1. Дайте определение понятию «Интернет».

**!ИНТЕРНЕТ – ВСЕМИРНАЯ.!:**

* **сеть, в основе которой лежит TCP/IP**
* **некоторое количество организаций, которые поддерживают эту сеть**
* **документация интернет**
* **службы интернет(DNS -** система для преобразования доменных имен в IP-адрес**, WWW, E-mail (STMP,POP3, IMAP), IRC(обмен сообщений в реальном времени), FTP, Telnet (управление удаленным компьютером в терминальном режиме))**

1. Дайте определение понятию «Служба Интернет».

**интернет-служба - один из видов интернет-ресурса (сервер, который имеет IP-адрес, порт в сети, WWW). Протокол + сервер**

1. Дайте определение понятию «Узел сети Интернет».

**интернет-ресурс - любая сущность, которая в интернете имеет адрес (опубликованная в интернет)**

1. Дайте определение понятию «клиент-серверное приложение».

программа, состоящее из двух компонент — клиента и сервера. Клиент и сервер взаимодействуют между собой в соответствии с заданными правилами (протоколами). Инициатором соединения всегда является клиент.

1. Дайте определение понятию «сетевой протокол».

Набор правил и стандартов, которые позволяют осуществить соедиение и обмен данными между 2 и более устройствами в сети.

1. Перечислите основные свойства протокола HTTP.

**Свойства HTTP: (RFC2616)**

* 2 типа абонентов (клиент и сервер)
* 2 типа сообщений : Request и Response.
* более распростр версия 1.1, но есть и 1.2 и 1.3, САМАЯ ПЕРВАЯ 0.9, есть HTTP 2(не распространен, бинарный) и 3 (5% прилож ток поддерживает эту версию)
* на 1 Request 1 Response и наоборот, иначе ошибка
* порты 80(HTTP) и 443 (HTTPS)
* для адресации используется URI
* поддерживается организацией W3C и описан в нескольких RFC
* stateless – не сохраняет состояние

1. Перечислите состав информации, пересылаемой в HTTP-запросе.

**Структура запроса – серверный объект, кот образ в рез обработки сервером http-запроса, поступающего от клиента и передается серверному программному коду для обработки:**

* метод(get для извлечения инфы, post для отправки данных на сервер, put для изменения данных на сервере, delete для удаления);
* URI, по которому идет обращение к серверу;
* версия протокола (HTTP/1.1);
* заголовки (пары: имя/значение): которые описывают протокол HTTP(нужны для управления протоколом HTTP, управляющих сигналов между клиентом и сервером);

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

* параметры (пары: имя/значение) могут находится в 2ух местах: в самом URI (?a=амперсанд(&)b=), в теле запроса;



* расширение = тело запроса(может находиться инфа, соответств майну(описывает то, что находится в теле запроса))

1. Перечислите состав информации, пересылаемой в HTTP-ответе.

**Структура ответа- серверный объект, который автоматически формируется сервером, при получении http-запроса (одновременно с объектом Request), заполняется данными серверными программным кодом, преобразуется в http-ответ и отправляется клиенту:**

* версия протокола (HTTP/1.1);
* код состояния (1xx, 2xx, 3xx, 4xx, 5xx) (ответ, который указывает как завершилась обработка запроса);
* пояснение к коду состояния (небольшое символьное сообщение);
* заголовки (пары: имя/заголовок);
* расширение = тело ответа.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Контент, сгенерированный ИИ, может содержать ошибки.

**ТИПЫ ЗАГОЛОВКОВ**

* **General**: общие заголовки, используются в запросах и ответах; (**Cache-Control**, **Connection**, **Date**, **Pragma**, **Warning**, Via, **Upgrade**, Transfer-Enconding, Trailer)
* **Request**: используются только в запросах; (**Accept**, Accept-Charset, **Accept-Encoding**, **Accept-Language**, **Host**, **Authorization**, Expect, From, If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, **User-Agent**, Range, TE, **Referer**, **Proxy-Authorization**, Max-Forwards)
* **Response**: используются только в ответах; (Accept-Ranges, Age, ETag, **Location**, **Proxv-Authenticate**, Retry-After, **Server**, Vary, **WWW-Authenticate**)
* **Entity**: для сущности в ответах и запросах. (Allow, **Content-Encoding**, **Content-Language**, **Content-Length**, **Content-Location**, Content-MD5, Content-Range, **Content-Type(text/html, application/png)**, Expires, Last-Modified, extension-header), что находится в расширении запроса
* **Пользовательский**

1. Дайте определение понятию «web-приложение».

**!Web-приложение! - клиент-серверное приложение, у которого клиент и сервер взаимодействуют по протоколу HTTP.**

1. Дайте определение понятиям «frontend» и «backend».

* **Frontend** - часть веб-приложения, с которой взаимодействует пользователь. Это включает в себя пользовательский интерфейс, который отображается в браузере, и клиентскую логику, которая выполняется на стороне клиента (например, JavaScript).
* **Backend** - серверная часть веб-приложения, которая обрабатывает запросы от клиентов, выполняет бизнес-логику, взаимодействует с базами данных и отправляет данные клиентам. Backend обычно работает на сервере.

1. Дайте определение понятию «кроссплатформенное приложение».

Приложение, которое может работать на разных ОС или платформах без значительных изменений.

1. Изобразите и поясните общую схему web-приложения.



1. Назовите основные технологии разработки серверных кроссплатформенных приложений.

**Технологии программирования в Internet:**

* LAMP (Linux, Apache, MySQL, PHP/Perl, кроссплатформенная технология);
* Java EE (кроссплатформенная технология на основе Java);
* ASP.NET (ASP.NET CORE – кроссплатформенная версия);
* NODE.JS (кроссплатформенная технология, на основе JavaScript);
* Ruby on Rails(кроссплатформенная технология на основе Ruby и фреймворка MVC для web-разработки);
* Python Django (кроссплатформенная технология на основе Python и фреймворка MVC для web-разработки).

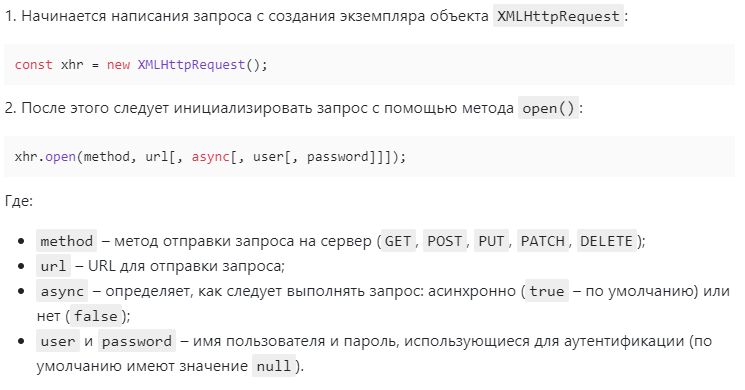
1. Поясните понятие «асинхронная операция».

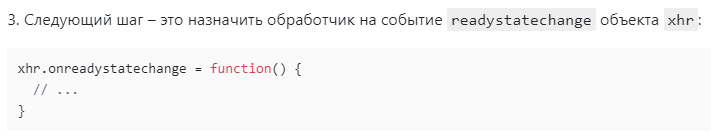
Операция, в которой выполнение осуществляется в 2 фазы: заявка на исполнение и получение результата.

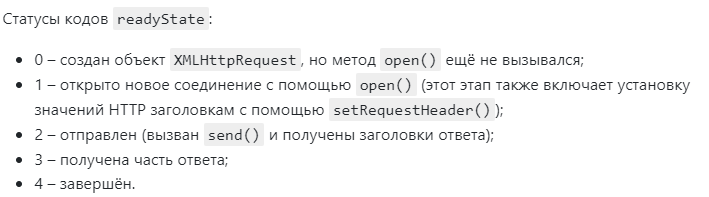
1. Поясните принцип выполнения асинхронного запроса с помощью объекта XMLHTTPRequest и Fetch.

Оба способа позволяют делать запрос, но fetch возвращает Promise, что позволяет избавиться от большого количества коллбэков.

**Для XHR:**





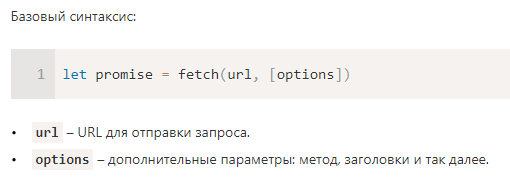








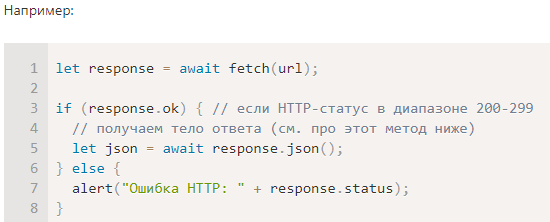
**Для fetch:**

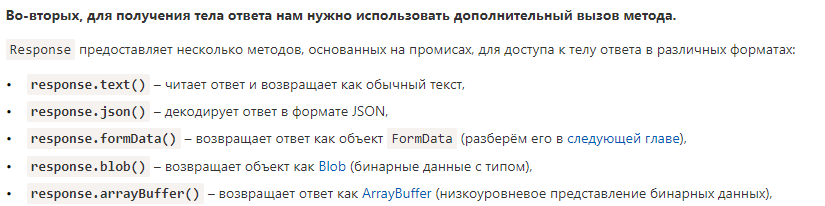
****

****

****

****

****

****

### **Асинхронные запросы:** XMLHttpRequest **vs** fetch

Асинхронные запросы позволяют отправлять HTTP-запросы на сервер без перезагрузки страницы. В JavaScript есть два основных способа выполнения таких запросов: **XMLHttpRequest (XHR)** и **fetch**.

## **1.** XMLHttpRequest **(устаревший способ)**

Объект **XMLHttpRequest** предоставляет низкоуровневый интерфейс для выполнения HTTP-запросов.

### **Принцип работы** XMLHttpRequest**:**

1. Создаётся объект XMLHttpRequest.
2. Открывается соединение с сервером методом .open().
3. Отправляется запрос с помощью .send().
4. Когда сервер отвечает, срабатывает обработчик onreadystatechange или onload, который получает ответ.

### **Пример** XMLHttpRequest**:**

const xhr = new XMLHttpRequest();

xhr.open("GET", "https://jsonplaceholder.typicode.com/todos/1", true);

xhr.onreadystatechange = function () {

if (xhr.readyState === 4 && xhr.status === 200) {

console.log(JSON.parse(xhr.responseText));

}

};

xhr.send();

### **Проблемы** XMLHttpRequest**:**

* Код сложен в управлении (callback hell).
* Требуется больше кода для обработки ошибок.
* Интерфейс устарел (заменён fetch).

## **2.** fetch **(современный стандарт)**

fetch — это новая, более удобная альтернатива XMLHttpRequest, основанная на промисах.

### **Принцип работы** fetch**:**

1. Вызывается функция fetch(url, options).
2. Возвращается **промис**, который разрешается, когда сервер отвечает.
3. Ответ преобразуется в нужный формат (JSON, text и т. д.).

### **Пример** fetch**:**

fetch("https://jsonplaceholder.typicode.com/todos/1")

.then(response => {

if (!response.ok) {

throw new Error("Ошибка сети: " + response.status);

}

return response.json();

})

.then(data => console.log(data))

.catch(error => console.error("Ошибка:", error));

### **Преимущества** fetch**:**

✅ Гораздо проще и понятнее, чем XMLHttpRequest.  
✅ Основан на промисах, что делает код более читаемым.  
✅ Легко использовать async/await.

**Пример fetch с async/await:**

async function fetchData() {

try {

let response = await fetch("https://jsonplaceholder.typicode.com/todos/1");

if (!response.ok) {

throw new Error("Ошибка сети");

}

let data = await response.json();

console.log(data);

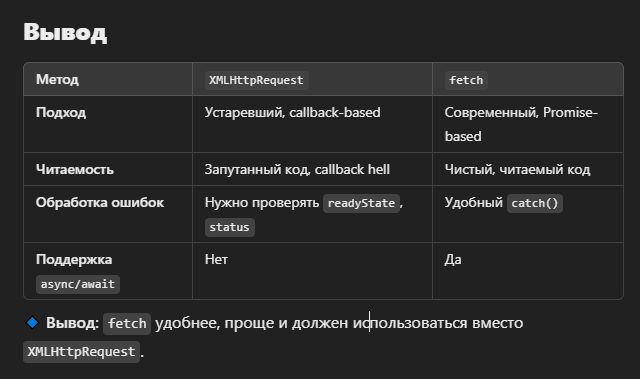
} catch (error) {

console.error("Ошибка:", error);

}

}

fetchData();

****

1. Поясните основное назначение сервера NODE.JS.

Это облегченная среда, используемая для разработки веб-приложений на стороне сервера.

1. Перечислите основные свойства сервера NODE.JS.

* основан на **Chrome V8**;
* **среда (контейнер) исполнения** приложений на JavaScript;
* поддерживает механизм **асинхронности**;
* ориентирован на **события**;
* **однопоточный** (код приложения исполняется только в одном потоке, один стек вызовов); обычно в серверах для каждого соединения создается свой поток, в Node.js все соединения обрабатываются в одном JS-потоке;
* **не блокирует** выполнение кода при вводе/выводе (в файловой системе до 4х одновременно);
* в состав Node.js входят инструменты: **npm** – пакетный менеджер; **gyp** - Python-генератор проектов; **gtest** – Google фреймворк для тестирования С++ приложений;
* использует библиотеки: **V8** – библиотека V8 Engine, **libuv** – библиотека для абстрагирования неблокирующих операций ввода/вывода (представляет собой обертку над epoll, kqueue, IOCP); **llhttp** – легковесный парсер http-сообщений (написан на C и не выполняет никаких системных вызовов); **c-ares** -библиотека для работы с DNS; **OpenSSL** – библиотека для криптографии; **zlib** – сжатие и распаковка.