**Node JS**

**From Piyush Garg**

**(**[**https://www.youtube.com/playlist?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://www.youtube.com/playlist?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**Lecture 1 : What is NodeJS?**

**(**[**https://youtu.be/ohIAiuHMKMI?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/ohIAiuHMKMI?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

* Node.js is a runtime environment for JavaScript. You should have a good understanding of JavaScript before using Node.js.
* **Node.js is a JavaScript runtime environment for creating web servers and APIs.**
* Node.js is not a framework or library, but a runtime environment.
* It allows JavaScript to be executed outside of the browser.

**✦JavaScript is a language of the browser that allows interactivity.**

* JavaScript can only be executed inside a browser because every browser has a JavaScript engine.
* Executing JavaScript outside the browser was not possible in the past.

**✦V8 engine is the most popular JavaScript engine**

* Different browsers have their own JavaScript engines
* NodeJS uses the V8 engine for executing JavaScript outside the browser

**✦NodeJS allows executing JavaScript outside the browser**

* NodeJS uses V8 engine extracted from Chrome
* V8 engine embedded with C++ allows JavaScript to interact with native machine

**✦JavaScript can now talk to native machines and perform tasks like file handling**

* JavaScript internally uses C++ through the V8 engine
* Node.js is a runtime environment for JavaScript to create web servers

**✦Node.js has a runtime environment for executing JavaScript code.**

* The V8 engine is used to execute JavaScript in the browser.

**✦Node.js is a runtime environment for JavaScript**

* Node.js allows JavaScript code to be executed outside of a browser, such as in a terminal
* Node.js is an open source cross-platform JavaScript runtime environment

**✦Node.js is a runtime environment for JavaScript.**

* Node.js is built on JavaScript, so a good understanding of JavaScript is necessary.
* A JavaScript tutorial series is recommended for those who are unfamiliar with JavaScript.

Browser with v8 Engine = Can Execute JavaScript within(INSIDE) the browser.

**BUT**

Browser with v8 engine Embedded(With the help of) C++ = Can Execute JS OUTSIDE the browser.

**Lecture 3 : Hello World NodeJS**

**(**[**https://youtu.be/XhCs5cTYW\_8?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/XhCs5cTYW_8?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/Hello%20World**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/Hello%20World)**)**

* First Node JS File and how to execute it.

**✦Setting up a new NodeJS project**

Creating a new folder and opening it in a code editor

Organizing code files in separate folders for better tracking

**✦Create and Run Node JS file**

Create a new JS file named **hello.js** with a simple JavaScript code

Run the JS file using the terminal by specifying the file name as **node hello.js** and enter or simply node hello and enter

**✦Issues with window object and console in browser environment**

Window and console are not defined in NodeJS environment

Certain JavaScript functions behave differently in different environments

**✦NodeJS contains core functionalities needed on the server side**

NodeJS is essentially JavaScript with the core functionality for server-side use

Features like cryptography, file handling, and unnecessary client-side functionalities have been added or removed accordingly

**✦Node Package Manager is used to manage packages and dependencies in NodeJS projects.**

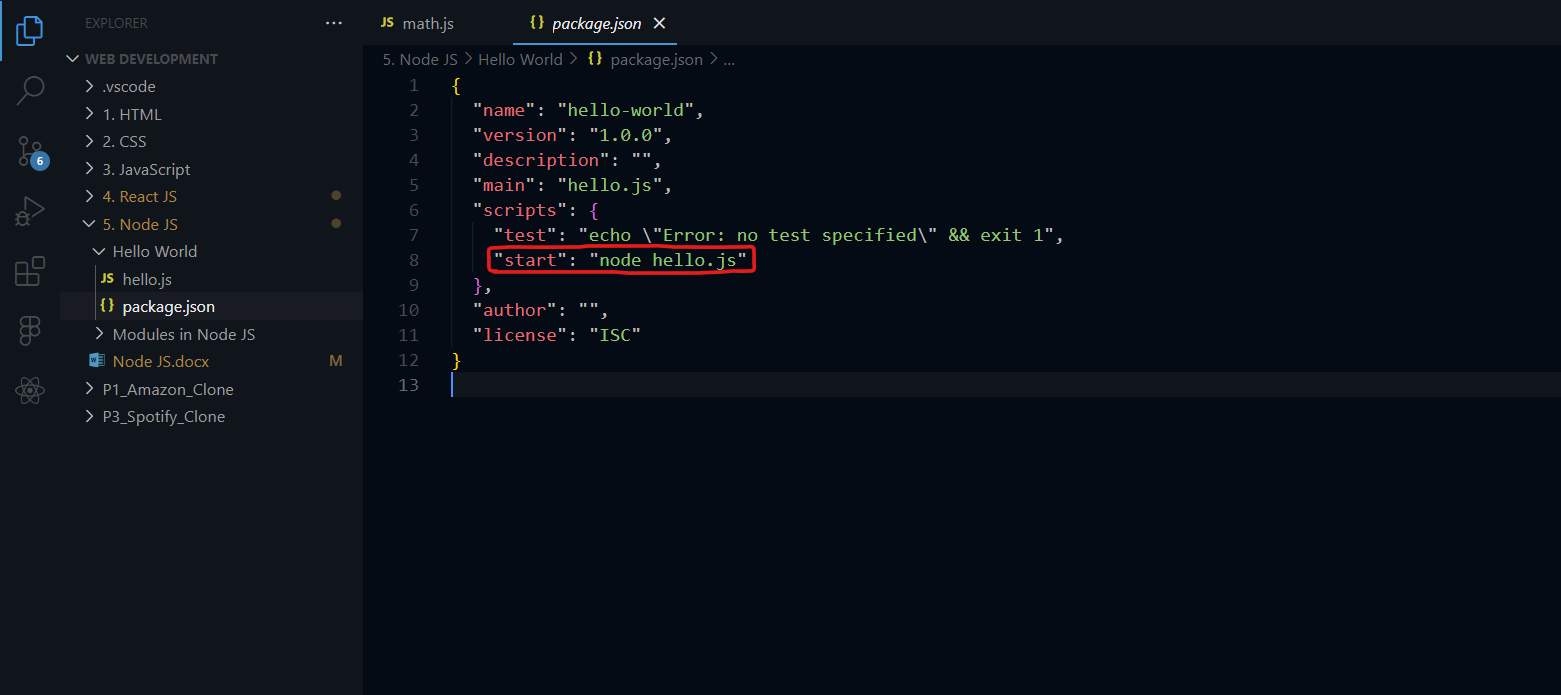
Using 'npm init' command we can initialize a new project and create a template.

During initialization, it creates a file for the project and asks for package name and other details.

**✦You can create your own scripts in NodeJS**

You can name the script whatever you want as follows

When you command as ‘npm start’ it will automatically converted into ‘node hello.js’ and will execute the file



**✦NodeJS provides a convenient way to create and manage scripts and dependencies.**

NodeJS allows for easy installation of dependencies and running scripts.

NodeJS includes a pre-generated package.json file for easy configuration and management of scripts and dependencies.

**Lecture 4 : Modules in NodeJS**

**(**[**https://youtu.be/FSRo41TaHFU?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/FSRo41TaHFU?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/Modules%20in%20Node%20JS**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/Modules%20in%20Node%20JS)**)**

**✦Understanding modular programming in NodeJS**

Modular programming involves dividing the code base into small modules for better organization

This allows for creating different files for different functionalities and dividing the code accordingly. For example, we have different JS file that contains math function and we use it in different file.

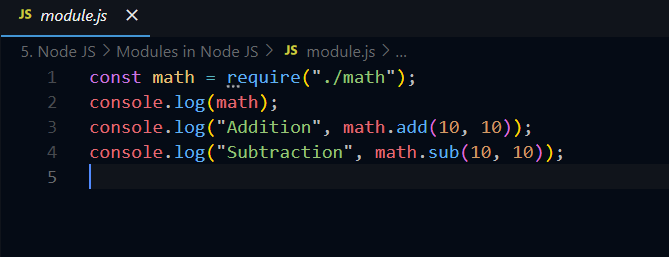
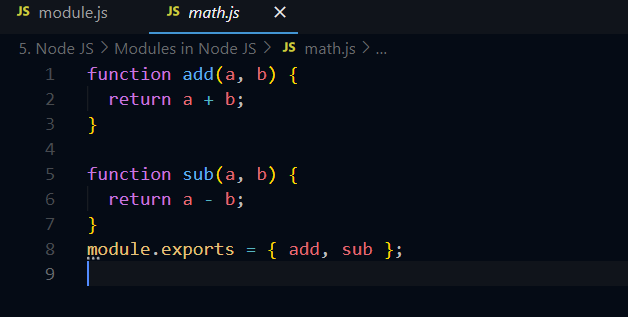
**✦Importing modules in NodeJS**

Use the require() function to import modules

Specify the module name or file path to import

**✦Exporting functions in NodeJS modules**

Functions can be exported from a NodeJS module using module.exports

Exported functions can be accessed and used in other files

**✦Exporting and importing functions and objects in NodeJS**

Functions and objects can be exported using the 'export' keyword

They can be imported using the 'required' function and utilized in other modules

Moreover, Node JS has built in packages which is also need to be imported as require(‘fs’). It is for file handling.

**Lecture 5 : File Handling in NodeJS**

**(**[**https://youtu.be/YazJFb\_i4A0?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/YazJFb_i4A0?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/File%20Handling**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/File%20Handling)**)**

**✦Understanding file system in NodeJS and different types of tasks**

Explaining asynchronous and synchronous tasks in file handling

Differentiating between blocking and non-blocking tasks

**✦File handling in NodeJS involves reading and decoding different types of files.**

Files can be in different formats such as text or binary.

Decoding files requires understanding the file type and using appropriate methods.

**✦File handling in NodeJS involves synchronous and asynchronous operations with file reading and writing.**

Synchronous file handling returns the result directly, while asynchronous file handling uses callback functions to handle the result or error.

Understanding the concepts of synchronous and asynchronous operations in file handling is important for back end developers and anyone creating their own applications or startups.

**✦File handling in NodeJS allows creating, copying, and deleting files**

File handling includes read file, write file, append file, copy file, and delete file operations

Additional options such as unlink, file status, and sync can be used for more control over file handling



**Lecture 6 : How NodeJS Works?**

**(**[**https://youtu.be/y0aTs56DJWk?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/y0aTs56DJWk?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

**✦NodeJS architecture and request handling**

NodeJS architecture starts from client request to server

Incoming requests are handled within the event loop

✦**NodeJS uses event loop to handle requests efficiently**

Requests are picked up from a queue based on FIFO principle

Event loop differentiates between blocking and non-blocking operations to process requests

When the request has come it is stored it in Event Queue.

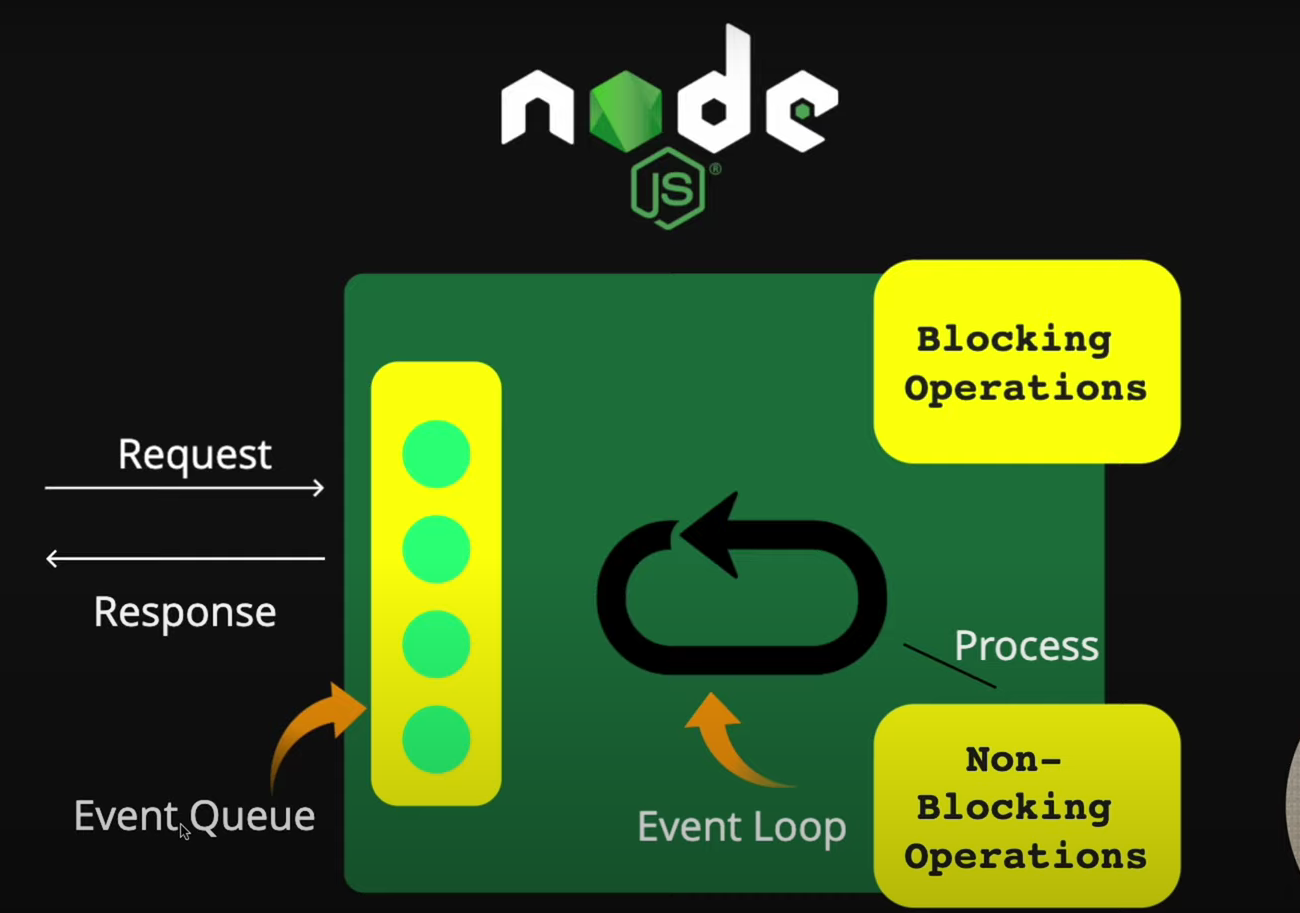
Event Loop is a Loop set on Event Queue. Continuously checks on Event Queue.

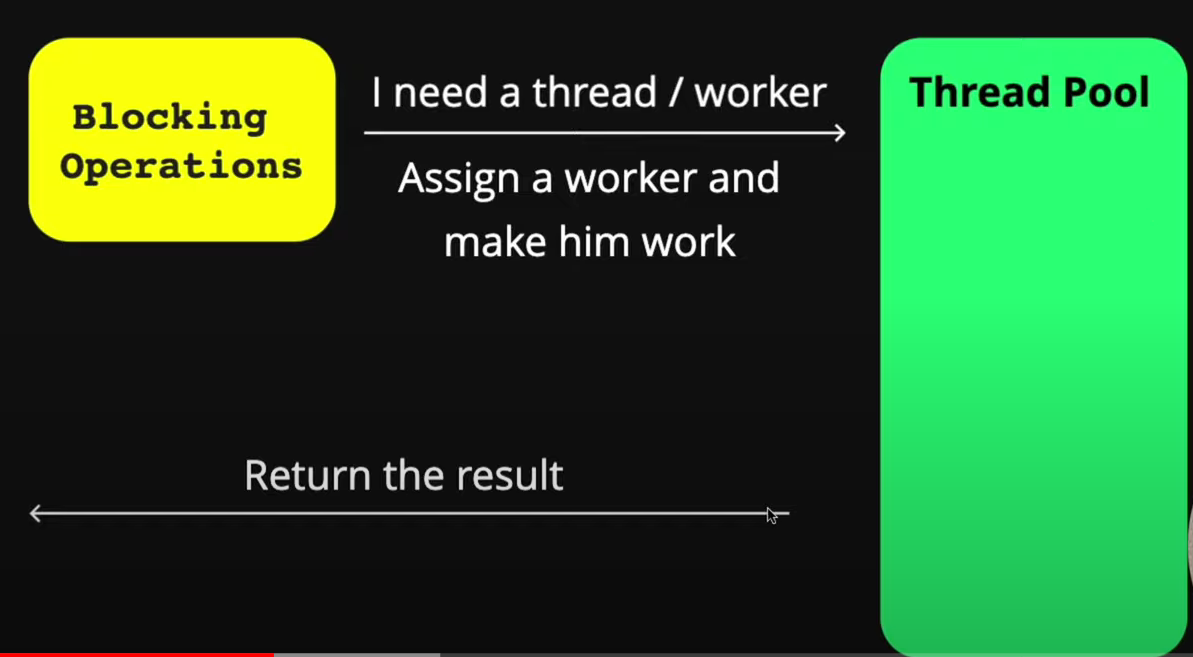
**(Blocking and Non Blocking Operations explained below this section)**

If the Request is Non-Blocking(Asynchronous) operation, then It executes and gives response.

If the Request is Blocking(Synchronous) operation, then the operation needs a thread to Handle the operation. The thread is assigned from the Thread Pool. While Blocking Operation are being executed, it blocks other operation until the currently executing operation is completed.

So Blocking Operations are not scalable. So it is ensured that to use Non Blocking Operations.

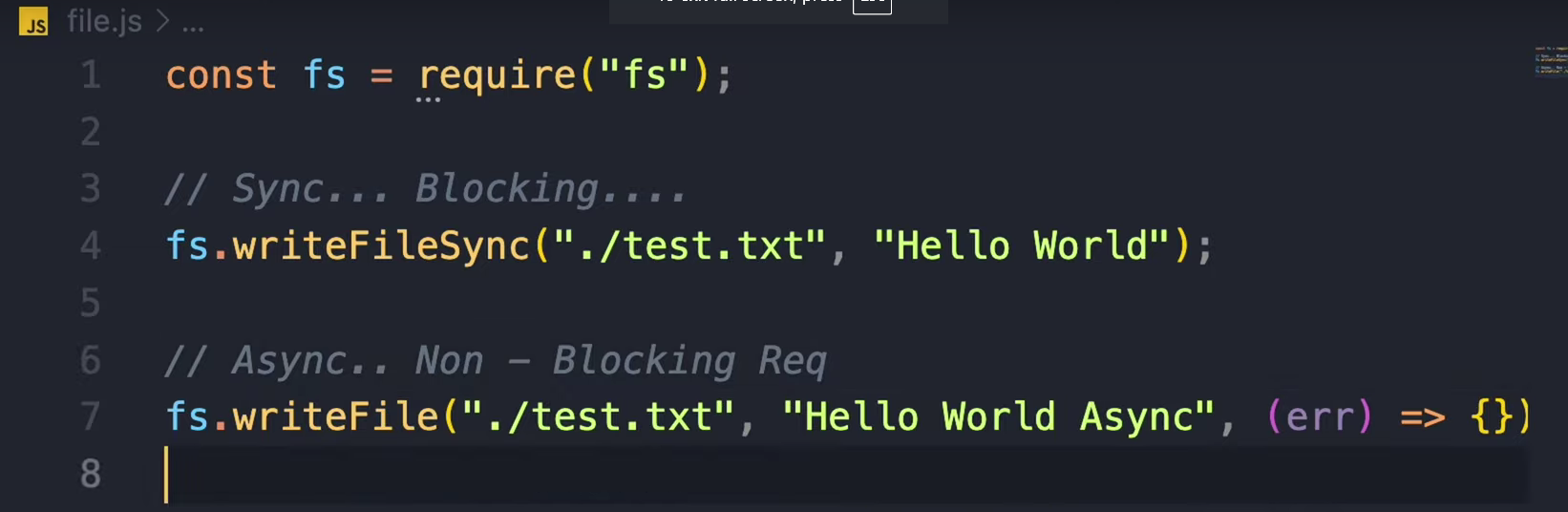


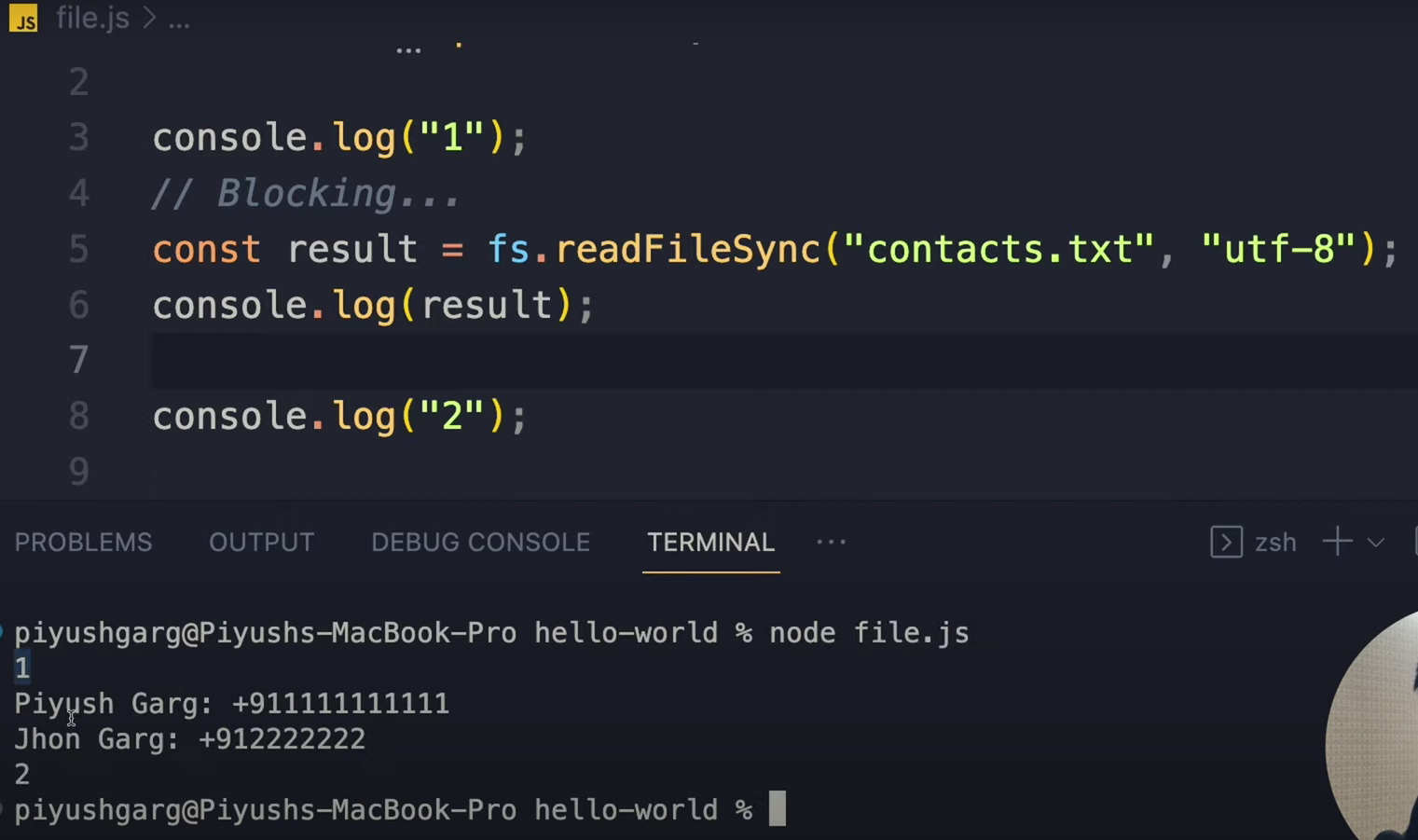


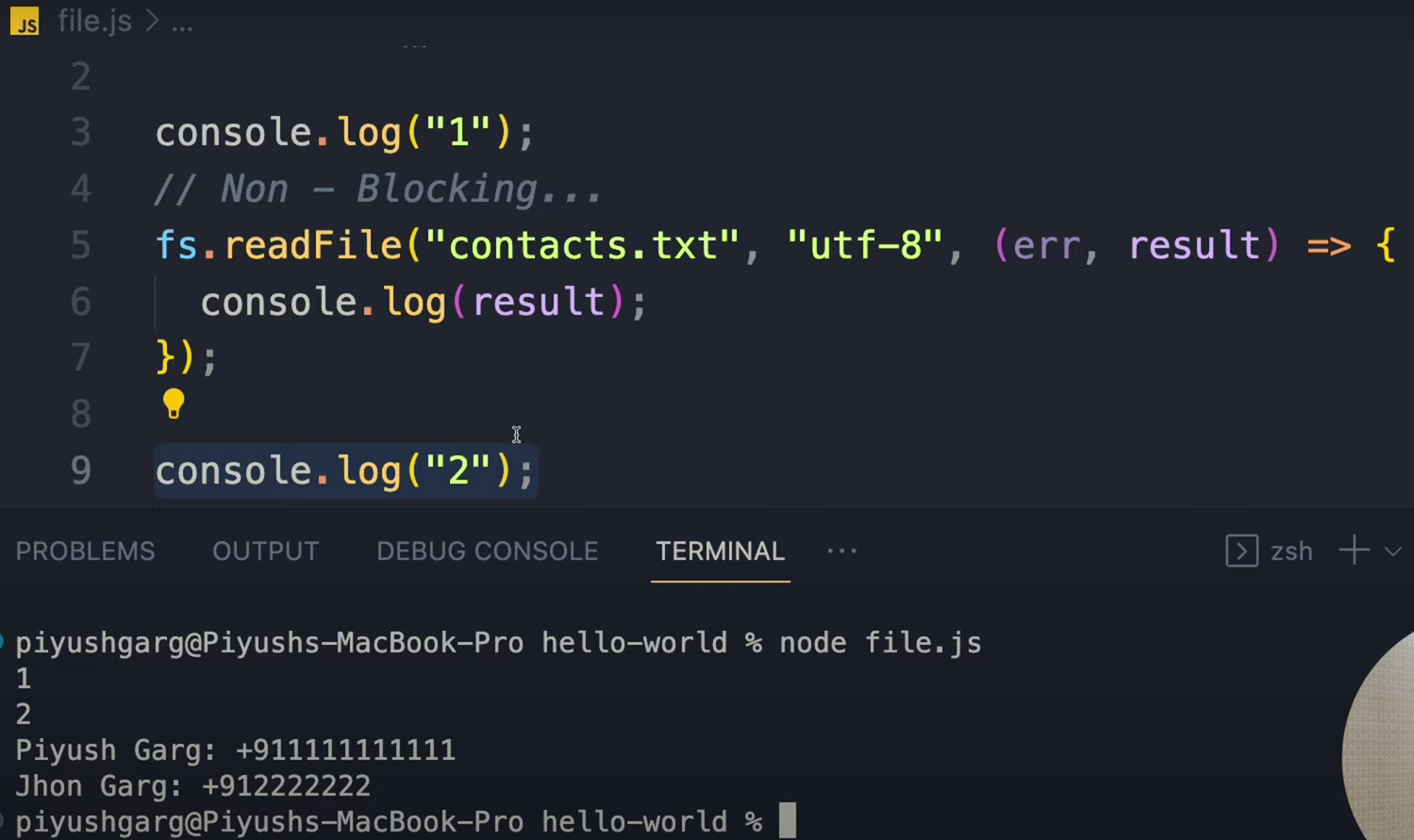
✦**NodeJS handles non-blocking and blocking operations differently**

Consider…

* Blocking Operation as Synchronous Operation: That means first current operation will be executed till then no next operation will be executed.(Everything will go sequentially) (Means Blocks other operations)
* Non-Blocking Operation as Asynchronous Operation: That means currently executing operation will be executing. Till then next operation will be executed. (Means Does not Block other operations)







Non-blocking operations are processed and the response is sent to the user

Blocking operations are handled by the thread pool, where a worker is assigned to fulfill the operation

✦**NodeJS architecture and event loop processing**

The event loop picks up requests and delegates non-blocking operations or assigns workers for blocking operations.

Thread pool with workers handles blocking operations, and limited thread availability can lead to processing delays.

✦**NodeJS has scalability issues with blocking operations.**

When the server uses blocking operations, it can cause excessive waiting time for users.

The difference between blocking and non-blocking requests in NodeJS is crucial for understanding its operation.

✦**Understanding the sequencing and execution in NodeJS**

In NodeJS, the execution starts from the top and blocks the thread, then continues once the file is read.

Converting code to non-blocking by removing synchronous and adding a callback function.

✦**NodeJS uses non-blocking requests to handle asynchronous operations.**

Non-blocking requests allow the program to continue executing other tasks while waiting for a response.

NodeJS event loop efficiently handles non-blocking operations and can be configured to increase worker threads.

✦**NodeJS architecture and thread management**

Maximum thread capacity is dependent on the machine and CPU cores

Operating System provides information about the computer and manages thread allocation.

**"Blocking threads are non-blocking" is not entirely accurate.**

Here’s a clearer way to understand it:

1. **Nature of the operation:**
   * **Blocking operation:** Stops the current thread until it’s done, so the thread can't do anything else during this time.
   * **Non-blocking operation:** Allows the thread to continue executing other code while waiting for the operation to complete.
2. **Execution model in Node.js:**
   * **Single-threaded event loop:** Manages non-blocking I/O tasks and schedules them.
   * **Worker threads:** Handle blocking operations asynchronously, so the main thread can still be responsive and handle other tasks.

**Key Point:**

**Node.js uses worker threads and the event loop to handle blocking operations in a non-blocking way, keeping the main thread free to manage other tasks.**

**Additional Details:**

Node.js decides whether to use the main thread or a worker thread for tasks based on:

* **Nature of the operation:**
  + Long-running or blocking tasks go to worker threads.
  + Short-lived or non-blocking tasks stay on the main thread.
* **Available resources:**
  + If CPU and memory are limited, Node.js might run blocking tasks on the main thread to avoid the overhead of worker threads.
* **Application configuration:**
  + Some libraries and frameworks can influence whether a task uses the main thread or a worker thread.
* **Node.js version and configuration:**
  + Different versions and settings of Node.js might change how tasks are allocated.

Even for a blocking operation like **fs.readFileSync**, Node.js might run it on the main thread if:

* The operation is very quick and doesn’t block much.
* Using a worker thread would add more overhead than just waiting.
* The system has limited resources, making worker threads less efficient.

**Summary:** Node.js smartly manages tasks between the main thread and worker threads to optimize performance and resource use.

# **Lecture 7 : Building HTTP Server in NodeJS**

**(**[**https://youtu.be/ZQsrcayZcSk?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/ZQsrcayZcSk?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/7.%20Building%20HTTP%20Server**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/7.%20Building%20HTTP%20Server)**)**

**✦Creating an HTTP web server using NodeJS**

* Setting up the project with npm and creating a package.json file for basic configuration
* Creating a file named index.js, a good practice for a project's entry point

**✦Creating HTTP server using NodeJS.**

* Using the 'http' package, a server can be created in NodeJS with the 'http.createServer' function.
* The 'http.createServer' function takes a callback function called 'requestListener' to handle incoming requests by processing the request and sending the response.

**✦Handling requests and sending responses in NodeJS**

* Requests and data are handled using the request object
* Responses are sent using the response object, and the server runs on a specific port

**✦Setting up a callback function for starting the server**

* Using a start script to initialize the server on port 8000(You can choose any)
* Accessing the server on localhost and receiving a response from the server

**✦Headers provide extra information about the request.**

* Headers include details such as the request origin(request is came), IP address, and client information.
* Changing any details in the request object requires restarting the server.

**✦Building HTTP Server in NodeJS**

* The server stores information about the type of request, access rights, and user making the request.
* The server can handle various types of responses such as text, images, and HTML.

**✦Creating a log for web server requests**

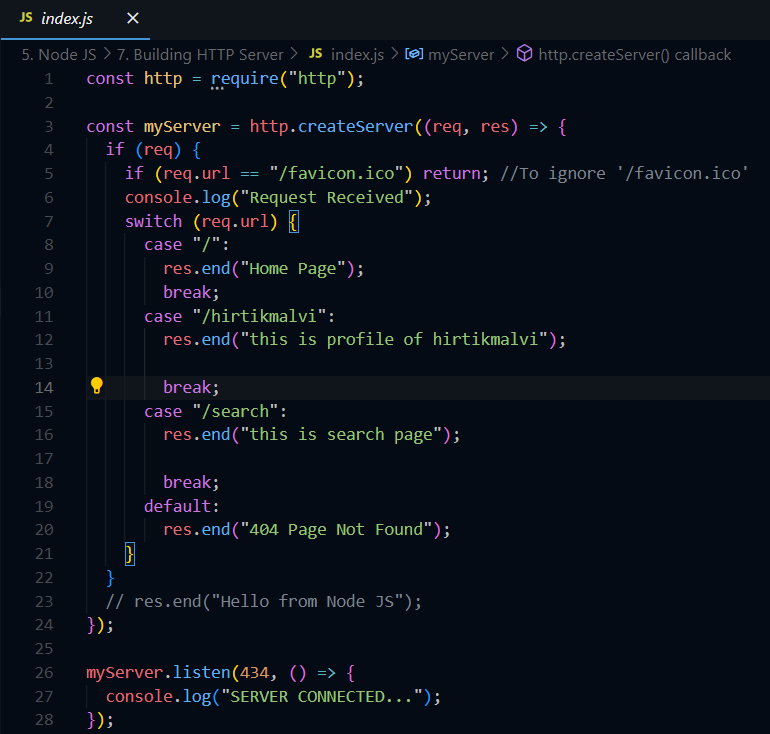
* Logging request time and user's IP address in a file
* Responding to requests and logging requested paths

**✦Creating multiple pages using switch case on request URL.**

* Using switch case to handle different request URLs and creating specific responses.
* Demonstrating the creation of a basic server with multiple pages using NodeJS HTTP module.

**✦Always support non-blocking requests for better performance**

* Blocking requests can kill threads and lead to long wait times for users
* Avoid CPU intensive tasks to prevent blocking the event head pool



# **Lecture 8 : Handling URL's in NodeJS**

**(**[**https://youtu.be/Nt-AsZh5woE?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/Nt-AsZh5woE?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/8.%20URL%20Handling**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/8.%20URL%20Handling)**)**

****

**✦Understanding the components of a URL**

* A URL stands for Uniform Resource Locator and consists of major components like protocol, domain, and IP address.
* The protocol in a URL represents the communication rules for the browser, while the domain serves as a user-friendly name for the website.

**✦URL is a user-friendly name for an IP address.**

* URL consists of the protocol, domain, and path.
* Path can be a root path, a nested path, or represent different pages on a website.

**✦ require(”url”)**

* **We have seen that req.url gives us path of the URL. But what if we have to deal with the Query Provided with the URL? And if we need the Parameters of Query String then?**
* **Then this imported url helps. It gives object that contains many information of the requested URL.**
* **Pathname, query, protocol name, port number etc etc**

**✦Understanding URL parameters in NodeJS**

* URL parameters include domain, path, query parameters, and any additional information passed in the URL.
* Query parameters allow passing values, such as user ID, to the server, separated by the question mark in the URL.

**✦Handling URL's in NodeJS involves reducing long URL requests to improve efficiency.**

* URLs cannot have spaces and need to be well-structured for efficient communication with servers.
* The process involves generating a key for the search request, reducing the URL length and achieving clear communication with the server.

**✦Using the HTTP module to handle and parse request URLs in NodeJS.**

* The HTTP module provides the path we request in the URL.
* We can use an external package to parse the URL and extract its components such as protocol, host, etc.

**✦Understanding the process of installing and handling URLs in NodeJS.**

* Installing a package using npm install creates a dependency inside the package URL and brings a specific version of the package.
* Accidentally deleting a model or package URL can be resolved by using npm install again to reinstall the dependency.

**✦Handling URL path parameters in NodeJS**

* Explaining the process of passing multiple parameters in the URL, including user ID and name
* Demonstrating how to exclude parameter strings while handling the URL path

**✦Handling URL's in NodeJS includes parsing query parameters and processing user requests.**

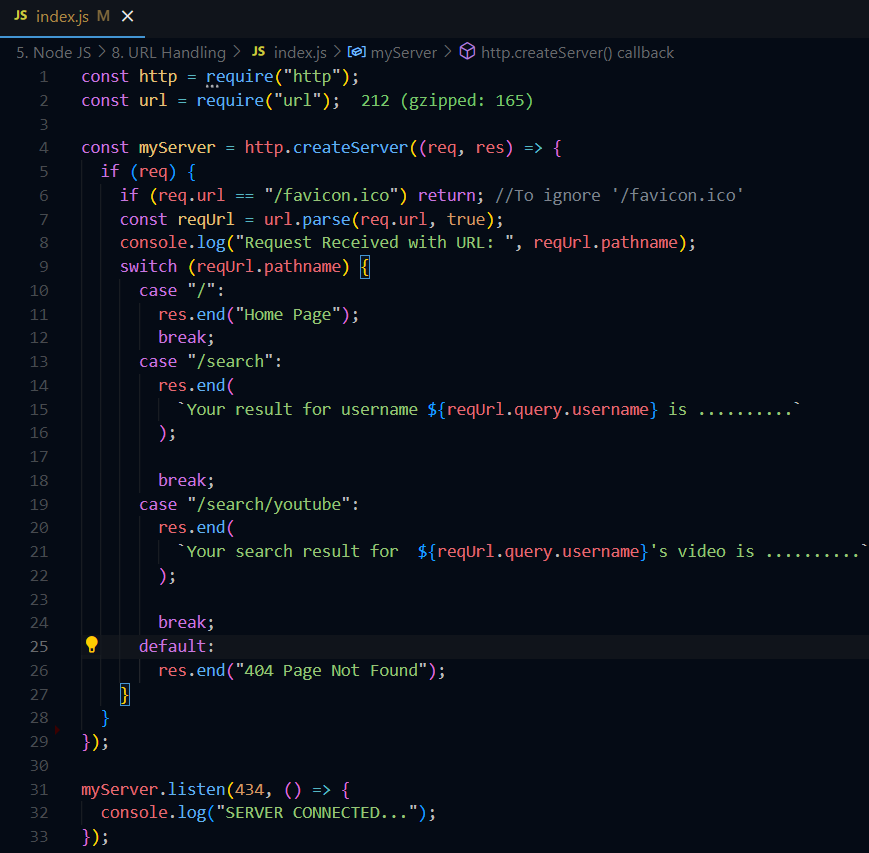
* Understanding query parameters and how to extract data from the URL.
* Exploring the process of handling user requests and generating appropriate responses.

**✦Handling URL's in NodeJS involves components like protocol, domain, path, and query parameters.**

* The URL consists of a protocol, domain, and path with query parameters separated by a question mark and ampersand.
* Handling URL in NodeJS involves interacting with the database and sending data based on the parameters.

**✦Handling URL's in NodeJS requires careful attention.**

* Proper validation and sanitization of input URLs is crucial for security.
* NodeJS provides built-in modules such as 'url' to handle and manipulate URLs.



# **Lecture 9 : HTTP METHODS**

**(**[**https://youtu.be/kREAjKyPbSI?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/kREAjKyPbSI?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

**HTTP Methods**

HTTP methods are ways for a client (like a web browser) to communicate with a server. Here are the main ones:

1. **GET:**
   * **Purpose:** Retrieve data from the server.
   * **Example:** Loading a webpage or fetching a list of items.
2. **POST:**
   * **Purpose:** Send data to the server to create a new resource.
   * **Example:** Submitting a form to create a new user account.
3. **PUT:**
   * **Purpose:** Send data to the server to update an existing resource.
   * **Example:** Updating user information in a database.
4. **DELETE:**
   * **Purpose:** Remove a resource from the server.
   * **Example:** Deleting a user account.
5. **PATCH:**
   * **Purpose:** Partially update an existing resource.
   * **Example:** Changing a user's email address without altering other details.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **PUT** | **PATCH** |
| **Purpose** | Update an entire resource | Partially update a resource |
| **Data Sent** | Full representation of the resource | Only the changes or updates |
| **Idempotency** | Yes (repeated requests have the same effect) | Yes (repeated requests have the same effect) |
| **Use Case** | Replace an existing user profile with a new one | Update just the user's email address |
| **Example** | Updating all user details (name, email, etc.) | Updating just the user's email address |

**Summary:**

* **PUT**: Sends the complete updated resource, effectively replacing the old one.
* **PATCH**: Sends only the specific changes to the resource.

# **Lecture 10 : Getting Started with Express and NodeJS**

**(**[**https://youtu.be/N2-FyBBxOZA?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/N2-FyBBxOZA?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

✦Introduction to the problems that Express framework solves.

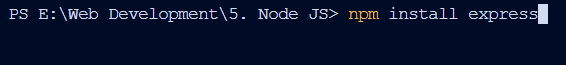
* Express framework solves the confusion in handling server code using the HTTP module.
* The problem is the confusing structure of callback functions and the need to make the code easier to manage.
* Example is shown: making server using without Express vs Using Express

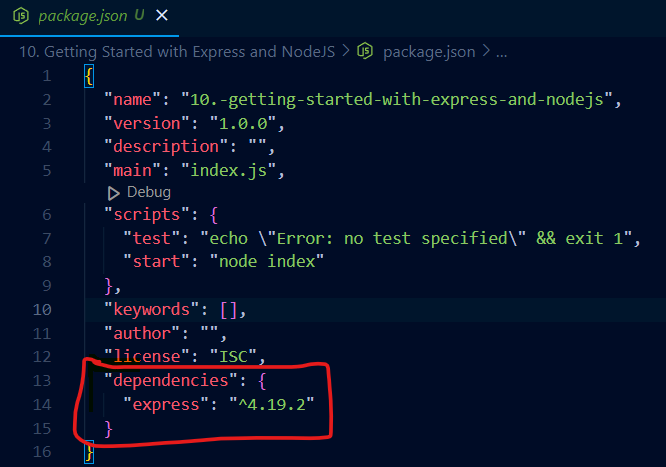
✦Express requires creating individual cases for different HTTP methods like get, post, put, patch, delete, which can be complex and time-consuming.

* Handling multiple HTTP methods requires creating separate cases, which can be a cumbersome process.
* Passing parameters from inside the HTTP.create server handler requires the use of different modules and packages.

✦Installing and initializing Express

* Installing Express using npm and package.json file





* Importing Express and creating an application

✦Creating handler functions for different HTTP methods and routes in NodeJS using Express.

* Understanding how to assign handlers for different HTTP methods such as GET, POST, PUT, DELETE.
* Registering handler functions for different routes and testing the functionality through requests.

✦Using Express to handle routing and requests in NodeJS

* The application.get method is used to define a route and a callback function for handling the HTTP GET requests.
* The application.post method is used to define a route and a callback function for handling the HTTP POST requests.

✦Express and NodeJS make code clean and modularized

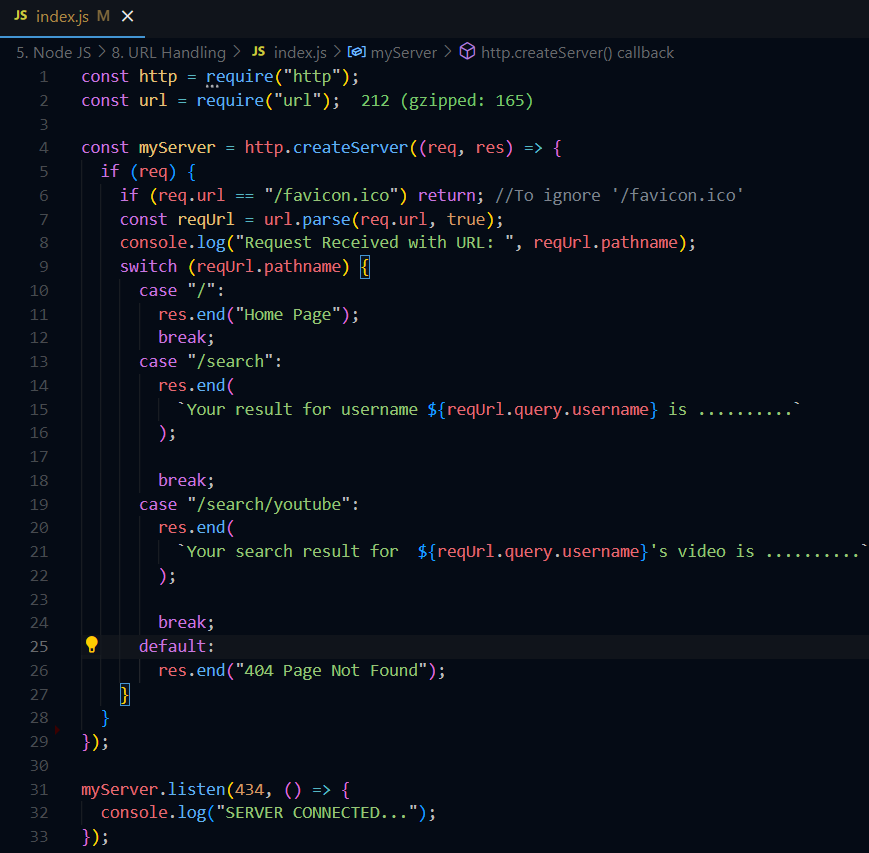
* Easy handling of routes and dynamic outputs
* Code becomes clean and organized, minimizing the need for additional steps

✦Using Express for handling HTTP requests and creating a server.

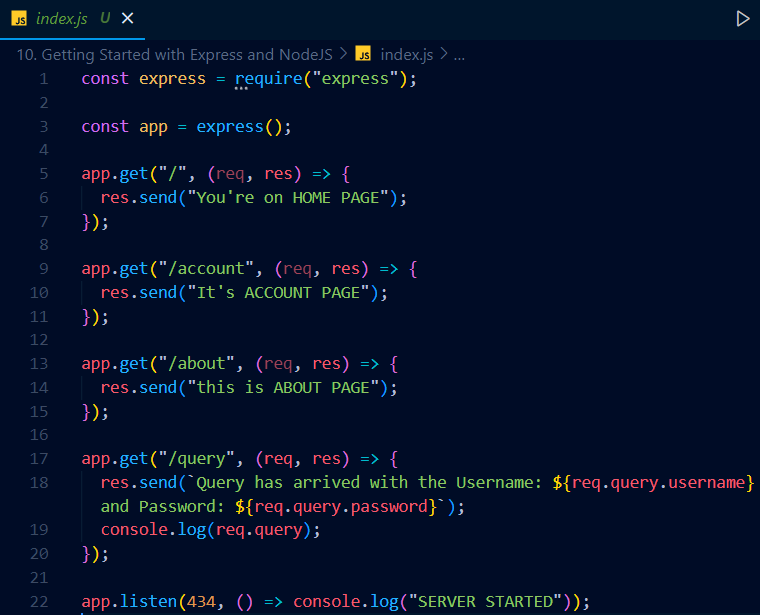
* Express eliminates the need for handling HTTP modules separately.
* Express simplifies the process of creating a server and handling listener installation.

✦Understanding Express and its benefits in server-side coding

* Express makes server-side code writing less painful and more efficient, providing a good structure and functionality.
* Basic routing using get, post, put, patch, delete methods and defining paths with corresponding handlers.



**Before Express:**

****

**After Express:**

# **Lecture 11 : How Versioning Works in NodeJS?**

**(**[**https://youtu.be/ORmB\_ABimjM?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/ORmB_ABimjM?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

In Node.js (and in many JavaScript projects that use npm), version numbers often follow Semantic Versioning (SemVer).

The version **^4.19.2** can be broken down as follows:

* **4**: Major version. Indicates significant changes, potentially including backward-incompatible changes.
* **19**: Minor version. Adds new features but is backward-compatible.
* **2**: Patch version. Includes bug fixes and minor changes that are backward-compatible.

The ^ symbol is a **caret**. It tells npm to allow updates upto the number beside the caret symbol.

* ^4.19.2 means any version from 4.19.2 to before 5.0.0. means no updates after ^4.x.x is allowed

**Summary:**

* ^4.19.2 allows updating to versions 4.x.x as long as the major version stays the same (4).

# **Lecture 12 : What is REST API?**

**(**[**https://youtu.be/cJAyEOZQUQY?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/cJAyEOZQUQY?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**