**Node Js**

1. **Defination :** 
   * + Node.js is an open-source, cross-platform runtime environment that allows developers to run JavaScript code on the server side.
     + It is built on the V8 JavaScript runtime engine, the same engine that powers the Google Chrome browser.

V8 is developed using C++.

* + - Node.js enables the execution of JavaScript outside the browser, allowing developers to build scalable and high-performance network applications.

1. **Latest Version :** 21.4.0
2. **Release On :** 27 May 2009
3. **Developed Using :** C , C++, Javascript
4. **Applications :** Web servers , API Development , Real-Time Applications , Microservices , etc.
5. **Javascript v/s node Js :** 
   * + - **Javascript :** JavaScript is a programming language that is primarily used for scripting in web browsers.(Run on browser).
       - **Node Js :** Node Js is a runtime environment that allows the execution of JavaScript code outside of the browser. It is commonly used for server-side development.(Run on Server Side).
6. **Npm v/s npx :** 
   * + - **Npm (node package manager) 🡪** npm is the package manager for Node.js. It is used for installing, managing, and sharing packages or libraries written in JavaScript. npm is also the default package manager for Node.js.

Usage **🡪** Installing a package globally: **npm install -g package-name**

* Installing a package locally: **npm install package-name**
* Running scripts defined in package.json: **npm run script-name**
* Managing dependencies for a project: **npm init** (to initialize a new project and create a package.json file)
* npm install (to install dependencies listed in package.json).
  + - * **Npx (node package execute ) 🡪** npx is a tool that comes with npm and is used to execute Node.js packages. It allows you to run binaries from Node.js packages without having to install them globally.

1. **To check node version :** node -v npm -v
2. **To Check Where the node install use this : (CMD : )** where node -> E:\APPS\Node Js\node.exe
3. **To Run node js file :**  node **.\filename.js** node **filename.js**
4. **Modules Import & Export :**

|  |  |
| --- | --- |
| **Export\_modules.js** | **Import\_modules.js** |
| **1. Exporting Variables**  var name = "Piyush";  module.exports = name;  **2. Exporting Objects**  var obj = {      first\_name: "Herik",      last\_name: "Sevak",      email: "Herik@gmail.com",      skills: ['HTML', 'CSS', 'Js', "Bootstrap"]  }  module.exports = obj;  **3. Exporting Functions**  function greet(name) {      return 'Hello, World! ' + name;  }  module.exports = greet; | **1. Importing Variables**  const import\_name = require('./export\_modules.js');  console.log('Importing Variables : ', import\_name);  **2. Importing Objects**  const import\_obj = require('./export\_modules.js');  console.log('Importing Objects : ', import\_obj);  **3. Importing Functions**  const import\_function = require('./export\_modules');  console.log('Importing Function:', import\_function("Piyush")); |

1. **Const path = require(‘path’); 🡪** The path module in Node.js provides utilities for working with file and directory paths.
2. **Modules in node Js :**

|  |  |  |
| --- | --- | --- |
| **1. Core Module**  Core modules are built-in modules that come bundled with Node.js. They provide fundamental functionality such as file system access, networking, and more.  **Example :**  fs (File System)  http (HTTP)  path (Path)  util (Utilities)  **Usage :**  const fs = require('fs');  const http = require('http'); | **2. Local Module Or Custom Module**  Local modules are user-defined modules created to organize code into separate files. These modules can be created for specific functionalities and reused across different parts of an application.  **Usage :**  const utils = require(‘.\recently created file name’); | **3. Third-party (npm) Modules**  Third-party modules are external packages or libraries available on the npm (Node Package Manager) registry. Developers can use these packages to add specific functionalities to their applications.  **Usage :**  **>> npm install express**  const express = require('express');  **>> npm install axios**  const axios = require(axios); |

|  |
| --- |
| **CORE MODULES : File System** |
| const fileSystem = require('fs');  **Handling Paths**  const path = require('path');  console.log(path); // Output : return all the function provided by path  const current\_directory\_path = path.join(\_\_dirname,'temp.txt');  console.log(current\_directory\_path);  // Output : D:\My Learning\Node Js\Node\03\_modules\join folder or file  **1.Reading File :**  The **fs.readFile(filePath,function(error,data))** method is used to read files on your computer.  const format = 'utf-8';  fileSystem.readFile(current\_directory\_path,format,(error,data)=>{      console.log("Reading : ",data);  })  **2. Writing File :** The **fs.writeFile(filePath,DataToWrite,Error Function)** method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created.  const data\_to\_write = "Writing Data to a file";  fileSystem.writeFile(current\_directory\_path,data\_to\_write,(error)=>{      console.log(error);  })  **3. Appending File :** The **fs.appendFile(filePath,dataToAppend,Error Function)** method appends the specified content at the end of the specified file  const data\_to\_append = "Appending new content data";  fileSystem.appendFile(current\_directory\_path,data\_to\_append,(error)=>{      console.log(error);  })  **4. Renaming File :** The **fs.rename(filePath,NewName,Error Function)** method renames the specified file.  const new\_file\_name = "temp2.txt";  fileSystem.rename(current\_directory\_path,new\_file\_name,(error)=>{      console.log(error);  })  **5. Deleting File :** The **fs.unlink(filePath,Error Function)** method deletes the specified file  fileSystem.unlink(current\_directory\_path,(error)=>{      console.log(error);  }) |

|  |
| --- |
| **CORE MODULE : HTTP ( createServer )** |
| **1.Handling Response**  const http = require('http');  **SYNTAX :  http.createServer(function(request,response).listen(port number)**  http.createServer((request, response) => {      response.write("Hello this is the response from server 'Method-2'");      response.end();  }).listen(2000);  **TERMINAL :** node filename.js |

1. **Npm init :** npm init is a command in Node.js and npm (Node Package Manager) that initializes a new npm package.**It creates a package.json file.**
2. **Package.json :** The package.json file is a central configuration file used in Node.js projects. It plays a crucial role in managing dependencies, scripts, and metadata for a Node.js application or module.
3. **Package-lock.json :** package-lock.json is a file automatically generated by npm (Node Package Manager) that describes the exact tree of dependencies, their versions, and additional metadata needed to reproduce the structure of the node\_modules folder in a consistent way.

|  |
| --- |
| **THIRD PARTY MODULES :** |
| 1. **Nodemon 🡪** nodemon is a utility that monitors for changes in files in a Node.js application and automatically restarts the server when changes are detected.   **Install : npm install nodemon**  **Add to Package.json : "scripts": {**  **"test": "echo \"Error: no test specified\" && exit 1",**  **"dev":"nodemon filename.js"**  **},**  **TERMINAL : npm run dev** |

1. **Getting Inputs from Terminal**

|  |  |
| --- | --- |
| const input = process.argv;  console.log(input);  **TERMINAL : node 1.js**  **OUTPUT :**  [    'E:\\APPS\\Node Js\\node.exe',    'd:\\My Learning\\Node Js\\Node\\04\_Getting\_Input\\1.js'  ] | const value1 = input[2];  const value2 = input[3];  **TERMINAL : node 1.js piyush thaware**  console.log(value1,value2);  **OUTPUT :**  [    'E:\\APPS\\Node Js\\node.exe',    'D:\\My Learning\\Node Js\\Node\\04\_Getting\_Input\\1.js',  **'piyush',**  **'thaware'**  ] |

1. **Programming Languages :**

|  |  |
| --- | --- |
| 1. **Synchronous Programming Language** | 1. **Asynchronous Programming Language** |
| A synchronous programming language is a type of programming language where tasks are executed sequentially and one at a time, blocking the execution of subsequent tasks until the current task is completed.  **Code :**  console.log("Task-1");  console.log("Task-2");  console.log("Task-3");  **Output :**  Task-1  Task-2  Task-3 | Asynchronous programming language is a programming language that supports asynchronous programming, a programming paradigm that allows tasks to be executed independently, without blocking the overall flow of the program.  **Code :**  console.log("Job-1");  setTimeout(() => {  console.log("Job-2");  }, 3000);  console.log("Job-3");  **Output :**  Job-1  Job-3  Job-2 |

**Express Js**

1. **Defination :** Express is a node js web application framework that provides broad features for building web and mobile applications.

It is used to build a single page, multipage, and hybrid web application.

It's a layer built on the top of the Node js that helps manage servers and routes.

1. **Install** : npm i express
2. **Features** :
   * + - Express provides a simple and effective way to define routes for your application.
       - Routes define how the application responds to different HTTP requests (GET, POST, PUT, DELETE, etc.) and URL patterns.
       - Express uses middleware functions to perform tasks in the request-response cycle.
       - Middleware functions have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. This allows developers to add functionality to the request or response, such as logging, authentication, or data parsing.
3. **Routing Using Express :**

|  |  |
| --- | --- |
| **Index.js**  const express = require('express');  console.log(express);   **------------------> Display Set of Tools Provided by Express**  const app = express();  **1. Sending Simple String Data.**  app.get('',(request,response)=>{    response.send("Hello , This is Home Page");  })  **2. Sending an Object as a Data.**  app.get('/about',(request,response)=>{  **const params = {**  **fname : "Piyush",**  **lname : "Thaware",**  **email : "Piyush@gmail.com"**  **}**      response.send(**params**);  })  Const port\_no = 2000;  app.listen(port\_no);  **Chrome :** http://localhost:2000 OR http://localhost:2000/about | **Response.send() :**  This function basically sends the HTTP response. The body parameter can be a String or a Buffer object or an object or an Array.  **Syntax :** res.send( **[body]** ) |

1. **Render HTML Templates & JSON Data :**

|  |  |
| --- | --- |
| const express = require('express');  const app = express();  **// Getting Current Path**  const path = require('path');  const publicPath = path.join(\_\_dirname, 'public');  **// load static files below from public folder**  app.use(express.static(publicPath));  app.listen(1000);  **Chrome :** <http://localhost:1000>  <http://localhost:1000/index.html>  http://localhost:1000/about.html | **App.use(path,callback) :**  The app.use() function is used to mount the specified middleware function(s) at the path which is being specified. It is mostly used to set up middleware for your application. |

1. **Remove Extension (index.html or about.html) and 404 Page handling :**

|  |  |
| --- | --- |
| const express = require('express');  const app = express();  **// Getting Current Path**  const path = require('path');  const publicPath = path.join(\_\_dirname,'public');  **// Handle Routing**  app.get('',(request,response)=>{  **response.sendFile(`${publicPath}/index.html`);**  })  app.get('/about',(request,response)=>{  **response.sendFile(`${publicPath}/about.html`);**  })  **// 404 Page Not Found**  app.get(' **/\*** ',(request,response)=>{  **response.sendFile(`${publicPath}/404.html`);**  })  app.listen(1000);  **Chrome :** <http://localhost:1000/>  http://localhost:1000/about | **Response.sendFile(fileName, options,function)**  The res.sendFile() function basically transfers the file at the given path and it sets the Content-Type response HTTP header field based on the filename extension.  **Syntax :** res.sendFile(fileName, options,function) |

1. **Getting Request in above code :**

|  |
| --- |
| app.get('',(request,response)=>{  **console.log(request.query);**  **console.log(request.query.fname);**  **console.log(request.query.lname);**      response.sendFile(`${publicPath}/index.html`);  })  **Chrome :** <http://localhost:1000/?fname=Piyush&lname=thaware>  **Output :** { fname: 'Piyush', lname: 'thaware' }  Piyush  thaware |

1. **EJS Template Engine : Passing Data To the html pages like as params or dictionary in python.**
   * + - EJS, or Embedded JavaScript, is a popular template engine for Node.js.
       - It allows you to embed JavaScript code directly within your HTML or other markup languages, making it easy to generate dynamic content on the server side.
       - **Install :** **npm install ejs**

|  |
| --- |
| 1. Create a views folder inside the current path where public is also present. 2. Create files with **extension** as **“ .ejs “** inside the views folder. 3. **Code :**   const express = require('express');  const app = express();  **// Setting Views Engine**  **app.set('view engine', 'ejs');**  **// creating individual views of each page**  app.get('', (\_, response) => {      const params = {          first\_name: "Piyush",          last\_name: "Thaware",          email: [Piyush@gmail.com](mailto:Piyush@gmail.com),  skills : ['Python','Django','React','Javascript']      }  **response.render('Signin', { obj: params });**  })  app.listen(1000);   1. Accessing Params in html page in signin page :    <h5>First Name : **<%= obj.first\_name  %></**h5>    <h5>Last Name : **<%= obj.last\_name  %></**h5>    <h5>Email Address : **<%= obj.email  %></**h5>   <h5>Skills : </h5>          <ul>  **<% obj.skills.forEach((item)=> { %>**  **<li>**  **<%= item %>**  **</li>**  **<% }); %>**          </ul>   1. **Chrome :** <http://localhost:1000/> |

1. **Template Inheritance (include) :** 
   * + - Create a common folder inside views an add the components like Navbar,Footer,Headers,Card.ejs file to apply to all the files.
       - ADD this component where you want :

**<%- include('common/Navbar')  %>**

1. **Middleware :**

* In Node.js, middleware refers to a function or a series of functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.
* Middleware functions can perform various tasks such as modifying the request and response objects, terminating the request-response cycle, or calling the next middleware in the stack.
* Middleware functions are essential in building scalable and modular web applications. They can be used for tasks such as authentication, logging, error handling, and more.
* **Types Of Middlewares :**

1. **Application Level Middleware :** These are middleware functions that are bound to the entire application and are executed for every incoming request.

|  |  |
| --- | --- |
| **Syntax :**  app.use((req, res, next) => {  // Application-level middleware  next();  }); | **Example :**  const express = require('express');  const app = express();  **const middleware\_name = (request,response,next)=>{**  **if(request.query.name=="Piyush" && request.query.password==12345){**  **response.send("Login Success");**  **}**  **else{**  **next();**  **}**  **}**  **app.use(middleware\_name);**  **// Routes**  app.get('', (request, response) => {      response.send("Home Page");  })  app.get('/about', (request, response) => {      response.send("About Page");  })  app.listen(1000);  **Chrome :** http://localhost:1000/?name=Piyush&password=12345  http://localhost:1000/about/?name=Piyush&password=12345 |

1. **Router Level Middleware:** Similar to application-level middleware, but bound to a specific router instead of the entire application.

|  |  |
| --- | --- |
| **Syntax :**  app.use((req, res, next) => {  // Application-level middleware  next();  });  **App.get(‘ ‘,middleware,function())** | **Example :**  const express = require('express');  const app = express();  **const middleware\_name = (request, response, next) => {**  **if (request.query.name == "Piyush" && request.query.password == 12345) {**  **response.send("Login Success");**  **}**  **else {**  **next();**  **}**  **}**  app.get('', (request, response) => {      response.send("Home");  });  app.get('/contact', **middleware\_name**, (request, response) => {      response.send("Contact");  })  app.listen(2000);  **Chrome :** http://localhost:2000/contact/?name=Piyush&password=12345 |

1. **Import/Export With Middleware :**

|  |  |
| --- | --- |
| **Export Middleware Function :** | **Import Middleware Function :** |
| const export\_middleware = (request, response, next) => {      if (request.query.name == "Piyush" && request.query.password == 12345) {          response.send("Login Sucess");      }      else if (request.query.name == "Piyush" || request.query.password == 12345) {          response.send("Login Failed")      }      else {          next();      }  }  module.exports = export\_middleware; | const express = require('express');  var app = express();  **// Importing Middleware**  **const import\_middleware = require('./middleware');**  app.get('',(request,response)=>{      response.send("Home");  })  app.get('/about',**import\_middleware**,(request,response)=>{      response.send("About");  })  app.get('/contact',**import\_middleware**,(request,response)=>{      response.send("Contact");  })  app.listen(3000);  **Chrome :** http://localhost:3000/contact/?name=Piyush&password=12345 http://localhost:3000/about/?name=Piyush&password=12345 |

**Alternative of above import code :**

|  |
| --- |
| const express = require('express');  var app = express();  **const import\_middleware = require('./middleware');**  const route = express.Router();  **route.use(import\_middleware);**  app.get('', (request, response) => {      response.send("Home");  })   * **Apply Routes To About & Contact Pages Below**   **route.get('/about', (request, response) => {**  **response.send("About");**  **})**  **route.get('/contact', (request, response) => {**  **response.send("Contact");**  **})**  **app.use('/',route);**  app.listen(2000);  **Chrome :** <http://localhost:2000/contact/?name=Piyush&password=12345>  http://localhost:2000/about/?name=Piyush&password=12345 |

**MongoDB + NodeJs**

1. **Install :** npm install nodemon
2. **Connect NodeJs with MongoDB :**

|  |
| --- |
| const { MongoClient } = require('mongodb');  **// dbName and collectionName are available in mongoDB**  **const dbName = "shopDB";**  **const collectionName = "productCollection";**  const url = "mongodb://127.0.0.1:27017";  **// Create Client**  const client = new MongoClient(url);  async function connectToMongo() {  **// Establish a connection between client created with mongoDB**      let create\_connection = await client.connect();  **// Extract Database from connection**      let database = create\_connection.db(**dbName**);  **// Extract Collection from database**      let collection = database.collection(**collectionName**)  **// Reading Data of the collections**      let readData = collection.find().toArray();  **// We need to handle readData as it is a promise need to handle by await method**      let readHandle = await readData;      console.log(readHandle);  }  connectToMongo(); |

1. **Import / Export With MongoDB & nodejs**

|  |  |
| --- | --- |
| **Export.js** | **Import.js** |
| const { MongoClient } = require("mongodb");  const url = "mongodb://127.0.0.1:27017";  const client = new MongoClient(url);  async function connectToMongo() {      const create\_connection = await client.connect();      let database = create\_connection.db('shopDB');      let collection = database.collection('productCollection');      return collection;  }  module.exports = connectToMongo; | const import\_connectToMongo = require(**'export file path’**);  async function main(){      const collection = await import\_connectToMongo();      const readData = collection.find().toArray();      const readHandle = await readData;      console.log(readHandle);  }  main(); |

1. **CRUD – Create , Read , Update , Delete. “operations” :**

|  |
| --- |
| **Config.js 🡪 Main File** |
| const { MongoClient } = require("mongodb");  const url = "mongodb://127.0.0.1:27017";  const client = new MongoClient(url);  async function connectToMongo() {      const create\_connection = await client.connect();      let database = create\_connection.db('shopDB');      let collection = database.collection('productCollection');      return collection;  }  module.exports = connectToMongo; |

|  |
| --- |
| 1. **Create or Insert Data To Database** |
| const connectToMongo = require('./config'**); 🡨 Import Config file**  let data\_to\_insert = {      name: "Insert product",      price: 100000,      category: "Insert"  }  async function main() {      let collection = await connectToMongo();  **let insertData = await collection.insertOne(data\_to\_insert);**  **<--- await is necessary as it return a promise needs to handle**      console.log(insertData.acknowledged ? "Data Inserted Successfully" : "Failed To Insert Data");  }  main();  Note : insertData.acknowledged 🡪 **It return a boolean value used to check the conditions as we have done in this code** |

|  |
| --- |
| 1. **Read Data From Database** |
| const connectToMongo = require('./config');  async function main(){      let collection = await connectToMongo();  **let readData = await collection.find().toArray(); <--- await is necessary as it return a promise needs to handle , you can fetch single data it as well like “ find({\_id:1}) ” in above code line.**      console.log(readData);  }  main(); |

|  |
| --- |
| 1. **Update Data In Database** |
| const connectToMongo = require('./config');  **// Following are the update you want to perform in database**  **let updates = {**  **name: "updated name",**  **price: "updated price",**  **category: "updated category"**  **}**  async function main() {      let collection = await connectToMongo();  **let updateData = await collection.updateOne(**  **{ \_id: 1 },                // <---- filter**  **{ $set: updates }          // <---- updates $set: is a atomic operator in mongodb**  **)**      console.log(updateData.acknowledged ? "Data Updated Successfully" : "Failed To Update Data");  }  main(); |
| 1. **Delete Data In Database** |
| const connectToMongo = require('./config');  async function main() {      let collection = await connectToMongo();  **let deleteData = await collection.deleteOne({ \_id: 2 }); 🡨 \_id is for filtering**      console.log(deleteData.acknowledged ? "Data Deleted Successfully" : “Failed To Delete Data");  }  main(); |

1. **Integrate MongoDB + ExpressJs in nodejs**

**API Methods :**

**1. GET**: Retrieve data from the server.

**2. POST:** Create a new resource on the server.

**3. PUT or PATCH:** Update an existing resource on the server.

**4. DELETE:** Remove a resource from the server.

**Examples :**

|  |
| --- |
| **Connect MongoDB + Express** |
| **Express Setup**  const express = require("express");  const app = express();  **Import config**  const connectToMongo = require("./config");  **1.GET**  **app.get**('/', async (request, response) => {      let collection = await connectToMongo();      let readData = await collection.find().toArray();      response.send(readData);  **// this is send to the server or webpage**  })    **2.POST**  **app.use(express.json()); // read requested data in json format server by a server**  **app.post**('/', async (request, response) => {      let data\_to\_insert = request.body; **// this is received from server dynamic data**      let collection = await connectToMongo();      let insertData = await collection.insertOne(data\_to\_insert);      console.log(insertData);      response.send(insertData);  })  **3. Put**  **app.use(express.json()); // read requested data in json format server by a server**  **app.put**('**/:pname'**, async (request, response) => {  **let updates = request.body; // getting updates from this server to perform operation on db.**      const collection = await connectToMongo();      const updateData = await collection.updateOne(          { name: **request.params.pname** },  **// getting name from webpage url**          { $set: updates }      );      response.send(updateData);  })  **4. Delete**  **app.delete**('**/:pname'**, async (request, response) => {      const collection = await connectToMongo();      const deleteData = await collection.deleteOne({ name: **request.params.pname** })      response.send(deleteData);  })  app.listen(1000, **() => { 🡨------------------------------- Callback function**  **console.log("server started at port 1000");**  **})** |

**TO Connect Backend With Frontend :** npm install cors

|  |
| --- |
| var express = require('express')  **var cors = require('cors')**  var app = express()  **app.use(cors())**  app.get('/products/:id', function (req, res, next) {  res.json({msg: 'This is CORS-enabled for all origins!'})  })  app.listen(80, function () {  console.log('CORS-enabled web server listening on port 80')  }) |

**Mongoose + Nodejs**

1. **Difference between MongoDB and Mongoose**

|  |  |
| --- | --- |
| **MongoDB** | **Mongoose** |
| * MongoDB is a NoSQL database management system. * It stores data in flexible, JSON-like BSON (Binary JSON) documents. * It supports a wide variety of data types and is schema-less. | * Mongoose is an ODM (Object-Document Mapping) library for MongoDB in Node.js. * It provides a higher-level, schema-based abstraction over the MongoDB driver. * **Install :** npm I mongoose |

1. **Connect To Mongoose in NodeJs**

|  |
| --- |
| const mongoose = require("mongoose");  **const dname = "shopDB";**  **const collectionname = "productCollection"**  async function dbConnect() {      let url = `mongodb://127.0.0.1:27017/${dname}`;      let connect = await mongoose.connect(url);  **Creating Schema 🡪 A Mongoose schema defines the shape of documents within a MongoDB collection. It specifies the fields, their data types, and any additional options such as default values, validation rules, and more. A schema is a blueprint for how documents should be structured in the database.**  **const ProductSchema = new mongoose.Schema({**  **name: String,   // Validation - 1**  **price: Number   // Validation - 2**  **});**  **Creating Model 🡪 A Mongoose model is a constructor function that takes a schema and creates an instance of a document that can be saved to the database.**  **const ProductModel = mongoose.model(collectionname, ProductSchema);**  **let data = new ProductModel({**  **name: "Fasssst Charger",**  **price: 1000**  address: "ABCD" <----------- It will not insert or stored to Database as it is not define in the schemas.  **});**      let store = await data.save()      console.log(store);  }  dbConnect(); |

1. **CRUD Operation :**

|  |
| --- |
| const mongoose = require('mongoose');  const dbName = "employeeDB";  const collectionName = "records";  **Creating Connection Only**  const url = `mongodb://127.0.0.1:27017/${dbName}`;  async function connectDB() {      const connect = await mongoose.connect(url);      connect ? console.log("Mongoose Server is Connected") : console.log("Failed To Connect");  }  connectDB();  **Define Schema Globally**  const EmployeeSchema = {      fname: String,      lname: String,      email: String,      phone: Number  }  **Operations : CRUD**  **1. Create or insert data to db**  async function insertDB() {      const EmployeeModel = mongoose.model(collectionName, EmployeeSchema);      let insertData = new EmployeeModel({          fname: "Mongoose Fname",          lname: "Mongoose lname",          email: "Mongoose email",          phone: 99999      })      let store = await insertData.save()      console.log(store);  }  insertDB();  **2. Read**  async function readDB() {      const EmployeeModel = mongoose.model(collectionName, EmployeeSchema);      let readData = await EmployeeModel.find();      console.log(readData);  }  readDB();  **3. Update**  async function updateDB() {      const EmployeeModel = mongoose.model(collectionName, EmployeeSchema);      let updates = {          fname: "Updated",          lname: "Updated",      }      let updateData = await EmployeeModel.updateOne(          { name: "Piyush" }, **🡨 filter**          { $set: updates } **🡨 update or projection**      );      console.log(updateData.acknowledged);  }  updateDB();  **4. Delete**  async function deleteDB() {      const EmployeeModel = mongoose.model(collectionName, EmployeeSchema);      let deleteData = await EmployeeModel.deleteOne({ fname: "Updated" })      console.log(deleteData.acknowledged);  }  deleteDB(); |

1. **API Methods**

|  |
| --- |
| **Config.js** |
| const mongoose = require("mongoose");  let url = "mongodb://127.0.0.1:27017/employeeDB";  mongoose.connect(url); |
| **Method.js 🡪 Schema & Method Creation and Method Export** |
| const mongoose = require("mongoose");  // creating Schemas  const EmployeeSchema = mongoose.Schema({      \_id: Number,      fname: String,      lname: String,      email: String,      salary: Number  });  // Creating Models  const EmployeeModel = mongoose.model('records', EmployeeSchema);  module.exports = EmployeeModel; |
| **Api.js** |
| const express = require('express');  const app = express();  **require('./config');**  **const import\_employee\_model = require('./methods');**  **1. Post**  app.use(express.json()) // convert data received from post to json format  app.post('/create', async (request, response) => {      console.log(request.body);      let insertData = new import\_employee\_model(request.body);      let store = await insertData.save();      console.log(store);      response.send(store);  })  app.listen(1000, () => {      console.log("server started at port 1000");  })  **2. Get**  app.get('/display', async (request, response) => {      let display = await import\_employee\_model.find()      console.log(display);      response.send(display);  })  **3. Delete**  app.delete('/delete/:\_id', async (request, response) => {      let deleteData = await import\_employee\_model.deleteOne(request.params);      response.send(deleteData)  })  **4. Put**  app.put('/put/:\_id', async (request, response) => {      let updateData = await import\_employee\_model.updateOne(          request.params,          { $set: request.body }      )      response.send(updateData);  }) |

**Upload Files Using Multer in nodejs**

* **Multer :** Multer is a node.js middleware for handling **multipart/form-data**, which is **primarily used for uploading files.** It is written on top of [busboy](https://github.com/mscdex/busboy) for maximum efficiency.
* **Install :** npm install multer

|  |
| --- |
| const express = require("express");  const app = express();  **const multer = require("multer");**  **const store\_location = "uploads"**  **const uploadMiddleware = multer({**  **storage: multer.diskStorage({**  **destination: function (request, file, callback) {**  **callback(null, store\_location);**  **},**  **filename: function (request, file, callback) {**  **callback(null, file.originalname + Date.now() + ".jpg");** 🡨 (null,filename)  **}**  **})**  **}).single("image");** 🡨 <input type="file" name="image" >  app.post('/upload', **uploadMiddleware**, (request, response) => {      response.send("Post Upload");  })  app.listen(1000, () => {      console.log("server started at port 1000");  }) |

**IMP TOPIC**

* **Event:**

1. An event is a signal that something has happened.
2. Events can be triggered by various actions or occurrences within a program.
3. Examples of events include user interactions (clicking a button), file system changes, network activities, etc.

* **EventEmitter:**

1. EventEmitter is a class provided by the Node.js events module.
2. It allows you to implement the observer pattern, where an object (the EventEmitter) maintains a list of dependent observers (listeners) that are notified of state changes.
3. Instances of the EventEmitter class can emit named events. Other parts of your application can then listen for these events and respond accordingly.

|  |
| --- |
| const express = require("express");  const app = express();  **Event & Event Emitter Setup ------------------------**  **const EventEmitter = require("events");**  **const event = new EventEmitter();**  **let count = 1;**  **event.on("APIcalled", () => {**  **console.log("Hello Event", count);**  **count++;**  **})**  app.get('/', (request, response) => {      response.send("API Called");  **event.emit("APIcalled")**   🡨 his will generate an event for this route.  })  app.get('/about', (request, response) => {      response.send("About API Called");  })  app.get('/contact', (request, response) => {      response.send("Contact API Called");  **event.emit("APIcalled")**   🡨 his will generate an event for this route.  })  app.listen(1000, () => {      console.log("server Started at port 1000");  });  **Postman :** [http://localhost:1000/](http://localhost:1000/contact) OR <http://localhost:1000/contact/> |

**MySQL + Nodejs**

**Install :** npm install mysql

1. **Connect To MySQL :**

|  |
| --- |
| const mysql = require("mysql");  **const new\_connection = mysql.createConnection({**  **host: 'localhost',**  **user: 'root',         <---- username**  **password: '',         <---- default password of mysql**  **database: 'employee'   <---- Database name**  **});**  new\_connection.connect((error)=>{      if(error){          console.log("Error : ",error);      }      else{          console.log("Connected To MySQL");      }  }) |

1. **API Methods :**

|  |
| --- |
| **Config.js** |
| const mysql = require("mysql");  const new\_connection = mysql.createConnection({      host: 'localhost',      user: 'root',         // <---- username      password: '',         // <---- default password of mysql      database: 'employee'  // <---- Database name  });  new\_connection.connect((error) => {      if (error) {          console.log(error);      }      else {          console.log("Connected To MySQL");      }  })  module.exports = new\_connection; |
| 1. **GET API**   const express = require("express");  const app = express()  // connections  const importConnection = require("./config");  app.get('/', (request, response) => {      importConnection.query("select \* from records", (error, result) => {          result ? response.send(result) : "Error To Fetch Data"  // ternary operator      })  })  app.listen(1000, () => {      console.log("Server Started at port 1000");  }) |
| 1. **POST API**   const express = require("express");  const app = express()  // connections  const new\_connection = require("./config");  app.use(express.json());  //  <-- read data to json from request  app.post('/', (request, response) => {      /\*      const insertData = {          fname: "Herik",          lname: "Sevak",          email: "Herik@gmail.com"      }      \*/      const insertData = request.body;      console.log(insertData);      new\_connection.query("INSERT INTO records SET ?", insertData, (error, result, fields) => {          if (error) {              console.log("Error Occurs Failed to post data");          }          else {              response.send(result);          }      })  })  app.listen(1000, () => {      console.log("Server Started at port 1000");  }) |
| 1. **PUT API**   const express = require("express");  const app = express()  // connections  const importConnection = require("./config");  app.use(express.json());  //  <-- read data to json from request  app.put('/:id', (request, response) => {      let updateData = [request.body.fname, request.body.lname, request.body.email, request.params.id];      console.log(updateData);      importConnection.query("UPDATE records SET fname = ?, lname = ?, email = ? where id = ?", updateData, (error, results, fields) => {    // (query,update data list,callback function)          if (error) {              console.log("Error Occur Failed To Change");          }          else {              response.send(results);          }      });  })  app.listen(1000, () => {      console.log("Server Started at port 1000");  }) |
| 1. **DELETE API**   const express = require("express");  const app = express()  // connections  const importConnection = require("./config");  app.use(express.json());  //  <-- read data to json from request  app.delete('/:id', (request, response) => {      importConnection.query("DELETE FROM records WHERE id =" + request.params.id, (error, results, fields) => {          if (error) {              console.log("Failed To Delete Data");          }          else {              response.send(results)          }      });  })  app.listen(1000, () => {      console.log("Server Started at port 1000");  }) |