JS: Fetch

AIT
Hi-tech team

НАШИ ПРАВИЛА

Включенная камера

Вопросы по поднятой руке

Не перебиваем друг друга

Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору

Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ПОВТОРЕНИЕ

- Какой аргумент принимает Promise.all?
- В чём отличие Promise.all от Promise.race?
- На что указывает ключевое слово await?
- Три главных компонента в клиент-серверной архитектуре
- Назовите 4 главных метода в запросах
- Назовите группы статус кодов и за что каждая из них отвечает

ЦЕЛЬ

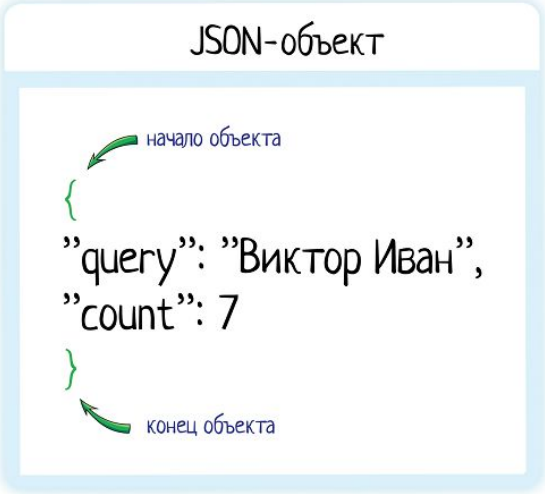
Изучить способ отправки запросов на сервер - Fetch

ПЛАН ЗАНЯТИЯ

- JSON
- Fetch


{JSON}
JavaScript Object Notation

JSON - текстовый формат данных, следующий за синтаксисом объекта JavaScript, который был популяризирован Дугласом Крокфордом. Несмотря на то, что он очень похож на буквенный синтаксис объекта JavaScript, его можно использовать независимо от JavaScript, и многие среды программирования имеют возможность читать (анализировать) и генерировать JSON.



JSON представляет собой коллекцию пар ключ-значение. Ключи и значения разделяются двоеточием, а пары ключ-значение - запятой. Объект начинается с { и заканчивается }.

```
{
  "orderID": 12345,
  "shopperName": "Ivan Ivanov",
  "shopperEmail": "ivanov@example.com",
  "contents": [
    {
      "productID": 34,
      "productName": "Super product",
      "quantity": 1
    },
    {
      "productID": 56,
      "productName": "Wonderful product",
      "quantity": 3
    }
  ],
  "orderCompleted": true
}
```

Примечания

- JSON - это чисто формат данных - он содержит только свойства, без методов.
- JSON требует двойных кавычек, которые будут использоваться вокруг строк и имён свойств. Одиночные кавычки недействительны.
- Даже одна неуместная запятая или двоеточие могут привести к сбою JSON-файла и не работать.

Методы работы с JSON

1. Преобразует JavaScript объект в JSON-строку - **`JSON.stringify(obj)`**

```
const obj = { name: "John", age: 30 };
const jsonString = JSON.stringify(obj);
```

1. Преобразует JSON-строку в JavaScript объект - **`JSON.parse(jsonString)`**

```
const jsonString = '{"name":"John","age":30}';
const obj = JSON.parse(jsonString);
```

- Простой и легко читаемый синтаксис, который понятен и компьютеру, и человеку.
- Лёгкость и компактность. JSON-файлы весят меньше, чем файлы других форматов, и загружаются быстрее.
- Отсутствие зависимости от конкретного языка программирования.
- Универсальность в типах данных. JSON можно использовать для хранения массивов, упорядоченных списков и коллекций пар «ключ — значение».
- Широкая поддержка в браузерах. Все современные браузеры поддерживают работу с JSON.
- Самостоятельное документирование.

XML

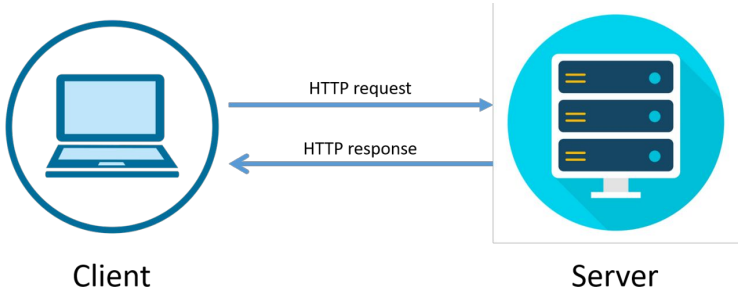
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<person>
 <name>Иван</name>
 <age>37</age>
 <mother>
  <name>Ольга</name>
  <age>58</age>
 </mother>
 <children>
  <child>Маша</child>
  <child>Игорь</child>
  <child>Таня</child>
 </children>
 <married>true</married>
 <dog null="true" />
</person>
```

VS

JSON

```json
{
 "person":{
  "name":"Иван",
  "age":37,
  "mother":{
   "name":"Ольга",
   "age":58
  },
  "children":[
   "Маша",
   "Игорь",
   "Таня"
  ],
  "married":true,
  "dog":null
 }
```

Fetch

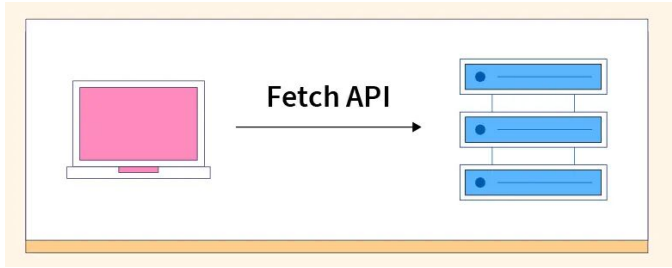


Метод fetch() — современный и очень мощный способ для создания сетевых запросов и получения информации с сервера

```
let promise = fetch(url, [options])
```

url – URL для отправки запроса.

options – дополнительные параметры: метод, заголовки и так далее.



fetch() запускает запрос и возвращает **promise**.

Когда запрос удовлетворяется, promise разрешается(resolved) объектом ответа (Response object)

Если ответ не получается получить из-за, например, неполадок сети, **promise** отклоняется (promise is rejected).

async/await синтаксис очень помогает при работе с **fetch()**

Для примера сделаем запрос, чтобы получить информацию о фильмах:

```
async function fetchMovies() {
  const response = await fetch('/movies');
  // waits until the request completes...
  console.log(response);
}
```

Объект **Response** имеет некоторые свойства, с помощью которых можно обращаться к некоторым данным ответа

Response.status– читает ответ и возвращает как обычный текст,

Response.statusText – возвращает ответ как объект

Response.ok – булевское значение, которое указывает, выполнился ли запрос успешно или нет

Response.headers – объект, который описывает заголовок ответа.

Существует метод, который помогает извлечь из объекта **Response** информацию ответа в **JSON** формате:

Response.json()

Этот метод тоже возвращает **promise**, поэтому его необходимо дожидаться при помощи **await**:

```
async function f() {
    let url = 'https://api.github.com/repos/javascript-tutorial/commits';
    let response = await fetch(url);
    let commits = await response.json(); // читаем ответ в формате JSON
    alert(commits[0].author);
}
```

Работа с ошибками

Промис завершается с ошибкой, если fetch не смог выполнить HTTP-запрос, например при ошибке сети или если нет такого сайта. HTTP-статусы 404 и 500 (и другие) не являются ошибкой.

```
let response = await fetch(url);

if (response.ok) { // если HTTP-статус в диапазоне 200-299
  // получаем тело ответа (см. про этот метод ниже)
  let json = await response.json();
} else {
  alert("Ошибка HTTP: " + response.status);
}
```

Метод GET

Метод GET используется для получения данных с сервера.

```
const response = await fetch('https://api.example.com/data', {
  method: 'GET',
  headers: {
    'Content-Type': 'application/json',
    // Дополнительные заголовки, если необходимо
  },
  // Дополнительные параметры запроса, например, параметры URL
  // Также можно использовать URLSearchParams для удобной работы с параметрами
});

const data = await response.json();
console.log('GET data:', data);
```

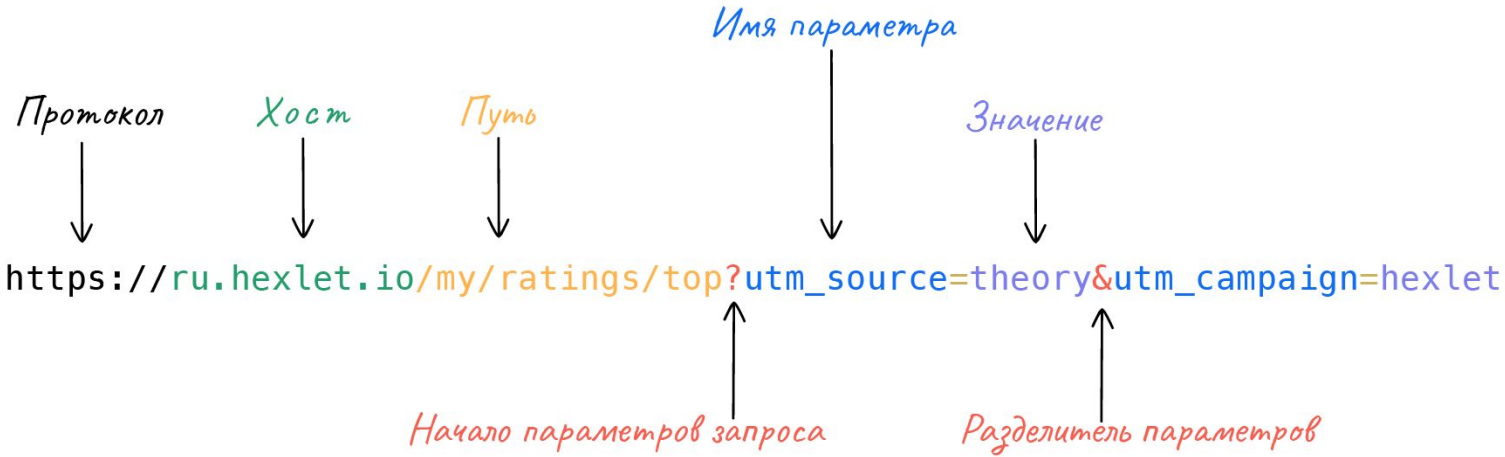
Для метода GET можно не указывать options

```
const response = await fetch('https://api.example.com/data');
```

Метод GET

GET-запросы, по определению, не предназначены для отправки данных на сервер.

Однако, иногда может потребоваться отправить данные в URL в виде параметров запроса.



Метод POST

Метод **POST** отправляет данные на сервер и создает новый ресурс

```
const response = await fetch('https://api.example.com/data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    // Дополнительные заголовки, если необходимо
  },
  body: JSON.stringify({
    key1: 'value1',
    key2: 'value2',
    // Данные для отправки на сервер
  }),
});

const data = await response.json();
console.log('POST data:', data);
```

Метод PUT

Метод **PUT** чаще всего используется для обновления существующего ресурса. Для этого необходим URL ресурса и новая его версия.

```
const response = await fetch('https://api.example.com/data/123', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
    // Дополнительные заголовки, если необходимо
  },
  body: JSON.stringify({
    key1: 'updatedValue1',
    key2: 'updatedValue2',
    // Обновленные данные для отправки на сервер
  }),
});

const data = await response.json();
console.log('PUT data:', data);
```

Метод DELETE

Метод **DELETE** используется для удаления ресурса, который указывается с помощью его URL.

```
const response = await fetch('https://api.example.com/data/123', {
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
    // Дополнительные заголовки, если необходимо
  },
});

const data = await response.json();
console.log('DELETE data:', data);
```

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team