

REST: ресурсы, эндпоинты и HTTP-методы

В базе данных проекта может храниться множество разнородной информации, и задача API — обеспечить доступ к этой информации. Ключевая абстракция в REST — это ресурс. Любая информация, которая может быть названа, может быть ресурсом: пост в социальной сети, коллекция постов, подборка актуальных новостей, пользователь сайта, коллекция любых объектов или других ресурсов.

Унифицированный указатель ресурса, или URL (от англ. *Uniform Resource Locator*), используют для указания, где находится тот или иной ресурс.

| | |
|----------------------------------|--|
| <code>/users</code> | # Ресурсом может быть коллекция сущностей |
| <code>/users/12</code> | # Пользователь с <code>id = 12</code> — это тоже ресурс |
| <code>/latest-news</code> | # Ресурс не обязательно должен быть статическим: # новости каждый день разные, но ресурс — один, постоянный |
| <code>/users/12/playlists</code> | # Все плейлисты пользователя с <code>id=12</code> — это ресурс, # содержащий коллекцию ресурсов |

В терминах REST URL-адрес, идентифицирующий ресурс, принято называть **эндпоинтом** (от англ. *endpoint*, «конечная точка»).

Правила именования ресурсов

Для обращения к ресурсам через HTTP-запросы их необходимо как-то именовать. Чем понятнее имена ресурсов, тем проще разобраться в API. Здорово, когда по названию ресурса можно понять, что именно он содержит.

Чтобы с вашим API было просто и удобно общаться, соблюдайте простые правила.

Ресурсы — существительные

Почти всегда ресурсы именуют существительными во множественном числе:

| |
|-----------------------------|
| <code>/users</code> |
| <code>/api/starships</code> |

Перед вами API «Звёздных войн». Попробуйте догадаться, какой список объектов вернёт GET-запрос на <https://swapi.dev/api/starships>.

- ☒ Список космических кораблей
- ☐ Список планет
- ☐ Список межпланетных барашков

1 из 3 правильно и 0 неправильно

Иногда для именования ресурсов применяют существительные в единственном числе:

| |
|---------------------------------------|
| <code>/users/{user-id}/profile</code> |
| <code>/users/me</code> |

Иногда можно выбрать глагол в качестве имени. Но такое имя бывает удачным крайне редко:

| | |
|---|---------------------|
| <code>/users/{user-id}/cart/checkout</code> | # Проверка корзины |
| <code>/refresh</code> | # Обновление токена |

Слеш для иерархии

Слеш в URL используется для указания иерархии ресурсов по принципу «от общего к частному»:

| |
|-------------------------------------|
| <code>/users/{user-id}/posts</code> |
|-------------------------------------|

Все пользователи → Конкретный пользователь по ID → Все его посты

А чтобы обратиться к конкретному посту, нужно указать его ID:

| |
|---|
| <code>/users/{user-id}/posts/{post-id}</code> |
|---|

Все пользователи → Конкретный пользователь по ID → Все его посты → Конкретный пост по ID

«Висячий» слеш

В разных API встречаются два способа описания URL ресурсов:

| |
|---|
| # Первый вариант |
| <code>/users/{user-id}/posts/</code> # Слеш в конце URL |
| # Второй вариант |
| <code>/users/{user-id}/posts</code> # Нет слеша в конце URL |

Бытует мнение, что лучше избегать такого «висячего» слеша в конце URL — ведь он не добавляет информации.

Другое мнение заключается в том, что висячий слеш обязателен для любого ресурса, который может содержать дочерние элементы.

В REST нет однозначных рекомендаций, какой из подходов предпочтительнее. Придерживайтесь документации тех API, с которыми будете работать.

Дефисы вместо пробелов

В URL не должно быть пробелов, их заменяют дефисами или подчёркиваниями. Лучше применять дефисы:

- на некоторых устройствах подчёркивание может выйти за базовую линию строки, и его будет не видно вовсе;
- несколько подчёркиваний сливаются в одно.

| |
|--|
| # Делайте так: |
| <code>/users/{user-id}/user-devices</code> |
| # А не так: |
| <code>/users/{user-id}/user_devices</code> |

HTTP-методы

Любой API предназначен для получения доступа к ресурсам. К ресурсу всегда можно обратиться по URL.

HTTP-метод запроса определяет, что следует сделать:

- GET** получает ресурсы;
- POST** создаёт ресурс;
- PUT** заменяет существующий ресурс целиком;
- PATCH** частично изменяет существующий ресурс;
- DELETE** удаляет ресурс.

Реже применяют ещё два метода:

- HEAD** получает только заголовок ответа. HEAD похож на GET, но в ответе на этот запрос есть только заголовок, а тела ответа нет.
- OPTIONS** получает перечень HTTP-методов, которые поддерживает сервер.

GET — это «получить», DELETE — это «удалить»

Сам по себе метод не определяет логику работы сервера. Можно запрограммировать API так, что при DELETE-запросе будет создаваться ресурс, а при GET — редактироваться.

Не следует так делать, если на то нет особо важных причин.

| |
|--|
| # Получить список пользователей |
| GET .../users |
| # Создать пользователя |
| POST .../users |
| # Удалить пользователя с <code>id = 2</code> |
| DELETE .../users/2 |

Не стоит включать в название ресурса имя HTTP-метода:

| |
|-------------------|
| # Так не надо |
| GET /get-users |
| POST /create-user |

API должен понимать по HTTP-методу, какие действия от него требуются; дублировать указания в имени ресурса не нужно.

Идемпотентность и безопасность методов

У HTTP-методов есть две важные характеристики — безопасность и идемпотентность.

Безопасность: метод, который не изменяет ресурс, называется безопасным. Например, если в качестве ресурса рассматривать никнеймы пользователей, то сколько ни отправляй GET-запрос на получение никнейма конкретного пользователя — к изменению никнейма это не приведёт. Метод GET — безопасный.

Метод, который может изменить ресурс, в терминах архитектуры REST считается небезопасным. Такими методами могут быть PUT, PATCH, DELETE или POST.

Идемпотентность (от лат. *idem* — «тот же самый» и *potens* — «способный») — это свойство, заключающееся в том, что многократное выполнение этого метода по результату равно однократному. То есть, выполняя один и тот же запрос много-много раз, мы всегда будем получать один и тот же результат. Даже если сто раз отправить GET-запрос на получение никнейма конкретного пользователя, ответ не изменится, он будет таким же, как если бы его отправили всего лишь один раз. Метод GET — идемпотентный.

А вот метод POST работает иначе: обычно это создание новой записи, например добавление никнейма, независимо от того, есть уже аналогичный никнейм или нет. Если десять раз послать один и тот же POST-запрос, то будет добавлено десять идентичных никнеймов. Метод POST — неидемпотентный.

Выберите верные утверждения.

☒ Ресурсы следует именовать как существительные во множественном числе. Верно, таквы традиции.

☒ HTTP-методы должны соответствовать действиям, которые требуется выполнить с ресурсами.

Верно, из названия HTTP-метода становится понятно, какое действие производится над ресурсом.

☐ Методы PUT и DELETE идемпотентны. Верно, но есть нюанс с кодами ответа: DELETE после первого успешного запроса вернёт 200 (OK), а потом будет возвращать 204 (No Content), потому что элемент уже удалён.

☐ Безопасный метод — метод, в котором передаются данные о том, кто выполняет запрос.

☒ Эффект от многократного выполнения идемпотентного метода равен эффекту от его однократного выполнения.

Верно. Сто раз запроси на swapi.dev данные о космическом корабле с `id=9` — сто раз получишь один и тот же ответ: `Death Star, model D5-1 Orbital Battle Station...`

☒ Безопасный метод никогда не изменяет данные, а только запрашивает. Верно, поэтому он так и называется. От вопроса ничего не сломается.

☒ GET — безопасный метод. Да, GET-запросы безопасны: их цель — получить информацию, а не изменить данные.

5 из 7 правильно и 0 неправильно