**Q. What is a node JS?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Node js is not language basically it is a server environment means we can say node js provide server environment for JavaScript as well as using node we can communicate with database and it is open source and using node js we can create API and call at client side and node use the chrome v8 engine to execute.

**2. Features of Node.js**

* **Asynchronous and Event-Driven**: All APIs are non-blocking.
* **Fast Execution**: Powered by the V8 engine.
* **Single Threaded but Highly Scalable**: Uses an event loop for concurrency.
* **No Buffering**: Streams data in chunks.
* **Open Source** and supported by a huge community

**Q. Are JavaScript and Node the same?**

* JavaScript and Node syntax are same
* If you know JavaScript you can easily understand Node..
* Both are exactly not same
* Node can connect with a database
* Node create server environment but JavaScript create client side environment
* If we want to work with node js we need to know the following things

**1. What is a server?**

Server is an application /software which is responsible for execute application  at central location and provide access of application to remote clients as well as accept the request from client and process on it may be communicate with database also and generate proper response according to client request and send response to client.

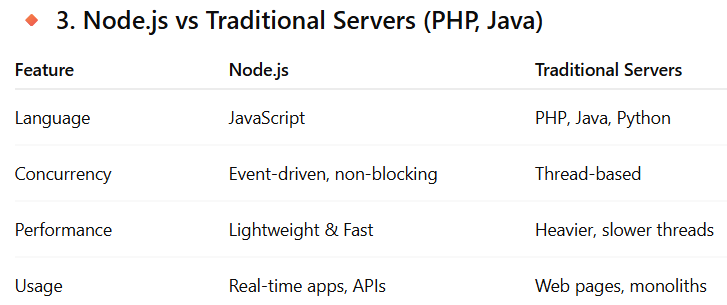
**2. What is a client?**

Client is also an application which is responsible to send requests to server applications by using hyperlink or form and accept the response generated by server and show to the end users.

**Note :** if we think about web application browser act as client software

**What do developers make with Node JS?**

* Developer make API with Node JS
* Node js can connector or provide communication between client and server
* Node JS Can communicate with database
* Node JS create API and integrate with web application /mobile application.

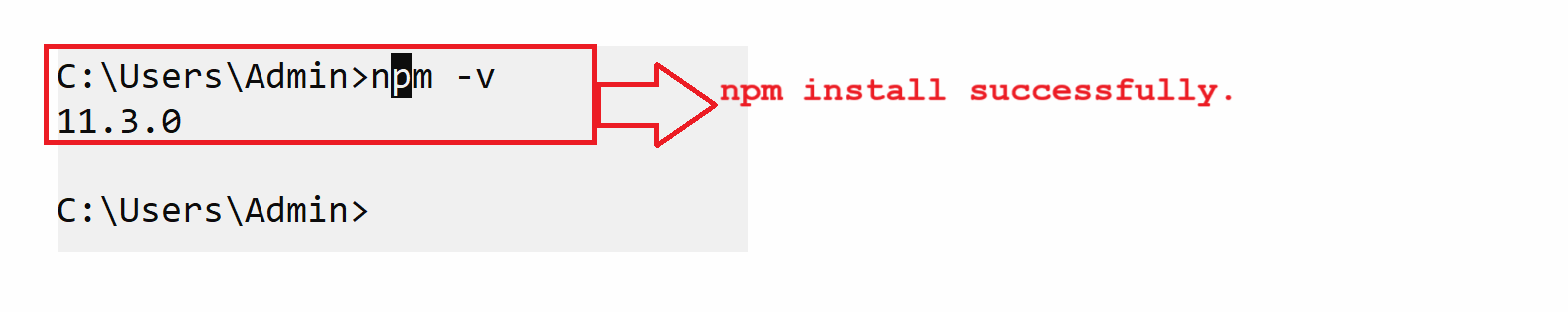


**How to  work with Node JS?**

**1. Download and install the node js :** <https://nodejs.org/en/download>

**Q. What is NPM :** NPM stands for Node Package Manager which is used for downloading the external libraries required for node applications like maven in java.

****

****

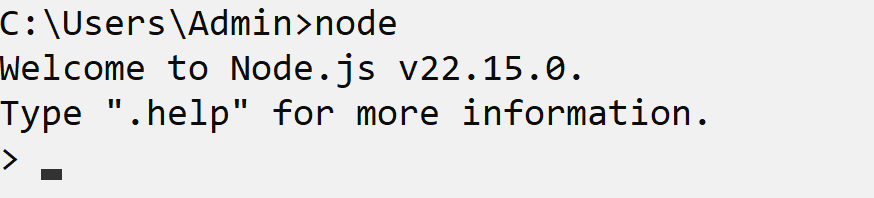
**How to create application by using node :**

If we want to create application using node we have two ways

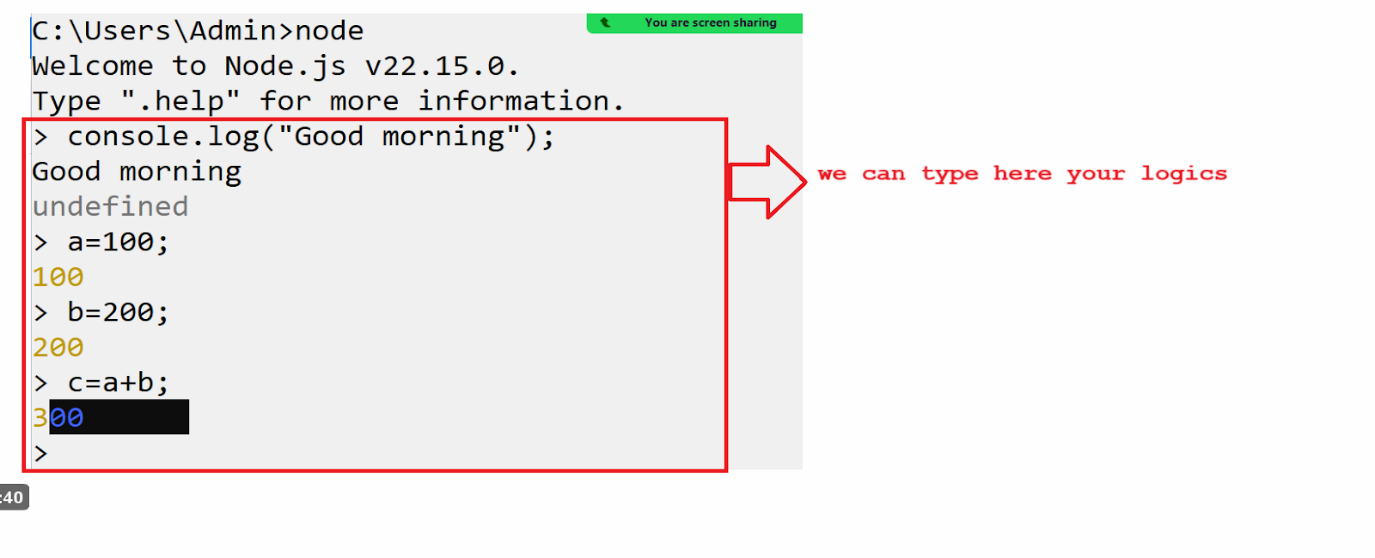
* **Script with command line**
* **Make folder and files**

**A] Script with command line :** **Steps -**

**1. Open command prompt and type command node on it.**



**2.** **We can write code on command prompt.**



**Note :** this is not standard way to work with node , We have one more way to work with **node :** use any text editor and write code in text editor save file using .js extension and save and run using node.

**B] Make folder and files : Steps –**

* Write javascript code using any text editor
* Save file using .js extension
* Run the file by using node environment

**Core Module In Node JS**

**Q. What are the core modules?**

In every programming language there is some basic feature like as database connection feature, file creation feature , API calling feature as well as to process the process the code called as core module

**Example :** FS , Buffer , Http etc.

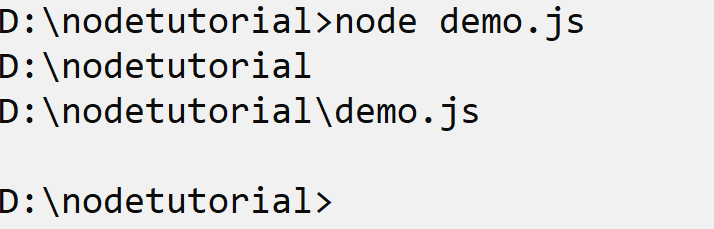
**There are two types of core module in Node**

1. **Global Module :** global module means module not needed to import in an application called as global module.

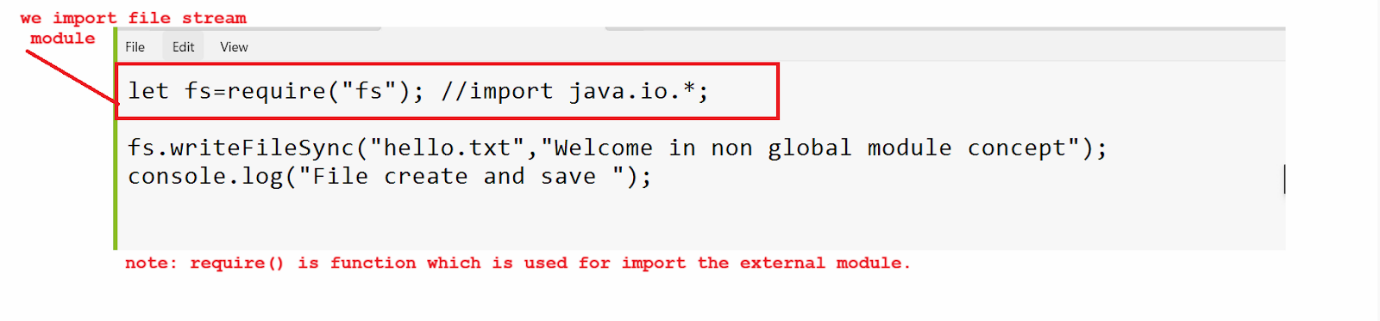
**Example:** console.log(“good morning”);

****

**Output**

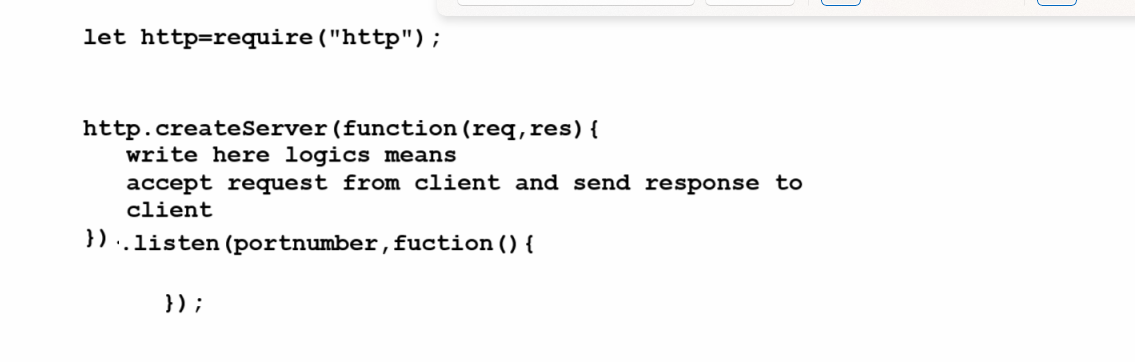
****

**2. Non Global module :** non global module means need to import in application called as non global module



**Now we want to discuss about non global http module**

Http module help us to create server means http module has one function name as createServer() which is used for create server and this function contain call back as parameter and callback function contain two parameters req , res and when we create we required to use one method name as listen() and this method is used for provide port number to server .

****

**Example:**

let http=require("http");

http.createServer((req,res)=>{

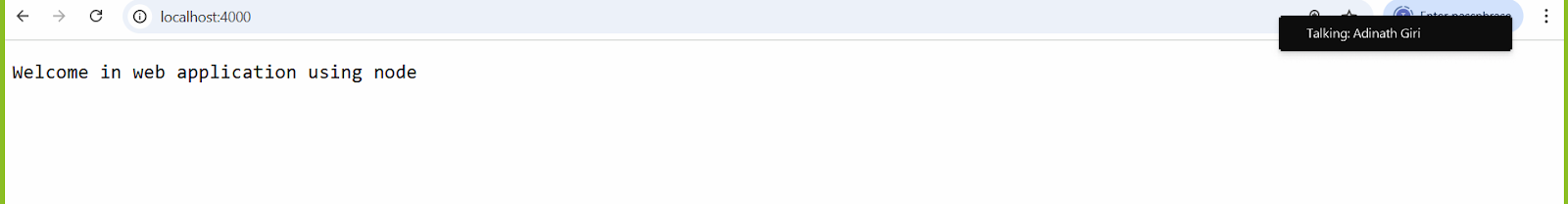
    res.write("Welcome in web application using node");

    res.end();

}).listen(4000,function(){

    console.log("server started");

});

**Output : **

**Q. What is the port number & Why use it?**

Port number is unique identity to server means if we want to start or access any server we require a unique identity number to server known as port number.

Means two server cannot work on same port number. Because single computer can have more than one server and so if we think about network then if we want to access any machine or server machine then we can access using IP but if we want to access particular server from that machine then we required access it  using port number.

**Q. What is the request and response parameter?**

Request means when we visit any web page or click on hyperlink or send data via hyperlink using browser or submit form using browser to server page known as request

If we think about createServer() method it contain one parameter name as req which help us to access the data send by browser and process on it

**Example :** send database also.

**Response:** response means when the server page sends data to the browser or displays data on the browser called as response.

**Example :** res.write(“welcome in web application using node”);

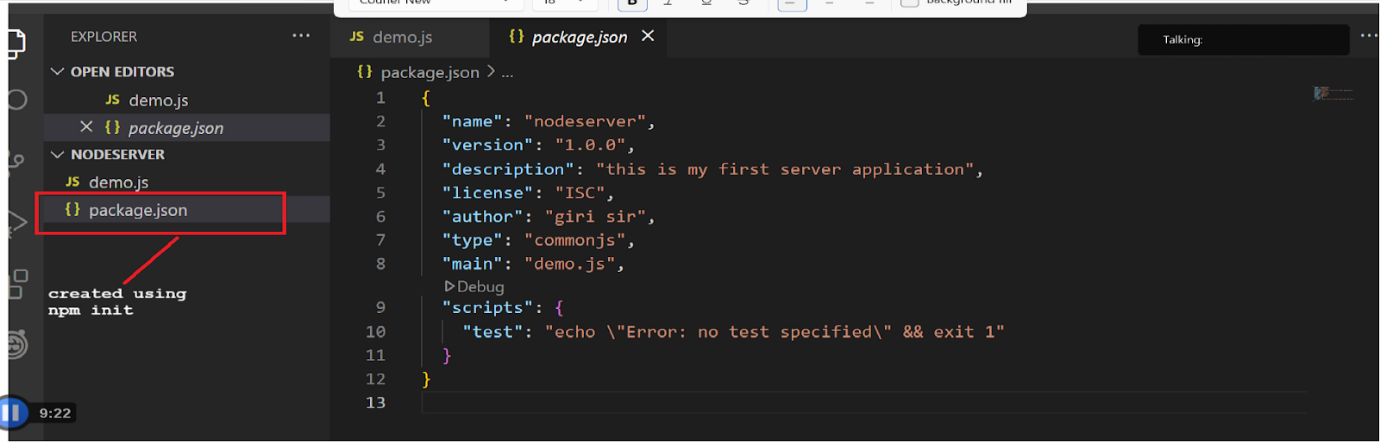
This message display on web browser so it response data send by node server to browser

**What is package.json?**

Package.json file hold the all detail about project like as project name , project version , repository name, command used in project , installed packages like as mongodb ,logging etc.

**How to create package.json file in project**

**Syntax :** npm init

****

When we work with node js we required to create one more folder in project name as node\_module

Node\_module is folder where all libraries present or when we install any library then

Library install under node\_module folder.

If we want to create node\_module folder project we have command

**Example :** we want to create a project to display output text in color.

**Step -**

1. **Create folder for project**
2. **Create package.json file**

**Syntax :** npm init

**3. Create package-lock.json**

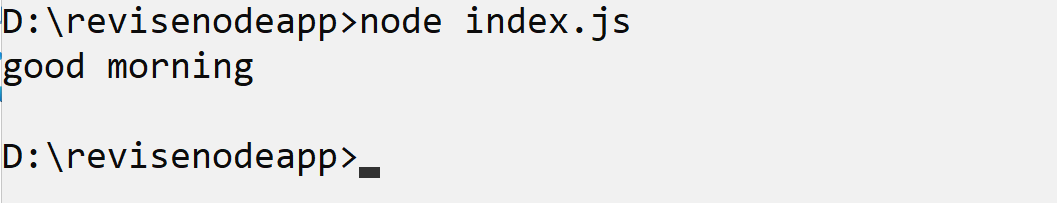
**Syntax :** npm install

**4.** **Create index.js file and write  following logic**s

**index.js**

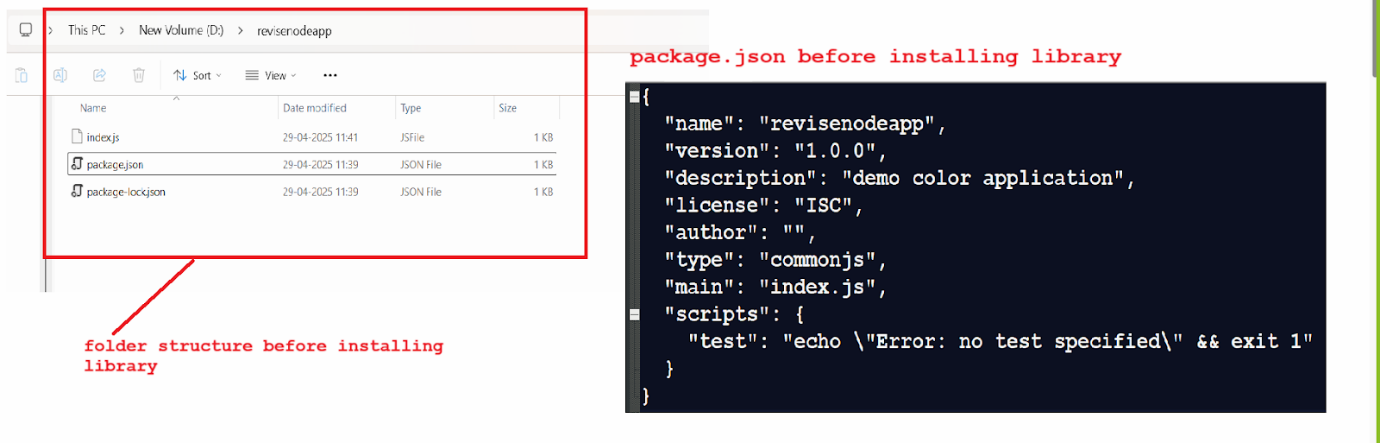
console.log("good morning");

**Output** -

****

If we think about above output we get output text in black color we want to display this output text in red color , So if we want to show output text in different colours we have one dependency or library provided node and we are required to install this library in the project  and the name of the library is color.

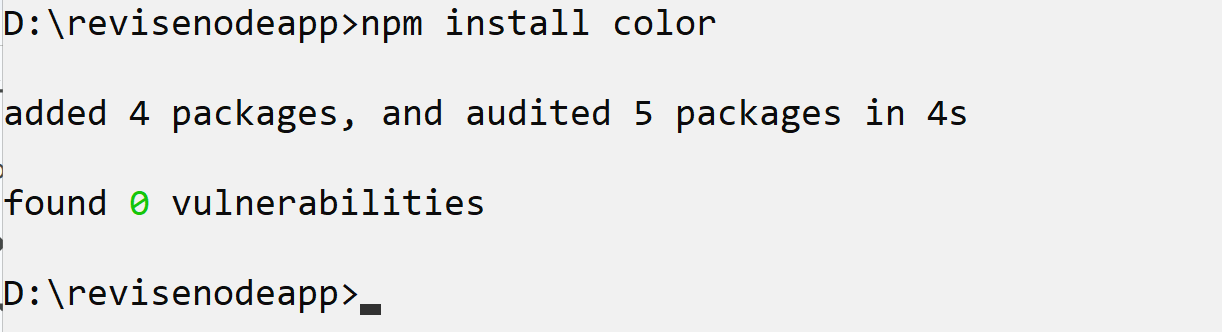
**Note :** check the folder structure and package.json file before installing library



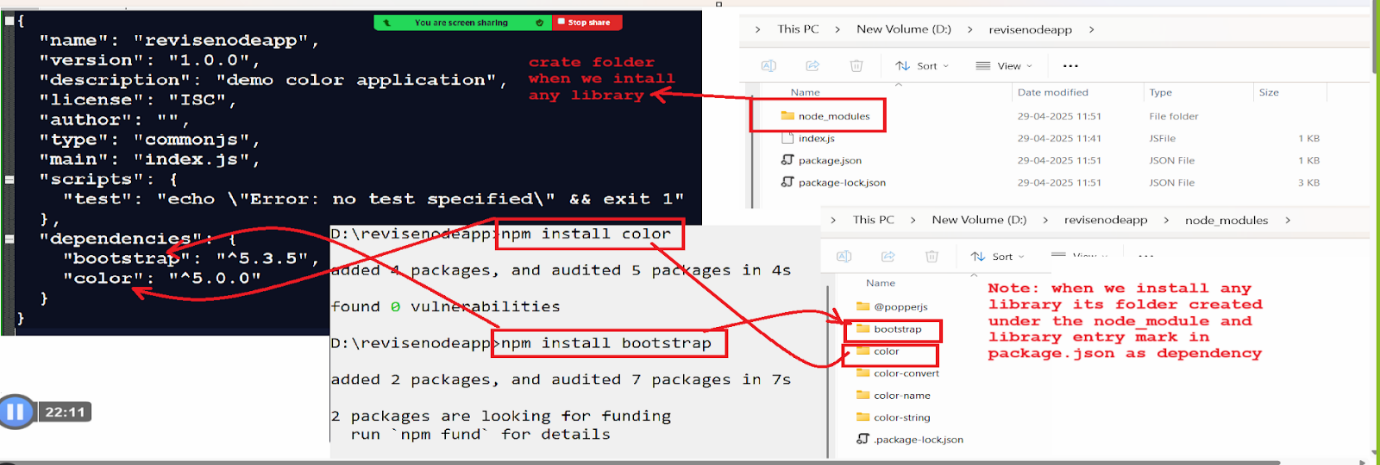
**Note :** if we want to install any dependency required for project we have generalize command or syntax

**Syntax:** npm install dependency name; **Or** npm i dependency name;

**So if we want to display output text in color we have color dependency**

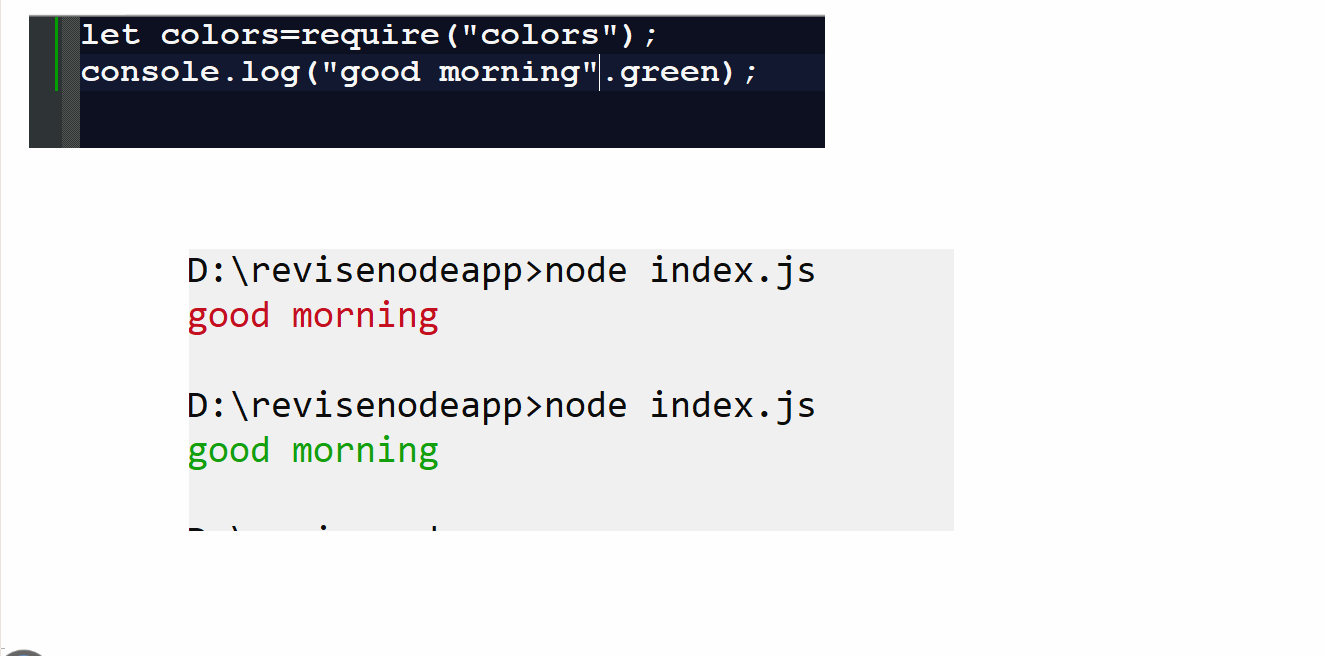
****

**Check folder and package.json file after installing library**

****

**Note :** package.json maintain the record of installed libraries because when we push project on give we not need to push node\_module just we can push project

Without node\_module and push package.json so when pull project anywhere and run command npm install then node\_module created automatically with dependencies whose record maintain in package.json file



**Express JS**

**Q. What is express JS?**

Express JS a framework of node which is used for web application development purposes and mobile applications.  It simplifies the development of server side applications by offering easy to use API for routing middleware , http utilities

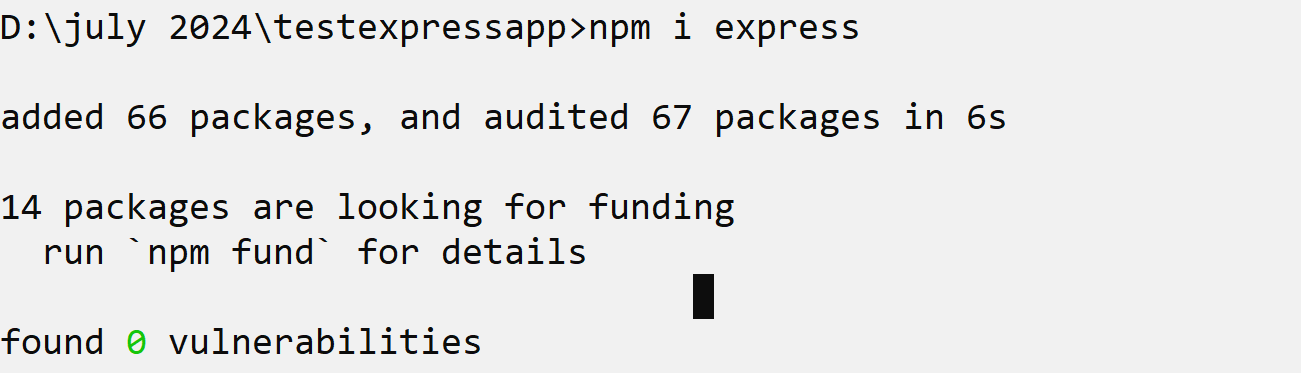
**Q. What kind of task can we develop using express JS?**

1. Building RESTFUL API’S
2. Creating Single page , multi page and hybrid applications
3. Developing Server side logics for web application and mobile applications
4. Handing routing and middleware
5. Integrate view and rendering engines
6. Adding request processing middleware

**How to use express practically using node environment**

**Steps.**

**1. Install express library**

****

**2. Import express library in application**

****

**3. Use standard http methods to work with web applications to handle request and response.**

**get() :** get() method is used for access resources from a server or read resources from a server and send data to client as response. Get method normally use in url rewriting technique or use using path variable working rest API.

**post() :** post() method is used for creating resources at server side

**Example :** suppose we have a form at client side like name, email, contact and we want to store it in the database table then we can use the post method at server side.

**put() :** if we want to update or modify the resource at server side then we have one standard resource at server side name as put.

**delete() :** this method is used for delete resources from a server side.

**patch():** this method is used for updating partial resources at the server's side.

**Example :** we want to create web page at server side name as home and display the message.

let express=require("express");

let app=express();

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

    res.send("welcome on first web page using express");

});

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

  res.send("I am about page");

});

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

res.send("I am service page");

});

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

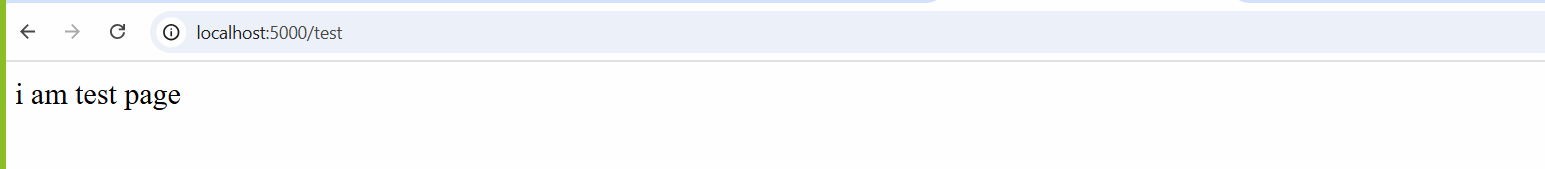
  res.send("i am test page");

});

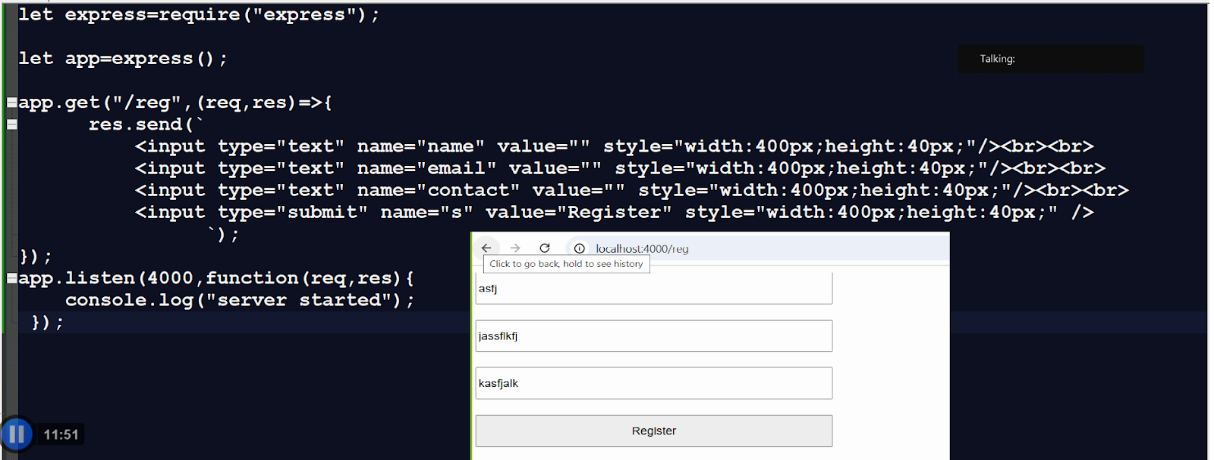
app.listen(5000,()=>{

  console.log("Server Started");

});



**Note:** using express JS we can generate the HTML content or html controls on a web page.

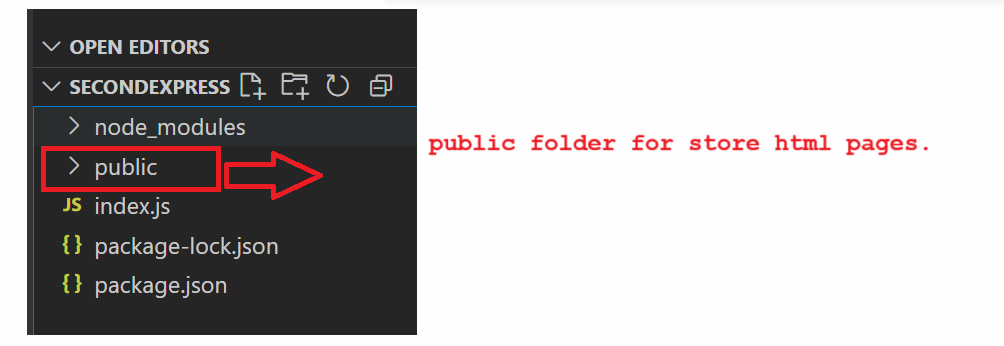


**Note:** if we think about above code we  design UI using express JS but it is not standard way to design UI so we want to design ui pages using html or ejs and call it using express JS

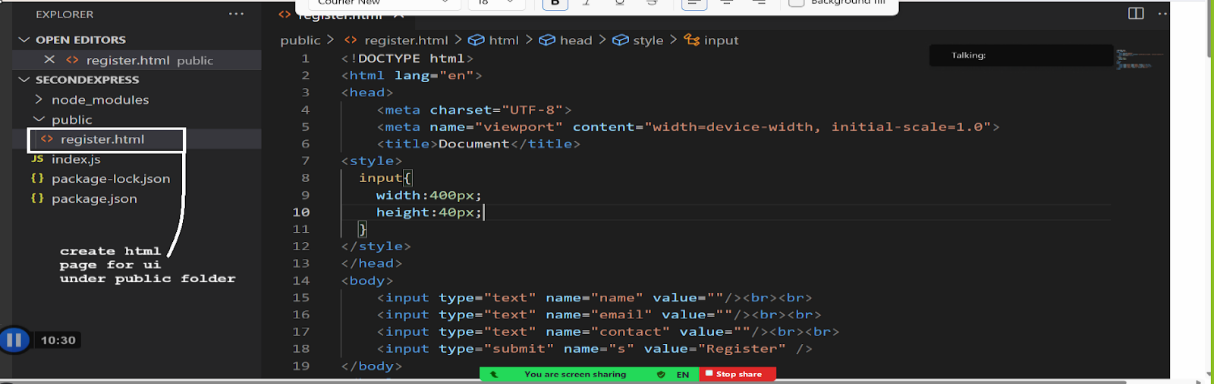
**How to create web page call it using express JS**

**Steps :**

1. **Create public folder in project**



1. **Create html pages under the public folder**

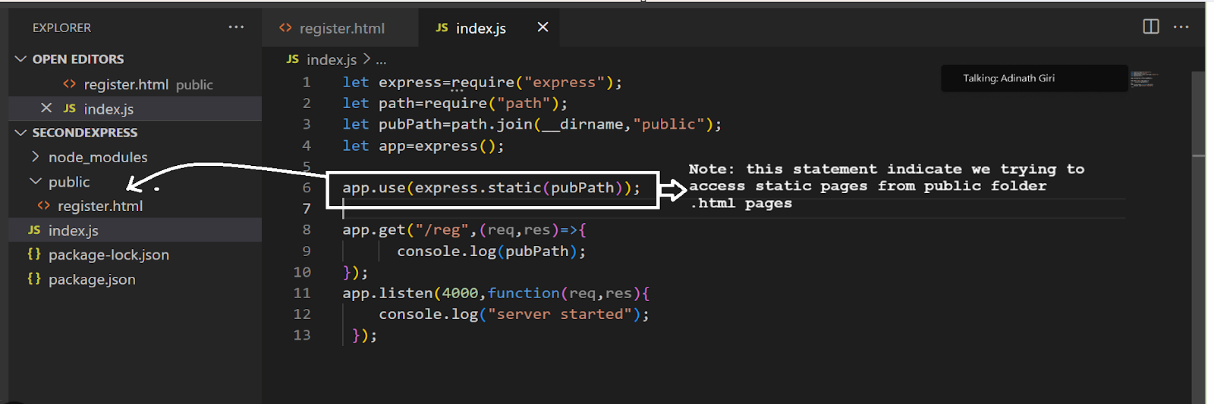


1. Access path of public folder in express js code or js file



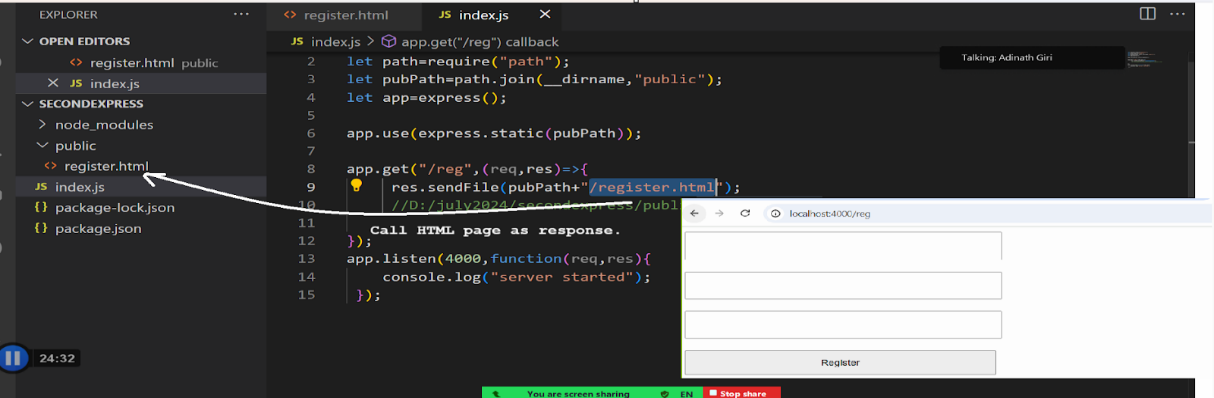
1. Access static pages or take permission to access statics or html pages in express

Note: if we want to access static resources we have use() method of express and static method of express



**5. Call HTML page as response from express**

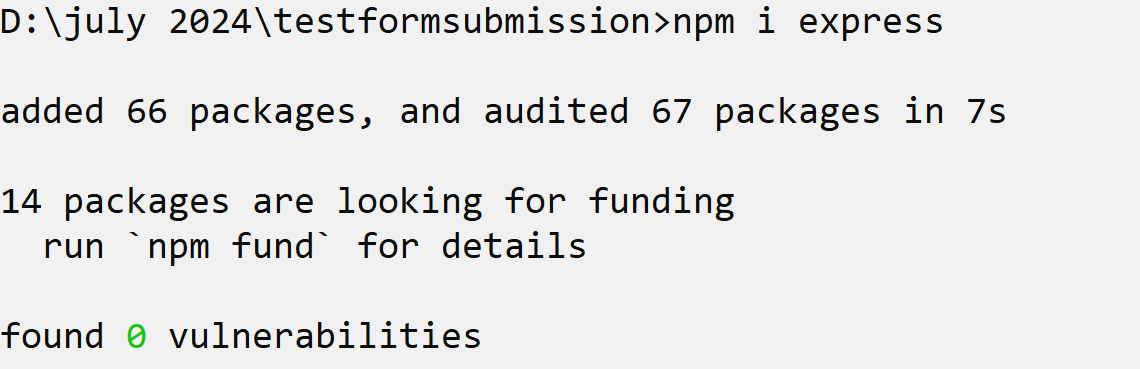
If we want to call html page as response from express we have method of response object known as sendFile()



**How to submit HTML form request to server page or express page**

**Step1:** create project and generate package.json and package-lock.json file

**Step2: install express library**

****

**Step3: create a public folder under the project and design the html page and save it.**

**Style.css**

input{ width:400px; height:40px; }

**Register.html**

<html>

 <head>

  <title>registration form</title>

  <link rel="stylesheet" href="style.css" />

 </head>

 <body>

   <input type='text' name='name' value=''/>

   <input type='text' name='email' value=''/>

   <input type='text' name='contact' value=''/>

   <input type='submit' name='s' value='Register'/>

 </body>

</html>

**4. Create index.js file or .js call html page as response to browser or client**

**index.js**

let express=require("express");

let path=require("path");

let app=express();

let pubPath=path.join(\_\_dirname,"public");

app.use(express.static(pubPath));

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

    res.sendFile(pubPath+"/register.html");

});

app.listen(4000,()=>{

   console.log("server started");

});

**5. Submit to server page or express url**

If we want to submit html form to server page means we going send to request to server page from client page or client side page and for that we have to use form tag in HTML

**Syntax:**

<form name=’formname’ action=’serverurl’ method=’GET/POST’  enctype=’application/x-www-urlencoded or multipart/form-data’>

</form>

**<form> :** normally help us to submit html form data to server as request

**name :** name is identity of form at server side

**action :** a server page url where we want to send request or we want to accept data as request by server

**method :** method means way to submit form to server

**GET :** when we submit form using get method then form data sent via URL of web page in the  form of name and value pair.

When we send data via URL the technical name is query string.

**POST :** when we submit a form using the post method then form data not sent via URL address bar or query string.

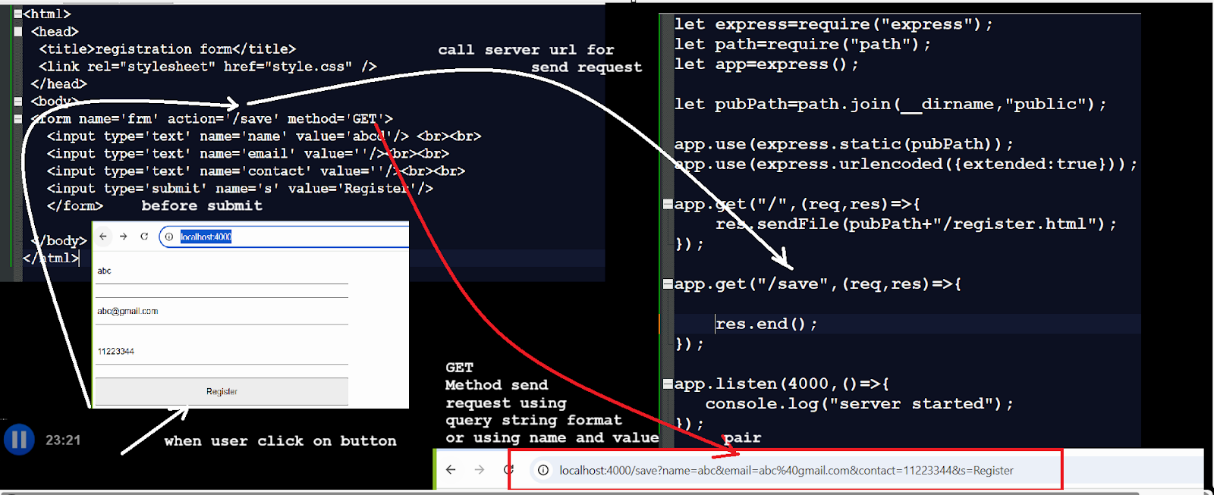
Form data sent via the body of the page means we can say the post method is more secure than the GET method.

**enctype=’application/x-www-urlencoded or multipart/form-data’: this parameter decide how form data should access at server side**

**Note :** application/x-www-urlencoded is default enctype means when we not specify enctype then form data by default application/x-url-encoded and this enctype decide access html form control name and return its value means when we submit request using name and value pair then we can use application/x-www-urlencoded we not specify it because it is default.

We have one more enctype known as multipart/form-data : this enctype we will use when we want to upload file to server

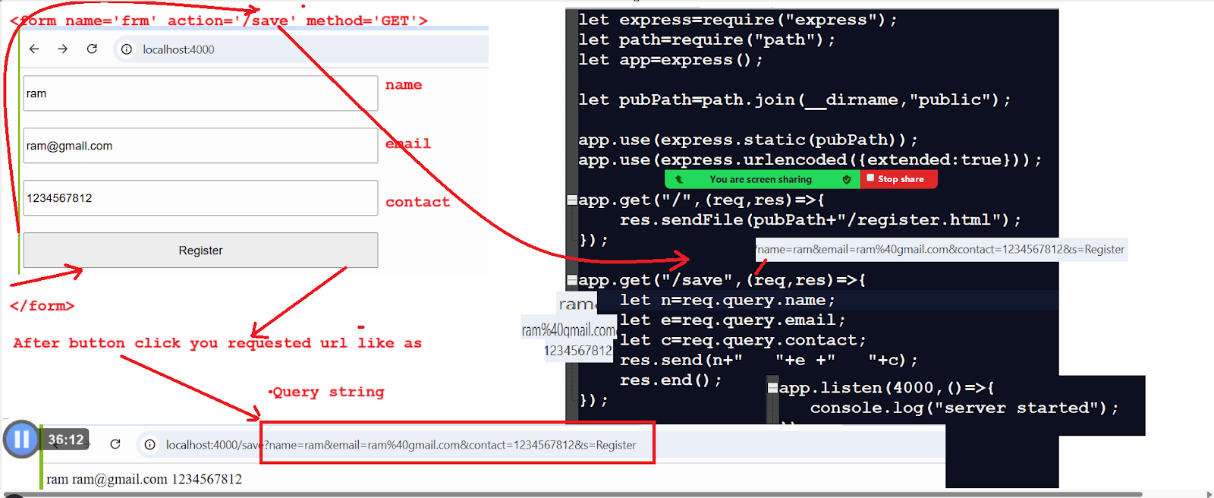
**Example of form submission using GET**

****

**Note:** when we submit form to server page or express page then we can access form data in express url function using req parameter

And for that we can use following syntax

**Syntax:** let variablename = req.query.requestparamname



**Style.css**

input{ width:400px; height:40px; }

**register.html**

<html>

 <head>

  <title>registration form</title>

  <link rel="stylesheet" href="style.css" />

 </head>

 <body>

 <form name='frm' action='/save' method='GET'>

   <input type='text' name='name' value='abcd'/> <br><br>

   <input type='text' name='email' value=''/><br><br>

   <input type='text' name='contact' value=''/><br><br>

   <input type='submit' name='s' value='Register'/>

   </form>

 </body>

</html>

**index.js**

let express=require("express");

let path=require("path");

let app=express();

let pubPath=path.join(\_\_dirname,"public");

app.use(express.static(pubPath));

app.use(express.urlencoded({extended:true}));

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

    res.sendFile(pubPath+"/register.html");

});

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

let n=req.query.name;

let e=req.query.email;

let c=req.query.contact;

res.send(n+"   "+e +"   "+c);

    res.end();

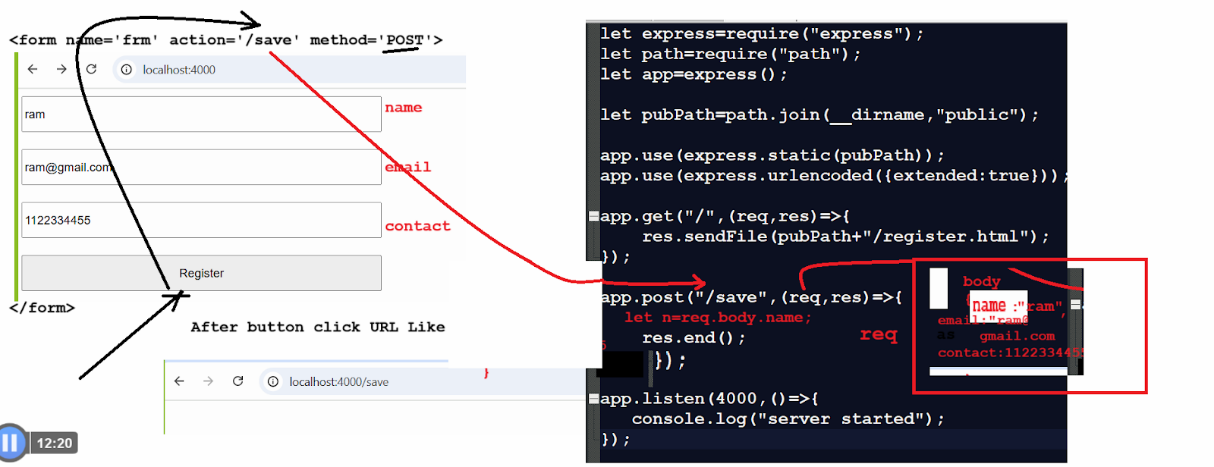
});

app.listen(4000,()=>{

   console.log("server started");

});

**Example : form submission using post method**



**Style.css**

input{  width:400px; height:40px; }

**Register.html**

<html>

 <head>

  <title>registration form</title>

  <link rel="stylesheet" href="style.css" />

 </head>

 <body>

 <form name='frm' action='/save' method='POST'>

   <input type='text' name='name' value='abcd'/> <br><br>

   <input type='text' name='email' value=''/><br><br>

   <input type='text' name='contact' value=''/><br><br>

   <input type='submit' name='s' value='Register'/>

   </form>

 </body>

</html>

**index.js**

let express=require("express");

let path=require("path");

let app=express();

let pubPath=path.join(\_\_dirname,"public");

app.use(express.static(pubPath));

app.use(express.urlencoded({extended:true}));

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

    res.sendFile(pubPath+"/register.html");

});

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

   let n=req.body.name;

   let e=req.body.email;

   let c=req.body.contact;

   res.send(n+"      "+e+"   "+c);

    res.end();

});

app.listen(4000,()=>{

   console.log("server started"); });

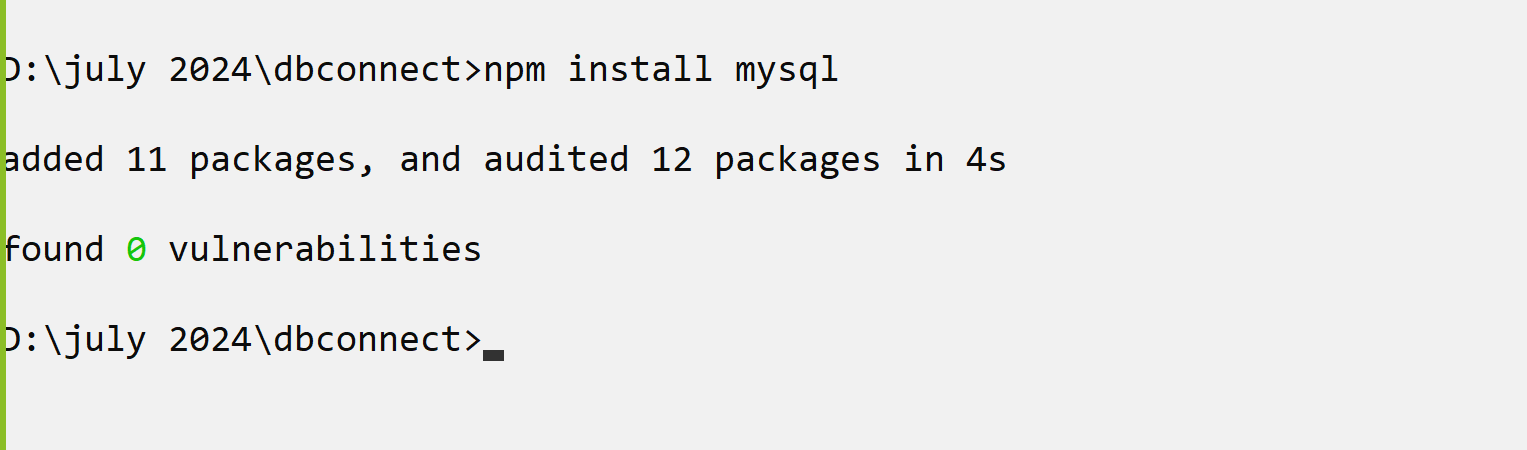
**How to connect node with mysql**

If we want to connect node with MYSQL we have some important steps.

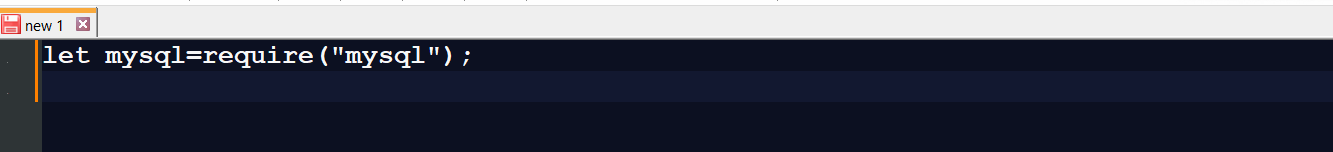
**1. Install mysql library or dependencies**

If we want to install dependency we have statement

**Syntax** : npm i mysql



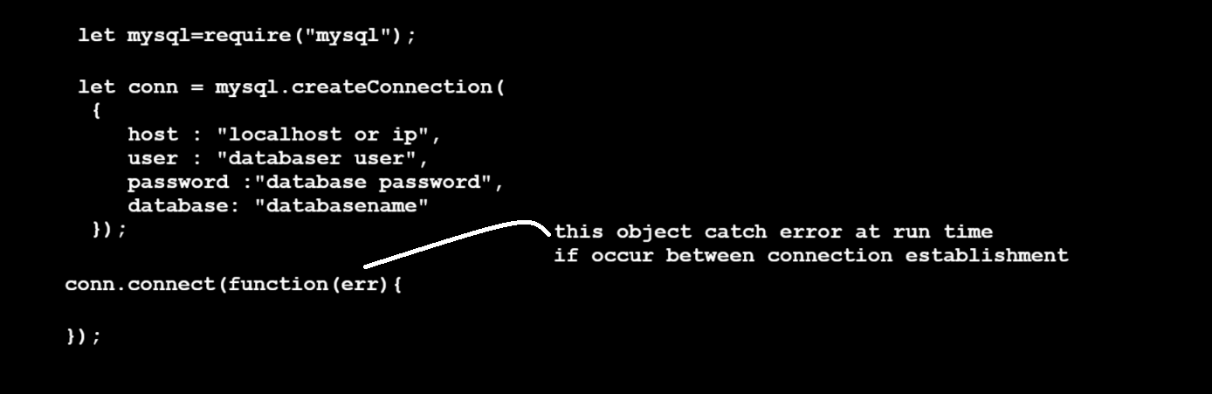
**2. import mysql dependency in application**

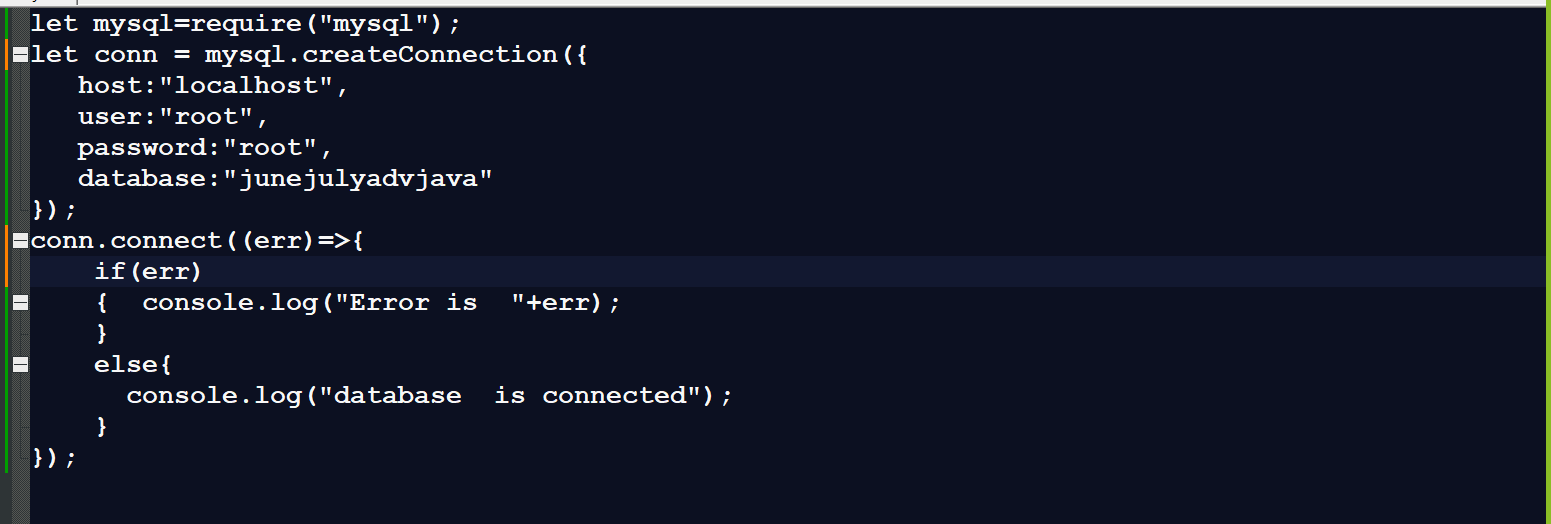


**3. Establish the connection with database**

If we want to establish connection between node environment and mysql database we have function name as **createConnection().** And this function accept the JavaScript object as parameter which contain host name, username of database, password of database and database name shown in following screen shot.

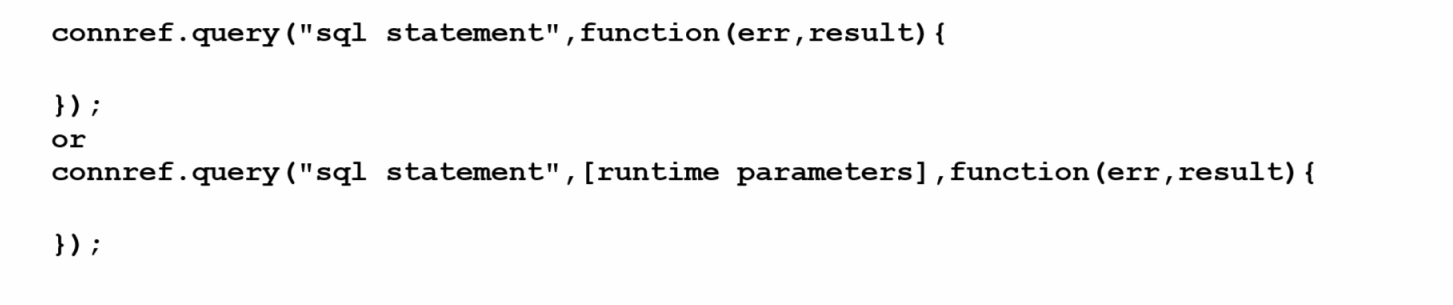
If we want to check your connection is established or not we have connect() function provide by connection object or createConnection() method reference.





Once we establish the connection between node and mysql we can work with mysql database or perform SQL operation like as  insert/delete/update/select etc.

So if we want to perform any operation on database we have query() function provided by conn object.



**Example :**



**How to Fetch data from database :**

**Example: index.js**

let mysql=require("mysql");

let conn = mysql.createConnection({

   host:"localhost",

   user:"root",

   password:"root",

   database:"junejulyadvjava"

});

conn.connect((err)=>{

    if(err){

console.log("Error is  "+err); }

else{

console.log("database  is connected");

}

});

conn.query("select \*from player",function(err,result){

if(err){

console.log("Operation Failed "+err);

}else{

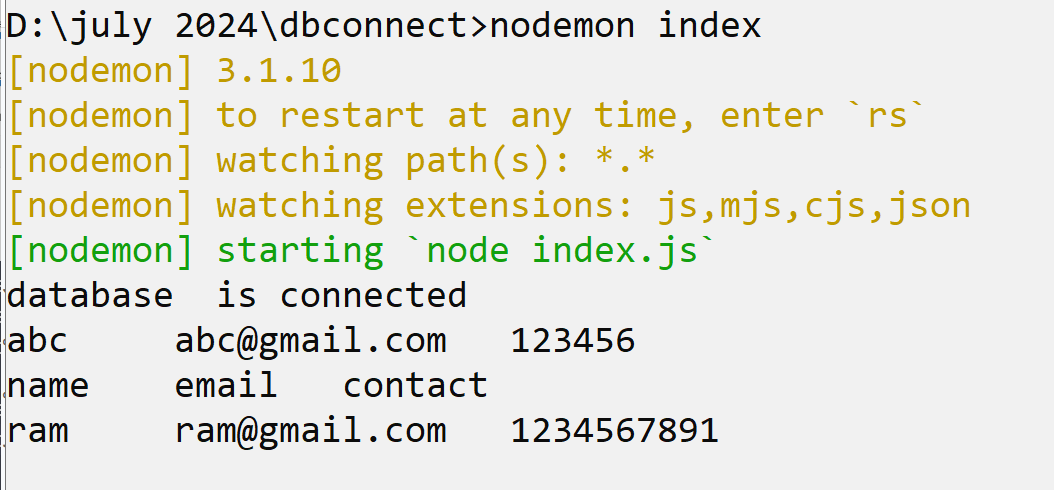
        for(var i=0; i<result.length;i++){

    console.log(result[i].name+"\t"+result[i].email+"\t"+result[i].contact);

  } }

});

**Output :**



**How to insert record in database table using node**

let mysql=require("mysql");

let conn = mysql.createConnection({

   host:"localhost",

   user:"root",

   password:"root",

   database:"junejulyadvjava"

});

conn.connect((err)=>{

    if(err)

{  console.log("Error is  "+err); }

else{ console.log("database  is connected"); }

});

conn.query("insert into player values('abcd','abcd@gmail.com','444444')",function(err,result){

if(err){ console.log("Operation Failed "+err); }

else {console.log("Record Save"+ result); }

});

**How to pass run time parameter to SQL Query using Node**

let mysql=require("mysql");

let conn = mysql.createConnection({

   host:"localhost",

   user:"root",

   password:"root",

   database:"junejulyadvjava"

});

conn.connect((err)=>{

    if(err) {  console.log("Error is  "+err); }

else { console.log("database  is connected"); }

});

let pname="rajesh";

let pemail="rajesh@gmail.com";

let pcontact="4455667788";

conn.query("insert into player values(?,?,?)",[pname,pemail,pcontact],function(err,result){

if(err){ console.log("Operation Failed "+err); }

else { console.log("Record Save"+result); }

});

**How to delete record from database table**

let mysql=require("mysql");

let conn = mysql.createConnection({

   host:"localhost",

   user:"root",

   password:"root",

   database:"junejulyadvjava"

});

conn.connect((err)=>{

    if(err) {  console.log("Error is  "+err); }

else{ console.log("database  is connected"); }

});

let pemail="abc@gmail.com";

conn.query("delete from player where email=?",[pemail],function(err, result){

if(err) { console.log("Operation Failed "+err); }

else {console.log("Record deleted"+result); } });

**How to update record in database table**

let mysql=require("mysql");

let conn = mysql.createConnection({

   host:"localhost",

   user:"root",

   password:"root",

   database:"junejulyadvjava"

});

conn.connect((err)=>{

    if(err) {  console.log("Error is  "+err);

}else{ console.log("database  is connected"); }

});

let pname="rama";

let pemail="ram@gmail.com";

let pcontact="8888888888";

conn.query("update player set name=?,contact=? where  email=?",[pname,pcontact,pemail],function(err,result){

if(err) { console.log("Operation Failed "+err);

} else { console.log("Record Updated "+result); }

});

**EJS in express JS**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EJS stands for Embedded JavaScript templating. Basically it is a template  engine used by node js and EJS specially design for generate dynamic web page using HTML means using ejs file or ejs template engine we accept data send by server via express at client side and can generate dynamic view to end user

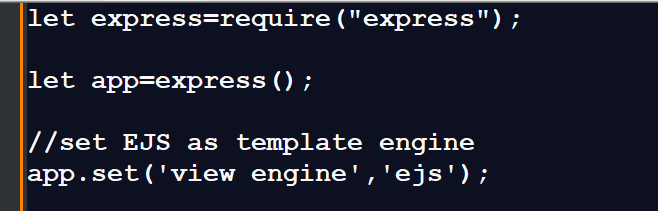
**Steps to work with EJS :**

**1. Install EJS library or dependency  -**

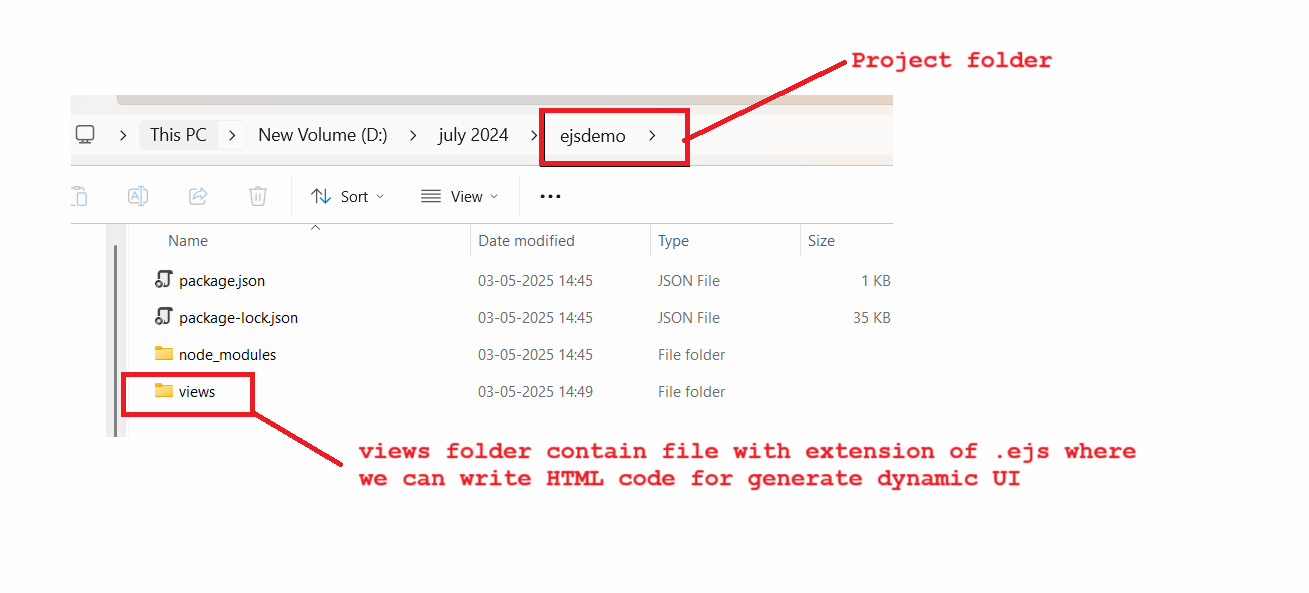
**Syntax:** npm i ejs --save  or npm i express ejs  --save



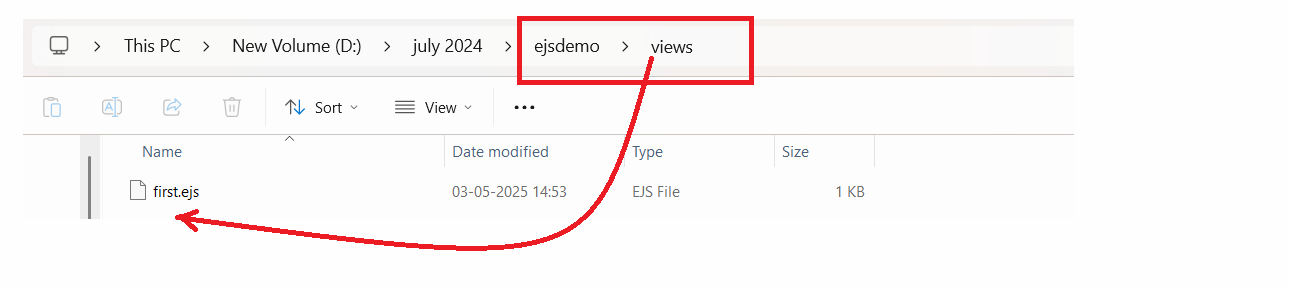
**2. Create .js file and set ejs engine with express -**

****

**3. Create views folder under the project folder**

****

**4. Create file with extension .ejs under the views folder  -**

****

**Example with source code :**

**first.ejs**

<html>

 <head>

   <title>i am ejs file</title>

 </head>

 <body>

   <h1>Hey i am EJS file </h1>

 </body>

</html>

**5. Call ejs file from express API –**

If we want to ejs file from express API we have render() method res object

****

**Note : same thing we did with HTML file so**

**Q. What is the difference between ejs and HTML file?**

If use ejs then we can accept dynamically data from a server side and generate dynamic data at view page which is not possible using .html file.

If we want to accept dynamically from a server side and display dynamic content on web page using EJS we have know the some expression and tags provided by EJS which help us to generate dynamic UI content at view page

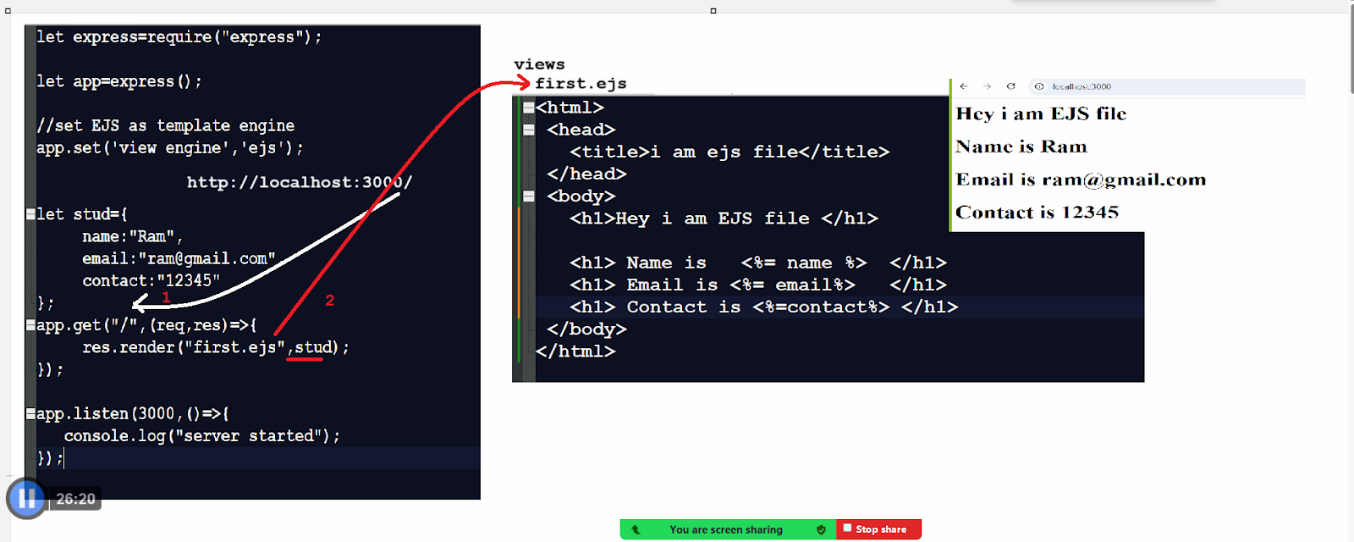
**1) Express tag :** <%= %> :  this is used for display the output on web page like as document.write() of javascript.

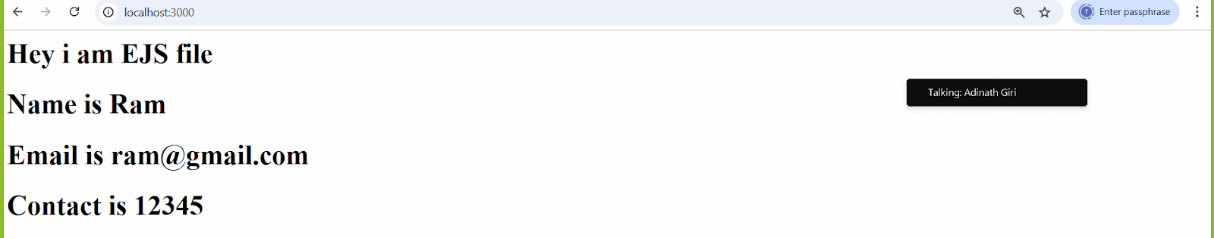
**2) <%  write here your logics like as loop, if etc  %> :** this express tag help us to write dynamic logic on web page using ejs.

**How to send data from express JS API to ejs file -**

If we want to send data from node API to ejs we can send using JavaScript object format using key and value pair. we can access that data by using Ejs expression tags by name

**Example :** we want to send student info from express JS API to ejs file means want to send data of student like as name,email and contact and show on web page using ejs file





**Example with source code : Design page like as  -**



**index.js :**

let express=require("express");

let app=express();

//set EJS as template engine

app.set('view engine','ejs');

let stud={

name:"Ram",

email:"ram@gmail.com",

contact:"8888888888",

marks:[60,70,80,90,70,80]

};

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

     res.render("first.ejs",stud);

});

app.listen(3000,()=>{

   console.log("server started");

});

**First.ejs**

<html>

 <head>

   <title>i am ejs file</title>

    <!-- Bootstrap CSS -->

 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.min.css" >

 </head>

 <body>

   <table class="table table-striped">

  <thead>

    <tr>

      <th scope="col">  NAME:&nbsp;&nbsp; <%=name%>  </th>

      <th scope="col">  EMAIL: &nbsp;&nbsp; <%=email%></th>

  <th scope="col">  CONTACT: &nbsp;&nbsp; <%=contact%></th>

    </tr>

  </thead>

  <tbody>

     <% marks.forEach((item,index)=>{

%>

  <tr>

    <td>Sub: <%=(index+1)%> </td>

<td colspan='2'> <%=item%></td>

  </tr>

<%

  });

%>

  </tbody>

</table>

 </body>

</html>

**Mini Project**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Tech Stack :**  HTML,CSS,bootstrap,JavaScript(validation), ejs

**Server:**          Node, express, MYSQL , AJAX

**Title :** Student Submission Project

**URL rewriting**

URL Rewriting is technique where we can send requested data via URL in the form of query string or name and value pair

URL rewriting is a technique used to modify the appearance or structure of URLs, often to make them more user-friendly, secure, or search engine optimized (SEO-friendly). Instead of showing a complex and technical URL, URL rewriting allows you to display a cleaner, more meaningful one.

**🔧 Example Before and After URL Rewriting**

* **Without URL rewriting (original URL) :**  
  http://example.com/products.php?id=123&category=shoes
* **With URL rewriting (rewritten URL) :**  
  http://example.com/shoes/123

**📌 Why Use URL Rewriting?**

1. **Improves readability** – Easier for users to understand and remember.
2. **Better SEO** – Search engines rank cleaner URLs higher.
3. **Hides technology details** – Keeps internal scripts and query strings hidden.
4. **Security** – Prevents users from manipulating query parameters easily.

**🛠️ How It's Done :**

1) In **Apache** (via .htaccess file), using mod\_rewrite:

RewriteEngine On

RewriteRule ^shoes/([0-9]+)$ products.php?id=$1&category=shoes [L]

2) In **Node.js (Express)** or other backend frameworks, routes are defined to match friendly URLs :

app.get('/shoes/ : id', (req, res) => {

const productId = req.params.id;

// Fetch product based on productId

});

**Middleware concept in node**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

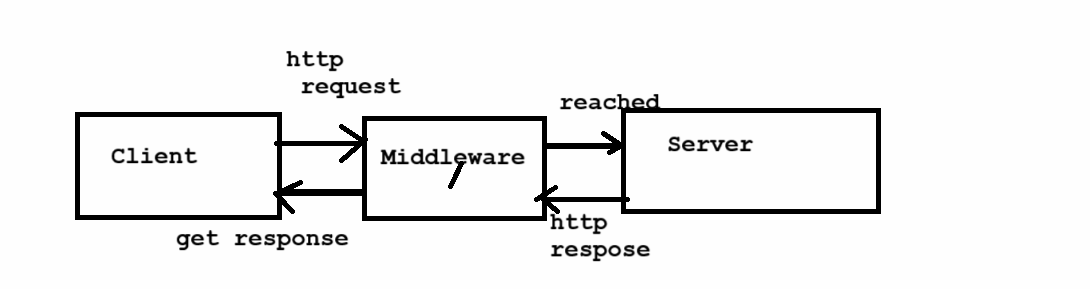
Middleware in express to function that process request before reaching the router handlers.

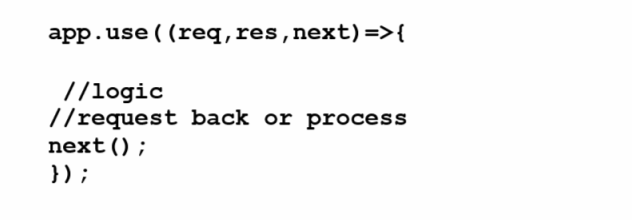
This function can modify the request and response object and end the request and response cycle or the next middleware function.

Using middleware you can perform task like as authentication , logging or error handling etc.

**Middleware** is software that acts as a bridge between different parts of an application, especially in web development. In backend frameworks like **Express.js**, **Spring Boot**, or **Django**, middleware functions **intercept HTTP requests and responses** to perform tasks like:

* Authentication and Authorization
* Logging
* Error handling
* Request parsing (e.g., JSON body parsing)
* Setting headers (e.g., CORS, cookies)
* Validating inputs





**(req,res,next)=>{} :** this is the middleware function where you can perform action on the request and response objects before  the final handler is executed

**next() :** this function is called to pass control to the next middleware in the stack if the current does not end the request and response cycle.

**Example :**

let  express= require('express');

let app=express();

app.use((req,res,next)=>{

  console.log(`${req.method} and ${req.url}`);

  next();

});

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

   res.send('<h1>Home Page</h1>')

});

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

  res.send("<h1>Hello Guys</h1>");

  res.end();

});

app.listen(4000,()=>{

  console.log('server Started..!');

});

**What Middleware does in express?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Execute the code  :** middleware can run any code when request is received
2. **Modify Request and Response:** means using middleware you can modify the request object and response object.
3. **End the Request and Response :** Middleware can send a response to the client ending the cycle.
4. **Call the next middleware** :  middleware can call next() to pass control to the next function in the middleware stack.

**How Middleware works in express JS?**

In express js middle functions are executed sequentially in the order they added to the application

1. Request arrives at the server
2. Middleware functions are applicated to request one by one
3. Each Middleware can either :
   1. Send a response and end the request and response cycle
   2. Call next() to pass control to the next middleware
4. If no middleware end the cycle the router handler is reached and final response send to client

**🔄 Types of Middleware**

* **Application-level middleware** (used globally in app)
* **Router-level middleware** (specific to routes)
* **Error-handling middleware**
* **Built-in middleware** (like express.json())
* **Third-party middleware** (like cors, morgan)

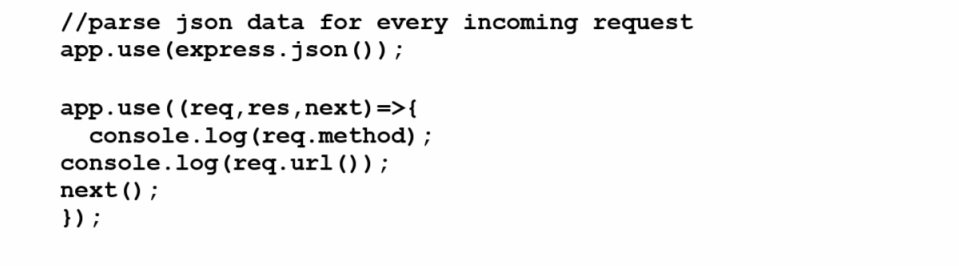
**Explanation Of Each Middleware :**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1. Application level middleware :** Application level middleware is bound to the entire express application using app.use() or methods like app.get(), app.post(), etc.

It execute  for all routes in the application , regardless of the specific path of Http method

This type of middleware is commonly used for tasks like logging, body parsing , authentication checks or setting header for every common request.



Middleware that is bound to the entire application and runs for **every incoming request** (unless restricted by path or method).

**✅ Use case:** Logging, request validation, parsing JSON, etc.

let express=require('express');

let app=express();

app.use((req,res,next)=>{

   console.log(`${req.method} And ${req.url}`)

   next();

});

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

    res.send("<h1>Application Level Middleware</h1>")

    res.end();

});

app.listen(4000,()=>{

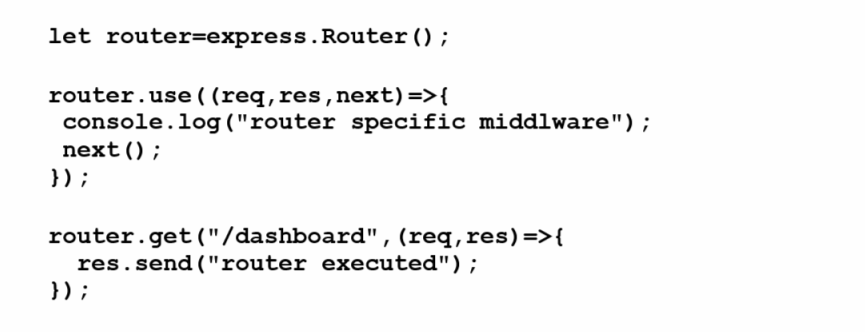
  console.log('server start..!')

});

**2. Router-level middleware :** Router level middleware is applied to a specific route instance using router.use()  or route.METHOD(). It only applies to routers defined  within that particular router, making it perfect for modular applications where the middleware is only relevant to a specific router group.

This type of middleware is often used to group related routes (e.g all routes related to authentication or user management)  apply middleware logic to them.

Router-level middleware works exactly like application-level, but is bound to an instance of express.Router() instead of the whole app.



**Code : admin.js**

let express = require('express');

let router = express.Router();

router.use((req,res,next)=>{

   console.log(`admin page url : ${req.url}`);

   next();

});

router.get('/',(req,res)=>{

    console.log('hello admin');

    res.send('<h1>Welcome to the Admin Page..!</h1>');

    res.end();

});

module.exports = router;

**user.js**

let express=require('express');

let router = express.Router();

router.use((req,res,next)=>{

    console.log(`User Page Info : ${req.url} & ${req.method}`)

    next();

});

router.get('/',(req,res)=>{

  res.send('Welcome to User Dashboard..!');

  res.end();

});

module.exports = router;

**app.js**

let express = require('express');

let app=express();

//Import the page..

let userPage=require('./router/user');

let adminPage=require('./router/admin');

//use the page..

app.use('/admin',adminPage);

app.use('/user',userPage);

app.listen(4000,()=>{

    console.log('Start server..!')

});

**3. Error Handling Middleware :** Error handling middleware is a special type of middleware used to catch and respond to errors during the request and response cycle. It is defined with four parameters : **Err , Req , Res , Next**

This middleware is essential for sending a consistent error response and avoiding unhandled exceptions that might crash server



**Code : app.js**

//Error handling Middleware

app.use((err,req,res,next)=>{

   console.error('Error Handler : ' , err.message);

   res.status(500).send('Something broke!');

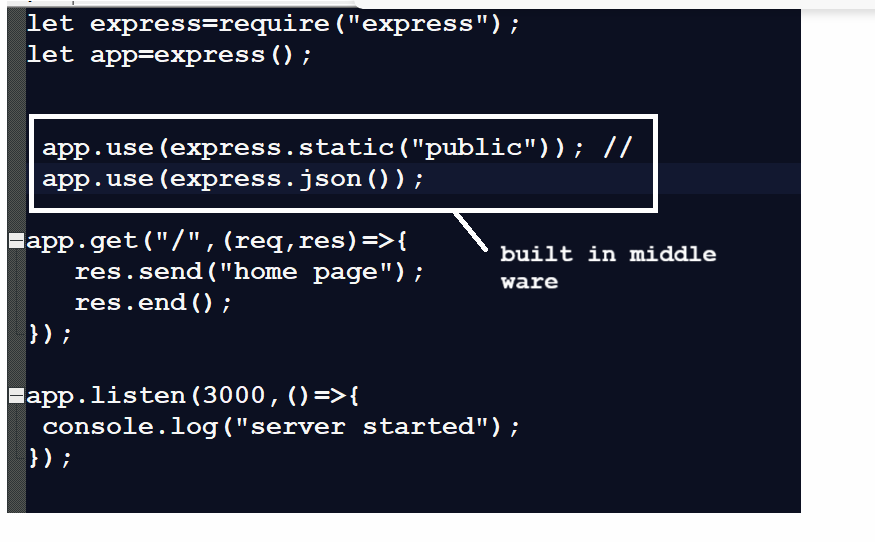
});

**4. Built in Middleware :** Express provides built in middleware to help with common tasks like serving static files or parsing data.

**Example :** express.static(),express.json() etc

**✅ 1. express.static() - Serve Static Files :**

serves static files like html,css or images and express.json() help parse incoming JSON data. Use it to serve HTML, CSS, JS, images, etc.



**✅ 2. express.json() — Parse JSON Request Body :**

Used to handle incoming JSON data (like from POST requests).

app.use(express.json());

**Example :**

app.post('/data', (req, res) => {

console.log(req.body); // parsed JSON

res.send('Data received');

});

**✅ 3. express.urlencoded() — Parse URL-Encoded Form Data :**

Used to handle data submitted via HTML forms. Post method time use compulsory.

app.use(express.urlencoded({ extended: true }));

**Example :**

app.post('/form', (req, res) => {

console.log(req.body); // parsed form data

});

**Note :**

 Use express.json() for API backends (POST/PUT with JSON).

 Use express.urlencoded() for traditional HTML form handling.

 Use express.static() to serve frontend assets.

**5. Third party middleware :** Third party middleware is developed by external developers and packaged as npm modules. These middleware packages add additional functionality to your application.

**✅ Common Examples of Third-Party Middleware**

| **Middleware** | **Purpose** | **Installation** |
| --- | --- | --- |
| morgan | Logging incoming requests | npm install morgan |
| cors | Handle Cross-Origin Resource Sharing | npm install cors |
| helmet | Secure app by setting HTTP headers | npm install helmet |
| cookie-parser | Parse cookies in requests | npm install cookie-parser |
| express-session | Handle session management | npm install express-session |

**🔧 Example: morgan (Request Logger) :**

Show the Info those are request like method , server code , time ms

**Step 1: Install : npm install morgan**

**Code :**

const express = require('express');

const morgan = require('morgan');

const app = express();

app.use(morgan('dev'));

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

   res.send('Hello World.');

});

app.listen(4000,()=>{

   console.log("Start Sarver..!");

});

## **🔧 Example: cors (Cross-Origin Access)**

Step 1: Install : npm install cors

const cors = require('cors');

app.use(cors());

**🔧 Example: helmet (Security Headers)**

npm install helmet

const helmet = require('helmet');

app.use(helmet());

**🧠 When to Use:**

**🔐 helmet :** Secure your app in production.

**🌍 cors :** Allow frontend apps (React, Angular) to access your API.

**🧾 morgan :** Log all API requests (for debugging and monitoring).

**🍪 cookie-parser :** When you're working with cookies or sessions.

**✅ What is a Session?**

A **session** is a way to **store user data across multiple requests** from the same client.

When a user logs in, we create a session for them and store data (like userId, role, etc.) on the server side. This data is preserved as long as the session is valid.

Session is a communication period between client and server over the network called as session  or user login to user logout time or process called as session

**Q. Why do we need to use sessions?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Basically http is a **stateless** **protocol** that means every request of a client is considered as a new client at server side even if the same client sends multiple requests . This behavior of **http protocol is known as stateless protocol.**

If we want to convert stateless behavior of http protocol to stateful behavior then we can use session in web application.

When we use session in web application then server maintain the state of client i.e create one session id and store in header of client and one copy maintain at server side and when client revisit to server with new request then server verify session id of that client present on server or not if session id of that client present at server then consider it is existing client if session id not present on server then consider it is new client.

## **🧠 How Does it Work?**

1. Client makes a request (e.g., login)
2. Server creates a **session object** and stores it in memory or a session store
3. Server sends back a **Session ID** stored in a cookie (connect.sid)
4. On every next request, the browser sends this cookie
5. Server checks the session ID, retrieves the session data, and processes the request

**Q. Can you give an example where I can use sessions in my project?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Session help us to main user privacy :** if we session in web application then we can maintain user privacy means we can access of data to user according to his login

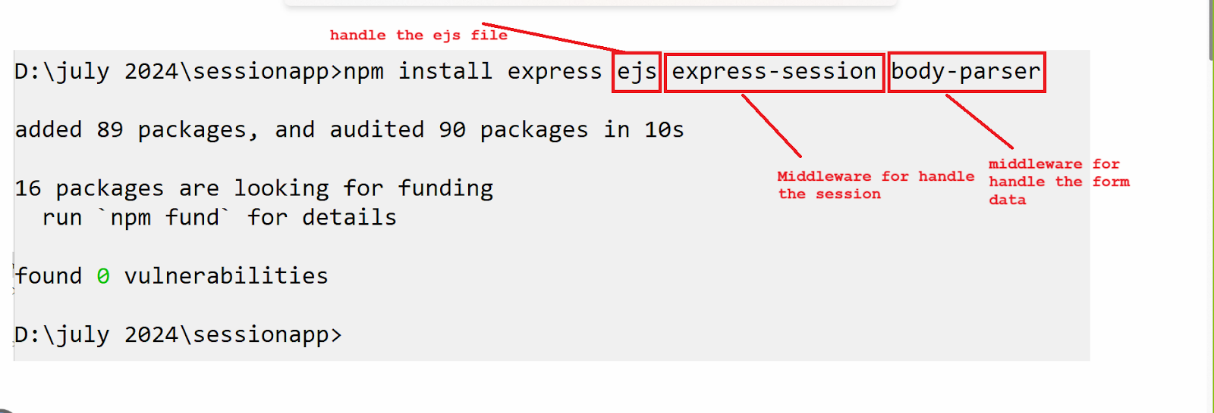
Means one user cannot see the data of another users if we session because session is separate every user or client

1. **Session help us to store data temporary at client side**
2. **Session can provide security to user page**
3. **Count the active user or login user on portal**

**Etc**

**How to session practically using express?**

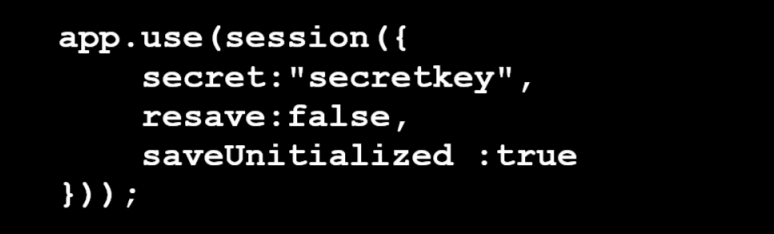
1. Install the express-session middleware dependency /library



2. Import expression-session library

****

3. Configure session : if we want to handle the session we require to configure session in the form of JavaScript object.



**secret :** A string (or array of string) used to sing the session ID. this prevents tampering

**resave :** if true session is saved on every request event it wasn't change so usually set false.

**saveUninitialized :**  if true session that are new but modified will be save useful for implementing login flows or tracking visit.

**cookie( optional ) :** you can pass additional cookie options like maxAge, secure etc.

**How to generate secret key for security**

D:\july 2024\sessionapp>

node -e "console.log(require('crypto').randomBytes(64).toString('hex'))"

**Output :** 95856526a58e003135b1ffbb6fb34014573acfbfa9b6fffa18f5f640a5eff13395

**Note :** if we think about session object internally for every client / browser server generate separate session object to us and every session object has different session id means every client has different session.

If we want to check session id or get session id of client we have following syntax

**Syntax:** let sessionid = req.sessionID;

**Example with source code**

let express=require("express");

let session=require("express-session");

let app=express();

app.use(session({

     secret : "95856526a58e13ba116440743155b1 73acfbfa9b6fffa18f5f640a5eff13395",

resave : false,

saveUnitialized : true

 }));

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

  res.send("Hey client your session id "+ req.sessionID);

  res.end();

});

app.listen(3000,()=>{

  console.log("Server started");

});

**4.** **Store data in session**

When we store data in session means every session object has different means every client has different session means every client has different data.

**Syntax :** req.session.key = data

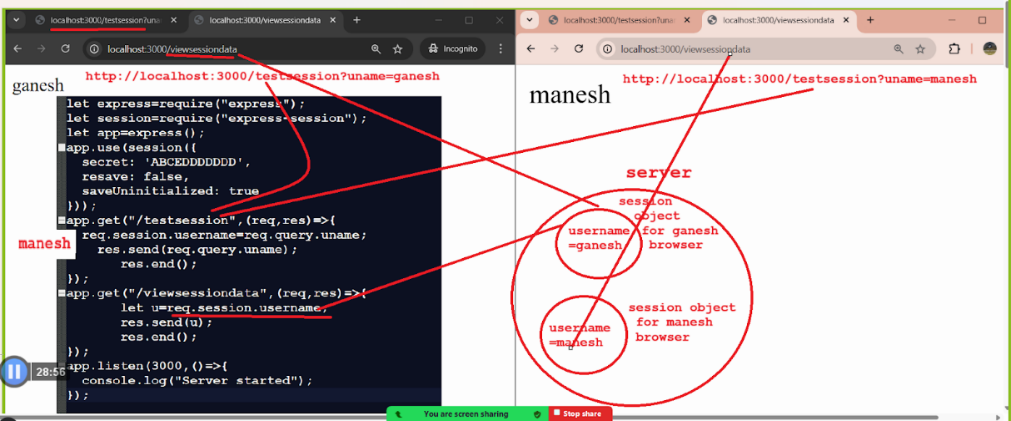
**Note :** we store in session using key and value pairs

**Example :** req.session.username=”ABC”

**5. Retrieve data from a session  :**

If we want to retrieve data from a session we have following syntax –

**Syntax :** let variablename = req.session.key;



**Example with source code**

let express=require("express");

let session=require("express-session");

let app=express();

app.use(session({

  secret : 'ABCEDDDDDDD',

  resave : false,

  saveUninitialized : true

}));

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

   req.session.username=req.query.uname;

   res.send(req.query.uname);

res.end();

});

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

  let u=req.session.username;

  res.send(u);

  res.end();

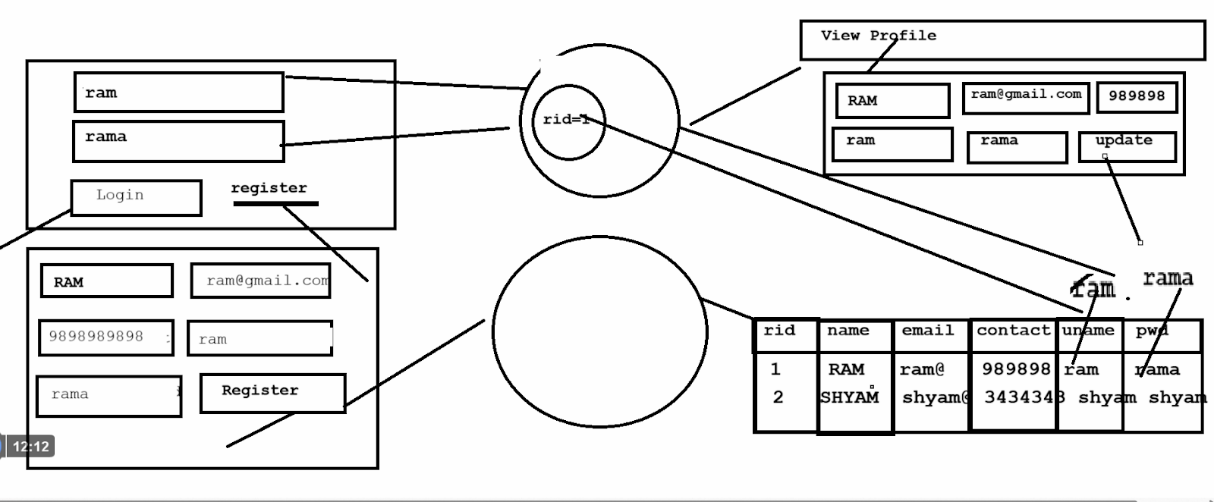
});

app.listen(3000,()=>{

  console.log("Server started");

});

**Assignment goal :**



When we design any application or project then we should have to  consider following thing.

**1. Requirement Gathering :**

1. Functional Requirement : What should the system do?
2. Non Functional Requirement : performance , scalability , security etc.
3. Stakeholder input : who will use it and what do they need?
4. Constraints : Timeline, budget, technology, legal etc.

**2. System Architecture (HLD) :**

1. Architectural style :Monolithic, microservices , serverless etc**.**
2. Technological stack :Front End, Backend , Database, third party etc.
3. Data Flow and Module Interaction :how components talk to each other
4. Deployment model :cloud, on-premises?

**3. Database Design :**

Data Modeling :ER diagrams, entity relationship ,normalization

Storage technology : SQL, NOSQL, distributed DB etc.

Data Access Technology :ORM or SQL etc.

**4. API design :**

API Contracts : **REST , GraphQL etc.**

Authentication/authorization :JWT , OAuth, API keys , session etc**.**

**5. Low Level Design (LLDP) :**

   Class diagram and interfaces

   Design pattern : singleton, factory, adapter, builder etc.

   Object modeling : SOLID principles ,DRY/YAGNL etc.

**6. Security consideration :**

Input validation and sanitization

Data encryption

Secure deployment practices like as https , firewalls

**7. Performance and Scalability**

1. Caching strategy
2. Load balancing
3. Horizontal and vertical scaling
4. Asynchronous processing
5. **DevOps & CI/CD**
6. Version control
7. Automated testing
8. Continuous integration and deployment pipelines
9. Infrastructure as Code( Terraform, docker etc)
10. **Monitoring & Logging**
11. Health check
12. Centralized logging
13. **Testing Strategy**
14. **Documentation**

**API documentation (Swagger/Postman)**

**Requirement understanding  :**

Functional requirement :A functional requirement describe what a system should do , the specific behavior , functions or problems it must support to full fill user needs or business rules  Example :

1. **User Registration :** the system must be allow use to register using their name, email and contact, username and password
2. **User Login :** The system must be allow use to log in using valid username and password
3. **Password Security : password must be stored in hashed format**
4. **View Profile :** After logging the user must be able to view their profile details
5. **Session Management :** system should maintain the session after  login until logout or token expiry

**Non-Functional requirement :** A non functional requirement defines how a system performs its function rather than what it does. This type requirement describe the quality attributes of the system Performance, usability , security, reliability 

**High level Design**

Focus : Architecture , components, technology etc

Audience : Architect , senior developers , stakeholders etc

Details : No code level details, focus on module and data flow

Purpose : Guide to LDD and ensure all pieces together in picture

**High level Design :**

Architecture: monolithic , MVC design pattern  Database : Single table

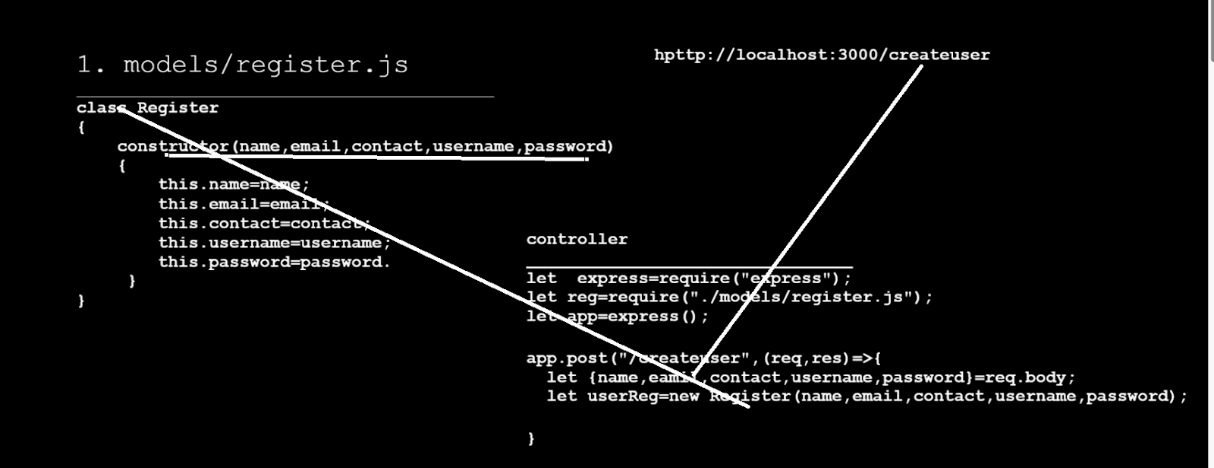
**Low level design :**

Focus: internal logic , class structure , function level design

  Audience : developers and testers

  Details: class diagram , database query , API, algorithm

**Example:**

****

**Q. What is MVC?**

MVC stands for Model View Controller basically MVC is standard design pattern in web application development.

**M ( Model  ) :** Model are the classes which is used for to store data and work with database as well as model help us to accept the data send by  view and store in model object and work with db via controller

**V ( View ) :** view means user interface from user can provide input and get result

If we think about view we can develop view using react, angular, ejs , html etc.

**C ( Controller ) :** controller is used for accept the data as request send by view and store in model and process on it and pass to service and database

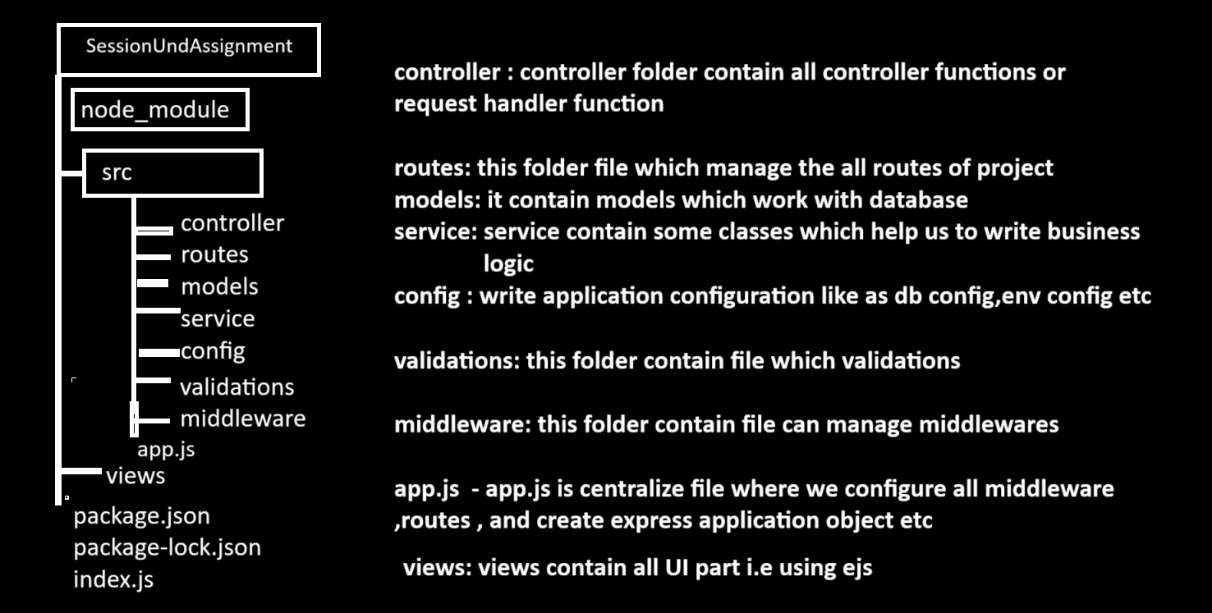
If we think about node we use request handler function as controller

**Q. Why use  MVC?**

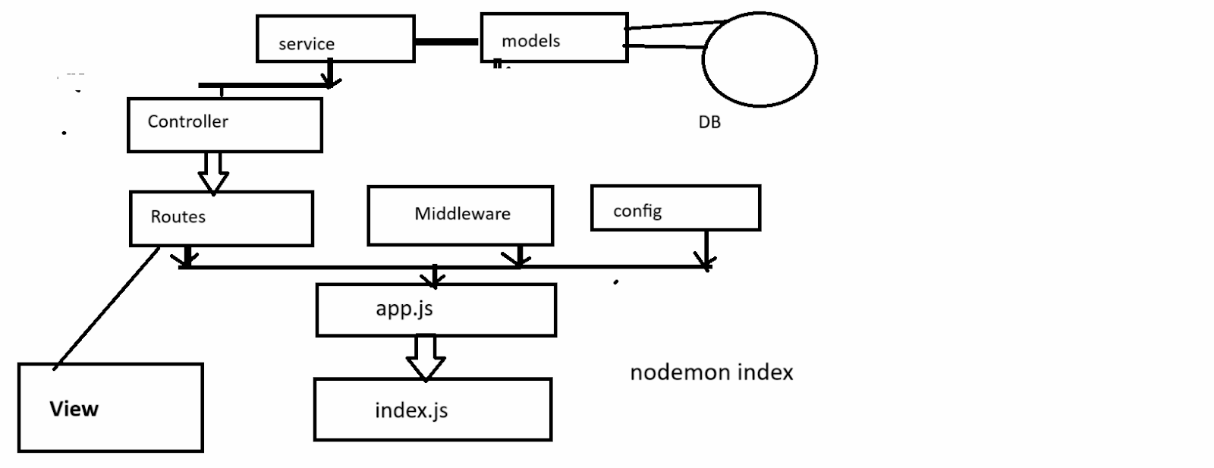
The major goal of MVC is code separation means using MVC we separate designing as well as business logic and database logic

**Start Development of Project**

If we want to work with node project we have some standard folder structure



**How internally folders and files communicate with each**



**Cookie**

**Q. What is a cookie?**

A Cookie is small piece of data that a website stores on a user’s device (like as computer or smartphone) through the web browser

Cookie help website remember information about the user such as login status , preferences or items in shopping cart

Basically cookie is part of session handling means cookie create separate for every user or browser.

A **cookie** is a small piece of data (key-value pairs) that a **server sends to a user's web browser**. The browser stores it and sends it back to the server with **each subsequent request** to the same server. Cookies are mainly used for **:**

* **Session management** (e.g., login sessions)
* **Personalization** (e.g., language preferences, themes)
* **Tracking** (e.g., analytics, ad targeting)

**Q. What is the difference between cookie and session?**

1. Cookie stored its data at client side i.e. in browser and session store its data at server side
2. Cookie can store its data for longer period at client side after browser close and session can vanish its data when client logout or close the browser or client application

**Note :** if we use the cookie and if user disable its cookie option at browser side then cookie vanish automatically when close the browser means work as session

If we use the cookie then try to avoid store confidential or important information in cookie because cookie at client side so there may be possibility of data leakage or privacy break

**📦 Structure of a Cookie**

A cookie typically contains:

* **Name**: The key or identifier
* **Value**: The actual data stored
* **Domain**: The website that set the cookie
* **Path**: The URL path the cookie is valid for
* **Expires/Max-Age**: How long the cookie should last
* **Secure**: Whether the cookie should only be sent over HTTPS
* **HttpOnly**: Prevents JavaScript from accessing the cookie (for security)
* **SameSite** : Controls whether cookies are sent with cross-site requests (helps prevent CSRF)

**There are two types of cookie**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Session based cookie/temporary cookie :** cookie without age life called as session cookie or temporary means these types of cookie vanish after browser close
2. **Persistent cookie :** persistent cookie means a cookie with age life means when we create a cookie we set its age called as persistence cookie.

## **🧠 Cookie Types**

| **Type** | **Description** |
| --- | --- |
| **Session** | Deleted when the browser is closed. No expiration time set. |
| **Persistent** | Stored until the expiry date is reached. |
| **Secure** | Sent only over HTTPS. |
| **HttpOnly** | Inaccessible to JavaScript (for security). |
| **Third-party** | Set by a domain different from the one being visited (e.g., ads). |

**Example or Real life example of cookie**

**Online shopping (Amazon)** : Suppose in online shopping we create cart and add data or item in cart but we leave the website without placing order and Later after some time if we visit again website e.g. amazon then we get your cart which we created  So this type of data stored in cookie and set its age by server.

**How does amazon remember the user or facebook remember the user :**

So if we think about above scenario cart information and user information stored in cookie and when user revisit using its browser then server check the user info and cart info at server side  using cookie object and if server found data then consider it is existing user and its data otherwise create new cookies.

**📍 Cookie Use Cases**

1. **Login Sessions**  
   Track logged-in users with session cookies.
2. **Shopping Carts**  
   Remember items added to the cart.
3. **Preferences**  
   Store language or theme preference.
4. **Analytics & Tracking**  
   Track user activity (e.g., Google Analytics).

**🚫 Cookie Limitations**

* **Size limit**: ~4 KB per cookie.
* **Number limit**: 20-50 cookies per domain depending on the browser.
* **Security**: Vulnerable to XSS (if not using HttpOnly) and CSRF (if not using SameSite).

🔧 **1. Using Native http Module**

**📁 File: server.js :**

**Code :**

const http = require('http');

const server = http.createServer((req, res) => {

  if (req.url === '/') {

    // Set a cookie

    res.setHeader('Set-Cookie', 'username=Kiran; HttpOnly');

    res.writeHead(200, { 'Content-Type': 'text/plain' });

    res.end('Cookie set');

  }

  else if (req.url === '/read') {

    // Read cookies

    const cookies = req.headers.cookie;

    res.writeHead(200, { 'Content-Type': 'text/plain' });

    res.end('Cookies: ' + cookies);

  }

});

server.listen(3000, () =>

  console.log('Server running at http://localhost:3000'

));

**How to use cookies in express JS**

If we want to use cookie in express JS we have to use cookie-parser middleware

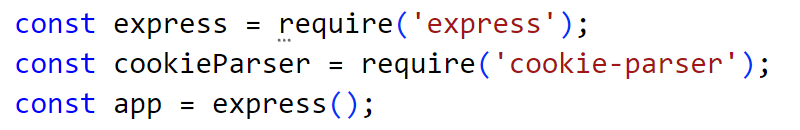
This middleware allows the express server to set , read or clear cookies in http request and response.

**Steps to work with cookie in Express**

1. Install cookie parser

npm install express cookie-parser

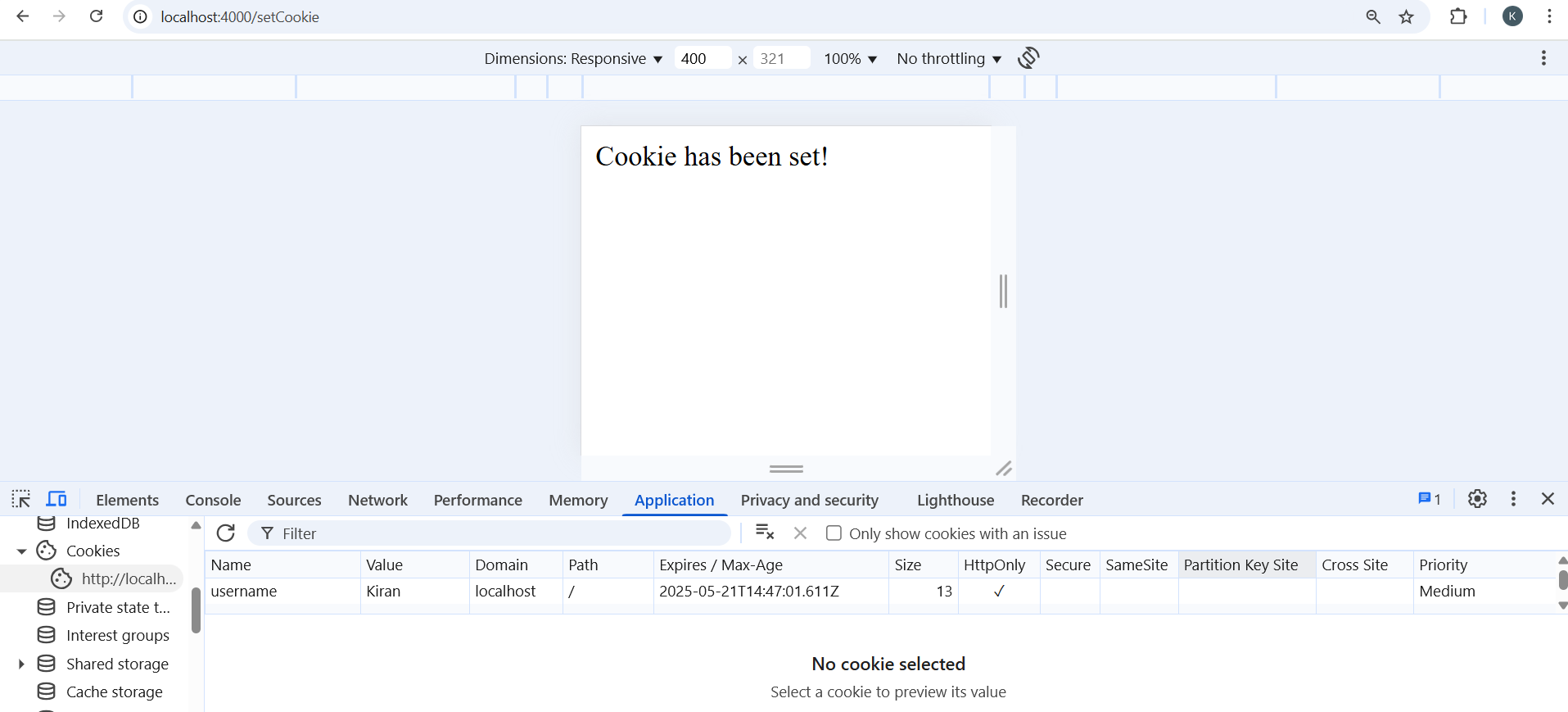
2. Import cookie parser in .js file

****

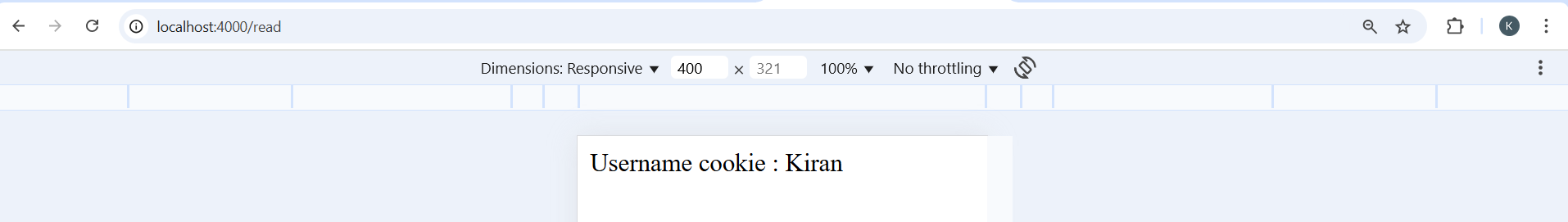
3. Use cookie as application middleware :



**Output :** localhost:4000/setCookie



localhost:4000/read

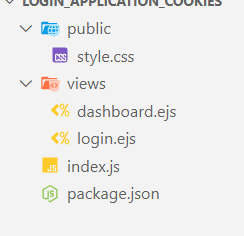


**Login system example using the following tech stack** :

**🛠 Tech Stack :**

* **Express.js** ( Web framework)
* **EJS** (Template engine)
* **cookie-parser** (Cookie handling)
* **body-parser** (Form data parsing)

**Folder Structure :**



**Install some dependencies :**

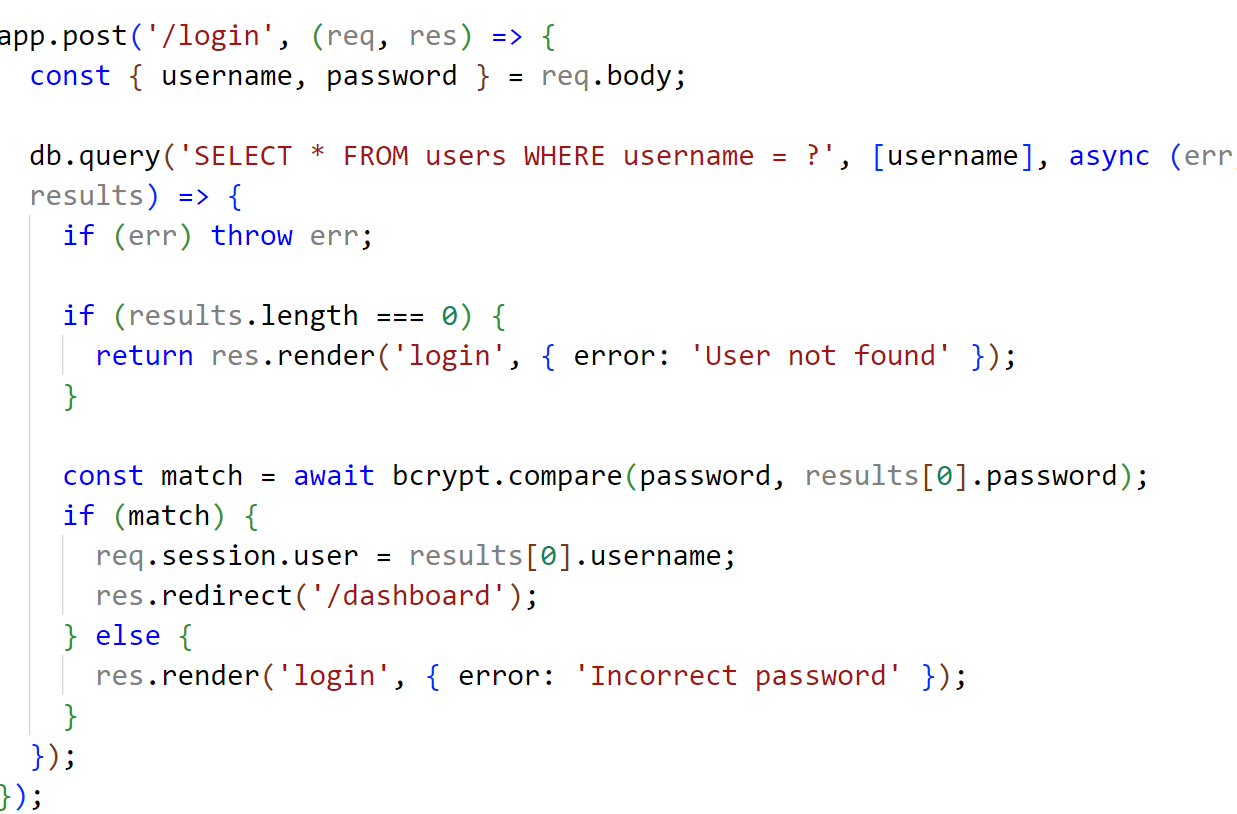
* npm install
* npm install express ejs mysql2 body-parser bcrypt express-session cookie-parser

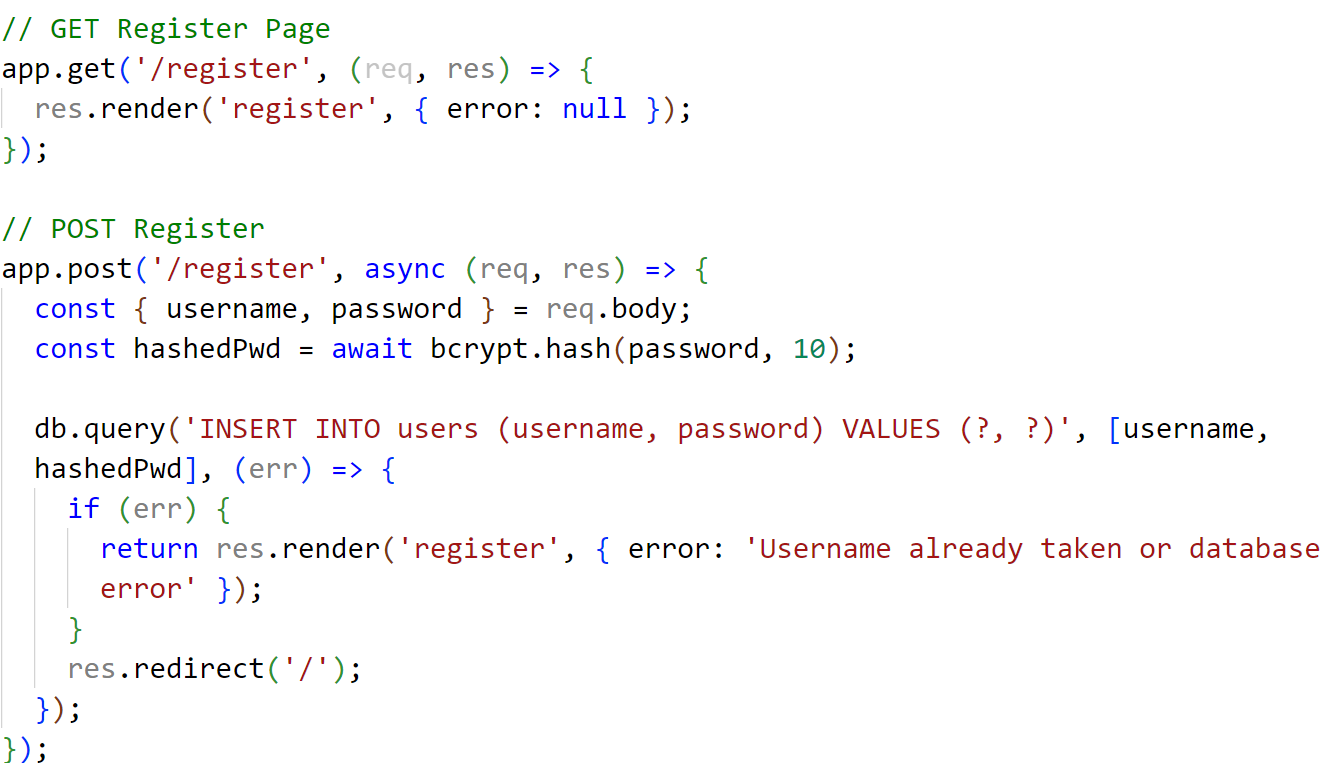
**index.js**













**login.ejs**



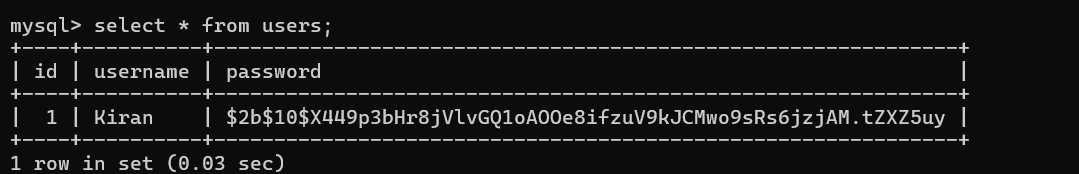
**dashboard.ejs**

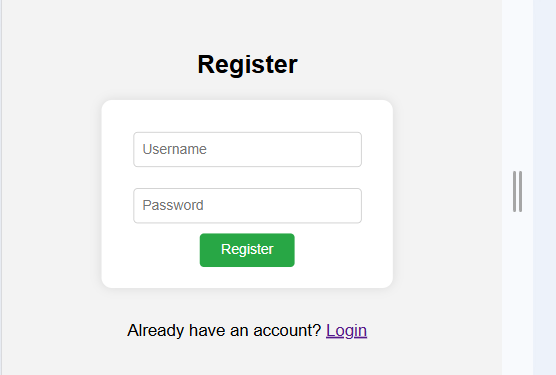


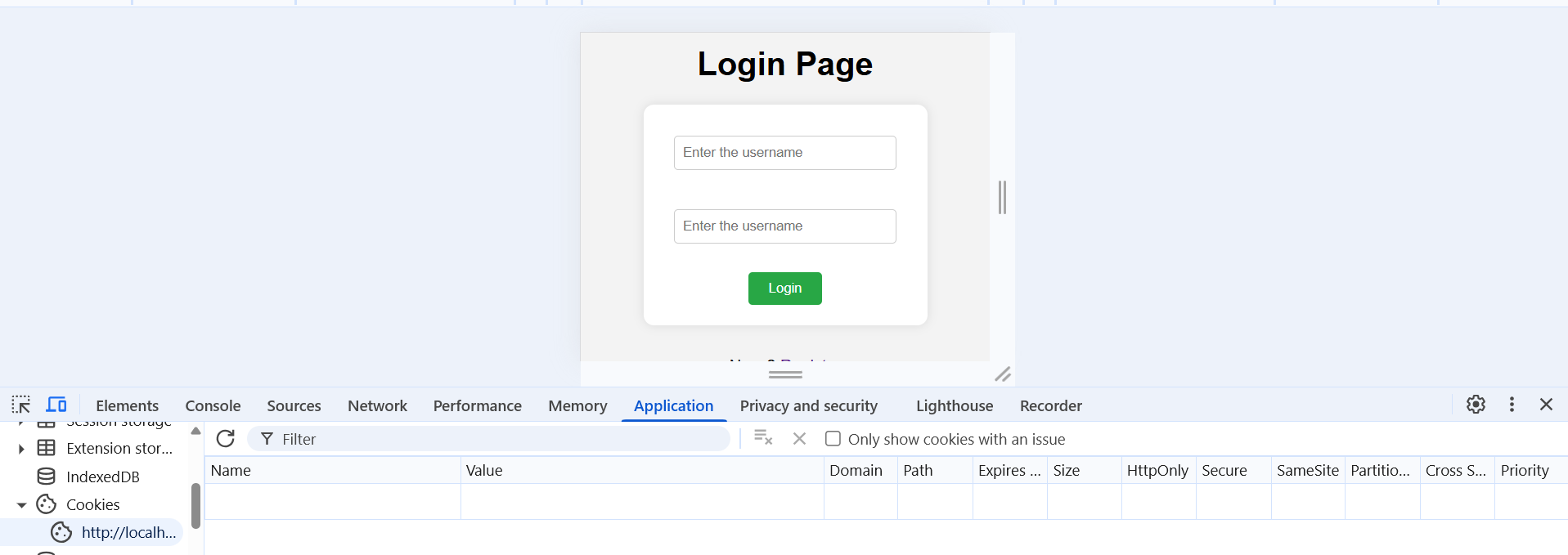
**register.ejs**



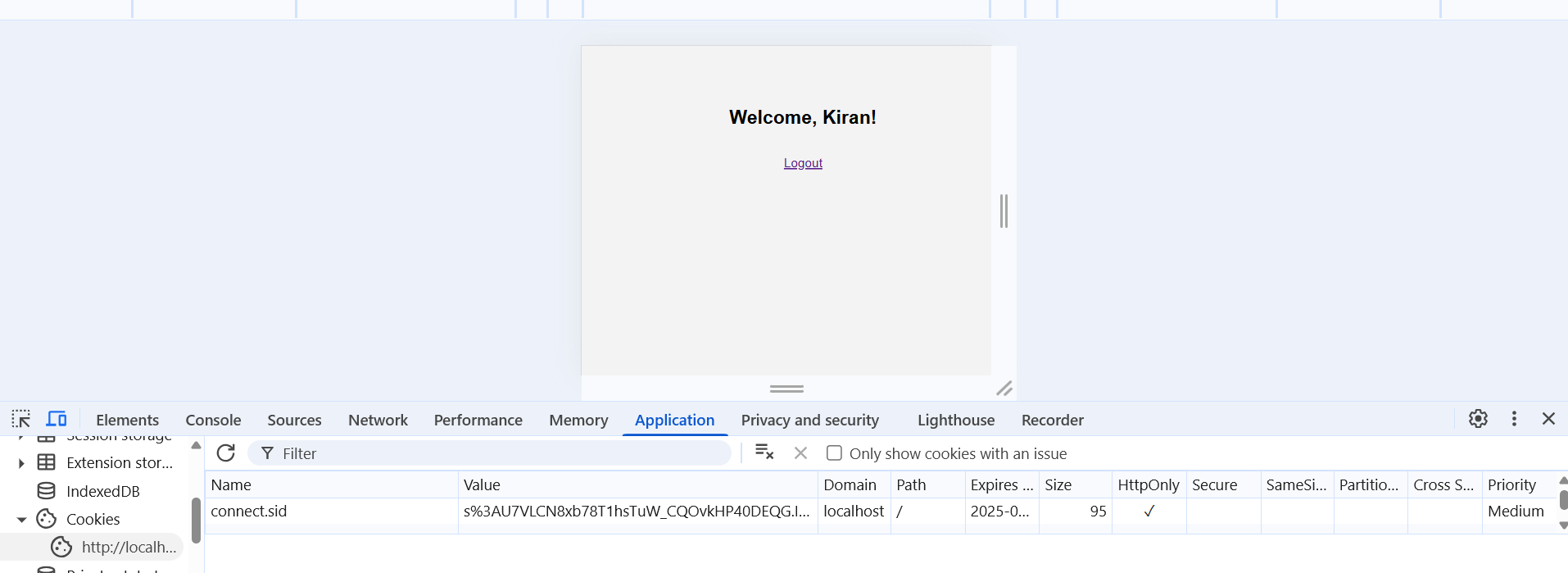
**Output :**







After login success cookies create :



After logout then cookies are destroy :

