**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.**

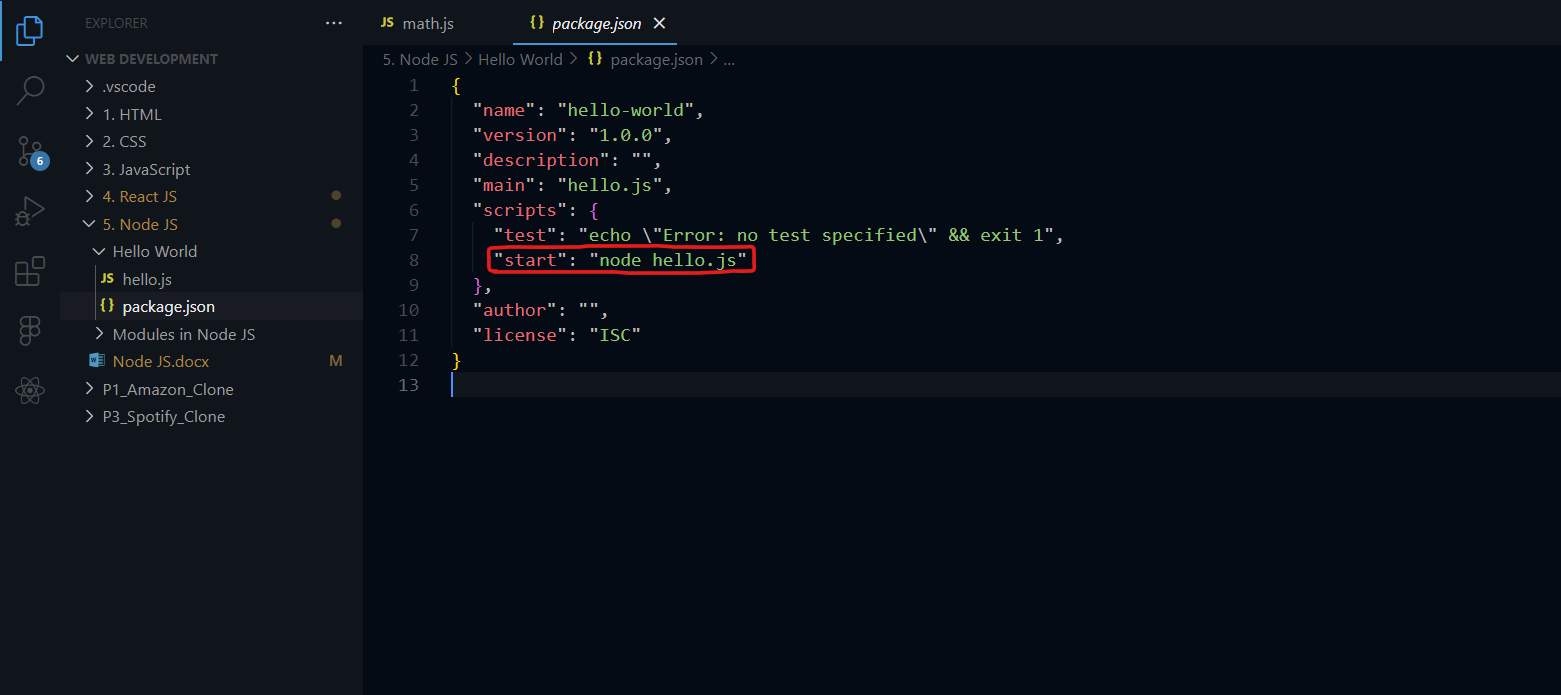
Using 'npm init' command we can initialize a new project and create a template.

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.

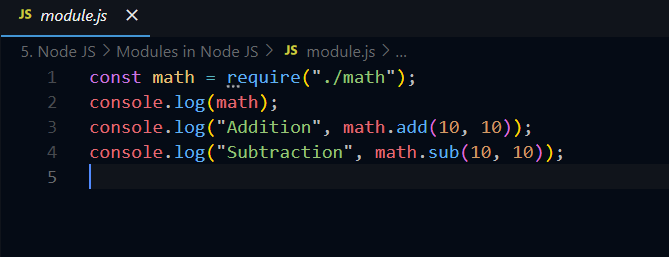
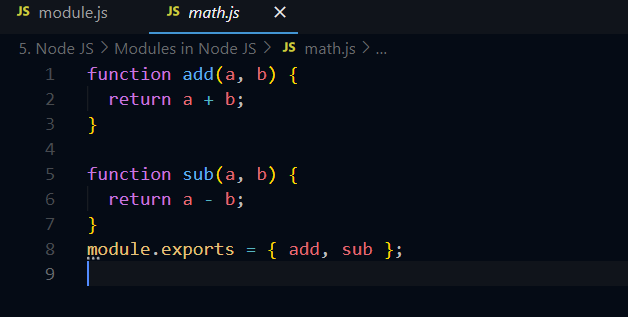
**✦Importing modules in NodeJS**

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.

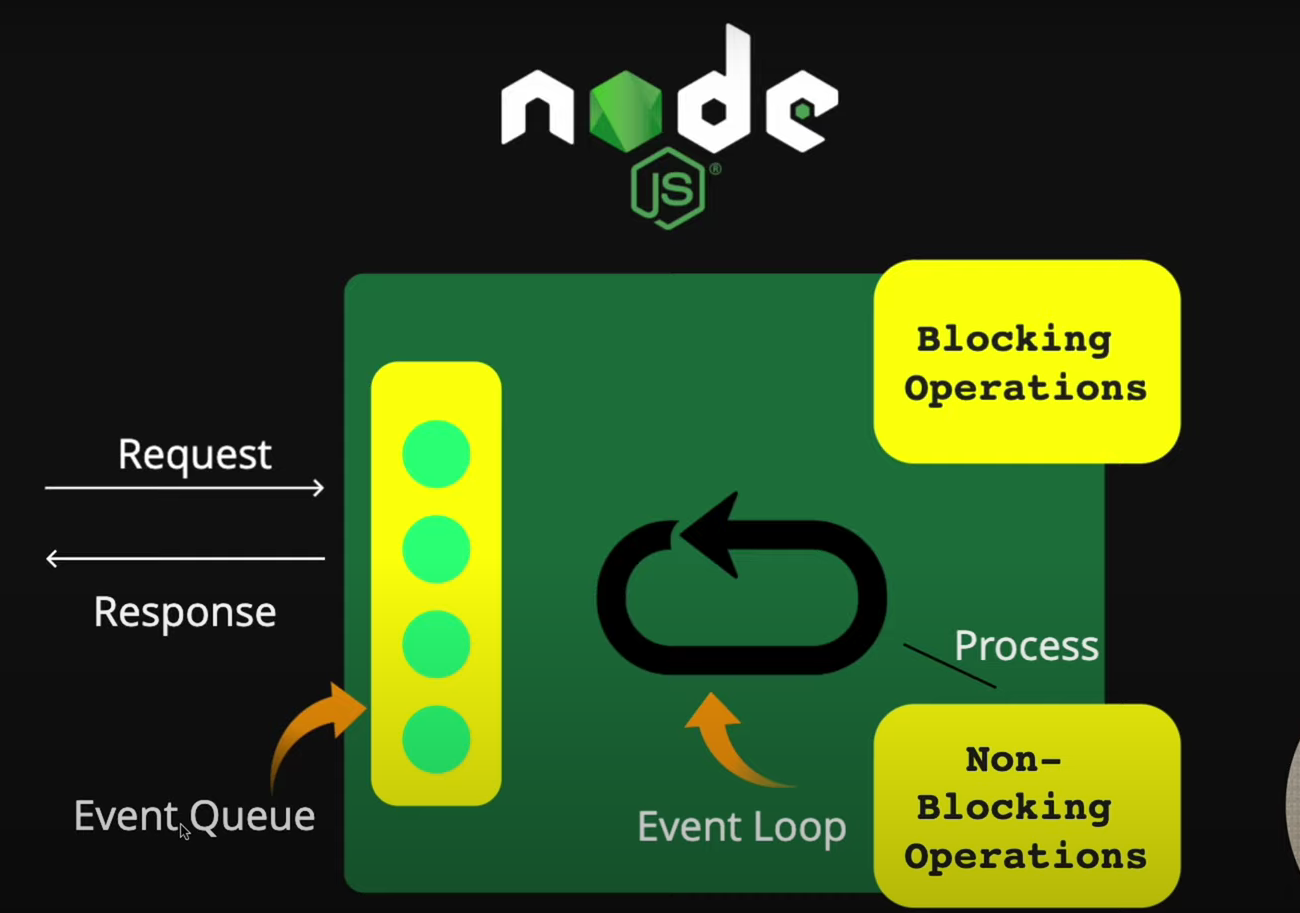
Event Loop is a Loop set on Event Queue. Continuously checks on Event Queue.

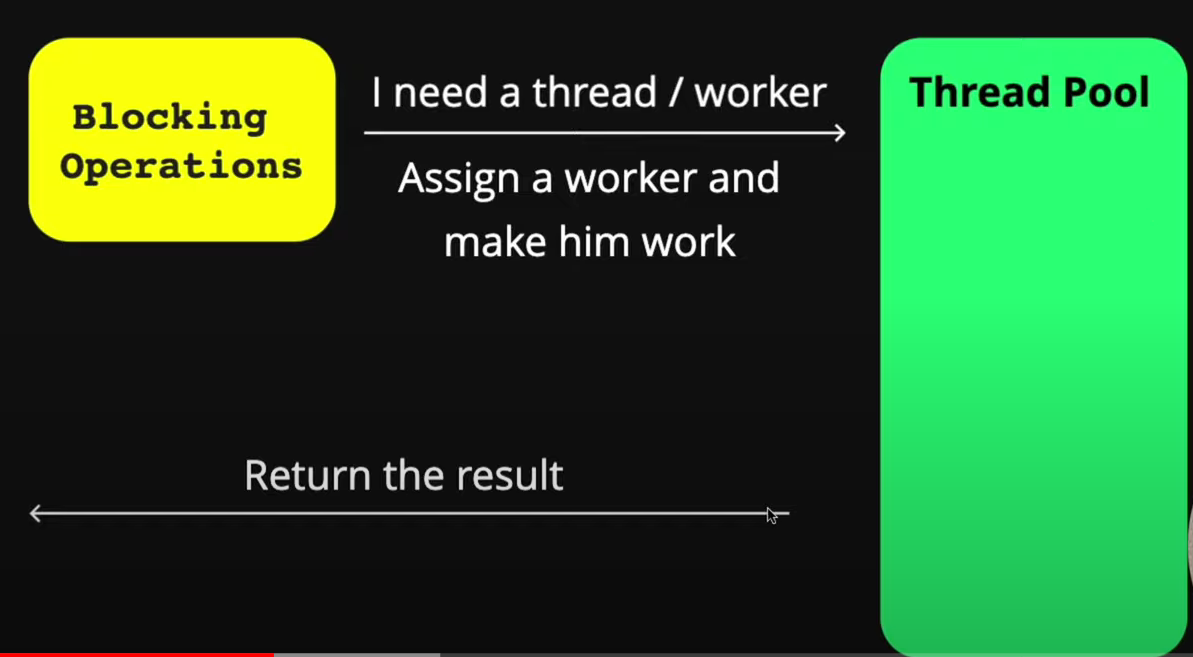
**(Blocking and Non Blocking Operations explained below this section)**

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.



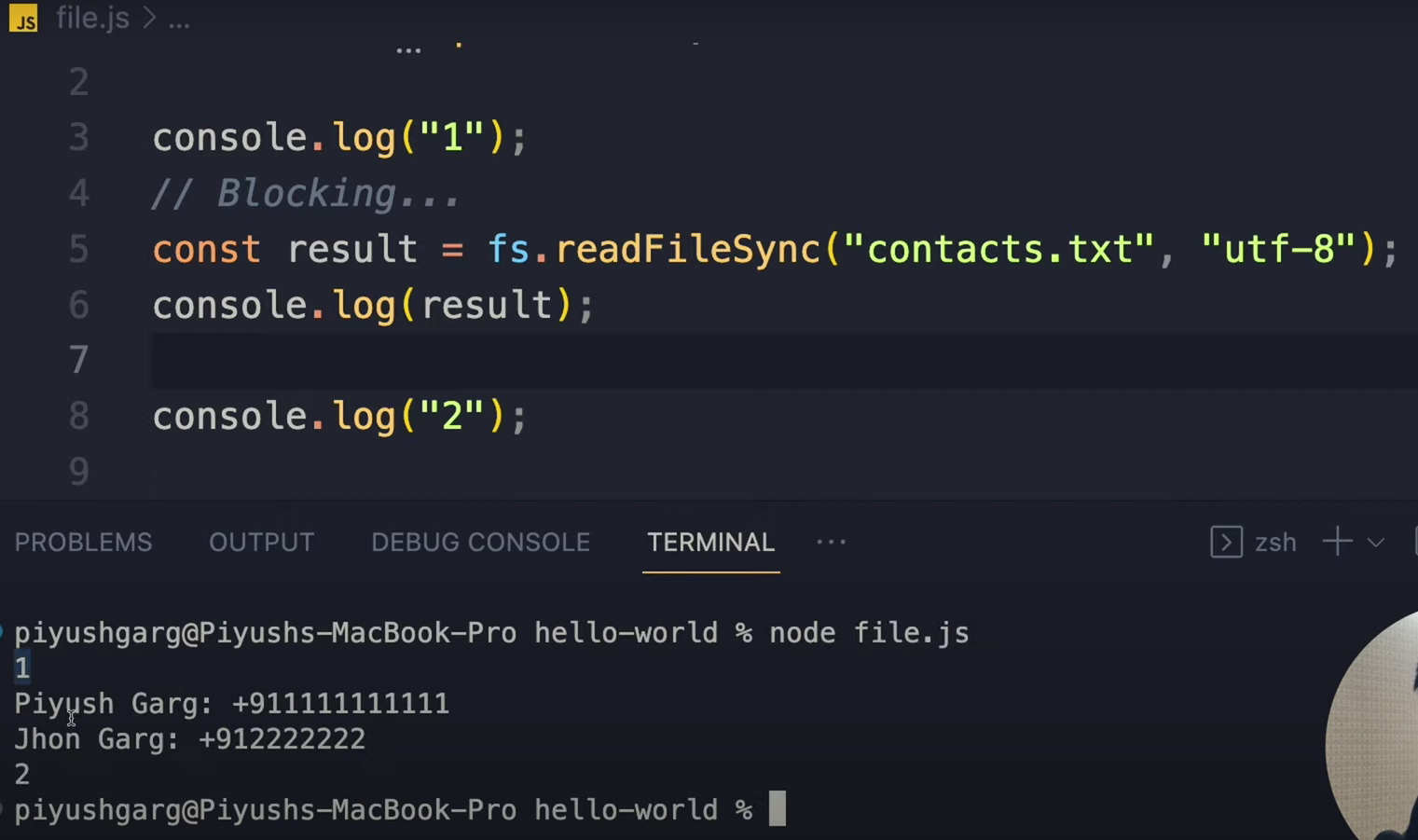


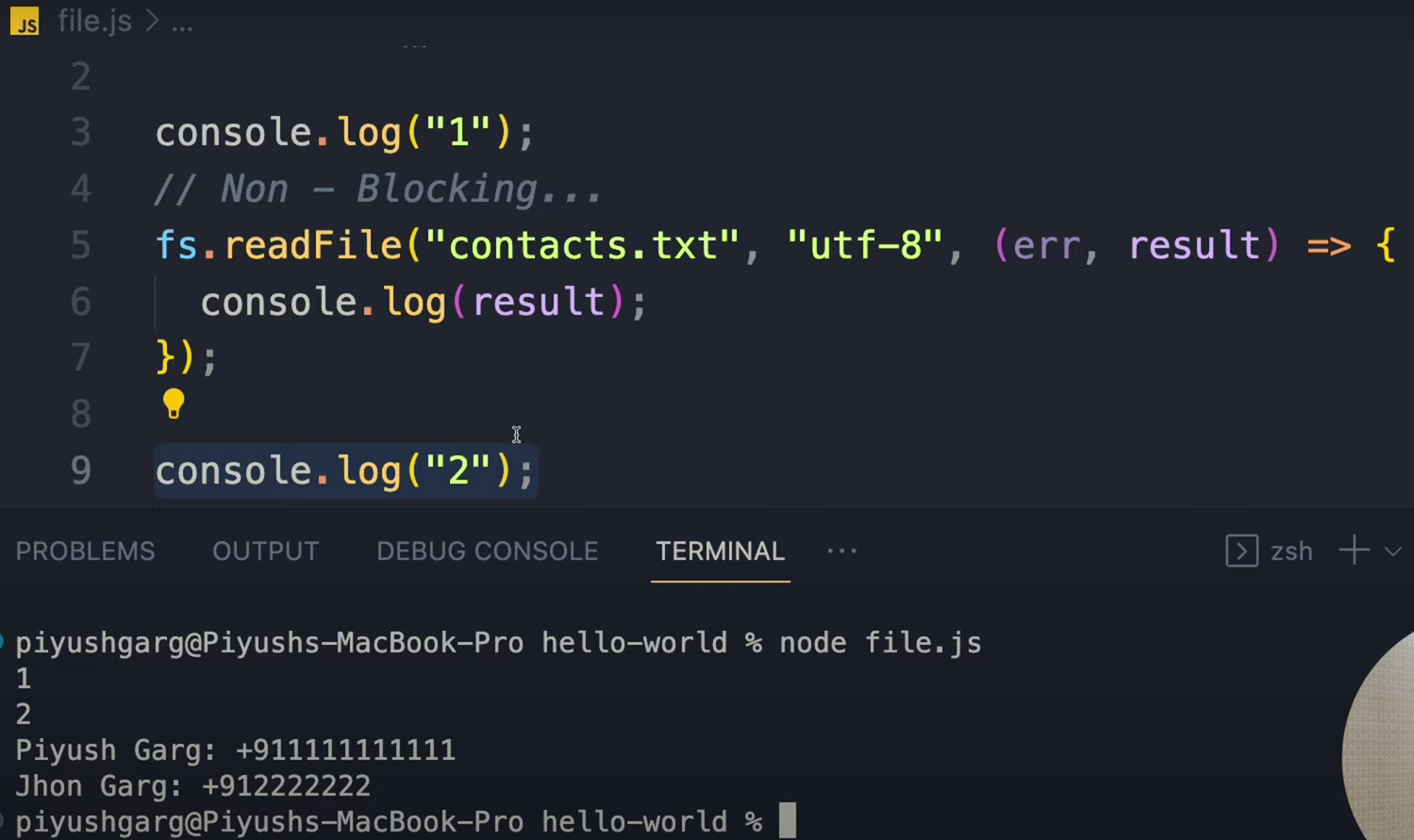
✦**NodeJS handles non-blocking and blocking operations differently**

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**

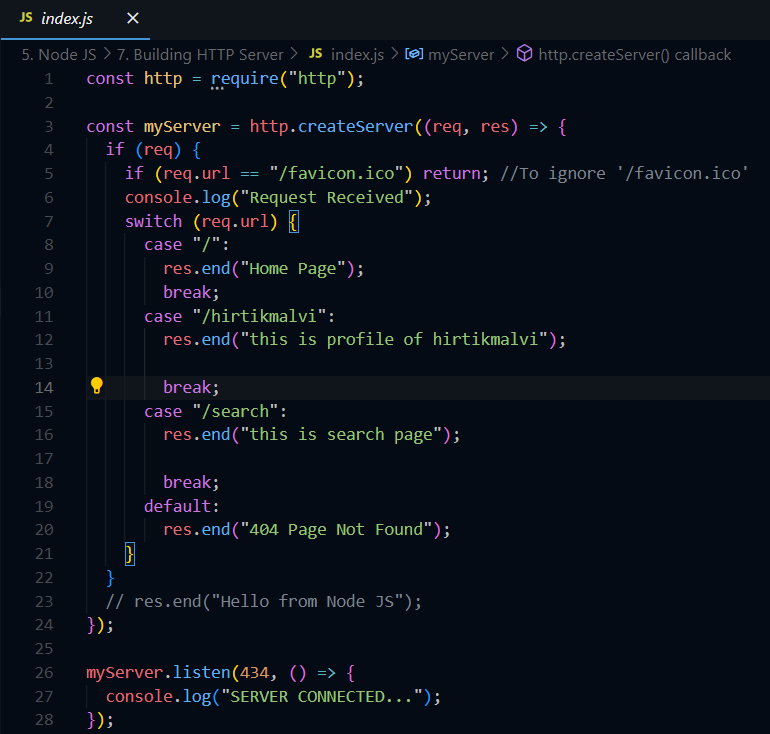
* Logging request time and user's IP address in a file
* Responding to requests and logging requested paths

**✦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.**

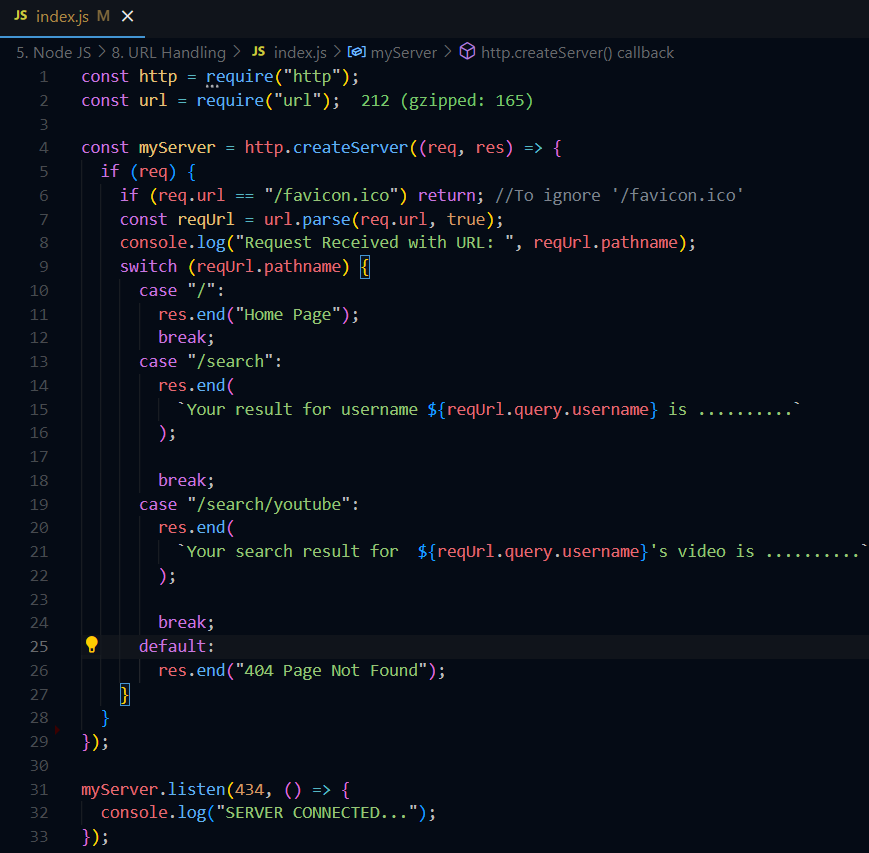
* Understanding query parameters and how to extract data from the URL.
* Exploring the process of handling user requests and generating appropriate responses.

**✦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.

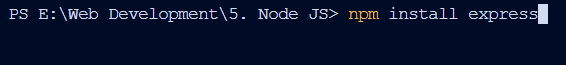
* Express framework solves the confusion in handling server code using the HTTP module.
* The problem is the confusing structure of callback functions and the need to make the code easier to manage.
* Example is shown: making server using without Express vs Using Express

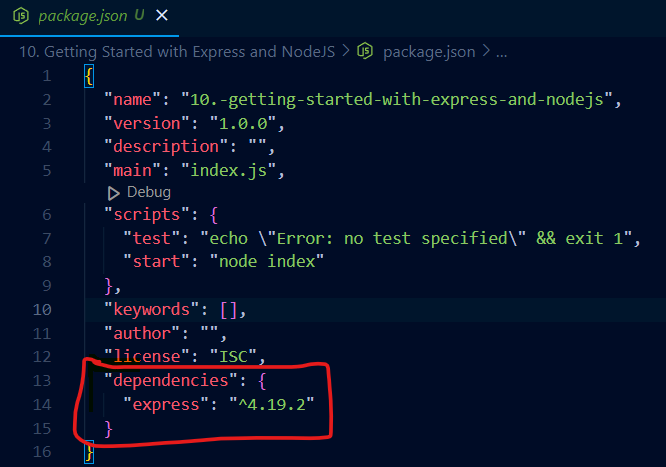
✦Express requires creating individual cases for different HTTP methods like get, post, put, patch, delete, which can be complex and time-consuming.

* 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

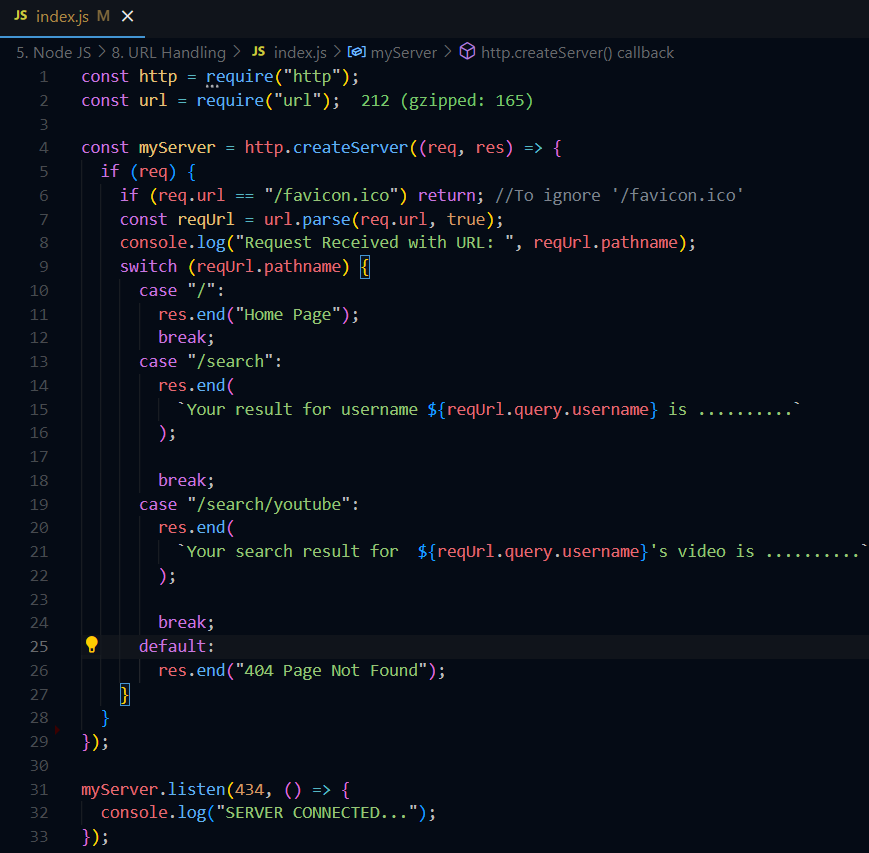
* Easy handling of routes and dynamic outputs
* Code becomes clean and organized, minimizing the need for additional steps

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

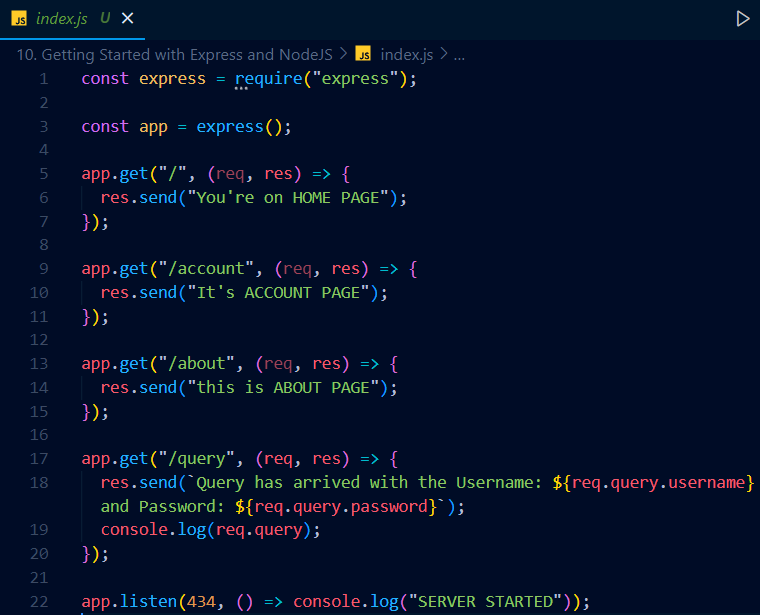
* Express eliminates the need for handling HTTP modules separately.
* Express simplifies the process of creating a server and handling listener installation.

✦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)**)**

**(…)**

✦REST API is a set of rules that defines how a client can interact with a server over HTTP.

REST API follows certain standards and rules which are important to understand and follow for best practices.

It allows different clients, such as browsers, mobile phones, and smart devices, to communicate with the server.

✦REST API allows communication between server and client via request and response.

The communication between server and client is based on request-response model with some best practices and standards.

The main rule is that the architecture should not be dependent and the response can be in various formats like text, image, HTML, or JSON.

✦REST API helps in fetching and rendering data from the server to different clients.

Fetching data from the database and creating an HTML document

Rendering data based on the client, and server-side rendering and its advantages

✦REST API is used to send and receive data in a raw format like JSON.

REST API allows data to be transferred between server and client using key-value pairs in JSON format.

Client applications like mobile apps can easily read and render the data received from the server independently.

✦REST API follows server and client architecture.

HTML should be sent if the client is a browser to reduce processing on the client.

Always respect all HTTP methods like GET, POST, PUT, PATCH, DELETE.

✦Understanding REST API for handling user data

Explaining the use of GET, POST, and PATCH methods for different user operations

Discussing best practices and adherence to RESTful architecture

✦Using HTTP methods to respect the server's rules

Server should respect methods like GET, POST, DELETE

Each method should be used for its specific purpose

✦Rendering JSON and HTML in Express GS

Rendering JSON using response.json() method

Rendering HTML using response.render() method

# **Lecture 13 : Building REST API's using Node and Express.js**

### Lecture 14 : Introduction to POSTMAN for REST API's

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

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/P1\_L13\_14%20Lec%20Building%20REST%20API%20using%20Node%20and%20Express**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/P1_L13_14%20Lec%20Building%20REST%20API%20using%20Node%20and%20Express)**)**

✦Creating REST API using Node and Express.js

Initial project setup with NPM and package.json creation.

Installation of Express and starting with boilerplate code.

✦Building basic REST API routes in Node with Express.js

Setting up GET routes for listing users and getting user details by ID

Defining POST, PATCH, and DELETE routes for creating, updating, and deleting users

✦Generating fake user data for testing purposes

Created fake user data with fields like ID, name, email, gender, job title, and user count.

Downloaded JSON data containing thousands of fake user increments for testing.

✦Building REST API's using Node and Express.js

Setting up request and response for users list

Creating a hybrid server to support browsers and mobile apps

✦Rendering HTML content for /users endpoint

Creating a list of users in HTML format using ul and li tags

Using a dynamic approach to map over users and display their first names in the list

✦Creating a REST API with dynamic path parameters in Node and Express.js

Explained the concept of dynamic path parameters and how to use them in Express.js for creating flexible endpoints.

Demonstrated the process of creating a REST API with dynamic path parameters and handling requests.

✦Creating routes for POST requests

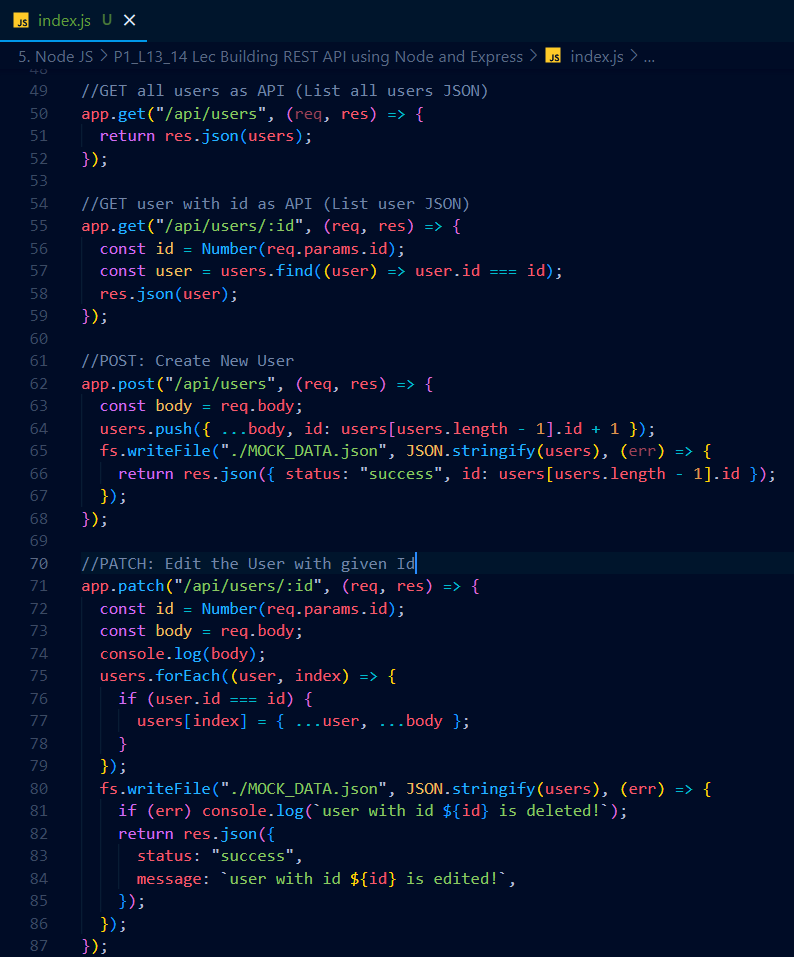
Converting string ID to number for matching

Returning specific user information based on ID

✦Setting up different request handlers for GET, PUT, and DELETE methods

Checking status to make it pending and leaving comments for self

Approach for handling future requests and using Postman tool for testing APIs in the next video



# **Lecture 15 : Express Middleware**

**(**[**https://youtu.be/n2c0mf1sza4?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/n2c0mf1sza4?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

✦Understanding the concept of express middleware.

Middleware is a crucial topic in express, important for request-response cycle and running custom functions.

Middleware intercepts the client's request before it reaches the server, allowing for processing and customization.

✦Middleware allows for various processing and decision making within the request cycle.

It can process the request, verify details, validate, or execute any required actions.

It can either approve the request and forward it, or reject the request and end the response cycle.

✦Express Middleware handles incoming requests and responses

The middleware checks for authorization, regions, and user credentials

It can end the response or call the next middleware function

✦Express Middleware functions can execute code, make changes to request and response, and call the next middleware function.

Middleware can manipulate the request and response cycle and call the next middleware function in the stack.

Creating custom middleware involves defining a handler function with request, response, and next parameters.

✦The middleware neither ends the request nor allows it to go ahead.

The middleware keeps the request within itself, causing it to get stuck in the middle.

The middleware does not call the next function, preventing the request from proceeding.

✦Understanding the flow of Express middleware

Middleware is called before the route handler and can perform tasks like logging, authentication, etc.

Middleware functions can control the flow of request-response cycle by calling the next function or ending the cycle.

✦Express middleware allows for modifying the request and response objects.

Middleware functions have access to request and response objects.

You can modify the request and response objects to make changes throughout the request lifecycle.

✦Express middleware processes request body and enhances request handling.

Middleware turns form data into an object and provides access to request data.

It allows for validation, database queries, and manipulation of request and response objects.

✦Understanding request methods and middleware

Exploring different request methods and their effects

Testing middleware functionality and troubleshooting

✦Express Middleware simplifies code structure.

Middleware allows sending requests to the next processing point.

Express website provides detailed guide on using middleware.

[**https://expressjs.com/en/guide/using-middleware.html**](https://expressjs.com/en/guide/using-middleware.html)



# **Lecture 16 : What are HTTP Headers in API**

**(**[**https://youtu.be/mhg3Vwsb88M?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/mhg3Vwsb88M?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

✦HTTP headers play a crucial role in the communication process of sending and receiving data over the web.

* HTTP headers are vital in providing necessary information for successfully transmitting and receiving data packets over the internet.
* They contain essential details like addresses, packet weight, and other relevant information necessary for the delivery and processing of data.

✦HTTP headers contain additional information outside the packet.

* Headers contain information about the request and response, such as where the request came from and its destination.
* HTTP headers are essential for handling metadata associated with app requests and responses.

✦HTTP headers carry information for request and response body

* The headers contain extra information for the request and response like date, contact, about, etc.
* The video demonstrates a real-world example of inspecting and understanding the headers of a request sent to youtube.com

✦HTTP headers contain various information related to requests and responses.

* HTTP headers include details such as language, cash control, cookies, referral, user agent, and device information.
* Response headers may include time passed, permissions, and server responses relevant to the website.

✦Understanding HTTP response headers and creating custom headers in API

* Explaining the 'powered by Express' header and its significance
* Detailed explanation of content type, content length, date, connection, and custom headers

✦HTTP headers are used to control requests and hold pertinent information.

* HTTP headers can include purpose, user agent, postman accept, postman token, host, accept encoding, and connection.
* Custom headers can also be read and managed, and best practices include appending 'X-' to custom headers.

✦HTTP headers contain important information for requests and responses.

* HTTP headers are pre-defined and include data such as size and custom information.
* Custom headers can be created to contain specific information, such as user authentication.

✦Middleware checks content type and makes data available in request body

* The middleware checks the content type of data in each request
* It makes the data available in the request body after processing

Header: <https://dev.to/nonunicorn/getting-and-setting-headers-in-nodejs-http-server-3nl7>

To make custom Headers: <https://nodemailer.com/message/custom-headers/>

# **Lecture 17 : HTTP Status Codes**

**(**[**https://youtu.be/fLGw2GK884s?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/fLGw2GK884s?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
* <https://www.restapitutorial.com/httpstatuscodes>

# **Lecture 18 : Getting Started with MongoDB**

# **Lecture 19 : Connecting NodeJS with MongoDB | Mongoose + Express**

**(**[**https://youtu.be/xrglM8U0Zv8?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/xrglM8U0Zv8?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

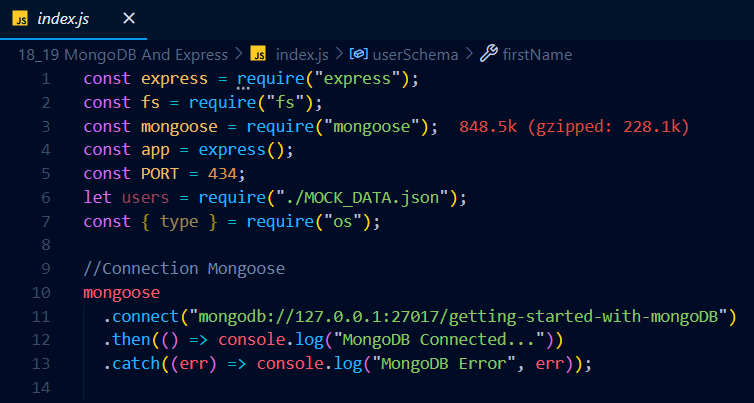
**(…)**



Mongoose is used to connect Express and MongoDB. So we’ve to install mongoose as **npm install mongoose**

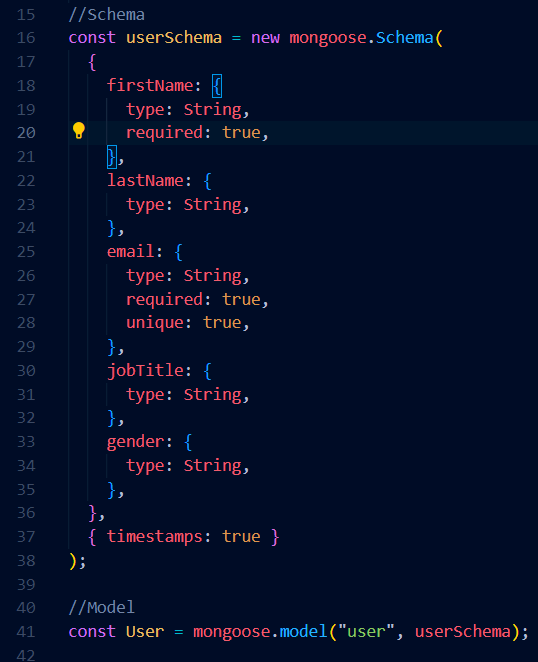
To Use MongoDB remember the steps:

* Make Connection:



Here, ”/getting-started-with-mongoDB” is a Database name.

* Then we’ve to make a model. For that model first we have to make a schema(Design/Pattern/Structure) of the data. And using that Schema We can make a model.
* This model helps us to perform CRUD – Create, Read, Update, Delete Operations.



* Syntax of model: const modelName = mongoose.model(collectionName,schema);
* collectionName you have to define in the model syntax only.

**For CRUD Operations:**

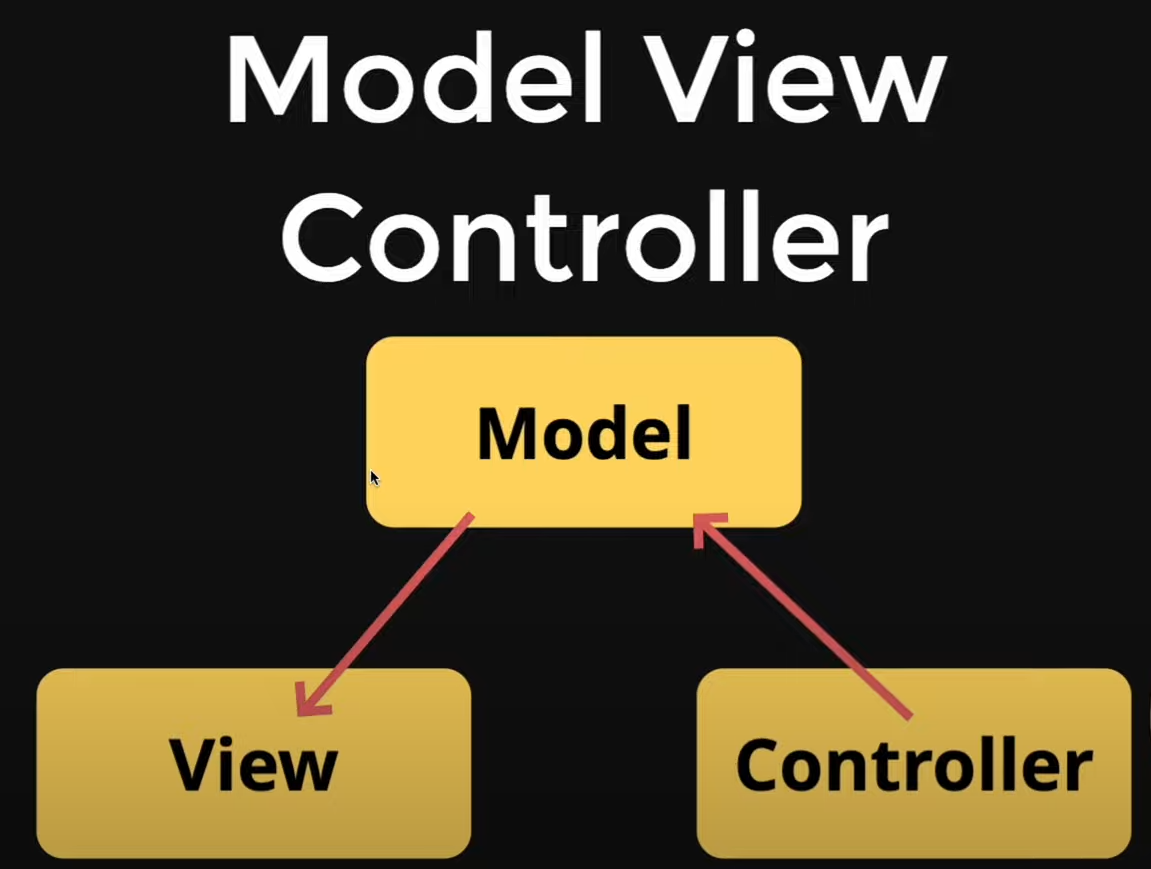
[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/18\_19%20MongoDB%20And%20Express/index.js**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/18_19%20MongoDB%20And%20Express/index.js)

# **Lecture 20 : Model View Controller in NodeJS | MVC Pattern**

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

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

* If we put all the code together like connection of MongoDB, Its schema and model declaration, Creating Server, CRUD Operations, middleware etc etc becomes HEADACHE figure out many things when to solve some error.
* That’s why MVC – Model View Controller is used.



* Controller MANIPULATES the Model and Model updates the View.

✦Introduction to Model View Controller in NodeJS

* Model View Controller is made up of three components: Model, Views, Controllers.
* The need for refactoring due to code maintainability issues and pollution.

✦Creating a model and routes in NodeJS using MVC pattern

* The process of creating a model with module.exports and removing unnecessary code
* Creating and registering routes with Express and isolating the router

✦Implementing routing for user-related functionality

* Explaining how requests for different user-related actions are handled by the router
* Setting up a separate connection file for database configuration and implementation

✦Creating a mongoose connection to MongoDB.

* Setting up and exporting the function to connect to MongoDB using Mongoose.
* Explaining the role and purpose of the connectMongodb function.

✦Setting up controllers and routes in NodeJS

* Creating controllers to handle different routes and functions for processing requests
* Defining handlers or controllers to manipulate data and respond to requests

✦Refactoring and exporting functions in NodeJS Controller

* Refactoring functions by exporting and importing
* Handling different types of requests like get all users, get user by ID, and update user by ID

✦Demonstration of model view controller actions in NodeJS

* The video demonstrates the process of handling delete requests and creating new users
* It also covers importing, exporting, and server connection testing

✦Setting up MangoDB connection and middleware in NodeJS

* Creating a middleware to handle connection requests and response
* Registering routes to handle different types of requests and directing them to controllers

✦Creating and working with models and controllers in NodeJS

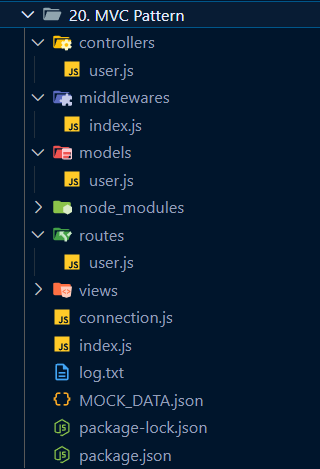
* Explaining the creation of schemes and models for the controller
* Debugging and testing the functionality using Postman and Mango TV

✦Understanding the organization and structure of code in NodeJS with MVC pattern

* Segregating folders and files in production environment for better code management
* Easy collaboration and understanding of code structure in a team setting with clear roles and dependencies

✦Changing the route of Slash API for user outer

* The benefit of changing the route of user outer is discussed.
* Upcoming videos will focus on making projects and subscription requests.



# **Lecture 21 : Create a Custom URL Shortener using Node.JS and MongoDB**

# **Lecture 22 : Server Side Rendering with EJS and Node.JS**

**(**[**https://youtu.be/4WvX9dBjiJo?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/4WvX9dBjiJo?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://youtu.be/yy9cbu\_e3Xg?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/yy9cbu_e3Xg?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/21\_22.%20Create%20a%20Custom%20URL%20Shortener**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/21_22.%20Create%20a%20Custom%20URL%20Shortener)**)**

* In lecture 21, we have made a thing Like Bitly. Which takes URL of any website and make it short and using which we can use the same website.
* In lecture 22, we did the same but this time made a simple and basic Web page using Server Side Rendering through EJS.

What it does it, it takes URL and stores it in the Database along with the shortId generated for the URL entered. Whenever user use the new URL which is with shortId, it finds the actual URL of website in the database using the shortId. Whenever it is found it is redirected to the site.

# **Lecture 23 : Building Node.js Authentication from Scratch**

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

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/23.%20Building%20Node.js%20Authentication%20from%20Scratch**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/23.%20Building%20Node.js%20Authentication%20from%20Scratch)**)**

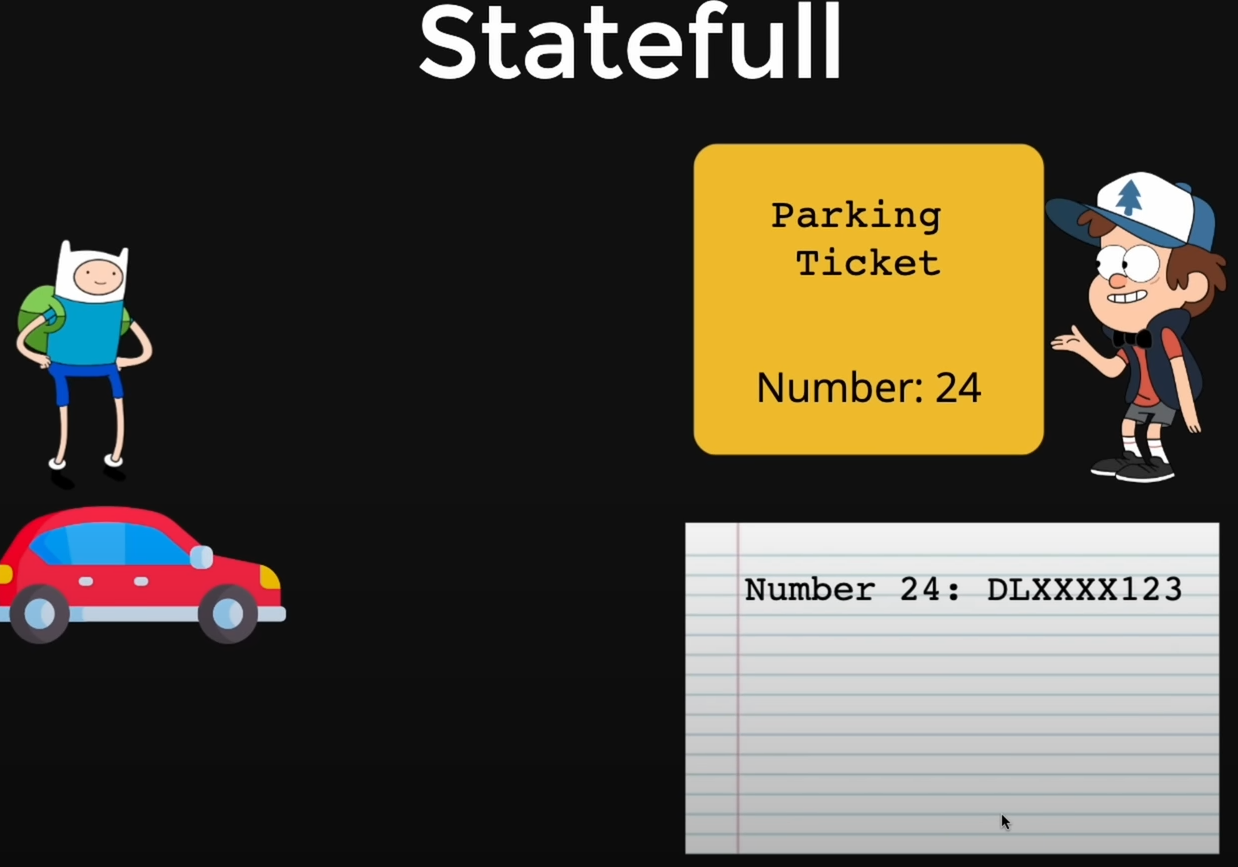
* In this lecture, we have seen/build a basic and simple Authentication for the URL Shortener project. In which If user is Logged In then and then only he can access the page.

✦Authentication is crucial for service and backend developers.

* Authentication involves **stateful** and **stateless** patterns.
* Stateful authentication maintains data on the server side while stateless authentication does not.

✦The video discusses the concept of maintaining state and stateful architecture in the context of a parking boy's diary and how it relates to authentication in Node.js.

* The parking boy maintains a diary to record car assignments and returns the correct car based on the ticket number, demonstrating state maintenance.
* The concept of stateful architecture is then related to authentication in Node.js, where the client provides username and password to the server, similar to the parking boy giving his car to the attendant.



**Here we have discussed STATEFULL Authentication.**

✦Adding authentication to Node.js application with cookies

* Middleware in the express server checks for cookie value and authentication
* Creating a user model for the database with configuration for name and email

✦Creating user model and authentication routes in Node.js

* Creation of user model with email and password
* Set up authentication routes for user sign up

✦Creating a sign-up form with name, email, password, and submit button.

* The form action is set to '/user'
* The form method is set to 'post'

✦Limit short URL access to logged-in users

* Restrict short URL service to authenticated users.
* Implement user authentication using email and password.

✦Setting up and using unique session IDs for user authentication

* Installing the NBM ID package and importing it for use
* Creating a session ID, storing it with the user object, and using a utility or service to manage the session IDs

✦Implementing authentication with session ID and cookies.

* Setting up session ID and cookies in the server response.
* Utilizing the cookie value to authenticate users in the middleware.

✦Retiring the cookie parcel and implementing middleware for specific URL

* Utilizing app.use() to retire the cookie parcel and make cookie-less requests
* Implementing middleware to validate requests for specific URLs and debugging cookie-related issues

✦Building authentication and encountering server restart issues.

* Discussing the process of generating and rejecting requests.
* Exploring server restart problems and potential solutions.

✦Creating and testing URLs, finding users and current user authentication.

* Creating and testing URLs using Google.com
* Finding all URLs, finding all users, and implementing current user authentication.

✦Implementing user authentication without forcing login

* Checking for user authentication without forcing the user to log in
* Handling user authentication and redirection based on the user's authentication status

# **Lecture 24 : JWT Authentication in NodeJS**

**(**[**https://youtu.be/mGrVmEex6\_g?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/mGrVmEex6_g?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(**[**https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/24.%20JWT%20Authentication%20in%20NodeJS**](https://vscode.dev/github/hirtikmalvi/Web-Development/blob/main/5.%20Node%20JS/24.%20JWT%20Authentication%20in%20NodeJS)**)**



✦Token based authentication helps in solving stateful authentication issues.

* Stateful authentication is memory intensive, requires server state maintenance.
* Stateless authentication stores state in tokens, eliminating server memory usage.

✦Implementing JWT authentication for secure data storage and retrieval

* JWT tokens are used to store user data securely and ensure its integrity
* The tokens are created by including the user's ID and other data, allowing for secure and tamper-proof storage

✦Using JSON Web Tokens for user authentication

* Token is created and stored in user's cookies for authentication
* User data is stored inside the token for verification

✦Creating and managing secret keys for JWT authentication

* Secret key creation process to generate tokens securely.
* Usage of tokens for user authentication and session management.

✦Token authentication is crucial for security

* Without a valid token, the system does not proceed further
* Inspecting cookies and tokens on the webpage for validation

✦Understanding the role of secret key in token authentication

* The secret key is essential for generating and verifying tokens
* Changing or refreshing the token requires the secret key

✦JWT tokens are used to securely store user details and should not be shared publicly.

* JWT tokens store user details and act like a digital ID card, eliminating the need to repeatedly verify the user's status.
* If a token is shared or leaked, it can be misused, so it should be kept private and secure.

✦Tokens should never be leaked

* Session ID changes with refresh/log in/out
* Different types of tokens for different applications

# **Lecture 25 : What are Cookies in NodeJS?**

**(**[**https://youtu.be/kMErso06vHo?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo**](https://youtu.be/kMErso06vHo?list=PLinedj3B30sDby4Al-i13hQJGQoRQDfPo)**)**

**(…)**

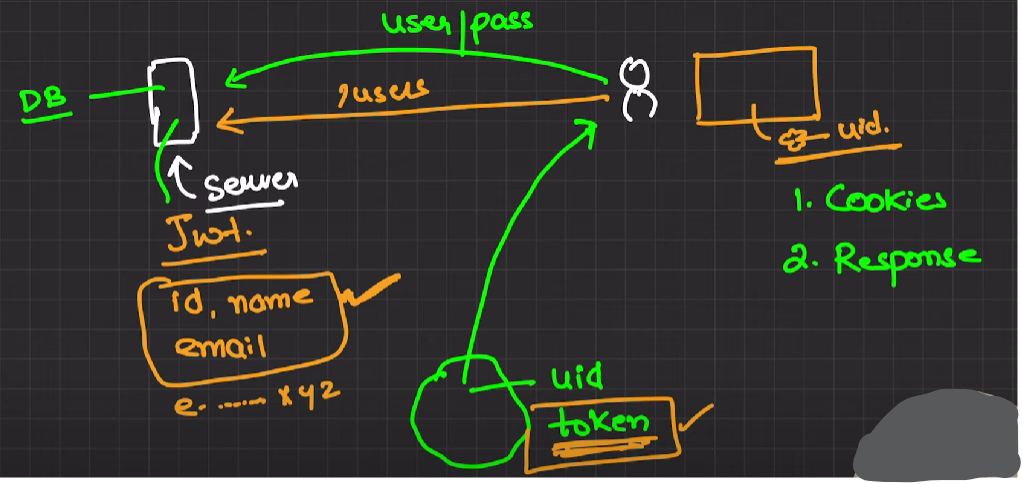
✦Stateful authentication architecture and its drawbacks

* Stateful authentication involves creating sessions and storing session IDs and corresponding users in the database
* The drawback of this architecture is the possibility of increased latency and dependency on database queries for user authentication

✦Using database for authentication can lead to high read operations and increased bills.

* Querying the database for authentication increases read operations and bill in real world scenarios.
* Limiting database usage for authentication and using stateful methods like JWT can be a better solution.

✦Browsers store and send cookies with requests for user authentication



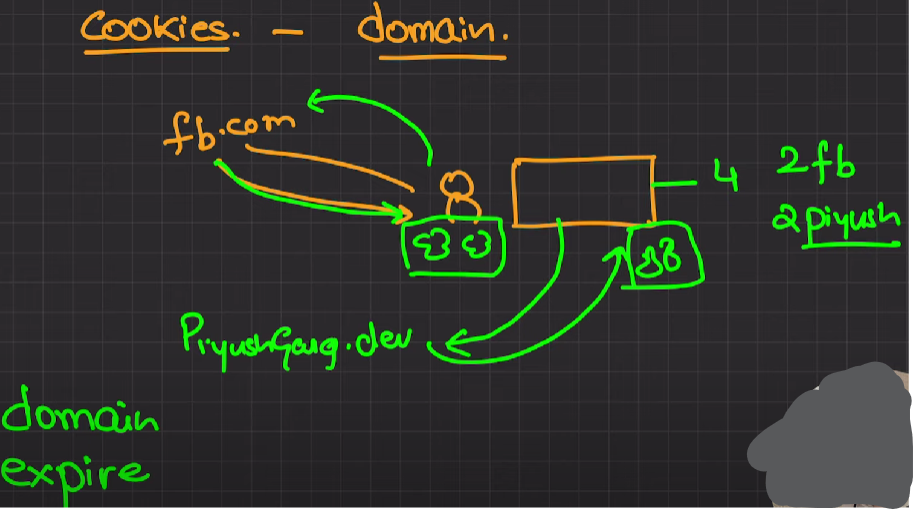
* When a user sends a request to the server, the browser sends the stored cookies along with the request
* Server can validate the user's authenticity and extract user information from the cookies

✦Understanding and managing cookies in NodeJS.

* Cookies hold user authentication and other data.
* Inspecting cookies through browser tools and extensions.

✦Cookies are specific to the domain they were created for.

* Any request to the local host will only send cookies that are associated with it.
* Cookies are specific to their own domain and can be manipulated using the 'domain' option in your code.



✦Cookies in NodeJS can be domain-specific

* NodeJS allows setting domain for cookies, restricting access outside specified domain
* Default behavior creates cookies accessible only by its own domain

✦Creating and handling tokens in NodeJS

* Server creating the token and sending it as response
* Storing and handling the token using cookies or headers for authentication

✦Setting the authorization header with the token

* The standard way is to include the token in the authorization header with the 'Bearer' scheme.
* The server will validate the token from the authorization header.

✦Making POST request with body data

* Sending email and password in the body of the request
* Handling response and authorization token to access protected resources

✦NodeJS cookies involve user ID, authorization, and token for authentication and store maintenance.

* Cookies can be used for automatic authentication for every request but have limitations for browser-based applications.
* Headers can be used to send tokens in JSON format, with the user maintaining the store and using the 'Bearer' authorization scheme for token-based authentication.