Документация к API

Общие данные блога - `GET /api/init`

Метод возвращает общую информацию о блоге: название блога и подзаголовок для размещения в хэдере сайта, а также номер телефона, e-mail и информацию об авторских правах для размещения в футере.

Авторизация: не требуется

Формат ответа:

```
{
  "title": "DevPub",
  "subtitle": "Рассказы разработчиков",
  "phone": "+7 903 666-44-55",
  "email": "mail@mail.ru",
  "copyright": "Дмитрий Сергеев",
  "copyrightFrom": "2005"
}
```

Получение настроек - `GET /api/settings`

Метод возвращает глобальные настройки блога из таблицы `global_settings`. Если значение параметра `YES` - возвращаем `true`, если `NO` - возвращаем `false`

Авторизация: не требуется

Формат ответа:

```
{
  "MULTIUSER_MODE": false,
  "POST_PREMODERATION": true,
  "STATISTICS_IS_PUBLIC": true
}
```

Статус авторизации - `GET /api/auth/check`

Метод возвращает информацию о текущем авторизованном пользователе, если он авторизован. Он должен проверять, сохранён ли идентификатор текущей сессии в списке авторизованных.

Значение `moderationCount` содержит количество постов необходимых для проверки модераторами. Считаются посты имеющие статус `NEW` и не провереры модератором.

Если пользователь не модератор возращать 0 в `moderationCount` и `false` в `moderation`

Возвращайте `result:false` пока у вас не реализована авторизация пользователей.

Авторизация: не требуется

Форматы ответов: ✅ В случае, если пользователь авторизован:

```
{
  "result": true,
  "user": {
    "id": 576,
    "name": "Дмитрий Петров",
    "photo": "/avatars/ab/cd/ef/52461.jpg",
    "email": "petrov@petroff.ru",
    "moderation": true,
    "moderationCount": 56,
    "settings": true
  }
}
```

❌ В случае если пользователь неавторизован:

```
{"result": false}
```

Список постов - `GET /api/post`

Метод получения постов со всей сопутствующей информацией для главной страницы и подразделов "Новые", "Самые обсуждаемые", "Лучшие" и "Старые". Метод выводит посты, отсортированные в соответствии с параметром `mode` *(см. ниже более детальное описание каждого значения этого параметра)*. Первый раз **frontend** запрашивает список постов, начиная с нулевого (`offset`), в количестве `limit`, далее - в зависимости от параметра `count`, который был возвращён сервером.

Параметр `count` всегда содержит общее количество постов, которое доступно по данному запросу с учётом всех фильтров, параметров доступности, кроме `offset` и `limit`. Один запрос с разными `offset` и `limit` должен выводить одно и то же значение `count`.

Массив `posts` - только то количество, которое было передано при запросе в параметре `limit`, или меньшее, если это последняя порция постов.

Должны выводиться только активные (поле `is_active` в таблице `posts` равно 1), утверждённые модератором (поле `moderation_status` равно `ACCEPTED`) посты с датой публикации не позднее текущего момента (движок должен позволять откладывать публикацию).

Авторизация: не требуется

Параметры:

- `offset` - сдвиг от 0 для постраничного вывода. Если в запросе не задан, принимаем 0 как значение по умолчанию

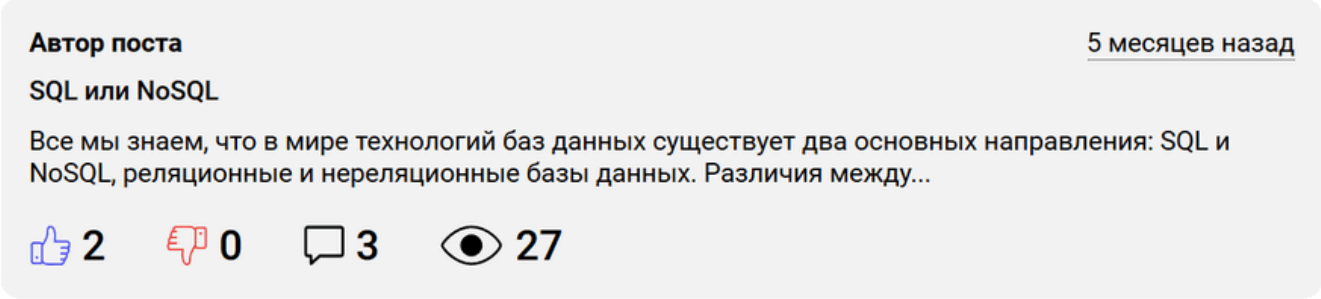
- `limit` - количество постов, которое надо вывести. Если в запросе не задан, принимаем 10 как значение по умолчанию.
- `mode` - режим вывода (сортировка):
  - `recent` - сортировать по дате публикации, выводить сначала новые (если mode не задан, использовать это значение по умолчанию)
  - `popular` - сортировать по убыванию количества комментариев (посты без комментариев выводить)
  - `best` - сортировать по убыванию количества лайков (посты без лайков и дизлайков выводить)
  - `early` - сортировать по дате публикации, выводить сначала старые

Формат ответа:

```
{
  "count": 390,
  "posts": [
    {
      "id": 345,
      "timestamp": 1592338706,
      "user":
        {
        "id": 88,
        "name": "Дмитрий Петров"
        },
      "title": "Заголовок поста",
      "announce": "Текст анонса поста без HTML-тэгов",
      "likeCount": 36,
      "dislikeCount": 3,
      "commentCount": 15,
      "viewCount": 55
    },
    {...}
  ]
}
```

`timestamp` - время в UTC (секунды)

`announce` - предпросмотр поста, длина не более 150 символов, все HTML теги должны быть удалены, в конце полученной строки добавить троеточие ...



если не найдено постов по запросу, то вернуть: Status code: 200

```
{
  "count":0,
  "posts":[]
}
```

Получение списка тэгов - `GET /api/tag`

Метод выдаёт список тэгов, начинающихся на строку, заданную в параметре `query`. В случае, если она не задана, выводятся все тэги. В параметре `weight` должен быть указан относительный нормированный вес тэга от 0 до 1, соответствующий частоте его встречаемости. Значение 1 означает, что этот тэг встречается чаще всего. Пример значений `weight` для разных частот встречаемости:

Тэг	Кол-во публикаций с данным тэгом	Общее кол-во публикаций на сайте	Вес	Нормированный вес
Java	18	20	0,90	1,00
Spring	10	20	0,50	0,56
Hibernate	4	20	0,20	0,22
Hadoop	3	20	0,15	0,17

Пример расчета для тега Hibernate

Получаете количество постов count и получаете количество постов с тегом Hibernate. Находим ненормированный вес dWeightHibernate:

```
dWeightHibernate = hibernate / weights = 4 / 20 = 0.20
```

Чтобы нормировать каждый вес, надо найти коэффициент k для нормализации. Для этого находим самый популярный тег, в нашем случае это Java, получаем его ненормализованный вес и делим 1 на вес

```
dWeightMax = java / weights = 18 / 20 = 0.90
```

```
k = 1 / dWeightMax = 1 / 0.90 = 1.11
```

Ненормализованный вес каждого тега надо умножить на k. Продолжая пример с Hibernate:

```
weightHibernate = hibernate * k = 0.20 * 1.11 = 0.22
```

Значение 0.22 возвращаем для тега Hibernate.

В таблице приведено количество активных публикаций, утверждённых модератором со временем публикации, не превышающем текущее время - то есть количество, соответствующее реальному количеству публикаций на сайте, которые видит любой неавторизованный посетитель сайта.

Авторизация: не требуется

Параметры:

- `query` - часть тэга или тэг (например "Java", "PHP"), может быть не задан или быть пустым (в этом случае выводятся все тэги).

Формат ответа:

```
{
  "tags":
  [
   {"name":"Java", "weight":1},
   {"name":"Spring", "weight":0.56},
   {"name":"Hibernate", "weight":0.22},
   {"name":"Hadoop", "weight":0.17},
  ]
}
```

Поиск постов - `GET /api/post/search`

Метод возвращает посты, соответствующие поисковому запросу - строке `query`. В случае, если запрос пустой или содержит только пробелы, метод должен выводить все посты (запрос `GET /api/post` с параметров `mode=recent`).

Параметр `count` содержит общее количество постов, которые найдены по переданному в параметре `query` запросу. Массив `posts` - только то количество, которое было передано при запросе в параметре `limit`, или меньшее, если это последняя порция постов.

Должны выводиться только активные (поле `is_active` в таблице `posts` равно 1), утверждённые модератором (поле `moderation_status` равно `ACCEPTED`) посты с датой публикации не позднее текущего момента (движок должен позволять откладывать публикацию).

Авторизация: не требуется

Параметры:

- `offset` - сдвиг от 0 для постраничного вывода
- `limit` - количество постов, которое надо вывести
- `query` - поисковый запрос

Формат ответа:

```
 {
   "count": 20,
   "posts": [
     {
       "id": 345,
       "timestamp": 1592338706,
       "user":
         {
```

```
          "id": 88,
          "name": "Дмитрий Петров"
          },
        "title": "Заголовок поста",
        "announce": "Текст анонса поста без HTML-тэгов",
        "likeCount": 36,
        "dislikeCount": 3,
        "commentCount": 15,
        "viewCount": 55
      },
      {...}
    ]
  }
```

`timestamp` - время в UTC

если не найдено постов по запросу, то вернуть: Status code: 200

```
{
  "count":0,
  "posts":[]
}
```

Календарь (количества публикаций) - `GET /api/calendar`

Метод выводит количества публикаций на каждую дату переданного в параметре `year` года или текущего года, если параметр `year` не задан. В параметре `years` всегда возвращается список всех годов, за которые была хотя бы одна публикация, в порядке возрастания.

Должны учитываться только активные посты (поле `is_active` в таблице `posts` равно 1), утверждённые модератором (поле `moderation_status` равно `ACCEPTED`) посты с датой публикации не позднее текущего момента.

Авторизация: не требуется

Параметры:

- `year` - год в виде четырёхзначного числа, если не передан - возвращать за текущий год

Формат ответа:

```
{
  "years": [2017, 2018, 2019, 2020],
  "posts": {
    "2019-12-17": 56,
    "2019-12-14": 11,
    "2019-06-17": 1,
    "2020-03-12": 6
```

```
    }
  }
```

Список постов за указанную дату - `GET /api/post/byDate`

Выводит посты за указанную дату, переданную в запросе в параметре `date`. Параметры `offset` и `limit`, а также выдаваемое значение `count` аналогичны таковым в методе `GET /api/post`, выдающем список постов.

Должны выводиться только активные (поле `is_active` в таблице `posts` равно 1), утверждённые модератором (поле `moderation_status` равно `ACCEPTED`) посты с датой публикации не позднее текущего момента (движок должен позволять откладывать публикацию).

Авторизация: не требуется

Параметры:

- `offset` - сдвиг от 0 для постраничного вывода
- `limit` - количество постов, которое надо вывести
- `date` - дата в формате "2019-10-15"

Формат ответа:

```
{
"count": 2,
"posts": [
  {
    "id": 345,
    "timestamp": 1592338706,
    "title": "Заголовок поста",
    "announce": "Текст анонса (часть основного текста) поста без HTML-тэгов",
    "likeCount": 36,
    "dislikeCount": 3,
    "commentCount": 15,
    "viewCount": 55,
    "user":
      {
        "id": 88,
        "name": "Дмитрий Петров"
      }
  },
  {...}
  ]
}
```

`timestamp` - время в UTC

если не найдено постов по запросу, то вернуть: Status code: 200

```
{
  "count":0,
  "posts":[]
}
```

Список постов по тэгу - `GET /api/post/byTag`

Метод выводит список постов, привязанных к тэгу, который был передан методу в качестве параметра `tag`.

Параметры `offset` и `limit`, а также выдаваемое значение `count` аналогичны таковым в методе `GET /api/post`, выдающем список постов.

Должны выводиться только активные (поле `is_active` в таблице `posts` равно 1), утверждённые модератором (поле `moderation_status` равно `ACCEPTED`) посты с датой публикации не позднее текущего момента (движок должен позволять откладывать публикацию).

Авторизация: не требуется

Параметры:

- `offset` - сдвиг от 0 для постраничного вывода
- `limit` - количество постов, которое надо вывести
- `tag` - тэг, по которому нужно вывести все посты

Формат ответа:

```
{
"count": 2,
"posts": [
  {
    "id": 345,
    "timestamp": 1592338706,
    "title": "Заголовок поста",
    "announce": "Текст анонса (часть основного текста) поста без HTML-тэгов",
    "likeCount": 36,
    "dislikeCount": 3,
    "commentCount": 15,
    "viewCount": 55,
    "user":
      {
        "id": 88,
        "name": "Дмитрий Петров"
      }
  },
  {...}
  ]
}
```

`timestamp` - время в UTC

если не найдено постов по запросу, то вернуть: Status code: 200

```
{
  "count":0,
  "posts":[]
}
```

Получение поста - `GET /api/post/{ID}`

Метод выводит данные конкретного поста для отображения на странице поста, в том числе, список комментариев и тэгов, привязанных к данному посту. Выводит пост в любом случае, если пост активен (параметр `is_active` в базе данных равен 1), принят модератором (параметр `moderation_status` равен `ACCEPTED`) и время его публикации (поле `timestamp`) равно текущему времени или меньше его формата UTC.

При успешном запросе необходимо увеличивать количество просмотров поста на 1 (поле `view_count`), кроме случаев:

- Если модератор авторизован, то не считаем его просмотры вообще
- Если автор авторизован, то не считаем просмотры своих же публикаций

Параметр `active` в ответе используется админ частью фронта, должно быть значение `true` если пост опубликован и `false` если скрыт (при этом модераторы и автор поста будет его видеть)

Авторизация: не требуется

Формат ответа:

```
{
 "id": 34,
 "timestamp": 1592338706,
 "active": true,
 "user":
 {
   "id": 88,
   "name": "Дмитрий Петров"
 },
 "title": "Заголовок поста",
 "text": "Текст поста в формате HTML",
 "likeCount": 36,
 "dislikeCount": 3,
 "viewCount": 55,
 "comments": [
   {
     "id": 776,
     "timestamp": 1592338706,
     "text": "Текст комментария в формате HTML",
     "user":
```

```
        {
          "id": 88,
          "name": "Дмитрий Петров",
          "photo": "/avatars/ab/cd/ef/52461.jpg"
        }
    },
    {...}
  ],
  "tags": ["Статьи", "Java"]
}
```

`timestamp` - время в UTC

Если пост не найден: вернуть 404 статус код (Документ на найден)

Запрос каптчи - `GET /api/auth/captcha`

Метод генерирует коды капчи, - отображаемый и секретный, - сохраняет их в базу данных (таблица `captcha_codes`) и возвращает секретный код `secret` (поле в базе данных`secret_code`) и изображение размером 100x35 с отображённым на ней основным кодом капчи `image` (поле базе данных `code`).

Также метод должен удалять устаревшие капчи из таблицы. Время устаревания должно быть задано в конфигурации приложения (по умолчанию, 1 час).

Авторизация: не требуется

Формат ответа:

```
{
  "secret": "car4y8cryaw84cr89awnrc",
  "image": " код_изображения_в_base64"
}
```

Уточнение работы каптчи:

При запросе `GET /api/auth/captcha`:

- бэк генерирует изображение `image` и без сохранения на диск конвертирует в строку формата base64, обязательно добавляя к результату заголовок `" "`
- генерирует уникальный идентификатор `secret` и сохраняет его в бд. По этому идентификтору в дальнейшем будет возможность найти в бд правильный текст каптчи.

При восстановлении пароля, регистрации и прочих запросов с captcha, после того как пользователя вводит данные каптчи, отправляется форма содержащая текст-расшифровка каптчи пользователем и `secret`. Сервис ищет по значению `secret` запись о каптче в таблице и сравнивает ввод пользователя со значением поля `code` таблицы `captcha_codes`. На основе сравнения решается - каптча введена верно или нет.

Вариант генерации captcha Cage - CAptcha GEnerator Java Library

Как конвертировать изображение в base64

Регистрация - `POST /api/auth/register`

Метод создаёт пользователя в базе данных, если введённые данные верны. Если данные неверные - пользователь не создаётся, а метод возвращает соответствующую ошибку.

Авторизация: не требуется

Запрос:

```
{
  "e_mail":"konstantin@mail.ru",
  "password":"123456",
  "name":"Константин",
  "captcha":"d34f",
  "captcha_secret":"69sdFd67df7Pd9d3"
}
```

- `e_mail` - e-mail пользователя
- `name` - имя пользователя
- `password` - пароль для аккаунта
- `captcha` - код капчи
- `captcha_secret` - секретный код капчи

Форматы ответов:

✅ **В случае, если все данные отправлены верно:**

```
  {"result": true}
```

❌ **В случае ошибок *(на frontend ошибку повтора пароля необходимо отображать без обращения к серверу)*:**

```
{
  "result": false,
  "errors": {
    "email": "Этот e-mail уже зарегистрирован",
    "name": "Имя указано неверно",
    "password": "Пароль короче 6-ти символов",
    "captcha": "Код с картинки введён неверно"
  }
}
```

Список постов на модерацию - `GET /api/post/moderation`

Метод выводит все посты, которые требуют модерационных действий (которые нужно утвердить или отклонить) или над которыми мною были совершены модерационные действия: которые я отклонил или утвердил (это определяется полями `moderation_status` и `moderator_id` в таблице `posts` базы данных).

Параметры `offset` и `limit`, а также выдаваемое значение `count` аналогичны таковым в методе `GET /api/post`, выдающем список постов.

Должны выводиться только активные (поле `is_active` в таблице `posts` равно 1) посты. Скрытые посты - это черновики, которые пользователь создал, но ещё не решил опубликовать.

❗**Авторизация:** требуется

Параметры:

- `offset` - сдвиг от 0 для постраничного вывода
- `limit` - количество постов, которое надо вывести
- `status` - статус модерации:
  - `new` - новые, необходима модерация
  - `declined` - отклонённые мной
  - `accepted` - утверждённые мной

Формат ответа:

```
{
"count": 3,
"posts": [
  {
    "id": 31,
    "timestamp": 1592338706,
    "title": "Заголовок поста",
    "announce": "Текст анонса (часть основного текста) поста без HTML-тэгов (HTML
тэги необходимо удалить из текста анонса)",
    "likeCount": 36,
    "dislikeCount": 3,
    "commentCount": 15,
    "viewCount": 55,
    "user":
      {
        "id": 88,
        "name": "Дмитрий Петров"
      }
  },
  {...}
  ]
}
```

`timestamp` - время в UTC

если не найдено постов по запросу, то вернуть: Status code: 200

```
{
  "count":0,
  "posts":[]
}
```

Список моих постов - `GET /api/post/my`

Метод выводит только те посты, которые создал я (в соответствии с полем `user_id` в таблице `posts` базы данных). Возможны 4 типа вывода (см. ниже описания значений параметра `status`).

Параметры `offset` и `limit`, а также выдаваемое значение `count` аналогичны таковым в методе `GET /api/post`, выдающем список постов.

❗**Авторизация:** требуется

Параметры:

- `offset` - сдвиг от 0 для постраничного вывода
- `limit` - количество постов, которое надо вывести
- `status` - статус модерации:
  - `inactive` - скрытые, ещё не опубликованы (`is_active` = 0)
  - `pending` - активные, ожидают утверждения модератором (`is_active` = 1, `moderation_status` = `NEW`)
  - `declined` - отклонённые по итогам модерации (`is_active` = 1, `moderation_status` = `DECLINED`)
  - `published` - опубликованные по итогам модерации (`is_active` = 1, `moderation_status` = `ACCEPTED`)

Формат ответа:

```
{
"count": 3,
"posts": [
  {
    "id": 31,
    "timestamp": 1592338706,
    "title": "Заголовок поста",
    "announce": "Текст анонса (часть основного текста) поста без HTML-тэгов (HTML
тэги необходимо удалить из текста анонса)",
    "likeCount": 36,
    "dislikeCount": 3,
    "commentCount": 15,
    "viewCount": 55,
    "user":
      {
        "id": 88,
        "name": "Дмитрий Петров"
      }
  },
```

```
  {...}
  ]
}
```

`timestamp` - время в UTC

если не найдено постов по запросу, то вернуть: Status code: 200

```
{
  "count":0,
  "posts":[]
}
```

Добавление поста - `POST /api/post`

Метод отправляет данные поста, которые пользователь ввёл в форму публикации. В случае, если заголовок или текст поста не установлены и/или слишком короткие (короче 3 и 50 символов соответственно), метод должен выводить ошибку и не добавлять пост.

Время публикации поста также должно проверяться: в случае, если время публикации раньше текущего времени, оно должно автоматически становиться текущим. Если позже текущего - необходимо устанавливать введенное значение.

Пост должен сохраняться со статусом модерации `NEW`.

❗**Авторизация:** требуется

Параметры запроса:

```
{
  "timestamp":1592338706,
  "active":1,
  "title":"заголовок",
  "tags":["java","spring"],
  "text":"Текст поста включащий <b>тэги форматирования</b>"
}
```

- `timestamp` - дата и время публикации в формате UTC
- `active` - 1 или 0, открыт пост или скрыт
- `title` - заголовок поста
- `text` - текст поста в формате HTML
- `tags` - тэги через запятую (при вводе на frontend тэг добавляется при нажатии Enter).

✅ **Формат успешного ответа:**

```
{"result": true}
```

❌ **Формат ответа в случае ошибки или нескольких ошибок:** Status code: 200

```
{
  "result": false,
  "errors": {
    "title": "Заголовок не установлен",
    "text": "Текст публикации слишком короткий"
  }
}
```

Редактирование поста - `PUT /api/post/{ID}`

Метод изменяет данные поста с идентификатором ID на те, которые пользователь ввёл в форму публикации. В случае, если заголовок или текст поста не установлены и/или слишком короткие (короче 3 и 50 символов соответственно), метод должен выводить ошибку и не изменять пост.

Время публикации поста также должно проверяться: в случае, если время публикации раньше текущего времени, оно должно автоматически становиться текущим. Если позже текущего - необходимо устанавливать указанное значение.

Пост должен сохраняться со статусом модерации `NEW`, если его изменил автор, и статус модерации не должен изменяться, если его изменил модератор.

❗**Авторизация:** требуется

Параметры запроса:

```
{
  "timestamp":1592338706,
  "active":1,
  "title":"заголовок",
  "tags":["java","spring"],
  "text":"Текст поста включащий <b>тэги форматирования</b>"
}
```

- `timestamp` - дата и время публикации в формате UTC
- `active` - 1 или 0, открыть пост или скрыть
- `title` - заголовок поста
- `text` - текст поста в формате HTML
- `tags` - тэги через запятую (при вводе на frontend тэг должен добавляться при нажатии Enter или вводе запятой).

✅ **Формат успешного ответа:**

```
{
  "result": true
```

```
}
```

❌ **Формат ответа в случае ошибки или нескольких ошибок:**

```
{
  "result": false,
  "errors": {
    "title": "Заголовок слишком короткий",
    "text": "Текст публикации слишком короткий"
  }
}
```

Загрузка изображений - `POST /api/image`

Метод загружает на сервер изображение в папку `upload` и три случайные подпапки. Имена подпапок и изображения можно формировать из случайного хэша, разделяя его на части.

Метод возвращает путь до изображения. Этот путь затем будет вставлен в HTML-код публикации, в которую вставлено изображение.

❗**Авторизация:** требуется

Запрос: `Content-Type: multipart/form-data`

- `image` - файл изображения

✅ **Формат успешного ответа:** Строка

```
/upload/ab/cd/ef/52461.jpg
```

❌ **Формат ответа в случае ошибки:** Status Code: 400, "Bad Request"

```
{
  "result": false,
  "errors": {
    "image": "Размер файла превышает допустимый размер"
  }
}
```

также возвращать ошибку, если отправлен файл не формата изображение jpg, png.

Отправка комментария к посту - `POST /api/comment`

Метод добавляет комментарий к посту. Должны проверяться все три параметра. Если параметры parent_id и/или post_id неверные (соответствующие комментарий и/или пост не существуют), должна

выдаваться ошибка 400 (см. ниже раздел “Обработка ошибок”). В случае, если текст комментария отсутствует (пустой) или слишком короткий, необходимо выдавать ошибку в JSON-формате.

❗**Авторизация:** требуется

Пример запроса на добавление комментария к самому посту:

```
{
  "parent_id":"",
  "post_id":21,
  "text":"привет, какой <span style="font-weight: bold;">интересный</span> <span
style="font-style: italic;">пост!</span>"
}
```

Пример запроса на добавление комментария к другому комментарию:

```
{
  "parent_id":31,
  "post_id":21,
  "text":"текст комментария"
}
```

- `parent_id` - ID комментария, на который пишется ответ
- `post_id` - ID поста, к которому пишется ответ
- `text` - текст комментария (формат HTML)

✅ **Формат успешного ответа:**

```
{
  "id": 345
}
```

❌ **Формат ответа в случае ошибки:** Status Code: 400, "Bad Request"

```
{
  "result": false,
  "errors": {
    "text": "Текст комментария не задан или слишком короткий"
  }
}
```

Модерация поста - `POST /api/moderation`

Метод фиксирует действие модератора по посту: его утверждение или отклонение. Кроме того, фиксируется `moderator_id` - идентификатор пользователя, который отмодерировал пост.

Посты могут модерировать только пользователи с `is_moderator = 1`

Авторизация: требуется

Запрос:

```
{
  "post_id":31,
  "decision":"accept"
}
```

- `post_id` - идентификатор поста
- `decision` - решение по посту: `accept` или `decline`.

Формат ответа:

```
{
  "result":true
}
```

если по какой-то причине изменение статуса поста не удалось изменить, то возвращать `false`

Вход - `POST /api/auth/login`

Метод проверяет введенные данные и производит авторизацию пользователя, если введенные данные верны. Если пользователь авторизован, идентификатор его сессии должен запоминаться в `Map<String, Integer>` со значением, равным `ID` пользователя, которому принадлежит данная сессия.

В параметрах объекта `user` выводятся имя пользователя, ссылка на его аватар, e-mail, параметры `moderation` (если равен true, то у пользователя есть права на модерацию и в выпадающем меню справа будет отображаться пункт меню Модерация с цифрой, указанной в параметре `moderationCount`) и `settings` (если равен `true`, то пользователю доступны настройки блога). Оба параметра - `moderation` и `settings` - должны быть равны `true`, если пользователь является модератором.

Значение `moderationCount` содержит количество постов необходимых для проверки модераторами. Считаются посты имеющие статус `NEW` и не проверерны модератором. Если пользователь не модератор возращать 0 в `moderationCount`.

Авторизация: не требуется

Запрос:

```
{
  "e_mail":"my@email.com",
  "password":"dHdf6dDHfd"
}
```

- `e_mail` - e-mail пользователя
- `password` - пароль пользователя

Форматы ответов: ✅ В случае успешной авторизации:

```
{
  "result": true,
  "user": {
    "id": 576,
    "name": "Дмитрий Петров",
    "photo": "/avatars/ab/cd/ef/52461.jpg",
    "email": "my@email.com",
    "moderation": true,
    "moderationCount": 0,
    "settings": true
  }
}
```

❌ Формат ответа в случае ошибки или нескольких ошибок:

```
{
  "result": false
}
```

сообщения о причине отказа в авторизации уточнять не требуется, пользователь увидит сообщение: Логин и/или пароль введен(ы) неверно

Редактирование моего профиля - `POST /api/profile/my`

Метод обрабатывает информацию, введённую пользователем в форму редактирования своего профиля. Если пароль не введён, его не нужно изменять. Если введён, должна проверяться его корректность: достаточная длина. Одинаковость паролей, введённых в двух полях, проверяется на frontend - на сервере проверка не требуется.

❗**Авторизация:** требуется

Запрос без изменения пароля и фотографии: `Content-Type: application/json`

```
{
  "name":"Sendel",
```

```
  "email":"sndl@mail.ru"
}
```

Запрос с изменением пароля и без изменения фотографии: `Content-Type: application/json`

```
{
  "name":"Sendel",
  "email":"sndl@mail.ru",
  "password":"123456"
}
```

Запрос с изменением пароля и фотографии: `Content-Type: multipart/form-data;`

```
{
  "photo": <binary_file>,
  "name":"Sendel",
  "email":"sndl@mail.ru",
  "password":"123456",
  "removePhoto":0
}
```

⚠ при загрузке файла изображения фотографии пользователя, необходимо выполнять обрезку и изменение размера фотографии до 36x36 пикселей.

Запрос изменение фотографии и изменение данных, без смены пароля: `Content-Type: multipart/form-data;`

```
{
  "photo": <binary_file>,
  "name":"Sendel",
  "email":"sndl@mail.ru",
  "removePhoto":0
}
```

Запрос на удаление фотографии без изменения пароля: `Content-Type: application/json`

```
{
  "name":"Sendel",
  "email":"sndl@mail.ru",
  "removePhoto":1,
  "photo": ""
}
```

- `photo` - файл с фото или пустое значение (если его требуется удалить)

- `removePhoto` - параметр, который указывает на то, что фотографию нужно удалить (если значение равно 1)
- `name` - новое имя
- `email` - новый e-mail
- `password` - новый пароль

Форматы ответов:

✅ **В случае, если все данные отправлены верно:**

```
{"result": true}
```

❌ **В случае ошибок *(на frontend ошибку повтора пароля необходимо отображать без обращения к серверу)*:**

```
{
 "result": false,
 "errors": {
   "email": "Этот e-mail уже зарегистрирован",
   "photo": "Фото слишком большое, нужно не более 5 Мб",
   "name": "Имя указано неверно",
   "password": "Пароль короче 6-ти символов",
  }
}
```

Восстановление пароля - `POST /api/auth/restore`

Метод проверяет наличие в базе пользователя с указанным e-mail. Если пользователь найден, ему должно отправляться письмо со ссылкой на восстановление пароля следующего вида - `/login/change-password/HASH`, где `HASH` - сгенерированный код вида `b55ca6ea6cb103c6384cfa366b7ce0bdcac092be26bc0` (код должен генерироваться случайным образом и сохраняться в базе данных в поле `users.code`).

Авторизация: не требуется

Запрос:

```
{
  "email":"petrov@petroff.ru"
}
```

- `email` - e-mail пользователя

Форматы ответов: ✅ **В случае, если логин найден и ссылка восстановления отправлена:**

```
{
  "result": true
}
```

❌ В случае, если логин не найден:

```
{
  "result": false
}
```

Изменение пароля - `POST /api/auth/password`

Метод проверяет корректность кода восстановления пароля (параметр `code`) и корректность кодов капчи: введённый код (параметр `captcha`) должен совпадать со значением в поле `code` таблицы `captcha_codes`, соответствующем пришедшему значению секретного кода (параметр `captcha_secret` и поле `secret_code` в таблице базы данных).

Авторизация: не требуется

Запрос:

```
{
  "code":"b55ca6ea6cb103c6384cfa366b7ce0bdcac092be26bc0",
  "password":"123456",
  "captcha":"3166",
  "captcha_secret":"eqKIqurpZs"
}
```

- `code` - код восстановления пароля
- `password` - новый пароль
- `captcha` - код капчи
- `captcha_secret` - секретный код капчи

Форматы ответов:

✅ В случае, если все данные отправлены верно:

```
{"result": true}
```

❌ В случае ошибок *(на frontend ошибку повтора пароля необходимо отображать без обращения к серверу)*:

```
{
  "result": false,
  "errors": {
    "code": "Ссылка для восстановления пароля устарела.
        <a href=
        \"/auth/restore\">Запросить ссылку снова</a>",
    "password": "Пароль короче 6-ти символов",
    "captcha": "Код с картинки введён неверно"
  }
}
```

Моя статистика - `GET /api/statistics/my`

Метод возвращает статистику постов текущего авторизованного пользователя: общие количества параметров для всех публикаций, у который он является автором и доступные для чтения.

❗**Авторизация:** требуется

Формат ответа:

```
{
"postsCount":7,
"likesCount":15,
"dislikesCount":2,
"viewsCount":58,
"firstPublication":1590217200
}
```

`firstPublication` - Время в формате UTC

Статистика по всему блогу - `GET /api/statistics/all`

Метод выдаёт статистику по всем постам блога. В случае, если публичный показ статистики блога запрещён (см. соответствующий параметр в `global_settings`) и текущий пользователь не модератор, должна выдаваться ошибка 401 **(см. ниже Обработка ошибок)**.

Авторизация: не требуется

Формат ответа:

```
{
"postsCount":7,
"likesCount":15,
"dislikesCount":2,
"viewsCount":58,
"firstPublication":1590217200
}
```

`firstPublication` - Время в формате UTC

Лайк поста - `POST /api/post/like`

Метод сохраняет в таблицу `post_votes` лайк текущего авторизованного пользователя. В случае повторного лайка возвращает `{result: false}`.

Если до этого этот же пользователь поставил на этот же пост дизлайк, этот дизлайк должен быть заменен на лайк в базе данных.

❗**Авторизация:** требуется

Запрос:

```
{
  "post_id": 151
}
```

- `post_id` - id поста которому ставим лайк

✅ **Формат ответа в случае, если лайк произошёл:**

```
{
  "result": true
}
```

❌ **Формат ответа в случае, если лайк не произошёл:**

```
{
  "result": false
}
```

Дизлайк поста - `POST /api/post/dislike`

Метод сохраняет в таблицу `post_votes` дизлайк текущего авторизованного пользователя. В случае повторного дизлайка возвращает `{result: false}`.

Если до этого этот же пользователь поставил на этот же пост лайк, этот лайк должен заменен на дизлайк в базе данных.

❗**Авторизация:** требуется

Запрос:

```
{"post_id": 151}
```

- `post_id` - id поста

✅ **Формат ответа в случае, если дизлайк произошёл:**

```
{"result": true}
```

❌ **Формат ответа в случае, если дизлайк не произошёл:**

```
{"result": false}
```

Выход - `GET /api/auth/logout`

Метод разлогинивает пользователя: удаляет идентификатор его сессии из списка авторизованных. Всегда возвращает `true`, даже если идентификатор текущей сессии не найден в списке авторизованных.

❗**Авторизация:** требуется

Формат ответа:

```
{"result": true}
```

Сохранение настроек - `PUT /api/settings`

Метод записывает глобальные настройки блога в таблицу `global_settings`, если запрашивающий пользователь авторизован и является модератором.

❗**Авторизация:** требуется

Запрос:

```
{
  "MULTIUSER_MODE": false,
  "POST_PREMODERATION": true,
  "STATISTICS_IS_PUBLIC": false
}
```

Глобальные настройки блога

MULTIUSER_MODE — если включен этот режим, в блоге разрешена регистрация новых пользователей. Если режим выключен, регистрация пользователей не доступна, на фронте на месте ссылки на страницу регистрации появляется текст `Регистрация закрыта`. При запросе на `/api/auth/register` необходимо возвращать статус 404 (NOT FOUND).

POST_PREMODERATION - если включен этот режим, то все новые посты пользователей с `moderation = false` обязательно должны попадать на модерацию, у постов при создании должен быть установлен `moderation_status = NEW`. Если значения `POST_PREMODERATION = false` (режим премодерации выключен), то все новые посты должны сразу публиковаться (если у них установлен параметр `active = 1`), у постов при создании должен быть установлен `moderation_status = ACCEPTED`.

STATISTICS_IS_PUBLIC - если включен этот режим, статистика блога должна быть доступна по запросу `GET /api/statistics/all` для всех групп пользователей. Если режим выключен, по запросу `GET /api/statistics/all` только модераторам отдавать данные статистики. Пользователям и гостям блога необходимо возвращать статус 401.

Статус коды ответов

- **Если пользователь не авторизован** при запросе требующего авторизации, возвращайте код 401
- **Если при запросе данные (пост, комментарий и прочего) не найдены** - 404
- **При возврате ответа включающий {result:false}** возвращайте код 200

Обработка статус кодов фронтендом

- **Страница не существует** - пустой ответ с кодом **404 (Not found)** На frontend должна показываться заставка "Запрошенная вами страница была скрыта или не существует"
- **Пользователь не авторизован** - пустой ответ с кодом **401 (Unauthorized)** На frontend будет происходить переадресация на главную страницу в неавторизованном состоянии
- **Неверный параметр на входе** - ответ с кодом **400 (Bad request)** и сообщением (объект с ключом "message") На frontend будет выводиться alert с текстом сообщения, переданным в ответе сервера в параметре "message"
- **Ошибка на сервере** - пустой ответ с кодом **500 (Internal Server Error)** На frontend будет выводиться alert с текстом: "Произошла ошибка! Пожалуйста, попробуйте позже или обратитесь к администратору"