
CSS
HTML
BACK END
JS

History and Introduction of Node.js

- Before 2008, JavaScript only ran in browsers, but then came a visionary named Ryan Dahl.
- Ryan Dahl took the V8 engine (originally used in Chrome browsers) and placed it in an independent environment, creating what we now know as Node.js.
- Node.js is not a library and not a framework—it is simply a runtime environment for running JavaScript outside the browser.
- **Why the name "V8 engine"?**
  - The V8 engine was named after the powerful engine launched by Ford in its new car model.
  - This engine, written in C++, was the fastest at that time.

Purpose of Nodejs.

- with help of nodejs we can create → drone software, video games, native applications.
- command line tools → we can do through nodejs.
- electron.js →is used to create desktop applications example → discord.

Initializing Node.js in the Terminal

- To initialize Node.js in the terminal, simply type node and press Enter. This will start the Node.js runtime environment.
- When Node.js is running in a shell or terminal, it operates in REPL mode, which stands for Read, Evaluate, Print, Loop.
- Typing global in the REPL will display the global object.

How to Run a Node.js File

- To run a JavaScript file in Node.js, you must first navigate to the folder where the file is located using the terminal.
- Once in the correct folder, you can run the file by typing node app.js (replace app.js with the name of your JavaScript file).

What is Process in Node.js?

- A process is an object in Node.js.
- It stores all the ongoing information regarding Node.js processes or tasks.
- The process keeps track of everything happening in the current Node.js runtime environment.
- While it's not a window object like in browsers, it similarly stores important information.

argv Inside the Process

- argv is also an object inside Process that returns an array.
- This array contains two main pieces of information:
  - **Path to Node.js**: The location where Node.js is installed on your system.
  - **Path to the current script**: The file's location (or process) currently being executed.
- Example output:
- The first path points to where Node.js is installed, and the second path points to where the ongoing Node.js process is running.
- We use **process.argv** in Node.js to **accept input from the command line** when running a script. Users can pass arguments dynamically.

Difference between cwd() and __dirname:

1. **cwd() (Current Working Directory):**
   - This is a function that gives you the path of the folder from where your project or file is being run.
   - It shows the path of the directory you are currently in (where you "stand").
   - However, it does not provide the path to the location where the code (process) is actually running.

1. **__dirname (Directory Name):**
   - This gives the path where the current code or process is running.
   - It shows the path up to the exact file or program (like app.js) that is being executed.

Path Module in Node.js

- The Path module is a core module in Node.js.
- To use anything from this module, we need to require it first.
- One important method in the Path module is path.join().
- path.join() accepts multiple strings as arguments and returns a single string.
- It automatically joins all the provided strings into a valid path by adding the appropriate slashes (/ or \, depending on the operating system).
- If there are any extra slashes in the provided strings, path.join() will remove them, ensuring a clean and consistent path.

What is a File System?

A simple way to store data (alternative to a database).

- It helps manage files and directories on a computer.
- Less efficient than databases but still useful in some cases.

In this lesson, we'll learn how to perform CRUD (Create, Read, Update, Delete) operations using the File System module in Node.js.

How Does Node.js Handle File Systems?

📌 **When we install Node.js, what happens?**

- Node.js installs with npm (Node Package Manager).
- It also includes many built-in modules.
- One of them is the File System (fs) module, which helps in handling files.

📌 **What are modules?**

- Pre-built libraries in Node.js that help perform various tasks.

NPM (Node Package Manager)

NPM is used to manage Node.js packages, but it goes by different names based on its functionality.

Story:

There were some **big developers** who worked hard and wrote helpful code day and night. These developers decided to store their code in one place so that anyone who needed it could easily access and use it.

- Just as an example, they stored their code on the cloud.
- The reason for storing it on the cloud is that whenever someone needs it, they can easily install it and use it.
- What is this cloud? → This cloud is known as CPM (Central Package Manager).
- In the case of Node.js, our CPM is called NPM (Node Package Manager).
- The package could be a framework or a library.

- NPM helps manage different packages within Node.js.
- When you install Node.js, npm automatically installs along with it.
- Packages are pieces of code written by third-party developers.
- NPM is stored in the cloud, allowing anyone to install and use it.
- The cloud where the packages are stored is called CPM (Central Package Manager).
- A package can be either a library or a framework.
- In the case of Node.js, our CPM is NPM.
- For Python, the CPM equivalents are pip and conda.

How to Install a Package Using NPM

1.Create a Folder/Project:

- First, you need to create a folder for your project. For example, name the folder give-me-a-joke.
- Navigate to this folder where you will initialize the project and manage the Node.js packages.

2. Initialize the Project:

- To install npm packages, you need to initialize your project to let npm know it can manage the packages. This initialization is like paying an "entry fee."
- Run the following command in the terminal where your project folder is located
- This command initializes your project to allow npm to manage its dependencies.

3. NPM Init Setup:

- After running npm init, npm will prompt you to fill in some information about your project.
- Once all this is filled in, npm will create a **package.json** file for you.

4. Install the Package:

- Now you can install the package from npm's cloud (CPM). For example **npm install give-me-a-joke**
- This command downloads the package and installs it in your project. Don’t worry if you see any npm warnings; the package is successfully installed.

5. Make the Main File:

- After running npm init, npm will prompt you to fill in some information about your project.
- Once all this is filled in, npm will create a **package.json** file for you.
- Create an index.js file (the entry point you specified during npm init).

When you run npm init ➜ it creates a package.json file**.**

1. **package.json**: This file contains metadata about your project, including the dependencies it relies on. It acts as an overview of your project and helps npm manage the packages.

When you run npm install (package name) ➜ two things are created:

1. **node_modules:** This folder stores the source code of the installed packages.
2. **package-lock.json:** This file contains detailed information about the installed packages, including their internal dependencies.

What is Express?

- Is it a tool, library, runtime, or framework?
- Express is a framework.

What Does Express Do?

- Express is used to create web servers.
- It's an advanced version of the HTTP server (in fact, it's built on the HTTP server).
- Express is a third-party package that we need to install using npm (Node Package Manager).
- We create our own server with Express.
-

Require Express :

- Express returns a function, and when we run it, it gives back an **entirely new object** (the app), which represents our web server.
- The **app** object is an **instance of the entire application**. Think of it as the core of your web server. Everything in your web server runs through this **app** object.

```
const express = require('express'); // Import Express
const app = express(); // Create an Express app instance


console.log(app); // Outputs a huge object (our server instance)
```

Microprocessor

- Microprocessors are mini computers.
- it will run your low level code in bites formate.(0,1).
- microprocessors have small small pins(port) →Eg. 8085(40 pins), 8086.
- these single pins act as a Port
- port means esi jagha jaha aap apne operations perform kr paa rahe ho.Eg. phone charging port.
- if we want to perform any operations/ application we have to decide and give a port where your web app will run
- esi jagha de sake jaha vo apne instance of the application run kr sake .
- There are a few ports in our system that are decided for specific tasks performed in our system, so we can't change them. (pehle se occupied hote hai).
- Generally, 8000, 8080, 5000, 5050, and 3000 refer to the port.
- But the default port for the backend is 8080.
- Listen is a function that accepts a callback function.
- It builds a connection with server 8080 and waits for the request or incoming request to perform a task.

Middleware

Middleware functions act like an agent that intercepts and responds to every incoming request. app.use() is used to define middleware that is executed on every request or a specific path.

app.use([path,] callback)

- **What it does:** It's a method to define middleware functions in Express.
- Parameters:
- **path:** Optional. It's the specific route the middleware should run (e.g., /middleware). If omitted, it applies to all routes.
- **callback:** A function that is mandatory and runs every time a request hits the server.

```
app.use((req, res) => {
    console.log('you made a request');
    res.send('<h1>Response</h1>');
});
```

Understanding Request and Response Objects (req and res)

- Both req (request) and res (response) are objects.
- Whenever a client sends a request, the server stores that request as a req object.
- Similarly, when the server sends a response, it stores it as a res object.
- You can log these objects to inspect the incoming request and outgoing response.

Sending a Response with res.send()

- The res.send() method sends a response back to the client.

```
app.use((req, res) => {
    console.log('you made a request');
    res.send('<h1>This is the response</h1>');
});
```

Specifying Middleware for a Specific Path

- You can make middleware execute only when a specific route is matched by passing a path to app.use().
- If the path doesn’t match, this middleware will not be executed.

```
app.use('/middleware', (req, res) => {
    console.log('you made a request at /middleware');
    res.send('<h1>Path matched: /middleware</h1>');
});
```

Routing in Express

- Different routes return different responses based on the request made.
- Each incoming request generates a distinct response, determined by the route and the type of HTTP method used.

Types of HTTP Requests (Methods)

- To learn more, you can visit MDN's page on HTTP methods: MDN HTTP Methods
- GET → This method is used to retrieve or fetch data from the server.
- POST → Sends data to the server to create/update resources.
- PATCH → Applies partial modifications to a resource.
- DELETE → Deletes a specified resource.

app.get(path, callback [, callback ...])

- path and callback function are mandatory parameters.
- This function is used to define a route in the application using the GET method.
- Even if you don't explicitly add a slash (/) at the end of the path, the slash is assumed by default.
- The slash (/) is the default root path for the application.
- GET is also the default method to retrieve data for this route.

```
app.get('/', (req, res) => {
    res.send('<h3>This is the root route</h3>');
});

app.get('/cat', (req, res) => {
    res.send('<h3>This is the /cat route</h3>');
});
```

Using nodemon to Automatically Restart Server

- **nodemon** is a tool that automatically restarts the server whenever you make changes in your code. This is useful as it saves you from manually restarting the server after every change.
- **npm install nodemon**

Path and Query Parameter

Path Parameters:

- Path parameters are part of the URL structure.
- Example: When you visit reddit.com/r/anything, the **/r/anything** portion is a **path parameter.**
  - /r/ is the constant part **(the subreddit path)**.
  - anything (e.g., dogs, cricket, etc.) is the variable part.
- When you remove the /r/ from the URL, you get a 404 error (page not found) because /r/ is a required path that leads to different subreddits.

Why /r/ is important?

- /r/ is a constant that identifies a subreddit.
- Anything after /r/ is treated as a subreddit name, and the content changes accordingly.

```
let express = require('express');
let app = express();

app.get('/', (req, res) => {
  res.send('root route hai');
});

// Instead of creating infinite routes, use a path parameter
app.get('/r/:subreddit', (req, res) => {
  console.log(req.params); // Object containing the subreddit part
  let { subreddit } = req.params; // Object destructuring
  res.send(`<h2> my route was ${subreddit} </h2>`);
});

app.listen(8080, () => {
  console.log('server running at port 8080');
});
```

- Here, :subreddit is a path parameter. It captures whatever comes after /r/ and stores it in req.params.
- You can access the subreddit using req.params.subreddit.

Query Parameters:

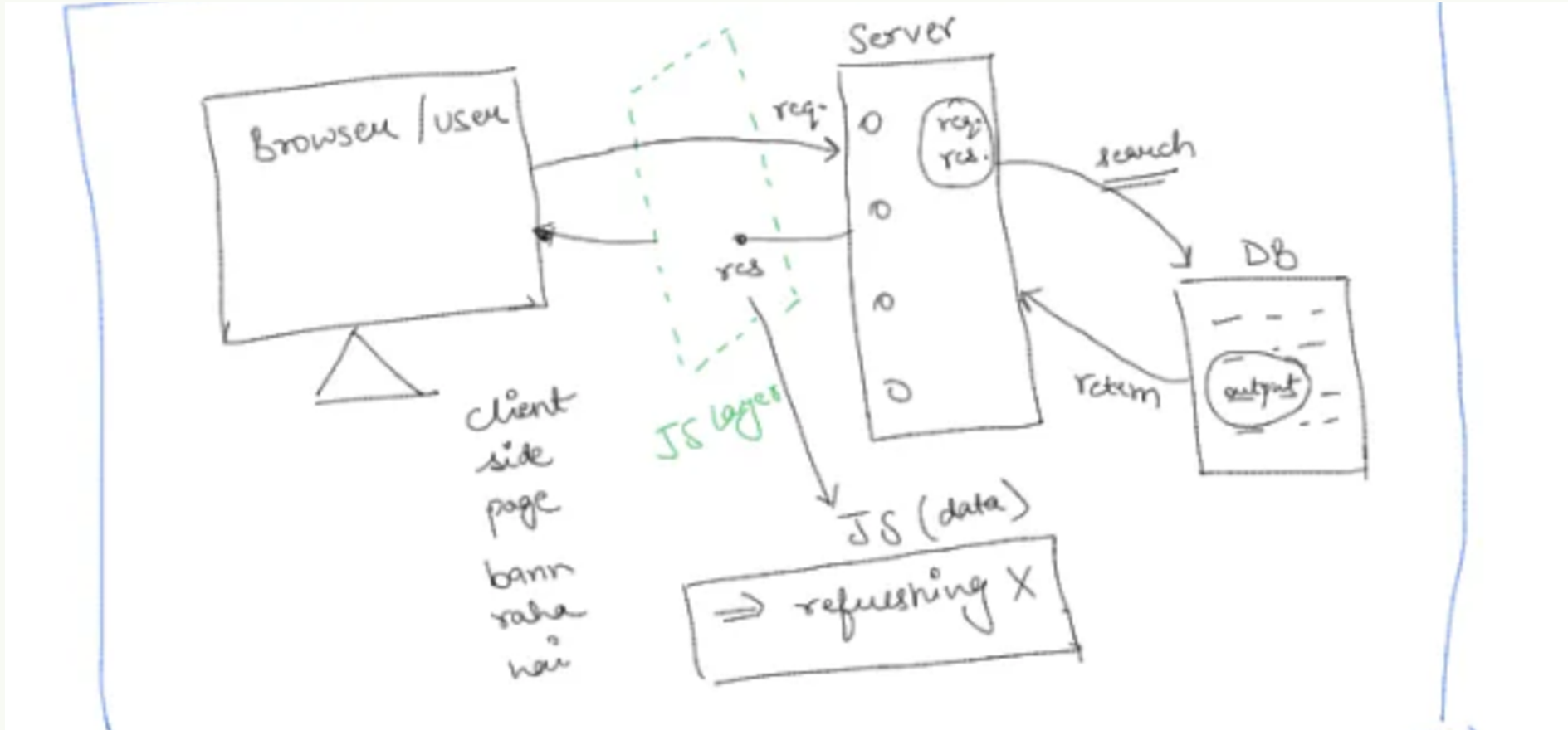
- Query parameters appear after a **?** in the URL and are **key-value pairs**.
- Example: When you search something on Wikipedia or Google, the URL may look like this: **localhost:8080/search?search=movie.**
    - The part after ? (i.e., **search=movie**) is the query string

```
const express = require('express');
const app = express();

app.get('/search', (req, res) => {
  console.log(req.query); // Logs the query object
  let { search } = req.query; // Destructure to get the value of the query
  res.send(`You searched for: ${search}`);
});

app.listen(8080, () => {
  console.log('Server running on port 8080');
});
```

Templating (ejs)

- layout same rehta hai different different routes pr
- eak hi route banega and eak hi page banega
- this concept is called templating.
- eak baar hi dhacha/ structure banao or usye use krte raho.

Mental Model



res.render(view [, locals] [, callback])

- Express uses res.render() to display views.
- view: the template to be rendered.

what is view?

- view is a template/ strucuture .
- and template ko krna hota hai show
- and show krne ke liye mujhe eak engine ki zarurat padegi (browser engines)
- we have different templating engines to show template.

Templating Engines

- Engines like **EJS, Handlebars, Jade/Pug.**
- all these are my templating engines and this will help me to show the template on browser.
- But we will use ejs templating engine.

EJS Tags

1. <% %>: Scriptlet tag, used for control flow, but doesn’t output anything.
    - E.g., for loops, if conditions.
2. <%= %>: Outputs and escapes HTML to prevent XSS attacks.
    - E.g., displaying variables dynamically.

File Structure

- All EJS templates are stored in a views folder.
- Default View Path: process.cwd + '/views'. It's a good practice to maintain this folder for templates.

```
// Set view engine to EJS
app.set('view engine', 'ejs');
```

Mongoose

- Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js.
  - **ODM** → Object Data Mapper converts JavaScript objects into MongoDB documents and vice versa.
  - This mapping layer helps manage and manipulate MongoDB data using JavaScript objects.

Key Features of Mongoose:

- **Third-party library**: Mongoose is not built-in; you must install it using npm.
- **Connect returns a promise**: When we connect to the database using Mongoose, it returns a promise that we can handle with .then() or async-await.

Mongoose Model:

- A Model in Mongoose is like a JavaScript class in VSCode.
- A model represents a collection in MongoDB.
- By convention, model names are written with an uppercase letter.
- The name of the model should be singular -> Product.

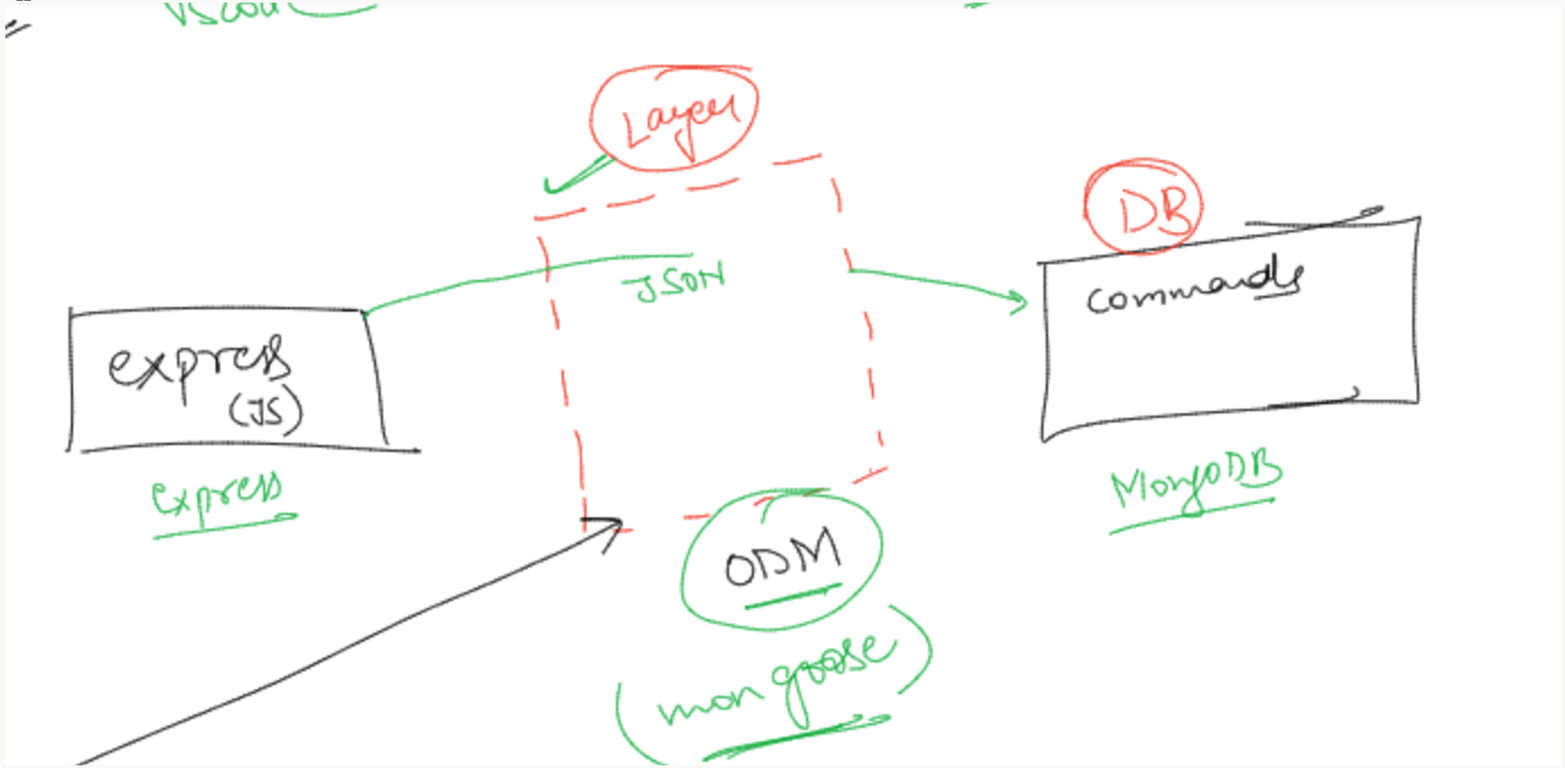
Schema:

- Every model is built using a Schema.
- A Schema is a blueprint of the data that defines the structure of documents within a collection.
- Schema in Mongoose always accepts an object with the key-value pairs defining the data types.

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

// schema always accepts an object
let blogSchema = new mongoose.Schema({
    title: {
        type:String,
        trim:true,
        required:true
    },
    author: {
        type:String,
        trim:true,
        required:true
    },
    comment: {
        type:String,
        trim:true
    }
});

const Blog = mongoose.model('Blog', blogSchema);
module.exports = Blog;
```

Mongoose Static Methods

- Static methods are methods that apply directly to the model (collection) and not to individual instances of documents. These methods usually return promises:
- **Product.insertMany():** Inserts multiple documents at once into the collection.
- **Product.create():** Creates a new document in the collection.
- **Product.find():** Fetches documents from the collection.



RESTful API / Architecture

- RE stands for Representational, S for State, and T for Transfer.
- RESTful is a convention, or a structured way, of writing routes to manage communication between the client and the server.
- It makes the code more readable and easier to understand for developers.
- It helps show HTML, CSS, and JavaScript pages using well-defined RESTful routing.

Task to Perform RESTful Routing

- RESTful routing is often used for performing CRUD operations:
  - Create
  - Read
  - Update
  - Delete
- Performing a CRUD operation. → Create Read Update Delete
- It is a convention not compulsory → kuch bhi kr kste ho
- We will read this table from right to left.
- jab bhi database ke under change hoga to →post request jayegi
- redirect → means get request bhejo, and jis path par bhejna hai vo hum btate hai.

Table for RESTful CRUD Operations

We read this table **from right to left** to understand which HTTP method corresponds to each action:

Route name	URL	HTTP Verb	Description
Index	/blogs	GET	Display a list of all blogs
New	/blog/new	GET	Show form to make new blogs
Create	/blogs	POST	Add new blog to database, then redirect
Show	/blogs/:id	GET	Show info about one blogs
Edit	/blogs/:id/edit	GET	Show edit form of one blog
Update	/blogs/:id	PUT	Update a particular blog, then redirect
Destroy	/blogs/:id	DELETE	Delete a particular blog, then redirect

Get vs Post

Get	Post
This should be used to fetch / retrive the data	this should be used to make the changes in the server, (DB)
can we send data using it → yes we can do but hum esa nhi krte why? here the data is available inside your URL as query paramter (?)	**Here data is available as req.body(), if we send data to URL me nhi ata to usye dekhna ho to we have req.body()**
get request are less secure because data is available on Url. data is visible	these are more secure because data is not present in url
data set using Get that willbe a plain string and only available as query paramter.	data of different type can be sent text, xml, json etc.
limited amount of data can be send 4096 (not sure)	huge huge amount of data can be sent by post.

Sending data from Post method

req.body()

- if i want to see data arha hai ya nhi we will console req.body()

```
// post request ko handle
app.post('/user' , (req,res)=>{
    console.log(req.body);
    res.send('post method ke through')
})
```

but it is giving me undefined → i cannot see the data why?

```
[nodemon] starting `node index.js`
server connected at port 8080
undefined
```

Why undefined?

- body() is also an object
- jab bhi me post req ki madat se data bhejta hun → the entire data is coming in the formate of form data (type of data)
- by default mera jo req.body() undefined hoti hai, hum jab bhi dekhe.
- Therefore because of this default behavior, I can see undefined.

req.body

- It contains key-value pairs of data submitted to the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as express.json() or express.urlencoded().
- here it is given that we have to use body parsing middleware to resolve the undefined problem.
- if mujhe form data ke sath kaam krna hai to i will use → **express.urlencoded()**

```
//middleware to get the form data
app.use(express.urlencoded({ extended: true }))
```

i am getting the data in my req.body()

```
[nodemon] starting `node index.js`
server connected at port 8080
{ username: 'Rahul Prajapati', age: '24' }
```