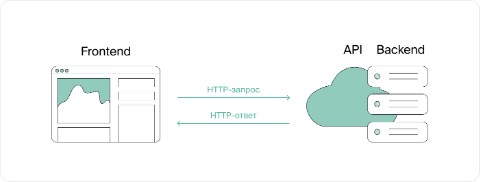
**Что такое API**

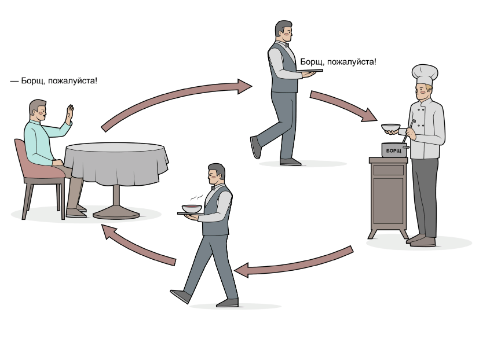
**API** (Application Programming Interface — программный интерфейс приложения) — интерфейс, который помогает приложениям взаимодействовать. Приложения тоже могут отправлять команды, и другие приложения будут их выполнять.

Вспомни клиент-серверную архитектуру: фронтенд общается с бэкендом, отправляя запросы и получая ответы. Чтобы сделать запрос к бэкенду, фронтенд обращается к API бэкенда.

Здесь API бэкенда — это фиксированный набор URL-адресов. По ним можно передавать данные в определённом формате.



Представь, что ты хочешь заказать борщ в ресторане. Тебе не нужно самостоятельно идти на кухню: можно просто передать свой «запрос» через официанта. Официант здесь — как API, которое приносит готовый «ответ» посетителю — фронтенду от кухни — бэкенда.

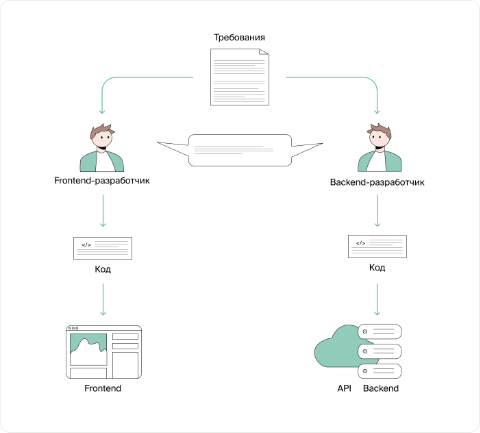


В вымышленном сервисе фронтенд шлёт GET-запрос к API бэкенда, чтобы найти пользователя по email:

[/api/search?email=john@email.com](mailto:/api/search?email=john@email.com)

Бэкенд его обрабатывает: ищет пользователя в базе данных и отдаёт их фронтенду. У сервера можно не только запросить информацию о пользователе, но и создать его в базе данных; а также обновить или удалить существующего. Набор таких адресов, который помогает управлять данными о пользователе — пример API.

Откуда От куда у бэкенда API?

API как часть бэкенда «пишет» разработчик. Обычно он делает это по спецификации или техническому заданию приложения.  
 

Например, в приложении хотят использовать данные сервиса Яндекс.Погода.

Как это сделать?

Обратиться к API Яндекс.Погоды. Данные Яндекс.Погоды не лежат в открытом виде — к тому же это объёмный массив информации: и история наблюдений, и старые прогнозы, и погодные предупреждения. Чтобы получить к ним доступ, сервис Я.Погоды сделал API: если обратиться к нему, можно получить запрошенный набор данных, чтобы применять его в своём приложении.

Если нужно встроить в приложение возможность писать комментарии и отправлять их сразу в Facebook, можно подключиться к Facebook API.

**Архитектурные решения — REST**

Архитектуру приложения можно выстраивать по-разному. Самые распространённые способы — по SOAP и REST.

REST — один из способов спроектировать приложение. Его главная особенность — простота отображения данных. В других архитектурных стилях данные «обёрнуты» в дополнительный слой разметки. В REST дополнительного слоя нет: по запросам и ответам сразу понятно, что происходит с данными.

Например, метод POST создаёт данные, DELETE — удаляет.

**Сервер не хранит информацию о клиенте.** Запрос клиента несёт ту информацию, которая нужна, чтобы сервер отправил ответ.

**Многослойная структура.** В архитектуре есть компоненты, которые обмениваются данными, — клиент и сервер; кроме них есть ещё и компоненты, которые распределяют потоки данных.

**Единообразные компоненты.** Каждый компонент архитектуры в REST построен по чётким правилам. К любому компоненту можно обратиться с запросом по определённому URL. В запросе будет вся информация, чтобы компонент смог обработать его корректно.

**Кэширование.** Когда сервер отправляет ответ, у него остаётся информация, какие данные нужно кэшировать, а какие нет.

От архитектурного стиля API зависит, как компоненты архитектуры обмениваются сообщениями: по какому протоколу и в каком формате. От этого зависит, как именно и какими инструментами ты будешь тестировать приложение.

Например, тебе нужно протестировать API. Ты знаешь, что оно построено по REST. Значит, оно принимает запросы по HTTP в форматах JSON или XML. Эта информация помогает определиться с инструментами: скорее всего, здесь лучше тестировать через Postman.

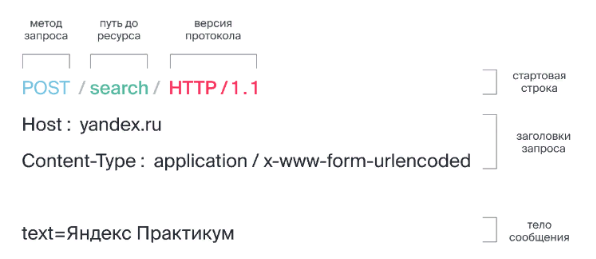
# HTTP: структура запросов и ответов

как работает HTTP. Это протокол, который позволяет клиенту и серверу обмениваться сообщениями.

Клиент отправляет серверу **HTTP-запрос,** а сервер возвращает **HTTP-ответ**.

HTTP-запрос (Request) состоит из трёх блоков:

* **стартовая строка** содержит метод запроса, URL и версию протокола;
* **заголовки** передают дополнительную информацию. Например, доменное имя ресурса или длину контента в байтах;
* **тело сообщения** содержит данные, которые передаёт клиент. Например — текст, который пользователь ввёл в форме регистрации.

 ****

 Если отправить такой запрос, откроется поиск по тексту «Яндекс Практикум».

# 

**Метод** сообщает серверу, какое действие нужно выполнить. Самые распространённые методы:

* GET — запрашивает данные;
* POST — создаёт или получает данные;
* DELETE — удаляет данные;
* PUT — обновляет данные.

**URL** — адрес, по которому клиент запрашивает данные, а сервер передаёт.

**код состояния, текст состояния** и **тело сообщения.**

Из чего состоит HTTP-ответ (Response):

* **строка состояния** содержит версию протокола, код состояния и текст состояния;

* **заголовки** передают дополнительную информацию: например, дату и время создания ответа;
* **тело** содержит набор данных, которые клиент получает в ответ на свой запрос. Например, информацию о всех зарегистрированных в приложении пользователях.

****

**Код состояния** (Status Code) сообщает, успешно ли сервер обработал запрос. Например, код 200 означает, что всё прошло успешно. А код 500 указывает на внутреннюю ошибку сервера.

**Текст состояния** — текст сообщения кода состояния от сервера. Например, ошибка 500 — «внутренняя ошибка сервера».

Вспомни, какие есть коды состояния:

****

# Postman: основная функциональность

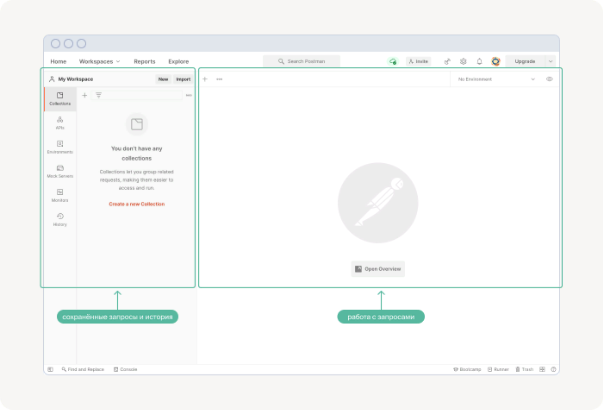
 Postman — инструмент для тестирования API. Он помогает отправлять HTTP- и HTTPS-запросы в API и получать ответы.

Как в Postman создавать запрос, добавлять параметры, работать с заголовками и телом запроса.

Но сначала Postman нужно скачать и установить: <https://www.getpostman.com/downloads/>.

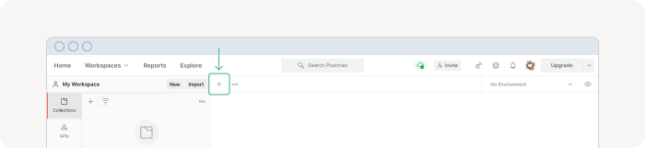
Запустить приложение. Перед тобой главный экран, который делится на две части:

* слева — список сохранённых запросов и история изменений;
* справа — область работы с запросами. Тут можно создавать запросы и настраивать их.



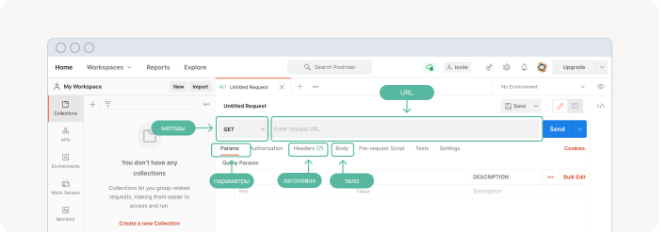
Интерфейс может немного отличаться в зависимости от версии, но функциональность остаётся та же.

Чтобы создать запрос, нажать на кнопку в виде плюса.



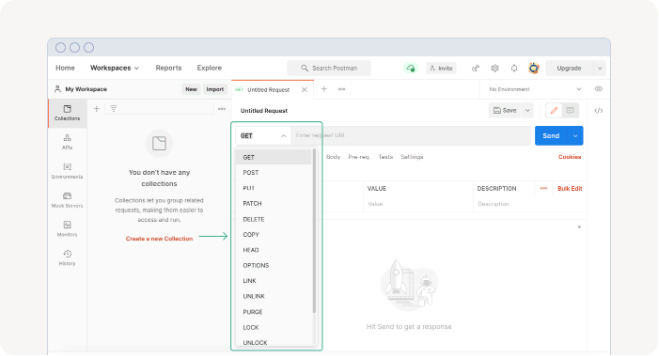
Откроется окно работы с запросом. В нём можно:

* выбрать метод запроса;
* добавить и отредактировать URL, по которому нужно отправить запрос;
* задать параметры запроса;
* добавить запросу заголовки;
* составить тело запроса.



### Как выбрать метод запроса

Посмотри на адресную строку. Найди выпадающий список слева от неё: по умолчанию в нём указан метод GET.



Если нажать на выпадающий список, увидишь все методы HTTP-запросов. В работе ты чаще всего будешь использовать GET, POST, PUT, DELETE.

Не забывай проверять, какой метод указываешь. По умолчанию новые запросы создаются с GET, и можно забыть поменять его на нужный метод.

Если указать неправильный метод, вернётся ошибка.

### Как настроить параметры

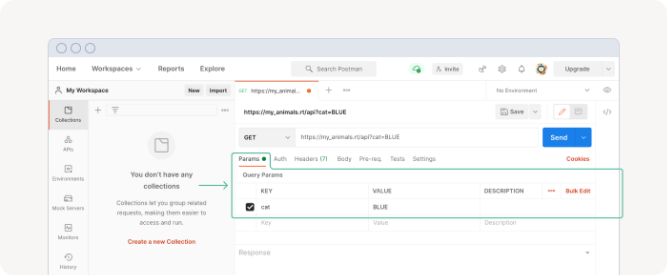
Ты уже знаешь, что в URL можно указывать параметры запроса. Они позволяют уточнить, какие именно данные нужно передать.

Параметры указывают в виде ключ=значение. Они перечисляются в URL после знака ? и разделяются знаком &, если параметров несколько. Например, в запросе кофта.ру?тип=свитшот&цвет=белый ключи — «тип» и «цвет», значения — «свитшот» и «белый».

Чтобы добавлять и редактировать параметры, используй вкладку Params.

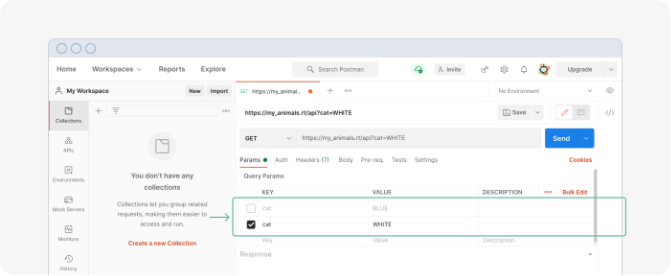
Укажи KEY (ключ) и VALUE (значение) параметра. Например, нужно задать цвет кота. Тогда KEY — cat, VALUE — BLUE.

Параметры автоматически подставятся в URL. Задай параметр cat=BLUE и посмотри на URL: он появился в запросе после вопросительного знака.

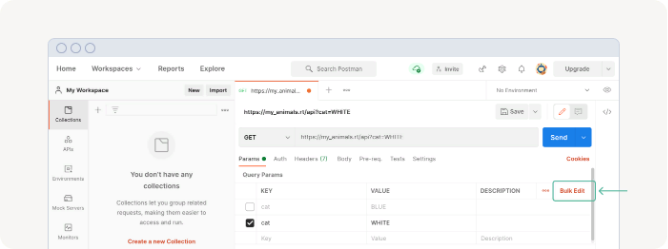


Напротив каждого параметра стоит чек-бокс. Если поставить там галочку, параметр добавится в URL. Если отключить — исчезнет.

Например, добавлено два параметра: цвет кота — голубой и белый. Голубой отключён — его нет в URL. Белый активный — он стоит в URL после вопросительного знака.

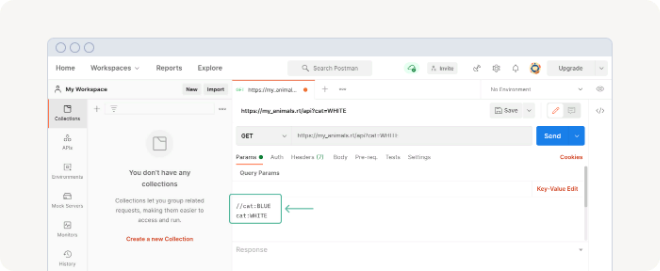


Есть ещё один способ добавить параметр — с помощью кнопки Bulk Edit («Массовое редактирование»). Она находится во вкладке Query Params — справа.



Этот режим напоминает блокнот: нужно записать все параметры в столбик в формате key:VALUE. Например, так: cat:BLUE.

Если нужно отключить один из параметров — достаточно поставить перед ним два слеша (//key:VALUE):



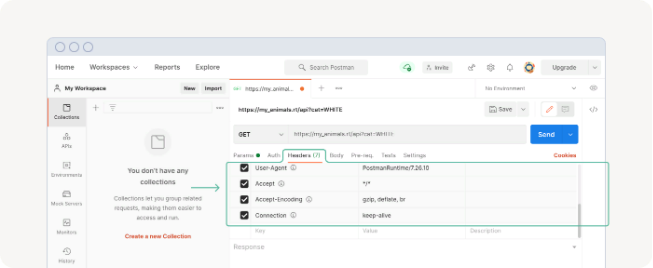
Режимы ничем не отличаются. Используй тот, который тебе удобнее.

### Как задать заголовок запроса

Ты уже знаешь из предыдущих уроков, что заголовки — это дополнительная информация от клиента серверу. Например, доменное имя ресурса или длина контента в байтах.

В Postman заголовки можно задавать вручную — но некоторые из них проставляются автоматически. Когда ты будешь создавать запросы в следующих уроках, Postman сам подберёт для них заголовки. Например, host — для имени домена или content-type — для формата данных.

Ты можешь посмотреть заголовки на вкладке Headers, если у тебя установлен Postman версии выше 7.36.



Заголовки можно отключать и включать — с помощью чекбокса. Это пригодится, чтобы проверить, как ведёт себя API, если какого-то заголовка нет.

Подробнее про заголовки — [в документации к Postman](https://learning.postman.com/docs/getting-started/introduction/).

### Как задать тело запроса

Как ты уже знаешь, в теле указывают данные, которые нужно отправить в запросе — чаще всего для создания и редактирования объекта. Например, такое тело описывает объект «кот» с именем Барсик красного цвета:

{

    "animal\_type": "сat",

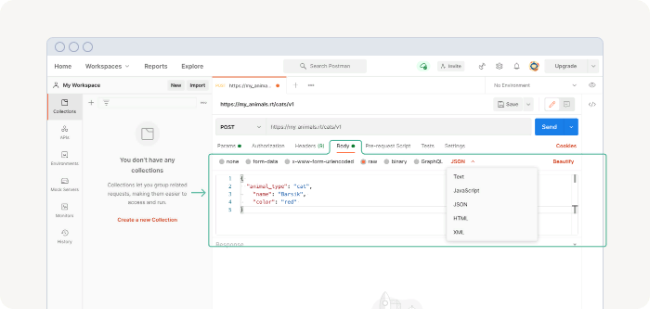
    "name": "Barsik",

    "color": "red"

}

Для создания кота будет использоваться метод POST.

Тело задаётся во вкладке Body. Здесь нужно выбрать, в каком формате его отправить на сервер: none, form-data и другие.



Пока что тебе пригодится только формат raw: с ним ты будешь работать в следующих уроках. Он означает, что данные в запросе будут переданы так, как ты их и напишешь. Внутри raw можно выбрать:

  Text,

* JavaScript,
* JSON,
* HTML,
* XML.

  Указывай тот формат, который умеет читать сервер. Это можно узнать из документации к приложению или у разработчика. В следующих уроках ты будешь работать с JSON и XML.

Обязательно убедись, что выбран правильный формат данных. В raw по умолчанию указан Text. Если сервер ожидает JSON, а запрос придёт в другом формате, вернётся ошибка.

Итак, удалось указать:

* метод запроса;
* URL, куда отправлять запрос;
* тело и параметры — если они нужны.

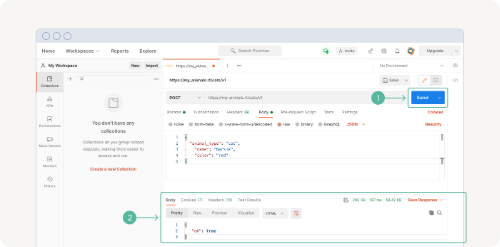
Теперь можно отправить запрос.

### Как отправить запрос

Нужно нажать кнопку Send.

После этого от сервера придёт ответ. Он появится в нижней части экрана на вкладке Body.

На картинке URL написан для примера: его вводить не нужно. С рабочим URL ты попрактикуешься дальше.



 На вкладке Body можно настроить, как именно ты хочешь прочитать ответ:

* Pretty — позволяет посмотреть тело ответа так, чтобы было удобно читать: с подсветкой элементов и отступами;
* Raw — ответ будет в «сыром» виде: без отступов и подсветки элементов;
* Preview — показывает результат: например, если ответ — страница сайта, можно посмотреть, как она выглядит.
* Visualize — позволяет преобразовать данные в график или таблицу.

Также здесь можно увидеть код ответа, куки, заголовки и другую полезную информацию. Можно сохранить ответ на компьютер.

# Задание https//: petstore.swagger.io/v2

# Добавить информацию о продаваемой собаке по кличке “ Layma” на сайт магазина.

# Request: POST <https://petstore.swagger.io/v2/pet>

**Body:** {

  "id": 160,

  "category": {

    "id": 2,

    "name": "Dogs"

  },

  "name": "Layma",

  "photoUrls": [

    "string"

  ],

  "tags": [

    {

      "id": 2,

      "name": "string"

    }

  ],

  "status": "available"

}

**Response**: Status: 200 Ok

{

    "id": 160,

    "category": {

        "id": 2,

        "name": "Dogs"

    },

    "name": "Layma",

    "photoUrls": [

        "string"

    ],

    "tags": [

        {

            "id": 2,

            "name": "string"

        }

    ],

    "status": "available"

}

**Найти питомца по ID=160**

# Request: GET <https://petstore.swagger.io/v2/pet/160>

**Response**: Status: 200 Ok

{

    "id": 160,

    "category": {

        "id": 2,

        "name": "Dogs"

    },

    "name": "Layma",

    "photoUrls": [

        "string"

    ],

    "tags": [

        {

            "id": 2,

            "name": "string"

        }

    ],

    "status": "available"

}

**Изменить информацию о питомце с ID=160**

# Request: PUT <https://petstore.swagger.io/v2/pet>

**Response**: Status: 200 Ok

{

    "id": 160,

    "category": {

        "id": 2,

        "name": "Rex"

    },

    "name": "doggie",

    "photoUrls": [

        "string"

    ],

    "tags": [

        {

            "id": 2,

            "name": "string"

        }

    ],

    "status": "available"

}

**Удалить информацию о питомце с ID=160**

# Request: DELETE <https://petstore.swagger.io/v2/pet/160>

# Body

{

  "id": 160,

  "category": {

    "id": 2,

    "name": "Rex"

  },

  "name": "doggie",

  "photoUrls": [

    "string"

  ],

  "tags": [

    {

      "id": 2,

      "name": "string"

    }

  ],

  "status": "available"

}

**Response**: Status: 200 Ok

{

    "code": 200,

    "type": "unknown",

    "message": "160"

}

# Проверить удаление питомца с ID=160

# Request:GET <https://petstore.swagger.io/v2/pet/160>

# Response: Status:404 Not Found

{

    "code": 1,

    "type": "error",

    "message": "Pet not found"

}