full one-shot videos on :**JK Coding Pathshala YouTube channel**

JK Coding Pathshala

https://youtube.com/@jayeshkande9215?feature=shared

Unit IV	BACK END TECHNOLOGIES	(06 hrs)

Node.JS: Introduction to Node.JS, Environment Setup, Node.JS Events, Node.JS Functions, Node.JS Built-in Modules, File System, NPM, Install External Modules, Handling Data I/O in Node.JS, Create HTTP Server, Create Socket Server, Microservices- PM2.

ExpressJS: Introduction to ExpressJS, Configure Routes, Template Engines, ExpressJS as Middleware, Serving Static Files, REST HTTP Method APIs, Applying Basic HTTP Authentication, Implement Session Authentication.

MongoDB: NoSQL and MongoDB Basics, MongoDB-Node.JS Communication, CRUD Operations using Node.JS, Mongoose ODM for Middleware, Advanced MongoDB.

Q3) a) What is CRUD? Explain the CRUD using node. JS. **[9]**

b) What is node. JS? Explain file handling in node. js. **[8]**

OR

Q4) a) Write a code using Express. JS to illustrate the concept of roots. **[9]**

b) What is replication? Enlist the advantages of replication in database system. **[8]**

P.T.O.

Q3) a) List and explain the features of advanced MongoDB. **[6]**

b) Explain any four methods of console object in node.js with suitable examples. **[6]**

c) Write a short note on PM2 microservices. **[5]**

OR

Q4) a) Explain the callbacks in node.js with a suitable example. **[6]**

b) What is the purpose of map reduce? Explain it with a suitable example. **[6]**

c) Write a short note on Mongoose ODM **[5]**

Q3) a) Write and explain a simple application using REST HTTP Method API in node JS. **[6]**

b) Explain how to perform CRUD operations in a Node JS application. Provide examples of CRUD implementation. **[6]**

c) Write a short note on PM2 microservices. **[5]**

OR

Q4) a) What is template engine? How to create and use it using Express.JS? **[6]**

b) List and explain the features of advanced MongoDB. **[6]**

c) Explain the role of NPM (Node Package Manager) in Node.js development. **[5]**

Q3) a) What is CRUDE? Explain the crude CRUDE in node.JS. **[9]**

b) What is node.JS? Explain file handling in node.js. **[8]**

OR

Q4) a) Write a code using Express.JS to illustrate the concept of roots. **[9]**

b) What is replication? Enlist the advantages of replication in database system. **[8]**

Q3) a) Explain ExpressJS as middleware with example. **[5]**

b) What is NoSQL? Explain different features of MongoDB. **[6]**

c) Explain callbacks in NodeJS with suitable example. **[6]**

OR

Q4) a) Explain concept of routes in express with example. **[5]**

b) Write code to create collection, insert data & delete data in MongoDB using NodeJS. **[6]**

c) Explain NodeJS events with example. **[6]**

Node.JS

🌐 Node.JS: Introduction

🔹 What is Node.js?

Node.js ek open-source aur cross-platform JavaScript runtime hai, jo aapko JavaScript ko **server pe** run karne ki permission deta hai.

📌 **Use Case:** Web apps, APIs, Real-time chat, IoT, File servers, etc.

◆ Why Node.js?

✅ **Fast Execution (V8 Engine)**

💬 JavaScript code ko bahut hi fast run karta hai using Google's V8 Engine.

✅ **Asynchronous & Non-Blocking I/O**

💬 Ek task ke complete hone ka wait nahi karta – doosre tasks parallelly chalte hain.

✅ **Single Programming Language**

💬 Backend + Frontend dono me JavaScript use kar sakte ho.

✅ **Large Community & NPM (Node Package Manager)**

💬 Bahut saare free packages/middleware available hain.

⬘ Features of Node.js

Feature	Explanation
🔄 Non-blocking I/O	Requests ko block nahi karta, dusre requests bhi handle karta hai
⚡ Fast Performance	Google V8 engine use karta hai
🌐 Cross Platform	Windows, Mac, Linux sab pe chal sakta hai
🎁 NPM Support	1.5+ million packages free me available hain

◆ Where Node.js is used?

Popular Use-Cases:

- Web Applications
- REST APIs
- Real-Time Applications (e.g., Chat apps)
- IoT Devices
- Streaming Services

◆ Who Uses Node.js?

🌍 Big Companies using Node.js:

- Netflix
- LinkedIn
- PayPal
- Uber
- Walmart

⚙️ Node.js Environment Setup

🔹 Step 1: Download & Install Node.js

🌐 **Website:** https://nodejs.org
🔷 **LTS Version** choose karo – ye long-term support version hota hai.

🔹 Step 2: Verify Installation

Installation ke baad command prompt ya terminal me verify kar sakte ho:

```
node -v
npm -v
```

- Step 3: Install Code Editor (VS Code Recommended)

https://code.visualstudio.com

- Step 4: Create Your First Node.js File

```
// app.js
console.log("Hello from Node.js!");
```

```
node app.js
```

🔁 Node.js Events

🔹 What are Events in Node.js?

Node.js me **event ka matlab hai koi bhi ghatna** – jaise file read hona, data receive hona, user ka request bhejna. Node.js ek **event-driven** system hai.

📌 **Think Like:**
Jaise koi bell bajta hai (event) aur peon aata hai (listener).

◆ Event-Driven Architecture

🎁 **Node.js** has something called the **EventEmitter** class.

You can:

- **Emit** (trigger) an event
- **Listen** (handle) that event

its > Event.js > app.get("/") callback

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

const EventEmitter=require("events");//no need to install
//EventEmitter ek class

const app=express();

//event ka object
let count=0;
const event =new EventEmitter();
event.on("countAPI",()=>{
  count++
  console.log("hello event called",count)
})

app.get("/",(req,res)=>{

  res.send("hello everyone!!!");
  event.emit("countAPI");
})
```

```
app.get("/",(req,res)=>{

  res.send("hello everyone!!!");
  event.emit("countAPI");
})

app.get("/add",(req,res)=>{

  res.send("hello everyone add!!!");
   event.emit("countAPI");
})

app.get("/search",(req,res)=>{

  res.send("hello everyone search!!!");
   event.emit("countAPI");
})

app.listen(5000);
```

Event.js | http://localhos... ●

http://localhost:5000/add

Save | No Environment

GET http://localhost:5000/add Send

Params | Authorization | Headers (7) | Body | Scripts | Settings | Code | Cookies

Query Params

	Key	Value
	Key	Value

Body | Cookies | Headers (7) | Test Results | Status: 200 OK | Time: 10 ms | Size: 249 B

Pretty | Raw | Preview | HTML

```
1  hello everyone add!!!
```

OUTPUT | TERMINAL | PORTS | POSTMAN CONSOLE

```
PS C:\Users\Admin\Desktop\Unit4> node "c:\Users\Admin\Desktop\Unit4\NodeEvents\Event.js"
hello event called 1
hello event called 2
hello event called 3
```

🔧 What is a Function in Node.js?

In Node.js (just like in regular JavaScript), a **function** is a reusable block of code designed to perform a particular task.

✅ Types of Functions in Node.js

1. **Named Functions**
2. **Anonymous Functions (Function Expressions)**
3. **Arrow Functions (ES6)**
4. **Callback Functions**
5. **Asynchronous Functions (with setTimeout)**
6. **Async/Await Functions (Promise-based)**

1. Named Functions

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
greet("Jayesh");
```

2. Anonymous Functions (Function Expressions)

```
const greet = function(name) {
  console.log("Hello, " + name + "!");
};
greet("Jayesh");
```

3. **Arrow Functions (ES6)**

```
const greet = (name) => {
  console.log(`Hello, ${name}!`);
};
greet("Jayesh");
```

In Node.js, **callbacks** are functions passed as arguments to other functions and are executed after some kind of operation completes. Because Node.js is asynchronous, callbacks are essential to handle operations like reading files, making network requests, or querying databases without blocking the program.

4. Callback Functions

Functions passed as arguments to other functions.

```
function greet(name, callback) {
  console.log("Hello, " + name);
  callback();
}

function sayBye() {
  console.log("Goodbye!");
}

greet("Jayesh", sayBye);
```

5. **Asynchronous Functions (with setTimeout)**

```
function delayedGreeting(name) {

  setTimeout(() => {

    console.log(`Hello after 2 seconds, ${name}`);
  }, 2000);
}
delayedGreeting("Jayesh");
```

6. **Async/Await Functions (Promise-based)**

1. Promise-based Function

📌 Ye function ek **Promise return** karta hai jo 1 second ke baad "Data fetched!" message ke saath **resolve** hota hai.

```
function getData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Data fetched!");
    }, 1000);
  });
}
```

2. Async/Await Function

📌 **Ye function async hai aur getData() ke resolve hone ka wait karta hai using await, phir result ko console mein print karta hai.**

```
async function fetchData() {
  const data = await getData();
  console.log(data);
}
```

Final Call

```
fetchData();
```

📌 Ye line fetchData() function ko call karti hai jisme async/await use ho raha hai.

🎁 Example with Node.js Module

You can **export functions** in Node.js to use in other files.

👉 math.js

```
function add(a, b) {
  return a + b;
}

module.exports = add;
```

👉 app.js

```
const add = require('./math');
console.log(add(5, 3)); // Output: 8
```

Tips:

- Use function keyword for old-style functions.
- Use => arrow functions for cleaner syntax.
- Use module.exports and require() to reuse functions across files.

important methods of the console object in Node.js

1. console.log()

•**Purpose:** Prints output or messages to the console (standard output).

•**Usage:** Used for general logging.

```
console.log("Hello, World!");
console.log("Sum of 2 + 3 =", 2 + 3);
```

```
Hello, World!
Sum of 2 + 3 = 5
```

2. console.error()

•**Purpose:** Prints error messages to the console (standard error).

•**Usage:** Used for logging errors.

```
console.error("This is an error message");
```

```
This is an error message
```

3. console.warn()

- **Purpose:** Prints warning messages.
- **Usage:** Used to indicate potential issues.

```
console.warn("Warning! Low disk space.");
```

```
Warning! Low disk space.
```

4. console.table()

•**Purpose:** Displays data in a table format for better readability.

•**Usage:** Useful to show arrays or objects in a tabular form

```
const users = [
  { name: "Amit", age: 25 },
  { name: "Sita", age: 30 },
];

console.table(users);
```

```
┌─────────┬─────────┬─────┐
│ (index) │  name   │ age │
├─────────┼─────────┼─────┤
│    0    │ 'Amit'  │ 25  │
│    1    │ 'Sita'  │ 30  │
└─────────┴─────────┴─────┘
```

🔧 Core Built-in Modules in Node.js

1. fs (File System)
Used to interact with the file system —
reading, writing, updating, deleting files.

Example:

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

fs.writeFileSync('hello.txt', 'Hello from Node.js!');
```

2. http
Used to create web servers and handle HTTP requests and responses.

Example:

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

http.createServer((req, res) => {
  res.end("Server is running");
}).listen(3000);
```

3. path
Helps in handling and transforming file paths.

Example:

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

const filePath = path.join(__dirname, 'folder', 'file.txt');
console.log(filePath);
```

4. os

Gives information about the system like memory, CPU, etc.

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

console.log("OS Platform:", os.platform());
console.log("Free Memory:",
os.freemem());
```

5. events

Implements event-driven programming using EventEmitter.

Example:

```
const EventEmitter = require('events');

const event = new EventEmitter();
event.on('greet', () => {
  console.log("Hello Event Triggered");
});
event.emit('greet');
```

File System (fs) module in Node.js

📁 fs Module in Node.js

The fs (File System) module allows you to **interact with files and directories** — like reading, writing, updating, and deleting files.

✅ How to use it?

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

Simple Examples

📌 1. **Create or Overwrite a File**

```
fs.writeFileSync('example.txt', 'Hello, File System!');
```

📌 2. **Read a File**

```
const data = fs.readFileSync('example.txt', 'utf8');
console.log(data);
```

📌 3. **Append Data to a File**

```
fs.appendFileSync('example.txt', '\nAppended text.');
```

📌 4. **Delete a File**

```
fs.unlinkSync('example.txt');
```

📌 5. **Check if a File Exists**

```
if (fs.existsSync('example.txt')) {
  console.log("File exists");
} else {
  console.log("File does not exist");
}
```

⚠️ **Notes:**

- Sync functions block the execution until the task is complete.
- For non-blocking operations, use asynchronous versions like fs.writeFile, fs.readFile, etc.

NPM

🎁 What is NPM?

NPM stands for **Node Package Manager**.

It is the **default package manager** for Node.js used to:

- Install libraries (packages/modules)
- Share your own packages
- Manage project dependencies

Basic NPM Commands

Command	Description
npm init	Create a package.json for your project
npm install <package>	Install a package locally
npm install -g <package>	Install a package globally
npm uninstall <package>	Remove a package
npm update	Update all dependencies
npm list	List installed packages

🎁 What is the Role of NPM in Node.js?

NPM (Node Package Manager) is an essential part of Node.js.
It helps **manage external packages** (libraries/modules) that you want to use in your project.

🎯 Why is NPM important in Node.js?

🔑 Role of NPM	🔍 Explanation
✅ **Install Packages**	You can install reusable Node.js libraries (like express, mongoose, etc.).
✅ **Manage Dependencies**	NPM keeps track of which packages your project needs in package.json.
✅ **Share Your Code**	You can publish your own packages to NPM.
✅ **Run Project Scripts**	NPM allows you to define and run custom scripts (e.g., start server, build project).
✅ **Version Control**	NPM helps manage and lock package versions using package-lock.json.

📁 NPM Structure in Node.js Projects

After running:

```
npm init -y
```

You get:

- **package.json** → Project metadata + dependencies.
- **node_modules/** → Folder with installed packages.
- **package-lock.json** → Locks exact version of dependencies.

🛠 **Example**

Let’s say you want to build a Node.js web server using **Express**:

```
npm install express
```

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

app.get('/', (req, res) => {
  res.send("Hello from Express!");
});

app.listen(3000);
```

- **In Short:**

NPM is like the App Store for Node.js — it gives you access to thousands of ready-made libraries and tools to speed up your development.

🎁 What Are External Modules?

External modules are third-party libraries (not built into Node.js) that you can install using NPM to add features to your project.
Examples: express, mongoose, chalk, lodash, etc.

Data Input/Output (I/O) in Node.js

📌 **What is Data I/O?**

Data I/O (Input/Output) refers to:

- **Input:** Reading data (from files, terminal, etc.)
- **Output:** Writing data (to files, terminal, etc.)

Create HTTP Server

🌐 How to Create an HTTP Server in Node.js

✅ Step 1: Import the http module

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

🔧 Node.js me HTTP Server ka matlab:

Node.js me hum **khud apna server bana sakte hain** bina kisi external software ke.

Ye server:

- Browser se request lega
- Reply karega jaise: “Hello, world!” ya koi web page

✅ Step 2: Create the server

```
const server = http.createServer((req, res) => {
  // Set response header
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Send response
  res.end('Hello from Node.js HTTP Server!');
});
```

✅ Step 3: Listen on a port

```
server.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

Cheez	Matlab
HTTP Server	Browser se baat karta hai
Request	Browser se aata hai ("Page do!")
Response	Server se jaata hai ("Yeh lo page!")
Node.js HTTP Server	Apna chhota web server banana

Create Socket Server

Socket Server ek aisa program hota hai jo real-time me clients se connect hokar unse **continuous baat** kar sakta hai — bina page refresh kiye.

HTTP Server	Socket Server
Request aata hai → Reply	Continuous 2-way connection hota hai
Page refresh lagta hai	Real-time updates possible
Static websites ke liye	Chat apps, games, live feeds

Step	Action
1	Install Node.js
2	Project setup (npm init)
3	Install packages (socket.io, socket.io-client, readline)
4	Write server and client code
5	Run server (node server.js)
6	Run client (node client.js)
7	Chat in real-time via terminal

```
npm install socket.io socket.io-client readline
```

Yahan:

socket.io – server ke liye

socket.io-client – client ke liye

readline – input lene ke liye terminal se

```node
// server.js

const { Server } = require('socket.io');
const http = require('http');
const readline = require('readline');

// HTTP server banaya
const server = http.createServer();
// Socket.IO ko HTTP server ke saath bind kiya
const io = new Server(server);

// Terminal se input lene ke liye interface
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Jab client connect kare
io.on('connection', (socket) => {
  console.log('Client connected:', socket.id);

  // Jab client se message aaye
  socket.on('message', (msg) => {
    console.log('Client:', msg);
  });
```

Create file: server.js

```
  // Jab terminal se server koi message likhe
  rl.on('line', (input) => {
    socket.emit('message', 'Server: ' + input);
  });

  // Jab client disconnect ho
  socket.on('disconnect', () => {
    console.log('Client disconnected:', socket.id);
  });
});

// Server ko port 3000 pe run kiya
server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

```javascript
// client.js

const { io } = require('socket.io-client');
const readline = require('readline');

// Server se connect hone ka try
const socket = io('http://localhost:3000');

// Terminal se input lene ke liye
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Jab connection successful ho jaye
socket.on('connect', () => {
  console.log('Connected to server');

  // Terminal se user ka input leke server ko bhej do
  rl.on('line', (input) => {
    socket.emit('message', input);
  });
});
```

- Create file: client.js

```
// Jab server se message aaye
socket.on('message', (msg) => {
  console.log(msg);
});

// Jab server se disconnect ho jaye
socket.on('disconnect', () => {
  console.log('Disconnected from server');
});
```

👉 Open two terminal windows (or tabs):

Terminal 1 (Server):

```
node server.js
```

```
Server running on port 3000
Client connected: iD_ABC123
```

Terminal 2 (Client):

```
node client.js
```

```
Connected to server
```

Kaise input dena / output lena hai

➤ Server terminal me show hoga:

Client: Hello server!

Server terminal me likho:

Hello client!

Client terminal me kuch likho, jaise:

Hello server!

➤ Client terminal me show hoga:

Server: Hello client!

✅ **Kya isse "Socket Server" kehte hain?**
🔹 **Yes**, server.js file me jo tumne Socket.IO ke Server ka use kiya hai, **wo ek "WebSocket server" ya "Socket.IO server"** kehlata hai.

Ye ek real-time communication server hai, jo TCP/IP ke upar WebSocket protocol ka use karta hai

Microservices – Short Notes

◆ **Definition:**
Microservices ek aisa architecture hota hai jisme **poori application ko chhoti-chhoti self-contained services** me divide kiya jaata hai. Har service ka **apna logic, database, aur deployment pipeline** hoti hai.

◈ **Example Application**: eCommerce Site

Yeh sab alag-alag microservices ho sakte hain. Har ek service ka **own code, database, aur deployment** hota hai.

Service	Kaam
🛒 Product	Products add, update, list karna
👤 User	Login, register, profile
💳 Payment	Payment gateway integration
🎁 Order	Order create, track, update

Har service apne HTTP endpoints (like REST APIs) provide karti hai.

✅ **Benefits of Microservices:**

1. **Independent Deployment** – Har service ko alag se deploy/update kar sakte ho.
2. **Fault Isolation** – Agar ek service down ho jaaye, baaki app chalti rahegi.
3. **Scalability** – Sirf heavy traffic wale part ko scale kar sakte ho (e.g., payment).
4. **Tech Stack Flexibility** – Har service alag language, framework use kar sakti hai.
5. **Faster Development** – Teams parallel me kaam kar sakti hain.

❌ **Challenges of Microservices:**

- Service-to-service **communication complex** ho jaata hai.
- Deployment and orchestration tools (like Docker, Kubernetes) seekhna padta hai.
- **Debugging** distributed system tough hota hai.

⚙️ PM2 (Process Manager 2)

◆ **Definition:**
PM2 ek **production-grade process manager** hai jo mainly **Node.js applications** ke liye use hota hai. Ye system resources monitor karta hai aur crash hone par app ko **automatically restart** kar deta hai.

🔧 PM2 Key Features:

Feature	Description
🎁 Process Management	Start, stop, restart, list, logs sab handle karta hai
🔄 Auto Restart	App crash ho to automatically restart hota hai
📊 Monitoring	Live CPU, memory stats (pm2 monit)
🛠️ Cluster Mode	Multi-core CPU pe load balance karta hai
📜 Startup Script	Server reboot hone par apps auto-run (pm2 startup)

Common PM2 Commands:

```
pm2 start app.js          # Start app
pm2 list                  # List of running apps
pm2 stop app              # Stop app
pm2 restart app           # Restart app
pm2 logs                  # Show logs
pm2 monit                 # Live CPU/memory usage
pm2 delete app            # Remove app from PM2
pm2 startup               # Configure app to run on
system boot
```

🔗 Microservices + PM2 – Real World Use

Har microservice (like user.js, product.js, etc.) ko alag se PM2 se run kar sakte ho:

```
pm2 start user.js --name user-service
pm2 start product.js --name product-service
pm2 start payment.js --name payment-service
```

Or, use a config file for all:
📄 ecosystem.config.js

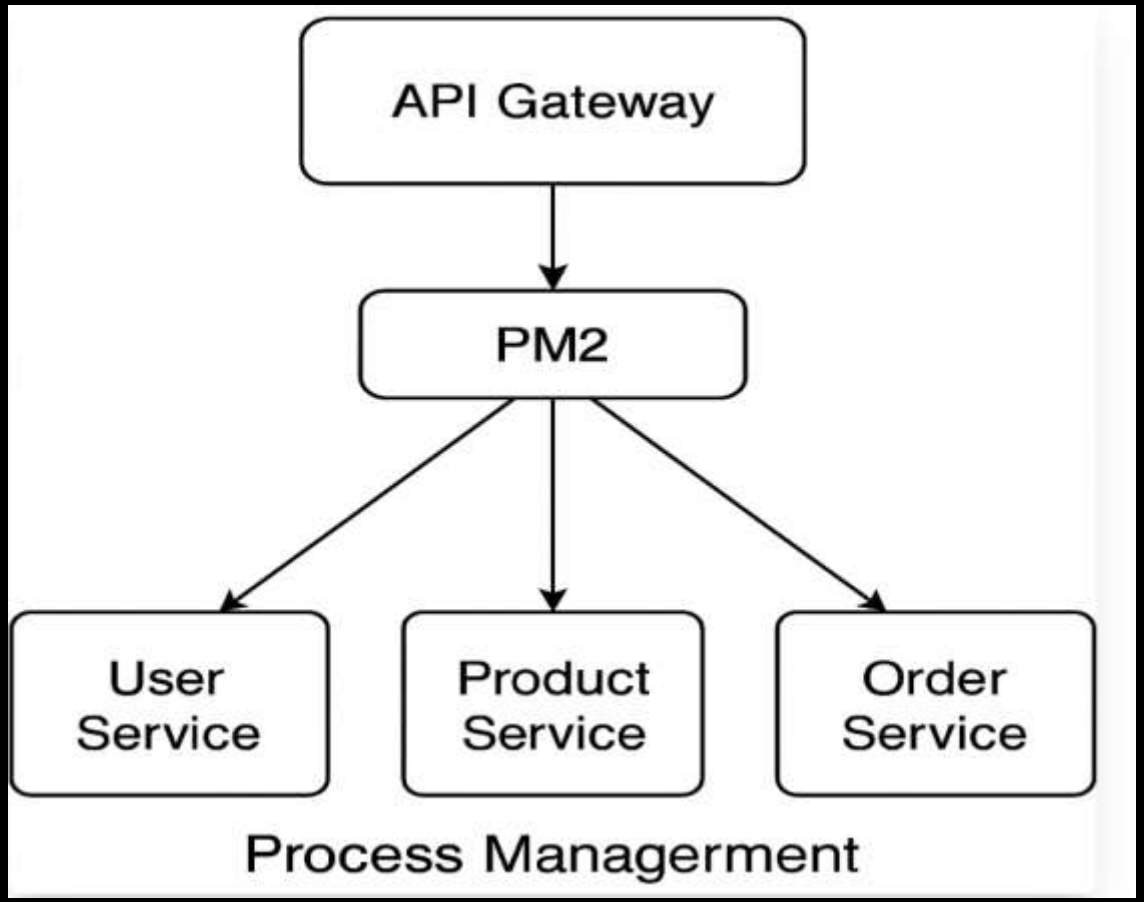
```
module.exports = {
  apps: [
    { name: 'user-service', script: 'user.js' },
    { name: 'product-service', script: 'product.js' },
    { name: 'payment-service', script: 'payment.js' },
  ]
};
```

Run all services with:

```
pm2 start ecosystem.config.js
```

Topic	Detail Summary
Microservices	App ko modules me todna (har ek chhota service), jo independently kaam karta hai
Benefits	Fast deployment, scalability, fault tolerance, tech flexibility
PM2	Node apps ko production me manage, monitor aur auto-restart karta hai
Together	Microservices ko alag-alag process ke form me run karna via PM2

Microservices Architecture with PM2 Management



ExpressJS

🚀 ExpressJS – Introduction

◆ **What is ExpressJS?**

ExpressJS ek **minimal aur flexible web application framework** hai jo **Node.js** ke upar kaam karta hai.

Ye framework web apps, RESTful APIs, aur backend services banane ke liye use hota hai — **fast, simple aur powerful** way mein.

🎁 Features of ExpressJS

Feature	Explanation
✅ Minimal	Bohot lightweight hai, sirf basic tools deta hai
🔧 Middleware Support	Request-response cycle ko control karne ke liye
🚦 Routing	URL-based request handling system
📂 Static File Support	HTML, CSS, images serve kar sakte ho
📡 REST API Ready	GET, POST, PUT, DELETE sab support karta hai
🔄 Integration Friendly	MongoDB, MySQL, JWT, etc. easily integrate ho jaate hain

💡 **Real-World Use**

ExpressJS ka use karke tum **backend server**, **API endpoints**, **auth system**, **CRUD operations**, aur **web applications** develop kar sakte ho.

🔧 How to Install ExpressJS

```
npm init -y
npm install express
```

Basic Example

📄 index.js

```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
  res.send('Hello Express!');
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

🔁 **ExpressJS Lifecycle**

1. Client sends request (browser / Postman)
2. Express router matches route
3. Middleware(s) execute (optional)
4. Response sent back to client

📘 Summary

Topic	Description
What is it?	Node.js web framework
Used for	Web apps, APIs, backend
Pros	Fast, simple, middleware-based
Common Uses	Routing, REST API, middleware, auth

how to configure routes in ExpressJS

What are Routes in ExpressJS?

Routes define **how your server responds** to different types of HTTP requests (like GET, POST, PUT, DELETE) at specific URLs.

✍ Basic Route Syntax

```
app.METHOD(PATH, HANDLER)
```

Part	Meaning
app	Your Express app instance
METHOD	HTTP method (GET, POST, etc.)
PATH	URL path (e.g. '/', '/users')
HANDLER	Callback function (req, res)

Example: Basic Routes

```
const express = require('express');
const app = express();
const PORT = 3000;

// GET route
app.get('/', (req, res) => {
  res.send('Welcome to Home Page');
});

// POST route
app.post('/login', (req, res) => {
  res.send('Login request received');
});

// PUT route
app.put('/user/:id', (req, res) => {
  res.send(`Update user with ID ${req.params.id}`);
});

// DELETE route
app.delete('/user/:id', (req, res) => {
  res.send(`Delete user with ID ${req.params.id}`);
});

app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

🎁 Route Parameters

```
app.get('/user/:name', (req, res) => {
  res.send(`Hello ${req.params.name}`);
});
```

➡️ GET /user/Amit will respond with:
Hello Amit

🔄 Modular Route Configuration

📁 File Structure:

```
project/
│
├── index.js
└── routes/
    └── userRoutes.js
```

routes/userRoutes.js

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('User list');
});

router.get('/:id', (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});

module.exports = router;
```

📄 index.js

```
et.js > ...
const express = require('express');
const app = express();
const userRoutes = require('./routes/userRoutes');

app.use('/users', userRoutes); // Mount route

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

➡ Now:

- GET /users → User list
- GET /users/101 → User ID: 101

Concept	Description
Route	URL + method combo
Modular Routing	Routes in separate files
Dynamic Parameters	Use :param in path
app.use()	Mounts route file to base path

Template Engine

- **What is a Template Engine?**

◆ **Template Engine ek aisa tool hota hai jo:**

- HTML pages me **dynamic data inject** karta hai
- JavaScript variables ko **HTML ke andar render** karta hai
- Server-side rendering ke liye use hota hai (SSR)

- **Real-World Example:**

Suppose you want to show a user’s name on a webpage:

<h1>Hello {{name}}</h1>

At runtime, {{name}} gets replaced by actual data like Hello Amit.

🚀 Popular Template Engines in ExpressJS

Template Engine	Syntax Style
EJS	<%= %>
Pug (Jade)	Indented
Handlebars (hbs)	{{ }}

How to Use Template Engine in ExpressJS (using **EJS**)

1. Initialize project & install packages

```
npm init -y
npm install express ejs
```

2. Create File Structure

```
project/
│
├── views/
│   └── index.ejs
└── app.js
```

3. Configure Template Engine in Express

📄 app.js

```
const express = require('express');
const app = express();
const PORT = 3000;

// Set EJS as the template engine
app.set('view engine', 'ejs');

// Route to render view
app.get('/', (req, res) => {
  const username = 'Amit';
  res.render('index', { name: username });
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

4. Create Template File

📄 views/index.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title>My Page</title>
</head>
<body>
  <h1>Hello <%= name %>!</h1>
</body>
</html>
```

➡️ Output in browser: Hello Amit!

How It Works?

- app.set('view engine', 'ejs') – tells Express to use EJS
- res.render('index', { name: 'Amit' }) – loads index.ejs and injects data
- <%= name %> – renders variable inside HTML

ExpressJS as Middleware

□ What is Middleware in ExpressJS?

◆ **Middleware ek function hota hai jo:**

•**Request (req)** aur **Response (res)** ke beech execute hota hai
•Request ko **process**, **modify**, **log**, ya **validate** karta hai
•Next middleware ko call karta hai using next()

🔁 Middleware Flow

Client → Middleware 1 → Middleware 2 → Route Handler → Response

🔍 **Middleware ko samjho ek traffic police ki tarah**

•Jab koi request (jaise browser se / path par jaana) server ko aati hai...

•Middleware us request ko **beech mein pakadta hai**, **kuch kaam karta hai** (jaise time log karna)

•Phir next() bol kar us request ko **aage bhej deta hai** actual response ke liye

Syntax of Middleware

```
function middleware(req, res, next) {
  // Logic
  next(); // Pass control to next handler
}
```

🔧 Types of Middleware

Type	Description
Application-level	Used via app.use() or app.get() etc.
Router-level	Works only on specific router files
Built-in	Like express.static()
Third-party	Like body-parser, morgan

Goal:

- Ek middleware banayenge jo har request ka **time log karega**
- Phir actual route se response bhejenge

- Step 1: Project Setup

```
mkdir express-middleware-demo
cd express-middleware-demo
npm init -y
npm install express
```

- Step 2: Create app.js

```
// app.js
const express = require('express');
const app = express();
const PORT = 3000;

//  Step 3: Create a simple middleware
app.use((req, res, next) => {
  console.log(' Time:', new Date().toLocaleString());
  next(); // very important! Pass control to next middleware or route
});

//  Step 4: Define a route
app.get('/', (req, res) => {
  res.send(' Hello from Express with Middleware!');
});

// Step 5: Start the server
app.listen(PORT, () => {
  console.log(` Server running at http://localhost:${PORT}`);
});
```

📌 **Example ka kya ho raha hai?**
1.app.use(...) se middleware set kiya
2.Jab bhi koi bhi route par request aayegi (like /), ye middleware pehle chalega
3.console.log() se time print karega
4.next() bolega to aage app.get('/') par control jaayega

🔁 **Agar next() nahi likhenge to?**
➡️ Response stuck ho jaayega!
Server request ko aage nahi bhejega, aur browser wait karta rahega.

🎁 **Route:**

```
app.get('/', (req, res) => {
res.send('Hello from Express with Middleware!’);
});
```

Yahaan actual response diya jaa raha hai jab middleware apna kaam kar ke request ko aage bhej deta hai.

Real Life Example:

Socho koi event entry gate hai:

- Sabse pehle guard (middleware) aapki ID check karta hai (ya time note karta hai)
- Phir aapko entry milti hai (route handler)

- Step 6: Run the App

```
node app.js
```

📤 Output on Terminal:

```
Time: 15/05/2025, 12:34:56 PM
Server running  http://localhost:3000
```

And in browser:

Hello from Express with Middleware!

Serving Static Files in ExpressJS

🎁 **What are Static Files?**

Static files are files like:

- HTML
- CSS
- JS (frontend)
- Images (PNG, JPG)
- Fonts, videos, etc.

These files **don't change** on the server and are sent **as-is** to the client/browser.

🎯 **Goal:**
Hum Express app banayenge jisme:
•Ek public folder hoga
•Usme index.html, style.css, image file honge
•Express unhe **directly browser ko serve** karega

Step-by-Step Guide

```
mkdir static-demo
cd static-demo
npm init -y
npm install express
```

□ Step 2: Create Folder Structure

```
static-demo/
│
├── public/
│   ├── index.html
│   ├── style.css
│   └── logo.png
└── app.js
```

Step 3: index.html

```html
<!-- public/index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Static Demo</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Welcome to Static Site</h1>
  <img src="logo.png" width="100">
</body>
</html>
```

🎨 Step 4: style.css

```
/* public/style.css */

body {
  font-family: Arial;
  background-color: #f0f0f0;
  text-align: center;
}
```

⚙️ Step 5: app.js

```
const express =
require('express');
const app = express();
const PORT = 3000;

// Static middleware
app.use(express.static('public'));

app.listen(PORT, () => {
  console.log(`Server running at
http://localhost:${PORT}`);
});
```

REST HTTP Method APIs

REST API Kya Hota Hai?

REST (Representational State Transfer) ek architecture style hai web APIs banane ke liye.
Ye standard **HTTP methods** ka use karta hai data ko **Create, Read, Update, Delete (CRUD)** karne ke liye.

📋 Common HTTP Methods (CRUD)

HTTP Method	Purpose	REST Action
GET	Data **read** karo	Read
POST	Naya data **create** karo	Create
PUT	Poora data **update** karo	Update
PATCH	Data ka **partially update**	Update
DELETE	Data **delete** karo	Delete

simple REST HTTP Method API application in Node.js using Express

🎯 **Objective:**
Hum ek **User Management API** banayenge jisme:
•GET → sab users dekhna
•POST → naya user banana
•PUT → existing user update karna
•DELETE → user delete karna

Step-by-Step REST API with Express

```
mkdir user-api
cd user-api
npm init -y
npm install express
```

```javascript
// app.js
const express = require('express');
const app = express();
const PORT = 3000;

// Middleware to parse JSON
app.use(express.json());

// Sample data (temporary in-memory database)
let users = [
  { id: 1, name: "Amit" },
  { id: 2, name: "Priya" }
];


//  GET: Get all users
app.get('/users', (req, res) => {
  res.json(users);
});


//  GET: Get user by ID
app.get('/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (user) res.json(user);
  else res.status(404).send("User not found");
});
```

```
//  POST: Create a new user
app.post('/users', (req, res) => {
  const newUser = {
    id: users.length + 1,
    name: req.body.name
  };
  users.push(newUser);
  res.status(201).json(newUser);
});


//  PUT: Update an existing user
app.put('/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (user) {
    user.name = req.body.name;
    res.json(user);
  } else {
    res.status(404).send("User not found");
  }
});


// DELETE: Remove a user
app.delete('/users/:id', (req, res) => {
  users = users.filter(u => u.id !== parseInt(req.params.id));
  res.send("User deleted");
});
```

```
//  Start the server
app.listen(PORT, () => {
  console.log(`Server running at
http://localhost:${PORT}`);
});
```

📥 Test This API Using Postman or Thunder Client

Method	URL	Body (JSON)	Use
GET	/users	—	Get all users
GET	/users/1	—	Get user with ID 1
POST	/users	{ "name": "Ravi" }	Create new user
PUT	/users/2	{ "name": "Neha" }	Update user with ID 2
DELETE	/users/1	—	Delete user with ID 1

Basic HTTP Authentication using Express.js

📌 What is Basic Authentication?

Basic authentication ek system hai jisme:

•Client request bhejta hai Authorization header ke saath

•Format hota hai:

Authorization: Basic Base64(username:password)

•Server check karta hai user aur password sahi hai ya nahi

🎯 **Real Life Analogy: Basic Authentication = Office Gate Pass System**

🏢 **Scene:**
Tum ek **office building** me ja rahe ho jahan ek **security guard** entry gate pe khada hai.

🚶♂️ **Step-by-step:**

1. Security Guard (Server) tumse kehta hai:
"Apna **ID card (username-password)** dikhaiye!"
2. Tum apna ID card dikhate ho (📤 Request header me username/password bhejte ho):
"Main **admin** hoon aur mera password hai **1234**."
3. Guard (Server) tumhara ID check karta hai:
•Agar **ID sahi** hai: Tumhe andar jane deta hai (next() call karta hai).
•Agar **galat ID**: Tumhe rokh deta hai, aur kehta hai:
"❌ Aapka ID galat hai, andar nahi ja sakte." (401 Unauthorized)

Real Life	Code Equivalent
Security Guard	basicAuth middleware
ID Card	Authorization: Basic <Base64(username:pass)>
Guard verifying ID	Code checking username === 'admin' && password === '1234'
Entry allowed	next() called
Entry denied	res.status(401).send()
Guard asks for ID	res.setHeader('WWW-Authenticate', 'Basic') (browser popup)

🛠️ Step-by-step Guide

🔧 Step 1: Project Setup

```
mkdir basic-auth-example
cd basic-auth-example
npm init -y
npm install express
```

```
//  Express module ko import kar rahe hain
const express = require('express');

//  Express app create kar rahe hain
const app = express();

//  Ye ek middleware function hai jo Basic Auth check karta hai
function basicAuth(req, res, next) {
  //  Client se aaya hua 'Authorization' header le rahe hain
  const authHeader = req.headers.authorization;

  // Agar client ne Authorization header hi nahi bheja
  if (!authHeader) {
    // Client ko bol rahe hain ki "Basic Auth chahiye" (browser popup dikhayega)
    res.setHeader('WWW-Authenticate', 'Basic');

    // Unauthorized access dena mana kar diya (status 401)
    return res.status(401).send('Authorization required');
  }

  // Authorization header se credentials nikal rahe hain
  // Example: "Basic YWRtaW46MTIzNA=="
  const base64Credentials = authHeader.split(' ')[1]; // "YWRtaW46MTIzNA=="
```

```
  //  Base64 se decode karke "admin:1234" banayenge
  const decoded = Buffer.from(base64Credentials, 'base64').toString(); // "admin:1234"

  // ":" ke basis par username aur password alag kar rahe hain
  const [username, password] = decoded.split(':');

  // Agar sahi username aur password hain toh aage allow karo
  if (username === 'admin' && password === '1234') {
    next(); // middleware pass, next handler chalu hoga
  } else {
    //  Agar galat username/password hai toh access deny kar do
    res.status(401).send('Invalid username or password');
  }
}

// Ye ek public route hai — bina login ke bhi access ho sakta hai
app.get('/', (req, res) => {
  res.send('Welcome! This is a public page.');
});

//  Ye ek secure/protected route hai — isme Basic Auth lagayenge
app.get('/dashboard', basicAuth, (req, res) => {
  res.send('Welcome to the secret dashboard 🔒');
});

// Server ko port 3000 par chalu kar rahe hain
app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

How to Test:

- Visit in browser: http://localhost:3000/dashboard
- Aapko username/password maangne ka popup milega.

- Username: admin
- Password: 1234

- ahi login → dashboard dikhayega
- Galat login → 401 Unauthorized

Part	Explanation
req.headers.authorization	Client ka auth data aata hai header me
Buffer.from(...).toString()	Base64 se decode karte hain
res.setHeader('WWW-Authenticate', 'Basic')	Browser ko bolta hai popup dikhane ke liye
username === 'admin'	Hardcoded login check
next()	Agla route execute hota hai agar auth pass ho

Session Authentication

🔐 **What is Session Authentication?**
Session authentication me, **user ek baar login karta hai**, server uske liye **session store karta hai** (memory/database me),
aur user ko ek **cookie** di jati hai. Har request me wo cookie jaati hai, jisse user pahchana jaata hai.

🎯 **Real Life Analogy:**
"Cinema Hall Entry with Token Slip (Session)"
🎬 **Scene:**
1.Tum ek **cinema hall** (movie theater) ke counter pe jaate ho.
2.Tum **ticket kharidte ho** (login karte ho).
3.Ticket lene ke baad tumhe ek **entry slip** milti hai (session ID/cookie).
4.Jab tak tumhare paas slip hai, tum **andar ghoom sakte ho**, restroom ja sakte ho, canteen ja sakte ho – koi dubara ticket nahi poochta (authenticated ho).
5.Agar tum **slip kho dete ho** ya samay khatam ho jaata hai (logout/session expired), toh tumhe wapas **ticket dikhani padegi** (login again).

🎁 Step-by-step Setup

🔧 Step 1: Project setup

```
mkdir session-auth-example
cd session-auth-example
npm init -y
npm install express express-session body-parser
```

Step 2: app.js

```
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const app = express();

// Middleware to parse form data
app.use(bodyParser.urlencoded({ extended: true }));

//  Session configuration
app.use(session({
  secret: 'mySecretKey',       // secret key for session
  resave: false,               //  session ko har request me save mat karo
  saveUninitialized: false     //  jab tak session me kuch store na ho, tab tak save mat karo
}));

// Login form route
app.get('/', (req, res) => {
  res.send(`
    <h2>Login</h2>
    <form method="POST" action="/login">
      <input name="username" placeholder="Username" required /><br/>
      <input name="password" type="password" placeholder="Password" required /><br/>
      <button type="submit">Login</button>
    </form>
  `);
});
```

```
// Handle login
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  // Simple hardcoded check
  if (username === 'admin' && password === '1234') {
    //  Store user info in session
    req.session.user = username;
    res.send('Login successful! <a href="/dashboard">Go to dashboard</a>');
  } else {
    res.send('Invalid login. <a href="/">Try again</a>');
  }
});

// Protected route
app.get('/dashboard', (req, res) => {
  //  Check session user
  if (req.session.user) {
    res.send(`Welcome ${req.session.user}! <a href="/logout">Logout</a>`);
  } else {
    res.send(' You are not logged in. <a href="/">Login</a>');
  }
});

// Logout route
app.get('/logout', (req, res) => {
  req.session.destroy(); // 🧹 session data delete
  res.send(' You are logged out. <a href="/">Login again</a>');
});
```

```
// Start server
app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

- **Test the Flow**

1. Run app: node app.js
2. Go to: http://localhost:3000
3. Login with:
   - Username: admin
   - Password: 1234
4. Redirects to dashboard if login is correct
5. Logout clears session

Feature	Kya karta hai
express-session	Session manage karta hai
req.session.user	Login user ka data store hota hai
req.session.destroy()	Session logout ke time remove hota hai
Session cookie	Browser me ek cookie store hoti hai (by default named connect.sid)

Real Life	Code Equivalent
Ticket lena	Login (username + password)
Entry slip	Session ID / Cookie
Slip check karna	req.session.user check
Slip expire ho jana	Session expire / logout
Slip ke bina kuch nahi milta	Session ke bina protected route deny

Code me session kaha kaam karta hai?

- Jab user login karta hai, server ek session **create** karta hai
- Server ek **unique ID** client ko cookie ke through bhejta hai
- Har baar jab client koi request bhejta hai, wo cookie ke sath hoti hai
- Server us cookie/session ID se user ko **verify** karta hai

```
User                    Server

  | --- Login Request --->     (User sends username/password)
  |                         |
  | <--- Validate Login ------ (Server checks credentials)
  |                         |
  | <--- Send Session ID ----- (Server creates session & sends cookie)
  |                         |
  | --- Request with Cookie-> (User sends requests with session cookie)
  |                         |
  | <--- Verify Session ------ (Server checks session ID & user info)
  |                         |
  | <--- Send Protected Data - (Server responds with data)
  |
  | --- Logout Request ----->  (User logs out)
  |                         |
  | <--- Destroy Session ---- (Server destroys session)
```

MongoDB Basics

1. What is NoSQL?

NoSQL ka matlab hota hai "Not Only SQL".

Yeh ek database type hai jo traditional **Relational Databases (like MySQL, Oracle)** se alag kaam karta hai.

🔑 **Key Points:**

- Tables nahi hote, **collections/documents** hote hain.
- **Schema-less** – fix structure ki zarurat nahi.
- **Horizontal scaling** – data zyada ho to easily distribute kar sakte hain.
- Fast for big data and real-time applications.

🍃 2. What is MongoDB?

MongoDB ek open-source NoSQL database hai.
Isme data ko **JSON-like format (BSON)** me store kiya jata hai.

MongoDB ke Main Features:
- Document-oriented database
- High performance
- High availability
- Easy scalability
- Uses BSON (Binary JSON)

🔧 3. MongoDB Basics – Step-by-step

Step 1: Install MongoDB

•Go to: https://www.mongodb.com/try/download/community

•OS ke according installer download karo.

•Install karne ke baad mongod aur mongo commands available honge.

Structure Summary

SQL	MongoDB
Table	Collection
Row	Document
Column	Field
JOINs	Embedded Documents
Schema-based	Schema-less

Step 4: Create / Use Database

```
use myDatabase
```

Agar database hai to use karega, warna new bana dega.

Step 5: Create Collection & Insert Document

```
db.students.insertOne({
  name: "Rahul",
  age: 21,
  course: "BCA"
})
```

MongoDB ke CRUD operations

📂 **Collection: students**
Hum students naam ki collection banayenge jisme student details honge.

Step 1: Use Database

```
use mySchool
```

1. CREATE – Insert data

```
db.createCollection("students")
```

```
show collections
```

✅ Single insert

```
db.students.insertOne({
  name: "Amit",
  age: 20,
  course: "BCA"
})
```

✅ Multiple insert

```
db.students.insertMany([
  {name: "Priya", age: 21, course: "MCA"},
  {name: "Ravi", age: 22, course: "BSc"}
])
```

🔍 2. READ – Fetch data

✅ Sabhi documents dekhna

```
db.students.find()
```

✅ Filter laga ke dekhna

```
db.students.find({name: "Priya"})
```

✅ Thoda format me dekhna (pretty print)

```
db.students.find().pretty()
```

🔧 3. UPDATE – Change data

✅ Single update (set new age)

```
db.students.updateOne(
  {name: "Amit"},
  {$set: {age: 21}}
)
```

✅ Multiple update (e.g., sabhi BCA students ka course update karo)

```
db.students.updateMany(
  {course: "BCA"},
  {$set: {course: "BCA Honors"}}
)
```

❌ 4. DELETE – Remove data

✅ Delete one student

```
db.students.deleteOne({name: "Ravi"})
```

✅ Delete multiple students (age > 21)

```
db.students.deleteMany({age: {$gt: 21}})
```

⚙️ Step-by-Step: MongoDB & Node.js Communication

1. Requirements

✅ Node.js installed
✅ MongoDB server (local ya MongoDB Atlas)
✅ Code editor (e.g., VS Code)

📁 2. Project Setup

```
mkdir mongo-node-app
cd mongo-node-app
npm init -y
```

✅ MongoDB Driver install karo:

```
npm install mongodb
```

📄 3. Create index.js File

📄 4. Code: MongoDB se Connect Karna

```javascript
// 1. MongoDB client import karo
const { MongoClient } = require("mongodb");

// 2. Connection string (local MongoDB)
const uri = "mongodb://127.0.0.1:27017";

// 3. Client create karo
const client = new MongoClient(uri);

// 4. Async function to connect
async function run() {
  try {
    await client.connect();
    console.log("Connected to MongoDB");

    // 5. Database aur collection choose karo
    const db = client.db("mySchool");
    const collection = db.collection("students");

    // 6. Data insert (CREATE operation)
    const result = await collection.insertOne({
      name: "Rahul",
      age: 21,
      course: "BCA"
    });

    console.log("Inserted ID:", result.insertedId);
```

```javascript
  } catch (err) {
    console.error("Error:", err);
  } finally {
    await client.close();
  }
}

run();
```

🏃 Step 5: Run the File

```
node index.js
```

Agar sab sahi hai, to output dikhega:

```
✅ Connected to MongoDB
🎁 Inserted ID: <some id>
```

Node.js + Express + MongoDB Based CRUD API Project for Student Data Management

◆ What is a CRUD API?

CRUD ka full form hota hai:

- **C** – Create (Naya data add karna)
- **R** – Read (Data fetch karna)
- **U** – Update (Existing data update karna)
- **D** – Delete (Data delete karna)

API (Application Programming Interface) ke through hum frontend ya tools (Postman) se backend ke saath interact karte hain.

◈ Tools and Technologies Used:

Technology	Role
Node.js	JavaScript runtime environment (server side JavaScript)
Express.js	Lightweight backend web framework
MongoDB	NoSQL database jisme hum data store karte hain
Mongoose	MongoDB ke liye ODM (Object Data Modeling) library
Postman	API testing tool

◆ **Project Flow:**

1.User API ko call karta hai (via Postman ya browser).

2.Request server (Express) tak pahuchti hai.

3.Server request ko handle karta hai (GET, POST, PUT, DELETE).

4.MongoDB me data add, read, update ya delete hota hai.

5.Server response return karta hai (success ya data).

Method	Use (काम क्या करता है)	One Line Explanation
GET	Data **read/fetch** karne ke liye	"Server se data lene ke liye use hota hai."
POST	Naya data **add/create** karne ke liye	"Server me naya data bhejne ke liye use hota hai."
PUT	Existing data ko **update** karne ke liye	"Pehle se maujood data me changes karne ke liye."
DELETE	Data ko **remove/delete** karne ke liye	"Server se kisi data ko delete karne ke liye hota hai."

🛠 STEP 1: Install Node.js

Agar Node.js install nahi hai:
👉 Download from: https://nodejs.org

Check version:

node -v npm -v

📁 STEP 2: Create Project Folder

```
mkdir student-api
cd student-api
```

🎁 STEP 3: Initialize Node Project

```
npm init -y
```

This creates a package.json file.

📦 STEP 4: Install Required Packages

```
npm install express mongoose body-parser dotenv
```

- express → Backend framework
- mongoose → MongoDB se connect karne aur query karne ke liye
- body-parser → Request body ko JSON me convert karta hai
- dotenv → Secret variables manage karne ke liye

✍️ STEP 5: Create Files and Folders

```
student-api/
├── server.js        ◆ Main backend file
├── .env             ◆ Secret config file
├── models/
│   └── student.js   ◆ MongoDB schema
```

MongoDB

Log in to your account

Don't have an account? Sign Up

You have successfully logged out.

Google

GitHub

Or with email and password

Email Address

Next

MongoDB 8.0 is here

Up to 32% higher throughput, improved horizontal scaling, expanded queryable encryption capabilities, and more.

See everything that's new →

Clusters

Create cluster

Cluster0

Connect

Edit configuration

Monitoring for Cluster0 is paused.

Monitoring will automatically resume when you connect to your cluster.

Visit the documentation for more info

+ Add Tag

Connect to Cluster0

1. Set up connection security
2. Choose a connection method
3. Connect

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

Atlas SQL
Easily connect SQL tools to Atlas for data analysis and visualization

Go Back

Close

Connect to Cluster0

Set up connection security ✓ — Choose a connection method ✓ — 3 Connect

Connecting with MongoDB for VS Code

1. Install MongoDB for VS Code.

In VS Code, open "Extensions" in the left navigation and search for "MongoDB for VS Code." Select the extension and click install.

2. In VS Code, open the Command Palette.

Click on "View" and open "Command Palette."
Search "MongoDB: Connect" on the Command Palette and click on "Connect with Connection String."

3. Connect to your MongoDB deployment.

Paste your connection string into the Command Palette.

```
mongodb+srv://kbtug22146:<db_password>@cluster0.0iigr1v.mongodb.net/
```

Replace **<db_password>** with the password for the **kbtug22146** user. Ensure any options are URL encoded. You can edit your database user password in Database Access.

4. Click "Create New Playground" in MongoDB for VS Code to get started.

Learn more about Playgrounds

RESOURCES

- Connect to MongoDB through VSCode
- Explore your data with playgrounds
- Access your Database Users
- Troubleshoot Connections

Go Back

Done

JAYESH'S ORG - 2025-04-01 > FOODWEB

Clusters

Find a database deployment...

Cluster0 | Connect | View Monitoring | Browse Collections | ···

Monitoring for Cluster0 is Paused

Monitoring will automatically resume when you connect to

Visit the documentation for more info.

JAYESH'S ORG - 2025-04-01 > FOODWEB

Database Access

Database Users | Custom Roles

+ ADD NEW DATABASE USER

JAYESH'S ORG - 2025-04-01 > FOODWEB > DATABASES

Cluster0

Overview | Real Time | Metrics | **Collections** | Atlas Search | Query Insights | Performance Advisor

DATABASES: 2 COLLECTIONS: 9

+ Create Database

Search Namespaces

- food-del
  - foods
  - orders
  - users
- sample_mflix

food-del.foods

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 571B TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 36KB

Find | Indexes | Schema Anti-Patterns 0 | Aggregation | Search Indexes

Generate queries from natural language in Compass

Filter `Type a query: { field: 'value' }`

QUERY RESULTS: 1-4 OF 4

Insert Document

To collection foods

VIEW {}

```
1 {"_id":{"$oid":"6818e32a3fedf2d59e01b1ea"},
2 }
```

STEP 6: .env File (MongoDB Connection)

Create .env file:

```
PORT=3000
MONGODB_URI=your_mongodb_connection_string
```

👉 Get connection string from MongoDB Atlas

- STEP 7: Create models/student.js

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

const studentSchema = new mongoose.Schema({
  name: String,
  marks: Number
});

module.exports = mongoose.model("Student", studentSchema);
```

🚀 STEP 8: Create server.js

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
require("dotenv").config(); // .env file ko load karta hai

const Student = require("./models/student");
const app = express();

app.use(bodyParser.json());
```

```
// MongoDB connect
mongoose.connect(process.env.MONGODB_URI)
.then(() => console.log("MongoDB Connected"))
.catch((err) => console.log("Mongo Error:", err));
```

```
// Get all students
app.get("/students", async (req, res) => {
  const students = await Student.find();
  res.send(students);
});

// Get student by name
app.get("/students/:name", async (req, res) => {
  const { name } = req.params;
  const students = await Student.find({ name });
  res.send(students);
});
```

```
// Add student
app.post("/add-student", async (req, res) => {
  const { name, marks } = req.body;
  const newStudent = new Student({ name, marks });
  await newStudent.save();
  res.send("Student added");
});

// Delete student
app.delete("/delete-student/:name", async (req, res) => {
  const { name } = req.params;
  await Student.findOneAndDelete({ name });
  res.send("Student deleted");
});
```

```
//  Update student
app.put("/update", async (req, res) => {
  const { name, marks } = req.body;
  const updated = await Student.findOneAndUpdate(
    { name },
    { $set: { marks } },
    { new: true }
  );
  res.send(updated);
});

app.listen(process.env.PORT, () => {
  console.log(`Server is running on port ${process.env.PORT}`);
});
```

STEP 9: Test API using Postman

Method	URL	Body/Param Example	Description
POST	/add-student	{ "name": "Amit", "marks": 90 }	Create new student
GET	/students	—	Get all students
GET	/students/Amit	—	Get student by name
DELETE	/delete-student/Amit	—	Delete student
PUT	/update	{ "name": "Amit", "marks": 95 }	Update student

POST http://localhost:3000/add-student Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1  { "name": "Amit", "marks": 90 }
```

Status: 200 OK Size: 13 Bytes Time: 37 ms

Response Headers 6 Cookies Results Docs

```
1  Student added
```

GET http://localhost:3000/students Send

Status: 200 OK Size: 195 Bytes Time: 23 ms

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1
```

Response Headers 6 Cookies Results Docs


```
[
  {
    "_id": "6818de449a8d151dc346e89b",
    "name": "jk",
    "marks": 20
  },
  {
    "_id": "6818dff5c7ab255431890610",
    "name": "Amit",
    "marks": 95,
    "__v": 0
  },
  {
    "_id": "6818e070c7ab255431890613",
    "name": "Amit",
    "marks": 90,
    "__v": 0
  }
]
```

GET http://localhost:3000/students/Amit Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1
```

Status: 200 OK Size: 137 Bytes Time: 145 ms

Response Headers 6 Cookies Results Docs


```
1  [
2    {
3      "_id": "6818dff5c7ab255431890610",
4      "name": "Amit",
5      "marks": 95,
6      "__v": 0
7    },
8    {
9      "_id": "6818e070c7ab255431890613",
10     "name": "Amit",
11     "marks": 90,
12     "__v": 0
13   }
14 ]
```

DELETE http://localhost:3000/delete-student/Amit Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1
```

Status: 200 OK Size: 15 Bytes Time: 32 ms

Response Headers 6 Cookies Results Docs

```
1  Student deleted
```

PUT http://localhost:3000/update Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1  { "name": "Amit", "marks": 95 }
```

Status: 200 OK Size: 67 Bytes Time: 119 ms

Response Headers 6 Cookies Results Docs

```
1  {
2    "_id": "6818dff5c7ab255431890610",
3    "name": "Amit",
4    "marks": 95,
5    "__v": 0
6  }
```

"Kya aap ne itne easy way mein padha hai?"

📚 What is Mongoose Middleware?

Mongoose Middleware ya **Hooks** wo functions hote hain jo automatically **execute hote hain** model ke operations ke **before ya after** events, jaise ki save, update, delete, etc.

Yeh aapko data process karne, validation, logging, ya kisi aur custom logic ko database operation se pehle ya baad run karne mein madad karta hai.

🔑 Types of Middleware in Mongoose

1.Pre Middleware (pre)

Operation se pehle run hota hai.

Example: pre('save') runs before saving a document.

2.Post Middleware (post)

Operation ke baad run hota hai.

Example: post('save') runs after saving a document.

Example of Mongoose Middleware

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

const studentSchema = new mongoose.Schema({
  name: String,
  marks: Number,
});

// Pre-save middleware: naam ko capitalize karega save hone se pehle
studentSchema.pre("save", function (next) {
  this.name = this.name.charAt(0).toUpperCase() + this.name.slice(1);
  console.log("Pre-save middleware running");
  next();
});

// Post-save middleware: save hone ke baad message print karega
studentSchema.post("save", function (doc) {
  console.log("Post-save middleware: Student saved:", doc.name);
});

module.exports = mongoose.model("Student", studentSchema);
```

📝 Short Note on Mongoose ODM

Mongoose is an **ODM (Object Data Modeling)** library for **MongoDB and Node.js**. It provides a structured way to interact with MongoDB using **JavaScript objects**.

◆ **Key Features:**
•**Schema-Based**: You define schemas for your data models (e.g., User, Product).
•**Model Methods**: Easily perform CRUD operations using methods like .find(), .save(), .updateOne(), etc.
•**Validation**: You can define required fields, data types, default values, etc.
•**Middleware (Hooks)**: Functions that run before or after database operations (like save, delete, update).
•**Relationships**: Supports referencing other documents (like joins) using populate().

◆ Why Use Mongoose?

- Makes MongoDB easier to work with in Node.js
- Adds structure and validation to otherwise flexible NoSQL data
- Clean API for database operations

⬍ Example:

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

const userSchema = new mongoose.Schema({
  name: String,
  email: String,
});

const User = mongoose.model('User', userSchema);
```

With this, you can do:

```
const newUser = new User({ name: "Amit", email: "amit@gmail.com" });
await newUser.save();
```

In short: **Mongoose helps you work with MongoDB in a more organized, schema-based, and developer-friendly way inside Node.js apps.**

🚀 Advanced MongoDB Features

MongoDB is not just a simple NoSQL database; it offers many advanced features that make it powerful for complex, large-scale, and real-time applications.

1. Aggregation Framework

Powerful tool to perform complex data processing and analytics by chaining multiple stages like filtering, grouping, sorting, and projecting data.

2. Indexing

Supports different types of indexes (single, compound, text, geospatial, hashed) to speed up query performance dramatically.

3. Sharding

Distributes data horizontally across multiple servers to handle large datasets and high throughput, enabling scalable architecture.

4. Replication (Replica Sets)

Provides high availability and fault tolerance by keeping multiple copies of data synchronized across servers with automatic failover.

5. Multi-document ACID Transactions

Allows atomic operations on multiple documents and collections, ensuring data consistency even in complex updates.

6. Change Streams
Enables real-time monitoring of data changes, useful for event-driven applications, live updates, and syncing data across services.
7. GridFS
A specification to store and retrieve large files (e.g., images, videos) by splitting them into smaller chunks.
8. Schema Validation
Enforces data integrity rules at the database level using JSON Schema, preventing invalid data insertion.
9. Full-Text Search
MongoDB supports powerful text search capabilities, including fuzzy search, stemming, and ranking, especially in Atlas.
10. Time Series Collections
Optimized for storing and querying time-stamped data such as sensor readings, logs, and financial da

📌 What is Replication?

Replication is the process of copying and maintaining database **data copies (replicas)** on multiple servers or nodes. This means the same data is stored on more than one machine to improve availability, fault tolerance, and performance.
In databases like MongoDB, replication is done using **Replica Sets**, where one node is primary (read/write) and others are secondary (read-only copies).

✅ Advantages of Replication in Database Systems

1.High Availability

If the primary server fails, one of the secondary servers can automatically become primary, minimizing downtime.

2.Fault Tolerance

Data loss risk reduces because copies of data exist on multiple servers.

3.Load Balancing

Read operations can be distributed to secondary replicas, improving read performance and reducing load on primary.

4.Disaster Recovery

In case of hardware failure or data corruption, replicas help recover data quickly.

5.Backup Without Downtime

Backups can be taken from secondary servers without affecting the primary database's performance.

6.Data Locality

Replicas can be located closer to users geographically, reducing latency for read operations.

📌 What is the Purpose of MapReduce?

MapReduce is a programming model used for processing and generating large data sets with a **distributed algorithm** on a cluster. It breaks down a big task into smaller sub-tasks, processes them in parallel, and then combines the results.

In databases like MongoDB, MapReduce is used to perform complex data aggregation and analysis that can’t be easily done with simple queries.

How Does MapReduce Work?

- **Map phase**: Processes input data and converts it into key-value pairs.
- **Reduce phase**: Aggregates these key-value pairs and produces a summarized output.

Example: Counting Number of Students per Marks

Suppose you have a collection of students with their marks:

```
[
  { "name": "Amit", "marks": 80 },
  { "name": "Rahul", "marks": 90 },
  { "name": "Sita", "marks": 80 },
  { "name": "Geeta", "marks": 90 }
]
```

You want to count how many students got each marks

Map Function (Emit marks as key, and 1 as value)

```
function map() {
  emit(this.marks, 1);
}
```

Reduce Function (Sum all values for each marks)

```
function reduce(key, values) {
  return Array.sum(values);
}
```

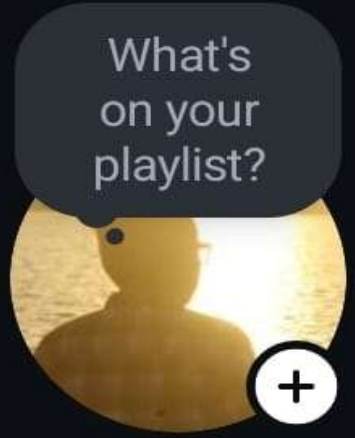
Running MapReduce in MongoDB

```
db.students.mapReduce(
  map,
  reduce,
  { out: "marks_count" }
);
```

Output in marks_count collection:

marks	value (count)
80	2
90	2

jayesh_kande_



Jayesh Kande

16	275	276
posts	followers	following

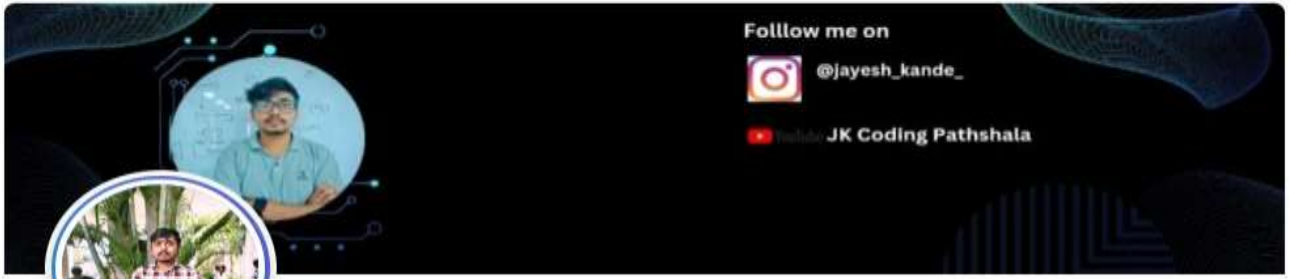
23

रास्ते बदलो, मंजिल नहीं

yt.openinapp.co/0y0qd

LinkedIn

Articles | People | Learning | Jobs | Games



Jayesh Kande

Third-Year IT Engineering Student | Aspiring Web Developer | Java Enthusiast | Data Structures & Algorithms Learner | Proficient in C, C++, Java, and MERN Stack | AI + Web Development Project Enthusiast

Nashik, Maharashtra, India · Contact Info

494 followers · 495 connections

See your mutual connections

Join to view profile | Message

Kbt engineering college nashik

✨ Thank You for Watching! ✨

➔📱 Follow us on Instagram:**@jayesh_kande_**
🔗 Connect with us on LinkedIn:[**Jayesh Kande]**