1. **Дайте определение понятию «Интернет».**

Всемирная компьютерная сеть, построенная на основе стека протоколов TCP/IP.

Сеть Интернет: 1) сеть на основе TCP/IP; 2) стандарты Internet (RFC, STD); 3) службы Интернет (DNS, SMTP/POP3/IMAP, WWW, FTP, Telnet, SSH,…); 4) организации, управляющие сетью Internet (ISOC, IETF, ICANN, IANA, IAB, …).

Служба WWW — это служба интернета, которая позволяет просматривать веб-страницы через браузер.

ISOC: Internet Society – международная организация (офисы США, Швейцария), занимающаяся развитием сети Internet. ISOC владелец RFC-стандартов. ISOC обеспечивает правовую поддержку и финансирует все другие организации, связанные с деятельностью Internet (IETF, IAB,…).

IETF: Internet Engineering Task Force - рабочая группа проектирования Internet. Публикует RFC (Request for Comments – заявка на отзывы = тема для обсуждения).

ICANN: Internet Corporation for Assigned Names and Numbers – корпорация по управлению доменными именами и IP-адресами.

IANA: Internet Assigned Numbers Authority – Администрация адресного пространства Internet.

1. **Дайте определение понятию «Служба Интернет».**

Internet-служба: один из видов Internet-ресурса, имеющий специальное предназначение (DNS, WWW, E-mail, FTP, ICQ, Telnet).

1. **Дайте определение понятию «Узел сети Интернет».**

узел Интернет: устройство, имеющее IP-адрес и подключенное сети Интернет (обычно к сети Интернет-провайдера). Каждый узел характеризуется своей программно-аппаратной платформой – аппаратурой и операционной системой.

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

клиент-серверное приложение = приложение (программа) с клиент-серверной архитектурой: приложение, состоящее из двух компонент – клиента и сервера; клиент и сервер взаимодействуют между собой в соответствии с заданными правилами (спецификациями, протоколами); для взаимодействия между клиентом и сервером в соответствии с правилами (спецификацией, протоколом) должно быть установлено соединение; инициатором соединения всегда является клиент.

1. **Дайте определение понятию «сетевой протокол».**

Сетевой протокол — это набор правил, определяющий принципы взаимодействия устройств в сети. Чтобы отправка и получение информации прошли успешно, все устройства-участники процесса должны принимать условия протокола и следовать им.

Он обеспечивает стандартизированный способ передачи информации, чтобы разные устройства могли понимать друг друга, даже если они используют разное аппаратное или программное обеспечение.

1. **Перечислите основные свойства протокола HTTP.**

* версии HTTP/1.1 – действующий (текстовый), HTTP/2 – черновой (не распространен, бинарный);
* два типа абонентов: клиент и сервер;
* два типа сообщений: request и response;
* от клиента к серверу – request;
* от сервера к клиенту – response;
* на один request всегда один response, иначе ошибка;
* одному response всегда один request, иначе ошибка;
* TCP-порты: 80, 443;
* для адресации используется URI или URN;
* поддерживается W3C, описан в нескольких RFC.

W3C (World Wide Web Consortium) — это международная организация, которая разрабатывает и поддерживает стандарты для веб-технологий

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

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. Стартовая строка (англ. Starting line) — определяет тип сообщения;

2. Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;

3. Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа.

Стартовые строки различаются для запроса и ответа.

Строка запроса выглядит так:

Метод URI HTTP/Версия

* Метод (англ. Method) — название запроса, одно слово заглавными буквами.
* URI определяет путь к запрашиваемому документу.
* Версия (англ. Version) — пара разделённых точкой арабских цифр. Например: 1.0.

 GET URI — для версии протокола 0.9.

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

Ответ(Response): 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. Содержит: всю информацию, которая должна быть помещена в http-ответ: статус, коллекция заголовков, поток данных, …

Стартовая строка ответа сервера имеет следующий формат:

HTTP/Версия КодСостояния Пояснение

* Версия — пара разделённых точкой арабских цифр как в запросе.
* КодСостояния (англ. Status Code) — три арабские цифры. По коду статуса определяется дальнейшее содержимое сообщения и поведение клиента.
* Пояснение (англ. Reason Phrase) — текстовое короткое пояснение к коду ответа для пользователя.

1. **Дайте определение понятию «web-приложение».**

Web-приложение: клиент-серверное приложение, применяющее для обмена данными протокол HTTP;

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

когда говорят о разработке web-приложения, говорят о разработке frontend (клиента) и backend (сервера).

1. Frontend (клиентская часть) — это часть приложения, с которой взаимодействует пользователь. Она включает в себя все элементы интерфейса, такие как кнопки, формы, меню, и любые другие визуальные компоненты, отображаемые в браузере. Frontend также отвечает за обработку пользовательских действий и передачу запросов на сервер. Основными технологиями для frontend являются HTML, CSS, JavaScript, а также фреймворки и библиотеки, такие как React, Angular, Vue.js.
2. Backend (серверная часть) — это часть приложения, которая работает на сервере и отвечает за обработку данных, бизнес-логику, взаимодействие с базой данных, а также за выполнение запросов от клиента. Backend принимает данные от клиента, обрабатывает их и отправляет результаты обратно. Для разработки backend используют различные языки программирования и фреймворки, такие как Node.js, Python (Django, Flask), Java (Spring), Ruby (Rails), PHP. Также в backend обычно включены базы данных (например, MySQL, PostgreSQL, MongoDB), которые хранят информацию.
3. **Дайте определение понятию «кроссплатформенное приложение».**

кроссплатформенное приложение: приложение, способное работать на более чем одной программно-аппаратной (аппаратура + операционная система) платформе. Кроссплатформенность может быть достигнута различными способами: 1) на уровне компилятора (С, С++); 2) на уровне среды (фреймворка) исполнения (Java/JVM, C#/.NET CORE, JS/Node.js/V8).

1. **Изобразите и поясните общую схему web-приложения.**















1. **Назовите основные технологии разработки серверных кроссплатформенных приложений.**

технологии для разработки кроссплатформенных web-серверов:

- PHP/Apache, LAMP;

- Java/JVM/Application Server;

- C#/ASP.NET CORE;

- Python/Django;

- Ruby on Rails;

- JS/Node.js, ….

1. **Поясните понятие «асинхронная операция».**

операция называется асинхронной, если ее выполнение осуществляется в 2 фазы: 1) заявка на исполнение; 2) получение результата; при этом участвуют два механизма: A-механизм, формирующий заявку и потом получающий результат; B-механизм, получающий заявку от A, исполняющий операцию и отправляющий результат A; продолжительность исполнения операции B-механизмом, как правило, непредсказуемо; в то время пока B-механизм исполняет операцию, А-механизм выполняет собственную работу. Применение асинхронности не противоречит применению многопоточности.

1. **Поясните принцип выполнения асинхронного запроса с помощью объекта XMLHTTPRequest и Fetch.**
2. **Поясните основное назначение сервера NODE.JS.**

NODEJS: программная платформа для разработки серверных web-приложений на языке JS/2.

1. **Перечислите основные свойства сервера NODE.JS**

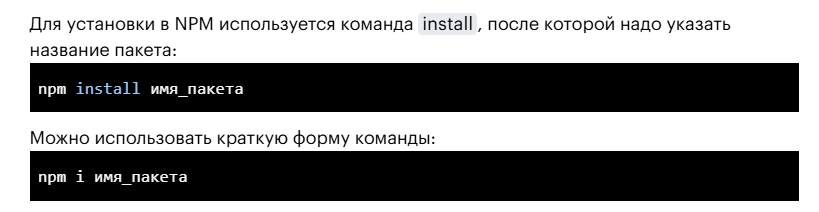
* основан на Chrome V8;
* среда (контейнер) исполнения приложений на JavaScript;
* поддерживает механизм асинхронности;
* ориентирован на события;
* однопоточный (код приложения исполняется только в одном потоке, один стек вызовов); обычно в серверах для каждого соединения создается свой поток, в Node.js 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 – асинхронный ввод/вывод; http-parser – легковесный парсер http-сообщений; c-ares - библиотека для работы с DNS; OpenSSL – библиотека для криптографии; zlib – сжатие и распаковка.

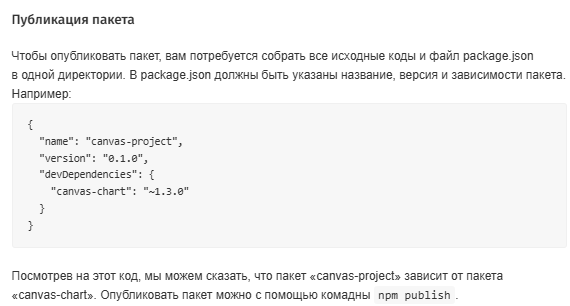
1. **Что такое npm?**

npm (аббр. node package manager) — это стандартный менеджер пакетов, автоматически устанавливающийся вместе с Node.js. Он используется для скачивания пакетов из облачного сервера npm, либо для загрузки пакетов на эти сервера.

Вкратце, пакетом в Node.js называется один или несколько JavaScript-файлов, представляющих собой какую-то библиотеку или инструмент.

В экосистеме Node.js пакеты устанавливаются с помощью менеджера NPM, а хранятся они в публичных репозиториях. Это значит, что любой желающий может изучить их код и принять участие в разработке.





1. **Поясните назначение HTTP-заголовка Content-Type.**

Заголовок-сущность Content-Type используется для того, чтобы определить [MIME тип](https://developer.mozilla.org/ru/docs/Glossary/MIME_type) ресурса.

MIME-тип (называемый "media type", а иногда "content type") - это строка, отправляемая вместе с файлом, которая указывает тип файла. (например, передаваемый аудиофайл может быть помечен как audio/ogg тип, а изображение - image/png). MIME-тип играет точно такую же роль, как и расширение файла в системе Windows. Когда HTTP-сообщение содержит Content-type заголовок, тело запроса будет парситься в соответствии с MIME-типом, указанным в заголовке.

В некоторых случаях браузеры пытаются сами определить MIME тип передаваемого контента, но их реакция может быть неадекватной.

1. **text/plain** — текст в обычном (неформатированном) виде.
2. **text/html** — HTML-документ.
3. **application/json** — данные в формате JSON.
4. **application/xml** — данные в формате XML.
5. **application/x-www-form-urlencoded** — данные, передаваемые в виде пар "ключ-значение", обычно используется при отправке формы с методом POST.
   * Пример: name=John&age=30.
6. **application/javascript** — JavaScript-файл.
7. **image/jpeg** — изображение в формате JPEG.
8. **image/png** — изображение в формате PNG.
9. **audio/mpeg** — аудиофайл в формате MPEG (MP3).
10. **video/mp4** — видеофайл в формате MP4.
11. **application/pdf** — PDF-документ.
12. **application/zip** — архивный файл в формате ZIP.
13. **Поясните назначение функции require.**

require в Node.js — это встроенная функция, предназначенная для импортирования модулей. С ее помощью можно загрузить объекты, экспортируемые из модулей, чтобы затем использовать их функции и свойства.

1. **Поясните понятие «Модуль Node.js».**

Модуль представляет блок кода, который может использоваться повторно в других модулях. При необходимости мы можем подключать нужные нам модули.

Вообще в мире JavaScript есть различные системы модулей. По умолчанию Node.js применяет систему модулей CommonJS, которая рассматривает отдельный файл как модуль, для загрузки модулей применяет функцию require(), в которую передается название модуля. После получения модуля мы сможем использовать весь определенный в нем функционал.

1. **Поясните понятие «Node.js built-in modules» («Node.js Core modules»).**

Node.js built-in modules (или Node.js Core modules) — это набор модулей, которые поставляются вместе с Node.js и предоставляют основные функциональные возможности для работы с файловой системой, сетью, буферами, потоками и другими важными аспектами серверной разработки. Эти модули уже включены в сам Node.js, и их не нужно устанавливать через npm — они доступны сразу после установки Node.js.

* fs (File System) — для работы с файловой системой. С его помощью можно читать, записывать, удалять файлы, создавать каталоги и выполнять другие операции с файлами.
* http — для создания веб-серверов и отправки HTTP-запросов. Этот модуль используется для создания серверов, обработки HTTP-запросов и ответов.
* path — для работы с путями файлов и директорий. Модуль помогает манипулировать путями, например, объединять части пути или извлекать расширения файлов.
* url — для парсинга и манипулирования URL. Модуль полезен для обработки адресов веб-страниц, разбиения URL на части и их модификации.
* events — для работы с событийной моделью. Модуль предоставляет класс EventEmitter, который позволяет создавать объекты, способные генерировать и обрабатывать события.
* buffer — для работы с бинарными данными. Модуль позволяет работать с буферами, которые представляют собой массивы байтов, и используется, например, для обработки данных в двоичном формате.
* timers — для работы с таймерами и интервальными функциями (например, setTimeout, setInterval).

1. **Какой модуль NODE.JS обеспечивает работу с протоколом HTTP?**

Модуль http в Node.js обеспечивает работу с протоколом HTTP. Он используется для создания веб-серверов, обработки HTTP-запросов и отправки HTTP-ответов.

1. **Какой модуль NODE.JS обеспечивает работу с файловой системой?**

Модуль fs (File System) в Node.js обеспечивает работу с файловой системой. С его помощью можно читать, записывать, удалять файлы и выполнять другие операции с файлами и директориями.

**Метод createServer** используется для создания HTTP-сервера, который будет обрабатывать входящие запросы и отправлять ответы. Этот метод принимает в качестве аргумента функцию, которая будет вызвана для каждого запроса.  Эта функция принимает два параметра:

* **req** — объект запроса, который содержит информацию о входящем HTTP-запросе, например, URL, заголовки и т.д.
* **res** — объект ответа, через который можно отправить ответ на запрос. С его помощью устанавливаются заголовки и тело ответа.

Метод **writeHead** используется для установки **кода статуса** ответа и **заголовков**. Он отправляет заголовки HTTP-ответа клиенту.

* **statusCode** — это код состояния ответа (например, 200 для успешного запроса, 404 для "не найдено" и так далее).
* **headers** — это объект, содержащий заголовки ответа в виде **ключ-значение**.

Метод **end** завершает отправку ответа. Он может опционально принимать **данные**, которые будут отправлены как тело ответа. После вызова **end** сервер завершает обработку и отправляет ответ клиенту.

* Если вы не передаете аргумент в end(), ответ будет отправлен пустым.
* Если передаете аргумент, например строку или буфер, он будет отправлен как тело ответа.
* Невозможность отправки данных после end(): Когда вы вызываете end(), это означает, что ответ завершен, и данные больше не могут быть отправлены. Если вы пытаетесь вызвать write() или end() снова после того, как ответ был завершен, это приведет к ошибке.

Метод **write()** используется для добавления данных в тело ответа **по частям**. Он позволяет отправить данные клиенту частями, и можно вызывать его несколько раз до того, как вызовете **end()** для завершения ответа. Метод **write()** в Node.js **записывает данные в тело ответа**, но **не отправляет их сразу**. Данные отправляются клиенту только когда вы вызываете метод **end()**.

Метод **listen()** в Node.js используется для того, чтобы **сервер начал прослушивать входящие соединения на указанном порту**. Это означает, что сервер готов принимать запросы от клиентов (например, браузеров или других приложений), которые подключаются к серверу через определённый порт.

server.listen(port, [hostname], [backlog], [callback]);

* **port** — номер порта, на котором сервер будет слушать входящие соединения.
* **hostname** (необязательный) — имя хоста, на котором сервер будет слушать. По умолчанию это 'localhost'.
* **backlog** (необязательный) — максимальное количество соединений, которые могут ожидать в очереди до того, как сервер начнёт их обрабатывать. Если параметр не указан, используется значение по умолчанию.
* **callback** (необязательный) — функция, которая будет вызвана, когда сервер успешно начнёт прослушивать соединения.

Метод **request.on()** в Node.js используется для обработки событий, связанных с объектом **request**.

Полученная информация при больших объёмах может разбиваться на чанки (фрагменты).

1. **data** событие получения фрагмента.
2. **end** событие получения последнего фрагмента.

Если у нас один фрагмент: сработает data и end. Если у нас два фрагмента: сработает два раза data и один раз end.

Метод **fs.readFileSync()** в Node.js используется для **синхронного чтения файлов**. Он считывает содержимое файла и возвращает его в виде буфера или строки, в зависимости от указанной кодировки. **Синхронное** означает, что выполнение программы будет приостановлено до тех пор, пока файл не будет полностью прочитан.

**XMLHttpRequest**– это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.

**Чтобы сделать запрос, нам нужно выполнить три шага:**

**1. Создать XMLHttpRequest.**

2. Инициализировать его.

xhr.open(method, URL, [async, user, password])

Заметим, что вызов open, вопреки своему названию, не открывает соединение. Он лишь конфигурирует запрос, но непосредственно отсылается запрос только лишь после вызова send.

**3. Послать запрос. xhr.send([body]). Этот метод устанавливает соединение и отсылает запрос к серверу. Необязательный параметр body содержит тело запроса.**

4. Слушать события на xhr, чтобы получить ответ.

Три наиболее используемых события:

* load – происходит, когда получен какой-либо ответ, включая ответы с HTTP-ошибкой, например 404.
* error – когда запрос не может быть выполнен, например, нет соединения или невалидный URL.
* progress – происходит периодически во время загрузки ответа, сообщает о прогрессе.

Метод fetch() — современный и очень мощный. Он не поддерживается старыми (можно использовать полифил), но поддерживается всеми современными браузерами.

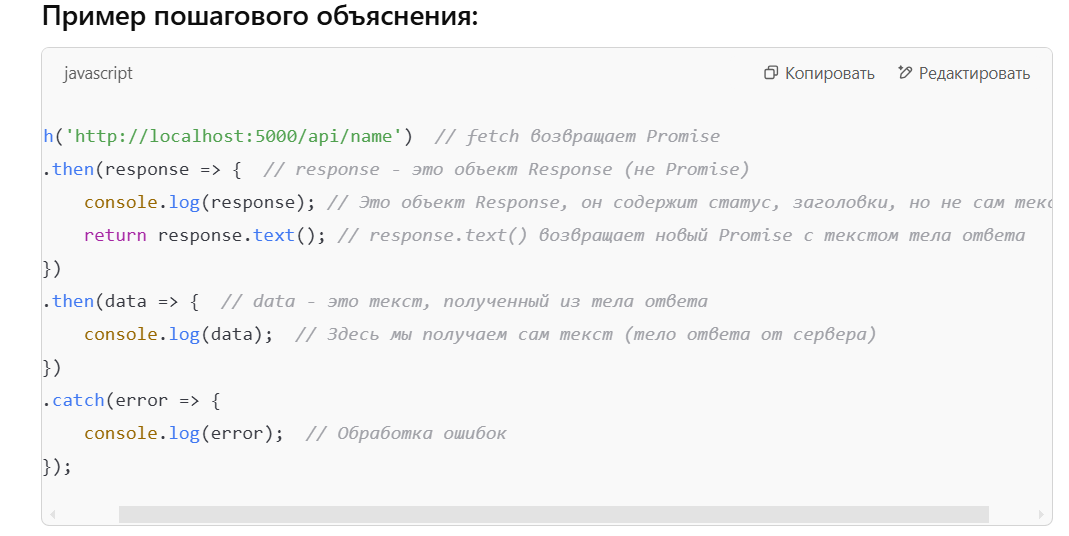
Promise – это специальный объект, который содержит своё состояние. Вначале pending («ожидание»), затем – одно из: fulfilled («выполнено успешно») или rejected («выполнено с ошибкой»).

Если вы передаете два аргумента в **then()**, то первый аргумент будет обработчиком успешного результата, а второй — для ошибок.

response.text() тоже возвращает промис, который будет завершён, когда тело ответа будет полностью прочитано как текст.

Если при выполнении запроса произошла ошибка (например, сервер не доступен или произошёл сетевой сбой), то ошибка будет поймана в блоке .catch().

Так как then() всегда возвращает новый промис, его удобно использовать для построения последовательностей асинхронных операций.



**AJAX (Asynchronous JavaScript and XML)** — это технология для асинхронных запросов к серверу без перезагрузки всей страницы. С помощью AJAX можно отправлять данные на сервер и получать их в ответ, обновляя только часть веб-страницы, а не перезагружая её целиком.

