
            "premium": {"finish": 1565741391.0, "start": 1534205391},
            "status": "свободны",
            "id": 23067,
            "birth": 801100962
        },{
            "email": "nonihiwwahigtegodyn@inbox.com",
            "premium": {"finish": 1557069862.0, "start": 1525533862},
            "status": "свободны",
            "sname": "Стаметаный",
            "fname": "Виталий",
            "id": 90883,
            "birth": 773847481
        }
    ]
}
```


Особенность 8. Если в хранимых данных не существует пользователя с переданным id, то ожидается код 404 с пустым телом ответа.

4. Подбор по похожим симпатиям: **/accounts/<id>/suggest/**

   Этот тип запросов похож на предыдущий тем, что он тоже про поиск "вторых половинок". Аналогично пересылается id пользователя, для которого мы ищем вторую половинку и аналогично используется GET-параметр limit. Различия в реализации. Теперь мы ищем, кого лайкают пользователи того же пола с похожими "симпатиями" и предлагаем тех, кого они недавно лайкали сами. В случае, если в запросе передан GET-параметр country или city, то искать "похожие симпатии" нужно только в определённой локации.

   Похожесть симпатий определим как функцию: `similarity = f (me, account)`, которая вычисляется однозначно как сумма из дробей `1 / abs(my_like['ts'] - like['ts'])`, где my_like и like - это симпатии к одному и тому же пользователю. Для дроби, где `my_like['ts'] == like['ts']`, заменяем дробь на `1`. Если общих лайков нет, то стоит считать пользователей абсолютно непохожими с `similarity = 0`. Если у одного аккаунта есть несколько лайков на одного и того же пользователя с разными датами, то в формуле используется среднее арифметическое их дат.

В ответе возвращается список тех, кого ещё не лайкал пользователь с указанным id, но кого лайкали пользователи с самыми похожими симпатиями. Сортировка по убыванию похожести, а между лайками одного такого пользователя - по убыванию id лайка.

Пример запроса и корректного ответа на него:

```
GET: /accounts/51774/suggest/?country=Испляндия&limit=6&query_id=152
```


```
{
    "accounts": [
        {
             "email": "itwonudiahsu@yandex.ru",
             "id": 94155,
             "status": "заняты",
             "fname": "Никита"
        },{
             "email": "neeficyreddohypot@ymail.com",
             "id": 93449,
             "status": "свободны",
             "fname": "Иван"
        },{
             "email": "sotheralnes@inbox.ru",
             "id": 89997,
             "sname": "Лукетатин",
             "fname": "Руслан",
             "status": "заняты"
        },{
             "email": "kihatneselritunuwryr@ya.ru",
             "id": 88119,
             "sname": "Лукушутин",
             "fname": "Николай",
             "status": "свободны"
        },{
             "email": "otnideonfomedec@icloud.com",
             "id": 87873,
             "status": "свободны",
             "sname": "Фаетавен",
             "fname": "Сидор"
        },{
             "email": "poodreantasis@me.com",
             "id": 85461,
             "sname": "Даныкалан",
             "fname": "Вадим",
             "status": "заняты"
        },
    ]
}
```


Особенность 9. Если в хранимых данных не существует пользователя с переданным id, то ожидается код 404 с пустым телом ответа.

Запросы изменения данных (POST):

1. Добавление нового пользователя: **/accounts/new/**

   Данный запрос просто добавляет новую запись о пользователе в хранимые данные. Новые данные записаны в теле запроса в формате json. Предполагается, что решение само проконтролирует уникальность полей и типы данных.

   В ответе ожидается код 201 с пустым json-ом в теле ответа (`{}`), если создание нового пользователя прошло успешно. В случае некорректных типов данных или неизвестных ключей нужно вернуть код 400 с пустым телом.

   Пример запроса и корректного ответа на него:

```
POST: /accounts/new/
...
{
    "sname": "Хопетачан",
    "email": "orhograanenor@yahoo.com",
    "country": "Голция",
    "interests": [],
    "birth": 736598811,
    "id": 50000,
    "sex": "f",
    "likes": [
        {"ts": 1475619112, "id": 38753},
        {"ts": 1464366718, "id": 14893},
        {"ts": 1510257477, "id": 37967},
        {"ts": 1431722263, "id": 38933}
    ],
    "premium": {"start": 1519661251, "finish": 1522253251},
    "status": "всё сложно",
    "fname": "Полина",
    "joined": 1466035200
}
```

```
{}
```

   Особенность 10. В случае некорректных типов данных, наличия неизвестных ключей или нарушения уникальности нужно вернуть код 400 с пустым телом.

2. Обновление данных пользователя: **/accounts/<id>/**

   Данный запрос обновляет данные одного единственного пользователя в хранимых данных. В теле запроса в формате json записаны только обновляемые поля и их значения. Поле id никогда не содержится среди обновляемых полей и посылается в URL запроса. Предполагается, что решение само проконтролирует уникальность обновляемых полей и типы данных.

   В ответе ожидается код 202 с пустым json-ом в теле ответа (`{}`), если обновление прошло успешно. Если запись с указанным id не существует в имеющихся данных, то ожидается код 404 с пустым телом. Если запись существует, но в теле запроса переданы неизвестные поля или типы значений неверны, то ожидается код 400.

Пример запроса и корректного ответа на него:

```
POST: /accounts/46133/?query_id=308
...
{
    "birth": 664945551,
    "city": "Санктобирск",
    "email": "fywdolpa@yandex.ru",
    "status": "заняты",
    "country": "Алмаль"
}
```

```
{}
```

Особенность 11. Аналогично добавлению нового пользователя, в случае некорректных типов данных, наличия неизвестных ключей или нарушения уникальности нужно вернуть код 400 с пустым телом.

3. Добавление новых лайков: **/accounts/likes/**

Данный запрос добавляет множество новых лайков к множеству разных пользователей. Примерно так работают сервисы знакомств в жизни: клиентское приложение рекомендует подходящих кандидатов, а пользователи ставят отметку симпатии (вдохновлялись badoo). Никаких ограничений по уникальности в лайках нет.

В теле запроса передаётся структура `{"likes": [ ... ]}`, где в likes лежит массив объектов с такими ключами:

- **liker** - id того, кто выставил отметку симпатии;
- **likee** - id того, кто симпатичен;
- **ts** - время в формате timestamp, когда отметка была выставлена;

В ответе ожидается код 202 с пустым json-ом в теле ответа (`{}`), если обновление прошло успешно. Если в теле запроса переданы неизвестные поля или типы значений неверны, то ожидается код 400.

Пример запроса и корректного ответа на него:

```
POST: /accounts/likes/?query_id=316
...
{"likes":[
    {"likee": 3929, "ts": 1464869768, "liker": 25486},
    {"likee": 13239, "ts": 1431103000, "liker": 26727},
    {"likee": 2407, "ts": 1439604510, "liker": 6403},
    {"likee": 26677, "ts": 1454719940, "liker": 22248},
    {"likee": 22411, "ts": 1481309376, "liker": 32820},
    {"likee": 9747, "ts": 1431850118, "liker": 43794},
    {"likee": 43575, "ts": 1499496173, "liker": 16134},
    {"likee": 29725, "ts": 1479087147, "liker": 22248}
]}
```

```
{}
```

Особенность 12. Если в теле запроса переданы неизвестные поля или типы значений неверны, нужно вернуть код 400 с пустым телом.

Особенность 13. Для всех URL, не указанных в приведённом API ожидается ответ с кодом 404 и пустым телом.

Теперь, дорогой участник, когда ты ознакомился с правилами проведения Highload Cup 2018 и постановкой задачи, наступила пора пробовать и побеждать!
Мы - Лаборатория Технопарка и Mail.ru Group, от всей души **желаем тебе удачи!**