

what is [node.js](#) ?

→ so basically [node.js](#) is the run time environment for the javascript.

→ so it internally uses v8 engine to execute the code properly there..

it is too simple and in the [node.js](#) when you need some type of import then what u will actually do ??

→ for that we are using the `require(<path>)`

→ it will return the module for access

→ so overall syntax is `const <var>=require(<path>);`

→ by this way we can require any of the things which we want actually here..

Now why we can't access that particular like the modules here

→ so for that we have to first export those modules here

→ so for exporting we have to first do

```
->module.exports={  
  key:value  
}
```

→ by this way we can export multiple things actually from the one folder to the another folder here actually ...

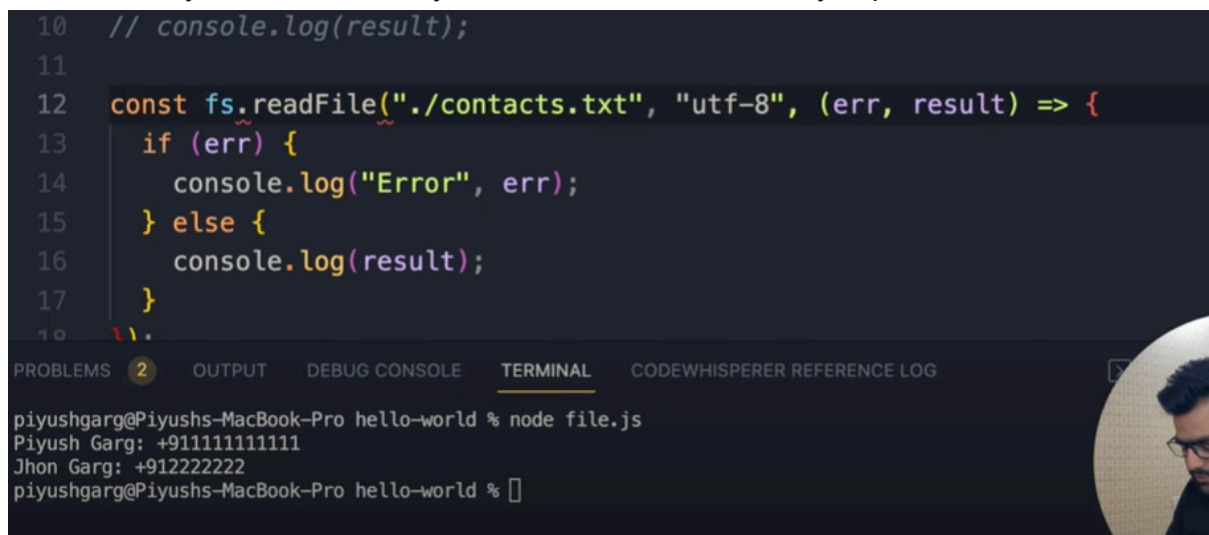
Now what is the alternative for exporting the modules here...

→ i have mentioned the methods for actually exporting the modules here..

→ `export.<property-name>=(a,b)=>a+b;` like that also we can pass the things here..

→ so by this way we can actually export the modules here ..

Now what is synchronous and asynchronous tasks ?????? <very-important>



The screenshot shows a code editor with a dark theme. The code is as follows:

```
10 // console.log(result);  
11  
12 const fs.readFile("./contacts.txt", "utf-8", (err, result) => {  
13   if (err) {  
14     console.log("Error", err);  
15   } else {  
16     console.log(result);  
17   }  
18 })
```

Below the code, the terminal output is visible:

```
piyushgarg@Piyushs-MacBook-Pro hello-world % node file.js  
Piyush Garg: +9111111111111  
Jhon Garg: +9122222222  
piyushgarg@Piyushs-MacBook-Pro hello-world %
```

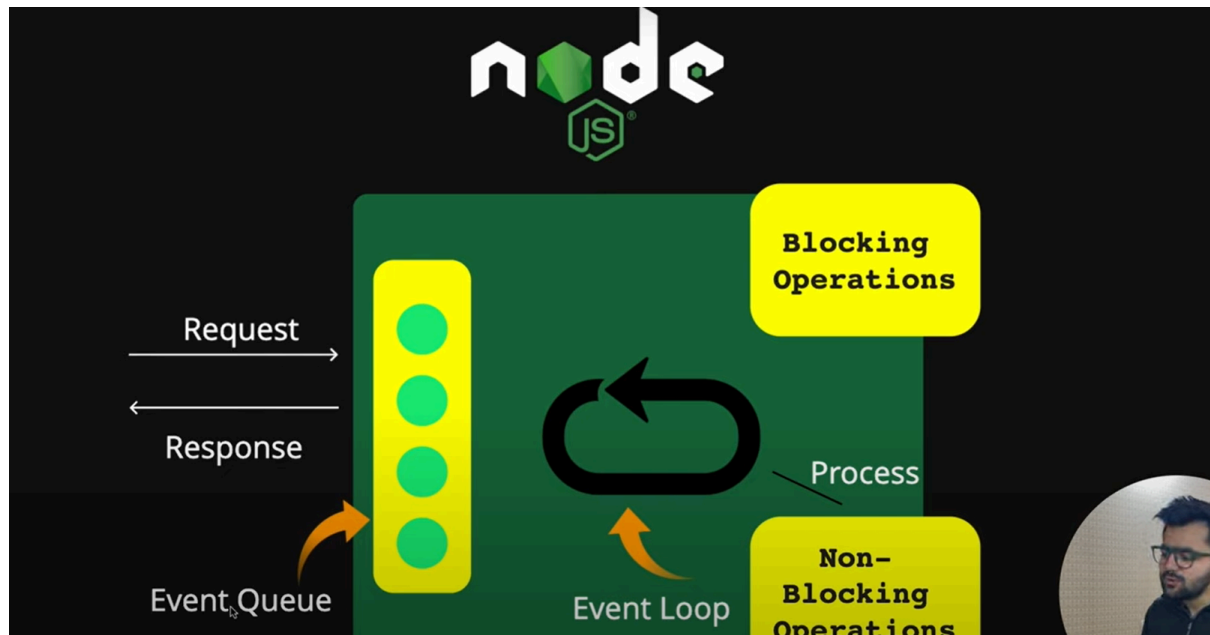
The terminal output demonstrates that the asynchronous `fs.readFile` call does not return the result immediately, as the subsequent `console.log` statements in the terminal output occur before the file content is printed.

→ so you can see asynchronous is not actually returning the result..

→ It is actually returning the call backs here..

→ and it is having err and result in its callbacks means what ever the code outputs it is having so by that way we are doing the things here actually...

→ now you think what is an event-loop so basically event-loop is here



→ the task can be blocking task

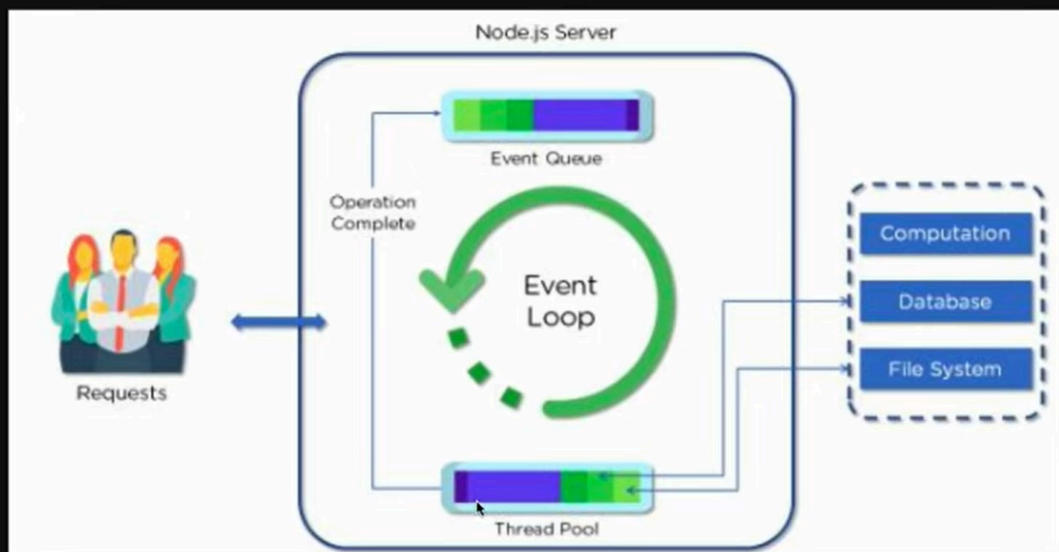
→ or the task can be non-blocking task

→ so to handle the different task we have event loops and worker pool to execute our tasks here..

→ you may think how the worker thread pool actually works ..

→ so the worker thread pool is basically doing its task by the assigning to the different thread here ...

Now the big question is how the event loop handling the multiple



→ here is the detail structure for how the actually event lops works here and how we do any kind of asynchronous tasks here ..

→ so in the interview this is the best topic to do so...

Now where we can write all the configurations related stuffs in [node.js](https://nodejs.org/en/docs/package-manager/) \

→ so , for that you have to just type the npm init

→ and it will initialise our package.json there

→ and then we can work on that

→ we can add scripts and dependencies and all in that very easily..

Now How Can We Create Http Server and all

→ for that we have a http server like for that we can require(<http>) module directly there ...

→ so by that we can directly create http server and then process the request which is actually received there

→ for processing the request we have the callback function like (req,res)={} there...

→ that will directly take the request and then gives us the response correctly via the synchronous or asynchronous way there..

→ now you think how can we install our dependencies so for that we are using the **npm** here..

→ we have to first define our task into the package.json and then whichever the dependencies are there that will automatically downloaded their after just typing the npm i <which-ever-dependencies>

- for downloading all our dependencies we are having `< npm i >` only there.
- now they are internally parsing the url and then handles all the parameters actually there...
- means there are key-value pair and we are actually managing there things there...

Now Let's Move Forward To [Express.JS](#)

- now what is the actual use of the [express.js](#) why we are using that
- so it is basically that when we are having an simple [node.js](#) that has many things to write means we have to create our own handler function for handling the requests and all....
- now we think how can we handle this much and more requests here
- we can use the same logic just import express and doing our tasks easily there without any too much load there...

```
const fs = require("fs");
const url = require("url");
const express = require("express");

const app = express();

app.get("/", (req, res) => {
  return res.send("Hello From Home Page");
});

app.get('/about', (req, res) => {
  return res.send("Hello From Home Page");
})
```

- now here we can see how we are writing our express code and handles the requests here with the url and all...
- so in the [express.js](#) everything is built in there
- we can directly make call and then fetch the parameters and then make an response there easily....
- so in the express we have to just make an express code and then just access by the path and just call the handler and just make an call there and get the proper details here...