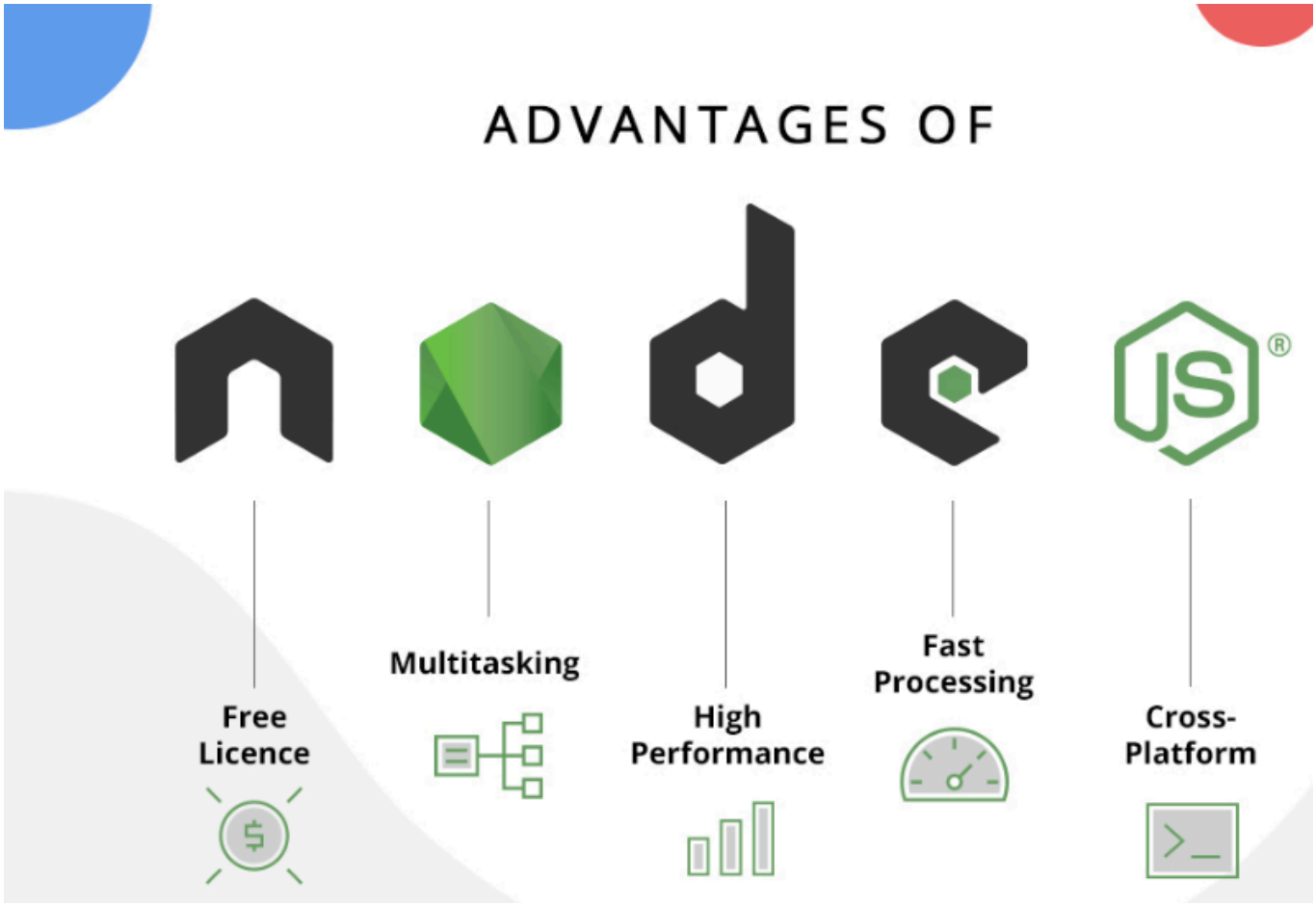
NODE JS



Contents-

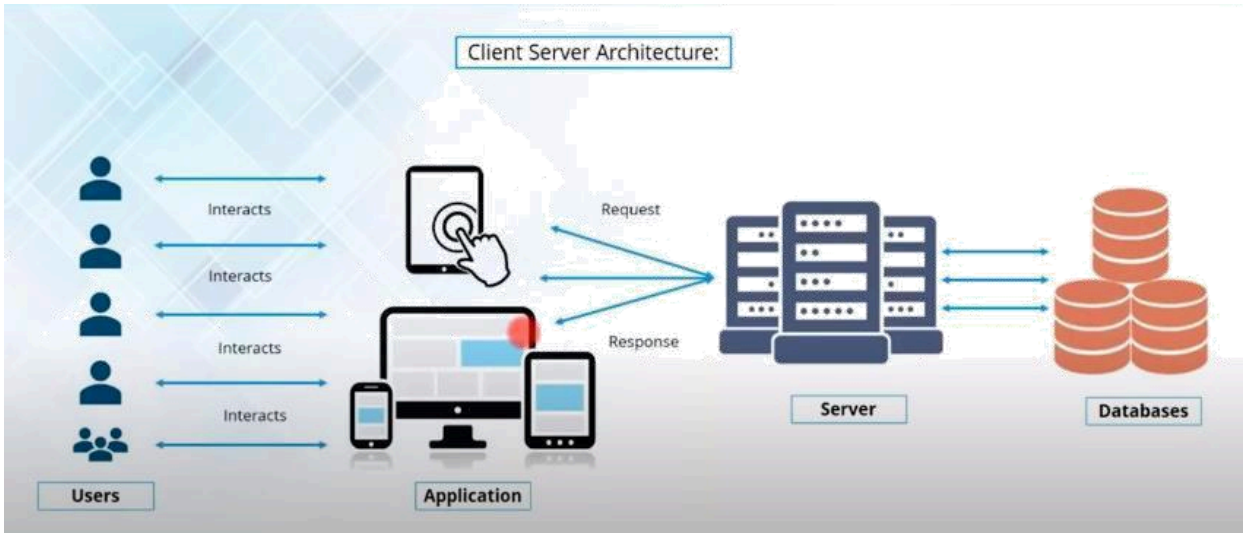
What is node js?

Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser.

It uses event driven, Non blocking I/O model which work async and on single thread architecture.

Why Node Js

Ans - Node Js Follows Client Server Architecture.



Client/Users ----------→ Websites/Apps ----------→ Servers → Databases (REQUEST)
Client/Users <---------- Websites/Apps <---------- Servers < Databases (RESPONSE)

OLA/Uber Example -
Uber customers/passengers make requests to book cabs using the Uber app. These requests are sent to the uber server and then further sent to the uber database to check the nearest available cab.

These cab info are sent back to the customer in the form of a response.

What is a Thread

Ans : -

Threads are like processes: they have their own pre-defined set of instructions and can execute one task at a time. Unlike processes, threads do not have their own memory. Instead, they reside within a process's memory.

What is single Thread

and Multi Thread Model

Ans : - **Single Thread**

All requests are handled by a single thread.

In a single-threaded environment like Node.js, there is only one main execution thread. However, the concurrency is achieved through an event-driven, non-blocking I/O model. When you have multiple incoming requests, Node.js can efficiently handle them asynchronously without creating a separate thread for each request. Instead, it uses a single thread to manage the event loop and delegates I/O operations to a background thread pool.

So, for example in the case of 10 incoming requests, there would still be only one main thread managing the event loop.

This design allows Node.js to efficiently handle a large number of concurrent connections with a single thread.

Advantages -

Simplicity of Programming Model: The single-threaded, event-driven model simplifies the programming model for developers. Handling asynchronous operations using callbacks allows for more straightforward code organization and can lead to cleaner and more maintainable code.

Efficient Resource Utilization: With a single thread, Node.js can efficiently manage concurrent connections and handle a large number of simultaneous requests without the overhead of creating and managing multiple threads.

Responsive and Fast: The event-driven, non-blocking I/O model makes Node.js highly responsive. It can efficiently handle a large number of concurrent connections, leading to fast response times for clients making requests to the server.

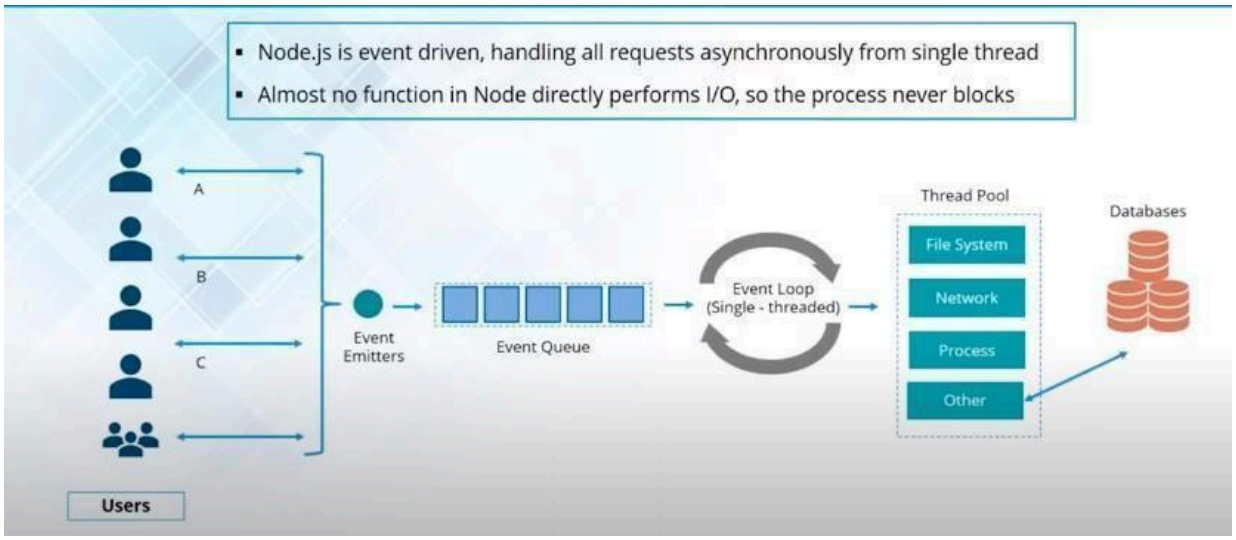
Disadvantage

1. **Blocking Operations:** If a particular operation takes a long time to complete and is synchronous (blocking), it can potentially block the entire event loop, causing delays in processing other requests. Developers need to be mindful of writing non-blocking code to avoid such issues.

2. **CPU-Intensive Tasks:** Node.js may not be the best choice for CPU-intensive tasks that require significant computation. Since Node.js uses a single thread, long-running CPU-bound tasks can block the event loop and negatively impact the responsiveness of the application.

Single Thread Model

Node js is event driven(events like – click, double click, mouse event etc). Handle all requests asynchronously using a **single thread.**
Almost no functions directly perform Input/Output operations, So process never blocks.



In a single thread, when a user performs an activity(click, submit etc), an event is generated. Every New request is threatened as an event. Event emitter allocates those events in the event queue which is a single thread. These events are executed/processed using an event-loop mechanism which is called a single thread mechanism. Single thread takes an event in the queue and sends in the thread pool.

There are different operation can be handled in Thread Pool Like File Operations

I/O
Operations
Network
Operation
CPU Intensive Operations
The Thread in thread pool, also called worker thread takes those operations asynchronously.
The processing of events in the event queue by event loop is also called event driven Modal or single thread modal.

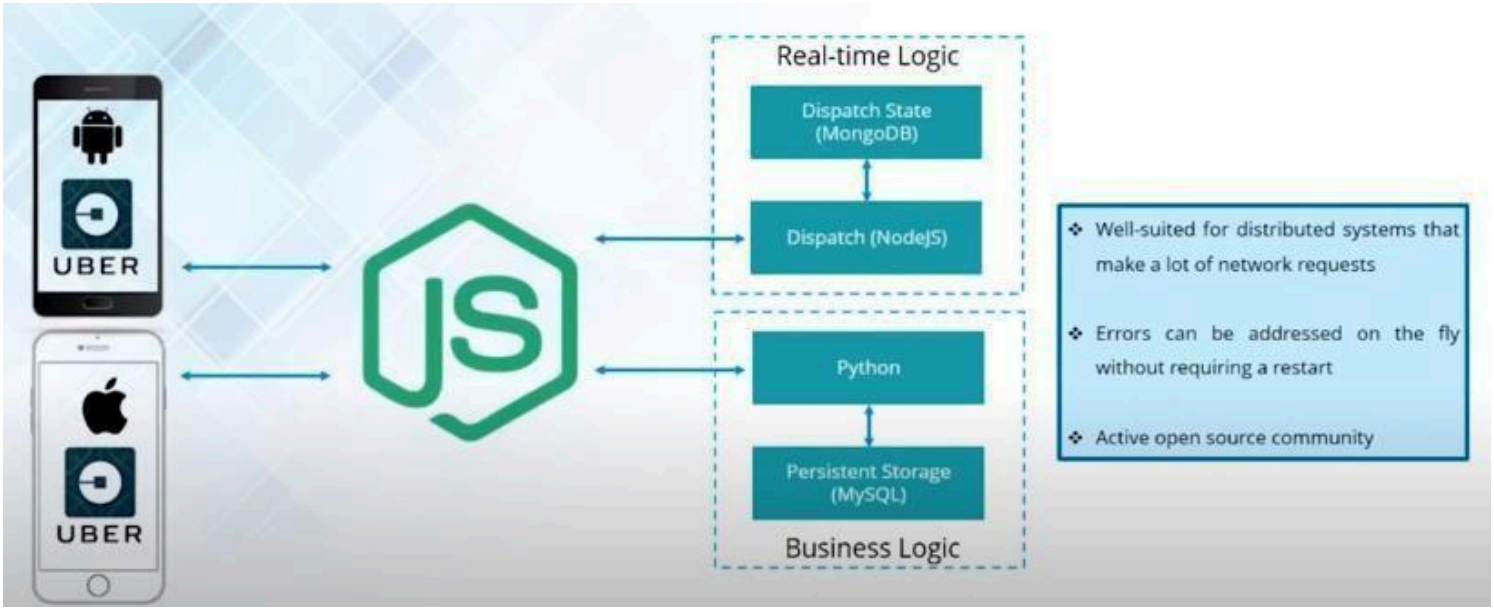
Multi Thread Vs Event Driven

Multi-Threaded	Asynchronous Event-driven
Lock application / request with listener-workers threads	Only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
Multithreaded server might block the request which might involve multiple events	Manually saves state and then goes on to process the next event
Using context switching	No contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

incoming request model

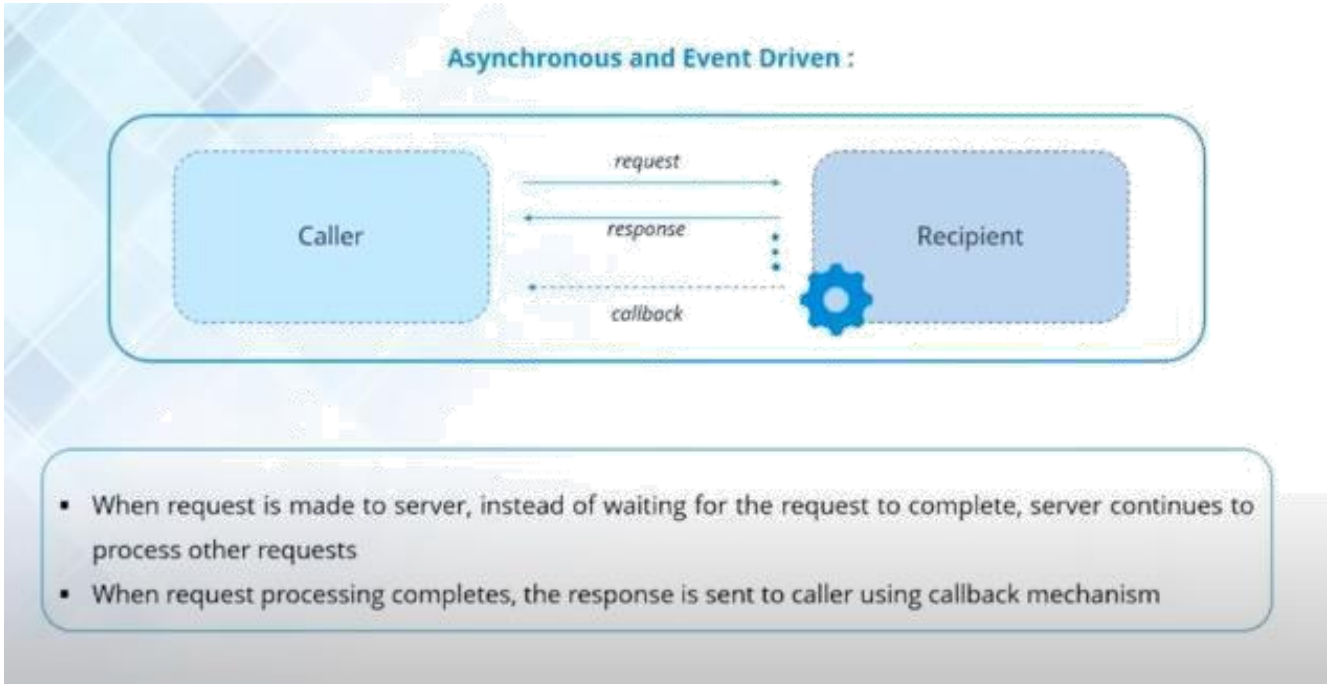
- a new thread is allocated for a new request every time. Multi thread works in synchrony.

UBER EXAMPLE -



Here there are two DBs. MongoDB for car details and MySql for driver details. Since Node is async so two requests can be sent simultaneously and get the response.

Error handling is easy and quick in node js.



NOTE : - All API in node js are single thread. So one API Never waits for another API to respond.
There is a notification mechanism used in node js which is a callback

function.

NODE JS FEATURES :-

Node Js Installation -

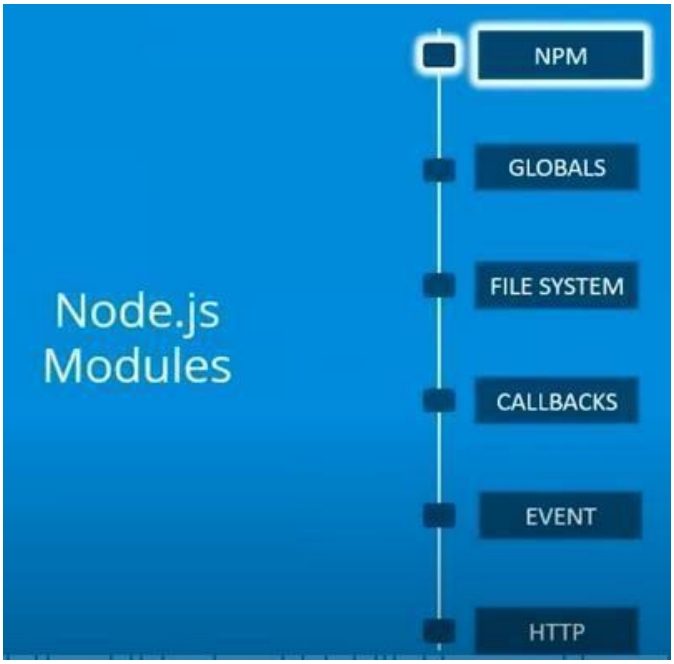
https://nodejs.org/en/download/

Simple node js example using sync(blocking)

```
let fs = require('fs');

let data = fs.readFileSync(‘blog.txt’);
console.log(data.toString());
console.log("End  Here");
//=========== blog.txt hello nodejs.
//=========== Output : hello nodejs end here
//======= Async (Non Blocking) :-

let fs = require('fs');

fs.readFile('blog.txt', (err, data) => {
  if (err) {
    console.log(err);
```

```
  } else {
    setTimeout(function () {
      //  this  request  will  go  to  event  queue.
      console.log('show  after  2  sec.');
       console.log(data.toString());
    }, 2000);
  }
});
console.log("start  here");
//======== output : - start here
//======== show after 2 sec.
```

NODE JS MODULES :-



NPM:-

npm stands for "Node Package Manager." It is a package manager for the JavaScript programming language. npm is primarily used to manage and install packages or libraries of reusable code that developers can use in their Node.js projects.

With npm, developers can easily install, update, and remove packages

from their projects. It provides a command-line interface (CLI) that allows developers to interact with the npm registry, which is a collection of publicly available JavaScript packages.

package.json

The package.json file is a metadata(**Metadata is data that describes other data**) file used in Node.js projects to manage various aspects of the project, including its dependencies, scripts, and other configuration details. It is located in the root directory of the project and follows the JSON (JavaScript Object Notation) format.

Here are some key elements typically found in a package.json file:

name: The name of the project or package.

version: The version number of the project.

description: A brief description of the project.

main: The entry point of the application, typically a JavaScript file.

dependencies: A list of dependencies required by the project. Each dependency is specified with its name and version range.

devDependencies: Similar to dependencies, but these are development dependencies that are not required in the production environment.
scripts: Allows the definition of custom scripts to automate tasks, such as running tests, building the project, or starting the application.

author: The author of the project.

license: The license under which the project is distributed.
The package.json file is essential for dependency management. When you run npm install in the project directory, npm reads the package.json file and installs all the listed dependencies into a node_modules folder. The package-lock.json file is also generated to lock down the versions of the installed packages for consistent behavior across different installations.

initiating package.json file

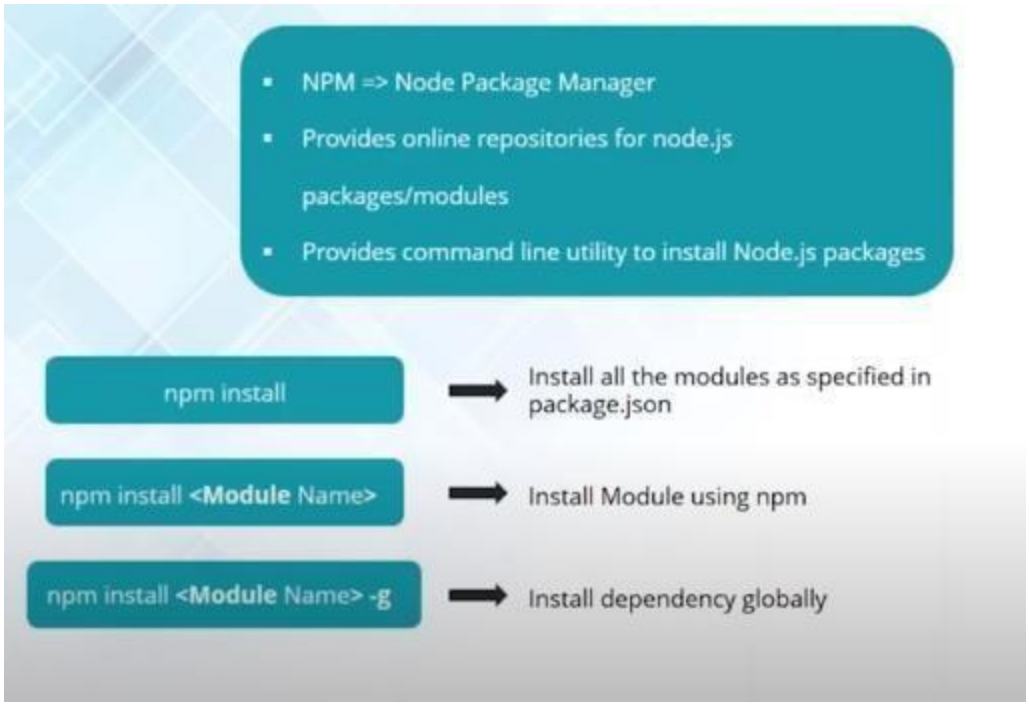
```
> npm init
```

installing dependencies

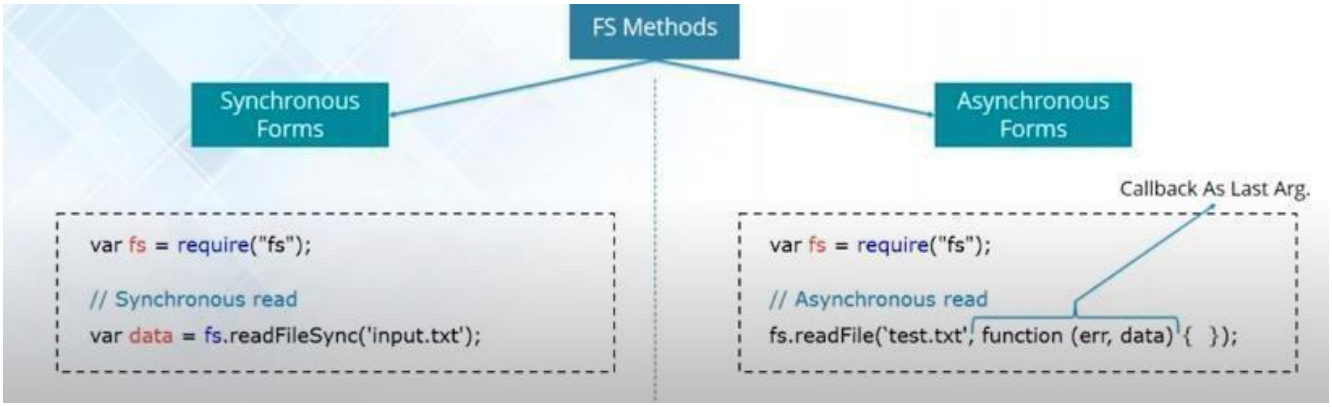
```
> npm i package_name
```

installing Dev dependencies

```
> npm i --save-dev package_name
```



NODE JS FILE SYSTEM MODULES:-

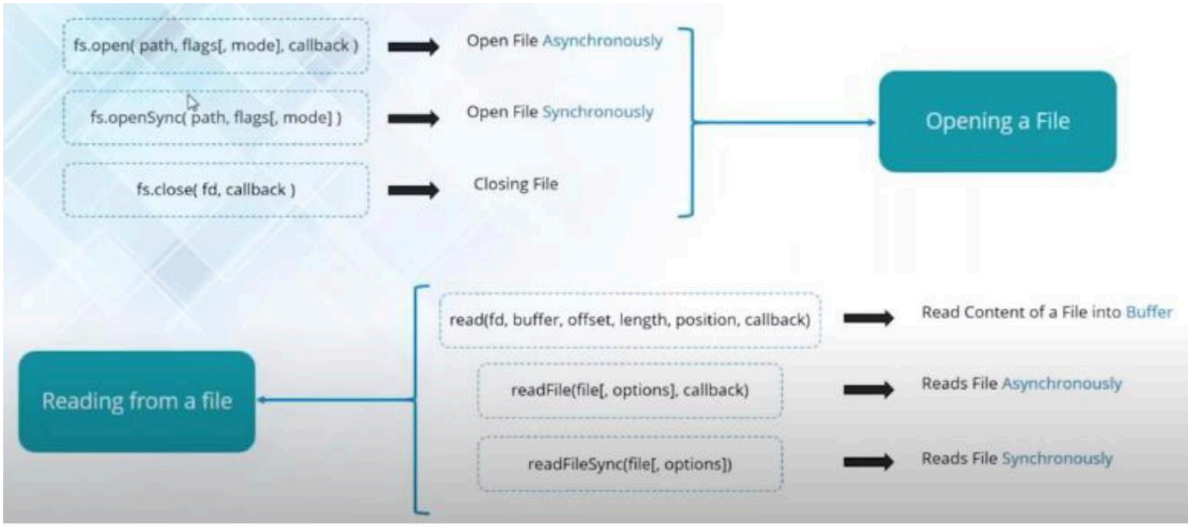


FS MODULE OPERATIONS :-

open | read | write | close
Common use for the File System module:

- Read files = fs.readFile() , fs.readFileSync()
- Create files = fs.writeFile()
- Update files = fs.appendFile()
- Delete files = fs.unlink()
- Rename files = fs.rename()

- Open File = fs.open()



Write file -

```
const fs = require('fs');
// Specify the file path and content
const filePath = 'files/HelloMsg.txt';
const fileContent = 'Hello, World!';

// Write the file
fs.writeFile(filePath, fileContent, (err) => {
    if (err) {
        console.error('An error occurred:', err);
    } else {
        console.log('File has been written successfully.');
    }
});
```

files/HelloMsg.txt

```
Hello, World!  //It will override existing text.
```

Append file -

```
const fs = require('fs');
```

```
// Specify the file path and content to append
const filePath = 'files/HelloMsg.txt';
const contentToAppend = 'New content to append.';

// Append to the file
fs.appendFile(filePath, contentToAppend, (err) => {
    if (err) {
        console.error('An error occurred:', err);
    } else {
        console.log('Content has been appended successfully.');
    }
});
```

files/HelloMsg.txt

Hello, World!New content to append.

Rename File -

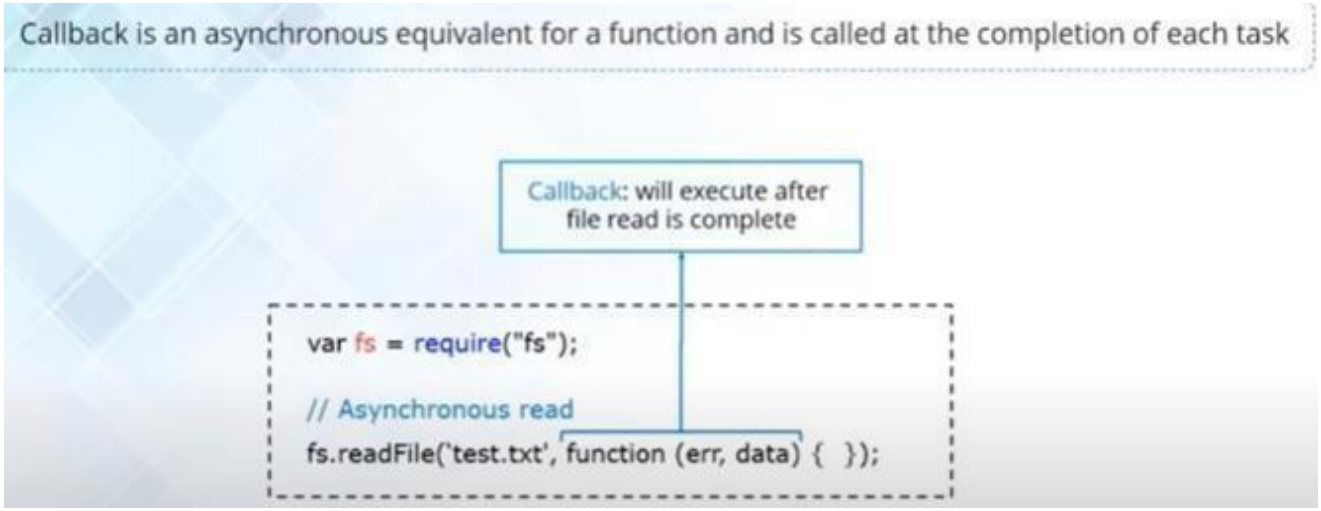
```
let fs = require("fs");

fs.rename("msg.txt", "msg1.txt", (err) => {

    if(err) {

        console.log(err)

    } else {

        console.log("renamed")

    }

})
```

Delete FIle -

```
let fs = require("fs");

fs.unlink("msg.txt", (err) => {

    if(err) {
```

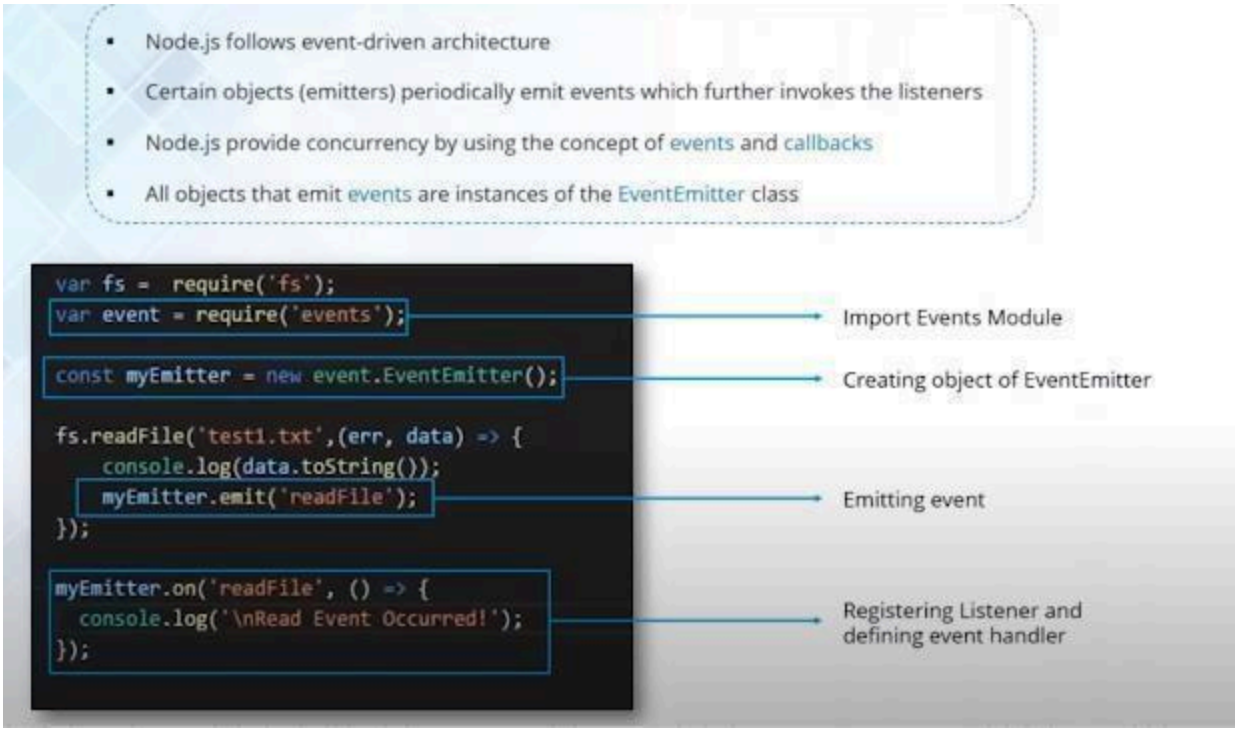
```
        console.log(err)

    } else {

        console.log("deleted")

    }

})
```

NODE JS CALLBACK:-



NODE JS EVENTS :-

We can emit our own event using the Event Emitter() Method.

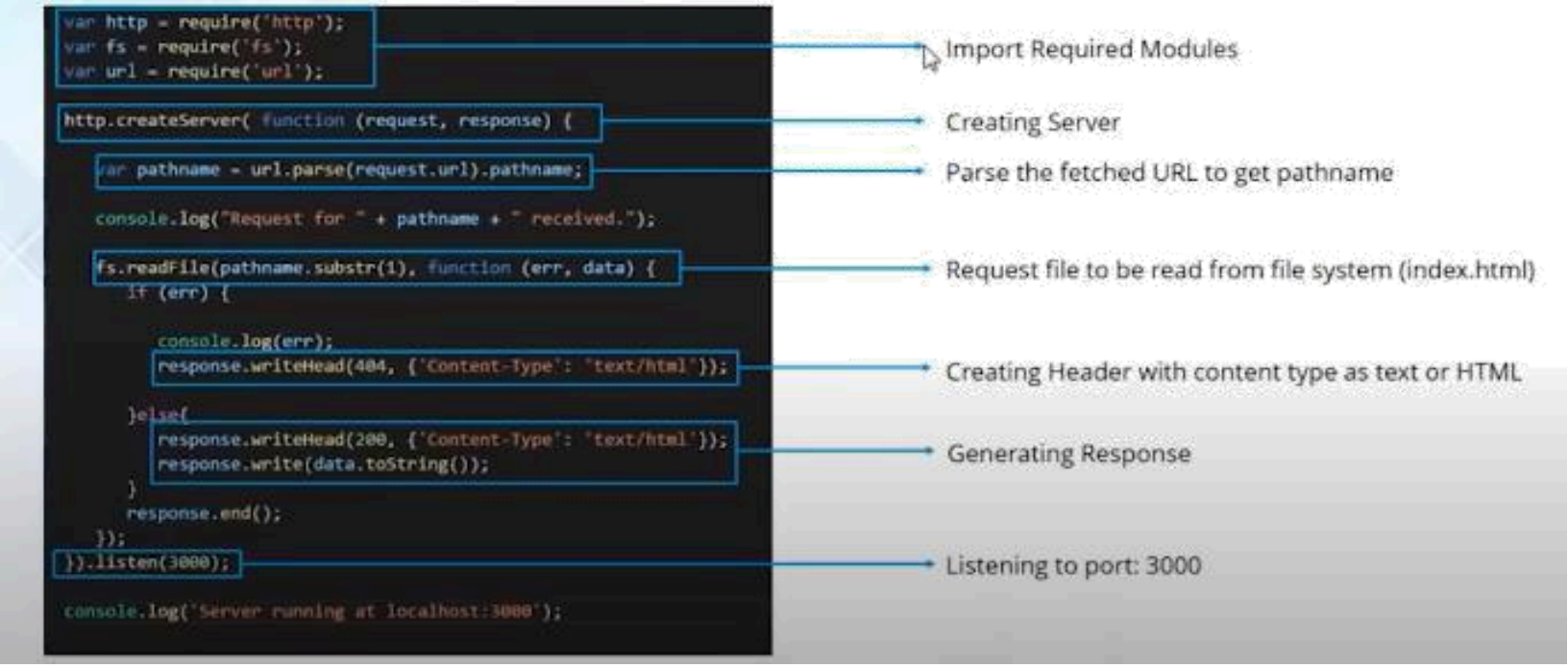



```
let fs = require("fs");
let events = require("events");

let eventEmiter = new events.EventEmitter();

fs.readFile("msg1.txt", (err, data) => {
    if(err) {
        console.log(err)
    } else {
        console.log(data.toString());
        eventEmiter.emit("openMessage");
    }
});

eventEmiter.on("openMessage", () => {
    console.log("file read success!!!");
});
```

NODE JS HTTP MODULE :-

Hypertext Transfer Protocol.

- To use the HTTP server and client one must require('http')
- The HTTP interfaces in Node.js are designed to support many features of the protocol




```
let fs = require("fs");

let http = require("http");

http.createServer((request, response) => {
    fs.readFile("index.html", (err, data) => {
        if(err) {
            console.log(err)
         } else {
            response.writeHead(200, {"content-type" : "text/html"});
            response.write(data.toString());
         }
        response.end();
    })
}).listen(5000);

console.log("Server is running on 5000 Port.")
```

Calling Event when Http connection got established -

```
let fs = require("fs");

let events = require("events");

let http = require("http");

let fileToRead = "folder1/txtFile.txt";

let appendMsg = "This message needs to be added afterwards."

let eventHandle = new events.EventEmitter();

http.createServer((request, response) => {

    fs.readFile("index.html", (err, data) => {

        if (err) {

            console.log(err)

        } else {

            response.writeHead(200, { "content-type": "text/html" })

            response.write(data.toString());

            eventHandle.emit("showMeOnAppend")

        }

        eventHandle.on("showMeOnAppend", () => {

            fs.appendFile(fileToRead, appendMsg, (err) => {

                if (err) {

                    console.log(err)

                } else {

                    console.log("append success")

                }
```

```
            });

        });

        response.end()

    });

}).listen(2000);

console.log("server is running on 2000")
```

Stubing in Node.js ?

Stubs are functions/programs that simulate the behaviors of components/modules.

Features of stub:

- Stubs can be either anonymous.
- Stubs are functions or programs that affect the behavior of components or modules.
- Stubs are dummy objects for testing.
- Stubs implement a pre-programmed response.