

Manual QA

Тестирование API



План занятия

- Клиент-серверная архитектура
- TCP/IP
- API
- REST API
- HTTP и HTTPS
- Postman



ОСНОВНОЙ БЛОК





Клиент-серверная архитектура

ВЕБ-СЕРВЕР

Сервер:

сервер — выделенный или специализированный компьютер для выполнения сервисного программного обеспечения

сервер — принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-поток или другими данными. (Веб-сервер)



КЛИЕНТ

Клиент:

клиент — запрашивает определенный сервис или ресурс

клиент — получает ответ и отображает результат

пользователю

примеры клиентов: е-мейл клиент (программа outlook например), веб-браузер

Веб-браузер:

это ПО для размещения, получения и отображения контента, такого как веб-страницы, картинки, видео и других файлов во всемирной паутине

это приложение работает на вашем локальном компьютере

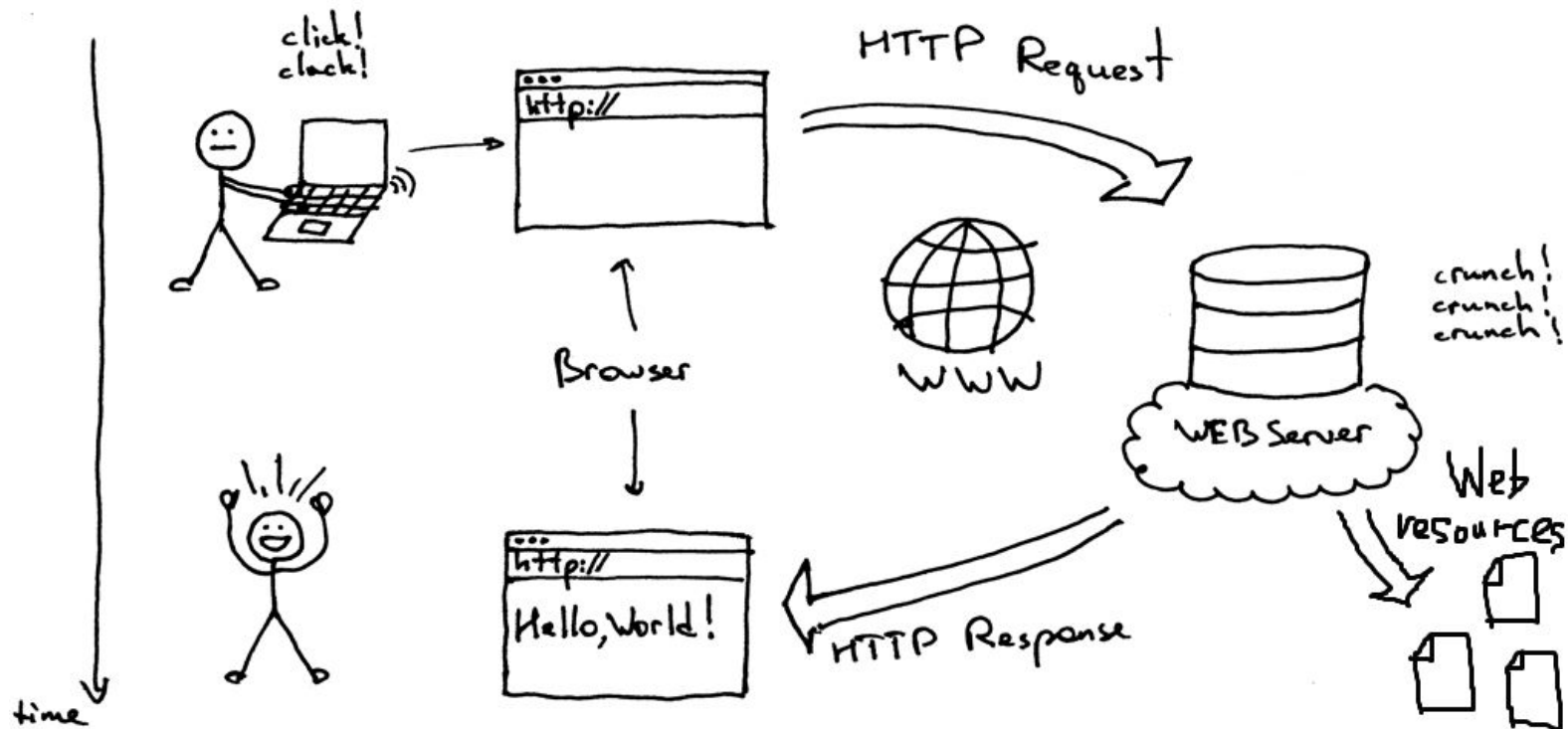




Клиент-серверная архитектура

Это базовая модель взаимодействия между устройствами в сети, в которой компьютеры, называемые клиентами, обращаются к другим компьютерам, называемым серверами, для получения различных услуг, данных или ресурсов.

ЗАПРОС - ОТВЕТ



EXAMPLE

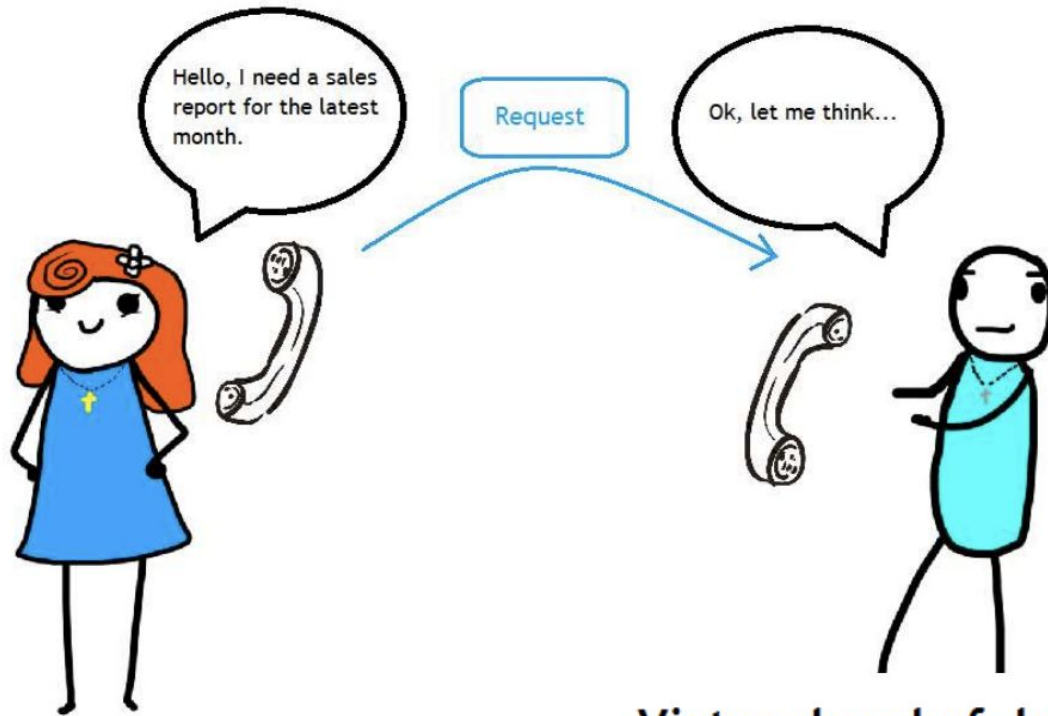


Boss (you)



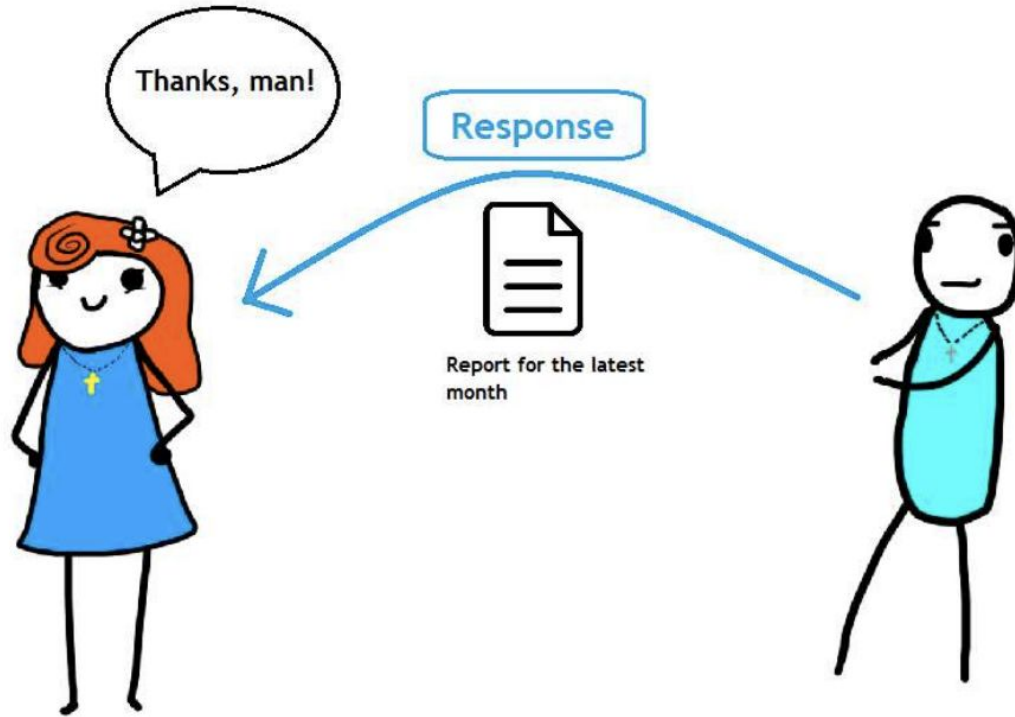
Alice, secreatry (browser)

EXAMPLE

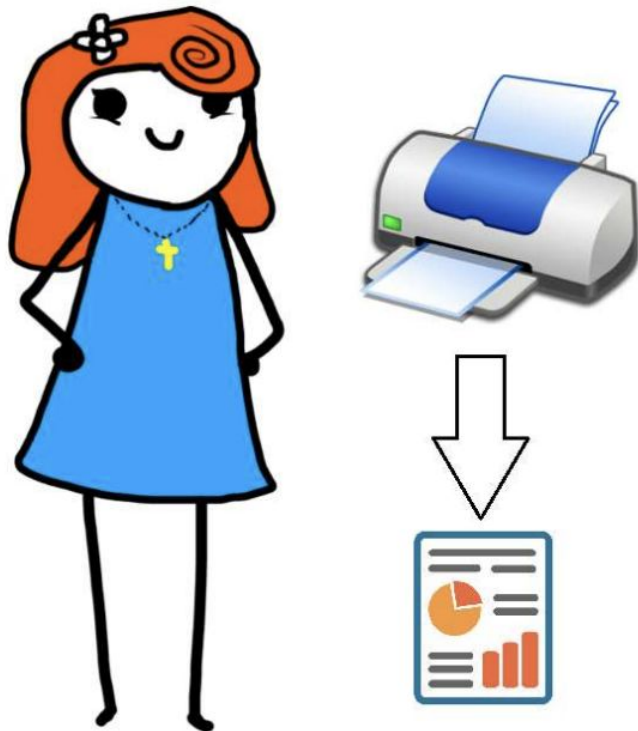


Victor, head of departement (server)

EXAMPLE



EXAMPLE



Основная суть браузера:

- Отображать, обрабатывать и показывать информацию полученную из сервера
- А затем взаимодействовать с полученной информацией (навигая, кнопки, ссылки)

EXAMPLE





TCP/IP

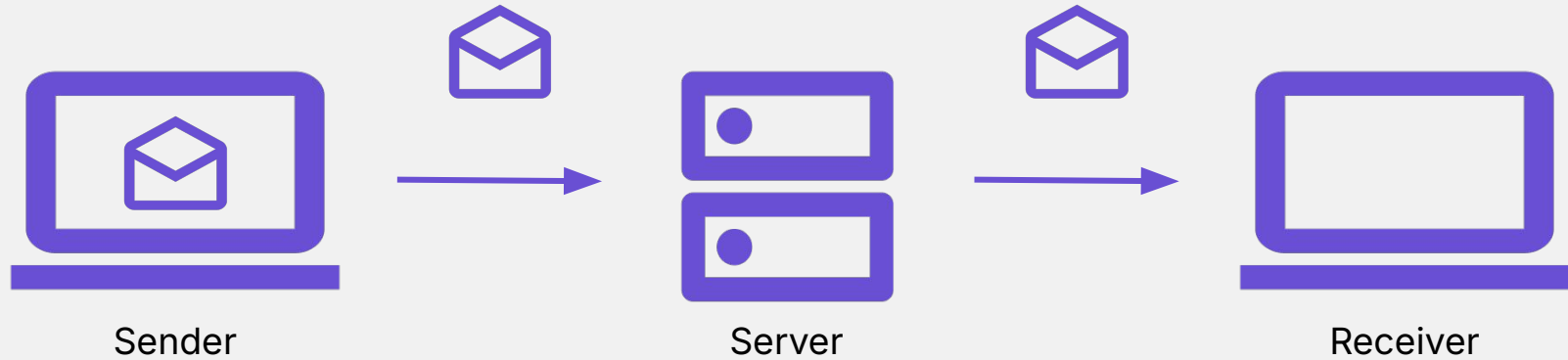




TCP/IP (Transmission Control Protocol/Internet Protocol)

Это набор протоколов, которые обеспечивают коммуникацию между устройствами в сети, в том числе в интернете. Был разработан для создания стандарта связи между компьютерами.

TCP/IP



Основные аспекты TCP/IP

- Все устройства, подключенные к интернету, используют этот набор протоколов для обмена данными.
- TCP/IP задает правила, которые компьютеры должны соблюдать, чтобы обмениваться данными через интернет: разбиение данных на пакеты, их доставка и сборка на стороне получателя.
- TCP/IP - это большой набор протоколов, который включает в себя множество уровней для различных задач.



ВОПРОСЫ





API





API (Application Programming Interface)

Это набор правил и механизмов, который позволяет различным программам взаимодействовать друг с другом, предоставляя стандартизированный способ передачи данных и вызова функций.

Подходы к построению API



REST (Representational State Transfer)



Основные особенности

- Обмен данными
- Использование стандартов HTTP
- Гибкость форматов

Примеры использования

- Работа с веб-приложениями (например, API социальных сетей).
- Интеграция микросервисов.

SOAP (Simple Object Access Protocol)



Основные особенности

- Жесткая стандартизация
- Использование HTTP(S)
- Обработка ошибок
- Межплатформенность

Примеры использования

- Банковские системы и финансовые сервисы.
- Взаимодействие корпоративных приложений.

Сравнение REST и SOAP

	REST	SOAP
Гибкость форматов данных	Поддерживает JSON, XML и другие форматы	Работает исключительно с XML
Простота реализации	Проще в использовании и легче в освоении, поэтому он чаще используется для публичных API	Более сложен, но предоставляет больше встроенных возможностей
Стандартизация	Более гибок и подходит для быстрого развертывания	Следует строгим стандартам, что делает его предпочтительным выбором для критически важных систем

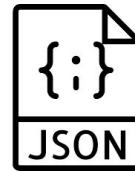
ФОРМАТЫ ОБМЕНА ДАННЫМИ

можно использовать для обмена информацией между двумя несовместимыми системами

используются для хранения данных в файловой системе так же как и для сохранения и выборки информации в базе данных

служат для описания данных


```
<?xml version="1.0" encoding="UTF-8"?><recipe><recipeName>Ice Cream Sundae</recipeName><ingredlist><listitem><quantity>3</quantity><itemdescription>chocolate syrup or chocolate fudge</itemdescription></listitem><listitem><quantity>1</quantity><itemdescription>nuts</itemdescription></listitem><listitem><quantity>1</quantity><itemdescription>cherry</itemdescription></listitem></ingredlist><preptime>5 minutes</preptime></recipe>
```



```
[{"name": "Петр", "surname": "Петров", "faculty": "Машиностроительный", "group": "М-72", "address": {"city": "Москва", "street": "Речная", "house": "12", "apartment": "24"}}, {"name": "Алексей", "surname": "Алексеев", "faculty": "Экономический", "group": "Э-51", "address": {"city": "Москва", "street": "Береговая", "house": "2", "apartment": "14"}}]
```

How API works

Web App

A video frame showing two men sitting at a table. The man on the left is wearing a maroon shirt, and the man on the right is wearing a light-colored shirt and glasses. They are both looking towards the camera. A white label with the text 'Web App' is positioned in the lower center of the frame, partially overlapping the man in the maroon shirt. The background shows a dark wooden wall and a white curtain.

ПРОТОКОЛЫ ВЕБ-СЕРВИСОВ

Restful API — это описание REST протокола.

Прилагательное. Описывает КАК выглядит и КАК работает ваш REST протокол. Это свод правил направленный на проверку и валидацию безопасности, правильности и принадлежности к общим принципам проектирования REST архитектуры

The logo features the text "RESTful API" enclosed in large, thin, grey curly braces. The word "REST" is in a bold, uppercase, sans-serif font. The word "ful" is in a lowercase, italicized, script font. The word "API" is in a bold, uppercase, sans-serif font, matching "REST".

{RESTful API}

ПРОТОКОЛЫ ВЕБ-СЕРВИСОВ

WebSocket — это независимый протокол, основанный на протоколе TCP. Он делает возможным более тесное взаимодействие между браузером и веб-сайтом, способствуя распространению интерактивного содержимого и созданию приложений реального времени





ВОПРОСЫ





URL, Domain,
Protocol, Extension

URL (UNIFORM RESOURCE LOCATOR)

кусочки информации на сервере называются ресурсами. У каждого ресурса есть уникальный идентификтор, URL

браузер обращается к серверу (тому серверу чьё имя указано в URL), устанавливает с ним сетевое соединение, отправляет идентификатор ресурса, то есть имя ресурса который он хочет получить



URL vs URI

**Где здесь Протокол, Домен и Расширение, URL,
URI, Endpoint?**

1. <https://polakohedonist.club/ru/about>
2. <https://datatracker.ietf.org/doc/html/rfc3986#section-1.2>



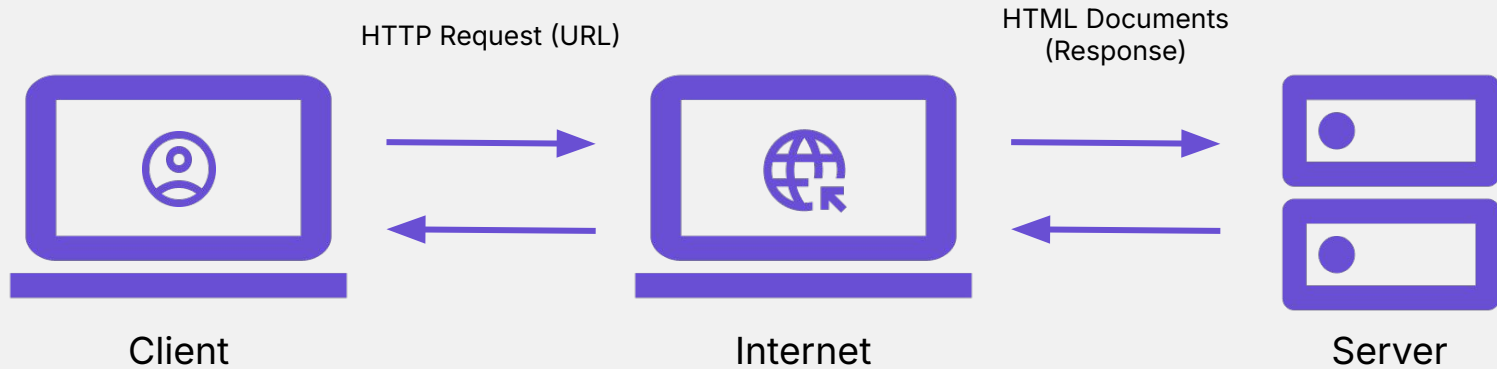
HTTP и HTTPS



HTTP (HyperText Transfer Protocol)

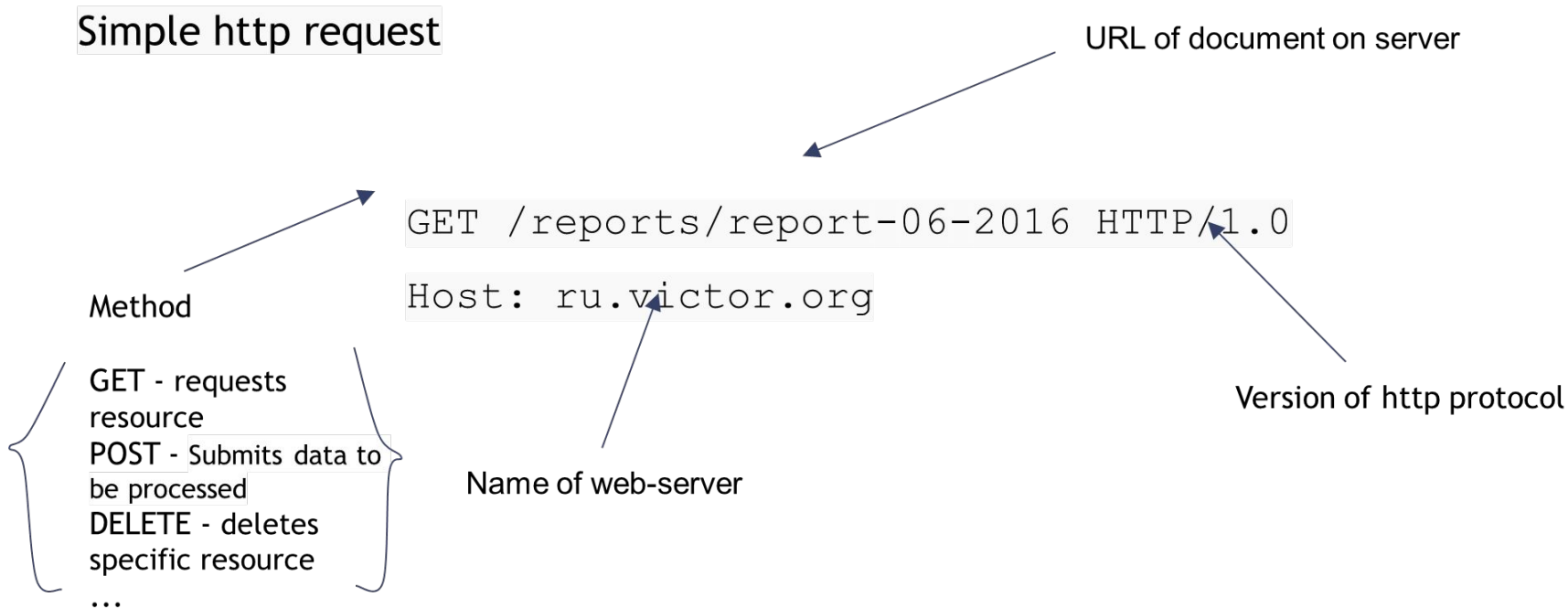
Это протокол передачи данных, который обеспечивает взаимодействие между веб-браузером (клиентом) и веб-сервером.

HTTP



HTTP запрос

Simple http request



HTTP ответ

```
HTTP/1.1 200 OK Date: Fri, 22 Nov 2024 12:00:00 GMT
Server: Apache/2.4.41 (Ubuntu) Content-Type: text/html;
charset=UTF-8 Content-Length: 157 Connection: close
<html> <head><title>Example Page</title></head>
<body> <h1>Welcome to Example.com!</h1> <p>This is
an example of an unencrypted HTTP response.</p>
</body> </html>
```



HTTPS (HyperText Transfer Protocol Secure)

Это защищенная версия протокола HTTP, которая обеспечивает безопасное взаимодействие между веб-браузером и веб-сервером.

HTTPS

Поставщики сетей будут видеть ваши HTTPS следующим образом:

Domain: secure-example.com

IP Address: 192.0.2.123 Port: 443 Protocol: TLS 1.3

Timestamp: 22-Nov-2024 12:00:00 Packet Size: 1.2 KB

Encrypted Data: fkdj83kfa#\$2jslkaf39fkjas...

НЕ увидят тело запроса и ответа, а также Headers + Endpoint

Методы HTTP/HTTPS

GET - Получить информацию с сервера не изменяя ее

Идемпотентность: Да

Повторные одинаковые запросы GET всегда возвращают один и тот же результат без побочных эффектов.

Тело запроса: Не используется

Все данные передаются через URL (параметры запроса или путь).

Примеры использования:

- Получение списка ресурсов:

GET /users

- Получение конкретного ресурса:

GET /users/123



Методы HTTP/HTTPS

POST – Создание новых ресурсов на сервере или отправка данных для обработки.

Идемпотентность: Нет

Повторные одинаковые запросы POST создают новые ресурсы или выполняют действие несколько раз.

Тело запроса: Обязательно

Данные для создания ресурса передаются в теле запроса, обычно в формате JSON.

Примеры использования:

- Создание нового ресурса:

POST /users

```
{ "name": "Иван Иванов", "email": "ivan.ivanov@example.com" }
```

- Отправка данных для обработки (например, формы или загрузка файлов).



Методы HTTP/HTTPS

PUT – Полное обновление существующего ресурса или его создание, если он отсутствует.

Идемпотентность: Да

Повторные одинаковые запросы PUT дают одинаковый результат (ресурс перезаписывается).

Тело запроса: Обязательно

Полное представление обновляемого ресурса.

Примеры использования:

- Обновление ресурса:

PUT /users/123

`{ "name": "Мария Иванова", "email": "maria.ivanova@example.com" }`

- Создание ресурса, если его нет:

PUT /users/456

PUT

Методы HTTP/HTTPS



Ключевые различия между POST и PUT

Характеристика	POST	PUT
Цель	Создание нового ресурса.	Обновление или создание ресурса (с заменой).
Идемпотентность	❌ Нет — повторный запрос создаст копию.	✅ Да — повторный запрос дает тот же результат.
URL	Указывает на коллекцию (e.g., <code>/users</code>).	Указывает на конкретный ресурс (e.g., <code>/users/123</code>).
Тело запроса	Данные для создания ресурса.	Полное представление ресурса.
Результат	Новый ресурс, созданный сервером.	Новый или замененный ресурс.

Методы HTTP/HTTPS

PATCH – Частичное обновление существующего ресурса (изменение только указанных полей)

Идемпотентность: Да

Повторные одинаковые запросы PATCH приводят к одному и тому же результату.

Тело запроса: Обязательно

. Содержит только поля, которые необходимо обновить.

Примеры использования:

- Обновление конкретных полей ресурса:

PATCH /users/123

{ "name": "Мария Иванова" }



Методы HTTP/HTTPS

DELETE – Удаление ресурса с сервера.

Идемпотентность: Да

Повторное удаление одного и того же ресурса не вызывает ошибок (ресурс уже удален).

Тело запроса: Используется редко

. Все необходимые данные обычно указываются в URL (например, ID ресурса).

Примеры использования:

- Удаление конкретного ресурса:

DELETE /users/123

Delete 

Методы HTTP/HTTPS

Метод	Цель	Идемпотентность	Тело запроса	Примеры использования
GET	Получение ресурсов	✓ Да	✗ Не используется	Получение данных, напр. <code>GET /users</code>
POST	Создание ресурсов	✗ Нет	✓ Обязательно	Создание данных, напр. <code>POST /users</code>
PUT	Полное обновление ресурса	✓ Да	✓ Обязательно	Перезапись данных, напр. <code>PUT /users/1</code>
PATCH	Частичное обновление ресурса	✓ Да	✓ Обязательно	Обновление полей, напр. <code>PATCH /users/1</code>
DELETE	Удаление ресурсов	✓ Да	✗ Редко используется	Удаление данных, напр. <code>DELETE /users/1</code>



CRUD

Это набор базовых операций для работы с данными.

- Create (Создание) — HTTP метод POST.
- Read (Чтение) — HTTP метод GET.
- Update (Обновление) — HTTP метод PUT + Patch.
- Delete (Удаление) — HTTP метод DELETE.



Заголовки (Headers)

Это набор пар «имя-значение», разделенных двоеточием. В заголовках передается служебная информация, которая сопровождает запрос или ответ.



Тело сообщения (Body)

Тело сообщения содержит передаваемые данные. В запросах: данные могут включать содержимое форм, файлы или другие передаваемые ресурсы, загружаемые на сервер. В ответах: тело сообщения обычно содержит HTML-страницу, JSON-данные или другой контент, запрошенный клиентом.

Коды состояния HTTP: 1xx – Информационные ответы



Пояснение

Эти коды информируют о том, что запрос клиента принят и продолжается обработка. Обычно такие ответы промежуточные.

Примеры

- **100 Continue** – сервер ожидает продолжения передачи данных.
- **101 Switching Protocols** – клиент запросил переключение на другой протокол, и сервер это подтверждает.

Коды состояния HTTP: 2xx – Успешные ответы



Пояснение

Эти коды означают, что запрос клиента был успешно обработан.

Примеры

- **200 OK** – запрос выполнен успешно, и данные переданы клиенту.
- **201 Created** – ресурс успешно создан на сервере.
- **204 No Content** – запрос выполнен успешно, но тело ответа пустое.

Коды состояния HTTP: 3xx – Перенаправления



Пояснение

Эти коды указывают, что для завершения запроса требуется дополнительное действие, например, переход на другой URL.

Примеры

- **301 Moved Permanently** – ресурс перемещен на другой URL, редирект постоянный.
- **302 Found** – временный редирект на другой URL.
- **304 Not Modified** – ресурс не изменился, можно использовать кэшированные данные.

Коды состояния HTTP: 4xx – Ошибки клиента



Пояснение

Эти коды сообщают о том, что запрос клиента содержит ошибку, и сервер не может его обработать.

Примеры

- **400 Bad Request** – запрос содержит синтаксическую ошибку.
- **401 Unauthorized** – требуется авторизация для доступа к ресурсу.
- **403 Forbidden** – доступ к ресурсу запрещен.
- **404 Not Found** – ресурс не найден на сервере.

Коды состояния HTTP: 5xx – Ошибки сервера



Пояснение

Эти коды говорят о том, что проблема возникла на стороне сервера.

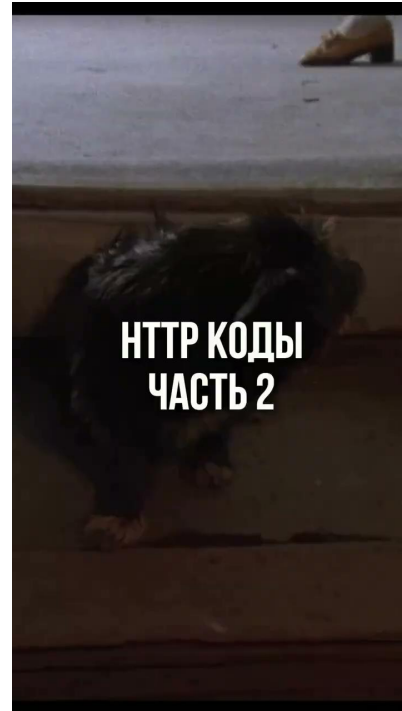
Примеры

- **500 Internal Server Error** – внутренняя ошибка сервера, запрос не может быть обработан.
- **502 Bad Gateway** – сервер выступает как шлюз и получил некорректный ответ от другого сервера.
- **503 Service Unavailable** – сервер временно недоступен, перегружен или находится на техническом обслуживании.

HTTP СТАТУС КОДЫ

`https://trln.notion.site/API-130b742b
fe1f80ed8334d7f1374cf91f`

HTTP СТАТУС КОДЫ





ВОПРОСЫ





Token, Cookie, Cache

Токен

Токен – это цифровой ключ, который используется для аутентификации и авторизации. Он передается вручную, обычно в заголовке HTTP-запроса.



Cookies

Cookies — это небольшие файлы, которые хранят данные на стороне клиента (в браузере) и автоматически отправляются с каждым HTTP-запросом к серверу.

Основное назначение:

- Хранение сессий (например, `session_id`).
- Сохранение настроек пользователя (например, язык интерфейса или корзина покупок).
- Используются для аутентификации с помощью сессионных данных.

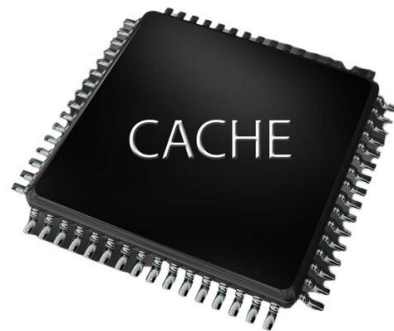


Cache

Cache — это механизм временного хранения данных, чтобы ускорить их повторное использование.

Суть: Кэш хранит данные (например, веб-страницы, изображения, или ответы API) локально, чтобы их не приходилось повторно запрашивать с сервера.

- Пример: Когда вы открываете веб-сайт, браузер сохраняет его изображения и файлы стилей (CSS) в кэше. При следующем посещении браузер загружает эти данные из кэша, а не запрашивает их с сервера.
- Зачем он нужен?
- Уменьшает время загрузки страниц.
- Снижает нагрузку на сервер.
- Экономит интернет-трафик.



Cache vs Cookies

Характеристика	Cache (Кэш)	Cookies (Куки)
Цель	Ускорение загрузки веб-страниц.	Хранение данных пользователя.
Тип данных	Ресурсы: HTML, CSS, JS, изображения, ответы API.	Данные сессии: ID пользователя, язык, корзина покупок.
Размер	Может быть большим (мегабайты).	Ограничен: 4 КБ на один файл.
Где хранится?	Локально в браузере или на промежуточных серверах.	Локально в браузере.
Как отправляется?	Не отправляется автоматически. Используется только локально.	Отправляется с каждым HTTP-запросом.
Истечение срока (TTL)	Контролируется через HTTP-заголовки (например, <code>Cache-Control</code>).	Задается через атрибут <code>Expires</code> или <code>Max-Age</code> .
Обновление	Требуется запрос нового ресурса, если срок действия истек.	Изменяется или удаляется вручную через сервер или клиентский скрипт.
Примеры использования	Сохранение изображений и файлов для повторного использования. 	Хранение ID сессии, авторизационных токенов.



JSON





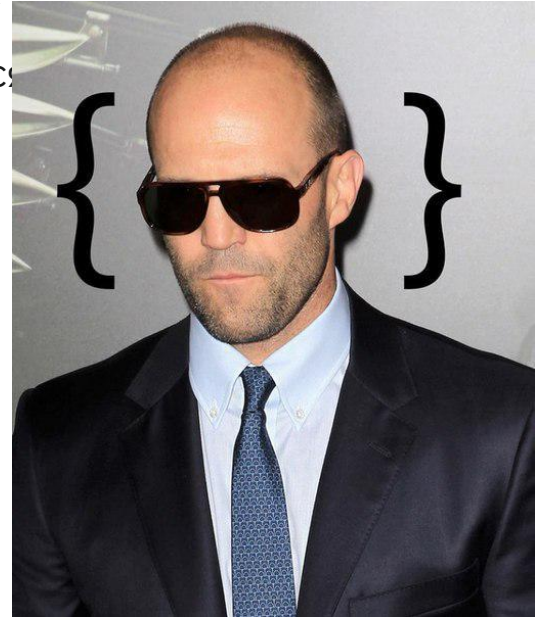
JSON (JavaScript Object Notation)

Это текстовый формат обмена данными, основанный на языке JavaScript.

JSON ФОРМАТ

Запись — это неупорядоченное множество пар ключ: значение, заключённое в фигурные скобки «{ }» между ними стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.

```
{  
  "firstName": "Иван",  
  "lastName": "Иванов",  
  "address": {  
    "streetAddress": "Московское ш., 101, кв.101",  
    "city": "Ленинград",  
    "postalCode": 101101  
  }  
}
```

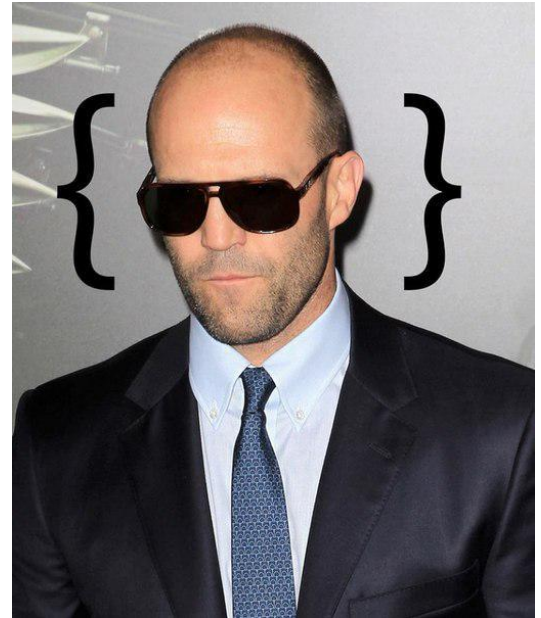


JSON ФОРМАТ

Ключ — это строка.

Значение — может быть массив, число, литералы, строка.

```
{  
  "firstName": "Иван",  
  "lastName": "Иванов",  
  "address": {  
    "streetAddress": "Московское ш., 101, кв.101",  
    "city": "Ленинград",  
    "postalCode": 101101  
  },  
  "phoneNumbers": ["812 123-1234", "916 123-4567"]  
}
```



```
```json
{
 "city": "New York",
 "country": "United States"
}
```
```

Пример JSON-объекта

Типы значений в JSON: строки



Пояснение

Это последовательность символов, заключенная в двойные кавычки.

Пример

```
```json
{
 "firstName": "Tom"
}
```
```

Типы значений в JSON: числа



Пояснение

Числа могут быть целыми или с плавающей точкой.

Пример

```
```json
{
 "age": 30
}
```
```

Типы значений в JSON: булевы значения



Пояснение

Принимают два значения: `true` или `false`.

Пример

```
```json
{
 "married": false
}
```
```

Типы значений в JSON: значение `null`



Пояснение

Указывает на отсутствие данных.

Пример

```
```json
{
 "type": null
}
```
```

Типы значений в JSON: массивы



Пояснение

Это упорядоченные списки значений, заключенные в квадратные скобки `[]`. В массиве могут быть строки, числа, объекты и другие массивы.

Пример

```
```json
{
 "hobbies": ["football", "reading",
"swimming"]
}
```



# Типы значений в JSON: массивы объектов



## Пояснение

Массив может содержать объекты JSON.

## Пример

```
```json
{
  "students": [
    {"firstName": "Tom", "lastName":
"Jackson"},
    {"firstName": "Linda",
"lastName": "Garner"},
    {"firstName": "Adam",
"lastName": "Cooper"}
  ]
}
```



ВОПРОСЫ





ЗАДАНИЕ



ПРАКТИКА JSON

Кузов	
Тип кузова	седан
Количество дверей/мест	4/5
Дорожный просвет, мм	160
Объем багажника, л	520
Размер шин	175/65 R14

Двигатель	
Тип двигателя	бензиновый
Рабочий объем, см3	1596
Топливо, рекомендованное производителем	АИ-95
Максимальная мощность, л.с.(об/мин)	87 (5100)

Трансмиссия	
Передний привода	ДА
Тип КПП, количество передач	МКПП, 5

```
{  
  "name": "Петр",  
  "surname": "Петров",  
  "faculty": "МАШ",  
  "group": "М-72",  
  "address": {  
    "city": "Москва",  
    "street": "Речная",  
    "house": "12",  
    "apartment": "24"  
  }  
}
```



Задание

Ответьте на вопросы в чат:



Представить информацию о Mark Zuckerberg, Bill Gates, Elon Musk, согласно представленной модели данных.
Использовать [JSON Editor Online](#)



ВОПРОСЫ





Postman





Postman

Это инструмент для разработки, тестирования и документирования REST API, который широко используется разработчиками, тестировщиками и аналитиками для проверки взаимодействия между клиентом и сервером.

Основные возможности Postman

Создание запросов API:

- Поддержка всех HTTP-методов: **GET**, **POST**, **PUT**, **DELETE**, **PATCH**, и других.
- Возможность добавлять заголовки, параметры, тело запроса и токены аутентификации.
- Работа с различными форматами данных: JSON, XML, HTML, текст.

Основные возможности Postman

Тестирование API:

- Тестирование запросов вручную: отправка запросов к серверу и анализ ответов.
- Автоматизация тестов: с использованием встроенного редактора JavaScript для написания тестов, например:

```
pm.test("Статус-код 200", function () {
    pm.response.to.have.status(200);
});
```

Основные возможности Postman



Работа с коллекциями запросов



Поддержка переменных



Тестирование окружений (environments)



Mock-сервисы



Интеграция с CI/CD



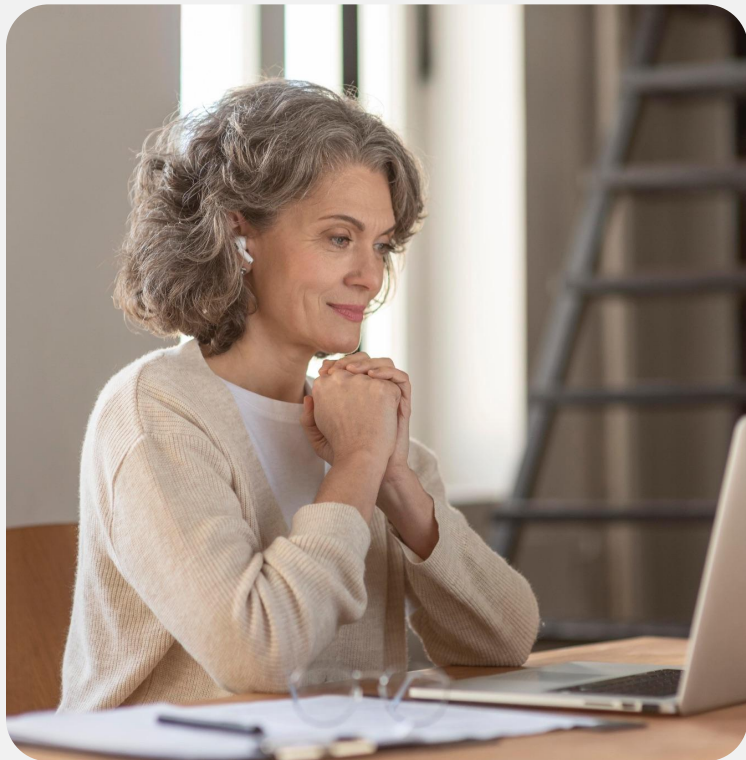
ВОПРОСЫ





ЗАДАНИЕ





Задание

1. Установить [Postman](#)
2. Составить запросы для [Regres](#)



ВОПРОСЫ



Домашнее задание

1. Реализовать вызов метода Register Successful: <https://reqres.in/api/register>
2. Добавить скрипт для автоматической проверки статус кода ответа 200.