1. **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.

Пакет - это дерево папок, описываемое файлом package.json. Пакет состоит из папки, содержащей файл package.json (описывает метаданные пакета, его зависимости и скрипты), и всех вложенных папок до следующей папки, содержащей другой файл package.json, или папки с именем node\_modules.

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.

Для начала работы с npm нужно инициализировать проект.

npm init

Для установки пакета используйте команду npm install. Например, установим пакет lodash.

npm install lodash

После установки пакета его можно импортировать и использовать в своем коде. Пример использования пакета lodash:

const lod = require('lodash');

Для обновления пакетов до последней версии используйте команду npm update.

npm update lodash

Чтобы удалить пакет, используйте команду npm uninstall

npm uninstall lodash

Чтобы опубликовать пакет, вам потребуется собрать все исходные коды и файл package.json в одной директории. В package.json должны быть указаны название, версия и зависимости пакета. Опубликовать пакет можно с помощью комадны npm publish.

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.

Кроме CommonJS существует ECMAScript.

Про CommonJS:

* + - 1. **Он полностью синхронный.** Это значит, что загрузка модулей выполняется **в момент вызова** require() и блокирует выполнение кода, пока модуль не будет загружен.
      2. Его можно "патчить по-обезьяньи" (Monkey Patching) — это изменение поведения существующего модуля в процессе работы. Это возможно, потому что CommonJS возвращает объект, который можно изменять во время выполнения.
      3. Если ты загружаешь .json, Node.js автоматически парсит его в объект.
      4. Любые файлы без расширений .json или .node считаются **JavaScript-кодом**.
      5. Файлы с расширением .node — это скомпилированные бинарные модули (написанные на C/C++), которые загружаются через process.dlopen().
      6. require() не может загрузить ESM (import/export), но ESM можно загрузить динамически из CommonJS через import()
      7. Когда указываешь путь к модулю, но не указываешь расширение, загрузчик пробует разные варианты: без расширения, .js, .json, .node

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.

**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.
* onreadystatechange – когда изменяется состояние запроса readyState
* и тд

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

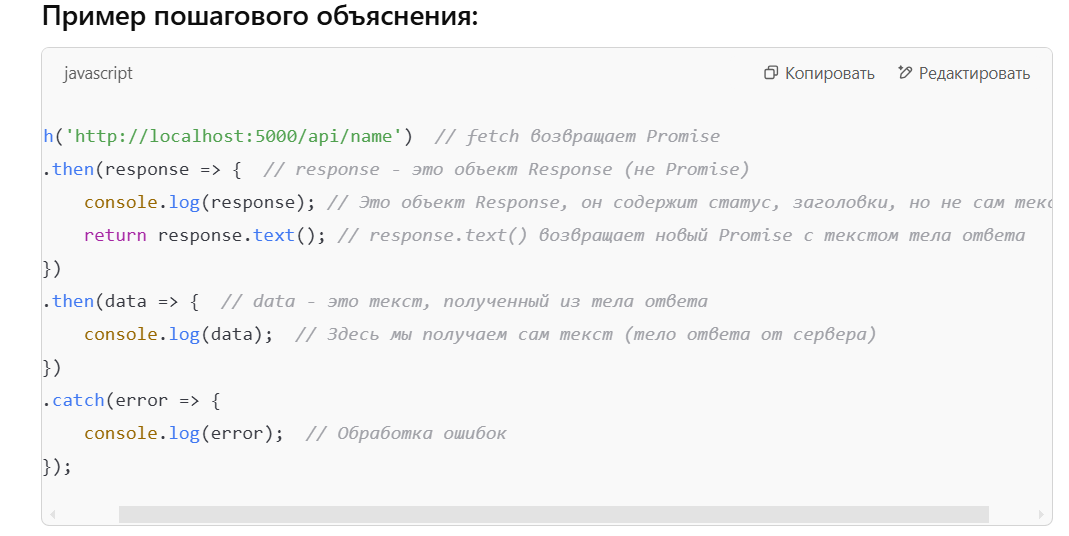
Promise – это специальный объект, который содержит своё состояние. Вначале pending («ожидание»), затем – одно из: fulfilled («выполнено успешно») или rejected («выполнено с ошибкой»).

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.

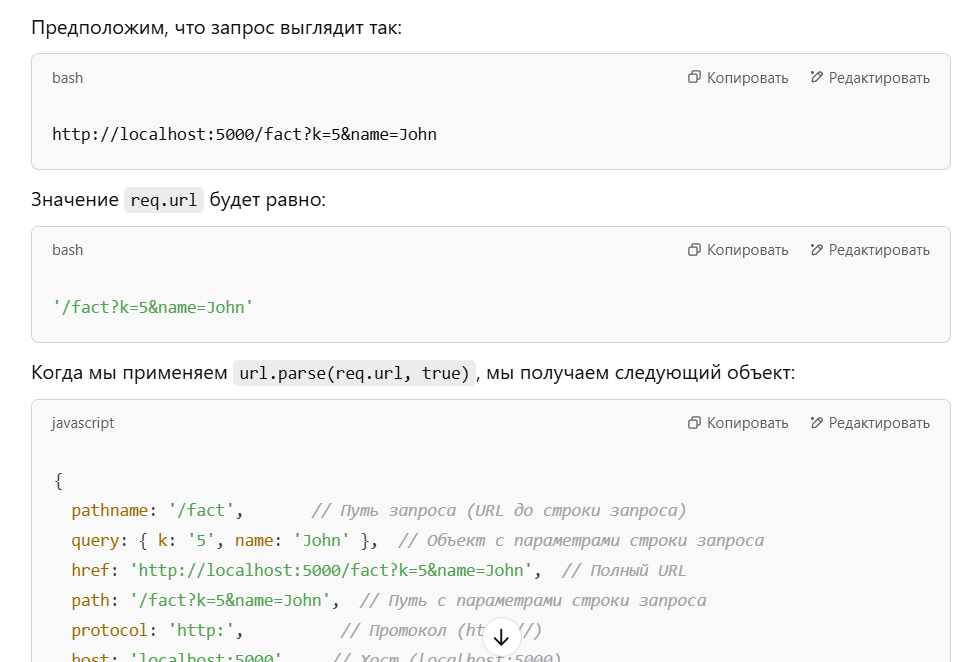


**AJAX (Asynchronous JavaScript and XML)** — это технология для асинхронных запросов к серверу без перезагрузки всей страницы. С помощью AJAX можно отправлять данные на сервер и получать их в ответ, обновляя только часть веб-страницы, а не перезагружая её целиком.

**url.parse(req.url, true)**

Метод url.parse() в Node.js анализирует (парсит) строку URL и возвращает объект, содержащий различные компоненты URL, такие как путь, хост, параметры строки запроса и другие.

Когда мы передаем true в качестве второго аргумента, это означает, что параметры строки запроса (query) будут разобраны в виде объекта. Вот что возвращает url.parse(req.url, true):



**JSON.stringify()** — это стандартная функция JavaScript, которая преобразует объект JavaScript в строку в формате JSON (JavaScript Object Notation).

# Про EventLoop

Дело в том, что Event Loop - единственный механизм в JavaScript, который позволяет реализовать асинхронность. Event Loop является стеком, где хранятся все задачи, которые не вошли в синхронный поток выполнения. После завершения синхронного потока - задачи начинают выполняться из Event Loop'а. Однако у Event Loop'а тоже есть свои правила. Он делит все задачи на подтипы:

* Микрозадачи: промисы (причём именно then, коллбэк внутри конструктора промиса является **синхронным,** process.nextTick() который имеет даже более высокий приоритет чем обычные микротаски, async/await)
* Макрозадачи (setTimeout(), setInterval(), setImmediate(), I/O операции например, fs.readFile(), addEventListener())
* Задачи отрисовки(Рендеринг изменений DOM, Пересчет стилей и макета)

1. Сначала Event Loop проверяет выполнились ли все синхронные задачи
2. Потом выполняются **все задачи** из микротасков
3. После выполнения всех микротасков - очередь очищается
4. Затем мы берем **одну** макрозадачу из списка и выполняем ее
5. После выполнения мы смотрим нужно ли нам сделать перерисовку страницы
6. Если перерисовать страницу нужно - делаем это
7. Все снова начинается с первого пункта :)
8. **Перечислите основные свойства глобальные объекты Node.js и поясните их предназначение.**

Глобальные объекты: global, process, buffer

global – это объект, который содержит глобальные переменные и функции, доступные в любом файле без необходимости импорта. Это аналог window в браузере.

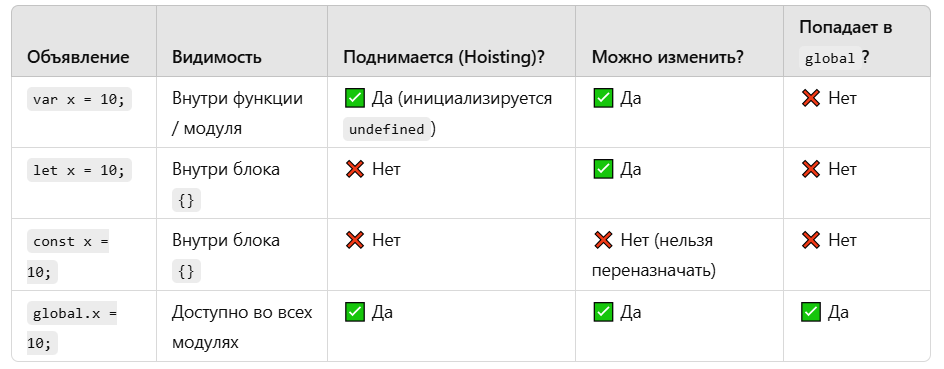
Любая переменная, объявленная с global, доступна везде. Но не рекомендуется использовать global, так как это может привести к конфликтам переменных между разными модулями.

Лучше использовать module.exports или import для передачи данных между модулями.

global.myVar = "Hello, World!";

console.log(myVar); // Hello, World!

Функции global: 🔹 setTimeout / setInterval / setImmediate – Таймеры, 🔹 console – Вывод информации в консоль



🔹 process – управление процессом в Node.js

* process.platform – платформа ОС (win32, linux, darwin).
* process.cwd() – текущая рабочая директория.
* process.env – переменные окружения.
* process.argv – аргументы командной строки.
* process.exit(code) – завершение процесса.
* process.pid – ID текущего процесса.

📌 Buffer – работа с бинарными данными

* Buffer.from(data, encoding) – создать буфер из строки/массива
* Buffer.alloc(size) – создать буфер указанного размера (обнуленный)
* Buffer.allocUnsafe(size) – создать буфер без очистки памяти
* buf.toString(encoding) – конвертировать буфер в строку (utf-8, hex, base64)
* buf.length – размер буфера в байтах
* buf.slice(start, end) – вырезать часть буфера
* Buffer.concat([buf1, buf2]) – объединить буферы
* buf.write(string, offset, length, encoding) – записать строку в буфер
* buf.compare(buf2) – сравнить два буфера
* buf.copy(targetBuffer, targetStart, sourceStart, sourceEnd) – копировать буфер

Объект process в Node.js позволяет работать с входными и выходными потоками данных, такими как стандартный ввод (stdin), стандартный вывод (stdout) и стандартный поток ошибок (stderr).

1. process.stdin (Стандартный ввод)

Позволяет читать данные, введенные пользователем (например, с клавиатуры).  
Работает в потоковом (stream) режиме.  
Можно слушать события data, end, error, readable.

2. process.stdout (Стандартный вывод)

Используется для вывода информации в консоль.  
Работает как поток (stream), можно писать через write().

Аналог console.log(), но без автоматического перевода строки.

3. process.stderr (Стандартный поток ошибок)

Используется для вывода ошибок.  
Работает как stdout, но логируется отдельно.



* server.close() гарантирует, что сервер больше не будет принимать соединения, но сама программа всё равно будет работать, ожидая дальнейших операций.
* process.exit(0) завершает всю программу, очищая ресурсы и завершая все операции.

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

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 понятие стандартные «системные потоки».**

стандартный ввод (stdin), стандартный вывод (stdout) и стандартный поток ошибок (stderr) выше

1. **Поясните назначение функций process.nextTick, setImmediate, поясните в чем разница.**

Микро и макро таски. Назначение – асинхронное выполнение кода.

**1. setTimeout(callback, delay):**

**Принцип работы:** setTimeout планирует выполнение callback функции *один раз* после истечения delay миллисекунд. Он возвращает *идентификатор таймера* (timer ID), который можно использовать для отмены таймера.

**Выполнение кода с задержкой:** Самый очевидный случай - выполнение какой-то операции через заданное время.

console.log("Начало");

setTimeout(() => {

console.log("Выполнилось через 2 секунды");

}, 2000);

console.log("Продолжение");

// Вывод:

// Начало

// Продолжение

// (через 2 секунды) Выполнилось через 2 секунды

**2. setInterval(callback, delay):**

**Принцип работы:** setInterval планирует *повторяющееся* выполнение callback функции каждые delay миллисекунд. Он также возвращает *идентификатор интервала* (interval ID), который можно использовать для остановки интервала.

let count = 0;

const intervalId = setInterval(() => {

count++;

console.log(`Выполнилось ${count} раз`);

if (count >= 3) {

clearInterval(intervalId); // Останавливаем интервал

console.log("Интервал остановлен");

}

}, 1000); // Выполнять каждую секунду

// Вывод:

// Выполнилось 1 раз

// Выполнилось 2 раз

// Выполнилось 3 раз

// Интервал остановлен

В Node.js методы ref() и unref() применяются к объектам Timeout и Interval, возвращаемым функциями setTimeout() и setInterval(), соответственно. Они управляют тем, как таймеры влияют на цикл событий (event loop) Node.js.

Цикл событий – это сердце Node.js. Он отвечает за обработку асинхронных операций. Node.js завершает свою работу, когда в цикле событий больше нет работы. Это значит, что если в очереди нет никаких задач (например, новых запросов, завершенных операций ввода/вывода, или активных таймеров), Node.js завершит процесс.

* **unref():**
  + Назначение: Метод unref() *отсоединяет* таймер от цикла событий. Это значит, что таймер больше не будет препятствовать завершению процесса Node.js. Даже если таймер еще не сработал, процесс может завершиться, если больше нет никаких других задач в цикле событий.
  + Возвращаемое значение: Возвращает таймер, чтобы можно было его зацепить обратно с помощью ref().
* **ref():**
  + Назначение: Метод ref() *присоединяет* таймер обратно к циклу событий. Это означает, что таймер *будет* препятствовать завершению процесса Node.js. Node.js продолжит работать, пока таймер не сработает или не будет отменен (например, с помощью clearTimeout() или clearInterval()).
  + Возвращаемое значение: Возвращает таймер.

1. **Поясните назначение 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.

Пакет - это дерево папок, описываемое файлом package.json. Пакет состоит из папки, содержащей файл package.json (описывает метаданные пакета, его зависимости и скрипты), и всех вложенных папок до следующей папки, содержащей другой файл package.json, или папки с именем node\_modules.

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.

Для начала работы с npm нужно инициализировать проект.

npm init

Для установки пакета используйте команду npm install. Например, установим пакет lodash.

npm install lodash

После установки пакета его можно импортировать и использовать в своем коде. Пример использования пакета lodash:

const lod = require('lodash');

Для обновления пакетов до последней версии используйте команду npm update.

npm update lodash

Чтобы удалить пакет, используйте команду npm uninstall

npm uninstall lodash

Чтобы опубликовать пакет, вам потребуется собрать все исходные коды и файл package.json в одной директории. В package.json должны быть указаны название, версия и зависимости пакета. Опубликовать пакет можно с помощью комадны npm publish.

1. **Перечислите команды npm, с помощью которых можно просмотреть список установленных пакетов.**

npm list (или npm ls): Выводит дерево зависимостей проекта (локально установленные пакеты).

npm list: показывает все пакеты, включая зависимости зависимостей.

npm list --depth=0: показывает только пакеты, установленные непосредственно в вашем проекте (без зависимостей зависимостей).

npm list -g: показывает глобально установленные пакеты.

npm list -g --depth=0: Показывает глобально установленные пакеты, без вывода зависимостей.

npm outdated: Показывает пакеты, для которых доступны более новые версии в реестре npm. Полезно для обновления.

1. **Поясните назначение параметра –g в командах npm.**

Параметр -g (или --global) указывает npm, что команда должна выполняться глобально, а не локально в контексте текущего проекта.

Когда команда npm должна выполняться глобально (с флагом -g или --global), это означает, что действие этой команды применяется не к конкретному проекту (не к текущей рабочей директории, где находится package.json), а к системе в целом.

1. **Перечислите команды npm,с помощью которых можно скачать пакет.**

npm install <package-name>: Скачивает и устанавливает пакет из реестра npm в локальную папку node\_modules текущего проекта. Информация о пакете добавляется в файл package.json (в раздел dependencies, если не указано иное).

npm install <package-name>@<version>: Устанавливает конкретную версию пакета (например, npm install react@16.8.0).

npm install <package-name> --save-dev (или -D): Устанавливает пакет как зависимость для разработки (devDependencies). Такие зависимости нужны только во время разработки (например, инструменты для тестирования, сборки) и не включаются в production-сборку. Информация о пакете добавляется в раздел devDependencies файла package.json.

npm install <package-name> --save-optional (или -O): Устанавливает пакет как необязательную зависимость. Если установка этого пакета не удастся, npm не прервет установку всего проекта. Информация о пакете добавляется в раздел optionalDependencies файла package.json.

npm install (без указания имени пакета): Устанавливает все зависимости, перечисленные в файле package.json текущего проекта (как dependencies, так и devDependencies). Это стандартная команда для установки всех необходимых пакетов после клонирования репозитория или при развертывании проекта.

npm install <github-username>/<repository-name>: Установка напрямую из GitHub репозитория.

npm install <local-folder>: Установка пакета из локальной папки.

1. **Поясните назначение файла package.json.**

package.json — это файл манифеста проекта на Node.js. Он содержит метаданные о проекте и играет центральную роль в управлении зависимостями и сборкой проекта.

1. **Описание проекта (Метаданные):**
   * Содержит основную информацию о проекте:
     + name: Название пакета/проекта.
     + version: Текущая версия.
     + description: Краткое описание.
     + author: Автор.
     + license: Лицензия (например, MIT, ISC).
     + repository: Ссылка на репозиторий кода (например, GitHub).
     + keywords: Ключевые слова для поиска пакета (если публикуется).
2. **Управление зависимостями:**
   * Это одна из **самых важных** функций.
   * dependencies: Список пакетов (библиотек, фреймворков), которые **необходимы для работы** вашего приложения в *production* (на конечном сервере или у пользователя). npm install устанавливает их.
   * devDependencies: Список пакетов, которые нужны только **во время разработки** (например, для тестирования, сборки, линтинга кода). Они не устанавливаются при npm install --production.
3. Перечислите последовательность действий, позволяющих опубликовать пакет с помощью **npm**.
4. Поясните аббревиатуру MIME, какой организаций поддерживается допустимые MIME, в каких компонентах запросов и ответов используется MIME.

**Аббревиатура MIME:** Расшифровывается как **Multipurpose Internet Mail Extensions** (Многоцелевые расширения интернет-почты)

* Изначально стандарт был разработан для расширения возможностей электронной почты (позволяя передавать не только текст ASCII, но и вложения, разные кодировки). Сейчас MIME-типы являются стандартом для всего Интернета и используются для определения типа контента. Реестр официальных MIME-типов ведется организацией **IANA** (Internet Assigned Numbers Authority - Администрация адресного пространства Интернет). Стандарты, описывающие MIME, публикуются **IETF** (Internet Engineering Task Force) в виде документов RFC (Request for Comments).
* **Использование в HTTP:** MIME-типы критически важны в протоколе HTTP и используются в основном в заголовках (headers):
  + **В HTTP-ответе (Response):** Заголовок Content-Type указывает браузеру или другому клиенту, как интерпретировать тело ответа (например, Content-Type: text/html говорит, что это HTML-документ, Content-Type: image/png - что это PNG-изображение).
  + **В HTTP-запросе (Request):**
    - Заголовок Content-Type используется, когда клиент отправляет данные на сервер (например, при отправке формы методом POST или при отправке JSON через API). Он указывает тип данных в теле запроса (например, Content-Type: application/json).
    - Заголовок Accept используется клиентом, чтобы сообщить серверу, какие MIME-типы он способен понять и принять в ответе (например, Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, \*/\*;q=0.8). Сервер может использовать эту информацию для отправки наиболее подходящего формата данных.

1. Перечислите теги HTML, интерпретация которых приводит к HTTP-запросам.

Браузер инициирует HTTP-запросы для загрузки ресурсов, указанных в атрибутах следующих (основных) HTML-тегов:

* <img> (атрибут src для загрузки изображения, srcset для адаптивных изображений)
* <script> (атрибут src для загрузки внешнего JavaScript файла)
* <link> (атрибут href для загрузки внешних ресурсов, чаще всего CSS (rel="stylesheet"), но также иконок (rel="icon"), шрифтов, манифестов (rel="manifest") и др.)
* <audio> (атрибут src или вложенные теги <source> с атрибутом src для загрузки аудиофайлов)
* <video> (атрибут src, poster или вложенные теги <source> с атрибутом src для загрузки видеофайлов и постера)
* <iframe> (атрибут src для загрузки содержимого вложенного фрейма)
* <object> (атрибут data для загрузки внешнего контента, например, PDF или Flash (устарело))
* <embed> (атрибут src для встраивания внешнего контента, часто для плагинов)
* <form> (атрибут action, запрос отправляется при *отправке* формы пользователем, а не при рендеринге тега)
* <input type="image"> (атрибут src для изображения на кнопке)

1. Перечислите способы выполнения HTTP-запросов из JS-сценария.

**fetch**

fetch('/api/data')

.then(response => response.json())

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

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

**XMLHttpRequest**

const xhr = new XMLHttpRequest();

xhr.open('GET', '/api/data');

xhr.onload = function() {

if (xhr.status === 200) {

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

} else {

console.error('Ошибка:', xhr.statusText);

}

};

xhr.onerror = function() {

console.error('Сетевая ошибка');

};

xhr.send();

**Библиотеки и фреймворки (jQuery)**

**WebSockets**

Хотя это не классические HTTP-запросы после установки соединения, начальное "рукопожатие" (handshake) для установки WebSocket-соединения происходит через HTTP.

1. Поясните понятие «параметризованный модуль».

**Параметризованный модуль** — это модуль (в контексте Node.js или других модульных систем), который экспортирует не статический объект или значение, а **функцию**. Эта функция принимает один или несколько параметров (конфигурацию, зависимости, настройки) и, будучи вызванной, **возвращает** реальный объект, функцию или класс, который будет использоваться в приложении.

**WasmFiddle:** Онлайн-инструмент, который позволяет писать код на C/C++ (и других языках), компилировать его в WASM и получать либо .wasm файл, либо его представление в виде массива байт (как в вашем коде).

* **Формат кода:** WebAssembly (WASM) — это низкоуровневый бинарный формат кода, который может исполняться веб-браузерами. Думайте о нем как об ассемблере, но для веба, и он не привязан к какой-то конкретной архитектуре процессора.
* **Цель компиляции:** Он не предназначен для написания кода напрямую (хотя это возможно). Вместо этого, языки высокого уровня, такие как C, C++, Rust и другие, компилируются в WASM.
* **Производительность и безопасность:** WASM спроектирован для выполнения кода с высокой производительностью (близкой к нативной скорости) внутри безопасной "песочницы" браузера. Это означает, что WASM-код не может напрямую получить доступ к вашей системе за пределами того, что разрешает браузер.
* **Дополнение к JavaScript:** WASM не заменяет JavaScript. Он работает вместе с JavaScript, позволяя переносить на веб ресурсоемкие задачи (например, игры, обработку видео, сложные вычисления), которые были бы слишком медленными на чистом JavaScript. JavaScript может загружать WASM-модули и вызывать их функции.

**10. Что такое emcc?**

* **Компилятор для WASM:** emcc (Emscripten Compiler Frontend) — это инструмент командной строки, который является основной частью **Emscripten SDK**. Его главная задача — компилировать исходный код, написанный на C или C++, в WebAssembly (.wasm файлы).
* **Цепочка инструментов:** emcc действует как "обертка" над более низкоуровневыми компиляторами, такими как Clang (часть LLVM). Он управляет всем процессом: от анализа C/C++ кода до генерации WASM и, при необходимости, JavaScript-кода ("клея"), который помогает WASM-модулю взаимодействовать с браузерным окружением (например, для вывода на экран, работы с файлами через API браузера и т.д.).
* **Результат:** На выходе emcc обычно производит .wasm файл (бинарный код) и, опционально, .js файл, который загружает и запускает этот WASM-модуль.
* Проще говоря, если у вас есть программа на C++, emcc — это основной инструмент, который вы используете, чтобы "пересобрать" эту программу так, чтобы она могла работать в веб-браузере в виде WASM.

Принцип работы **Long Polling** (длинного опроса) со стороны клиента (вашего бота) и сервера (Telegram API) следующий:

1. **Клиент (ваш бот) отправляет запрос серверу:**
   * Это обычный HTTP(S) GET-запрос (в вашем случае к /getUpdates).
   * В запросе клиент указывает timeout (например, 60 секунд). Это говорит серверу: "Если у тебя нет для меня новых данных прямо сейчас, пожалуйста, не отвечай сразу, а подожди до timeout секунд. Если данные появятся раньше, отправь их немедленно."
   * Также клиент передает offset, чтобы сервер знал, какие обновления клиент уже видел.
2. **Сервер (Telegram API) обрабатывает запрос:**
   * **Если есть новые данные (сообщения для бота) на момент получения запроса:** Сервер немедленно отправляет эти данные клиенту и закрывает соединение.
   * **Если новых данных нет:** Сервер **не закрывает соединение сразу**. Он удерживает его открытым.
     + **Пока соединение открыто:** Сервер продолжает проверять, не появились ли новые данные для этого клиента.
     + **Если новые данные появляются до истечения timeout:** Сервер отправляет эти данные клиенту и закрывает соединение.
     + **Если timeout истекает, а новых данных так и не появилось:** Сервер отправляет ответ (обычно пустой или со специальным флагом "нет обновлений") и закрывает соединение.
3. **Клиент (ваш бот) обрабатывает ответ:**
   * Получив ответ от сервера (с данными или без, или по таймауту), клиент обрабатывает его.
   * **Сразу после этого клиент немедленно отправляет новый Long Poll запрос серверу**, повторяя шаг 1.

Таким образом, создается цикл: запрос -> ожидание на сервере -> ответ -> немедленный новый запрос. Это создает иллюзию постоянного соединения, хотя технически это серия последовательных HTTP-запросов.