Why is body parser in Node js?

- ❖ body-parser extracts the entire body portion of an incoming request stream and exposes it on req.body.
- ❖ without middleware parsing your requests, your req.body will not be populated. You will then need to manually go research on the req variable and find out how to get the values you want.
- ❖ Your bodyParser acts as an interpreter, transforming http requests into an easily accessible format based on your needs.

1.) explain ES6 features?

There are many ES6 feature , i will mention few of them

spread operator::--->>The spread operator is represented by three dots (...) to obtain the list of parameters. It allows the expansion of an array, and object literals where more than zero arguments are expected.

The spread operator makes deep copies of data if the data is not nested. When you have nested data in an array or object the spread operator will create a deep copy of the top most data and a shallow copy of the nested data.

let array1 = [3, 4, 5, 6]
let clonedArray1 = [...array1];
// Spreads the array into 3,4,5,6
console.log(clonedArray1); // Outputs [3,4,5,6]

let obj1 = {x:'Hello', y:'Bye'};
let clonedObj1 = {...obj1}; // Spreads and clones obj1
console.log(obj1);

//--------------------------------------1---

```
function myFun(a,  b, ...manyMoreArgs) {
   console.log("a", a)
   console.log("b", b)
   console.log("manyMoreArgs", manyMoreArgs)
 }
  myFun("one", "two", "three", "four", "five", "six")
      //   a one
      // b two
      // manyMoreArgs [ 'three', 'four', 'five', 'six' ]
```

rest parameter ::---->>It improves the ability to handle the parameters of a function. Using the rest parameter syntax, we can create functions that can take an indefinite number of arguments.
****Note- Rest parameter should always be used at the last parameter of a function:**

The **spread operator** spreads out the elements of iterables such as arrays, objects, and strings into single elements. The **rest parameter** performs the opposite function and collects all remaining elements into a single array.
---------------------------------------1--
wWhat is the use of promises in javascript?
Promises are used to handle asynchronous operations in javascript.

By definition, a promise is an object that encapsulates the result of an asynchronous operation.

A promise object has a state that can be one of the following:

- **Pending**
- **Fulfilled with a value**
- **Rejected for a reason**

Before promises, callbacks were used to handle asynchronous operations. But due to limited functionality of callback(callback hell), using multiple callbacks to handle asynchronous code can lead to unmanageable code.
They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.
Promise object has four states -

Pending - Initial state of promise. This state represents that the promise has neither been fulfilled nor been rejected, it is in the pending state.
Fulfilled - This state represents that the promise has been fulfilled, meaning the async operation is completed.
Rejected - This state represents that the promise has been rejected for some reason, meaning the async operation has failed.
Settled - This state represents that the promise has been either rejected or fulfilled.

A promise is created using the Promise constructor which takes in a callback function with two parameters, resolve and reject respectively.

```
const myPromise = new Promise(function(resolve, reject) {});
```

//--------------------------------------2--
--
arrow functions::--->>

- Arrow functions don't have their own bindings to this, arguments or super, and should not be used as methods.
- They provide us with a new and shorter syntax for declaring functions.
- Arrow functions are declared **without the function keyword**. If there is only one returning expression then we don't need to use the return keyword as well in an arrow function as shown in the example above.
- Also, for functions having just one line of code, curly braces { } can be omitted.
-

```
var obj = {
  i: 10,
  b: () => console.log(this.i, this),
  c: function() {
    console.log(this.i, this);
  }
}

obj.b(); // prints undefined, Window {...} (or the global object)
obj.c(); // prints 10, Object {...}
```

//---------------------------------------3--

Default Parameters :-->>
we can put the default values right in the signature of the functions.

```
var calculateArea = function(height = 50, width = 80) {
console.log(height);
}
calculateArea();//50
```

In ES5, we were using logic OR operators.
var calculateArea = function(height, width) {

```
  height =  height || 50;
  width = width || 80;
  // write logic
   ...
}
```

//--4---

Template Literals::-->>

In ES6, we can use a new syntax ${PARAMETER} inside of the back-ticked string.

through which we can form multiline syntax in the easiest way.

//--5---

Destructuring assignment.

Which makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

var o = {p: 42, q: true};
var {p, q} = o;

console.log(p); // 42
console.log(q); // true

//--6--

var/let/const ::-->>

In javascript, you can create/declare variables using keywords var, let and const.

var declarations are function scoped or global scoped

It means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function..

let declarations are block scoped.so It can't be accessible outside the particular block ({block})

```
ReferenceError: b is not defined
```

Users cannot re-declare the variable defined with the *let* keyword but can update it.

```
Uncaught SyntaxError: Identifier 'a' has already been declared
```

const declarations are block scoped.

A **const ,**When users declare a **const variable**, they need to initialize it, otherwise, it returns an error. The user cannot update the *const* variable once it is declared.

```
const myName; //throws error as const needs to be initialized
const myName='John';
const myName='Doe'; //throws  error as const variable can not be
reassigned
var   myName='Doe'; //throws error as myName is reserved for constant
above,
```

same goes for let

console.log(x); // prints undefined
var x = 100;
console.log(x); //prints 100

console.log(y); //Reference error
ReferenceError: y is not defined
let y = 200;
console.log(y); //prints 200

console.log(z); //Reference error
const z = 300;
console.log(z); //prints 300

//--7--
Classes in ES6 ::-->>
We can create class in ES6 using "class" keyword.class definition can only include functions and constructors
//--8---
Modules in ES6

In ES6, there are modules with import and export operands.
export var userID = 10;
export function getName(name) {
  ...

};
We can import the userID variable and getName method using the import statement .
import {userID, getName} from 'module';
console.log(userID); // 10
//===
==

What is CORS?
CORS is shorthand for Cross-Origin Resource Sharing. It is a mechanism to allow or restrict requested resources on a web server depending on where the HTTP request was initiated.

This policy is used to secure a certain web server from access by another website or domain. For example, only the allowed domains will be able to access hosted files in a server such as a stylesheet, image, or a script.

Type of null is? :Object
null === undefined // false
null == undefined // true
null === null // true

What is the use of map reduce filters in js?
1 map — used to modify elements in an array and get a new array with modified elements.
Basically it takes 2 arguments, a callback and an optional context (will be considered as `this` in the callback) which I did not use in the previous example. The callback runs for **each value in the array** and **returns each new value** in the resulting array.`const officersIds = officers.map(officer => officer.id);`
2 reduce — to perform some operation in the array and return a single computed value.
https://medium.com/swlh/javascript-map-filter-reduce-illustrated-83897f144613
3 filter — to get a subset of an array that satisfies certain conditions.
`const rebels = pilots.filter(pilot => pilot.faction === "Rebels");`

What is callback and callback hell?

```
var myCallback = function(err,data) {
   console.log('got data: '+data);
 };
  var getUsers = function(callback) {
   callback(null,[1,2,3]);
};

getUsers(myCallback);
```

When a function simply accepts another function as an argument, this contained function is known as a **callback function.**

- ➔ When you write a code where there is a situation like writing many nested callbacks, which makes code hard to read and maintain and hard to debug.the code structure looks like a pyramid, making it difficult to read and maintain..Also, if there is an error in one function, then all other functions get affected and to notice errors also a bit complicated.

What is an EventEmitter in Node.js?
EventEmitter is a class that holds all the objects that can emit events
Whenever an object from the EventEmitter class throws an event, all attached functions are called upon synchronously.

What is the promise chain in js?
suppose we have a situation like to execute two or more asynchronous operations back to back, where each subsequent operation starts when the previous operation succeeds, with the result from the previous step. We accomplish this by creating a promise chain.
Example Link : **https://www.geeksforgeeks.org/how-to-avoid-callback-hell-in-node-js/**

What is hoisting ?
Irrespective of where the variables and functions are declared,those declarations (i.e. var x;) are valid for the entire scope they are written in, even if you declare after you assign. they are moved on top of the scope.
'NOTE' :::hoisting is avoided by using the let keyword instead of var.
Strict mode does not change any of this. It would throw an error if you omitted the var x; declaration altogether; without strict mode, the variable scope would implicitly be the global scope.

Explain node js internal architecture?

Assigning Independent Threads to Each Request is an Expensive Task. In technologies like C# and Java, each request is processed with an independent Thread. The problem with the approach is that the Server has Fixed No of Threads in the Thread Pool. Therefore at any

moment of time, Maximum Request processes are equal to the number of Threads in the Thread Pool.
While processing a Request, it may involve expensive I/O Operations, In that case, the Thread remains busy waiting for the time consuming process to be complete. This wastes a lot of Thread Processing Time waiting for I/O Operations. Thread might be Idle for maximum time, hence resulting in wastage of Resources and Time.

----

whenever Clients Send requests to Web Server.

Node JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.

Node JS Web Server receives those requests and places them into a Queue. It is known as "Event Queue".

Whenever an object from the EventEmitter class throws an event, all attached functions are called upon synchronously.

Explain node js internal architecture?

Assigning Independent Threads to Each Request is an Expensive Task. In technologies like C# and Java, each request is processed with an independent Thread. The problem with the approach is that the Server has Fixed No of Threads in the Thread Pool. Therefore at any moment of time, Maximum Request processes are equal to the number of Threads in the Thread Pool.

While processing a Request, it may involve expensive I/O Operations, In that case, the Thread remains busy waiting for the time consuming process to be complete. This wastes a lot of Thread Processing Time waiting for I/O Operations. Thread might be Idle for maximum time, hence resulting in wastage of Resources and Time.

----

whenever Clients Send requests to Web Server.

Node JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.

Node JS Web Server receives those requests and places them into a Queue. It is known as “Event Queue”.

Node JS Web Server internally has a Component, known as “Event Loop”. the event loop runs indefinitely to retrieve request in the event queue and processing

Event Loop uses Single Thread only. It is the main heart of Node JS Platform Processing Model.

Even Loop checks if any Client Request is placed in the Event Queue. If not, then wait for incoming requests for indefinitely.

If yes, then pick up one Client Request from Event Queue

Starts process that Client Request

If that Client Request Does Not involve in any Blocking IO Operations, then process everything, prepare response and send it back to client.

If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services,the Main Thread allocates a Background thread to process the I/O Operation.

Checks Threads availability from Internal Thread Pool

Picks up one Thread and assigns this Client Request to that thread.

The Background Thread uses Event Based Approach to Notify Main Thread. Each Async Task consists of some Callback Function associated with it, once the Async Task is Complete, Background Thread raises Event to Notify the Main Thread about the completion of the Async Task

Event Loop in turn, sends that Response to the respective Client.

What is the use of NPM?

NPM basically is the package manager for nodes. It helps with installing various packages and resolving their various dependencies so that node can find them, and manages dependency conflicts intelligently which are needed for your web development. It greatly helps with your Node development. Most commonly, it is used to publish, discover, install, and develop node programs.

This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.so

When someone installs our project through npm, all the dependencies listed in the package.json will be installed as well. Additionally, if someone runs npm install in the root directory of our project, it will install all the dependencies to ./node_modules directory.

2.what is package.lock.json?

used to lock dependencies to a specific version number.

The goal of package-lock. json file is to keep track of the exact version of every package that is installed so that a product is 100% reproducible in the same way even if packages are updated by their maintainers.

o when someone cloned your repo and ran npm install in their machine. NPM will look into package-lock.json and install exact versions of the package as the owner has installed.so the application will run smoothly without breaking any changes.

What is Package lock json vs package json?

this can be a huge issue if package developers break any of the functions on the minor version as it can make your application break down.So npm later released a new file called package-lock.json to avoid such scenarios

package-lock. json will simply avoid this general behavior of installing updated minor versions so when someone cloned your repo and ran npm install in their machine. NPM will look into package-lock. json and install exact versions of the package as the owner has installed so it will ignore the ^ and ~ from package

3. Can we change package-lock.json manually?

When you npm install some-package , the lock file is updated automatically. When you update the version of a package in your package

npm install can alter package-lock.json.For example, if someone manually alters package.json — say, for example, they remove a package since it's just a matter of removing a single line — the next time that someone runs npm install, it will alter package-lock.json to reflect the removal of the previous package.

What is the purpose or use of package-lock.json?

To avoid differences in installed dependencies on different environments and to generate the same results on every environment we should use the package-lock.json file to install dependencies.

Ideally, this file should be on your source control with the package.json file so when you or any other user will clone the project and run the command "npm i", it will install the exact same version saved in package-lock.json file and you will able to generate the same results as you developed with that particular package.

Why should we commit package-lock.json with our project source code?

During deployment, when you again run "npm i" with the same package.json file without the package-lock.json, the installed package might have a higher version now from what you had intended.

Now, what if you wanted to have that particular version for your dependency during deployment which you used at the time of development. This is the need of creating a package-lock.json file and keeping it with the source code. This file is created with the details of the specific version installed in your project.

What happens if you delete package-lock json?

So when you delete package-lock. json, all that consistency goes out the window. Every node_module you depend on will be updated to the latest version it is theoretically compatible with.

How to update package-lock.json without doing npm install?

npm

As of npm 6.x, you can use the following command:

```
npm i --package-lock-only
```

Sort array of objects by single key with date value?

```
var  a = [{name:"alex"},{name:"clex"},{name:"blex"}];

a.sort((a,b)=> (a.name > b.name ? 1 : -1))

a.sort((a,b)=> (a.name - b.name ))
```

What are array functions used in your application?

The **push,map,concat** and pop functions let you add and remove items to the end of an array, respectively:

The `indexOf()` method returns the first index at which a given element can be found in the array, or -1 if it is not present.

sort method is used to arrange the elements of the array in ascending order

```
var nums = [ 1, 1, 2, 3, 5, 8 ];

arr.reverse()

console.log(arr)

var position = arr.indexOf("avatar");
```

What is the purpose of module.exports?

This is used to expose functions of a particular module or file to be used elsewhere in the project. This can be used to encapsulate all similar functions in a file which further improves the project structure.

A module in Node.js is used to encapsulate all the related codes into a single unit of code, so if we do module.exports , it's like we are exporting the entire code from a given file so other files are allowed to access the exported code.

Explain the concept of middleware in Node.js.

Middleware is a function that receives the **request** and **response** objects. Most tasks that the middleware functions perform are:

There we can execute any code and Update or modify the request and the response objects and Finish the request-response cycle Invoke the next middleware in the stack.
basically **next()** is called when you want the routing for this request to continue onto the next middleware or the next route handler for this request.

What is the parameter "next" used for in Express?

next() is called when you want to pass the control to the **next middleware** in the chain or any **matching** route.
Suppose if I did not call next() , the request will be left hanging.The response will not be sent to the browser so the browser will keep waiting.

What is the purpose of NODE_ENV?

NODE_ENV is an environment variable popularized by the Express framework. It specifies the environment in which an application is running such as **development**, **staging**, **production**, **testing**, etc.

we can change the environment by changing process.env.NODE_ENV
process.env is a reference to your environment, so you have to set the variable there and we can update it.

export NODE_ENV=production

Why is node js single threaded?

Node.js is single-threaded for async processing. By doing async processing on a single-thread under typical web loads, more performance and scalability can be achieved instead of the typical thread-based implementation.so which makes that lightweight and efficient
Node JS Web Server internally has a Component, known as “Event Loop”. The event loop runs indefinitely to retrieve requests in the event queue and processing.

Event Loop uses Single Thread only. It is the main heart of Node JS Platform Processing Model.

If yes, then pick up one Client Request from Event Queue

Starts process that Client Request

If that Client Request Does Not require any Blocking IO Operations, then process everything, prepare response and send it back to client.

If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services,the Main Thread allocates a Background thread to process the I/O Operation.

Checks Threads availability from Internal Thread Pool

Picks up one Thread and assigns this Client Request to that thread.

The Background Thread uses Event Based Approach to Notify Main Thread. Each Async Task consists of some Callback Function associated with it, once the Async Task is Complete, Background Thread raises Event to Notify the Main Thread about the completion of the Async Task

Event Loop in turn, sends that Response to the respective Client.

Why should we commit package-lock.json with our project source code?

During deployment, when you again run “npm i” with the same package.json file without the package-lock.json, the installed package might have a higher version now from what you had intended.
Now, what if you wanted to have that particular version for your dependency during deployment which you used at the time of development. This is the need of creating a package-lock.json file and keeping it with the source code. This file is created with the details of the specific version installed in your project.

How do I get the domain originating the request in express.js?

var origin = req.get('origin');

If you're looking for the client's IP, you can retrieve that with:

var userIP = req.socket.remoteAddress;
If you are running your app behind Nginx or any proxy, every single IP address will be 127.0.0.1.
So, the best solution to get the ip address of user is :-
let ip = req.header('x-forwarded-for') || req.connection.remoteAddress;

What is the process in Nodejs?

- ➢ The term **process** is an **operating system term** and not a node.js term.
- ➢ Actually **process** in node allows users to obtain node related info and have some functions to control the node's behavior.
- ➢ The process module in node.js is a central place where the designers of node.js put a bunch of methods that relate to the overall process such as **process.exit()** which exits the application and thus stops the process or **process.env** which gives you access to the environment variables for your program or process.argv which gives you access to the command line arguments your process was started.
- Actually The **process binding** is available globally in Node. It provides various ways to inspect and manipulate the current node program.

What are the global objects in Nodejs?

Ans: Node.js global objects are available for all modules.
You don't need to include these objects in your application; rather they can be used directly.
These objects are **module, require(),exports,process,console, buffer**
global ::-There is no window in Node.js but there is another highest object called global.

- Whatever you assign to global.something in one module is accessible from another module.

Now when you declare a variable at one module var i=0; is it available from all modules? NO !
When a script is run, it is wrapped in a module. The top level variables in a script are inside a module function and are not global. This is how node.js loads and runs scripts whether specified on the initial command line or loaded with require().

so if you want to work with global, you have to use with the global prefix

https://stackoverflow.com/questions/34967530/about-global-variables-in-node-js

Difference between async/await and Promise in Nodejs?

Promise is an object representing an intermediate state of operation which is guaranteed to complete its execution at some point in future.

Async/Await is a syntactic sugar for promises, async/await simply gives you a synchronous feel to asynchronous code.

For simple queries and data manipulation, Promises can be simple, but if you run into scenarios where there's complex data manipulation,especially when the amount of promises which we are using increases. async/await might be more readable than promise chaining.

Explain Loggers in Nodejs?

If you want to store your error log in a remote location or separate database, Winston might be the best choice because it supports multiple transports.

Difference between req.query[] and req.params?

so **request.params** is an object containing properties to the named route and **request.query** comes from query parameters in the URL.
https://stackoverflow.com/questions/14417592/node-js-difference-between-req-query-and-req-params

How do you serve static files in Nodejs?

- To serve static files such as images, CSS files, and JavaScript files, use the **express.static built-in middleware function in Express**.
- express.static exposes a directory or a file to a particular URL so it's contents can be **publicly** accessed.

app.use(express.static(__dirname + '/public'));

What is a cluster in Nodejs?

- Node.js runs single threaded programming, which is very memory efficient, but to take advantage of computers multi-core systems, the Cluster module allows you to easily create child processes that each run on their own single thread by sharing the same server port to handle the load.

- **Clustering offers a way of improving your Node.js app performance by making use of system resources in a more efficient way.**
- A single instance of Node.js runs in a single thread. To take advantage of multi-core systems, the user will sometimes want to launch a cluster of Node.js processes to handle the load.

```
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
 console.log(`Master ${process.pid} is running`);
 // Fork workers.
 for (let i = 0; i < numCPUs; i++) {
   cluster.fork();
 }
 cluster.on('exit', (worker, code, signal) => {
   console.log(`worker ${worker.process.pid} died`);
 });
} else {
  // your server code default
  server.listen(config.port, config.ip, function () {
   console.log('Express server listening on %d, in %s mode',
config.port, app.get('env'));
 });
 console.log(`Worker ${process.pid} started`);
}
```

What are streams in Node.js?

- Streams are objects that enable you to read data or write data continuously.
- There are four types of streams:

- Readable – Used for reading operations
- Writable - Used for write operations
- Duplex - Can be used for both reading and write operations
- Transform - streams that can modify or transform the data as it is written and read

➢ Streams basically provide two major advantages compared to other data handling methods:
   - **Memory efficiency:** you don't need to load large amounts of data in memory before you are able to process it
   - **Time efficiency:** it takes significantly less time to start processing data as soon as you have it, rather than having to wait with processing until the entire payload has been transmitted
   - https://stackoverflow.com/questions/46424772/significance-of-flags-option-in-createreadstreams-createwritestreams-of-node-j

What are the security measures you take for node js routes?

Express-mongo-sanitize

Sanitize your express payload to prevent MongoDB operator injection

```
app.use(mongoSanitize());
```

Helmet helps you secure your Express apps by setting various HTTP headers like Content-Security-Policy

app.use(helmet())

How can I do Base64 encoding in Node.js?

- Buffers can be used for taking a string or piece of data and doing Base64 encoding of the result. For example:

```
console.log(Buffer.from("Hello World").toString('base64'));
//SGVsbG8gV29ybGQ
console.log(Buffer.from("SGVsbG8gV29ybGQ=", 'base64').toString('ascii'))//
Hello World
```

- Buffers are a **global object**, so no require is needed.
- Buffers created with strings can take an optional encoding parameter to specify what encoding the string is in.

Can we do async/await functions parallely?

We can use Promise.all

```
let [someResult,anotherResult] = await Promise.all([someCall(),
anotherCall()]);
```

```
const express =
require('express');
//Import the express
dependency
```

```
const app = express();              //Instantiate an
express app, the main work horse of this server

const port = 5000;                  //Save the port number
where your server will be listening



//Idiomatic expression in express to route and respond to
a client request

app.get('/', (req, res) => {        //get requests to the
root ("/") will route here

   res.sendFile('index.html', {root: __dirname});
//server responds by sending the index.html file to the
client's browser


//the .sendFile method needs the absolute path to the
file, see:
https://expressjs.com/en/4x/api.html#res.sendFile

});



app.listen(port, () => {            //server starts
listening for any attempts from a client to connect at
port: {port}

   console.log(`Now listening on port ${port}`);

});
```