**Вспоминаем, что такое API**

[На втором спринте](https://praktikum.yandex.ru/trainer/web/lesson/79c8e41f-4337-419d-85f5-d6ca95509654/task/559afb5d-4b67-47fe-9d3a-38e5ef3be580/) вы узнали, что API — это Application Programming Interface (англ. «программный интерфейс приложения»). Освежим это в памяти.

Интерфейс — средство взаимодействия с системой. Например, руль в автомобиле — интерфейс управления, посредник между человеком и электронной системой. С помощью руля система машины понимает, куда нужно повернуть колёса.

API — тоже интерфейс, благодаря которому разработчик может взаимодействовать с системой и не задумываться о её устройстве. Для этого создатели API предоставляют «инструкцию» — набор специальных команд (методов).

API есть в браузере: DOM — это API взаимодействия с элементами веб-страницы. API может быть и у сервера — вспомните API проекта Mesto, к которому вы обращались в последнем спринте курса JavaScript. Как и любой интерфейс, серверный API — набор инструментов для выполнения определённых действий: записи файлов на диск, обращения к базе данных, расчётов.

В этом спринте вам предстоит много работать с запросами к серверу, а мы подробнее расскажем, как сервер общается с фронтендом.

# API сайтов раньше и сейчас

Раньше сайты работали так: сервер отдавал клиенту готовые HTML-страницы. У этого подхода есть два важных недостатка:

* HTML-страница нужна только браузеру. Но если у сервиса есть мобильное приложение, оно будет общаться с тем же сервером. Мобильному приложению не нужен HTML-код — данные внутри него организованы иначе, поэтому придётся делать отдельный API.
* Поскольку HTML-код генерируется на сервере, при переходе от страницы к странице браузер перезагружает сайт целиком. Это неэффективно, поскольку приходится перерисовывать одинаковые элементы: шапку, меню, подвал. Разумнее получить данные об изменяющихся частях сайта и отрисовывать только их.

Из-за этих недостатков придумали другой подход. С сервера возвращают только данные, а клиент сам разбирается, как их отрисовать. Мобильное приложение будет использовать свои методы отрисовки, браузер — свои. Так мы ограничиваемся одним API для разных платформ.

Кроме того, подход позволяет разрабатывать SPA (англ. single page application — «одностраничные приложения»). SPA — это сайты, которые загружают единственный HTML-файл. Все последующие переходы выполняются из JS: он запрашивает на сервере нужные данные и точечно перерисовывает часть страницы. Знакомый вам «Реакт» позволяет собирать именно такие приложения.

Многие современные API возвращают только данные, а за представление данных отвечает клиент. Один из популярных методов для создания таких API — REST. В следующем уроке расскажем, что это такое.

# Что такое REST?

REST, или Representational State Transfer (англ. «передача состояния представления»), — набор принципов, которых рекомендовано придерживаться при создании API. Если API сделан по этим принципам, его называют REST API.

Раньше клиент и сервер были тесно связаны. Например, сервер собирал веб-страницу и отдавал клиенту. Серверу нужно было знать, какие технологии работают на стороне клиента, чтобы тот смог открыть вернувшуюся веб-страницу.

Принципы REST позволили отделить клиента от сервера. Благодаря этому:

* стало проще переносить веб-приложение на другие платформы;
* появилась возможность делать открытые API;
* разрабатывать и тестировать серверное ПО стало проще и быстрее.

Рассмотрим принципы REST, которые важно понимать для программирования сервера.

## Принципы REST

### 1. Клиент-сервер

Сервер и клиент отвечают за разные вещи. Ответственность клиента — пользовательский интерфейс, а ответственность сервера — данные. Если API возвращает HTML-страницу, его нельзя назвать REST API: ведь при этом сервер берёт на себя ответственность за интерфейс.

В REST API сервер обычно возвращает данные в формате JSON. Именно этот принцип делает возможным точечную отрисовку страницы и существование единого API для браузера и мобильного приложения.

### 2. Отсутствие состояния (англ. "Stateless")

Запрос клиента к серверу должен содержать всю информацию, необходимую для обработки этого запроса. В проекте Mesto вы отправляли на сервер токен, чтобы получить карточки. Токен сообщал системе, что вы — это вы: запрос содержал информацию о том, кто запрашивает данные и какие.

Раньше серверы программировали иначе. При аутентификации сервер создавал сессию, то есть запоминал, что пользователь совершил вход. Сейчас такой подход применяется редко.

При отправке токена создавать сессию не нужно. Сервер не хранит информацию о состоянии пользователя, поэтому принцип и называется «отсутствие состояния».

### 3. Единый интерфейс (англ. "Uniform Interface")

Интерфейс обращения к серверу не зависит от клиента. Он одинаковый для всех.

Скопировать код

GET https://nomoreparties.co/cards/5d1f0611d321eb4bdcd707dd

Так, запрос к карточке может быть сформирован из браузера, мобильного приложения и с умного чайника — для всех интерфейс един.

### 4. Многоуровневость

Первый принцип гласит, что в коммуникации участвуют двое: клиент и сервер. Но, благодаря многоуровневости, мы всё равно можем строить более сложные системы, не нарушая этого принципа.

API сервиса Яндекс.Такси может использовать API Яндекс.Навигатора. Вы как клиент взаимодействуете только с API Яндекс.Такси, а он, в свою очередь, является клиентом навигатора.

### 5. Кешируемость

Данные ответа могут быть закешированы. Это значит, мы можем сохранить данные на клиенте и при идентичном запросе взять их из кеша — памяти клиента. Чуть подробнее про кеширование мы поговорим в теме об Express — серверном фреймворке на Node.js.

### 6. Код по запросу (англ. "Code on demand")

Это необязательный принцип. Он подразумевает, что функциональность клиента может быть расширена кодом, который приходит с сервера. Сейчас такое можно встретить повсеместно: мы получаем с сервера JS-файлы и исполняем их в браузере. Но принципы формулировались в 2000 году — тогда с сервера код возвращали редко, потому разработчики REST API и выделили это в отдельный пункт.

## Зачем использовать REST API?

REST API помогает разработчику делать пользовательский опыт лучше. Например, при переходе страницы отрисовываются моментально, а пользователь не видит этот раздражающий белый экран или значок загрузки. А ещё REST API — настоящее спасение при нестабильном подключении сети. Теперь вашим приложением смогут пользоваться даже там, куда интернет, казалось бы, провели недавно.

В REST API есть некоторые нововведения, например авторизация через JWT, а не сессии. Но про это мы расскажем чуть позже. Кроме этого, вы узнаете, как правильно работать с REST API с фронтенда.

## Задание

Запросы и информацию о них можно посмотреть из инструментов разработчика. Для этого откройте браузер, а в нём — инструменты разработчика. Перейдите во вкладку Network, если вы используете Yandex.Browser, Chrome или Firefox. Затем перейдите по ссылке:

Скопировать код

https://nomoreparties.co/tests/1

1. Какой у ответа код статуса?



501



304

Верно. Этот статус приходит, когда данные «закешированы», то есть сохранены для быстрого доступа на сервере.



404

404 ошибка сообщает, что такой страницы не существует и ошибка произошла на стороне клиента.



200

1. Какое значение заголовка Content-Type?



application/json

Этот заголовок указывает, что пришёл json файл без указания кодировки.



text/html



text/plain; charset=utf-8



application/json; charset=utf-8

Верно. Этот заголовок указывает, что пришёл json с указанием кодировки.

1. Какое значение свойства answer в теле ответа?



У вас всё получится

«У вас всё получится», а мы и не сомневаемся!



Node.js — это сложно



Волга впадает в Каспийское море

# HTTP-методы

Любой API нужен, чтобы получить доступ к ресурсам сервера. Ресурс — единица информации, с которой работает API. В проекте Mesto такая единица — карточка с фотографией. Мы можем получить карточку, а её часть — фотографию или число лайков — нет.

К ресурсу всегда можно обратиться по URL:

Скопировать кодBASH

GET https://nomoreparties.co/cards/5d1f0611d321eb4bdcd707dd

Что делать с ресурсом, определяет HTTP-метод запроса. В REST применяются 5 основных методов:

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

Реже применяют ещё 2 метода:

* HEAD позволяет получить только заголовки ответа. HEAD похож на GET, но у его ответа нет тела;
* OPTIONS позволяет узнать, какие HTTP-методы поддерживает сервер.

Сам по себе метод не определяет логику работы сервера. Можно закодить API так, чтобы POST удалял ресурс, а DELETE — добавлял. Но такой подход противоречил бы идее REST о том, что методы должны применяться по назначению.

# Обработка методов на сервере

REST API может быть написан на любом языке, который умеет обрабатывать запросы, с использованием разных фреймворков. При этом алгоритм обработки методов всегда одинаковый:

* выделяется эндпоинт-ссылка, за которой закреплена функция-обработчик;
* происходит обращение к эндпоинт-ссылке;
* срабатывает обработчик;
* обработчик возвращает определённый результат.

Какой метод подойдёт для получения веб-страницы с сервера?



OPTIONS



GET

Верно, именно этот метод нужен для получения данных с сервера.



POST



PATCH

Какой метод пригодится для создания нового пользователя?



GET



POST

Этот метод обычно используется для совершения какого-то действия с передачей данных на сервер. Например для создания новой записи в БД. Он отлично подойдёт!



PUT



PATCH

Какой метод отвечает за обновление информации о пользователе?



GET



PATCH

Верно, этот метод подойдёт для обновления информации о пользователе.



PUT

Этот метод используется, чтобы заменить существующую запись на сервере на новую.



POST

# Правила именования ресурсов

Концепцию REST API предложил Рой Филдинг в 2000 году. В его диссертации понятие «ресурс» определяется так:

Ключевая абстракция информации в REST — ресурс. Ресурсом может быть документ, изображение, пост в блоге, пользователь социальной сети — всё, к чему можно обратиться при HTTP-запросе. Ресурс — концептуальное отображение какого-то объекта или набора объектов, а не сами эти объекты.

Для взаимодействия с ресурсами их необходимо как-то именовать. Чем понятнее имена, тем проще разобраться в API. Здорово, когда по названию ресурса можно понять, чтó он содержит.

В этом уроке расскажем о правилах именования ресурсов в REST.

### Ресурсы — существительные

Почти всегда ресурсы именуют существительными во множественном числе:

Скопировать код

https://nomoreparties.co/users

https://nomoreparties.co/cards

Реже применяют единственное число:

Скопировать код

https://nomoreparties.co/users/{user-id}/profile

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

Скопировать код

https://nomoreparties.co/users/{user-id}/cart/checkout

### Слеш для иерархии

Слеш в URL используется для указания иерархии:

Скопировать код

GET /users/{user-id}/posts

### «Висящий» слеш — зло

Висящий слеш в конце URL не добавляет информации, поэтому лучше обойтись без него:

Скопировать код

GET /users/{user-id}/posts/

// Не надо так

### Дефисы вместо пробелов

Поскольку пробелов в URL быть не может, их заменяют либо дефисами, либо нижними подчёркиваниями. Дефисы лучше, потому что:

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

Скопировать код

// делайте так:

GET /users/{user-id}/user-devices

// а не так:

GET /users/{user-id}/user\_devices

### Только строчные

URL чувствительны к регистру. Чтобы потом не думать, в каком регистре назван ресурс, лучше использовать строчные символы:

Скопировать код

// это разные URL:

/users/{user-id}/posts

/users/{user-id}/Posts

### Не дублировать в именовании HTTP-метод

Для указания действия, которое вы совершаете над ресурсом, нужно использовать HTTP-методы:

Скопировать код

// получить пользователей

GET /users

// создать пользователя

POST /users

Но к ним не стоит добавлять действия, потому что в названиях методов оно уже указано. Поэтому следующие конструкции в названии метода дублируют значение метода:

Скопировать код

GET /get-users

POST /create-user

GET-запрос сам по себе обозначает действие, как и POST. К тому же, один ресурс может отвечать как на POST, так и на GET-запросы.

Вы пишете приложение, которое во время карантина и пропусков помогает людям выйти на улицу и... выгулять чужую собаку. Приложение должно получать все данные обо всех собаках из нашей базы. Запрос к какому эндпоинту подойдёт для этого лучше всего?



GET /users



GET /dogs

GET /dogs — вот они! Все пушистики будут тут.



GET /dogs/1



POST /dogs

Нам нужно добавить информацию о новой собаке на сервер, какой запрос лучше использовать для этого?



PUT /dogs/1



POST /dogs

Верно! С помощью этого запроса можно создать нового дружочка!



POST /dogs/1

POST /dogs/1 — подойдёт для создания чего-то нового у конкретной собаки. Например, новой миски для корма.



PUT /dogs

Новая собака создана и её id — 42. Каким запросом можно добавить информацию о ней?



PATCH /dogs



PATCH /dogs/42

Верно! Запросом PATCH /dogs/42 можно, например, добавить собаке в личные вещи полотенце для лап.



PUT /dogs/42



PUT /dogs

Junior-backend разработчик задался вопросом, как ему назвать эндпоинт для получения всех заводчиков собак из базы данных. Что посоветуете?



GET /get-owners



GET /owners

GET /owners — очень по REST-у!



GET /owner



GET /owners/all

# Статус ответа сервера

Ответ от сервера всегда содержит статус. Правильно выставленный статус делает API более понятным. В этом уроке разберём основные статусы ответов сервера.

Статусы делят на 5 категорий. Запросы каждой из них начинаются с разных цифр:

* **1xx: Informational.** Информационный ответ. Они приходят только техническим системам, вам их отправлять не придётся.
* **2xx: Success.** Запрос прошёл успешно.
* **3xx Redirection.** Запрос не завершён, и клиенту нужно предпринять какие-то действия, чтобы запрос прошёл успешно.
* **4xx: Client Error.** Ошибка на стороне клиента: запрос сформирован неверно или у клиента нет нужных прав доступа.
* **5xx: Server Error.** Ошибка на стороне сервера: что-то сломалось или сервер перегружен.

# Самые частые статусы

**200 OK.** Запрос прошёл успешно. Ответ с таким статусом должен содержать тело. Чаще всего этот статус используют при ответе на GET-запрос ресурса.

**201 Created.** Ресурс был создан на сервере. Подходящий ответ, например, при создании нового поста в блоге.

**202 Accepted.** Сервер начал работу по удовлетворению запроса, но ещё не закончил. Этот статус применяют для ответа на запросы, обработка которых занимает много времени, например, при обработке большого объёма данных.

**301 Moved Permanently.** API переделали, а ресурс переместили в другое место. Новый URL указывают в заголовке Location ответа сервера.

**302 Found.** Запрос должен быть перенаправлен на другой URL. В заголовке Location сервер должен отправить новый URL. При получении запроса браузер автоматически отправит его на новый URL.

**400 Bad Request.** Ошибка на стороне клиента. Например, запрос был неправильно сформирован. Это общий статус: его отправляют в случае, когда ни один другой 4xx статус не подходит.

**401 Unauthorized.** Запрос требует авторизации, но соответствующие авторизационные заголовки отсутствуют или сформированы неправильно.

**403 Forbidden.** Запрос сформирован правильно, но у клиента недостаточно прав, чтобы запрос прошёл успешно. Например, клиент пытается удалить чужой пост.

**404 Not Found.** Ресурс не найден. Например, пользователя с запрошенным id не существует.

**405 Method Not Allowed.** Ресурс запроса не поддерживает HTTP-метод, которым сделали запрос.

**500 Internal Server Error.** Общий статус для ошибок на стороне сервера. Это не ошибка клиента.

**501 Not Implemented**. Ресурс есть на сервере, но способ обращения к нему пока не реализован.

С этими статусами вам предстоит работать чаще всего. [О других статусах можно почитать дополнительно](https://restfulapi.net/http-status-codes/).

Какой статус должен отправить сервер, если сломался?



404



500

Верно!



301



201

Какой статус должен приходить при успешном создании пользователя?



200



201

Верно, этот статус сообщает нам: «создано».



302



404

Какой статус сообщает пользователю, что сервер не может приготовить кофе, потому что он чайник? Подсмотрите значение некоторых статусов в поисковике.



404



418

418 — «я чайник!» Пожалуй, подходит.



425

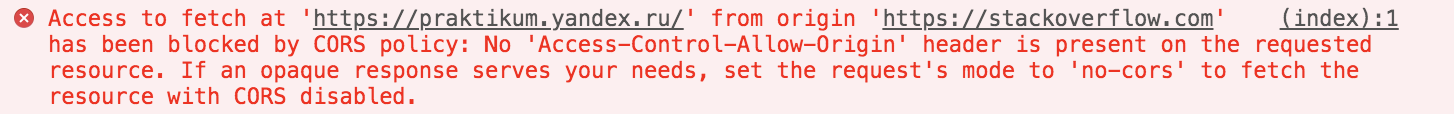
425 — значит, слишком рано. Возможно, слишком рано и чайник ещё не на работе. Но скорее всего сервер решил, что с клиента запросы приходят слишком быстро и он не успевает их обрабатывать, защищаясь от потенциальных атак.



451

# Защита от злоумышленников: CORS

В современных браузерах есть ограничение: по умолчанию вы не можете отправить запрос из фронтенда одного сайта к другому. Если вы находитесь на сайте https://one-site.com, не получится запросить что-то методом fetch у сайта https://another-site.com. Браузер выдаст ошибку:



Это браузерное ограничение было введено из соображений безопасности. Вместе с запросом браузер отправляет серверу дополнительные данные — «куки» (англ. cookie). Куки — строка, которая может быть использована сервером для авторизации на сайте.

Если бы ограничений на отправку запросов между сайтами не было, злоумышленники смогли бы использовать куки в своих злодейских целях. Например, создать сайт со специальным скриптом: пользователи заходили бы на него, а их интернет-банкам уходил бы запрос на перевод денег мошенникам. Так в интернете воцарились бы анархия и хаос.

Поэтому по умолчанию кросс-доменные запросы запрещены. Но поведение по умолчанию можно изменить. Это необходимо при создании открытых API — таких как Kanye API, которым вы уже пользовались. К открытому API должна быть возможность обратиться с любого домена, и браузер не должен это блокировать.

# Разрешение кросс-доменных запросов

Браузер не может самостоятельно решить, можно ли отправлять кросс-доменные запросы. Для этого ему нужно разрешение от сервера. Чтобы его получить, есть специальный механизм предзапросов (англ. "preflight request"). Сначала браузер посылает предварительный запрос к серверу и, если получает разрешение, отправляет основной запрос на нужный домен.

Если кросс-доменные запросы нужно разрешить, мы должны запрограммировать сервер так, чтобы он давал браузерам разрешение отправлять такие запросы. Это делают с помощью установки специальных заголовков ответа.

# Как это решается на сервере

Когда совершается запрос, клиент и сервер обмениваются заголовками, в которых прописаны настройки и разрешения для различных действий. За доступ к данным на сервере отвечает заголовок Access-Control-Allow-Origin. В публичных API, например kayne.rest, заголовок выглядит так:

Скопировать код

Access-Control-Allow-Origin: \*

Это позволяет получать доступ к данным с любого сайта, так как звездочка убирает ограничение по конкретным сайтам, с которых можно делать запрос.

В реальном проекте, где данные должны быть защищены, заголовок будет выглядеть примерно так:

Скопировать код

Access-Control-Allow-Origin: https://my-site.com

Это значит, что сервер разрешает запросы только с конкретного сайта.

Фронтенд-разработчики сталкиваются со сложностями с CORS в начале разработки или при работе с API, где требуется авторизация. Теперь вы знаете, кто виноват. А что делать — разберём в теме по серверной разработке, — там научимся правильно настраивать эти заголовки.

# Какие запросы обрабатывать? Методы и заголовки

В заголовке Access-Control-Allow-Methods указывают допустимые методы запросов. В Access-Control-Allow-Headers — какие заголовки могут в нём быть.

Скопировать код

Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,POST,DELETE

Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept

Полный список заголовков, которые можно разрешить, довольно длинный. [Ознакомиться с ними можно в дополнительных материалах](https://praktikum.yandex.ru/trainer/web/lesson/e13bfde8-aace-46b4-9935-e1d71c52efa8).

# Заключение

REST — один из подходов к созданию API. Он сильно повлиял на взаимодействие серверной и клиентской частей приложения: стало проще переносить функциональность на новые платформы и расширять серверные мощности, не затрагивая клиента.

Также благодаря REST веб-приложения лучше кооперируются. За последние 10 лет число открытых API сильно увеличилось. Когда веб-приложению нужна какая-то функциональность, можно найти подходящее API. Чтобы отправлять смс пользователям, мы просто подключим проект [к сервису Twilio](https://www.twilio.com/), и нам не нужно разбираться в протоколах, на которых работают смс.

Скорость разработки продуктов сильно возросла. Простота и «общительность» серверов стала повсеместной во многом благодаря REST.

Понимание принципов REST и создания API помогает экономить время — вы точно знаете, какими методами нужно обращаться к эндпоинтам сервера.

# Дополнительные материалы

# Открытые API

Актуальный и постоянно пополняющийся список публичных API. Загляните в него, когда захочется отточить навык взаимодействия с API или сделать нечто странное. Например, API из списка позволяют реализовать систему рекомендаций аниме на основе погоды за окном.

<https://github.com/public-apis/public-apis>

### CORS

Статья о CORS. Также в ней описаны заголовки, которые можно разрешить во входящих кросс-доменных запросах.

<https://developer.mozilla.org/ru/docs/Web/HTTP/CORS>