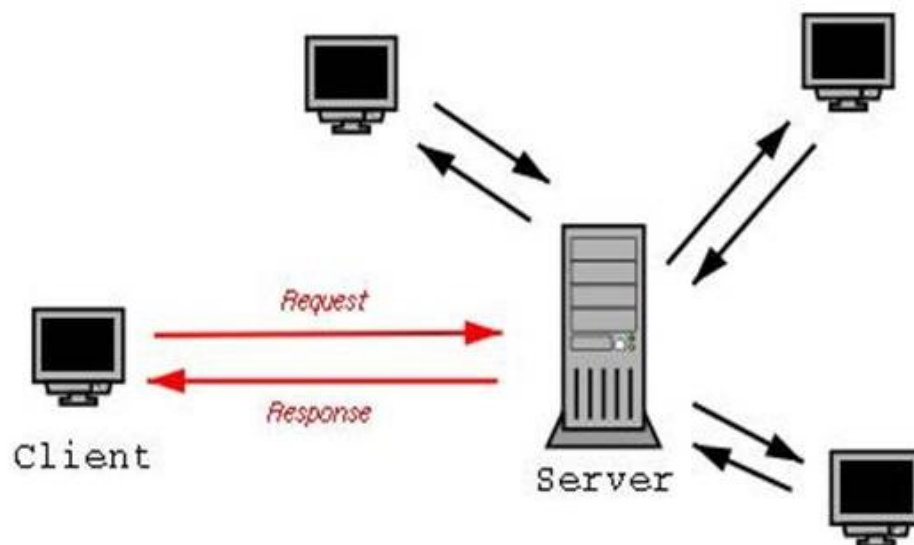


# **Основные понятия тестирования API**

# Клиент-серверная архитектура

- Многие приложения строятся по этому принципу
- Есть две взаимодействующие стороны – **клиент** и **сервер**, которые взаимодействуют друг с другом по сети при помощи **протокола**, например, HTTP



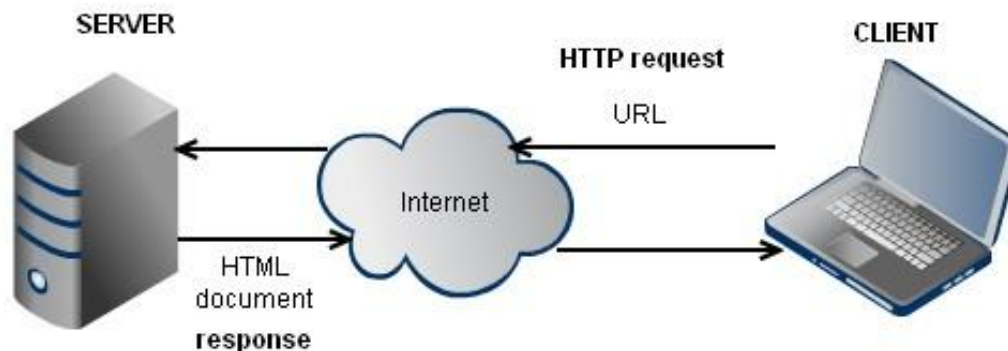
Клиентской программой является браузер

# Клиент (браузер)

- Иницирует взаимодействие с сервером
- Отправляет запросы, получает ответы
- Отображает **HTML** страницы
- Применяет к ним стили **CSS**
- Исполняет для страницы код на языке **JavaScript**
- Имеет ограничения:
  - не работает с файловой системой
  - хранит малые объемы данных
  - не влияет на другие приложения

# Сервер

- Представляет из себя серверную часть приложения (например, Apache, nginx, MSSQL сервер)
- Пассивен, «слушает» запросы от клиентов
- Хранит данные (БД)
- Обрабатывает данные (логика)
- Отдает данные в ответ на запросы



# HTTP протокол

- **HTTP (HyperText Transfer Protocol** — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных
- Изначально использовался для передачи гипертекстовых документов в формате HTML
- В настоящий момент используется для передачи произвольных данных

# Пример HTTP запроса

- **Пример запроса от браузера:**
- GET / HTTP/1.1      // метод, адрес и версия HTTP  
Host: ya.ru            // заголовки – пары ключ-значение  
Connection: keep-alive  
// тело запроса (тут его нет)  
// если тело есть, то оно отделяется одной пустой строкой
- **Метод** – это команда протокола HTTP. Есть методы GET, POST, PUT, DELETE и другие
- **Адрес** – идет относительно host
- <https://habrahabr.ru/post/215117/>

# Пример HTTP ответа

- **Пример ответа от браузера:**
- HTTP/1.1 200 OK // версия HTTP, код ответа  
Content-Type: text/html; charset=UTF-8  
Connection: keep-alive  
Cache-Control: no-cache,no-store,max-age=0,must-revalidate  
Content-Length: 11369  
  
<!DOCTYPE html><html>...</html> // тело ответа  
// в данном случае – HTML для страницы

# Коды HTTP ответов

- **1xx** – информационные
- **2xx** – успешное выполнение
- **3xx** – редирект (перееадресация)
- **4xx** – ошибки на стороне клиента
- **5xx** – ошибки на стороне сервера



# Коды HTTP ответов

- **200** OK, **201** Created, **202** Accepted
- **301** Moved Permanently – перемещено навсегда, постоянный редирект
- **302** Moved Temporarily – перемещено временно
- **400** Bad Request – неверный запрос, ошибка формата
- **401** Unauthorized – пользователь не авторизован
- **403** Forbidden – запрещено, нет прав доступа
- **404** Not Found – обращение по несуществующему адресу
- **405** Method Not Allowed – запрос не тем HTTP методом
- **500** Internal Server Error – внутренняя ошибка сервера
- **503** Service Unavailable – сервис недоступен

# Методы HTTP запросов

- **Первая строка запроса:** GET / HTTP/1.1
- В ее начале идет **имя метода** (тип операции в HTTP)
- Самые распространенные методы:

HTTP метод	Смысл	Пример (список контактов)
GET	Получить ресурс	Получить контакт
POST	Создать ресурс	Добавить контакт
PUT/PATCH	Обновить ресурс	Обновить контакт
DELETE	Удалить ресурс	Удалить контакт

# Методы HTTP запросов

- Некоторые разработчики для простоты не используют PUT и DELETE, а вместо них используют POST
- **Итого:**

HTTP метод	Смысл	Пример (список контактов)
GET	Получить ресурс	Получить контакт
POST	Любые действия с ресурсом, которые могут его изменить	Добавить контакт Удалить контакт Обновить контакт

# Отличия GET от POST

GET	POST
Получает информацию с сервера	Посылает информацию на сервер
Немодифицирующие операции	Модифицирующие операции
Передаёт данные (параметры) в URL: <a href="https://yandex.ru/search/?order=asc&amp;text=rest">https://yandex.ru/search/?order=asc&amp;text=rest</a>	Передаёт данные в теле запроса в виде JSON'а
Может кэшироваться	Никогда не кэшируется
Длина запроса – не более 2048 символов	Длина запроса не ограничена

# Кэш и кэширование

- **Кэш** – промежуточный буфер с быстрым доступом для хранения данных
- Использование кэша называется **кэшированием**
- Используется для оптимизации производительности за счет расхода памяти. Браузер может кэшировать HTML страницы, скрипты и CSS стили
- Браузеры кэшируют GET-запросы, а POST – никогда
- **Пример:**  
cache-control: private, max-age=0, no-cache  
cache-control: no-transform, public, max-age=300
- Самый длительный промежуток для кэша – год, или 31536000, в секундах

# Cookies

- **Cookie (куки)** – небольшой фрагмент данных, который может храниться на стороне клиента, и который прикрепляется к каждому запросу
- Это пары ключ-значение (Name и Value)
- У cookie есть срок истекания (**Expires**) – когда наступит указанное время, то cookie считается истекшей, и автоматически удаляется браузером
- Куки чаще всего применяются для: аутентификации, хранения сессии и хранения персональных настроек
- В инструментах Chrome есть раздел **Application -> Cookies**, там их можно смотреть/менять/удалять/добавлять

# Тестирование API

# Понятие API

- **API (Application Programming Interface)** – набор функций, предоставляемых приложением
- Т.е. программа предоставляет наружу некоторый набор функций, который может быть использован другими программами
- За счет этого программы могут взаимодействовать между собой – вызывать функции друг друга или передавать/получать данные



# API простыми словами

- Люди используют UI, а программы – API. То есть API – это способ общения программ
- Это «контракт», который предоставляет программа: «Ко мне можно обращаться так и так, я обязуюсь делать то и это»
- То есть API – это набор функций
- API включает в себя: саму функцию (операцию), данные на входе, данные на выходе



# Вызов API

- **Напрямую:**
  - Система вызывает функции внутри себя
  - Система вызывает метод другой системы
  - Человек вызывает метод (через Postman, например)
  - Автотесты «дергают» методы, то есть используют их
- **Косвенно:**
  - Пользователь работает с GUI
- <https://habr.com/ru/post/464261/>

# REST

- **REST** – архитектура для клиент-серверного взаимодействия, основанная на протоколе HTTP, которая характеризуется следующими признаками:
  - **Отсутствие состояния** – для сервера каждый запрос клиента никак не связан с предыдущим. Сервер не запоминает состояние между запросами
  - **Ориентированность на ресурсы** – API пишется в терминах ресурсов, а не команд
- <https://ru.wikipedia.org/wiki/REST>
- <https://habrahabr.ru/post/38730/>

# REST API

- **REST API** – формально это API, построенное по REST архитектуре
- Но обычно когда говорят «REST API», то имеют в виду Web API, которое принимает/выдает JSON
- Хотя бывают REST API, которые выдают XML

# REST API – как выглядят адреса

- **Получить список всех книг**  
GET <http://site.ru/book>
- **Получить книгу номер 3**  
GET <http://site.ru/book/3>
- **Добавить книгу (данные в теле запроса)**  
POST <http://site.ru/book>
- **Изменить книгу (данные в теле запроса)**  
PUT <http://site.ru/book/3>
- **Удалить книгу**  
DELETE <http://site.ru/book/3>

# Передача параметров по HTTP

- Допустим, мы хотим вызвать некоторую функцию API, и нам нужно передать туда параметры
- Способы передачи параметров:
  - в **URL** (т.е. в адресе запроса) – может применяться во всех видах запросов:
    - в **query string**
    - в самом адресе
  - в **теле запроса** – не применяется в GET запросах, т.к. в них не принято делать тело запроса
  - в **заголовках** – применяется крайне редко

# Передача параметров в query string

- Пример URL с параметрами:
- [http://auto.drom.ru/toyota/camry/?minprice=50000&minyear=2016&mv=1.0&go\\_search=2](http://auto.drom.ru/toyota/camry/?minprice=50000&minyear=2016&mv=1.0&go_search=2)
- Часть URL, начинающаяся с символа ?, называется **query string (строка запроса)**
- По идее там может быть любой текст, но общепринято передавать там параметры в виде **параметр1=значение1&параметр2=значение2** и т.д.
- То есть параметры разделяют символом & (амперсанд)

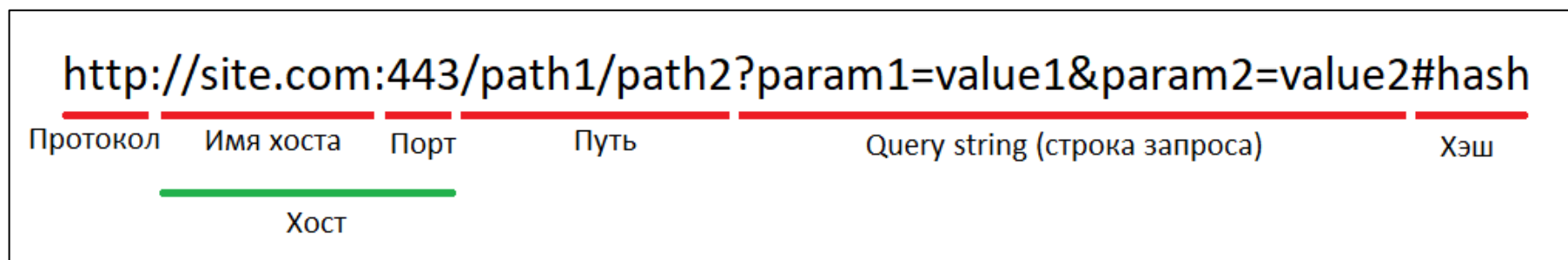
# Передача параметров в адресе

- Иногда параметры запроса передают в самом адресе запроса:
- <https://restcountries.eu/rest/v2/region/{region}>
- <https://restcountries.eu/rest/v2/region/europe>
- Здесь часть **europe** не является фиксированной, это название региона
- Другие доступные регионы: **africa, americas, asia, oceania**
- Сервер, если адреса прописаны соответствующим образом, умеет вытащить данные из самого URL и использовать их



# Формат URL

- Рассмотрим из чего состоит URL
- Некоторые части URL в разных источниках называют немного по-разному



- Порт не обязателен. Для протокола по HTTP по умолчанию берется порт 80, для HTTPS 443

# Передача параметров в не-GET запросах

- В не-GET запросах чаще всего передают параметры в теле запроса
- В данном примере передается JSON объект с полем **newsItemID** равным строке 2171
- POST /\_layouts/S7/S7NewsService.asmx/GetNewsFiles  
HTTP/1.1  
Host: partner.s7.ru  
Content-Length: 21  
Origin: https://partner.s7.ru  
Content-Type: application/json

```
{ "newsItemID": "2171" }
```

# JSON

- Текстовый формат обмена данными, основанный на JavaScript
- Синтаксис: наборы пар **ключ: значение**
- Поддерживает следующие типы данных:
  - **Числа** (целые и вещественные): 3, 4.4
  - **Логический тип** (boolean): true/false (истина/ложь)
  - **Строки**: "Строка 1"
  - **Объекты** – сущности, у которых могут быть поля
  - **Массивы** – списки значений через запятую: [1, 2, 5, 4]
  - Значение **null** – отсутствие данных

```
{  
  "name": "Ivan",  
  "age": 30,  
  "cat": {  
    "name": "Murka"  
  }  
}
```

# Тестирование API

- Вызываем endpoints API (URL)
- **Что проверяем:**
- Статус коды (положительные, отрицательные кейсы)
- Сообщения об ошибках для отрицательных кейсов
- Для GET-запроса: проверка параметров в URL (корректные, некорректные, отсутствие параметров)
- Для не-GET-запросов: тело запроса (например, валидация некорректных данных, отсутствие полей)
- Тело ответа (например, какие типы данных возвращаются)
- Аутентификация (например, вызов API без токена)
- Для методов PUT/PATCH: например, отправка не всех полей для PUT

# Документация API

- **Примеры:**
- <https://dev.hh.ru/>
- <https://reqres.in/>
- <http://rest-api.noveogroup.com/documentation>

**Postman**

# Postman

- Доступен в форме веб и десктоп приложения
- Позволяет отсылать любой тип HTTP-запросов, просматривать ответ сервера с подсветкой синтаксиса (HTML, JSON или XML)
- Хранит историю запросов, позволяет создавать свои коллекции
- Можно использовать переменные, создавать раннеры и мониторы
- Поддерживает возможность написания автотестов на JS
- <https://www.postman.com/>



# Выполнение запросов

- Выбрать тип запроса
- Указать URL запроса (endpoint)
- Заполнить необходимые параметры
- Нажать кнопку «Send»

The screenshot displays the REST Client interface with a POST request configured. Red arrows and boxes highlight the following elements:

- 1:** The request method dropdown menu, currently set to **POST**.
- 2:** The request URL input field, containing `https://reqres.in/api/users/2?name=Testname1&job=leader`.
- 3:** The **Params** tab, which is active and shows a table of query parameters.
- 4:** The **Send** button, used to execute the request.

The **Params** tab contains the following data:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> name	Testname1	
<input checked="" type="checkbox"/> job	leader	
Key	Value	Description

At the bottom of the interface, the status bar shows: **Status: 200 OK**, **Time: 633 ms**, **Size: 563 B**, and a **Download** button. The response body is displayed in **JSON** format.



# Результаты

- Код ответа (**Status**)
- Заголовки ответа (**Headers**) и тело ответа
- Результаты тестов (**Tests Results**)
- Тело ответа можно скачать в файл (**Save to a file**)

REST API платформа reqres / List users

GET <https://reqres.in/api/users?page=3> [Send](#)

Params Authorization Headers (6) Body Pre-request Script Tests Settings [Cookies](#)

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> page	3	
Key	Value	Description

Body Cookies Headers (18) Test Results

Pretty Raw Preview Visualize JSON [Save Response](#)

```
1 {
2   "page": 3,
3   "per_page": 6,
4   "total": 12,
5   "total_pages": 2,
6   "data": [],
7   "support": {
8     "url": "https://reqres.in/#support-heading",
9     "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
10  }
11 }
```

Status: 200 OK Time: 260 ms Size: 1.08 KB

Save as example  
Save to a file

# Практика

- Установите Postman (<https://www.postman.com/>)
- Выполнить запрос на получение информации о вашем аккаунте на gitHub:
- URL: [https://api.github.com/users/{your\\_account}](https://api.github.com/users/{your_account})
- Метод: GET
- Проверьте код ответа и данные в теле ответа в формате JSON

# Учебный API reqres

- Будем писать запросы на <https://reqres.in/>
- Создадим коллекцию с запросами
- Разберем следующие запросы:
  - GET
  - POST
  - PUT
  - PATCH
  - DELETE
- Base URL: <https://reqres.in/>

# GET

- GET **List users**: /api/users?page=2
- Проверяем структуру ответа согласно спецификации

```
{  
  "page": 2,  
  "per_page": 6,  
  "total": 12,  
  "total_pages": 2,  
  "data": [  
    {  
      "id": 7,  
      "email": "michael.lawson@reqres.in",  
      "first_name": "Michael",  
      "last_name": "Lawson",  
      "avatar": "https://reqres.in/img/faces/7-image.jpg"  
    },  
    ...  
  ]  
}
```

- Параметр 'page' является **необязательным**. По-умолчанию (/api/users) будет выведена первая страница.
- Проверяем поведение системы при обращении к несуществующей странице (/api/users?page=1111)
- Также проверяем параметр 'per\_page' и логику расчета 'total' и 'total\_pages' в зависимости от параметра 'per\_page'

# GET

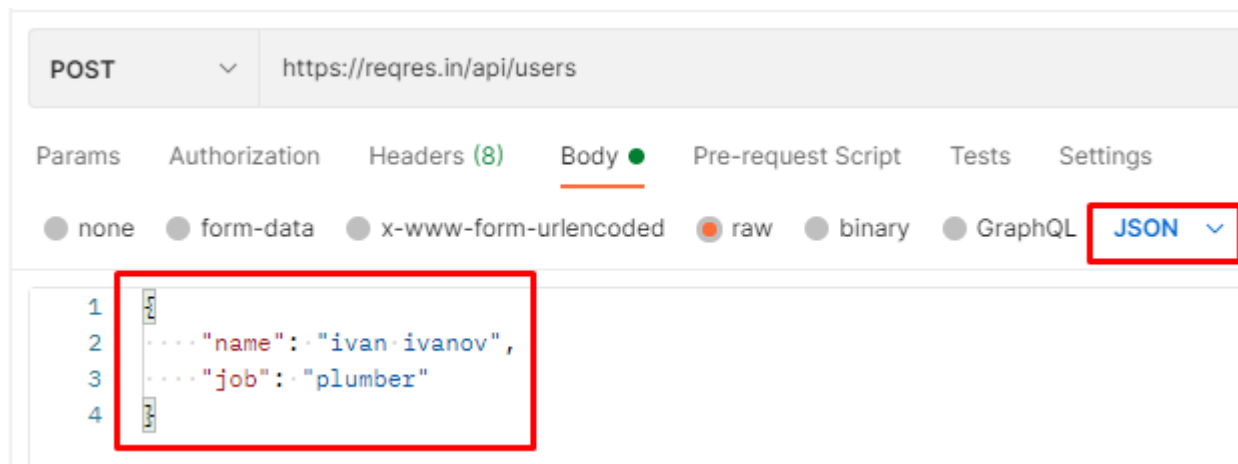
- GET **Single user**: /api/users/{user\_id}
- Проверяем позитивный кейс с существующим юзером (user\_id=2). Структура ответа согласно спецификации

```
"data": {  
  "id": 2,  
  "email": "janet.weaver@reqres.in",  
  "first_name": "Janet",  
  "last_name": "Weaver",  
  "avatar": "https://reqres.in/img/faces/2-image.jpg"  
},  
"support": {  
  "url": "https://reqres.in/#support-heading",  
  "text": "To keep ReqRes free, contributions towards server costs are appreciated!"  
}
```

- Проверяем негативный кейс (user\_id=23)
- Ожидаем статус код 404 и пустой ответ

# POST

- POST **Create user**: /api/users
- Передаем тело запроса в формате JSON согласно спецификации:



- Проверяем статус код ответа (201 Created)
- Проверяем структуру ответа согласно спецификации

# PUT/PATCH

- PUT/PATCH **Update user**: /api/users/{user\_id}
- **PUT**: передаем полностью тело запроса, как для POST create user

```
{
  ... "name": "Jane",
  ... "job": "Doctor"
}
```

- **PATCH**: можем передать только один параметр

```
{
  ...
  ... "job": "Doctor2"
}
```

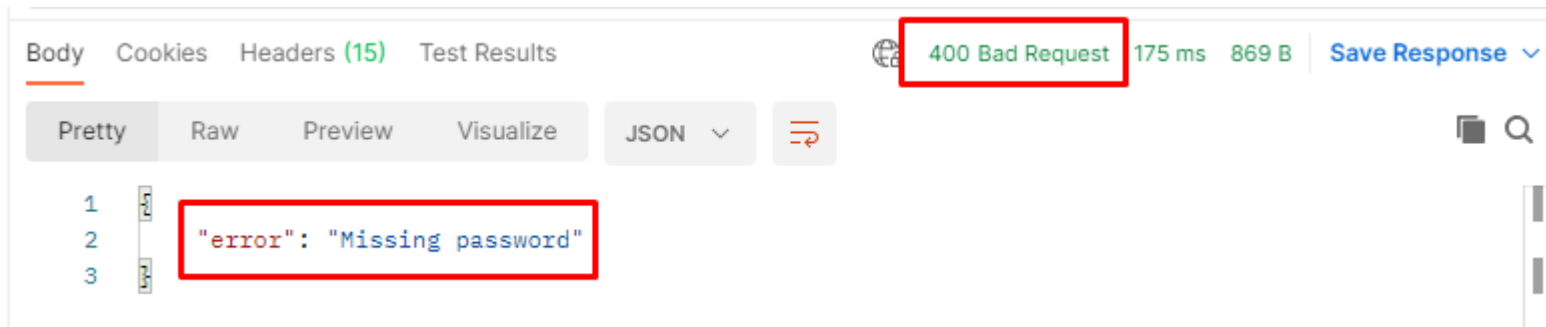
- Это учебный API, он реализован так, что PUT и PATCH работают одинаково

# Пример регистрации

- POST **Register user**: /api/register
- Передаем параметры согласно спецификации:

```
... "email": "eve.holt@reqres.in",  
... "password": "pistol1"
```

- В ответ получаем токен: "token": "QpwL5tke4Pnpja7X4«
- **Негативный кейс** – регистрация без пароля
- Проверяем статус код и сообщение об ошибке:





# Пример авторизации

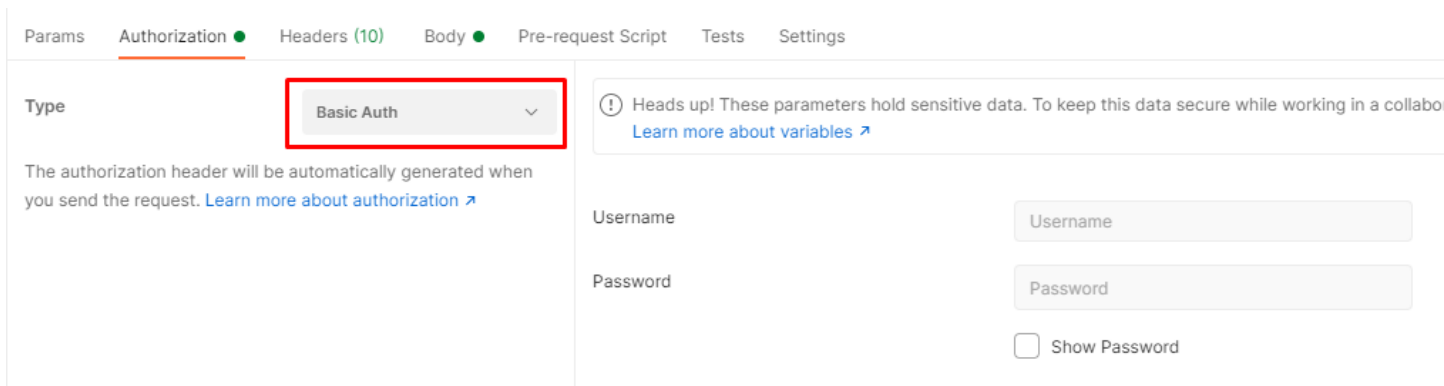
- POST **Login user**: /api/login
- Передаем параметры согласно спецификации:

```
... "email": "eve.holt@reqres.in",  
... "password": "pistol1"
```

- В ответ получаем токен: "token": "QpwL5tke4Pnpja7X4«
- **Негативный кейс** аналогичен негативному кейсу регистрации

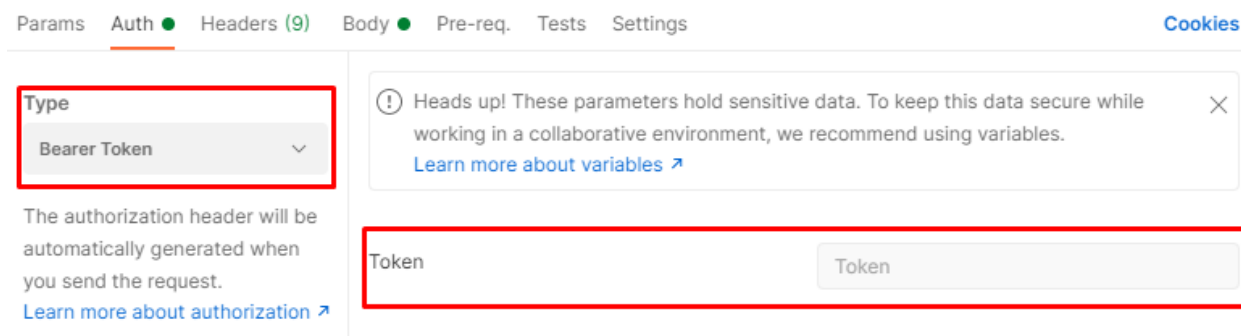
# Варианты авторизации в Postman

- Вкладка **Authorization** → Type
- По логину и паролю – Basic Auth



The screenshot shows the Postman interface with the 'Authorization' tab selected. The 'Type' dropdown is set to 'Basic Auth' and is highlighted with a red box. Below the dropdown, a text box explains that the authorization header will be automatically generated. To the right, there are input fields for 'Username' and 'Password', and a checkbox for 'Show Password'. A warning message at the top right states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)'.

- По ключу (API Key) или токenu (Bearer Token)



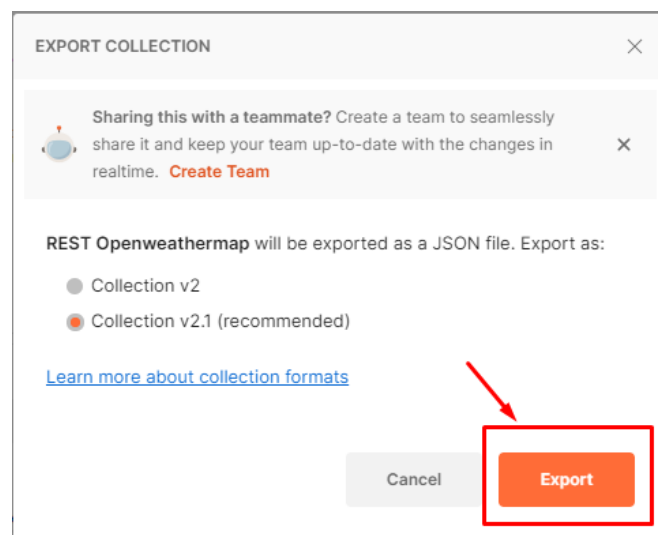
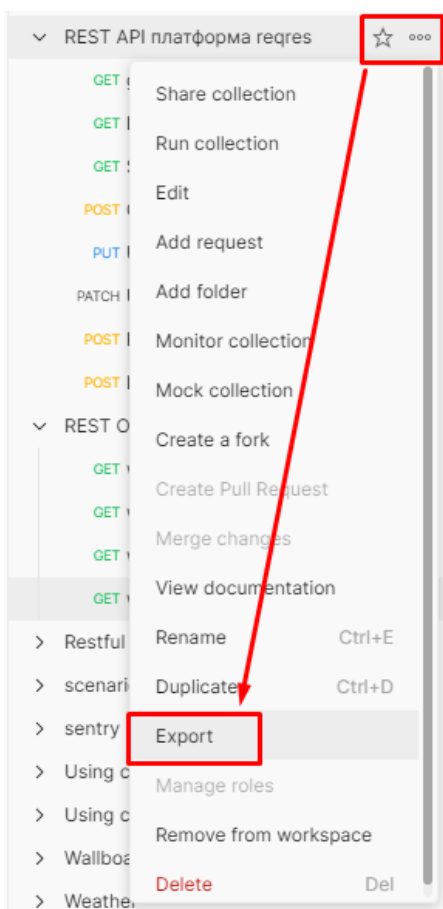
The screenshot shows the Postman interface with the 'Auth' tab selected. The 'Type' dropdown is set to 'Bearer Token' and is highlighted with a red box. Below the dropdown, a text box explains that the authorization header will be automatically generated. To the right, there is a large input field for the 'Token' value, which is also highlighted with a red box. A warning message at the top right states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)'.

# Дом. задание «Postman. Reqres»

- Создайте коллекцию с запросами для учебного API:  
<https://reqres.in/>
- Коллекция должна содержать **не менее 5 запросов**
- Необходимо реализовать вызовы следующих видов запросов:
  - GET
  - POST
  - PUT
  - PATCH
  - DELETE

# Дом. задание «Postman. Reqres»

- Домашнее задание присылать в виде файла в формате json
- Коллекцию в Postman можно экспортировать в json:



# К следующему занятию

- Открытое API для получения информации о погоде:  
<https://openweathermap.org/current>
- Зарегистрируйтесь на сайте и получите ключ APPID
- Прочитайте описание API