



React js-Node js-Express JS

Backend Book



Er. Rajesh Prasad(B.E, M.E)
Founder: TWF & RID Org.

- **RID ORGANIZATION** यानि **Research, Innovation and Discovery** संस्था जिसका मुख्य उद्देश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो।
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद **इस RID संस्था** की स्थपना किया हूँ।
- Research, Innovation & Discovery में रुचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियों से मैं आवाहन करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुद्धि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें।

त्वक्सा Node JS के इस ई-पुस्तक में आप Node JS से जुड़ी सभी बुनियादी अवधारणाएँ सीखेंगे। मुझे आशा है कि इस ई-पुस्तक को पढ़ने के बाद आपके ज्ञान में वृद्धि होगी और आपको कंप्यूटर विज्ञान के बारे में और अधिक जानने में रुचि होगी।

“In this E-Book of TWKSAA Node Js you will learn all the basic concepts related to Node Js. I hope after reading this E-Book your knowledge will be improve and you will get more interest to know more thing about computer Science”.

Online & Offline Class:

Python, Web Development, Java, Full Stack Course, Data Science, UI/UX Training, Internship & Research

करने के लिए Message/Call करें. 9202707903 E-Mail_id: ridorg.in@gmail.com

Website: www.ridtech.in

RID हमें क्यों करना चाहिए ?

(Research)	(Innovation)	(Discovery)
अनुसंधान हमें क्यों करना चाहिए ? Why should we do research? 1. नई ज्ञान की प्राप्ति (Acquisition of new knowledge) 2. समस्याओं का समाधान (To Solving problems) 3. सामाजिक प्रगति (To Social progress) 4. विकास को बढ़ावा देने (To promote development) 5. तकनीकी और व्यापार में उन्नति (To advances in technology & business) 6. देश विज्ञान और प्रौद्योगिकी के विकास (To develop the country's science & technology)	नवीनीकरण हमें क्यों करना चाहिए ? Why should we do Innovation? 1. प्रगति के लिए (To progress) 2. परिवर्तन के लिए (For change) 3. उत्पादन में सुधार (To Improvement in production) 4. समाज को लाभ (To Benefit to society) 5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition) 6. देश विज्ञान और प्रौद्योगिकी के विकास (To develop the country's science & technology)	खोज हमें क्यों करना चाहिए ? Why should we do Discovery? 1. नए ज्ञान की प्राप्ति (Acquisition of new knowledge) 2. अविष्कारों की खोज (To Discovery of inventions) 3. समस्याओं का समाधान (To Solving problems) 4. ज्ञान के विकास में योगदान (Contribution to development of knowledge) 5. समाज के उन्नति के लिए (for progress of society) 6. देश विज्ञान और तकनीक के विकास (To develop the country's science & technology)

❖ Research(अनुसंधान):

- अनुसंधान एक प्रणालीकरण कार्य होता है जिसमें विशेष विषय या विषय की नई ज्ञान एवं समझ को प्राप्त करने के लिए सिद्धांतिक जांच और अध्ययन किया जाता है। इसकी प्रक्रिया में डेटा का संग्रह और विश्लेषण, निष्कर्ष निकालना और विशेष क्षेत्र में मौजूदा ज्ञान में योगदान किया जाता है। अनुसंधान के माध्यम से विज्ञान, प्रोधोगिकी, चिकित्सा, सामाजिक विज्ञान, मानविकी, और अन्य क्षेत्रों में विकास किया जाता है। अनुसंधान की प्रक्रिया में अनुसंधान प्रश्न या कल्पनाएँ तैयार की जाती हैं, एक अनुसंधान योजना डिज़ाइन की जाती है, डेटा का संग्रह किया जाता है, विश्लेषण किया जाता है, निष्कर्ष निकाला जाता है और परिणामों को उनिवेदित दर्शाने के लिए समाप्ति तक पहुंचाया जाता है।

❖ Innovation(नवीनीकरण): -

- Innovation एक विशेषता या नई विचारधारा की उत्पत्ति या नवीनीकरण है। यह नए और आधुनिक विचारों, तकनीकों, उत्पादों, प्रक्रियाओं, सेवाओं या संगठनात्मक ढंगों का सृजन करने की प्रक्रिया है जिससे समस्याओं का समाधान, प्रतिस्पर्धा में अग्रणी होने, और उपयोगकर्ताओं के अनुकूलता में सुधार किया जा सकता है।

❖ Discovery (आविष्कार):

- Discovery का अर्थ होता है "खोज" या "आविष्कार"। यह एक विशेषता है जो किसी नए ज्ञान, अविष्कार, या तत्व की खोज करने की प्रक्रिया को संदर्भित करता है। खोज विज्ञान, इतिहास, भूगोल, तकनीक, या किसी अन्य क्षेत्र में हो सकती है। इस प्रक्रिया में, व्यक्ति या समूह नए और अज्ञात ज्ञान को खोजकर समझने का प्रयास करते हैं और इससे मानव सभ्यता और विज्ञान-तकनीकी के विकास में योगदान देते हैं।

नोट : अनुसंधान विशेषता या विषय पर नई ज्ञान के प्राप्ति के लिए सिद्धांतिक अध्ययन है, जबकि आविष्कार नए और अज्ञात ज्ञान की खोज है।

सुविचार:

1.	समस्याओं का समाधान करने का उत्तम मार्ग हैं	→ शिक्षा, RID, प्रतिभा, सहयोग, एकता एवं समाजिक-कार्य
2.	एक इंसान के लिए जरूरी हैं	→ रोटी, कपड़ा, मकान, शिक्षा, रोजगार, इज्जत और सम्मान
3.	एक देश के लिए जरूरी हैं -	→ संस्कृति-सभ्यता, भाषा, एकता, आजादी, संविधान एवं अखंडता
4.	सफलता पाने के लिए होना चाहिए	→ लक्ष्य, त्याग, इच्छा-शक्ति, प्रतिबद्धता, प्रतिभा, एवं सतता
5.	मरने के बाद इंसान छोड़कर जाता हैं	→ शरीर, अन-धन, घर-परिवार, नाम, कर्म एवं विचार
6.	मरने के बाद इंसान को इस धरती पर याद किया जाता हैं उनके	

→ नाम, काम, दान, विचार, सेवा-समर्पण एवं कर्मों से...

आशीर्वाद (बड़े भैया जी)



Mr. RAMASHANKAR KUMAR

मार्गदर्शन एवं सहयोग



Mr. GAUTAM KUMAR



.....सोच है जिनकी नये कुछ कर दिखाने की, खोज हैं मुझे आप जैसे इंसान की.....
 “अगर आप भी **Research, Innovation and Discovery** के क्षेत्र में रूचि रखते हैं एवं अपनी प्रतिभा से दुनियां को कुछ नया देना चाहते हैं एवं अपनी समस्या का समाधान **RID** के माध्यम से करना चाहते हैं तो **RID ORGANIZATION (रीड संस्था)** से जरुर जुड़ें” || धन्यवाद || **Er. Rajesh Prasad (B.E, M.E)**



React Js

Frontend Book

S. No:	Topic:	Page No:
1	What Is React JS ?	5
2.	How To Setup React Environment	5
3	React Js Folder & File Structure	6
4	Components In React JS	7
5	How To Create Class Based Component	8
6	How To Apply Css In React Components	9
7	How To Display Javascript Code In React Components	10
8	MAP () And Filter () In React JS	10
9	How To Display Image And Video In React Components	12
10	PROPS In React JS	15
11	Rendering Array Of Object	20
12	State In React JS	21
13	What Is Batch Processing	26
14	When To Use Call By Function	27
15	Useeffect	29
16	Usestate	32
17	Mini Project-1	33
18	Mini Project-2	35
19	Passing State As A Prop For Child Component	38
20	State Lifting	41
21	Data Access Through Api In React Js	43
22	Free Public Apis You Can Use	45
23	Route In React	49
24	Data Access From Form	69
25	How To Use Material UI And Ant Design From	72
26	Important Question And Answer 7	77

Node js-Express JS

Backend Book

S. No:	Topic:	Page No:
1	What is Node.js?	79
2	How to install Node.js & npm	80
3	Nodemon, npm start and npm init in Node.js	81
4	Module in Node.js	82
5	Promises in JavaScript / Node.js	84
6	Synchronous (Sync) and Asynchronous (Async) & await in Node.js	87
7	File Handling in node JS	91
8	Path Module in Node js	96
9	How to Build Server in Node JS	98
10	How to send external html	101
11	Route in Node JS	103
12	Express JS	104
13	How to send multiple response on browser	105
14	How to send image, video & video response on browser	105
15	How to send External multiple response on browser	106
16	Template Engines in Node.js/Express.js (EJS)	109
17	How to create Templates Engine or (EJS)	109
18	How to get data from front to backend	115
19	MVC Model in Node JS/ Express JS	118
20	How to setting up MongoDB Atlas and MongoDB Compass/Shell	119
21	Front End +Backend+ Database Connection in Node js/Express js	135
22	How you can fix your signup and login logic using bcrypt	138
23	How to connect a Node.js/Express app with local MongoDB	140
24	MySQL DB connection with Node JS/Express JS	143
25	Project with MySQL	152
26	What is API in node js/ Express js	154
27	How to create full flex RESTful API in Node JS	159
28	Cookies Node js	168
29	Session Management	172
30	Import Interview Question and answer	173
31	What is RID ?	176

❖ What is React JS ?

- React is an open-source JavaScript library used to build user interfaces (UIs) and single-page applications (SPAs).
- It is a component-based front-end library that handles only the **view** part of a web application. component is a small, reusable block that builds parts of a website.

❖ History (2011-2013)

- React was developed by **Jordan Walke** at Facebook and first used in the News Feed in 2011 to improve UI performance. 2013 public for everyone

❖ Features of React JS (Short Version)

1. **Component-Based** – Build UI using reusable components.
2. **Virtual DOM** – Faster updates using a virtual DOM.
3. **Declarative Syntax** – Easy to design UIs by describing what you want.
4. **Unidirectional Data Flow** – Data flows one-way, reducing bugs.
5. **JSX** – Write HTML-like code inside JavaScript.
6. **State & Props** – Manage dynamic data and pass values between components.
7. **Lifecycle Methods** – Control component behavior at different stages.
8. **Conditional Rendering** – Show UI parts based on state.
9. **Event Handling** – Handle user actions like clicks and inputs.
10. **Server-Side Rendering (SSR)** – Improve load time and SEO.
11. **React Native** – Build mobile apps using React.
12. **Large Ecosystem** – Plenty of tools, libraries, and community support.

❖ How to Setup React Environment

1. Install Node.js and npm

- Download from nodejs.org
- Verify: node -v and npm -v

2. Install a Code Editor

- Recommended: VS Code

3. How to install react js

- **Install globally:** npm install -g create-react-app

4. How to Create a React App

- **Create app using:** npx create-react-app appname

Note: Using npx is recommended because it runs tool without installing it globally.

- Alternatively, install create-react-app globally once: npm install -g create-react-app

- **Then create the app by running:** create-react-app appname

- **Change to the project folder:** cd appname

5. How to run the react js app

➤ npm start

➤ open this on browser

- Local: <http://localhost:3000>

- On Your Network: <http://172.16.50.150:3000>

Must-Have Extensions for React JS

1. **ES7+ Snippets** – Quick React shortcuts
2. **Prettier** – Auto code formatting
3. **ESLint** – Error checking
4. **Auto Rename Tag** – Auto update JSX tags

6. How to stop the react: Ctrl + C

7. How to install Install Webpack

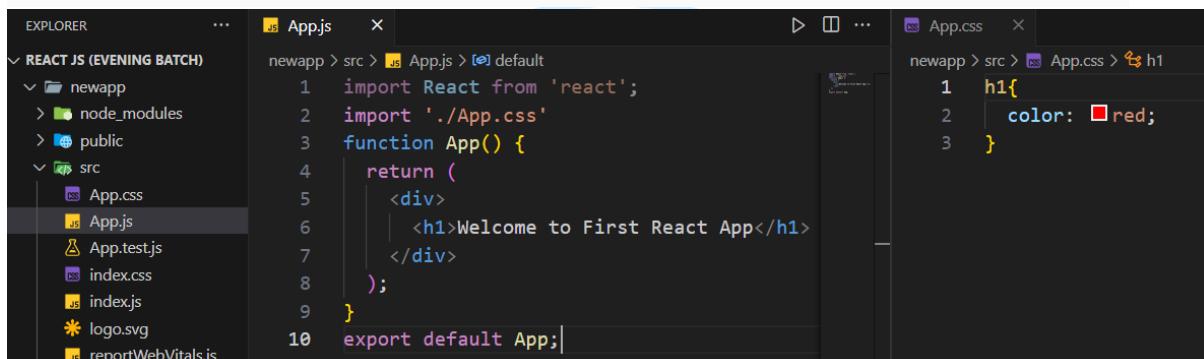
> npm install -g webpack for check its version runs this command: **webpack -v** then press **yes**

- Webpack is the "smart packager" that organizes, optimizes, and ships your React app efficiently to the browser. (Just like Amazon bundles your orders into a single box instead of sending 20 separate packages!)



❖ React JS Folder & File Structure:

```
my-react-app/
|
└── node_modules/          # Installed packages (auto-generated)
|
└── public/
    ├── index.html          # Main HTML file (entry point)
    └── favicon.ico         # Website icon
|
└── src/
    ├── components/          # React components folder
    │   └── Example.js        # Example component file
    ├── App.js                # Main app component (root component)
    ├── index.js              # Entry point for React (renders App.js)
    └── index.css             # Global styles
|
└── package.json            # Project config and dependencies
└── package-lock.json       # Locks dependency versions
└── README.md               # Project documentation
```



❖ How to change the React app icon and name

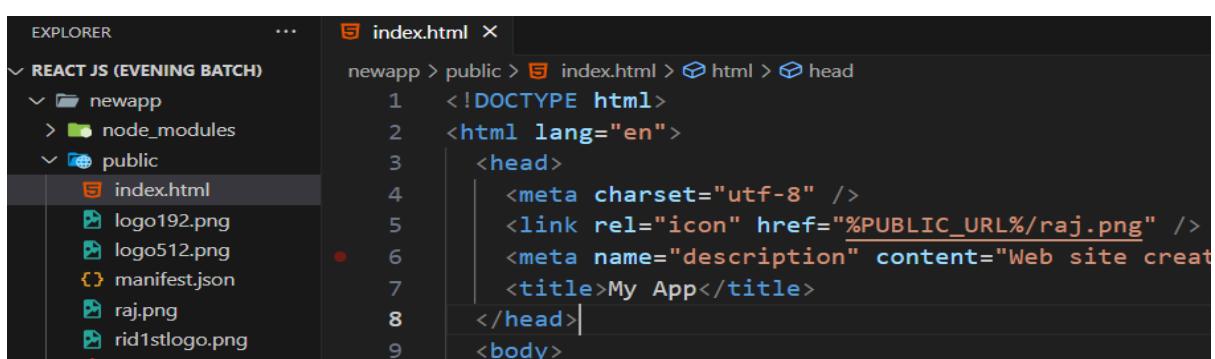
Step-1: Change App Icon:

- Replace public/favicon.ico with your new icon file (same name or update reference).
- Or update public/index.html <link rel="icon" href="path-to-new-icon" />.
- Ex: <link rel="icon" href="%PUBLIC_URL%/raj.png" />

Step-2: Change App Name:

- Open public/index.html. Modify <title> tag content to your new app name. Ex: <title>My App</title>

Step-3: Save and restart the app if running.



Components in React

Components are reusable, independent blocks of code that return HTML using JavaScript. There are two main types:

- **Functional Components** – simple functions that return JSX.
 - **Class Components** – use ES6 classes, can hold state and lifecycle methods (less common now).

How to create functional Based Component

Step-1: - Go to the src/ folder.

Step-2: - Create a new file: → ComponentName.jsx or (.js) extension

Difference Between .jsx and .js in React:

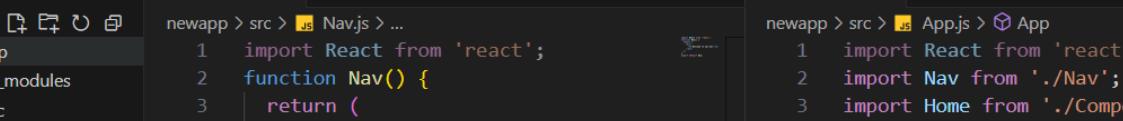
- **.js (JavaScript file):** Regular JavaScript file. Can include React code, but JSX is optional.
 - **.jsx (JavaScript XML file):** Specifically used when your file contains **JSX syntax** (HTML-like tags inside JS)

Rules for Naming Components in React:

1. **Always Start Component Names with a Capital Letter**
 - **Correct:** MyComponent, NavBar, App, **Incorrect:** mycomponent, navbar, app
 2. **Why Capital Letters?** :- React treats components starting with uppercase as custom components. Names starting with lowercase are treated as HTML tags.

Step-3: Import React (Optional in new versions) → import React from 'react';

Step-4: Write Functional Component



The screenshot shows the VS Code interface with two files open in the main editor area. The left file, 'Nav.js', contains the following code:

```
1 import React from 'react';
2 function Nav() {
3     return (
4         <div>
5             <h1>Welcome to nav bar</h1>
6         </div>
7     );
8 }
9 export default Nav;
```

The right file, 'App.js', contains the following code:

```
1 import React from 'react';
2 import Nav from './Nav';
3 import Home from './Compontes/Home';
4 function App() {
5     return (
6         <div>
7             <Nav />
8             <h1>Welcome to main page</h1>
9             <Home />
10        </div>
11    );
12 }
13 export default App;
```

```
File Edit Selection View Go ... < > React.js (Evening Batch) ... EXPLORE ... App.js ... REACT JS (E... newapp src App.js > Raj ... > node_modules > public > src > Componetes > Service.jsx > Raj 1 import Service from './Components/Service'; 2 function App() { 3     return ( 4         <div> 5             <Service /> 6             <h1>Welcome to main page</h1> 7             <Raj /> 8             <RID /> 9         </div> 10    );} 11    export default App; 12    function Raj() { 13        return ( 14            <h1>This is Raj function data</h1> 15        );} 16    const RID = () => { 17        return ( 18            <h1>This is Arrow data</h1> 19        );} ... Service.jsx ... newapp src Componetes Service.jsx > Service 1 import React from 'react'; 2 //To import Service.css from a folder 3 // import '../allcss/Service.css'; 4 // .. means go up one folder from com 5 // Then go into the allcss folder 6 // Then import the Service.css file 7 import './Service.css'; // Step 3: Im 8 export default function Service() { 9 10     return ( 11         <div className="card"> /* Step 4 12             <h3>Our Service</h3> 13             <p>We provide the best 14                 solutions for you.</p> 15         </div> 16     );} 17 ...
```

Note: - Always match filename, function name, and export. It's a React standard.

- **export default**: Sends the component to other files. **import**: Brings a component or module into a file.
 - **import React from 'react'** **Needed** for JSX (optional in new versions). **Note**: Use export to share, import to use.

Short key: - for create the functional based components in vs code enter → **rfce**

How to create class Based Component

Step 1: Import React

- To use class-based components, you must import React and Component.
- Ex: import React, {Component} from 'react';

Step 2: Create a Class Component

- Use the class keyword and extend it from Component.

Example:

```
class class_name extends Component {
  render() {
    return <h1>Hello, welcome to react js class </h1>;
  }
}
```

Note:

- render() is a **required method** in class components.
- The component must return **only one parent element**

Step 3: Export the Component: - So it can be used in other files. Ex:- export default class_name;

Step 4: Use the Component in App.js:- You can now import and use it in your App.js like this:

```
import React from 'react';
import class_name from './compontes_name';
function App() {
  return (
    <div>
      <class_name />
      <h1>welcome to main page </h1>
    </div>
  );
}
export default App;
```

Example:

Functional vs Class-Based Components (React)

Feature	Functional Component	Class Component
Syntax	Simple function	Uses class + extends
State	useState (Hooks)	this.state
Lifecycle	useEffect	componentDidMount
'this' keyword	Not needed	Required
Code	Short and clean	More complex
Hooks Support	Yes	No
Modern Use	Recommended	Outdated

--> Use Functional Components in modern React.

```
newapp > src > Mission.js > Mission > render
1 import React, { Component } from 'react'
2
3 class Mission extends Component {
4   render() {
5     return (
6       <div>
7         <h1>This is class based Components</h1>
8       </div>
9     );
10   }
11 }
12
13 export default Mission
```

```
newapp > src > App.js > default
1 import React from 'react';
2 import Mission from './Mission';
3 function App() {
4   return (
5     <div>
6       <Mission />
7       <h1>Welcome to main page</h1>
8     </div>
9   );
10 }
11
12 export default App;
```

```
newapp > src > App.js > ...
1 import Mission from './Mission';
2
3 function App() {
4   return (
5     <div>
6       <Mission />
7       <Service />
8       <h1>Welcome to main page</h1>
9       <Raj />
10      <RID />
11     </div>
12   );
13
14 export default App;
15
16 function Raj() {
17   return (
18     <h1>This is Raj function data</h1>
19   );
20 }
21 const RID = () => { // Arrow function component
22   return (
23     <h1>This is Arrow data</h1>
24   );
25 }
```



How to apply css in React components

We can apply css in react components in three ways

1. Inline **styles** (style prop)
2. Internal CSS (inject <style> from component)
3. External CSS (global or per-component)

1. Inline styles (style prop)

Steps 1. Create a JS object: const **s** = { backgroundColor: 'lightblue', padding: 8 }.

2. Use it: <div style={**s**}>Hello</div>.

3. Directly apply **inline styles** inside the tag in React by using double curly braces {{}}

Example: <div style={{ backgroundColor: 'lightblue', padding: '8px' }}>Research </div>

Note: Use camelCase for CSS properties (backgroundColor not background-color).

Values are usually strings (e.g., '8px'), unless unitless (e.g., padding: 8 for pixels).

2. Internal CSS (inject <style> from component)

Steps 1. Create a CSS string: const css = '.card{box-shadow:0 1px 3px rgba(0,0,0,.1)}'.

2. Render it: <style>{css}</style> somewhere in the component.

3. Use className="card".

Note: Injects global styles at render time — avoid repeated injection on many mounts.

Example:

```
export default function Profile() {
  const css = `
    .card {
      box-shadow: 0 1px 3px red;
      border-radius: 8px;
      padding: 16px;
      width: 250px;
      background: white;
    }
  `
  return (
    <div>
      <style>{css}</style>
      <div className="card">
        <h3>Sangam Kumar</h3>
        <p>Class X student</p>
      </div>
    </div>
  )
}
```

Note: In JSX, **className** is the correct attribute to add CSS classes because JSX is JavaScript, and class is reserved in JS. React will handle converting className → class in the actual HTML.

Rules of css unit

1. **Pixel values** → You can use **numbers** (React adds px automatically).
2. **Other units** (% em, rem, deg, vh, vw, etc.) → Must be **strings** with the unit.
3. **Colors, gradients, transforms, box-shadows** → Always strings.
4. **Unitless CSS properties** (like lineHeight, opacity, zIndex, flex) → Use numbers.

3. External CSS (global or per-component)

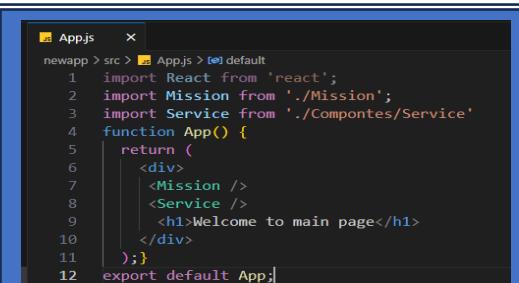
Steps 1. Create src/styles.css (or src/components/Profile.css).

2. Put rules like .card { padding: 12px; }.

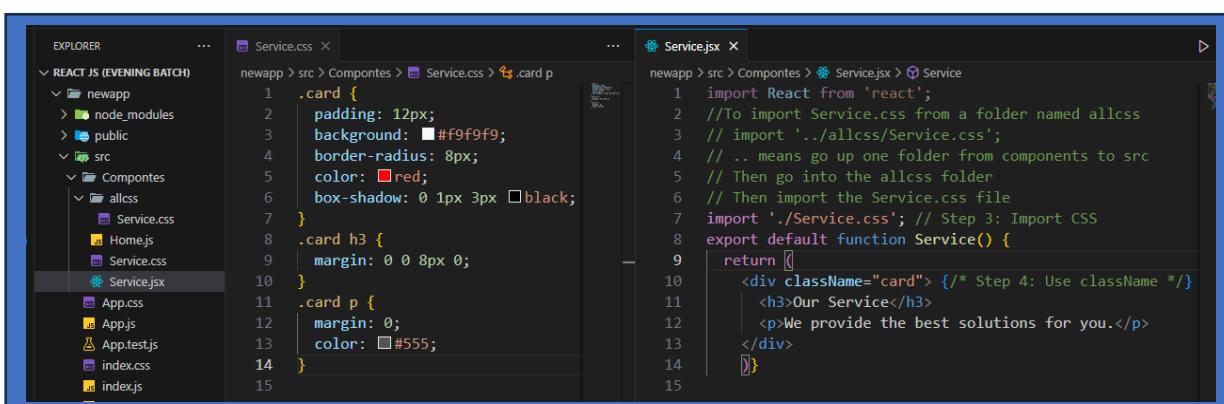
3. Import it (import './styles.css') in src/index.js (global) or inside component file (scoped-by-file).

4. Use className="card" in JSX.

Example:



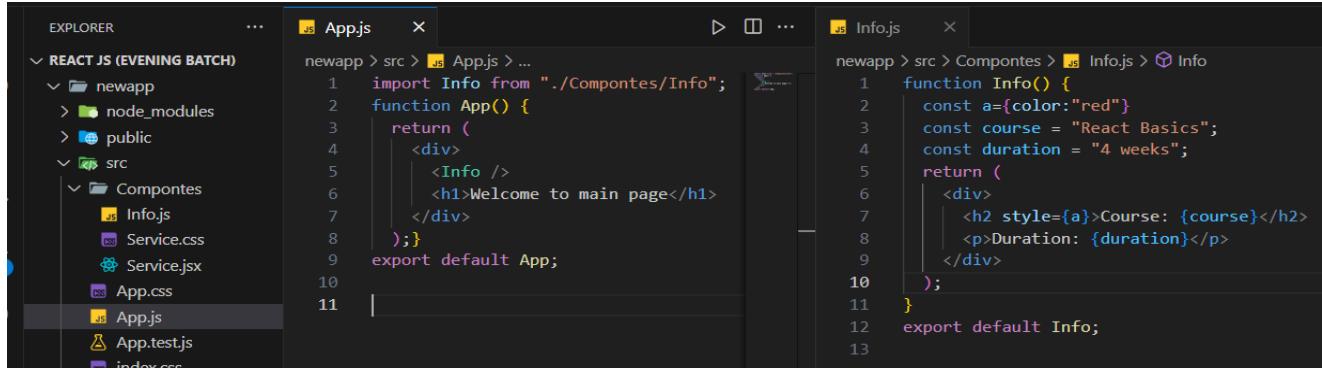
```
newapp > src > App.js > default
1 import React from 'react';
2 import Mission from './Mission';
3 import Service from './Components/Service';
4 function App() {
5   return (
6     <div>
7       <Mission />
8       <Service />
9       <h1>Welcome to main page</h1>
10    );
11 }
12 export default App;
```



How to display JavaScript code in React components

- **Functional component** in React is a JavaScript function that returns JSX
- We can display static or constant JavaScript data in a React component by using curly braces {} inside JSX.

Example-1: - Display simple js variable data in React JSX file



```

EXPLORER
REACT JS (EVENING BATCH)
  newapp
  node_modules
  public
  src
    Compontes
      Info.js
      Service.css
      Service.jsx
      App.css
      App.js
      App.test.js
      index.css

App.js
  1 import Info from "./Compontes/Info";
  2 function App() {
  3   return (
  4     <div>
  5       <Info />
  6       <h1>Welcome to main page</h1>
  7     </div>
  8   );
  9   export default App;
10
11

Info.js
  1 function Info() {
  2   const a={color:"red"}
  3   const course = "React Basics";
  4   const duration = "4 weeks";
  5   return (
  6     <div>
  7       <h2 style={a}>Course: {course}</h2>
  8       <p>Duration: {duration}</p>
  9     </div>
10   );
11   export default Info;
12
13

```

MAP () and Filter () in React JS

1. map () in React JS

- **What it does:** Loops through an array, transforms each item into something new (often JSX elements) Returns a new array of transformed items
- **Why use it?** - To render lists of data dynamically in React.
- **Syntax:**

```
arr_Name.map((currentValue, index, array) => {
  // return transformed element
});
```

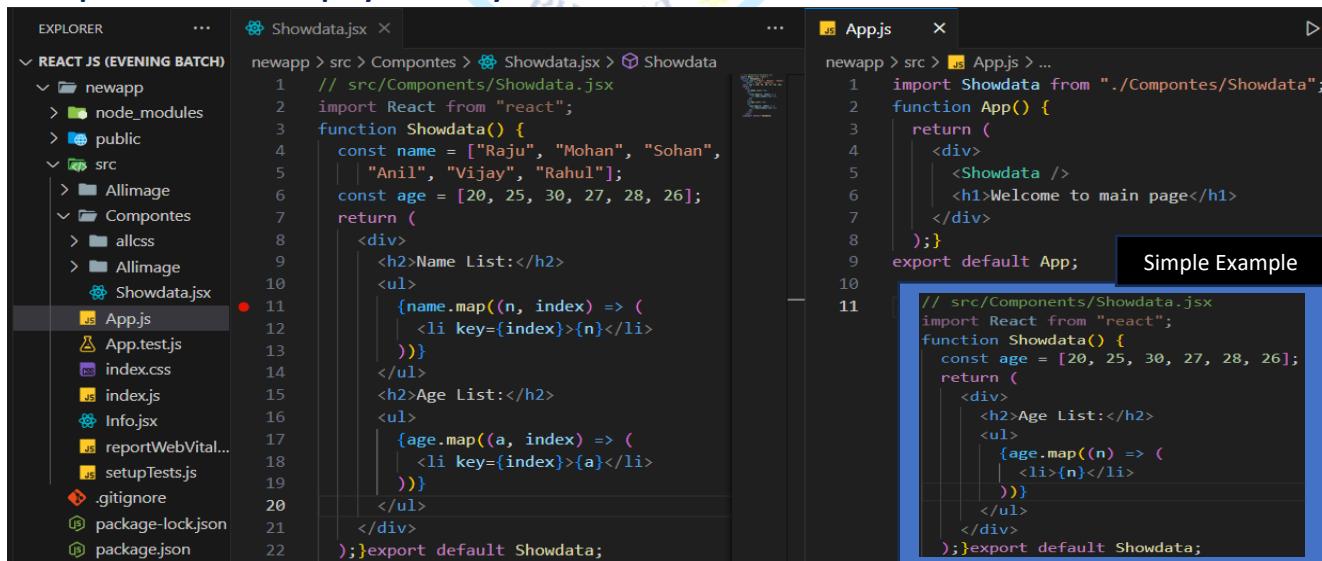
➤ Example-1:

```
const a = [1, 2, 3];
const doubled = a.map((num) => num * 2);
// doubled = [2, 4, 6]
```

- **Arr_name:** array you want to iterate over.
- **currentValue:** current item being processed in array.
- **index (optional):** index of the current item.
- **array (optional):** original array being traversed.
- **function** inside map **returns a new element** for each item.
- **result** is a **new array** of transformed elements.

- **In jsx :** a.map((num, index) => <li key={index}>{num});

Example-2: - How to display the array value in JSX file



```

EXPLORER
REACT JS (EVENING BATCH)
  newapp
  node_modules
  public
  src
    Allimage
    Compontes
      allcss
      Allimage
      Showdata.jsx
      App.js
      App.test.js
      index.css
      Info.js
      reportWebVital...
      setupTests.js
      .gitignore
      package-lock.json
      package.json

Showdata.jsx
  1 // src/Components>Showdata.jsx
  2 import React from "react";
  3 function Showdata() {
  4   const name = ["Raju", "Mohan", "Sohan", "Anil", "Vijay", "Rahul"];
  5   const age = [20, 25, 30, 27, 28, 26];
  6   return (
  7     <div>
  8       <h2>Name List:</h2>
  9       <ul>
10         {name.map((n, index) => (
11           <li key={index}>{n}</li>
12         )));
13       </ul>
14       <h2>Age List:</h2>
15       <ul>
16         {age.map((a, index) => (
17           <li key={index}>{a}</li>
18         )));
19       </ul>
20     </div>
21   );
22   export default Showdata;
23

App.js
  1 import Showdata from "./Compontes>Showdata";
  2 function App() {
  3   return (
  4     <div>
  5       <Showdata />
  6       <h1>Welcome to main page</h1>
  7     </div>
  8   );
  9   export default App;
10
11

```

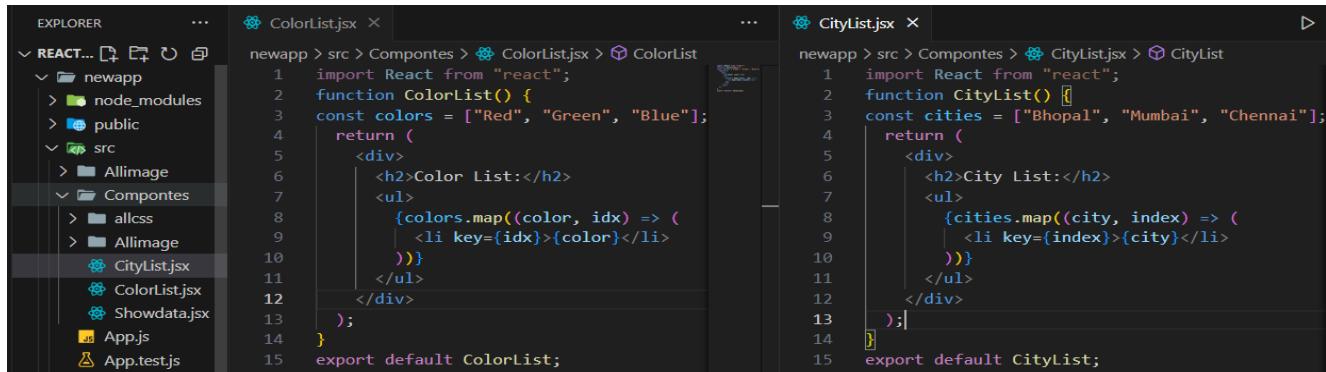
Simple Example

```
// src/Components>Showdata.jsx
import React from "react";
function Showdata() {
  const age = [20, 25, 30, 27, 28, 26];
  return (
    <div>
      <h2>Age List:</h2>
      <ul>
        {age.map((n) => (
          <li>{n}</li>
        )));
      </ul>
    </div>
  );
  export default Showdata;
```

- **name.map(...)** Loops over each element in the name array.
- **(n, index) => ...** For each element, n is the current name, index is its position in the array.
- **<li key={index}>...** Returns a list item () for each name.
- **key={index}** key helps React identify each list item uniquely to optimize rendering.
- **{n} inside ** Displays the current name inside the list item.
- **Note: key is a special prop in React** used to uniquely identify elements in lists. It's **not a JavaScript reserved keyword**,



Example-3: How to display the array value in JSX file



```

ColorList.jsx
newapp > src > Compontes > ColorList.jsx > ColorList
1 import React from "react";
2 function ColorList() {
3   const colors = ["Red", "Green", "Blue"];
4   return (
5     <div>
6       <h2>Color List:</h2>
7       <ul>
8         {colors.map((color, idx) => (
9           <li key={idx}>{color}</li>
10        )));
11      </ul>
12    );
13  }
14  export default ColorList;

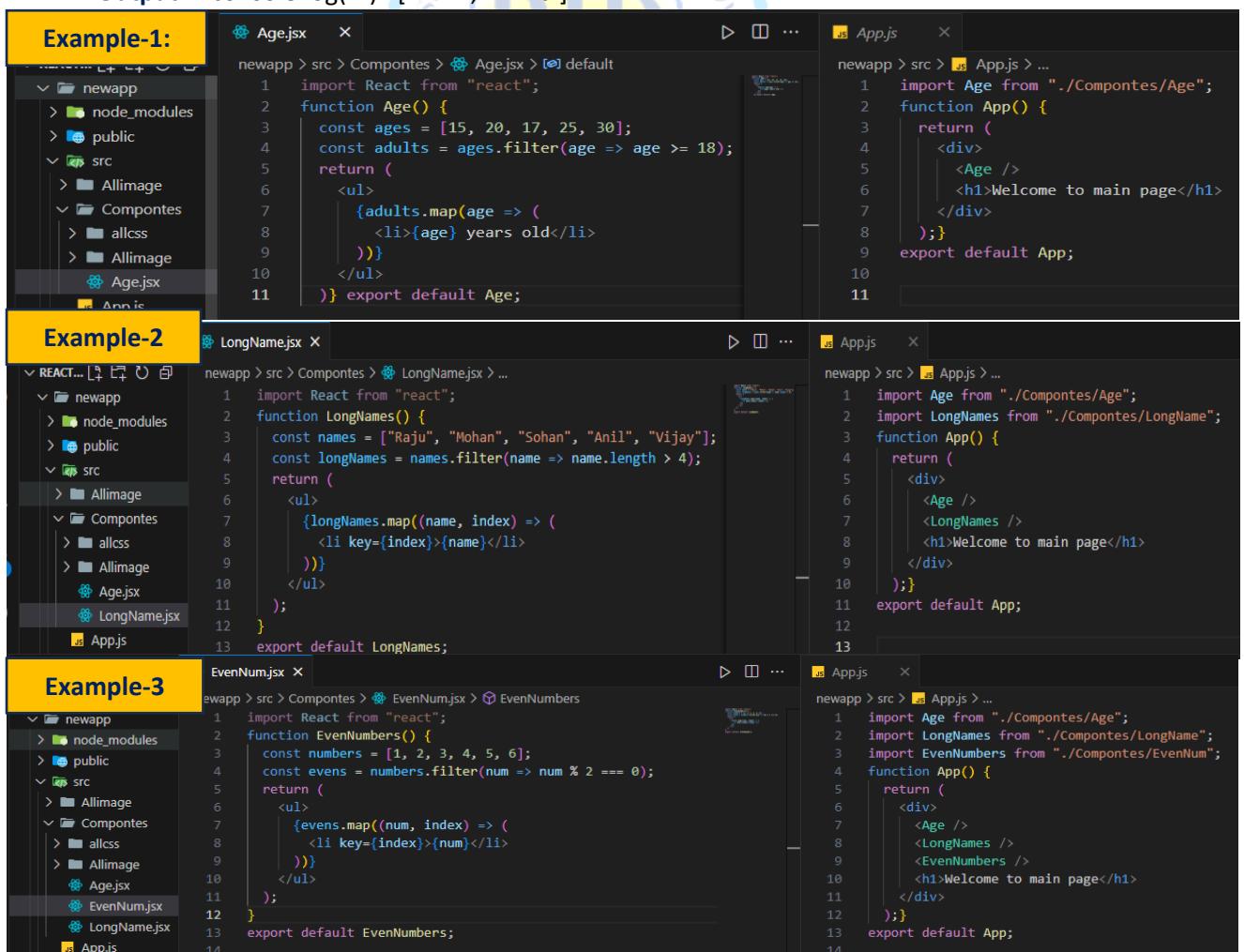
CityList.jsx
newapp > src > Compontes > CityList.jsx > CityList
1 import React from "react";
2 function CityList() {
3   const cities = ["Bhopal", "Mumbai", "Chennai"];
4   return (
5     <div>
6       <h2>City List:</h2>
7       <ul>
8         {cities.map((city, index) => (
9           <li key={index}>{city}</li>
10        )));
11      </ul>
12    );
13  }
14  export default CityList;

```

2. filter () in React JS

- **What it does:** Loops through an array Returns a new array containing only items that pass a test (condition)
- **Why use it?** To show only certain elements from a list based on some condition.
- **Syntax:** `const newArray = arr_Name.filter((element, index, array) => {
 // return true to keep element, false to exclude it
});`
- **Ex-1:** `const n = [5, 12, 8, 20, 3];
const fn = n.filter(num => num > 10); O/P: [12, 20]`
- **Ex-2:** `const n = ["Anil", "Raju", "Amit", "Sohan"];
const fn = n.filter(name => name.startsWith("A"));`
- **Output:** `console.log(fn) = ["Anil", "Amit"]`

- **array** — original array you want to filter
- **element** — current item in the array
- **index (optional)** — current item's index
- **Returns** a new array with only elements passing the condition



```

Example-1: Age.jsx
newapp > src > Compontes > Age.jsx > default
1 import React from "react";
2 function Age() {
3   const ages = [15, 20, 17, 25, 30];
4   const adults = ages.filter(age => age >= 18);
5   return (
6     <ul>
7       {adults.map(age => (
8         <li>{age} years old</li>
9       )));
10      </ul>
11    );
12  }
13  export default Age;

Example-2: LongName.jsx
newapp > src > Compontes > LongName.jsx > ...
1 import React from "react";
2 function LongNames() {
3   const names = ["Raju", "Mohan", "Sohan", "Anil", "Vijay"];
4   const longNames = names.filter(name => name.length > 4);
5   return (
6     <ul>
7       {longNames.map((name, index) => (
8         <li key={index}>{name}</li>
9       )));
10      </ul>
11    );
12  }
13  export default LongNames;

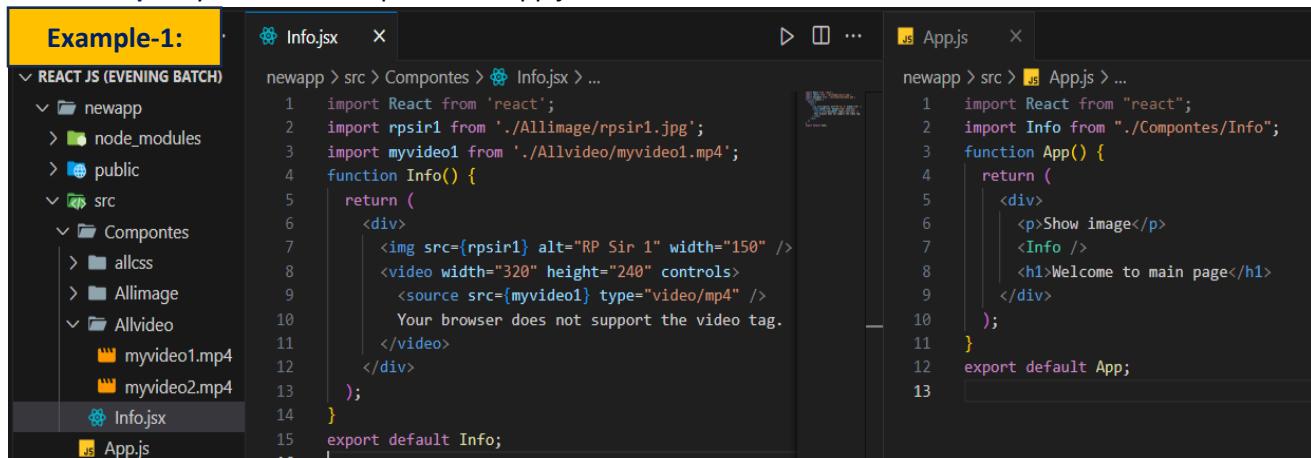
Example-3: EvenNum.jsx
newapp > src > Compontes > EvenNum.jsx > EvenNumbers
1 import React from "react";
2 function EvenNumbers() {
3   const numbers = [1, 2, 3, 4, 5, 6];
4   const evens = numbers.filter(num => num % 2 === 0);
5   return (
6     <ul>
7       {evens.map((num, index) => (
8         <li key={index}>{num}</li>
9       )));
10      </ul>
11    );
12  }
13  export default EvenNumbers;

```

How to display Image and video in React components

1. Put image/video files in your project folder.
2. Import them in your React component.
3. Use `` for images.
4. Use `<video controls><source src={video} type="video/mp4" /></video>` for videos.
5. Add alt for images and controls for videos.
6. Import your React component in App.js

Example-1:



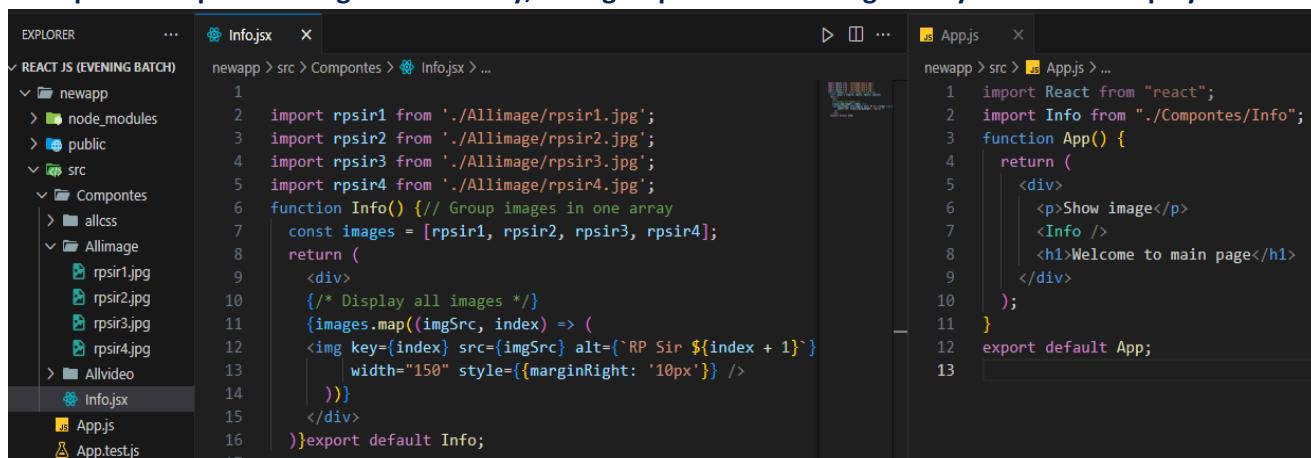
```

Info.jsx
1 import React from 'react';
2 import rpsir1 from './Allimage/rpsir1.jpg';
3 import myvideo1 from './Allvideo/myvideo1.mp4';
4 function Info() {
5   return (
6     <div>
7       <img src={rpsir1} alt="RP Sir 1" width="150" />
8       <video width="320" height="240" controls>
9         <source src={myvideo1} type="video/mp4" />
10        Your browser does not support the video tag.
11      </div>
12    );
13  }
14
15 export default Info;
16

App.jsx
1 import React from "react";
2 import Info from "./Compontes/Info";
3 function App() {
4   return (
5     <div>
6       <p>Show image</p>
7       <Info />
8       <h1>Welcome to main page</h1>
9     </div>
10  );
11 }
12 export default App;
13

```

Example-2: - import all images individually, then group them into a single array variable to display



```

Info.jsx
1 import rpsir1 from './Allimage/rpsir1.jpg';
2 import rpsir2 from './Allimage/rpsir2.jpg';
3 import rpsir3 from './Allimage/rpsir3.jpg';
4 import rpsir4 from './Allimage/rpsir4.jpg';
5 function Info() { // Group images in one array
6   const images = [rpsir1, rpsir2, rpsir3, rpsir4];
7   return (
8     <div>
9       /* Display all images */
10      {images.map((imgSrc, index) => (
11        <img key={index} src={imgSrc} alt={`RP Sir ${index + 1}`}
12          width="150" style={{marginRight: '10px'}} />
13      ))}
14    </div>
15  );
16  export default Info;
17

App.jsx
1 import React from "react";
2 import Info from "./Compontes/Info";
3 function App() {
4   return (
5     <div>
6       <p>Show image</p>
7       <Info />
8       <h1>Welcome to main page</h1>
9     </div>
10  );
11 }
12 export default App;
13

```

Ex-3: How to imports all image in one file & then import them all together from that file.

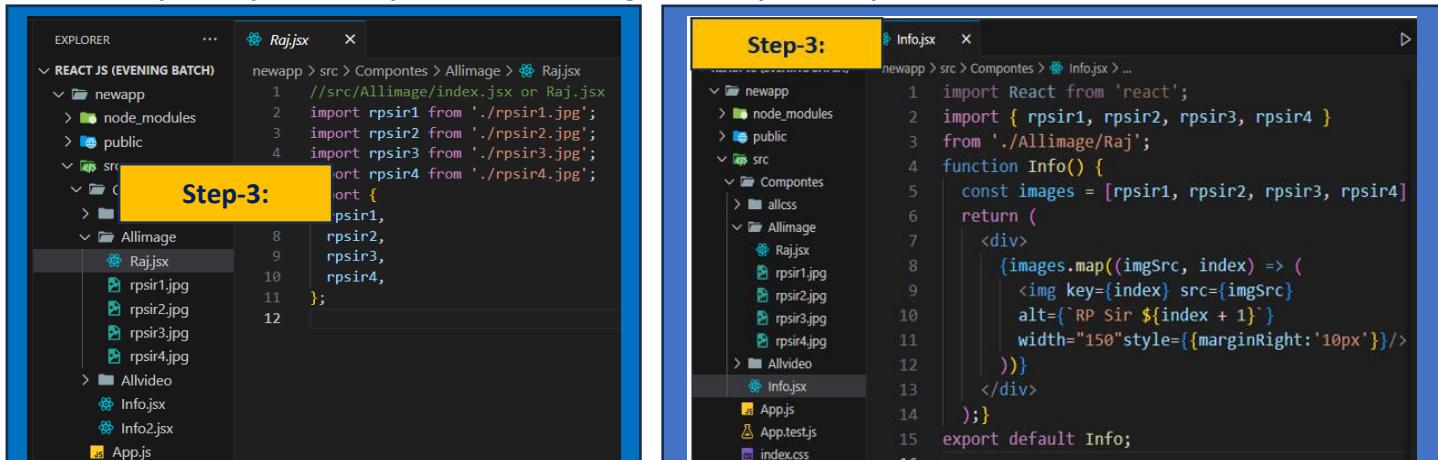
Step-1: Create a file: src/Allimage/index.js or any specific name or src/Allimage/Raj.jsx:

Why index.js is preferred?

- When you import from a folder path like `import { rpsir1 } from './Allimage'`; the system automatically looks for `./Allimage/index.js` by default.

Step-2: import and export the whole image Then in your component code:

Step-3:



```

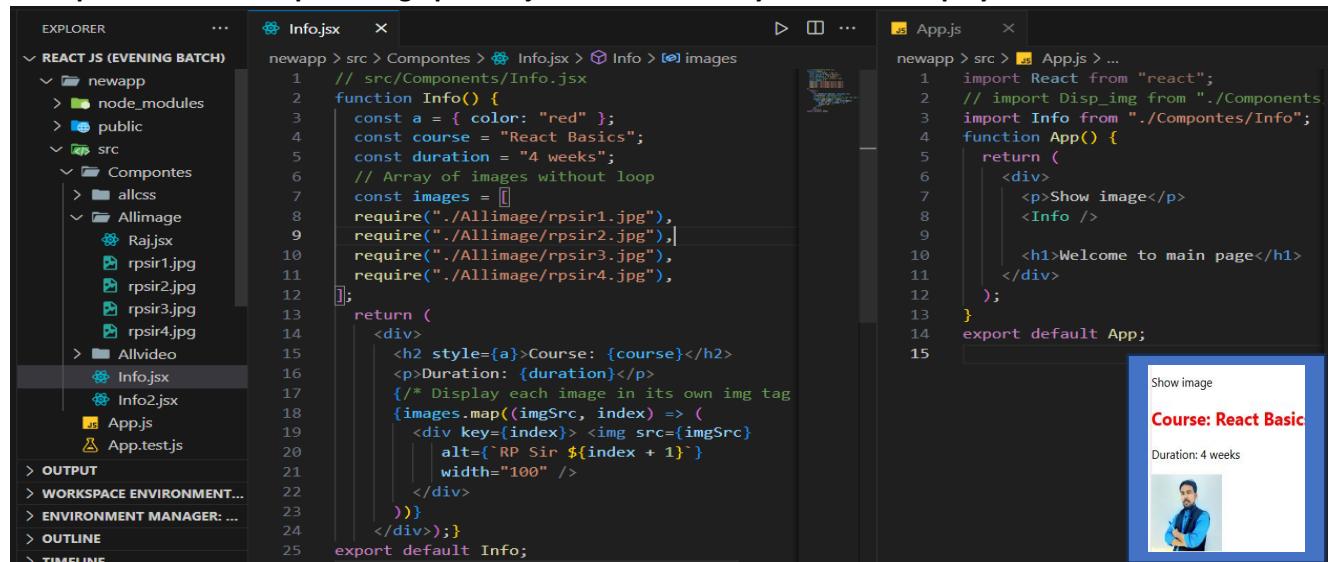
Raj.jsx
1 //src/Allimage/index.js or Raj.jsx
2 import rpsir1 from './rpsir1.jpg';
3 import rpsir2 from './rpsir2.jpg';
4 import rpsir3 from './rpsir3.jpg';
5 import rpsir4 from './rpsir4.jpg';
6 export {
7   rpsir1,
8   rpsir2,
9   rpsir3,
10  rpsir4,
11};

Info.jsx
1 import React from 'react';
2 import { rpsir1, rpsir2, rpsir3, rpsir4 } from './Allimage/Raj';
3 function Info() {
4   const images = [rpsir1, rpsir2, rpsir3, rpsir4];
5   return (
6     <div>
7       {images.map((imgSrc, index) => (
8         <img key={index} src={imgSrc}
9           alt={`RP Sir ${index + 1}`}
10          width="150" style={{marginRight: '10px'}} />
11      ))}
12    </div>
13  );
14  export default Info;
15

```



Example-4: - How to import image path in js variable from any folder and display on front end



```

REACT JS (EVENING BATCH)
  newapp
  node_modules
  public
  src
    Compontes
      allcss
      Allimage
        Raj.jsx
        rpsir1.jpg
        rpsir2.jpg
        rpsir3.jpg
        rpsir4.jpg
      Allvideo
        Info.jsx
        Info.jsx
        App.jsx
        App.test.js
    OUTPUT
    WORKSPACE ENVIRONMENT...
    ENVIRONMENT MANAGER: ...
    OUTLINE
    TIMELINE

Info.jsx
newapp > src > Compontes > Info.jsx > Info > images
1 // src/Components/Info.jsx
2 function Info() {
3   const a = { color: "red" };
4   const course = "React Basics";
5   const duration = "4 weeks";
6   // Array of images without loop
7   const images = []
8   require("./Allimage/rpsir1.jpg"),
9   require("./Allimage/rpsir2.jpg"),
10  require("./Allimage/rpsir3.jpg"),
11  require("./Allimage/rpsir4.jpg"),
12  ];
13  return (
14    <div>
15      <h2 style={a}>Course: {course}</h2>
16      <p>Duration: {duration}</p>
17      /* Display each image in its own img tag
18      {images.map((imgSrc, index) => (
19        <div key={index}> <img src={imgSrc}
20          alt="RP Sir ${index + 1}" width="100" />
21        </div>
22      ))}
23    </div>);
24  }
25  export default Info;

```

```

App.jsx
newapp > src > App.jsx ...
1 import React from "react";
2 // import Disp_img from "./Components/Disp_img";
3 import Info from "./Compontes/Info";
4 function App() {
5   return (
6     <div>
7       <p>Show image</p>
8       <Info />
9
10      <h1>Welcome to main page</h1>
11    </div>
12  );
13 }
14 export default App;

```

Show image

Course: React Basic

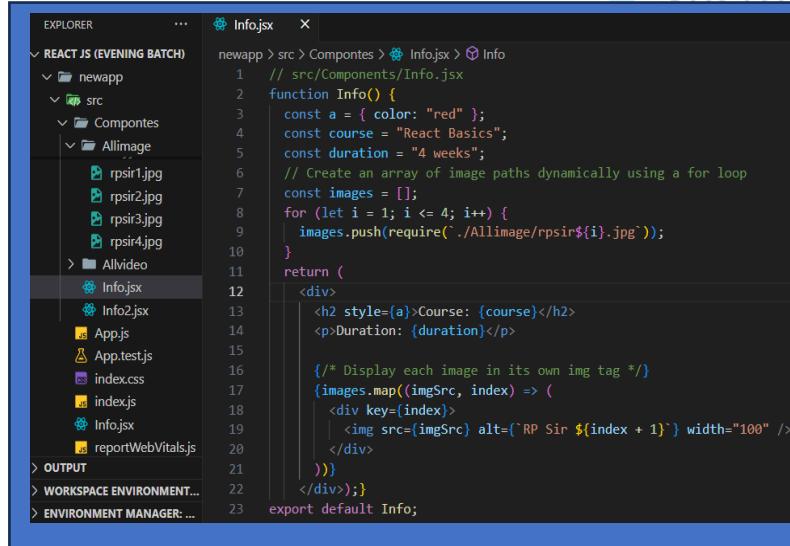
Duration: 4 weeks



Example: 4

How to using a for loop and require() dynamically for image path .

If your images don't have a numbered pattern (like mohan.jpg, sohan.jpg, raju.jpg), then you can't use a for loop with i — you'll need to store the file names in an array and loop through them instead.

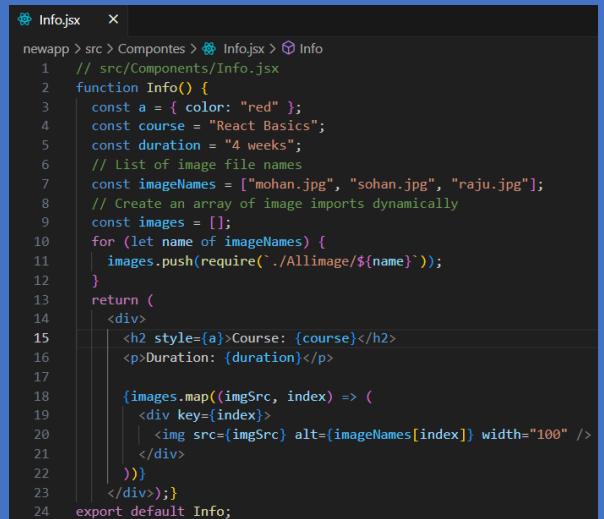


```

REACT JS (EVENING BATCH)
  newapp
  src
    Compontes
      Allimage
        rpsir1.jpg
        rpsir2.jpg
        rpsir3.jpg
        rpsir4.jpg
      Allvideo
        Info.jsx
        Info2.jsx
        App.jsx
        App.test.js
        index.css
        index.js
        Info.jsx
        reportWebVitals.js
    OUTPUT
    WORKSPACE ENVIRONMENT...
    ENVIRONMENT MANAGER: ...

Info.jsx
newapp > src > Compontes > Info.jsx > Info
1 // src/Components/Info.jsx
2 function Info() {
3   const a = { color: "red" };
4   const course = "React Basics";
5   const duration = "4 weeks";
6   // Create an array of image paths dynamically using a for loop
7   const images = [];
8   for (let i = 1; i <= 4; i++) {
9     images.push(require(`./Allimage/rpsir${i}.jpg`));
10  }
11  return (
12    <div>
13      <h2 style={a}>Course: {course}</h2>
14      <p>Duration: {duration}</p>
15
16      /* Display each image in its own img tag */
17      {images.map((imgSrc, index) => (
18        <div key={index}> <img src={imgSrc} alt="RP Sir ${index + 1}" width="100" />
19        </div>
20      ))}
21    </div>);
22  }
23  export default Info;

```



```

Info.jsx
newapp > src > Compontes > Info.jsx > Info
1 // src/Components/Info.jsx
2 function Info() {
3   const a = { color: "red" };
4   const course = "React Basics";
5   const duration = "4 weeks";
6   // List of image file names
7   const imageNames = ["mohan.jpg", "sohan.jpg", "raju.jpg"];
8   // Create an array of image imports dynamically
9   const images = [];
10  for (let name of imageNames) {
11    images.push(require(`./Allimage/${name}`));
12  }
13  return (
14    <div>
15      <h2 style={a}>Course: {course}</h2>
16      <p>Duration: {duration}</p>
17
18      {images.map((imgSrc, index) => (
19        <div key={index}> <img src={imgSrc} alt={imageNames[index]} width="100" />
20        </div>
21      ))}
22    </div>);
23  }
24  export default Info;

```

- a** → Inline CSS style object that makes text red.
- course / duration** → Variables storing course details.
- images array** → Filled using a for loop that imports (require) 4 images named rpsir1.jpg to rpsir4.jpg from the Allimage folder.
- map()** → Loops through the images array to display each image in its own <div>, giving each element a unique key and an alt text.

require() is a function used to load files or modules into your code.

In React (with Webpack), it's often used to import images or other assets dynamically when you can't use a normal import statement.

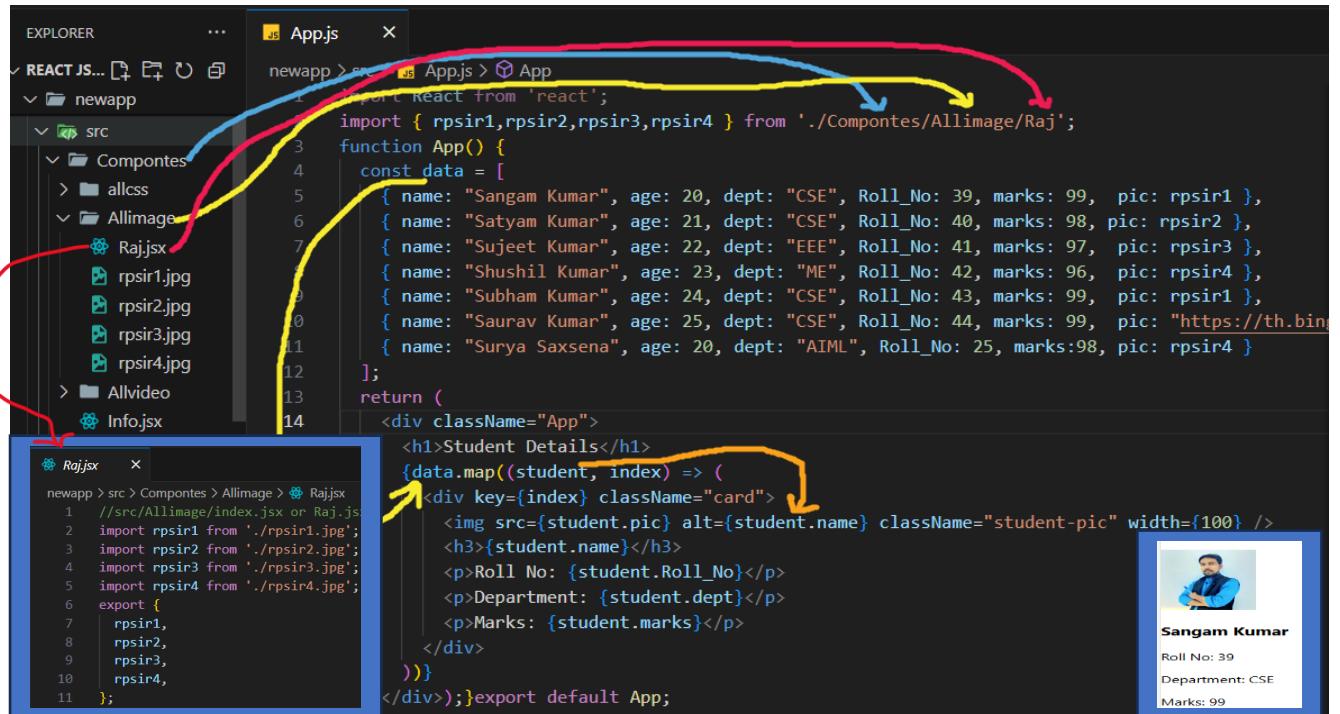
Example:

```
const img = require("./photo.jpg");
```

This tells Webpack to include photo.jpg in the build and gives you its URL to use in .



Example-5: How to Display the image with Basic Information's in react js



App.js

```

1  import React from 'react';
2  import { rpsir1,rpsir2,rpsir3,rpsir4 } from './Components/Allimage/Raj';
3  function App() {
4      const data = [
5          { name: "Sangam Kumar", age: 20, dept: "CSE", Roll_No: 39, marks: 99, pic: rpsir1 },
6          { name: "Satyam Kumar", age: 21, dept: "CSE", Roll_No: 40, marks: 98, pic: rpsir2 },
7          { name: "Sujeet Kumar", age: 22, dept: "EEE", Roll_No: 41, marks: 97, pic: rpsir3 },
8          { name: "Shushil Kumar", age: 23, dept: "ME", Roll_No: 42, marks: 96, pic: rpsir4 },
9          { name: "Subham Kumar", age: 24, dept: "CSE", Roll_No: 43, marks: 99, pic: rpsir1 },
10         { name: "Saurav Kumar", age: 25, dept: "CSE", Roll_No: 44, marks: 99, pic: rpsir1 },
11         { name: "Surya Saksena", age: 20, dept: "AIML", Roll_No: 25, marks:98, pic: rpsir4 }
12     ];
13     return (
14         <div className="App">
15             <h1>Student Details</h1>
16             {data.map((student, index) => (
17                 <div key={index} className="card">
18                     <img src={student.pic} alt={student.name} className="student-pic" width={100} />
19                     <h3>{student.name}</h3>
20                     <p>Roll No: {student.Roll_No}</p>
21                     <p>Department: {student.dept}</p>
22                     <p>Marks: {student.marks}</p>
23                 </div>
24             )));
25         </div>);}export default App;

```

Raj.jsx

```

1  //src/Allimage/index.jsx or Raj.jsx
2  import rpsir1 from './rpsir1.jpg';
3  import rpsir2 from './rpsir2.jpg';
4  import rpsir3 from './rpsir3.jpg';
5  import rpsir4 from './rpsir4.jpg';
6  export {
7      rpsir1,
8      rpsir2,
9      rpsir3,
10     rpsir4,
11 };

```



PROPS in React JS

- **Props:** - Props is a mechanism for passing data from a parent component to a child component.
- props are unidirectional we can send information from parent to child, It is similar to function arguments.
- **Props can be any JavaScript value** — primitive, object, array, function, JSX, another component, or even CSS styles — that we can pass from **one component** to another.
- **Note:** - Props are unidirectional in React — they flow **from parent to child** — but we *can* still send data from child to parent using a **callback function** passed down as a prop.

Props are read-only — you cannot modify a parent component's data directly through props, whether from the parent itself or from a child component.

Example: Parent–Child Conversation

Imagine a **mother** (parent component) telling her **child** (child component) what to do.

1. Props:

- Mother gives her child a shopping list: "Buy milk and bread".
- This shopping list is **props** — the data given from parent to child.

2. Unidirectional:

- The list only goes **from mother to child**, not the other way around.
- The child cannot change the mother's original list directly.

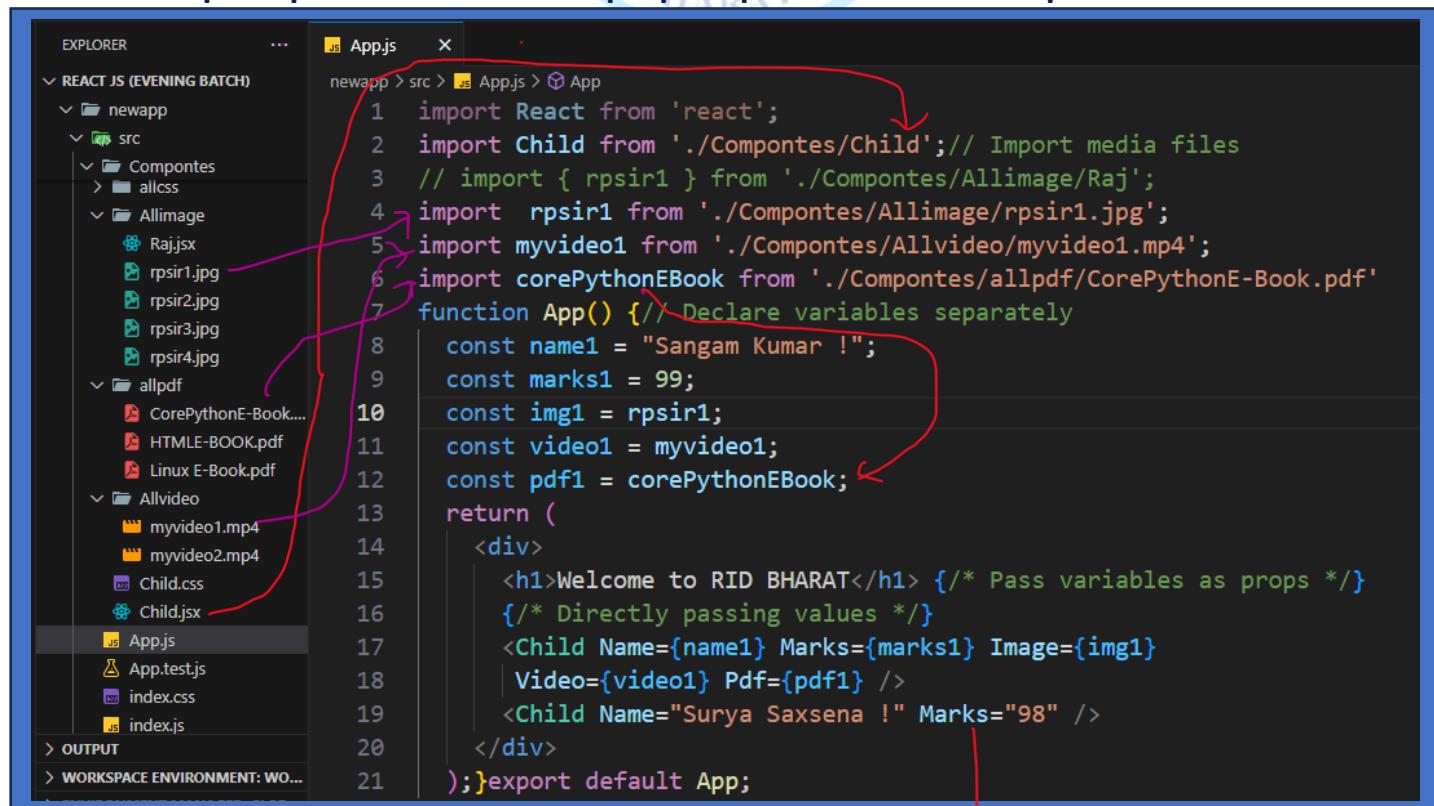
3. Callback function (sending data back):

- If the child wants to tell the mother "Milk was out of stock", the mother first gives the child a **notebook and pen** (callback function) and says, "*If you need to tell me something, write it here and give it back to me.*"
- This is like passing a function as a prop so the child can call it to send data back.

4. Read-only:

- The child **cannot erase or change** the original shopping list the mother gave.
- Props are the same — they can be used but not directly modified

Example-1 pass the Variable as props in parent to child Components



```

EXPLORER
REACT JS (EVENING BATCH)
  newapp
    SRC
      Compontes
        alcss
        Allimage
          Rajjsx
          rpsir1.jpg
          rpsir2.jpg
          rpsir3.jpg
          rpsir4.jpg
        allpdf
          CorePythonE-Book....
          HTMLE-BOOK.pdf
          Linux E-Book.pdf
      Allvideo
        myvideo1.mp4
        myvideo2.mp4
        Child.css
        Child.jsx
      App.js
      App.test.js
      index.css
      index.js
    OUTPUT
    WORKSPACE ENVIRONMENT: WO...
  App.js
    newapp > src > App.js > App
    1 import React from 'react';
    2 import Child from './Compontes/Child';// Import media files
    3 // import { rpsir1 } from './Compontes/Allimage/Raj';
    4 import rpsir1 from './Compontes/Allimage/rpsir1.jpg';
    5 import myvideo1 from './Compontes/Allvideo/myvideo1.mp4';
    6 import corePythonEBook from './Compontes/allpdf/CorePythonE-Book.pdf';
    7 function App() { // Declare variables separately
    8   const name1 = "Sangam Kumar !";
    9   const marks1 = 99;
   10  const img1 = rpsir1;
   11  const video1 = myvideo1;
   12  const pdf1 = corePythonEBook;
   13  return (
   14    <div>
   15      <h1>Welcome to RID BHARAT</h1> /* Pass variables as props */
   16      /* Directly passing values */
   17      <Child Name={name1} Marks={marks1} Image={img1}>
   18        <Child Name="Surya Saxsena !" Marks="98" />
   19      </Child>
   20    </div>
   21  );}export default App;
  
```

EXPLORER

REACT JS (EVENING BATCH)

- newapp
 - src
 - Compontes
 - allcss
 - Allimage
 - allpdf
 - CorePythonE-Book....
 - HTMLE-BOOK.pdf
 - Linux E-Book.pdf
 - Allvideo
 - myvideo1.mp4
 - myvideo2.mp4
 - Child.css
 - Child.jsx
 - App.js
 - App.test.js

```
Child.jsx  X
newapp > src > Compontes > Child.jsx > ...
1  import React from 'react';
2  import './Child.css';
3  export default function Child(props) {
4      return (
5          <div className="child-card">
6              /* Display Name and Marks iamge video and pdf  */
7              <h2>{props.Name} scored {props.Marks} marks</h2>
8              <img src={props.Image} alt="Student" />
9              <video src={props.Video} controls />
10             <p><a href={props.Pdf} download>Download PDF</a></p>
11         </div>
12     );
13 }
```

Welcome to RID BHARAT

Sangam Kumar !
scored 99 marks



[Download PDF](#)



Example-2: passing CSS properties as props from parents' components to child Components

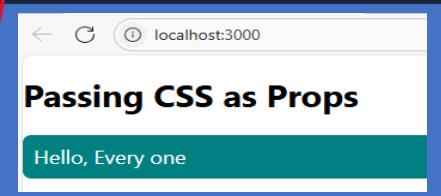
EXPLORER

REACT JS (EVENING BATCH)

- newapp
 - src
 - Compontes
 - App.js
 - App.test.js
 - Btn.jsx
 - Child.jsx
 - index.css
 - index.js
 - Info.jsx
 - reportWebVitals.js
 - setupTests.js
 - Child.js
 - index.css
 - index.js
 - Info.jsx
 - reportWebVitals.js
 - setupTests.js
 - .gitignore
 - package-lock.json
 - package.json
- OUTPUT
- WORKSPACE ENVIRONMENT: WO...

```
App.js  X
newapp > src > App.js > ...
1  import React from "react";
2  import Child from "./Child";
3  function App() {
4      const myStyles = {
5          color: "white",
6          backgroundColor: "teal",
7          padding: "10px",
8          borderRadius: "8px",
9          fontSize: "20px"
10     };
11     return (
12         <div>
13             <h1>Passing CSS as Props</h1>
14             /* Pass styles as a prop */
15             <Child styles={myStyles} />
16         </div>
17     );
18 }
19 export default App;
```

```
Child.js  X
newapp > src > Child.js > ...
1  import React from "react";
2
3  function Child({ styles }) {
4      return (
5          <div style={styles}>
6              Hello, Every one
7          </div>
8      );
9  }
11  export default Child;
```



Btn.jsx

```

1 import React from 'react';
2 function Btn(props) {
3   const buttonStyle = {
4     backgroundColor: props.bgColor,
5     color: props.color,
6     fontSize: props.fontSize,
7     padding: props.padding,
8     border: 'none',
9     borderRadius: '5px',
10    margin: '5px',
11    cursor: 'pointer'
12  };
13  return (
14    <button style={buttonStyle}>
15      {props.label}
16    </button>
17  );
18}
19 export default Btn;

```

App.js

```

1 import React from 'react';
2 import Btn from './Btn';
3 function App() {
4   return (
5     <div>
6       <Btn label="OK" color="white" bgColor="green" fontSize="18px" padding="10px" />
7       <Btn label="Login" color="white" bgColor="blue" fontSize="16px" padding="8px" />
8       <Btn label="Reset" color="white" bgColor="red" fontSize="16px" padding="8px" />
9       <Btn label="Submit" color="white" bgColor="orange" fontSize="20px" padding="12px" />
10    </div>
11  );
12}
13 export default App;

```

Example-3: Passing props parents to child and child to another child

1. App.js is parents
2. Heading.j is child components of App.js and parent Components of Btn.js
3. Btn.js is child components

App.js

```

1 import Heading from './Heading';
2 // import Btn from './Btn';
3 // Parent component of Heading and Btn
4 let App=()=>{
5   return (
6     <div>
7       <Heading name="Sangam Kumar" age="15" marks="99" />
8       <Heading name="Satyam Kumar" age="18" marks="98" />
9       <Heading name="Sujeet Kumar" age="21" marks="97" />
10      <Heading name="Shusil Kumar" age="24" marks="96" />
11      <Heading name="Suraj Kumar" age="27" marks="95" />
12      <Heading name="Saroj Kumar" age="30" marks="94" />
13      <Heading name="Surya saxsena" age="30" marks="94" />
14    </div>
15  );
16  export default App;

```

Heading.jsx

```

1 import Btn from './Btn';
2 // Child component of App.js
3 let Heading=(props)=>{
4   return(
5     <div>
6       <h3>Your Name is: {props.name}</h3>
7       <h4>Your Age is: {props.age}</h4>
8       <h4>Your Marks is: {props.marks}</h4>
9     </div>
10    <Btn label="Add" color="blue"/>
11    <Btn label="Delete" color="red"/>
12    <Btn label="Update" color="green"/>
13  );
14  export default Heading;

```

Btn.jsx

```

1 // Child component of heading
2 let Btn=(props)=>{
3   return(
4     <button style={{color: props.color}}> {props.label}</button>
5   );
6  export default Btn;

```

Example-4: passing an object as props in React JS

EXPLORER

App.js

```

1 import React from "react";
2 import Profile from "/Profile";
3 function App() {
4   // Create an object to pass as props
5   const userInfo = {
6     name: "Sangam Kumar",
7     age: 16,
8     country: "India"
9   };
10  return (
11    <div>
12      <h1>Passing Object as Props </h1>
13      {/* Pass the entire object as a prop */}
14      <Profile user={userInfo} />
15    </div>
16  );
17  export default App;

```

Profile.jsx

```

1 import React from "react";
2 // Create an object to pass as props
3 function Profile({ user }) {
4   return (
5     <div>
6       <h2>Name: {user.name}</h2>
7       <p>Age: {user.age}</p>
8       <p>Country: {user.country}</p>
9     </div>
10  );
11  export default Profile;

```

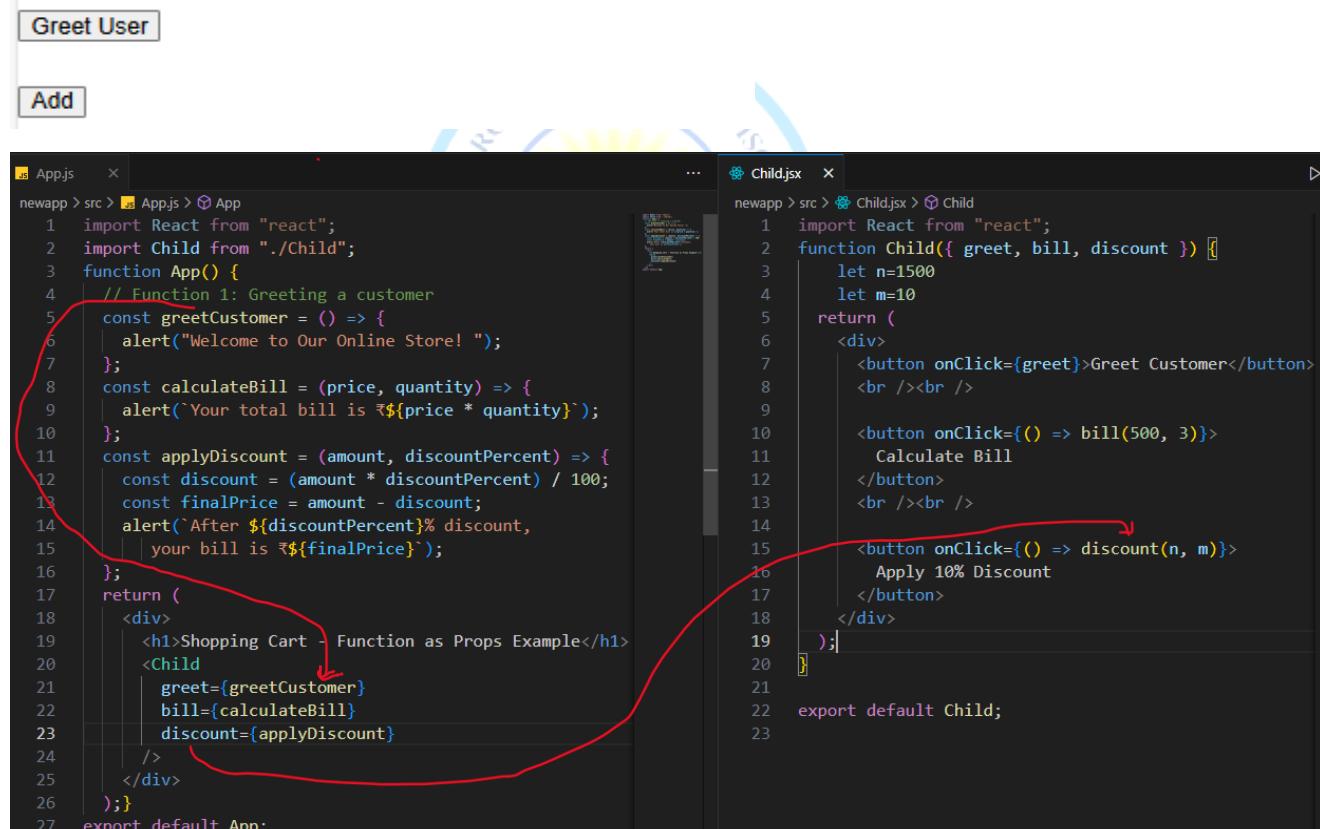
Example-5: Passing a function as prop in react js



```
newapp > src > App.js > App
1 import React from "react";
2 import Child from "./Child";
3
4 function App() {
5   // Function to be passed as a prop
6   const greetUser = () => {
7     alert("Hello Good Morning!");
8   };
9   const add = (a, b) => {
10     alert(`Sum = ${a + b}`);
11   };
12
13   return (
14     <div>
15       <h1>Passing Function as Props</h1>
16       {/* Pass both functions as props */}
17       <Child onGreet={greetUser} onAdd={add} />
18     </div>
19   );
20 }
21
22 export default App;
```

```
newapp > src > Child.jsx > ...
1 import React from "react";
2 function Child({ onGreet, onAdd }) {
3   let a=10
4   let b=20
5   return (
6     <div>
7       <button onClick={onGreet}>Greet User</button>
8       <br /><br />
9       <button onClick={() => onAdd(a,b)}>Add</button>
10     </div>
11   );
12 }
13
14 export default Child;
15
```

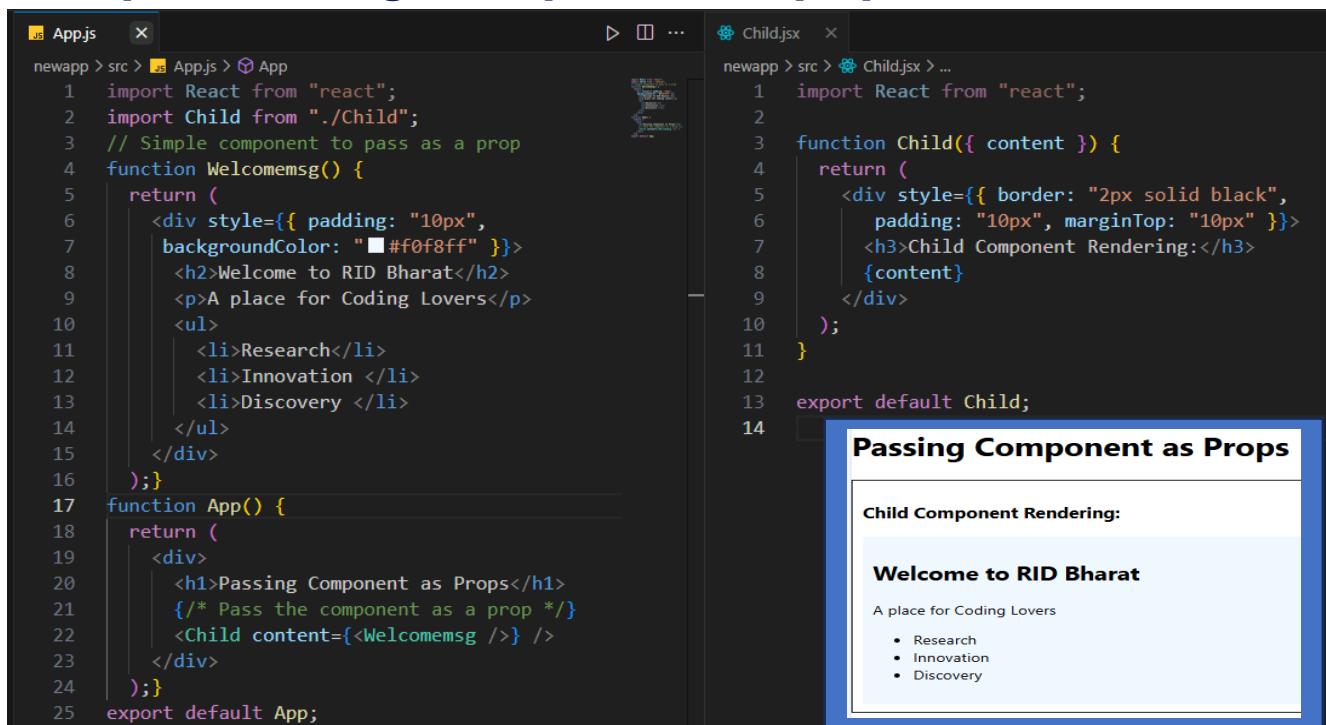
Passing Function as Props



Shopping Cart - Function as Props Example



Example-6: Passing a component as a prop in React JS



```

App.js
newapp > src > App.js > App
1 import React from "react";
2 import Child from "./Child";
3 // Simple component to pass as a prop
4 function Welcomemsg() {
5   return (
6     <div style={{ padding: "10px", backgroundColor: "#f0f8ff" }}>
7       <h2>Welcome to RID Bharat</h2>
8       <p>A place for Coding Lovers</p>
9       <ul>
10      <li>Research</li>
11      <li>Innovation </li>
12      <li>Discovery </li>
13    </ul>
14  </div>
15);
16
17 function App() {
18   return (
19     <div>
20       <h1>Passing Component as Props</h1>
21       {/* Pass the component as a prop */}
22       <Child content={<Welcomemsg />} />
23     </div>
24   );
25 }
26 export default App;

```

```

Child.js
newapp > src > Child.js > ...
1 import React from "react";
2
3 function Child({ content }) {
4   return (
5     <div style={{ border: "2px solid black", padding: "10px", marginTop: "10px" }}>
6       <h3>Child Component Rendering:</h3>
7       {content}
8     </div>
9   );
10
11 }
12
13 export default Child;
14

```

Passing Component as Props

Child Component Rendering:

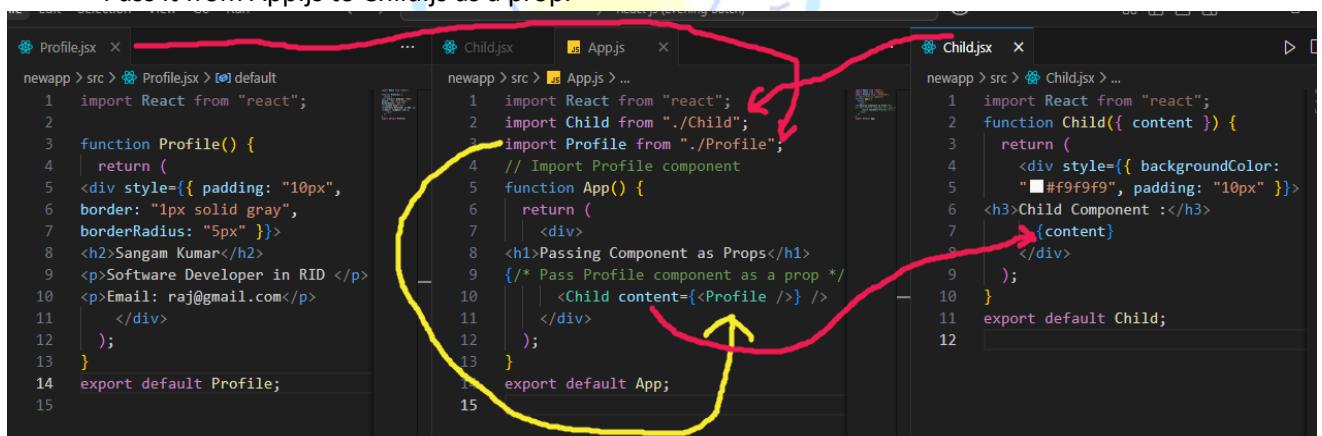
Welcome to RID Bharat

A place for Coding Lovers

- Research
- Innovation
- Discovery

Example-7: How can I import a one component into App.js and then pass it as a prop to Child.js

- Create a **Profile.jsx** component.
- Import it into **App.js**.
- Pass it from **App.js** to **Child.js** as a prop.



```

Profile.jsx
newapp > src > Profile.jsx > default
1 import React from "react";
2
3 function Profile() {
4   return (
5     <div style={{ padding: "10px", border: "1px solid gray", borderRadius: "5px" }}>
6       <h2>Sangam Kumar</h2>
7       <p>Software Developer in RID </p>
8       <p>Email: raj@gmail.com</p>
9     </div>
10  );
11 }
12
13 export default Profile;
14
15

```

```

App.js
newapp > src > App.js > ...
1 import React from "react";
2 import Child from "./Child";
3 import Profile from "./Profile";
4
5 // Import Profile component
6 function App() {
7   return (
8     <div>
9       <h1>Passing Component as Props</h1>
10      {/* Pass Profile component as a prop */}
11      <Child content={<Profile />} />
12    </div>
13  );
14
15  export default App;

```

```

Child.js
newapp > src > Child.js > ...
1 import React from "react";
2
3 function Child({ content }) {
4   return (
5     <div style={{ backgroundColor: "#f9f9f9", padding: "10px" }}>
6       <h3>Child Component :</h3>
7       {content}
8     </div>
9   );
10
11 }
12
13 export default Child;
14

```

Passing Component as Props

Child Component :

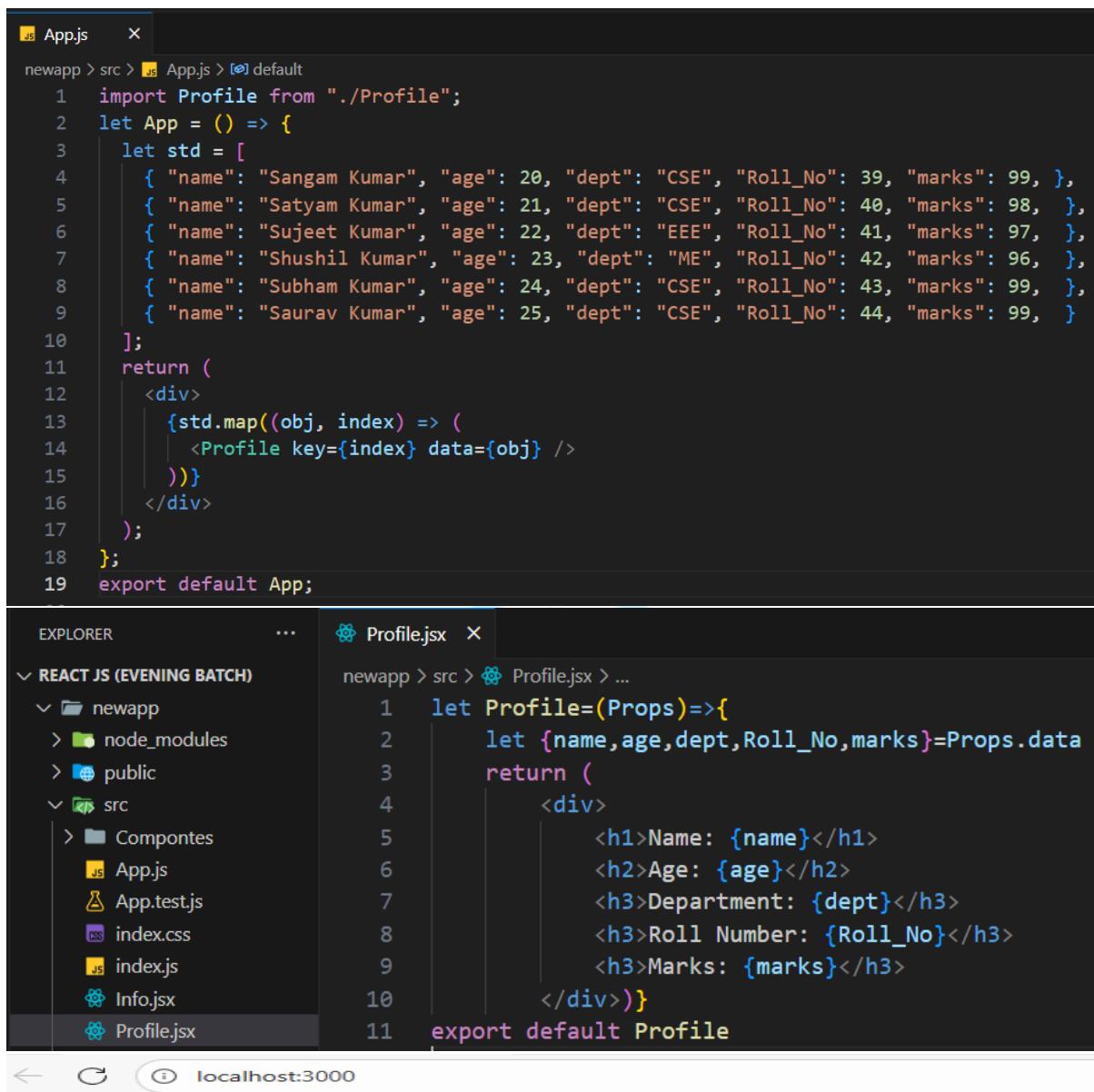
Sangam Kumar

Software Developer in RID

Email: raj@gmail.com



RENDERING ARRAY OF OBJECT



```

App.js
newapp > src > App.js > default
1 import Profile from "./Profile";
2 let App = () => {
3   let std = [
4     { "name": "Sangam Kumar", "age": 20, "dept": "CSE", "Roll_No": 39, "marks": 99, },
5     { "name": "Satyam Kumar", "age": 21, "dept": "CSE", "Roll_No": 40, "marks": 98, },
6     { "name": "Sujeet Kumar", "age": 22, "dept": "EEE", "Roll_No": 41, "marks": 97, },
7     { "name": "Shushil Kumar", "age": 23, "dept": "ME", "Roll_No": 42, "marks": 96, },
8     { "name": "Subham Kumar", "age": 24, "dept": "CSE", "Roll_No": 43, "marks": 99, },
9     { "name": "Saurav Kumar", "age": 25, "dept": "CSE", "Roll_No": 44, "marks": 99, }
10   ];
11   return (
12     <div>
13       {std.map((obj, index) => (
14         <Profile key={index} data={obj} />
15       )))
16     </div>
17   );
18 };
19 export default App;

```

```

Profile.jsx
newapp > src > Profile.jsx > ...
1 let Profile=(Props)=>{
2   let {name,age,dept,Roll_No,marks}=Props.data
3   return (
4     <div>
5       <h1>Name: {name}</h1>
6       <h2>Age: {age}</h2>
7       <h3>Department: {dept}</h3>
8       <h3>Roll Number: {Roll_No}</h3>
9       <h3>Marks: {marks}</h3>
10      </div>)
11  export default Profile

```

localhost:3000

Name: Sangam Kumar

Age: 20

Department: CSE

Roll Number: 39

Marks: 99

Name: Satyam Kumar

Age: 21

Department: CSE

Roll Number: 40

Marks: 98



State in React JS

State = Data that can change + React automatically updates UI when it changes.

❖ State in React JS

- State is the internal memory of a component.
- It stores **changeable/local data** of a component.
- State is a **JavaScript object** used to manage dynamic data.
- When state changes, React **re-renders** the component to update the UI.

❖ Why use State?

- If we store data in a normal variable, changes won't update the UI.
- Using state ensures UI updates automatically when data changes.

❖ Key Points:

1. **Local** – Each component has its own state.
2. **Mutable** – Data can change over time.
3. **Re-rendering** – UI updates when state changes.
4. **Immutable Update** – Don't modify directly, always use `setState()` (or `useState` in functional components).

Example: Light Bulb Switch

- You have a bulb in your room.
- Initially, the bulb is **OFF**.
- When you press the switch, the bulb turns **ON**.
- Press again → it turns **OFF**.

The **bulb's status (ON/OFF)** is the **state**.

- It **changes over time** when you interact with it.
- Whenever the state changes, the **UI (bulb image/text)** updates automatically.

Example: Student Attendance

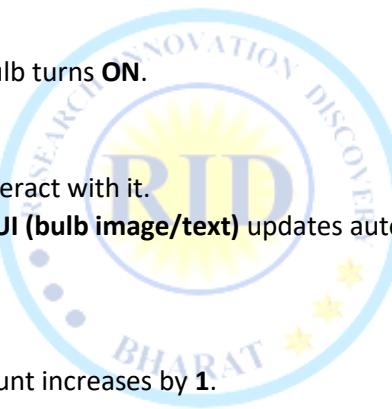
A teacher is marking attendance.

Present students count starts at **0**.

Every time a student says "Present", the count increases by **1**.

The updated count is shown on the screen.

attendance count is the **state**, because it keeps changing and the UI reflects it instantly.



```
App.js x
newapp > src > App.js > ...
1  let App=()=>{
2    let c=0
3    let inc=()=>{
4      c=c+1
5      console.log(c)
6    }
7    return(
8      <div>
9        <h1>Cout={c}</h1>
10       <button onClick={inc}>inc</button>
11     </div>
12   )
13   export default App
```

Cout=0

inc

- **Problem:** count will not increase in UI (Infront of user) but it will change the in console

```
App.js x
newapp > src > App.js > App
1  let App = () => {
2    let present = 0;
3    let markPresent = () => {
4      // increase present count
5      present = present + 1;
6      console.log("Present Students:", present);
7    };
8    return (
9      <div>
10        <h1>Present Students = {present}</h1>
11        <button onClick={markPresent}>Mark Present</button>
12      </div>
13    );
14  };
15  export default App;
```

Problem:

- **Present Students** count will not update on the screen (UI).
- It only changes in the **console**.

Reason: React does not re-render for normal variables.

Sol: Use **state** (`useState`) to update and show attendance in UI.



❖ How to Create State in React

- **useState** is used to create a **state variable** in a component.
- It takes an **initial value** and returns:
 1. The **state variable**
 2. A **function to update the state**

❖ Syntax: let [stateVariable, setStateFunction] = useState(initialValue);

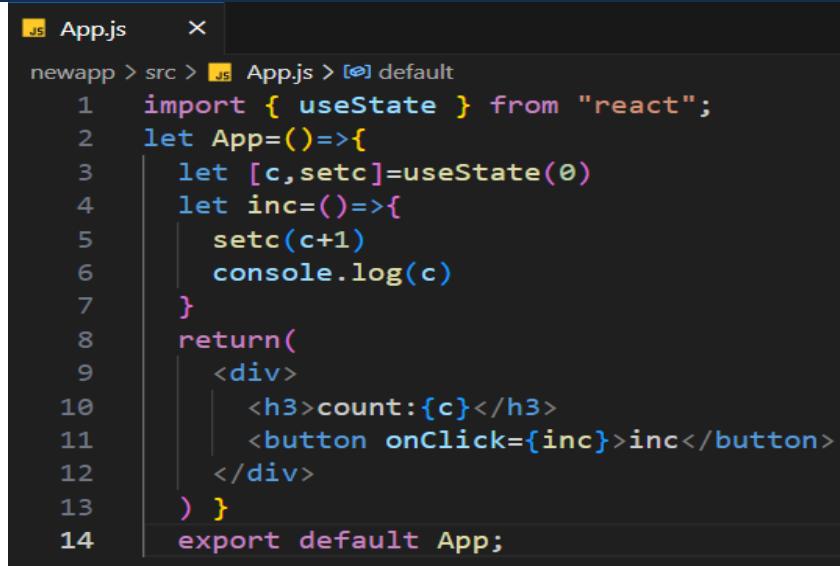
1. **useState(initialValue)** → creates a state with a starting value.
2. **stateVariable** → holds the current value of the state.
3. **setStateFunction** → updates the state and re-renders the component.
4. **Never update stateVariable directly**; always use the updater function.

❖ Example:

```
let [count, setCount] = useState(0);
setCount(count + 1); // updates count and UI
```

❖ Key Points:

- **Do not update state variable directly**. Always use the **state updating function**.
- Every state change creates a **new value** for re-rendering.
- The **latest state value** is available for the next render.



```
App.js      ×
newapp > src > App.js > default
  1  import { useState } from "react";
  2  let App=()=>{
  3    let [c,setc]=useState(0)
  4    let inc=()=>{
  5      setc(c+1)
  6      console.log(c)
  7    }
  8    return(
  9      <div>
10        <h3>count:{c}</h3>
11        <button onClick={inc}>inc</button>
12      </div>
13    )
14  export default App;
```

```
JS App.js  X
newapp > src > JS App.js > ...
1  import { useState } from "react";
2  let App = () => {
3    // create state for present students, initial value 0
4    let [present, setPresent] = useState(0);
5    let markPresent = () => {
6      // update state using setPresent
7      setPresent(present + 1);
8      console.log("Present Students:", present + 1);
9    };
10   return (
11     <div>
12       <h1>Present Students = {present}</h1>
13       <button onClick={markPresent}>Mark Present</button>
14     </div>
15   );
16 };
17 export default App;
```

❖ increment and decrement number application.

Example: App.js

```
JS App.js  X
newapp > src > JS App.js > [o] App > [o] dec
1  import {useState} from 'react'
2  let App=()=>{
3    let [count,setCount]=useState(0)
4    let inc=()=>{
5      // setCount((x)=>{
6      //   console.log(x)
7      //   return x+1  // })
8      setCount(count+1)
9    }
10   let dec=()=>{
11     setCount(count-1)
12   }
13   return(
14     <div>
15       <h1>Count={count}</h1>
16       <button onClick={inc}>inc</button>
17       <button onClick={dec}>dec</button>
18     </div> )
19 }
20 export default App
```



Example-3:

```
import { useState } from "react";
let App = () => {
  let [c, setc] = useState(0);
  let inc = () => {
    setc(c + 1);
    console.log("Incremented:", c + 1);
  };

  let dec = () => {
    setc(c - 1);
    console.log("Decrementated:", c - 1);
  };

  let inc2 = () => {
    setc(c + 2);
    console.log("Incremented by 2:", c + 2);
  };

  let dec2 = () => {
    setc(c - 2);
    console.log("Decrementated by 2:", c - 2);
  };

  return (
    <div>
      <h3>count: {c}</h3>
      <button onClick={inc}>inc</button>
      <button onClick={dec}>dec</button>
      <button onClick={inc2}>inc by 2</button>
      <button onClick={dec2}>dec by 2</button>
    </div>
  );
};

export default App;
```

count: 3

inc dec inc by 2 dec by 2

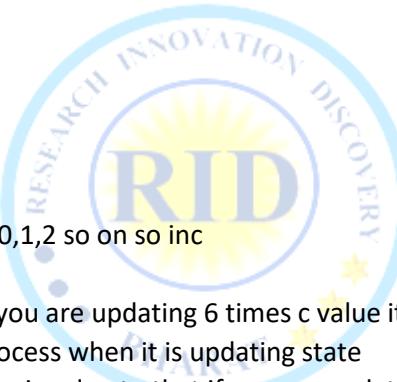
❖ Batch updating.

```
import {useState} from 'react'  
let App=()=>{  
  let [c, setCount]=useState(0)  
  let update=()=>{  
    setCount(c+1)  
    setCount(c+1)  
    setCount(c+1)  
    setCount(c+1)  
    setCount(c+1)  
    setCount(c+1)  
  }  
  return(  
    <div>  
      <h3>Count:{c}</h3>  
      <button onClick={update}>inc</button>  
    </div>  
  )  
}  
export default App
```

Count:1

inc

Output: count: 0 here count will 0,1,2 so on so inc



- in the above example even if you are updating 6 times c value it is considering old value in all the case to react follows batch process when it is updating state
- react state follow batch processing due to that if you are updating more than one time in a single function it will reflect only one time.

Note:

Expected Output:

You might expect the count to go up by 6 — from 0 to 6.

Actual Output:

It only increases by 1, not 6. So it goes:

Count: 0 → Click → Count: 1

Why?

React batches state updates when they happen inside the same synchronous function (like update here). And each time you're calling
setCount(c + 1)

you're using the old value of c, which is 0.

So even though you call setCount(c + 1) six times, they all effectively do
setCount(1)

❖ what is batch processing?

- Batch processing means all the statement will be consider as single group and update execute as a group or at time to avoid this
- to solve this issue in above update the state by help of call by function then call by function will take previous state as argument updated values as return value

Note: To update the state using call back function as a proper way.

Example:

```
import { useState } from 'react';
let App = () => {
  let [c, setCount] = useState(0);
  let update = () => {
    // Use the functional update form to ensure correct updates
    setCount((prevCount) => prevCount + 1); // call by function
    setCount((prevCount) => prevCount + 1);
    setCount((prevCount) => prevCount + 1);
  };
  return (
    <div>
      <h3>Count: {c}</h3>
      <button onClick={update}>inc</button>
    </div>
  );
};

export default App;
```



Output:

Count: 0 . here count will 0,6, 12 etc...
inc

Count: 0

inc

Count: 6

inc

Count: 12

inc

❖ when to use call by function?

- if current state depended into previous state that time, we are using call by function.

Example:

```
import React, { useState, useEffect } from "react";
let App = () => {
  let [c, setCount] = useState(0);
  let [lId, setlId] = useState(-1);
  useEffect(() => {
    // Cleanup on unmount
    return () => {
      if (lId !== -1) {
        clearInterval(lId);
      }
    };
  }, [lId]);
  let inc = () => {
    setCount((prevCount) => prevCount + 1);
  };
  let start = () => {
    if (lId === -1) {
      const intervalId = setInterval(inc, 1000);
      setlId(intervalId);
    }
  };
  let stop = () => {
    if (lId !== -1) {
      clearInterval(lId);
      setlId(-1);
    }
  };
  let reset = () => {
    setCount(0);
    if (lId !== -1) {
      clearInterval(lId);
      setlId(-1);
    }
  };
  return (
    <div>
      <h3>Count: {c}</h3>
      <button onClick={start}>Start</button>
      <button onClick={stop}>Stop</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
};
export default App;
```



Output:

Count: 9

Start **Stop** **Reset**

Key Concepts Used:

- **useState** — to store count (c) and interval ID (lId).
- **useEffect** — to clean up (stop interval) when the component is removed from the screen.
- **setInterval** — to run a function (inc) every second.
- **clearInterval** — to stop the automatic counting.

Code Explanation:

- let [c, setCount] = useState(0); Creates a state variable c (initial value 0) to keep track of count.
- let [lId, setlId] = useState(-1); Stores the interval ID. -1 means no interval is running.

Code-2

```
useEffect(() => {
  return () => {
    if (lId !== -1) {
      clearInterval(lId); // Clear interval when component unmounts
    }
  };
}, [lId]);
```

Note:

- When the component is removed from the screen, this code runs and stops the interval (to avoid memory leaks).
- It runs again whenever lId changes.

```
let inc = () => {
  setCount((prevCount) => prevCount + 1);
};
```

Increases the count by 1.

Code-3

```
let start = () => {
  if (lId === -1) {
    const intervalId = setInterval(inc, 1000);
    setlId(intervalId);
  }
};
```

Code-4:

```
let start = () => {
  if (lId === -1) {
    const intervalId = setInterval(inc, 1000);
    setlId(intervalId);
  }
};
```

If no interval is running, it starts one that calls inc() every 1000ms (1 second), and stores the ID.

Code-5:

```
let stop = () => {
  if (lId !== -1) {
    clearInterval(lId);
    setlId(-1);
  }
};
```

Stops the interval and marks it as "not running" by setting lId = -1.

Code-6

```
let reset = () => {
  setCount(0);
  if (lId !== -1) {
    clearInterval(lId);
    setlId(-1);
  }
};
```

Resets the count to 0 and stops the interval if it's running.

useEffect

- useEffect is a built-in hook in React that allows you to perform side effects in functional components. Side effects can include data fetching, manually changing the DOM, subscribing to external data sources, and more.

Syntax:

```
useEffect(effectFunction, dependencyArray)
useEffect(()=>{},[])
```

- effectFunction:** This is a required function that contains the code for the side effect you want to perform. It's called after the component renders.
- dependencyArray (optional):** This is an array of dependencies that determines when the effect should run. It's an optional argument. If you omit it, the effect will run after every render.
- dependencyArray:
- Empty Array ([]):** The effect runs once after the initial render and doesn't re-run.

Example:

```
useEffect(() => {
  // This effect runs only once after the initial render
  // Add cleanup code if needed
}, []);
```

No Dependency Array (Omitted): The effect runs after every render.

```
useEffect(() => {
  // This effect runs after every render
});
```

With Dependencies: The effect runs whenever any of the specified dependencies change.

```
const [count, setCount] = useState(0);
useEffect(() => {
  // This effect runs whenever 'count' changes
}, [count]);
```

The return value of the useEffect hook is an optional cleanup function.

Example: Digital Watch:

```
useEffect(() => {
  // Effect code
  // Cleanup function
  return () => {
    // Cleanup code
  };
}, [dependencies]);
```

digital clock watch:

Example:

```
import React, { useState, useEffect } from "react";
const DigitalClock = () => {
  const [time, setTime] = useState(new Date());
  useEffect(() => {
    // Create an interval to update the time every second
    const intervalId = setInterval(() => {
      setTime(new Date());
    }, 1000);
  }, []);
  return (
    <div>
      <h1>{time.toLocaleTimeString()}</h1>
    </div>
  );
}
```

```
}, 1000);
// Return a cleanup function to clear the interval when the component unmounts
return () => {
  clearInterval(intervalId);
};
[], []); // Empty dependency array, so the effect runs only once after the initial render
// Format the time as HH:MM:SS
const formattedTime = time.toLocaleTimeString();
return (
  <div>
    <h2>Digital Clock</h2>
    <div className="clock">{formattedTime}</div>
  </div>
);
};
export default DigitalClock;
```

output:



Twksaa Digital Clock

11:04:33 AM

Example-2: add dynamic data from user side.

```

import { useState } from "react";
let App = () => {
  let [name, setName] = useState(""); // 'name' stores the input field value
  let [data, setData] = useState([]); // 'data' stores the list of names
  let fun = (e) => { // Event handler for input field changes
    setName(e.target.value); // Update the 'name' state with the current input value
  }
  let add = () => { // Event handler for the 'Add' button click
    // Add the current 'name' to the 'data' array
    // Using the spread operator to create a new array with the updated data
    setData([...data, name]);
    setName(""); // Reset the 'name' state to an empty string, clearing the input field
  }
  // JSX for rendering the UI
  return (
    <div>
      <input type="text" onChange={fun} value={name} />
      <button onClick={add}>Add</button>
      <div>
        {data.map((item) => <li>{item}</li>)}
      </div>
    </div>
  );
}
export default App;

```

Output:

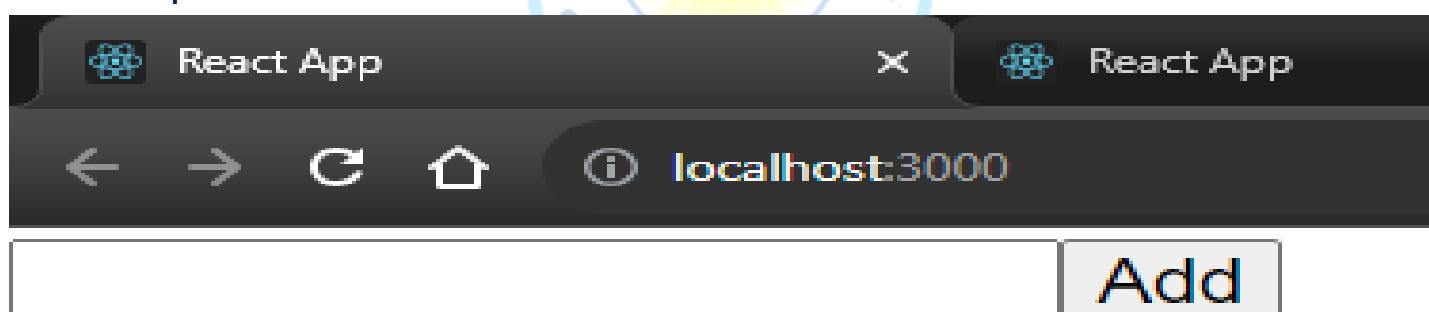
[...data]:- This is called the **spread syntax or spread operator** in JavaScript

Why we use the spread operator (...) in React state updates

setData([...data, name]);

You're doing **immutable state update** — meaning:

- You're **not modifying** the original data array.
- Instead, you're **creating a new array** that includes all previous items (...data) and the new one (name).



- Twksaa core Python
- Twksaa Advance python
- Twksaa HTML Book

Note:

- e is the **event object** passed by the browser.
- e.target refers to the **DOM element** that triggered the event (in this case, the <input>).
- target is a **property** of the event object — it is part of the DOM Events API.
- value is a **property** of an <input> element, containing what the user typed

```

<ol type="1">
  {data.map((item, index) => (
    <li key={index}>{item}</li>
  )))
</ol>

```

useState

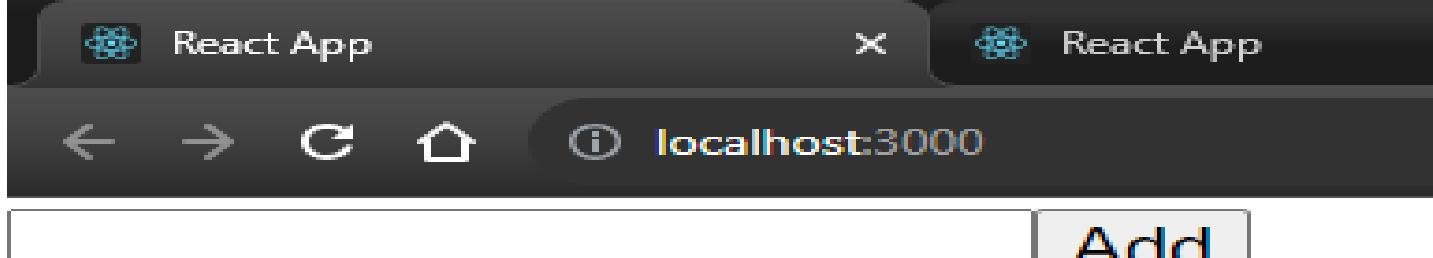
- useRef does not trigger re-renders when the value it references changes.
- To attach reference to any HTML element we will create reference by half useRef.
- we will attach reference by half ref props.

2nd method

Example-2:

```
import React, { useRef, useState } from "react";
let App = () => {
  let [data, setData] = useState([]); // Declare a state variable 'data' using the 'useState' hook and Initialize it as an empty array []
  let name = useRef(); // Create a ref called 'name' using the 'useRef' hook
  let add = () => { // Define a function 'add'
    // Update the 'data' state by adding the current value of 'name' ref
    // to the existing 'data' array using the spread operator [...data]
    setData([...data, name.current.value]);
    name.current.value = ""; // Clear the input field by setting its value to an empty string
  };
  return ( // Render the component's UI
    <div>
      <input type="text" ref={name} />
      <button onClick={add}>Add</button>
      <div>
        {data.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </div>
    </div>
  );
}
export default App;
```

output:



- **TWKSAA CSS BOOK**
- **TWKSAA JAVASCRIPT BOOK**
- **TWSAA REACT JS**

Mini Project-1

(Data Entry Apps)

```

import { useState } from "react";
let App = () => {
  let [std, setStd] = useState({ "name": "", "email": "", "phno": "", "branch": "" });
  // Initializing a state variable 'data' as an empty array.
  let [data, setData] = useState([]);
  let fun = (e) => {
    setStd({ ...std, [e.target.name]: e.target.value });
  }
  // function 'add' that adds the current 'std' object to 'data' array and resets 'std' to empty values.
  let add = () => {
    setData([...data, std]);
    setStd({ "name": "", "email": "", "phno": "", "branch": "" });
  }
  // The 'return' statement starts the rendering of the component.
  return (
    <div>
      <div>
        {/* Input fields for name, email, phone number, and branch, with 'onChange' event
        handlers to call 'fun' and 'value' to display the values from the 'std' state. */}
        <input type="text" placeholder="Enter the name" name="name" onChange={fun} value={std.name} />
        <input type="text" placeholder="Enter your Email" name="email" onChange={fun} value={std.email} />
        <input type="text" placeholder="Enter Phone Number" name="phno" onChange={fun} value={std.phno} />
        <select onChange={fun} name="branch" value={std.branch}>
          <option disabled selected value="">Select branch</option>
          <option value="CSE">CSE</option>
          <option value="ECE">ECE</option>
          <option value="EEE">EEE</option>
          <option value="ME">ME</option>
          <option value="CE">CE</option>
        </select>
        <button onClick={add}>Add</button>
      </div>
      {/* Conditionally rendering a table if 'data' is not empty. */}
      { data.length !== 0 && <table border={1}>
        <thead>
          <tr>
            <th>Name</th><th>Email</th><th>Phno</th><th>Branch</th>
          </tr>
        </thead>
        <tbody>
          { // Mapping over the 'data' array to display its content in rows of the table.
            data.map((std) => {

```

setStd(...) – Updates the std state using React's useState.

{ ...std, [e.target.name]: e.target.value } – Creates a new object:

- ...std – Copies all current fields of std.
- [e.target.name] – Dynamically picks the field to update (like "name" or "email").
- e.target.value – The new value typed by the user.

{ data.length !== 0 && ... }

This is a conditional rendering in React. It means: "Only show the table if data is not empty."

<table border={1}>

This creates an HTML table with a border of width 1.

```
        return (
      <tr>
        <td>{std.name}</td>
        <td>{std.email}</td>
        <td>{std.phno}</td>
        <td>{std.branch}</td>
      </tr>
    )
  })
}
</tbody>
</table>
}
</div>
)
}

export default App;
```

Output:



A screenshot of a web browser showing a table of student data. The browser address bar shows 'localhost:3000'. The table has columns for Name, Email, Phno, and Branch. The data is as follows:

Name	Email	Phno	Branch
Sangam Kumar	sangamkr393@gmail.com	9988776655	CSE
Satyam Kumar	satyamkr339@gmail.com	8877667755	ECE
Sujeet Kumar	sujeetkr5896@gmail.com	7799886655	EEE
Shusil Kumar	shushikr56@gmail.com	9656545452	ME
Raushni Kumari	raushnikr3453@gmail.com	9856548932	CSE
Priti Kumari	pritikumari53@gmail.com	8932542365	CSE

Mini Project-2 To do Application

Program:

```
import { useState } from 'react';
let App = () => {
  let [data, setData] = useState({ uid: "", title: "", desc: "", deadline: "" });
  let [todo, setTodo] = useState([]);
  let [ind, setInd] = useState();
  let [f, setFlag] = useState(true);
  let [comp, setComp] = useState([]);

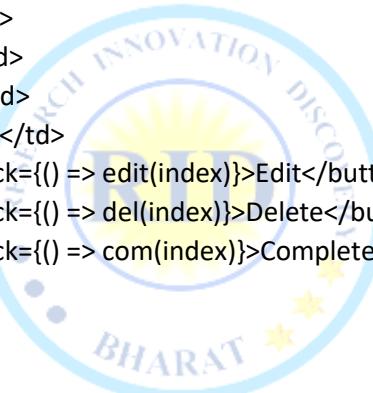
  let fun = (e) => {
    // Update the `data` state by spreading the previous state and updating the changed field
    setData({ ...data, [e.target.name]: e.target.value });
    console.log(data); // Log the current state (it may not reflect the update immediately
    // due to the asynchronous nature of `useState`)
  }; // Event handler function for adding a task
  let add = () => {
    // Add the current `data` state to the `todo` array and clear the input fields
    setTodo([...todo, data]);
    setData({ uid: "", title: "", desc: "", deadline: "" });
  }; // Event handler function for deleting a task from the `todo` array
  let del = (index) => {
    // Remove the task at the specified index and update the `todo` state
    todo.splice(index, 1);
    setTodo([...todo]);
  }; // Event handler function for moving a task to the completed tasks list
  let com = (index) => {
    // Add the task at the specified index in `todo` to the `comp` array and delete it from `todo`
    setComp([...comp, todo[index]]);
    del(index);
  }; // Event handler function for editing a task
  let edit = (index) => {
    // Set the `data` state to the task at the specified index in `todo`, set the index, and switch to edit mode
    setData(todo[index]);
    setInd(index);
    setFlag(false);
  }; // Event handler function for updating an edited task
  let upd = () => {
    // Replace the task at the specified index in `todo` with the updated `data`, and reset the state for editing
    todo[ind] = data;
    setTodo([...todo]);
    setFlag(true);
    setData({ uid: "", title: "", desc: "", deadline: "" });
  }; // Return the JSX (user interface) for the component
  return (
    <div>
      <input type="text" placeholder='Enter User Id' name="uid" value={data.uid} onChange={fun} />
```



```

<input type="text" placeholder='Enter Title' name="title" value={data.title} onChange={fun} />
<input type="text" placeholder='Enter Description' name="desc" value={data.desc} onChange={fun} />
<input type="date" name="dedline" value={data.dedline} onChange={fun} />
/* Conditional rendering of buttons based on whether the component is in add or edit mode */
{f && <button onClick={add}>Add</button>
 {!f && <button onClick={upd}>Update</button>}
 /* Display the to-do list in a table */
<div>
<table border={1}>
<tr>
<th>UID</th>
<th>Title</th>
<th>Description</th>
<th>Deadline</th>
</tr>
/* Map through the `todo` array to display tasks */
{todo.map((item, index) => {
return (
<tr>
<td>{item.uid}</td>
<td>{item.title}</td>
<td>{item.desc}</td>
<td>{item.dedline}</td>
<td><button onClick={() => edit(index)}>Edit</button></td>
<td><button onClick={() => del(index)}>Delete</button></td>
<td><button onClick={() => com(index)}>Completed</button></td>
</tr>
);
})}
</table>
</div>
/* Display the completed tasks list in a table */
<div>
<h1>Completed tasks:</h1>
<table border={1}>
<tr>
<th>UID</th>
<th>Title</th>
<th>Description</th>
<th>Deadline</th>
</tr>
/* Map through the `comp` array to display completed tasks */
{comp.map((item, index) => {
return (
<tr>
<td>{item.uid}</td>
<td>{item.title}</td>
<td>{item.desc}</td>

```



```

        <td>{item.dedline}</td>
        <td><button onClick={() => com(index)}>Delete</button></td>
        </tr>
    );
}
</table>
</div>
</div>
);
};

export default App;

```

Result:

Enter User Id		Enter Title		Enter Description		dd - mm - yyyy	<input type="button" value="Add"/>
UID	Title	Description	Deadline	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	<input type="button" value="Completed"/>	
101	THML	web design	2023-09-30	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	<input type="button" value="Completed"/>	
102	CSS	Style	2023-09-30	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	<input type="button" value="Completed"/>	
393	Twksaa skills guide book	Useful book	2023-09-30	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	<input type="button" value="Completed"/>	

Completed tasks:

UID	Title	Description	Deadline	<input type="button" value="Delete"/>
103	JavaScript	Action	2023-09-30	<input type="button" value="Delete"/>
106	Python	Back End	2023-09-27	<input type="button" value="Delete"/>
109	React Js	JavaScript Library	2023-09-14	<input type="button" value="Delete"/>

❖ Passing state as a prop for child component

1. Define the Parent Component:

- First, you'll need a parent component that holds the state you want to pass to the child component. The state can be defined using the `useState` hook.

Example-1: App.js (Parent Component)

```
import { useState } from "react";
import Display from "./Display"; // Import the child component
const App = () => {
  let [c, setc] = useState(0);
  let inc = () => setc(c + 1);
  let dec = () => setc(c - 1);
  let inc2 = () => setc(c + 2);
  let dec2 = () => setc(c - 2);
  return (
    <div>
      <Display value={c} /* Pass state to child via props */>
      <button onClick={inc}>inc</button>
      <button onClick={dec}>dec</button>
      <button onClick={inc2}>inc by 2</button>
      <button onClick={dec2}>dec by 2</button>
    </div>
  );
};

export default App;
```

Example-2: Display.js (Child Component)

```
const Display = ({ value }) => {
  return (
    <h3>Count: {value}</h3>
  );
};

export default Display;
```

Count: -2

inc dec inc by 2 dec by 2

Example-2: App.js (Parent Component)

```
import React, { useState, useEffect } from "react";
import Display from "./Display"; // Import the child component
let App = () => {
  let [c, setCount] = useState(0);
  let [lId, setlId] = useState(-1);
  useEffect(() => {
    return () => {
      if (lId !== -1) {
        clearInterval(lId);
      }
    };
  }, [lId]);
  let inc = () => {
    setCount((prevCount) => prevCount + 1);
  };
  let start = () => {
    if (lId === -1) {
      const intervalId = setInterval(inc, 1000);
      setlId(intervalId);
    }
  };
  let stop = () => {
    if (lId !== -1) {
      clearInterval(lId);
      setlId(-1);
    }
  };
  let reset = () => {
    setCount(0);
    if (lId !== -1) {
      clearInterval(lId);
      setlId(-1);
    }
  };
  return (
    <div>
      <Display value={c} /> /* Pass count to child */
      <button onClick={start}>Start</button>
      <button onClick={stop}>Stop</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
};
export default App;
```

2. Display.js (Child Component)

```
const Display = ({ value }) => {
  return <h3>Count: {value}</h3>;
};

export default Display;
```

Count: 6

Start Stop Reset

Example: using a functional component and the `useState` hook:

```
import React, { useState } from 'react';
function ParentComponent() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <ChildComponent count={count} />
    </div>
  );
}
```

2. Create the Child Component:

- Next, create the child component that will receive the state as a prop. In this example, `count` is passed as a prop to `ChildComponent`.

Example:

```
import React from 'react';
function ChildComponent(props) {
  return (
    <div>
      <p>Count: {props.count}</p>
    </div>
  );
}
```

3. Rendering the Parent Component:

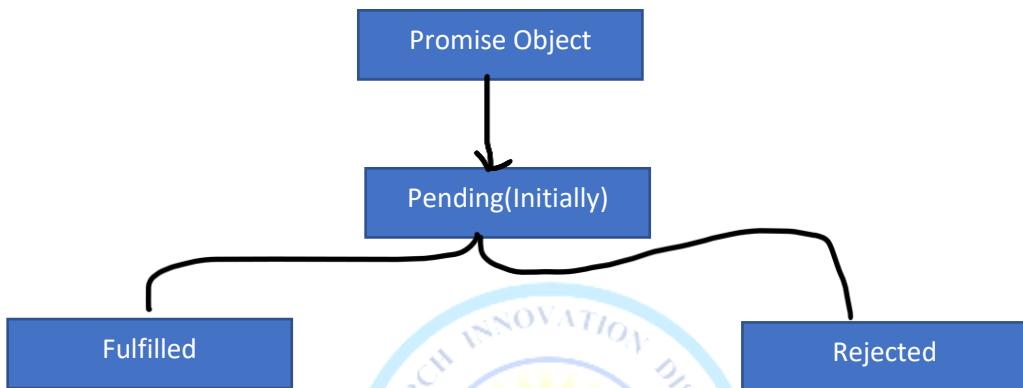
- To make this work, you need to render the `ParentComponent` in your main application file (e.g., `index.js` or `App.js`).

```
import React from 'react';
import ReactDOM from 'react-dom';
import ParentComponent from './ParentComponent';
ReactDOM.render(
  <React.StrictMode>
    <ParentComponent />
  </React.StrictMode>,
  document.getElementById('root')
);
```



❖ State lifting

- If you want to maintain a common state for the group of components, we will maintain state in its parent component it is call state lifting.
- To implement lifecycle method in functional components useEffect will take two arguments:
 1. **Call back function()**: which it need to execute
 2. Dependency array
- If dependency array is empty useEffect will create only one At time of component initial rendering
- In dependency array if you provide some variables when that variable are updated every time use Effect will execute



Syntax: Promiseobj.then((res)=>{ }).catch((err)=>{})

Note: to mange parent component state through its child component, we will pass state updation as props

❖ state lifting in React

1. State in React Components:

- components can have their own state, which is a JavaScript object that holds data specific to that component. State allows components to manage & update their data independently.

2. Parent and Child Components:

- In a typical React application, you often have a hierarchy of components, with some components being children of others.

3. When to Use State Lifting:

- State lifting becomes necessary when you have a piece of state in a child component that needs to be accessed or modified by a parent or other sibling components.

4. Lifting State Up:

- a. Identify the state that needs to be shared or modified by multiple components.
- b. Move this state up to the nearest common ancestor component that needs access to it.
- c. This parent component becomes the "source of truth" for the shared state.

5. Passing State as Props:

- After lifting the state, you pass it down to the child component(s) that need access to it as props.

6. Updating State:

- When a child component needs to update the state, it should call a function passed down from the parent component as a prop.

7. Benefits of State Lifting:

- Centralized State:** State lifting helps in centralizing the management of shared state, making it easier to understand and maintain.
- Data Flow Control:** It provides better control over the flow of data between components.
- Reusable Components:** By lifting state, you can create more reusable components because the state is managed by a parent, and child components can focus on rendering and behavior

Example-1:

➤ app.js npm install axios

```
import axios from "axios"
import { useEffect, useState } from "react"
import Row from "./Row"
let App=()=>{
  let [data, setData]=useState([])
  useEffect(()=>{
  axios.get("https://jsonplaceholder.typicode.com/posts").then((res)=>{
    setData(res.data)
  })
  },[])
  let del=(ind)=>{
    data.splice(ind,1)
    setData([...data])
  }
  return(
    <div>
      <table border={1}>
        <tr><th>uid</th><th>Id</th><th>title</th><th>body</th></tr>
        {
          data.map((item,index)=><Row data={item} del={del} i={index}/>)
        }
      </table>
    </div>
  )
}
export default App
```

- axios:** Library for making HTTP requests.
- useState:** To store and manage the data.
- useEffect:** To run code when the component mounts.
- Row:** A custom component to display a single row.

API Used: <https://jsonplaceholder.typicode.com/posts>

- This is a **fake REST API** for testing and prototyping.
- The endpoint /posts returns an array of 100 post objects, each like:

```
{
  "userId": 1,
  "id": 1,
  "title": "some title",
  "body": "some body text"
}
```

Row.js:

```
let Row=(props)=>{
  let data=props.data
  return(
    <tr>
      <td>{data.userId}</td>
      <td>{data.id}</td>
      <td>{data.title}</td>
      <td>{data.body}</td>
    <td><button onClick={()=>props.del(props.i)}>Delete</button></td>
  </tr>
)}
```

export default Row

uid	Id	title	body	
1	1	sunt aut facere repellat provident occaecati excepturi optio reprehenderit	quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto	<button>Delete</button>
1	2	qui est esse	est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla	<button>Delete</button>

Data access through Api in React js

Example-2: Display Users from API

API used: <https://jsonplaceholder.typicode.com/users>

➤ App.js

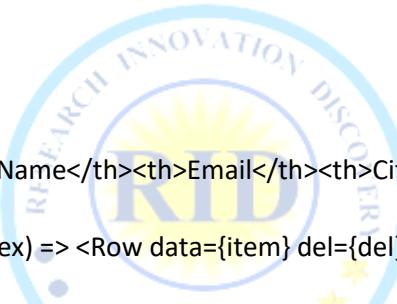
```
import axios from "axios"
import { useEffect, useState } from "react"
import Row from "./Row"

let App = () => {
  let [data, setData] = useState([])
  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/users").then((res) => {
      setData(res.data)
    })
  }, [])
  let del = (ind) => {
    data.splice(ind, 1)
    setData([...data])
  }
  return (
    <div>
      <table border={1}>
        <tr><th>ID</th><th>Name</th><th>Email</th><th>City</th><th>Action</th></tr>
        {
          data.map((item, index) => <Row data={item} del={del} i={index} />)
        }
      </table>
    </div>
  )
}
export default App
```

- **del(ind)** removes one user from the list based on their index.
- **splice(ind, 1)**: Removes 1 item at position ind.
- **setData([...data])**: Updates the state with the new array (by creating a copy using [...data]).

➤ Row.js

```
let Row = (props) => {
  let data = props.data
  return (
    <tr>
      <td>{data.id}</td>
      <td>{data.name}</td>
      <td>{data.email}</td>
      <td>{data.address.city}</td>
      <td><button onClick={() => props.del(props.i)}>Delete</button></td>
    </tr>
  )
}
export default Row
```



ID	Name	Email	City	Action
1	Leanne Graham	Sincere@april.biz	Gwenborough	<button>Delete</button>
2	Ervin Howell	Shanna@melissa.tv	Wisokyburgh	<button>Delete</button>
3	Clementine Bauch	Nathan@yesenia.net	McKenziehaven	<button>Delete</button>
4	Patricia Lebsack	Julianne.OConner@kory.org	South Elvis	<button>Delete</button>
5	Chelsey Dietrich	Lucio_Hettinger@annie.ca	Roscoeview	<button>Delete</button>
6	Mrs. Dennis Schulist	Karley_Dach@jasper.info	South Christy	<button>Delete</button>
7	Kurtis Weissnat	Telly.Hoeger@billy.biz	Howemouth	<button>Delete</button>
8	Nicholas Runolfsdottir V	Sherwood@rosamond.me	Aliyaview	<button>Delete</button>
9	Glenna Reichert	Chaim_McDermott@dana.io	Bartholomebury	<button>Delete</button>
10	Clementina DuBuque	Rey.Padberg@karina.biz	Lebsackbury	<button>Delete</button>



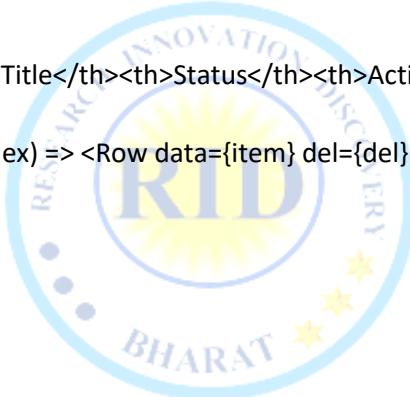
Example-3: Display Todos from API

API used: <https://jsonplaceholder.typicode.com/todos>

App.js

```
import axios from "axios"
import { useEffect, useState } from "react"
import Row from "./Row"

let App = () => {
  let [data, setData] = useState([])
  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/todos").then((res) => {
      setData(res.data.slice(0, 20)) // limiting to 20 todos
    })
  }, [])
  let del = (ind) => {
    data.splice(ind, 1)
    setData([...data])
  }
  return (
    <div>
      <table border={1}>
        <tr><th>ID</th><th>Title</th><th>Status</th><th>Action</th></tr>
        {
          data.map((item, index) => <Row data={item} del={del} i={index} />)
        }
      </table>
    </div>
  )
}
export default App
```



Row.js

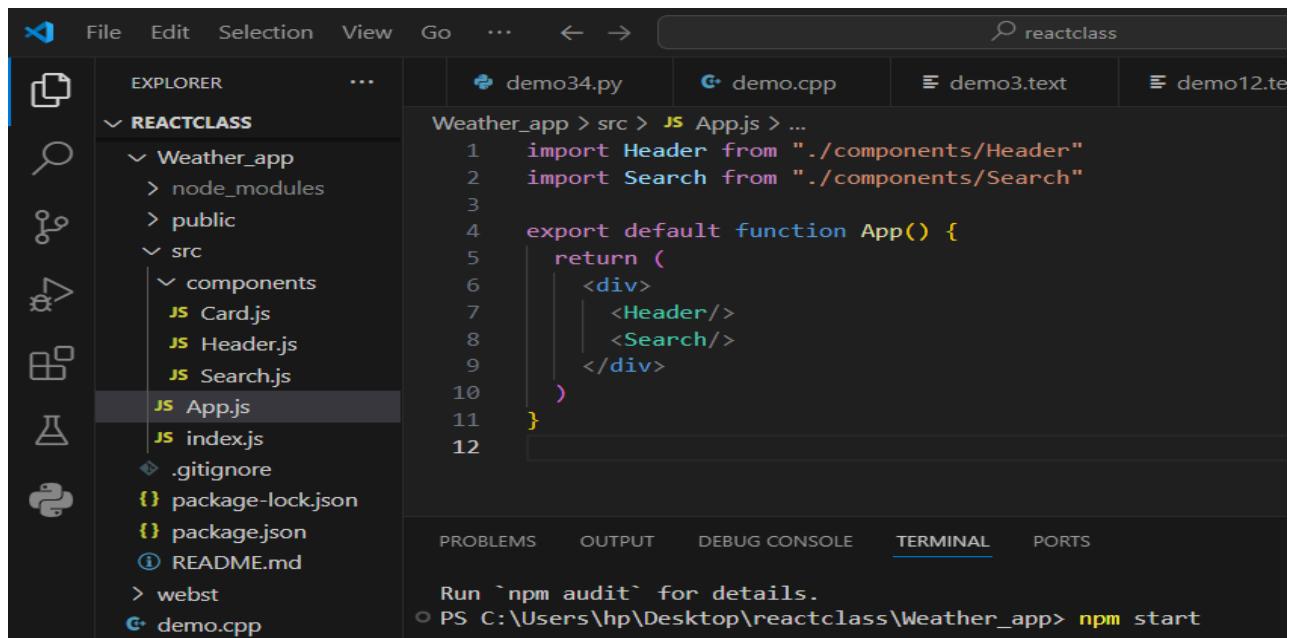
```
let Row = (props) => {
  let data = props.data
  return (
    <tr>
      <td>{data.id}</td>
      <td>{data.title}</td>
      <td>{data.completed ? "Completed" : "Pending"}</td>
      <td><button onClick={() => props.del(props.i)}>Delete</button></td>
    </tr>
  )
}
export default Row
```

ID	Title	Status	Action
1	delectus aut autem	Pending	<button>Delete</button>
2	quis ut nam facilis et officia qui	Pending	<button>Delete</button>
3	fugiat veniam minus	Pending	<button>Delete</button>
4	et porro tempora	Completed	<button>Delete</button>
5	laboriosam mollitia et enim quasi adipisci quia provident illum	Pending	<button>Delete</button>

Free public APIs you can use

Use Case	Free API
Weather	https://openweathermap.org/api
News	https://gnews.io/ https://newsdata.io/
Quotes	https://api.quotable.io
Jokes	https://official-joke-api.appspot.com/jokes/random
Crypto Prices	https://api.coingecko.com/api/v3/simple/price
Anime	https://api.jikan.moe/v4/

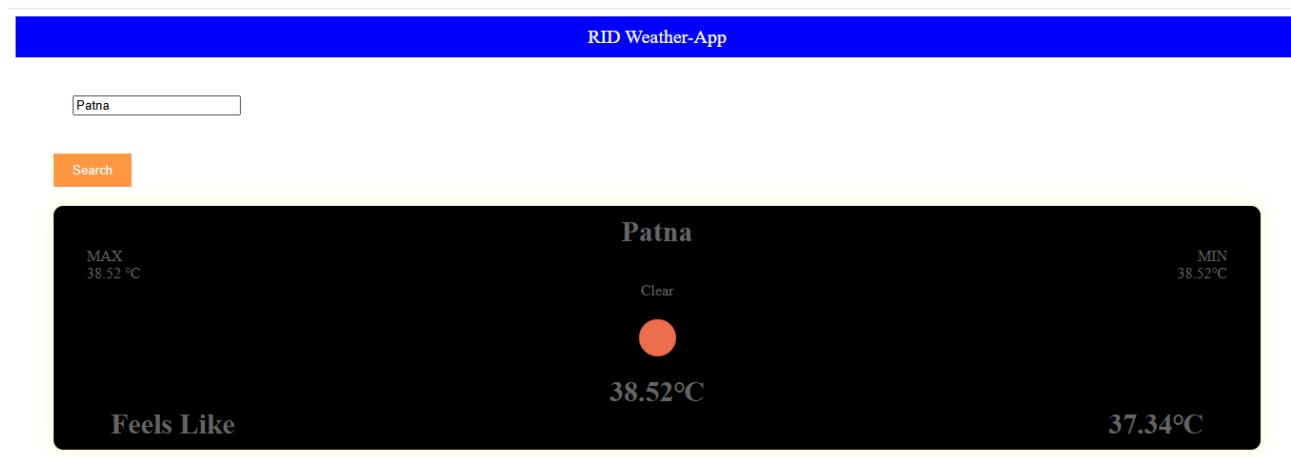
Weather Apps



```

File Edit Selection View Go ...
reactclass
EXPLORER ...
REACTCLASS
  Weather_app
    node_modules
    public
    src
      components
        Card.js
        Header.js
        Search.js
      App.js
      index.js
      .gitignore
      package-lock.json
      package.json
      README.md
    webst
    demo.cpp
demo34.py demo.cpp demo3.text demo12.te
Weather_app > src > JS App.js > ...
1  import Header from "./components/Header"
2  import Search from "./components/Search"
3
4  export default function App() {
5    return (
6      <div>
7        <Header/>
8        <Search/>
9      </div>
10   )
11 }
12
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run `npm audit` for details.
PS C:\Users\hp\Desktop\reactclass\Weather_app> npm start

```



App.js

```
import Header from "./components/Header"
import Search from "./components/Search"
export default function App() {
  return (
    <div>
      <Header/>
      <Search/>
    </div>
  )
}
```

Card.js: npm install axios

```
import { useState, useEffect } from "react"
import axios from "axios";
export default function Card(props) {
  const [main, setMain] = useState({});
  const [weather, setWeather] = useState({});
  const [name, setName] = useState("");
  const title = props.title;
  useEffect(() => {
    axios.get(
      `https://api.openweathermap.org/data/2.5/weather?q=${title}&units=metric&appid=eedd52cc943d9174f2c65178d154bdd` ).then((res) => {
        setMain(res.data.main);
        setWeather(res.data.weather[0]);
        setName(res.data.name);
        console.log(res.data.weather[0]);
      });
  }, [title]);
  const icon = `http://openweathermap.org/img/wn/${weather.icon}@2x.png`;

  return (
    <div style={{ backgroundColor: 'black', borderRadius: '10px', margin: '20px', padding: '10px', textAlign: 'center', color: '#646464', boxShadow: '0px 0px 20px #fff8cd' }}>
      <div style={{ fontWeight: 'bold', fontSize: '30px' }}>{name}</div>

      <div style={{ display: 'flex', justifyContent: 'space-between', flexDirection: 'row' }}>
        <div style={{ textAlign: 'left', paddingLeft: '25px' }}>
          <div>MAX</div>
          <div>{main.temp_max} &#8451;</div>
        </div>
        <div style={{ textAlign: 'right', paddingRight: '25px' }}>
          <div>MIN</div>
          <div>{main.temp_min}&#8451;</div>
        </div>
      </div>
    </div>
  )
}
```

```
import Header from "./components/Header"
import Search from "./components/Search"
function App() {
  return (
    <div>
      <Header/>
      <Search/>
    </div>
  )
}
export default App
```

```

<div>
  <div>{weather.main}</div>
  <div style={{ display: 'flex', justifyContent: 'space-around', flexDirection: 'row' }}>
    <img style={{ height: '80px', width: '80px' }} src={icon} alt="weather icon" />
  </div>
</div>

<div style={{ fontWeight: 'bold', fontSize: '30px' }}>
  {main.temp}&#8451;
  <div style={{ display: 'flex', justifyContent: 'space-between', flexDirection: 'row', paddingLeft: '50px', paddingRight: '50px' }}>
    <span>Feels Like </span>
    <span>{main.feels_like}&#8451; </span>
  </div>
</div>
</div>
)
}

```

Header.js

```

export default function Header() {
  return (
    <div style={{ backgroundColor: 'blue', color: '#fff', fontSize: '20px', padding: '10px', textAlign: 'center' }}>
      RID Weather-App
    </div>
  )
}

```



Search.js

```

import Card from "./Card"
import { useState } from "react";

export default function Search() {
  const [scr, setScr] = useState("");
  const [scrc, setScrc] = useState("");
  function handleChange(e) {
    setScr(e.target.value);
  }
  function handleClick(e) {
    setScrc(scr);
  }
  return (
    <div style={{ margin: '20px', position: 'relative' }}>
      <div style={{ padding: '20px', borderwidth: '2px', bordercolor: '#ffe05d', margin: '20px' }}>
        <input type="text" placeholder="City Name" value={scr} onChange={handleChange} />
      </div>
      <div style={{ margin: '20px', marginTop: '5px' }}>

```

```
<button style={{ backgroundColor: '#ff9642', color: 'fff', border: 'none', padding: '10px 20px', cursor: 'pointer' }} onClick={handleClick}>
  Search
</button>
</div>
/* conditional rendering is applicable below
If the value of scrc exists, then only the card component renders */
{scrc && <Card title={scrc} />}
</div>
)
}
```



Route in React

- Route is used to define **which component should be rendered** based on the **URL path**. It comes from react-router-dom library, which handles routing (navigation) in single-page applications.

Main useful component in route

1. <BrowserRouter> in React JS:

It is a component from **React Router** that enables navigation using the browser's URL (like a normal website) without reloading the page.

Why we use it: To create smooth, fast navigation between pages in a single-page React app.

How to use <BrowserRouter> in React JS:

- import { BrowserRouter } from 'react-router-dom';
- <BrowserRouter> {/* Your routes go here */} </BrowserRouter>

2. What is <Routes> in React JS:

- it is a container that holds all your route definitions and matches the URL to the correct component.

- **Why use:** To group & manage multiple <Route> components for different pages in a React app.

- **Syntax:**

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
</Routes>
```

- **How to use it:**

1. **Import:** import { Routes, Route } from 'react-router-dom';
2. **Use inside <BrowserRouter>:**

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes> </BrowserRouter>
```

3. What is <Route /> in React JS:

<Route /> defines a path and tells React which component to show when URL matches path.

- **Why we use it:** To display different components (pages) based on the URL in a React app.

How to use it (syntax):

1. **Import:** import { Route } from 'react-router-dom';
2. **Use inside <Routes>:** <Route path="/about" element={<About />} />

Full example:

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

1. What is <Link> in React JS:

<Link> is used to navigate between pages in a React app without reloading the browser.

Why we use it: To create clickable links for smooth page navigation in single-page applications (SPA).

How to use it (syntax):

1. **Import:** import { Link } from 'react-router-dom';
2. **Use in JSX:** <Link to="/about">Go to About</Link>

Example:

```
<Link to="/">Home</Link>
<Link to="/about">About</Link>
```

Main useful component in route in short

1. **<BrowserRouter>** – Enables navigation in a React app using browser URLs without reloading the page.
2. **<Routes>** – Holds and manages all the defined routes in the app.
3. **<Route />** – Maps a specific URL path to a component to display.
4. **<Link>** – Creates clickable links to navigate between pages without page reload.

How to install the react Router Dom

in terminal >> npm Install React Router DOM

How to import all in react

App.js or main.js or any component in react we import

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
```

Example-1: App.js

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
function App() {
  return (
    <div>
      <BrowserRouter>
        <Routes>
          <Route path='/' element={<h1>Home</h1>} />
          <Route path='/about' element={<h1>About</h1>} />
          <Route path='/gallery' element={<h1>Gallery</h1>} />
        </Routes>
      </BrowserRouter>
    </div>
  );} export default App;
```

← → ⌂ ⓘ localhost:3000/about

Why we use element={<Component />} in <Route>

- In React Router v6, the element prop is **mandatory** for defining what component should render when a route matches. You must pass a **React element** (like `<Home />`, `<About />`, etc.) to the element prop.

About

← → ⌂ ⓘ localhost:3000

Home

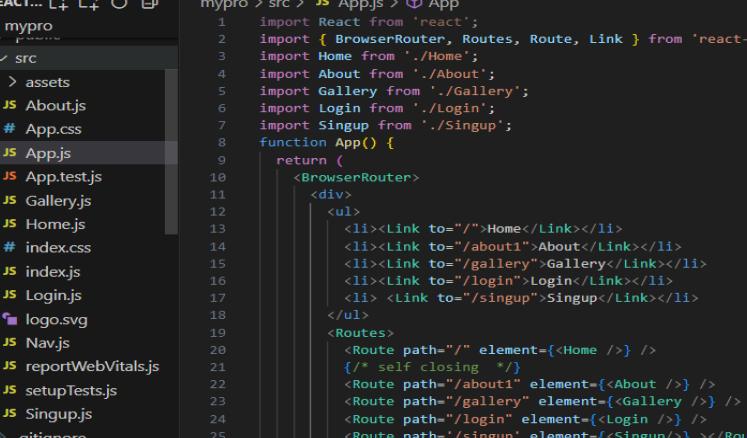
← → ⌂ ⓘ localhost:3000/gallery

Gallery

Note: change on browser url / then about then gallery get the result



Example-2: route within the page



The screenshot shows a code editor with the following structure and code:

EXPLORER (Left sidebar):

- REACT... (dropdown)
- mypro (selected project)
- src
- assets
- JS About.js
- # App.css
- JS App.js (selected)
- JS App.test.js
- JS Gallery.js
- JS Home.js
- # index.css
- JS index.js
- JS Login.js
- logo.svg
- JS Nav.js
- JS reportWebVitals.js
- JS setupTests.js
- JS Singup.js
- .gitignore
- { package-lock.json
- { package.json

OUTLINE (Bottom left):

JS App.js mypro\... (Active tab)

```
mypro > src > JS App.js > App
1  import React from 'react';
2  import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
3  import Home from './Home';
4  import About from './About';
5  import Gallery from './Gallery';
6  import Login from './Login';
7  import Singup from './Singup';
8  function App() {
9    return (
10      <BrowserRouter>
11        <div>
12          <ul>
13            <li><Link to="/">Home</Link></li>
14            <li><Link to="/about1">About</Link></li>
15            <li><Link to="/gallery">Gallery</Link></li>
16            <li><Link to="/login">Login</Link></li>
17            <li><Link to="/singup">Singup</Link></li>
18          </ul>
19          <Routes>
20            <Route path="/" element={<Home />} />
21            /* self closing */
22            <Route path="/about1" element={<About />} />
23            <Route path="/gallery" element={<Gallery />} />
24            <Route path="/login" element={<Login />} />
25            <Route path="/singup" element={<Singup />} />
26          </Routes>
27        </div>
28      </BrowserRouter>
29    )
30  }
```

JS index.js

JS Singup.js

JS Home.js mypro\...

App.js:

```
import React from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Gallery from './Gallery';
import Login from './Login';
import Singup from './Singup';
function App() {
  return (
    <BrowserRouter>
      <div>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about1">About</Link></li>
          <li><Link to="/gallery">Gallery</Link></li>
          <li><Link to="/login">Login</Link></li>
          <li><Link to="/singup">Singup</Link></li>
        </ul>
        <Routes>
          <Route path="/" element={<Home />} />
          {/* self closing */}
          <Route path="/about1" element={<About />} />
          <Route path="/gallery" element={<Gallery />} />
          <Route path="/login" element={<Login />} />
          <Route path='/singup' element={<Singup/>} /></Route>
        </Routes>
      </div>
    </BrowserRouter>
  );
}
export default App;
```

Home.js

```
import React from 'react'  
function Home() {  
  return (  
    <div>  
      <h1>welcome to home page </h1>  
    </div>  
)  
}  
export default Home
```

About.js

```
import React from 'react';
function About() {
  return <h2>This is the About Page</h2>;
}
export default About;
```

Gallery.js

```
import React from 'react'  
function Gallery() {  
  return (  
    <div>  
      <h1>welcome to gallery page </h1>  
    </div>  
)} export default Gallery
```

Login.js

```
import React from 'react'  
export default function Login() {  
  return (  
    <h1>welcome to login page </h1>  
  )  
}
```

Example-3: navigate the page from one components to another components

- npm install react-router-dom

Example-3 Folder Structure

```
src/
  ├── App.js
  ├── Nav.js
  ├── Home.js
  └── About.js
```

Home.js – Home Component

```
import React from 'react';
function Home() {
  return
    (

## Welcome to the Home Page

)
}
export default Home;
```

✓ How Routing Works

- BrowserRouter wraps the app and enables routing.
- Routes is a container for all Route definitions.
- Route maps a URL (path) to a component (element).
- Link is used for navigation without reloading the page.

⚠ Common Mistakes to Avoid

- Using href instead of Link: causes full page reload.
- Using incorrect path like ./home instead of /home in Route.
- Forgetting to wrap routes with BrowserRouter.

Nav.js – Navigation Component

```
import React from 'react';
import { Link } from 'react-router-dom';
function Nav() {
  return (
    <nav>
      <ul style={{ display: 'flex', gap: '20px', listStyle: 'none' }}>
        <li><Link to="/home-1">Home</Link></li>
        <li><Link to="/about-1">About</Link></li>
      </ul>
    </nav>
  );
}
export default Nav;
```

About.js – About Component

```
import React from 'react';
function About() {
  return <h2>This is the About Page</h2>;
}
export default About;
```

App.js

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Nav from './Nav';
import Home from './Home';
import About from './About';
function App() {
  return (
    <BrowserRouter>
      <Nav />
      <Routes>
        <Route path="/home-1" element={<Home />} />
        <Route path="/about-1" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

Home About

This is the About Page

↑ should Same



❖ Optional Enhancements

- Default route (redirect to /home):

```
js
<Route path="/" element={<Navigate to="/home" />} />
```

- 404 Page (Not Found):

```
js
<Route path="*" element={<NotFound />} />
```

- [Home](#)
- [About](#)
- [Gallery](#)
- [Login](#)
- [Singup](#)

welcome to home page

Example-4: app.js

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Gallery from './Gallery';
import Login from './Login';
import Singup from './Singup';
import Nav from './Nav';
function App() {
  return (
    <BrowserRouter>
    <Nav/>
    <Routes>
      <Route path="/" element={<Home />} />
      {/* self closing */}
      <Route path="/about1" element={<About />} />
      <Route path="/gallery" element={<Gallery />} />
      <Route path="/login" element={<Login />} />
      <Route path='/singup' element={<Singup />} />
    </Routes>
    </BrowserRouter>
  );
}
export default App;
```

Nav.js

```
import React from 'react';
import { Link } from 'react-router-dom';
function Nav() {
  return (
    <nav>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about1">About</Link></li>
        {/*  Use <a> tag for opening in new tab */}
        <li><a href="/gallery" target="_blank" rel="noopener noreferrer">Gallery</a></li>
        <li><Link to="/login">Login</Link></li>
        <li><Link to="/singup">Singup</Link></li>
      </ul>
    </nav>
  );
}
export default Nav;
```

Home.js

```
import React from 'react';
function Home() {
  return (
    <div>
      <h1>welcome to home page </h1>
    </div>
  );
}
export default Home;
```

Gallery.js

```
import React from 'react';
function Gallery() {
  return (
    <div>
      <h1>welcome to gallery page </h1>
    </div>
  );
}
export default Gallery;
```

About.js

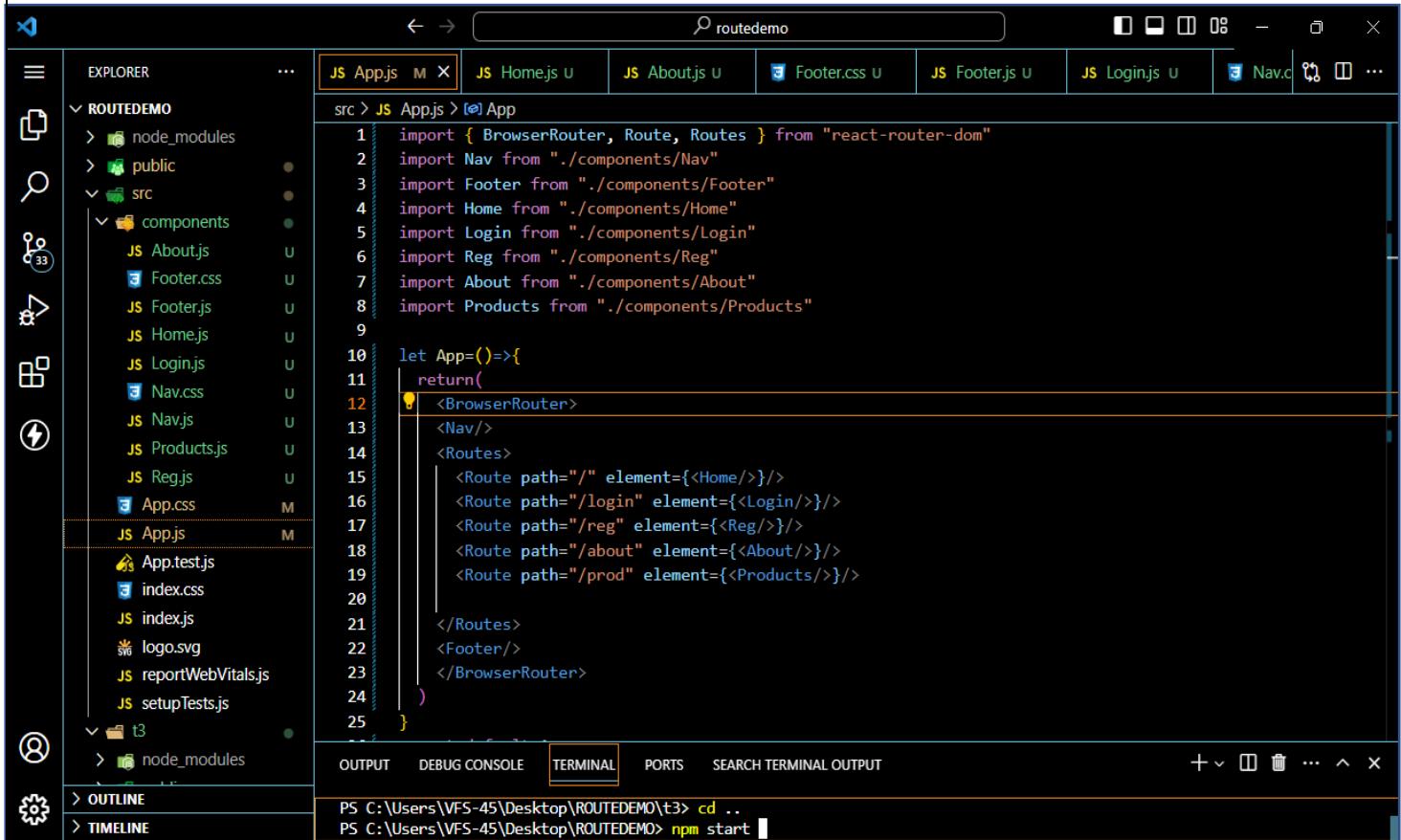
```
import React from 'react';
function About() {
  return <h2>This is the About Page</h2>;
}
export default About;
```

Login.js

```
import React from 'react';
export default function Login() {
  return (
    <h1>welcome to login page </h1>
  );
}
```



Example-5:



```

routedemo
  |- EXPLORER
  |- ROUTEDEMO
    |- node_modules
    |- public
    |- src
      |- components
        |- About.js
        |- Footer.css
        |- Footer.js
        |- Home.js
        |- Login.js
        |- Nav.css
        |- Nav.js
        |- Products.js
        |- Reg.js
        |- App.css
      |- App.js
        |- App.test.js
        |- index.css
        |- index.js
        |- logo.svg
        |- reportWebVitals.js
        |- setupTests.js
      |- t3
        |- node_modules
      |- OUTLINE
      |- TIMELINE
  |- JS App.js M X JS Home.js U JS About.js U JS Footer.css U JS Footer.js U JS Login.js U JS Nav.js U ...
  |- src > JS App.js > (App
    1 import { BrowserRouter, Route, Routes } from "react-router-dom"
    2 import Nav from "./components/Nav"
    3 import Footer from "./components/Footer"
    4 import Home from "./components/Home"
    5 import Login from "./components/Login"
    6 import Reg from "./components/Reg"
    7 import About from "./components/About"
    8 import Products from "./components/Products"
    9
   10 let App=()=>{
   11   return(
   12     <BrowserRouter>
   13       <Nav/>
   14       <Routes>
   15         <Route path="/" element={<Home/>}/>
   16         <Route path="/login" element={<Login/>}/>
   17         <Route path="/reg" element={<Reg/>}/>
   18         <Route path="/about" element={<About/>}/>
   19         <Route path="/prod" element={<Products/>}/>
   20       </Routes>
   21       <Footer/>
   22     </BrowserRouter>
   23   )
   24 }
   25
  
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT

PS C:\Users\VFS-45\Desktop\ROUTEDEMO\t3> cd ..
PS C:\Users\VFS-45\Desktop\ROUTEDEMO> npm start

Note add CDN link in index.css font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New', monospace;

➤ App.js:

```

import { BrowserRouter, Route, Routes } from "react-router-dom"
import Nav from "./components/Nav"
import Footer from "./components/Footer"
import Home from "./components/Home"
import Login from "./components/Login"
import Reg from "./components/Reg"
import About from "./components/About"
import Products from "./components/Products"
let App=()=>{
  return(
    <BrowserRouter>
      <Nav/>
      <Routes>
        <Route path="/" element={<Home/>}/>
        <Route path="/login" element={<Login/>}/>
        <Route path="/reg" element={<Reg/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/prod" element={<Products/>}/>
      </Routes>
    </BrowserRouter>
  )
}
  
```



```
</Routes>
<Footer/>
</BrowserRouter>
)}
export default App
```

➤ **Footer.js**

```
import './Footer.css'
let Footer=()=>{
  return(
    <div className="ft">@copyright reserved for twksaa.org</div>
  )
}
export default Footer
```

➤ **Footer.css:**

```
.ft{
  width: 100%;
  line-height: 30px;
  font-size: 20px;
  background-color: blue;
  color: white;
  text-align: center;
  font-style: italic;
}
```

➤ **About.js:**

```
let About=()=>{
  return(<div>About page</div>)
}
export default About
```

➤ **Home.js:**

```
let Home=()=>{
  return(
    <div>Home page</div>
  )
}
export default Home
```

➤ **Login.js:**

```
let Login=()=>{
  return(
    <div>Login page</div>
  )
}
export default Login
```

➤ **Nav.css:**

```
nav{
  width: 100%;
  height: 15vh;
  background-color: aqua;
  display: flex;
```



```
justify-content: space-evenly;
align-items: center;
}
Nav a{
  text-decoration: none;
  font-size: 20px;
  color: chocolate;
}
```

➤ **Nav.js:**

```
import { Link } from "react-router-dom"
import "./Nav.css"
let Nav=()=>{
  return(
    <nav>
      <Link to="/">Home <i class="fa-solid fa-house"></i></Link>
      <Link to="/login">Login</Link>
      <Link to="/about">About</Link>
      <Link to="/reg">Register</Link>
      <Link to="/prod">Products</Link>
    </nav>
  )
}
export default Nav
```

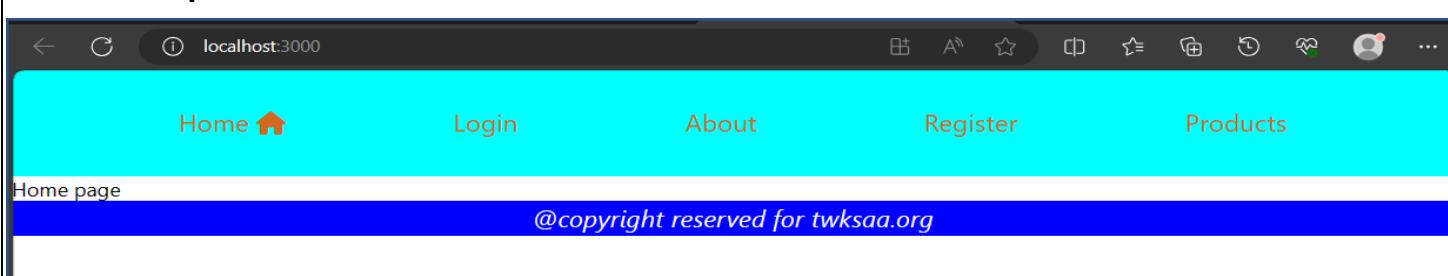
➤ **Product.js:**

```
let Products=()=>{
  return(
    <div>Products page</div>
  )
}
export default Products
```

➤ **Reg.js**

```
let Reg=()=>{
  return(
    <div>Reg page</div>
  )
}
export default Reg
```

Output:



Note: npm Install react-router-dom: for Example > npm i react-router-dom

Install axios: for Example, > npm i axios

Install create-react-app: npm install -g create-react-app

Create you react app now: create-react-app appname

Example-6

➤ App.js:

```
import { BrowserRouter, Route, Routes } from "react-router-dom"
import Nav from "./components/Nav"
import Footer from "./components/Footer"
import Home from "./components/Home"
import Login from "./components/Login"
import Reg from "./components/Reg"
import Product from "./components/Product"
let App=()=>{
  return(
    <BrowserRouter>
      <Nav/>
      <Routes>
        <Route path="/" element={<Home/>}/>
        <Route path="/login" element={<Login/>}/>
        <Route path="/reg" element={<Reg/>}/>
        <Route path="/prod" element={<Product/>}/>
      </Routes>
      <Footer/>
    </BrowserRouter>
  )
  export default App
```



➤ Footer.css

```
.ft{
  width: 100%;
  line-height: 30px;
  font-size: 20px;
  background-color: rgb(32, 32, 32);
  color: white;
  text-align: center;
  font-style: italic;
}
```

➤ Footer.js:

```
import './Footer.css'
let Footer=()=>{
  return(
    <footer className="ft">
      @copyright reserved to DKSRA.COM
    </footer>
  )
  export default Footer
```

➤ Home.css

```
.baner{
  width: 100%;
  height: 80vh;
  position: relative;
```



```
}

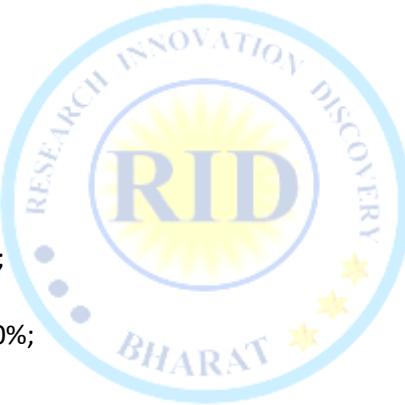
.baner img{
    width: 100%;
    height: 100%;
}

.baner h1{
    position: absolute;
    top:20px;
    left: 100px;
    color: red;
}

.coursecard{
    display: flex;
    width: 80%;margin-left: 10%;
    padding: 30px 0px;
    background-color: bisque;
    flex-wrap: wrap;
    justify-content: space-evenly;
    gap:10px;color: white;
    font-weight: bold;
    text-align: center;
}

.coursecard div{
    width: 35%;
    height: 200px;
    background-color: blue;
}

.coursecard h1{width: 100%;
    text-align: center;}
```



➤ Home.js

```
import { useEffect, useState } from "react"
import './Home.css'
let Home=()=>{
    let arr=[
        "https://images.pexels.com/photos/443446/pexels-photo-443446.jpeg?cs=srgb&dl=daylight-forest-glossy-443446.jpg&fm=jpg",
        "https://th.bing.com/th/id/R.be5169fe358cddf9251f1007b97d4cc8?rik=%2bCN4%2bGOBkw1ESw&riu=http%3a%2f%2fwww.pixelstalk.net%2fwp-content%2fuploads%2f2016%2f07%2fPeaceful-Desktop-Backgrounds.jpeg&ehk=23WIQbDrBOqACC%2bGzzRj9h%2fhZ4iGgWR0kDGAEM%2bHGjk%3d&risl=&pid=ImgRaw&r=0",
        "https://th.bing.com/th/id/R.9b81663a3934071f8de09681cec0684e?rik=DEV7m%2bUbPpmL2A&riu=http%3a%2f%2fwww.pixelstalk.net%2fwp-content%2fuploads%2f2016%2f08%2fCool-Nature-Background-Images-4523x2590.jpg&ehk=XxAKNdbLqeV84jakVaOYrmZWai83%2bRlxqGS7WF2x2%2bo%3d&risl=1&pid=ImgRaw&r=0",
    ]
```

```
https://th.bing.com/th/id/R.057b31be70cc73b9d0c829205f90e02f?rik=PxrTE7PkNVZoRg&iu=http%3a%2f%2fwww.pixelstalk.net%2fwp-content%2fuploads%2f2016%2f08%2fCool-nature-backgrounds-hd-resolution-1920x1080.jpg&ehk=hQzPW7PZMRmNSITgkm1V0lR4FopZ7yGKJtUMRedf8wM%3d&rls=&pid=ImgRaw&r=0",
"https://wallpapercave.com/wp/wp2555019.jpg",
"https://www.pixelstalk.net/wp-content/uploads/2016/06/Free-Images-Wallpaper-HD-Background.jpg"]
let txt=["Sangam Kumar","Shusil Kumar","Satyam Kumar","Sujeet Kumar","Raushni Kumari","T6"]
let [i,setI]=useState(0)
let fun=()=>{
  setI((i)=>(i+1)%6)
}
useEffect(()=>{
  let iid=setInterval(fun,2000)
  return ()=>clearInterval(iid)
},[])
return(
<>
  <div className="baner">
    <img src={arr[i]} />
    <h1>{txt[i]}</h1>
  </div>
  <div className="coursecard">
    <h1>Courses we offer</h1>
    <div>Python Full Stack</div>
    <div>HTML</div>
    <div>CSS</div>
    <div>JAVA SRCIPT</div>
    <div>REACT JS</div>
    <div>Java Full stack</div>
    <div>.Net Full Stack</div>
  </div> </> )
}
export default Home
```



➤ [Login.js](#):

```
let Login=()=>{
  return(
    <div>
      Login page
    </div>
  )
}
export default Login
```

➤ **Nav.css:**

```
nav{  
    width: 100%;height: 15vh;
```

```
background-color: blue;
display: flex;
justify-content: space-around;align-items: center;
nav a{
    text-decoration: none;font-size: 20px;
    color: white;
    font-weight: bold;}
```

➤ **Nav.js:**

```
import { Link } from "react-router-dom"
import "./Nav.css"
let Nav=()=>{
    return(
        <nav>
            <Link to="/">Home</Link> <Link to="/login">Login</Link>
            <Link to="/reg">Register</Link>
            <Link to="/prod">Products</Link>
        </nav>
    )
}
export default Nav
```

➤ **Product.js:**

```
let Product=()=>{
    return(
        <div>products page</div>
    )
}
export default Product
```

➤ **Reg.js**

```
let Reg=()=>{
    return(<div>Reg page</div>)
}
```



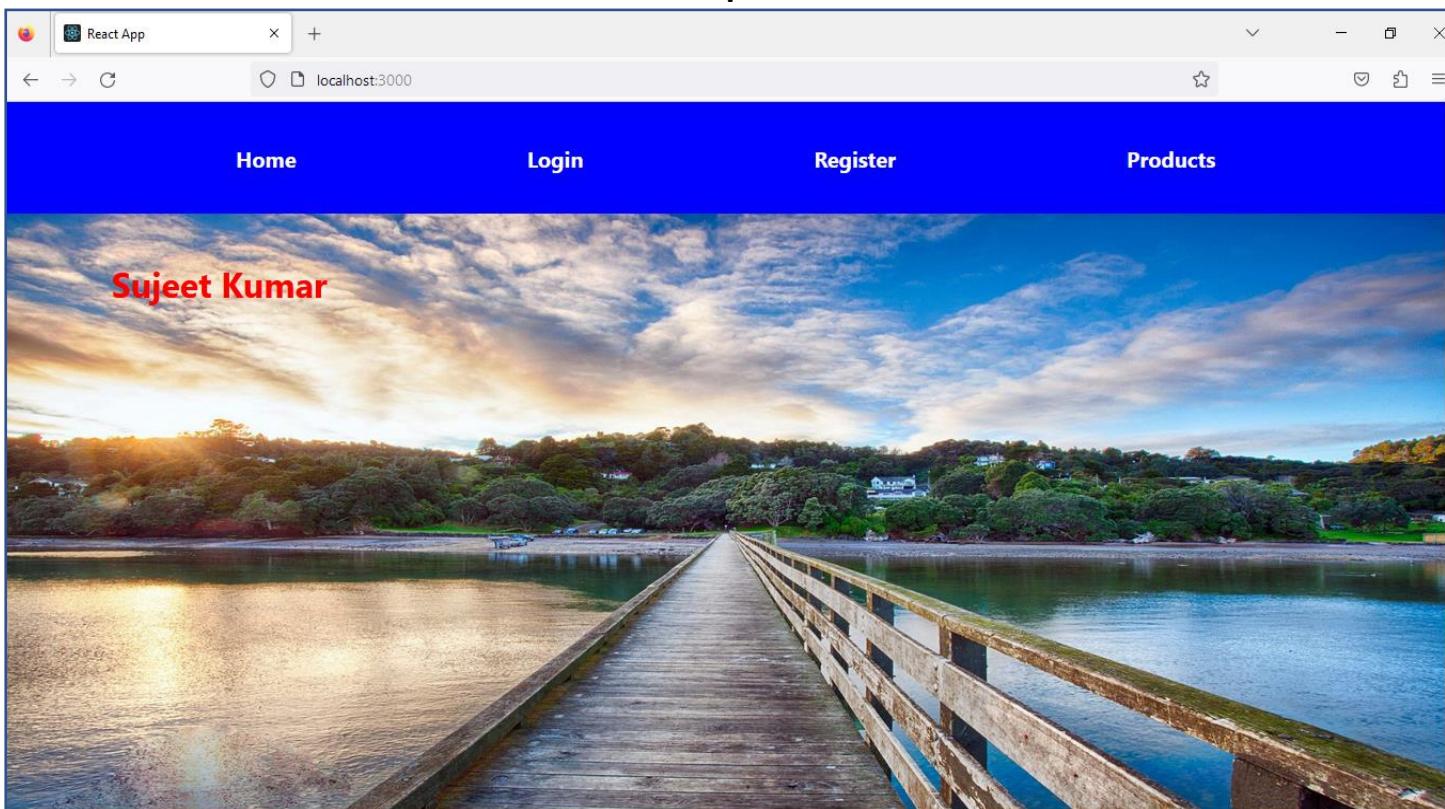
```
File Edit Selection View Go Run ... ← → ( appdemo
EXPLORER
APPDEMO
    raj
        > node_modules
        > public
            > src
                > components
                    # Footer.css
                    JS Footer.js
                    JS Home.css
                    JS Home.js
                    JS Login.js
                    # Nav.css
                    JS Nav.js
                    JS Product.js
                    JS Reg.js
                    # App.css
                JS App.js M
                JS App.test.js
                # index.css
                JS index.js
                logo.svg
                JS reportWebVitals.js
                JS setupTests.js
                .gitignore
                () package-lock.json
                () package.json
                README.md
            > OUTLINE
            > TIMELINE
JS App.js M ● JS Home.js U ● # Home.css U JS Reg.js U JS Product.js U JS Login.js U # Nav.css U # index.css
raj > src > JS App.js > [●] default
1 import { BrowserRouter, Route, Routes } from "react-router-dom"
2 import Nav from "./components/Nav"
3 import Footer from "./components/Footer"
4 import Home from "./components/Home"
5 import Login from "./components/Login"
6 import Reg from "./components/Reg"
7 import Product from "./components/Product"
8 let App=()=>{
9     return(
10         <BrowserRouter>
11             <Nav/>
12             <Routes>
13                 <Route path="/" element={<Home/>}/>
14                 <Route path="/login" element={<Login/>}/>
15                 <Route path="/reg" element={<Reg/>}/>
16                 <Route path="/prod" element={<Product/>}/>
17             </Routes>
18             <Footer/>
19         </BrowserRouter>
20     )
21 }
22 export default App
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
npm ERR! 404 tarball, folder, http url, or git url.

npm ERR! A complete log of this run can be found in: C:\Users\hp\AppData\Local\npm-cache\_logs\2023-10-01T11_36_26_654Z-debug-0.log
PS D:\appdemo\raj> npm start
○ Compiled successfully!

You can now view raj in the browser.

Local: http://localhost:3000
On Your Network: http://192.168.1.25:3000
```

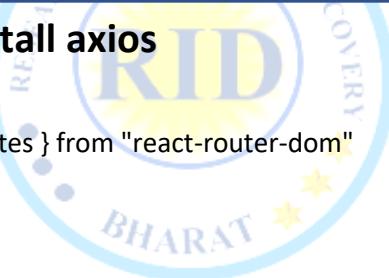
Output:



❖ Install axios: `npm install axios`

➤ Example-7: App.js:

```
import { BrowserRouter, Route, Routes } from "react-router-dom"
import Home from "./Home"
import Cart from "./Cart"
import Nav from "./Nav"
import './App.css'
import { useState } from "react"
import Gc from "./Gc"
let App=()=>{
  let [cart, setCart]=useState([])
  let [ctotal, setCtotal]=useState(0)
  let addcart=(item)=>{
    let res=cart.filter((prd)=>prd.id==item.id)
    if(res.length==0)
    {
      item={...item, "qty":1, "total":item.price}
      setCart([...cart, item])
      setCtotal(ctotal+item.price)
    }
    else{
      inc(item.id)
    }
  }
  let inc=(id)=>{
```



```

setCart(cart.map((item)=>{
  if(item.id==id)
  {
    setCtotal(ctotal+item.price)
    return {...item,"qty":item.qty+1,"total":item.total+item.price}
  }
  else{
    return item
  }
}))}

let dec=(id)=>{
  setCart(cart.map((item)=>{
    if(item.id==id && item.qty>1)
    {
      setCtotal(ctotal-item.price)
      return {...item,"qty":item.qty-1,"total":item.total-item.price}
    }
    else{
      return item
    }
  }))}

let del=(ind)=>{
  setCtotal(ctotal-cart[ind].total)
  cart.splice(ind,1)
  setCart([...cart])
}

let obj={"cart":cart,"addcart":addcart,"inc":inc,
"dec":dec,"del":del,"ctotal":ctotal}
return(
  <BrowserRouter>
  <Gc.Provider value={obj}>
  <Nav/>
  <Routes>
    <Route path="/" element={<Home/>}/>
    <Route path="/cart" element={<Cart/>}/>
  </Routes>
  </Gc.Provider>
  </BrowserRouter>
)
}

export default App

```

➤ App.css

```

*{
  margin: 0px;
  padding: 0px;
  outline: 0px;
}

```



➤ **Cart.js:**

```
import { useContext } from "react"
import Gc from "./Gc"
import './Cart.css'
let Cart=()=>{
  let obj=useContext(Gc)
  return(
    <div>
      <div className="con">
      {
        obj.cart.map((item,index)=>{
          return(
            <div className="card">
              <div className="img">
                <img src={item.images[0]} />
              </div>
              <h2>Name:{item.title}</h2>
              <h2>Desc:{item.description}</h2>
              <h2>Price:{item.price}</h2>
              <h2>Brand:{item.brand}</h2>
              <h2>Cat:{item.category}</h2>
              <h2><button onClick={()=>obj.dec(item.id)}>-</button>
                {item.qty}
              <button onClick={()=>obj.inc(item.id)}>+</button></h2>
              <h2>Total:{item.total}</h2>

              <button onClick={()=>obj.del(index)}>delcart</button>
            </div> )
        })
      </div>
      {obj.cart.length>0&& <div>GT:{obj.ctotal}</div>}
    </div> )
  export default Cart
```

➤ **Cart.css:**

```
.con{
  width: 100%;
  display: flex;
  justify-content: space-evenly;
  flex-wrap: wrap;
  gap:30px;
  padding: 30px 0px;
  background-color: aliceblue;
}.card{
  border: 2px solid gray;
  width: 40%;
  height: 250px;
  display: flex;
```

```
flex-direction: column;
justify-content: space-evenly;
padding: 10px;
}
.img{
  width: 100px;
  height: 100px;
}
.img img{
  width: 100%;
  height: 100%;
}
.card h2{
  font-size: 12px;
  color:darkcyan }
```

➤ **Home.js:**

```
import axios from "axios"
import { useContext, useEffect, useState } from "react"
import './Home.css'
import Gc from "./Gc"
let Home=()=>{
  let [prod, setProd]=useState([])
  let obj=useContext(Gc)
  useEffect(()=>{
    axios.get("https://dummyjson.com/products").then((res)=>{
      setProd(res.data.products)
    })
  },[])
  return(<div>
    <h1>Products:</h1>
    <div className="con">
      {
        prod.map((item)=>{
          return(
            <div className="card">
              <div className="img">
                <img src={item.images[0]}/>
              </div>
              <h2>Name:{item.title}</h2>
              <h2>Desc:{item.description}</h2>
              <h2>Price:{item.price}</h2>
              <h2>Brand:{item.brand}</h2>
              <h2>Cat:{item.category}</h2>
              <button onClick={()=>obj.addcart(item)}>Addcart</button>
            </div> )})
      </div>
    </div>)
  </div>)
  export default Home
```



➤ **Home.css**

```
.con{  
    width: 100%;  
    display: flex;  
    justify-content: space-evenly;  
    flex-wrap: wrap;  
    gap:30px;  
    padding: 30px 0px;  
    background-color: aliceblue;}  
.card{  
    border: 2px solid gray;  
    width: 40%;  
    height: 250px;  
    display: flex;  
    flex-direction: column;  
    justify-content: space-evenly;  
    padding: 10px;  
}  
.img{  
    width: 100px;  
    height: 100px;  
}  
.img img{  
    width: 100%;  
    height: 100%;  
}  
.card h2{  
    font-size: 12px;  
    color:darkcyan  
}
```



➤ **Gc.js**

```
import { createContext } from "react";  
let Gc=createContext({})  
export default Gc
```

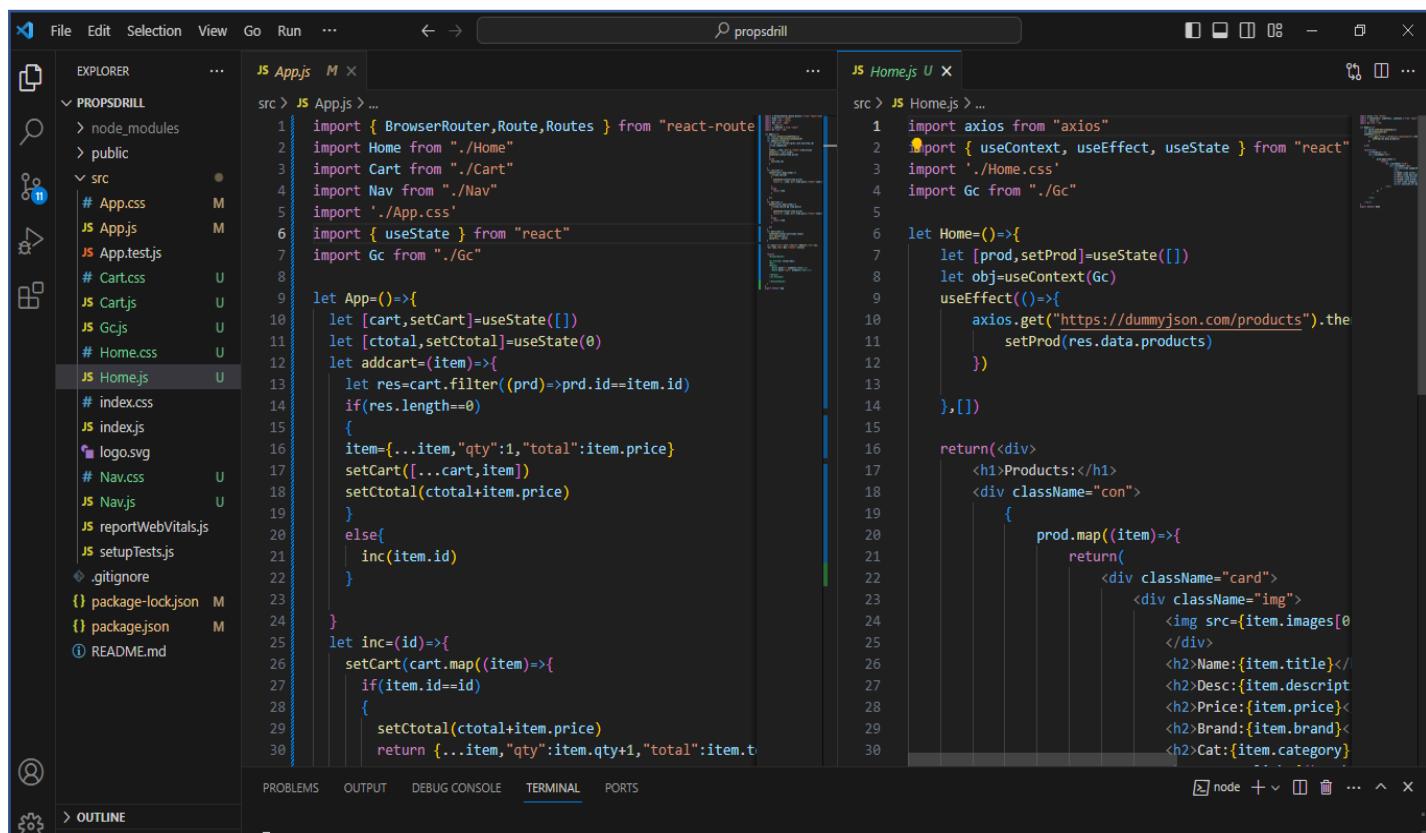
➤ **Nav.js:**

```
import {Link} from 'react-router-dom'  
import './Nav.css'  
import { useContext } from 'react'  
import Gc from './Gc'  
let Nav=()=>=>{  
    let obj=useContext(Gc)  
    return(<nav>  
        <Link to="/">Home</Link>  
        <Link to="/cart">Cart<button>{obj.cart.length}</button></Link>  
    </nav>)  
}  
export default Nav
```

➤ Nav.css:

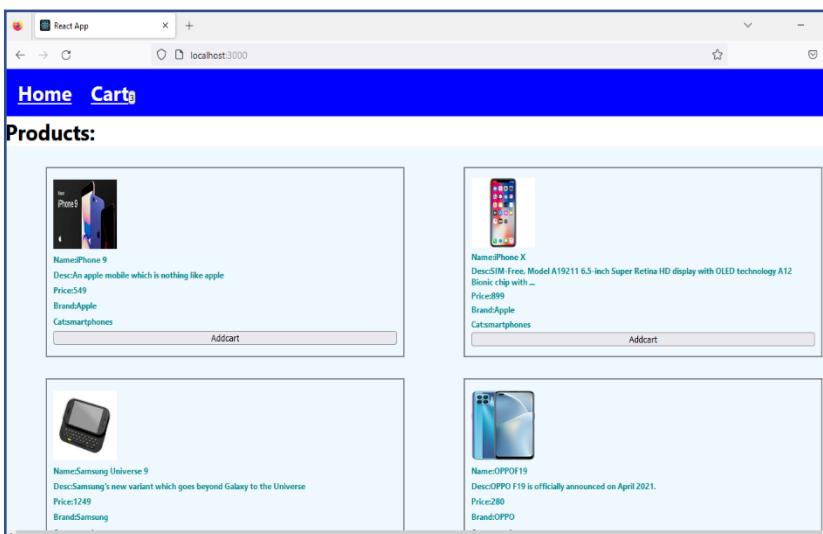
```
nav{
  width: 100%;
  height: 10vh;
  background: gray;
  display: flex;
  gap:30px;
  padding: 0px 20px;
  align-items: center;
}

nav a{
  font-size: 18px;
  color: blue;
}
```



The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure with files like 'node_modules', 'public', 'src', 'App.css', 'App.js', 'App.test.js', 'Cart.css', 'Cart.js', 'Gc.js', 'Home.css', 'Home.js', 'index.css', 'index.js', 'logo.svg', 'Nav.css', 'Nav.js', 'reportWebVitals.js', 'setupTests.js', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'.
- JS App.js:** The code for 'App.js' imports 'BrowserRouter', 'Route', 'Routes' from 'react-router-dom', and 'Home', 'Cart', 'Nav' from 'components'. It defines 'App' as a functional component that uses state for 'cart' and 'total'. It includes a 'addcart' function to handle item addition and an 'inc' function to handle quantity increments. It also includes a 'setCart' and 'setTotal' function to update state.
- JS Home.js:** The code for 'Home.js' imports 'axios' and 'useContext', 'useEffect', 'useState' from 'react'. It defines 'Home' as a functional component that uses 'Gc' from 'components'. It makes a GET request to 'https://dummyjson.com/products' to fetch data and then sets it to 'prod'. It then maps over the 'prod' data to create a list of products, each with a card and an 'Addcart' button.



Add user, Display user, Edit User Update user and delete user application React js:

➤ App.js

```
import { BrowserRouter, Route, Routes } from "react-router-dom"
import { useState } from "react"
import Adduser from "./Adduser"
import Nav from "./Nav"
import Deluser from "./Deluser"
import Dpusers from "./Dpusers"
import Edituser from "./Edituser"
import Gc from "./Gc"
import "./App.css"
let App=()=>{
  let [data,setData]=useState([])
  let addUser=(user)=>{
    setData([...data,user])
  }
  let delUser=(ind)=>{
    data.splice(ind,1)
    setData([...data])
  }
  let editUser=(user,ind)=>{
    data[ind]=user
    setData([...data])
  }
  let obj={"data":data,"fun":addUser,"edit":editUser,"del":delUser}
  return(
    <BrowserRouter>
    <Nav/>
    <Gc.Provider value={obj}>
    <Routes>
      <Route path="/" element={<Dpusers />}/>
      <Route path="/add" element={<Adduser />}/>
      <Route path="/edit" element={<Edituser />}/>
      <Route path="/del" element={<Deluser />}/>
    </Routes>
    </Gc.Provider>
  </BrowserRouter>
)
}
export default App
```



➤ **Deleteuser.js**

```
import { useContext } from "react"
import Gc from "./Gc"
let Deluser=()=>{
  let props=useContext(Gc)
  return(<div>
  {
    props.data.map((user,ind)=>{
      return(<li>
        {user}<button onClick={()=>props.del(ind)}>Delete</button>
      </li>
    })
  }
</div>
)
export default Deluser
```

➤ **Displayuser.js**

```
import { useContext } from "react"
import Gc from "./Gc"
let Dusers=()=>{
  let props=useContext(Gc)
  return(
    <div>
      {props.data.map((user)=>{
        return <li>{user}</li>
      })}
    </div>
  )
export default Dusers
```

➤ **Edituser.js**

```
import { useContext, useState } from "react"
import Gc from "./Gc"
let Edituser=()=>{
  let [user, setUser]=useState("")
  let [index, setIndex]=useState()
  let props=useContext(Gc)
  let edit=(name, ind)=>{
    setUser(name)
    setIndex(ind)
  }
  let upd=()=>{
    props.edit(user, index)
    setUser("")
  }
  let fun=(e)=>{
    setUser(e.target.value)
  }
}
```



```

return(
  <div>
    <input type="text" value={user} onChange={fun}/>
    <button onClick={upd}>update</button>
    <div>
    {
      props.data.map((user,ind)=>{
        return(<li>{user} <button onClick={()=>edit(user,ind)}>edit</button></li>)
      })
    </div>
  </div>
) }export default Edituser;

```

Gc.js

```

import { createContext } from "react";
let Gc=createContext({})
export default Gc

```



Data access from form

App.js



```

import { useState } from "react"
import "./App.css"
let App=()=>{
  let [data,setData]=useState({})
  let [lang,setLang]=useState([])
  let [users,setUsers]=useState([])
  let fun=(e)=>{
    setData({...data,[e.target.name]:e.target.value})
  }
  let fun1=(e)=>{
    if(e.target.checked)
    {
      setLang([...lang,e.target.value])
    }
    else{
      lang.splice(lang.indexOf(e.target.value),1)
      setLang([...lang])
    }
  }
  let add=()=>{
    setUsers([...users,...data,"lang": [...lang]])}
    setData({ "name": "", "pwd": "", "dob": "", "city": "", "ht": 5, "gen": "" })
    setLang([])
  }
  return(
    <div>
      <div>
        <input type="text" name="name" onChange={fun} value={data.name} placeholder="Enter

```



```

your name"/>
  <input type="password" name="pwd" onChange={fun} value={data.pwd} placeholder="Enter
your password"/>
  <input type="date" name="dob" onChange={fun} value={data.dob}/>
<input type="radio" name="gen" onChange={fun} value="male"
checked={data.gen==='male'} />Male
  <input type="radio" name="gen" onChange={fun} value="female"
checked={data.gen==='female'}/>female
  <input type="checkbox" value="Hindi" onChange={fun1}/>Hindi
  <input type="checkbox" value="Bhojpuri" onChange={fun1}/>Bhojpuri
  <input type="checkbox" value="Tamil" onChange={fun1}/>Tamil
  <input type="range" min="1" max="10" name="ht" onChange={fun} value={data.ht}/>
<select onChange={fun} name="city" value={data.city}>
  <option value="" disabled selected>Select</option>
  <option value=" Patna ">Patna</option>
  <option value=" Sasaram ">Sasaram</option>
  <option value=" Baheri">Baheri</option>
</select>
<button onClick={add}>ADD</button>
</div>
<table border={1}>
  <tr>
    <th>Name</th>
    <th>Password</th>
    <th>DOB</th>
    <th>Gen</th>
    <th>City</th>
    <th>Height</th>
    <th>Lang</th>
  </tr>
  {
    users.map((item)=>{
      return(<tr>
        <td>{item.name}</td>
        <td>{item.pwd}</td>
        <td>{item.dob}</td>
        <td>{item.gen}</td>
        <td>{item.city}</td>
        <td>{item.ht}</td>
        <td>{item.lang.toString()}</td>
      </tr>
    })
  })
</table>
</div>
)}
export default App

```

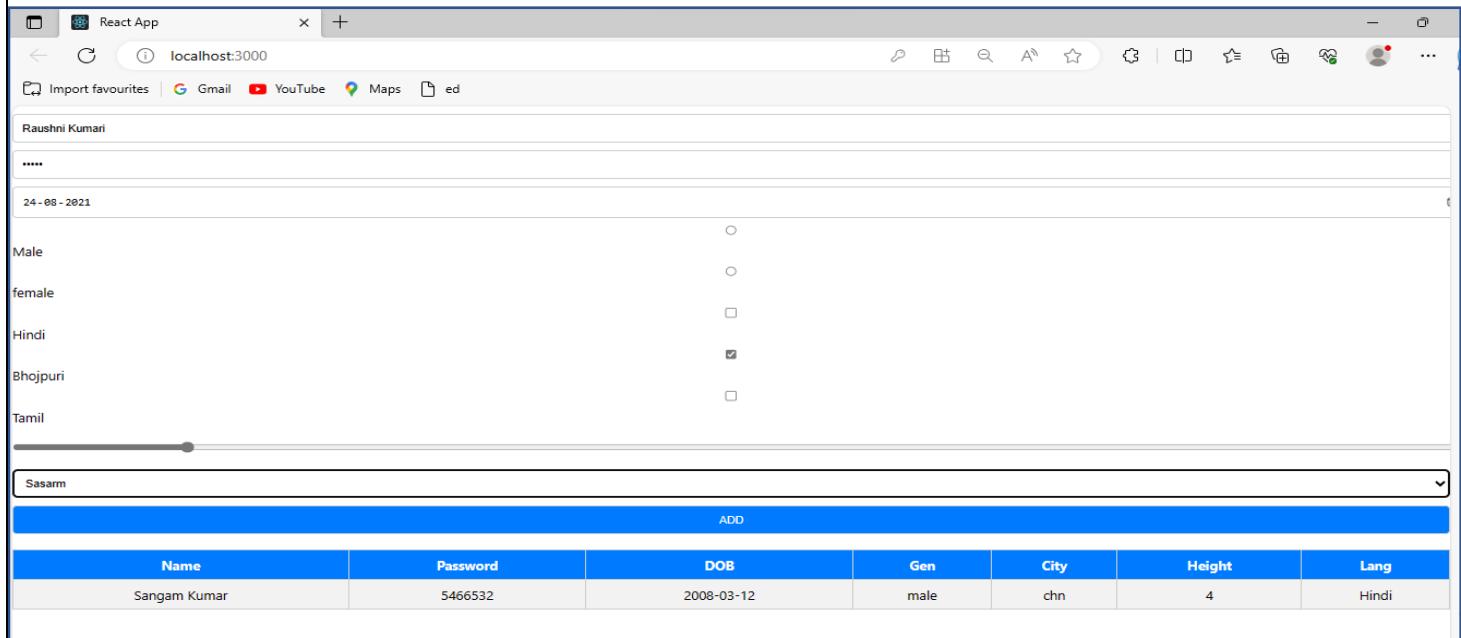


➤ **App.css**

```
.container {  
    width: 100%;  
    margin: 0 auto;  
    padding: 20px;  
    background-color: #f5f5f5;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
}  
  
input,  
select,  
button {  
    display: block;  
    margin: 10px 0;  
    width: 100%;  
    padding: 10px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
}  
  
button {  
    background-color: #007bff;  
    color: #fff;  
    cursor: pointer;  
}  
  
table {  
    width: 100%;  
    border-collapse: collapse;  
    margin-top: 20px;  
}  
  
table, th, td {  
    border: 1px solid #ccc;  
}  
  
th, td {  
    padding: 8px;  
    text-align: center;  
}  
  
th {  
    background-color: #007bff;  
    color: #fff;  
}  
  
tr:nth-child(even) {  
    background-color: #f2f2f2;  
}  
  
tr:hover {  
    background-color: #ddd;  
}
```



How to use material UI and Ant Design from



The screenshot shows a React application running on localhost:3000. The interface includes a sidebar with a navigation menu and a main content area. In the main area, there is a form with fields for Name, Password, DOB, Gender, City, Height, and Language. Below the form is a table with a single row of data. The table has columns for Name, Password, DOB, Gen, City, Height, and Lang. The data in the table is as follows:

Name	Password	DOB	Gen	City	Height	Lang
Sangam Kumar	5466532	2008-03-12	male	chn	4	Hindi

used pre define react js component

➤ App.js

```
import TextRating from "./TextRating"
import X from "./X"
import Y from "./Y"
let App=()=>{
  return(
    <div>
      <TextRating/>
      <X/>
      <div style={{"width":"100%","height":"500px"}>
        <Y/>
      </div>
    </div>
  )
}
export default App
```



➤ X.js

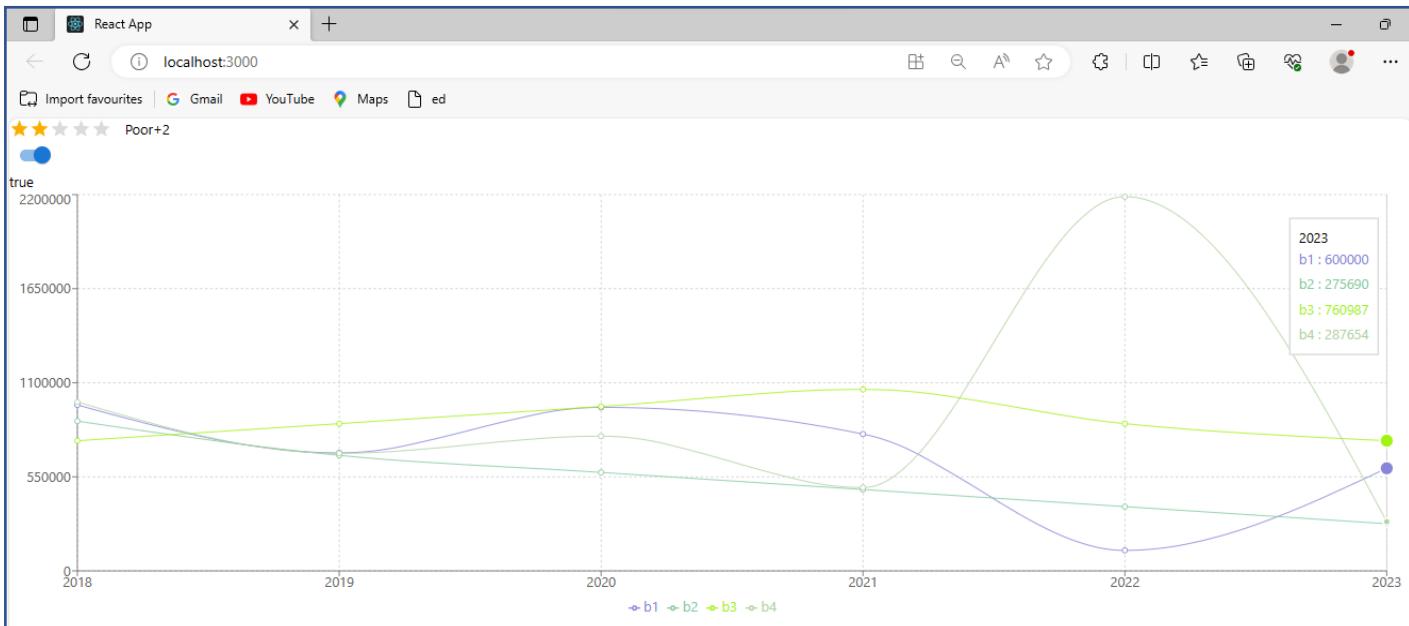
```
import * as React from 'react';
import Switch from '@mui/material/Switch';
export default function X() {
  const [checked, setChecked] = React.useState(true);
  const handleChange = (event) => {
    setChecked(event.target.checked);
  };
}
```

```
return (<div>
<Switch
  checked={checked}
  onChange={handleChange}
  inputProps={{ 'aria-label': 'controlled' }}
/>
<div>{checked+""}</div>
</div>
);}
```

Y.js

```
import React, { PureComponent } from 'react';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer } from
'recharts';
const data = [{"year":2018,"b1":970000,"b2":875690,"b3":760987,"b4":987654},
{"year":2019,"b1":690000,"b2":675690,"b3":860987,"b4":687654},
{"year":2020,"b1":956000,"b2":575690,"b3":960987,"b4":787654},
{"year":2021,"b1":800000,"b2":475690,"b3":1060987,"b4":487654},
{"year":2022,"b1":120000,"b2":375690,"b3":860987,"b4":2187654},
{"year":2023,"b1":600000,"b2":275690,"b3":760987,"b4":287654}
];
export default class Y extends PureComponent {
  render() {
    return (
      <ResponsiveContainer width="100%" height="100%">
        <LineChart
          width={500}
          height={300}
          data={data}
          margin={{{
            top: 5,
            right: 30,
            left: 20,
            bottom: 5,
          }}}
        >
          <CartesianGrid strokeDasharray="3 3" />
          <XAxis dataKey="year" />
          <YAxis />
          <Tooltip />
          <Legend />
          <Line type="monotone" dataKey="b1" stroke="#8884d8" activeDot={{ r: 8 }} />
          <Line type="monotone" dataKey="b2" stroke="#82ca9d" />
          <Line type="monotone" dataKey="b3" stroke="#a2f212" activeDot={{ r: 8 }} />
          <Line type="monotone" dataKey="b4" stroke="#b2d2a2" />
        </LineChart>
      </ResponsiveContainer>
    );
  }
}
```





TextRating.js

```
import * as React from 'react';
import Rating from '@mui/material/Rating';
import Box from '@mui/material/Box';
import StarIcon from '@mui/icons-material/Star';
const labels = {
  0.5: 'Useless',
  1: 'Useless+',
  1.5: 'Poor',
  2: 'Poor+',
  2.5: 'Ok',
  3: 'Ok+',
  3.5: 'Good',
  4: 'Good+',
  4.5: 'Excellent',
  5: 'Excellent+'};
function getLabelText(value) {
  return `${value} Star${value !== 1 ? 's' : ''}, ${labels[value]}`;
}
export default function TextRating() {
  const [value, setValue] = React.useState(2);
  const [hover, setHover] = React.useState(-1);
  return (
    <Box
      sx={{ width: 200, display: 'flex', alignItems: 'center' }}>
      <Rating
        name="hover-feedback"
        value={value}
        precision={0.5}
        onChange={setValue}
        onMouseEnter={() => setHover(1)}
        onMouseLeave={() => setHover(-1)}
        style={{ width: 150 }} />
      <span>${getLabelText(value)}</span>
    </Box>
  );
}
```



```
value={value}
precision={0.5}
getLabelText={getLabelText}
onChange={(event, newValue) => {
  setValue(newValue);
}}
onChangeActive={(event, newHover) => {
  setHover(newHover);
}}
emptyIcon=<StarIcon style={{ opacity: 0.55 }} fontSize="inherit" />/>
{value !== null && (
  <Box sx={{ ml: 2 }}>{labels[hover !== -1 ? hover : value]}</Box>
)
<div>{value}</div>
</Box>
);}
```

important question and answer

1. What is React.js?

- React.js is an open-source JavaScript library for building user interfaces.

2. What is JSX in React.js?

- JSX is a syntax extension for JavaScript that allows you to write HTML-like code in your JavaScript files.

3. What are the key features of React.js?

- React.js features a virtual DOM, component-based architecture, and one-way data binding.

4. What are components in React.js?

- Components are the building blocks of a React application, representing UI elements.

5. What is the difference between state and props?

- State is used for data that should be saved and modified by a component, while props are used to pass data from parent to child components.

6. How do you create a functional component in React?

- You can create a functional component by defining a JavaScript function that returns JSX.

7. How do you create a class component in React?

- You can create a class component by extending the `React.Component` class and defining a `render` method.

8. What is the purpose of the `render()` method in React components?

- The `render()` method is used to describe what the component's UI should look like.

9. What is the virtual DOM in React?

- The virtual DOM is a lightweight copy of the actual DOM, used to improve rendering performance.

10. What is the significance of keys in React lists?

- Keys are used to uniquely identify elements in a list and help React efficiently update the DOM.

11. What is the use of React Router?

- React Router is used for adding navigation and routing to React applications.

12. How do you handle user input in React forms?

- You can manage form input by using controlled components and handling the `onChange` event.

13. What is the purpose of `setState()` in React?

- `setState()` is used to update the state of a React component, triggering re-rendering.

14. What are props in React Router?

- Props are parameters that are passed to React components, allowing you to customize component behavior.

15. How do you pass data from a parent component to a child component?

- Data is passed from parent to child components by setting props in the child component.

16. What is React context?

- React context provides a way to share data across components without passing props manually.

17. What is conditional rendering in React?

- Conditional rendering involves showing or hiding components based on certain conditions.

18. How do you perform side effects in React?

- Side effects are managed using the `useEffect` hook in functional components.

19. What is Redux and why is it used in React?

- Redux is a state management library used to manage application state in a predictable and centralized manner.

20. What are actions and reducers in Redux?

- Actions are plain JavaScript objects that describe what happened, and reducers specify how the application's state changes.

21. How do you connect a React component to the Redux store?

- You can use the `connect` function from the `react-redux` library to connect a component to the store.

22. What is the purpose of the `mapStateToProps` and `mapDispatchToProps` functions in React Redux?

- `mapStateToProps` maps the state from the store to component props, while `mapDispatchToProps` maps action creators to component props.

23. What is the difference between React and React Native?

- React is used for building web applications, while React Native is used for building mobile applications for iOS and Android.

24. How do you handle navigation in a React Native app?

- Navigation in React Native can be managed using libraries like React Navigation.

25. What is the purpose of Redux Thunk in Redux?

- Redux Thunk is a middleware for handling asynchronous actions in Redux.

26. What is React Hook Form in React.js?

- React Hook Form is a library for managing forms in React using hooks.

27. What is the significance of the `keyExtractor` function in React Native?

- `keyExtractor` is used to extract unique keys for items in a FlatList or a SectionList component in React Native.

28. What is a Higher Order Component (HOC) in React?*

- A Higher Order Component is a pattern for sharing behavior among multiple React components.

29. What are the benefits of using functional components over class components?

- Functional components are simpler, have better performance, and support hooks for managing state and side effects.

30. What are hooks in React and why are they used?

- Hooks are functions that allow functional components to manage state and side effects, improving code reuse and component logic.

31. How does React handle security vulnerabilities like Cross-Site Scripting (XSS)?

- React provides built-in protections to prevent XSS attacks, such as escaping dynamic content by default.

32. What is the purpose of the `useMemo` hook in React?



- `useMemo` is used to memoize expensive calculations and optimize component performance.

33. What is React Portals and when would you use them?

- Portals allow rendering components outside the current DOM hierarchy, useful for modals, tooltips, and other overlays.

34. What is the significance of the `Fragment` component in React?

- The `Fragment` component is a wrapper that doesn't create extra DOM elements, useful for grouping elements.

35. How do you implement server-side rendering (SSR) in React applications?

- You can use libraries like Next.js to enable SSR in React applications.

36. What is code splitting in React and why is it important?

- Code splitting is the process of splitting your code into smaller, more manageable pieces, improving load times and performance.

37. How do you handle errors in React components?

- You can use error boundaries and the `componentDidCatch` lifecycle method to handle errors gracefully.

38. What is the difference between `ref` and `key` in React?

- `ref` is used to access the underlying DOM element or React component, while `key` is used for reconciliation in lists.

39. How do you update the state of a component in React?

You should never modify the state directly; instead, use the `setState` method to update the state.

40. What is the purpose of the `children` prop in React components?

The `children` prop is a special prop that allows you to pass arbitrary JSX content between opening and closing tags.

41. What are controlled components in React forms?

Controlled components have their value controlled by React state, making it easy to handle form input.

42. What is the purpose of the `useContext` hook in React?

The `useContext` hook allows functional components to access the context without a consumer component.

43. What are the benefits of using Redux for state management in React applications?

Redux provides a centralized, predictable, and easily testable way to manage state in complex



Node js / Express JS **Backend Notes**



Er. Rajesh Prasad(B.E, M.E)
Founder: TWF & RID Org.

- **RID ORGANIZATION** यानि **Research, Innovation and Discovery** संस्था जिसका मुख्य उद्देश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो।
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद **इस RID संस्था** की स्थपना किया हूँ।
- Research, Innovation & Discovery में रुचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियों से मैं आवाहन करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुद्धि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें।

त्वक्सा Node JS के इस ई-पुस्तक में आप Node JS से जुड़ी सभी बुनियादी अवधारणाएँ सीखेंगे। मुझे आशा है कि इस ई-पुस्तक को पढ़ने के बाद आपके ज्ञान में वृद्धि होगी और आपको कंप्यूटर विज्ञान के बारे में और अधिक जानने में रुचि होगी।

“In this E-Book of TWKSAA Node Js you will learn all the basic concepts related to Node Js. I hope after reading this E-Book your knowledge will be improve and you will get more interest to know more thing about computer Science”.

Online & Offline Class:

Python, Web Development, Java, Full Stack Course, Data Science, UI/UX Training, Internship & Research

करने के लिए Message/Call करें. 9202707903 E-Mail_id: ridorg.in@gmail.com

Website: www.ridbharat.com

RID हमें क्यों करना चाहिए ?

(Research)	(Innovation)	(Discovery)
अनुसंधान हमें क्यों करना चाहिए ? Why should we do research? 1. नई ज्ञान की प्राप्ति (Acquisition of new knowledge) 2. समस्याओं का समाधान (To Solving problems) 3. सामाजिक प्रगति (To Social progress) 4. विकास को बढ़ावा देने (To promote development) 5. तकनीकी और व्यापार में उन्नति (To advances in technology & business) 6. देश विज्ञान और प्रौद्योगिकी के विकास (To develop the country's science & technology)	नवीनीकरण हमें क्यों करना चाहिए ? Why should we do Innovation? 1. प्रगति के लिए (To progress) 2. परिवर्तन के लिए (For change) 3. उत्पादन में सुधार (To Improvement in production) 4. समाज को लाभ (To Benefit to society) 5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition) 6. देश विज्ञान और प्रौद्योगिकी के विकास (To develop the country's science & technology)	खोज हमें क्यों करना चाहिए ? Why should we do Discovery? 1. नए ज्ञान की प्राप्ति (Acquisition of new knowledge) 2. अविष्कारों की खोज (To Discovery of inventions) 3. समस्याओं का समाधान (To Solving problems) 4. ज्ञान के विकास में योगदान (Contribution to development of knowledge) 5. समाज के उन्नति के लिए (for progress of society) 6. देश विज्ञान और तकनीक के विकास (To develop the country's science & technology)

What is Node.js?

- Node.js is an **open-source, cross-platform** JavaScript runtime environment that allows you to run JavaScript code **outside the browser** — typically on the **server side**.
- It uses the **V8 engine** (developed by Google for Chrome) to execute JavaScript code, making it **fast and efficient**. V8 engine + c++=node js
- With Node.js, you can build backend services like APIs, web servers, real-time apps (e.g., chat apps), and more — all using JavaScript.

❖ History of Node.js:

➤ Event Details

- **Created By** :→ Ryan Dahl
- **First Released** :→ 2009
- **Written In** :→ C, C++, JavaScript
- **Current Maintainer** :→ OpenJS Foundation (previously managed by Joyent)
- **Goal** :→ To create scalable, fast, and non-blocking web servers using JavaScript on the backend

❖ Key Features of Node.js

➤ Feature Description

- **Fast Execution** Uses Google's V8 engine which compiles JS to machine code
- **Event-Driven Architecture** Uses events and callbacks for asynchronous tasks
- **Non-blocking I/O** Handles many requests at once without waiting for one to complete
- **Single Language Stack** JavaScript for both frontend and backend
- **Huge Package Ecosystem** npm (Node Package Manager) offers 1M+ libraries
- **Modular Architecture** Easily break code into reusable modules
- **Built-in Modules** Like http, fs, path, etc. to build without external libraries

❖ How Node.js Works

- Node.js uses a **single-threaded event loop** and **non-blocking I/O** to handle multiple concurrent requests.

❖ Event Loop & Non-blocking I/O

- When a request comes in (e.g., reading a file or hitting an API), Node.js **does not wait** for it to finish.
 - Instead, it assigns the task to a **worker thread** (background task) and **continues handling other requests**.
 - Once the I/O operation is done, the result is passed back via a **callback**.
- This makes Node.js highly scalable — it can handle thousands of requests with a single thread.

Node.js vs Other Backend Technologies

➤ Feature	Node.js	PHP / Python / Java
• Language	JavaScript	PHP, Python, Java, etc.
• Concurrency	Non-blocking, Event-driven	Multi-threaded, blocking I/O
• Speed	Very fast (V8 engine)	Slower (depends on language)
• Threading	Single-threaded (event loop)	Multi-threaded
• Use Case	Real-time apps, APIs	CMS, ML, large enterprise
• Community	Very active, npm ecosystem	Large (especially for Java, Python)
• Learning Curve	Easier for JavaScript devs	Depends on the language

How to install Node.js & npm

❖ What is NPM?

- NPM (Node Package Manager) is automatically installed with Node.js. It is used to manage **JavaScript packages/libraries**.

❖ What is a Package? - it is a folder containing a package.json file and code (modules), used to add specific functionality to your Node.js project.

Example: express, mongoose, nodemailer

❖ What is a Library in Node.js?

- A library is a collection of reusable code that performs common tasks, which you can import and use in your Node.js app.
 - Example: lodash (for utilities), axios (for HTTP requests)

❖ How to Install:

❖ Windows

1. Go to: <https://nodejs.org>
2. Download the **LTS (Long Term Support)** version
3. Run the installer → Accept default settings → Finish

❖ Verify Installation:

- node -v # Checks Node.js version
- npm -v # Checks NPM version

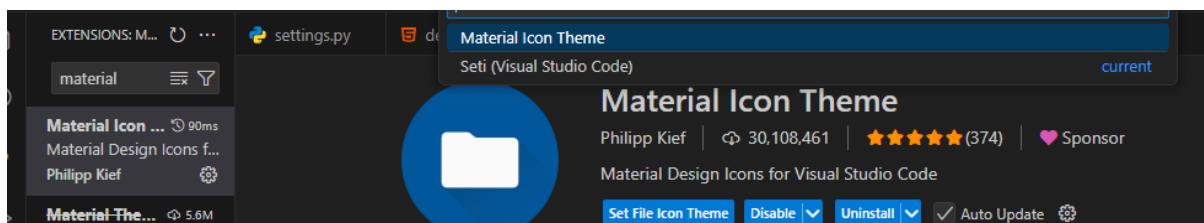
❖ Running a Simple Node.js Script

1. Open any code editor (like VS Code)
2. Create a file named app.js
3. Write a simple script:

```
console.log("Hello, Node.js!");
```
4. Run the script in terminal:
node app.js

➤ **Output will be:** Hello, Node.js!

Note: - Material Icon Theme most popular extension used for file and folder icons in vs code



How to run the first Node js program

1. Create a JavaScript file

Example: app.js

2. Write code in app.js

Example: `console.log ("Hello Everyone welcome to rid Bharat")`

3. Open terminal in the file's folder

In VS Code: Right-click → "Open in Integrated Terminal"

Run the file: - node app.js

Output: - Hello everyone welcomes to rid Bharat



Nodemon, npm start and npm init in Node.js

❖ What is Nodemon?

- Nodemon is a tool that automatically restarts your Node.js app whenever you make changes to your code.

❖ How to Install Nodemon (via npm)

Option 1: Install Globally(for any project) Check if installed:

- npm install -g nodemon
- nodemon -v (it will node.js current version)

Option 2: Install Locally (only in one project) > npm install --save-dev nodemon

❖ What is npm start?

- npm start is a command that runs **start script** defined in your package.json file.

❖ Why we use it?:- To easily start your Node.js application with a simple command.

❖ What is npm init?

- npm init is a command used to **create a package.json file** in a Node.js project.

Why we use it?

- To define the project's metadata (name, version, description, etc.)
- To manage **dependencies, scripts, and project configuration**
- It's required for installing and managing packages with npm

Note-1: - It helps organize and track everything your Node.js project needs.

Note-2: package.json file (created by npm init) works like the **overall project settings/configuration** file in a Node.js project.

❖ for open package.json file Run this command in your project folder

- npm init -y

❖ in package.json, add this script:

```
"scripts": {
  "start": "nodemon app.js"
}
```

❖ Why we use it:

1. Simplifies your command

Instead of typing this every time:

- nodemon app.js

You can just type:

- npm start

Note: - You can even change "start" to run any file you want (like index.js, server.js, etc.).

>> npm init

- **package name:** The project's name (default node_js; you wrote server.js which is usually a filename).
- **version:** Project version (default 1.0.0).
- **description:** Short info about the project (new project).
- **git repository:** URL of your project's Git repo (optional, left blank).
- **keywords:** Words to help find your project (optional).
- **license:** License type (default ISC).

These details create your project's package.json file.

Before adding nodemon

```
package.json > ...
1  {
2    "name": "djangopro",
3    "version": "1.0.0",
4    "main": "demo.js",
5    "scripts": {
6      "test": "echo \\"Error: no test \\
7      \",
8      "keywords": [],
9      "author": "",
10     "license": "ISC",
11     "description": ""
12   }
13 }
```

After adding nodemon

```
package.json > ...
1  {
2    "name": "djangopro",
3    "version": "1.0.0",
4    "description": "",
5    "main": "demo.js",
6    "scripts": {
7      "start": "nodemon demo.js",
8      "test": "echo \\"Error: no test specified\\" & exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC"
13 }
```



```
package.json > ...
1  {
2    "name": "djangopro",
3    "version": "1.0.0",
4    "description": "",
5    "main": "demo.js",
6    "scripts": {
7      "start": "nodemon demo.js",
8      "test": "echo \\"Error: no test specified\\" & exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC"
13 }
```

Note: npm init -y is used to quickly create a default package.json file with all standard settings without asking questions.



Module in Node.js

(Framework → Module → Function → Code → Instruction → Command)

Framework Hierarchy

1. **Framework** – collection of modules. (**Installed in the project**)
 2. **Module** – A collection of functions. (**Imported into files**)
 3. **Function** – A block of code. (**Needs to be called to run**)
 4. **Code** – A set of instructions. (*runs line by line*)
 5. **Instruction** – A single command. (**Runs directly**)

(write in any programming language like **JS or python**)

Examples:

1. **Framework:** Express, NestJS ([installed via npm](#)).
 2. **Module:** Core (fs, path) or third-party ([axios](#), [lodash](#)).
 3. **Function:** Callable blocks like `fs.readFile()`, `app.get()`.
 4. **Code:** Logical blocks (e.g., if, loops).
 5. **Instruction:** Single statements (`console.log ("learn Coding")`).

predefined methods/functions generally we called:

Technologies & Their

1. **React.js** – Hooks
 2. **Django** – Modules / Packages
 3. **Node.js** – Core Modules
 4. **Express.js** – Middleware / Router Methods
 5. **Python** – Built-in Functions
 6. **JavaScript** – Built-in Methods / Web APIs
 7. **Java** – Standard Library Methods
 8. **C++** – Standard Library Functions (STL)
 9. **Angular** – Services / Lifecycle Hooks
 10. **Flask** – Modules / Built-in Functions
 11. **Ruby on Rails** – Helpers / Gems
 12. **.NET (C#)** – Framework Classes
 13. **Spring Boot** – Annotations / Beans

How to Create, Export, and Import Modules in Node.js

1. **Using import (ES Modules) :- Syntax import / export**
 2. **Using require (CommonJS) :- require / module.exports**

1. Using import (ES Modules)

Step 1: Create module with any name with .js and write some function like : service.js e.g., add(), multiply()

Step-2: Export functions in botton in service.js file `export { function-1, function-2,... } Ex: export { add, multiply }`

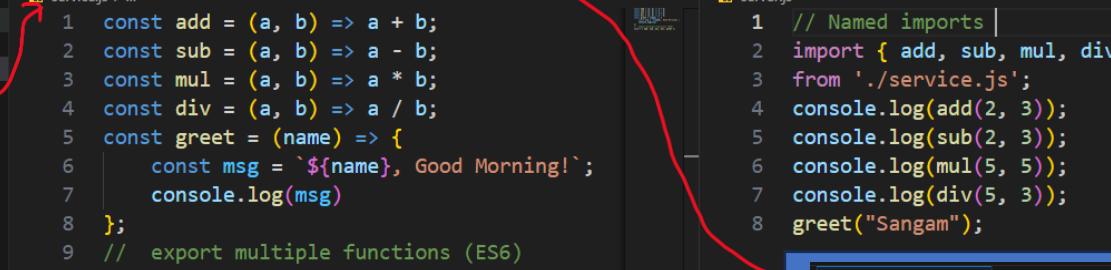
Step-3: Update package.json

Example:- { "type": "module", **// Enable ES Modules** }

Step-3: Create server.js to use the exported functions.

1: Import functions in server.js using: `import { functions } from 'firebase'`

Example: import { add, multiply } from './service.js'



```
service.js
1 const add = (a, b) => a + b;
2 const sub = (a, b) => a - b;
3 const mul = (a, b) => a * b;
4 const div = (a, b) => a / b;
5 const greet = (name) => {
6     const msg = `${name}, Good Morning!`;
7     console.log(msg)
8 };
9 //  export multiple functions (ES6)
10 export { add, sub, mul, div, greet };
11

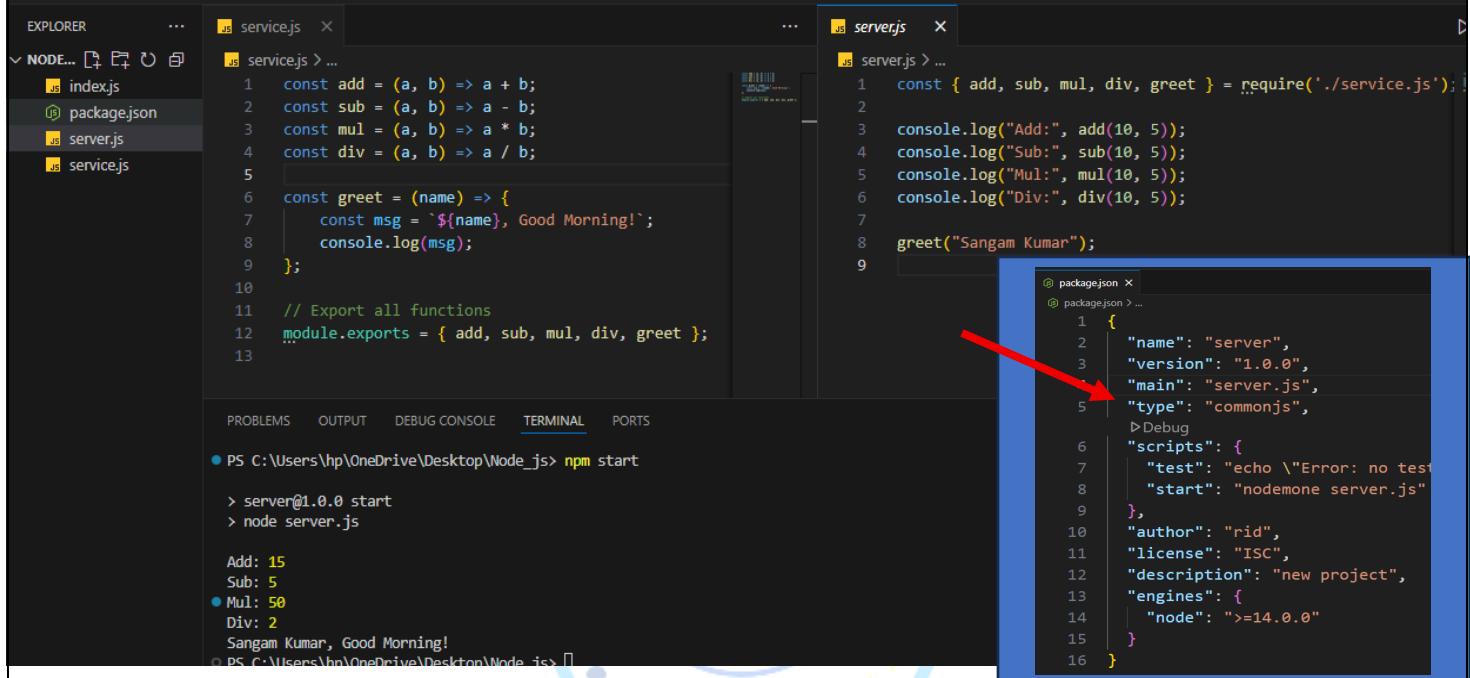
package.json
1 {
2     "name": "server",
3     "version": "1.0.0",
4     "main": "server.js",
5     "type": "module",
6     "scripts": {
7         "test": "echo \\\"Error: no test\\\"",
8         "start": "node server.js"
9     },
10    "author": "rid",
11    "license": "ISC",
12    "description": "new project",
13    "engines": {
14        "node": ">=14.0.0"
15    }
16 }
```

2. Using require (CommonJS)

How to Create, Export, and Import Modules in Node.js using require

1. **Create** service.js and write functions (e.g., add(), multiply()).
2. **Export** functions using: module.exports = { add, multiply };
3. **No need** to update package.json (CommonJS is default).
4. **Create** server.js to use the module.
5. **Import** functions using: const { add, multiply } = require('./service.js');
6. **Run using:** node server.js or npm start

Example:



```
service.js
1 const add = (a, b) => a + b;
2 const sub = (a, b) => a - b;
3 const mul = (a, b) => a * b;
4 const div = (a, b) => a / b;
5
6 const greet = (name) => {
7   const msg = `${name}, Good Morning!`;
8   console.log(msg);
9 }
10
11 // Export all functions
12 module.exports = { add, sub, mul, div, greet };

server.js
1 const { add, sub, mul, div, greet } = require('./service.js');
2
3 console.log("Add:", add(10, 5));
4 console.log("Sub:", sub(10, 5));
5 console.log("Mul:", mul(10, 5));
6 console.log("Div:", div(10, 5));
7
8 greet("Sangam Kumar");
9

package.json
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "server.js",
5   "type": "commonjs",
6   "scripts": {
7     "test": "echo \\"Error: no test specified\\" & exit 1",
8     "start": "nodemon server.js"
9   },
10  "author": "rid",
11  "license": "ISC",
12  "description": "new project",
13  "engines": {
14    "node": ">14.0.0"
15  }
16 }
```

Difference between import and require

1. Using ES Modules (ESM) — import / export

- **Syntax:** Uses import and export keywords.
- **File type:** Treated as ES modules when "type": "module" is set in package.json or using .mjs extension.
- **Modern** standard supported by browsers and Node.js (from v12+).
- **Supports static analysis** (imports resolved at compile time).
- **Example:** export function add(a, b) { return a + b; }
- **import** { add } from './service.js';

2. Using CommonJS — require / module.exports

- **Syntax:** Uses require() and module.exports.
- **Default** module system in Node.js before ES modules support.
- **Modules loaded dynamically at runtime.**
- **Widely** used in existing Node.js projects and many npm packages.
- **Example:** function add(a, b) { return a + b; }
- **module.exports = { add };**
- **const { add } = require('./service.js');**

Note: - ES Modules are more useful and preferred in modern development because they align with JavaScript standards and browser compatibility.

Promises in JavaScript / Node.js

❖ What is a Promise?

- A **Promise** is a way to handle things that happen **later** (asynchronous operations) like getting data from a server, reading a file, or waiting for a timer.

It says: I promise to give you result **later**. Either I will **fulfill** promise (give result), or I will **reject** it (if there's an error).

→ **Promise JavaScript** में एक वादा होता है, जैसे: "मैं तुम्हें भविष्य में रिजल्ट दूँगा - या तो सही (fulfill) या गलत (reject)

- रियल-लाइफ उदाहरण: मान लो आपने फोन का ऑर्डर दिया:
- Promise बनता है → दुकानदार कहता है: "3 दिन में डिलीवर कर दूँगा!"

❖ दो ही हालात हो सकते हैं:

- 1) **Fulfilled** (पूरा हुआ) → 3 दिन में फोन मिल गया!
- 2) **Rejected** (रिजेक्ट) → दुकानदार ने कहा "स्टॉक खत्म हो गया!"

Example:

दुकानदार का वादा (Promise) → "कल तक फोन दे दूँगा!"

.then() → अगर फोन मिला, तो खुशी!

.catch() → अगर नहीं मिला, तो शिकायत!

Promise = भरोसा + इंतज़ार का तरीका!

Syntax:

```
let newdata = new Promise((resolve, reject) => {
  // Async task here
  resolve("Success"); // or reject("Error");
});
newdata // variable name
  .then(result => console.log(result)) // On success
  .catch(error => console.log(error)) // On failure
  .finally(() => console.log("Done")); // Always runs
```

1. **new Promise()** → Creates a promise object.
2. **new Promise()** takes a callback with two arguments:
 - **resolve** → Marks promise as fulfilled (success).
 - **reject** → Marks promise as rejected (failure).
3. **.then()** → Runs when resolved (success) and handles the resolved value.
4. **.catch()** → Runs when rejected (error) and handles error.
5. **.finally()** → Runs always, after then or catch, regardless of success or failure.

server.js X

Example-1

server.js > ...

```
1  let data = new Promise((resolve, reject) => {
2    let haveSkills = true; // Change true to false
3    if (haveSkills) {
4      resolve("You got the job! ");
5    } else {
6      reject("No job – improve your skills first!");
7    }
8  });
9  data .then((result) => console.log(result)) // On success
10 .catch((error) => console.log(error)) // On failure
11 .finally(() => console.log("Job application process finished"));
12 //Note: .then((result) => console.log(result))
13 // represents the value passed to resolve() – in this case, "You got the job!"
14 // when the promise is fulfilled.
15 // If the promise is rejected, result is not used – instead, the .catch(error)
16 // block handles the value from reject()
```

new keyword with Promise in JavaScript to create a new Promise object. **new keyword** with Promise in JavaScript to create a new Promise object.



Example-2

```
demo.js
Lec-3 > Project folder > demo.js > orderFood
1 // Function to order food without Promises
2 function orderFood(success, error) {
3     // const shopopen = true;
4     const shopopen = false;
5
6     if (shopopen) {
7         success("Pizza will deliver on time");
8     } else {
9         error("Shop is closed!");
10    }
11}
12 // Call the function
13 orderFood(
14     (msg) => {
15         console.log(msg); // If success
16     },
17     (err) => {
18         console.error(err); // If error
19    }
20);
21
```

```
demo1.js
Lec-3 > Project folder > demo1.js > orderFood
1 // Function to order food with Promise
2 function orderFood() {
3     const shopopen = false;
4     return new Promise((resolve, reject) => {
5         if (shopopen) {
6             resolve("Pizza will deliver on time");
7         } else {
8             reject("Shop is closed!");
9         }
10    }
11)
12 orderFood()
13 .then((msg) => {
14     console.log(msg); // If success
15 })
16 .catch((err) => {
17     console.error(err); // If error
18 });
19
```

Call back function

We use new because Promise is a **JavaScript class (constructor function)** — and to create an object from a class, we must use the new keyword.

Example-2

```
demo.js
Lec-3 > Project folder > demo.js > ...
1 function getData(pra) {
2     setTimeout(() => {
3         pra("Data received after 2 sec");
4     }, 2000);
5 }
6
7 getData(function(result) {
8     console.log(result);
9 });
10
```

Note: resolve and reject are **not reserved keywords** in JavaScript we can write any things

```
demo1.js
Lec-3 > Project folder > demo1.js > then() callback
1 function getData() {
2     return new Promise((resolve, reject) => {
3         setTimeout(() => {
4             resolve("Data received after 2 sec");
5             // reject("Something went wrong");
6         }, 2000);
7     });
8 }
9 getData()
10 .then(result => {
11     console.log(result);
12 })
13 .catch(error => {
14     console.log(error);
15 });
16
```

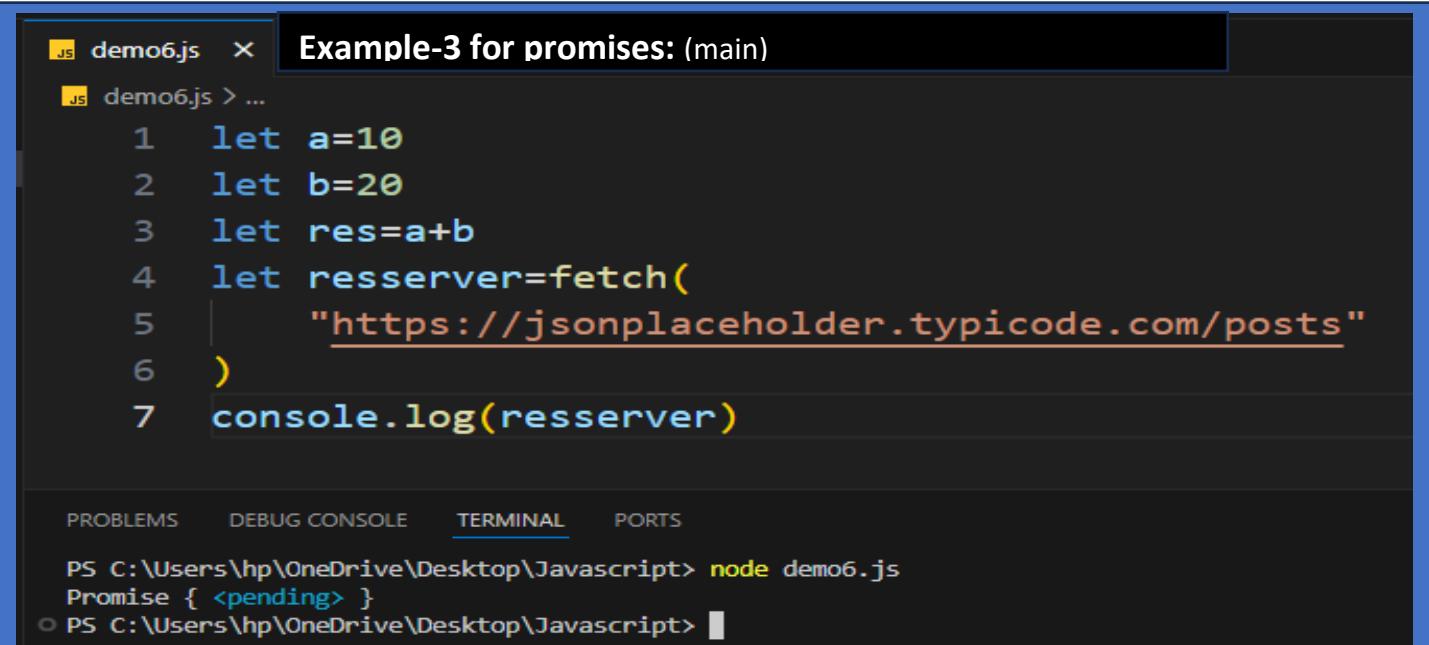
`new Promise((yes, no) => {
 yes("Task done");
 no("Something went wrong");
});`

```
service.js
service.js > ...
1 function getdata(pra) {
2     success = pra
3     const a = new Promise((accept, reject) => {
4         setTimeout(() => {
5             if (success) {accept("You are accepted ");}
6             else {reject("You are rejected ");}
7         }, 2000);
8     });
9     return a;
10}
11 getdata(false) // change arguments
12 .then((p) => {console.log("Success:", p)})
13 .catch((e) => {console.log("Error:", e)})
14 .finally(() => {console.log("I will run in any condition");});
15
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

Error: You are rejected
I will run in any condition
[nodemon] clean exit - waiting for changes before restart





```
demo6.js  Example-3 for promises: (main)
demo6.js > ...
1  let a=10
2  let b=20
3  let res=a+b
4  let resserver=fetch(
5      "https://jsonplaceholder.typicode.com/posts"
6  )
7  console.log(resserver)

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\hp\OneDrive\Desktop\Javascript> node demo6.js
Promise { <pending> }
PS C:\Users\hp\OneDrive\Desktop\Javascript>
```

Example 1: Fetch users list

```
let a = 10;
let b = 20;
let users = fetch("https://jsonplaceholder.typicode.com/users");
console.log(users);
```

Example 2: Fetch random dog image

```
let x = 5;
let y = 15;
let dogs = fetch("https://dog.ceo/api/breeds/image/random");
console.log(dogs);
```

Example 3: Fetch random joke

```
let p = 50;
let q = 100;
let joke = fetch("https://official-joke-api.appspot.com/random_joke");
console.log(joke);
```

Example 4: Fetch random meal recipe

```
let f1 = 5;
let f2 = 10;
let meal = fetch("https://www.themealdb.com/api/json/v1/1/random.php");
console.log(meal);
```

Example 5: Fetch list of categories of meals

```
let f3 = 15;
let f4 = 20;
let mealCategories = fetch("https://www.themealdb.com/api/json/v1/1/categories.php");
console.log(mealCategories);
```

Synchronous (Sync) and Asynchronous (Async) & await in Node.js

❖ Synchronous (Sync):

- Code runs one step at a time, and each task must finish before the next starts.
- Tasks are executed sequentially, one after another.
- If a task takes time, it blocks the rest of the code from executing.
- Note:** Synchronous (Sync) is not a variable, keyword, function, or method.
- It's a concept** in programming that describes how code executes in a sequential, blocking manner.
- Note:** Blocking manner → When a task stops the execution of the next tasks until it is completed.

Example: real-time example for Synchronous (Sync) Making tea step by step

- Boil water.** → Must finish before next step
- Add tea leaves.** → Waits until water is boiled
- Pour tea into cup.** → Waits until tea is ready
- Drink tea.** → Waits until tea is poured

Explanation: Each step waits for the previous one to finish.

Nothing else can happen until the current step is completed.

This is exactly how synchronous/blocking code works in programming.

Example: Coding Example in js

```
Example 1 console.log("Step 1");
            console.log("Step 2");
            console.log("Step 3");
```

Example 2

```
let sum = 5 + 3;
            console.log("Sum:", sum);
            console.log("Next task");
```

Example 3

```
for(let i=1; i<=3; i++){
            console.log("Loop iteration:", i);
            }
```

Note: by default, normal JavaScript code runs synchronously

❖ Asynchronous (Async):

- Code** can run multiple tasks at the same time without waiting for previous ones to finish.
- Time-consuming** tasks (like API calls, file reading, or timers) run in the background.
- Async** code does not block the rest of the program.
- Uses callbacks**, Promises, or `async/await` to handle results.
- Note:** Asynchronous (Async) is not a variable, keyword, function, or method.
- It's a concept** in programming that describes how code executes in a non-blocking manner.
- Note:** Non-blocking manner → When a task runs in the background while the rest of the code continues executing.

Example: Real-time example for Asynchronous (Async) Making tea while doing other tasks:

- Boil water.** → Starts, but you don't wait
- Wash** cups while water is boiling → Runs at the same time
- Add tea** leaves when water is ready → Runs after boiling, but other tasks already done
- Pour tea** into cup → Runs after tea is ready
- Drink tea** → Final step

Explanation: Tasks like washing cups can run while water is boiling.

Nothing is blocked; multiple things can happen at the same time.

This is exactly how asynchronous/non-blocking code works in programming.

Example: Coding Example in JavaScript

Example 1 - Using setTimeout without promises

```
console.log("Step 1: Start boiling water");
            setTimeout(() => {
                console.log("Step 2: Water boiled");
            }, 2000); // Runs after 2 seconds
            console.log("Step 3: Wash cups");
```

Output: Step 1: Start boiling water

Step 3: Wash cups

Step 2: Water boiled

Example-2: Using Promise

```
let data = new Promise((resolve) => {
            setTimeout(() => resolve("Step 2: Promise resolved!"),
            1500);
        });
        data.then(result => console.log(result));
        console.log("Step 3: Do other tasks");
```

Output Step 3: Do other tasks

Step 2: Promise resolved!



await

await

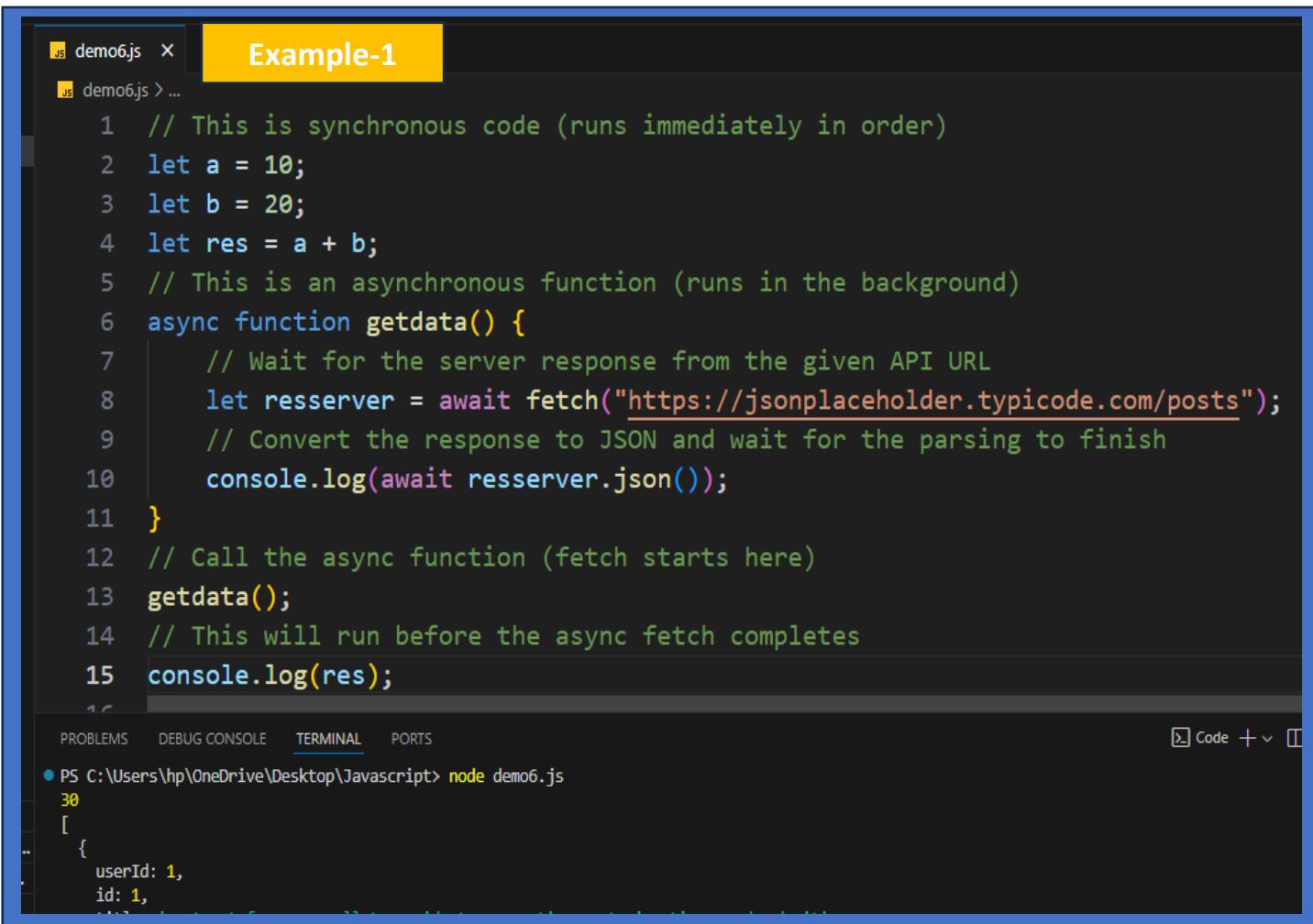
- **await** is a **JavaScript keyword** with a specific function: it pauses an **async** function until a **Promise** is resolved.
- Used **only inside an **async** function**.
- Makes **asynchronous code look and behave like synchronous code**.
- **Note:** A **concept** would describe an idea like “asynchronous programming” or “blocking vs non-blocking,” not a specific keyword in the language.

Example: Waiting for food delivery

1. You **order food** (this is like creating a **Promise**).
2. You **wait** for the food to arrive before eating (this is like **await**).
3. Once the food arrives, you **eat it** and then do the next task.
Order food → Wait for delivery → Eat food → Do other tasks

Explanation:

- **await** is like **waiting for something important to finish** before moving on.
- Even though other things could be done in parallel, you **pause at that line** until the awaited task completes.



The screenshot shows a code editor with a tab labeled 'demo6.js'. The code is as follows:

```
demo6.js x
Example-1
demo6.js > ...
1 // This is synchronous code (runs immediately in order)
2 let a = 10;
3 let b = 20;
4 let res = a + b;
5 // This is an asynchronous function (runs in the background)
6 async function getdata() {
7     // Wait for the server response from the given API URL
8     let resserver = await fetch("https://jsonplaceholder.typicode.com/posts");
9     // Convert the response to JSON and wait for the parsing to finish
10    console.log(await resserver.json());
11 }
12 // Call the async function (fetch starts here)
13 getdata();
14 // This will run before the async fetch completes
15 console.log(res);
16
```

Below the code editor, the terminal window shows the output of running the script:

```
PS C:\Users\hp\Desktop\Javascript> node demo6.js
30
[...]
{
  userId: 1,
  id: 1,
```

Note:

Key points:

- Without **await**, the code continues immediately (non-blocking).
- With **await**, the function **waits** for the **Promise** to finish before moving to the next line.

server.js X Example-2

```
server.js > ...
1  async function getData() {
2      let promise = new Promise((resolve) => {
3          setTimeout(() => resolve("Data loaded!"), 2000);
4      });
5      let result = await promise; // Wait here until promise resolves
6      console.log(result);
7      console.log("Next task");
8  }
9
10 getData();
11 console.log("Hello Everyone !")
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

- PS C:\Users\hp\OneDrive\Desktop\Javascript> **node server.js**

```
Hello Everyone !
Data loaded!
```
- Next task

server.js X Example-3

```
server.js > ...
1  function orderFood() {
2      return new Promise((resolve) => {
3          setTimeout(() => resolve("Food delivered!"), 3000);
4      });// 3 seconds delay
5  }
6 // Async function to use await
7 async function eatFood() {
8     console.log("Order placed, waiting for food...");
9     let food = await orderFood(); // Wait until food is delivered
10    console.log(food); // Food delivered!
11    console.log("Now eating the food");
12    console.log("After eating, do other tasks");
13 }
14 eatFood();
15 console.log("Hello ")
16
```

PROBLEMS DEBUG CONSOLE TERMINAL

Explanation:

- await orderFood() pauses the function until the food (Promise) is delivered.
- After that, the next steps (eating and other tasks) continue.
- This makes asynchronous code behave like **synchronous code** for readability.

```
PS C:\Users\hp\OneDrive\Desktop\Javascript> node server.js
Order placed, waiting for food...
Hello
Food delivered!
Now eating the food
After eating, do other tasks
```



Example-4: Washing clothes

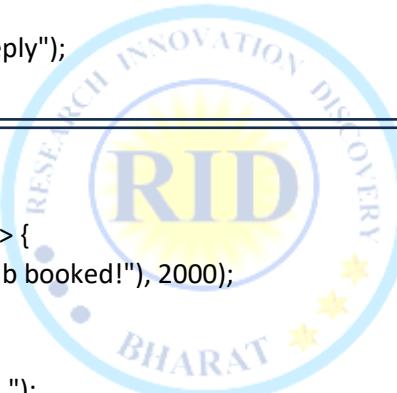
```
function washClothes() {  
  return new Promise((resolve) => {  
    setTimeout(() => resolve("Clothes are clean!"), 1000);  
  });}  
async function laundryRoutine() {  
  console.log("Start washing clothes...");  
  let result = await washClothes();  
  console.log(result);  
  console.log("Now fold the clothes");  
} laundryRoutine();
```

Example-5: Sending an email

```
function sendEmail() {  
  return new Promise((resolve) => {  
    setTimeout(() => resolve("Email sent!"), 1500);  
  });}  
async function emailTask() {  
  console.log("Composing email...");  
  let result = await sendEmail();  
  console.log(result);  
  console.log("Check inbox for reply");  
} emailTask();
```

Example-6: Booking a cab

```
function bookCab() {  
  return new Promise((resolve) => {  
    setTimeout(() => resolve("Cab booked!"), 2000);  
  });}  
async function travelPlan() {  
  console.log("Looking for a cab...");  
  let result = await bookCab();  
  console.log(result);  
  console.log("Start traveling to destination");  
} travelPlan();
```



Example-7: Downloading a file

```
function downloadFile() {  
  return new Promise((resolve) => {  
    setTimeout(() => resolve("File downloaded successfully!"), 2500);  
  });}  
async function downloadTask() {  
  console.log("Downloading file...");  
  let result = await downloadFile();  
  console.log(result);  
  console.log("Open the file now.");  
} downloadTask();
```

File Handling in node JS

❖ What is a File and a Folder?

- **File:** A file is a container in which data is stored. It can be text, image, video, or code.

Example: data.txt, app.js

- **Folder (Directory):** A folder is a container that holds files and other folders.

Example: Documents, Projects

❖ What is File Handling in Node.js?

- **File Handling** means **creating, reading, updating, and deleting files** using Node.js.

- Node.js provides the built-in **fs module** (File System) for this purpose.

- You can perform both **synchronous** (blocking) and **asynchronous** (non-blocking) operations.

Function

- Create File fs.writeFile()
- Read File fs.readFile()
- Update File fs.appendFile()
- Delete File fs.unlink()
- Create Folder fs.mkdir()
- Delete Folder fs.rmdir()

1. How to Create a File

- **Syntax:** fs.writeFile("filename.txt", "File content here", callback);

- **Note:** If you want to create a file inside another folder

- **Syntax:** fs.writeFile("myFolder/filename.txt", "File content here", callback);

➤ **Parameter Mandatory / Optional Description**

- "filename.txt" Mandatory The name (and path) of the file to create or overwrite.
- "File content here" Mandatory The content to write into the file.
- **Callback** Optional A function to handle success or error after writing the file.

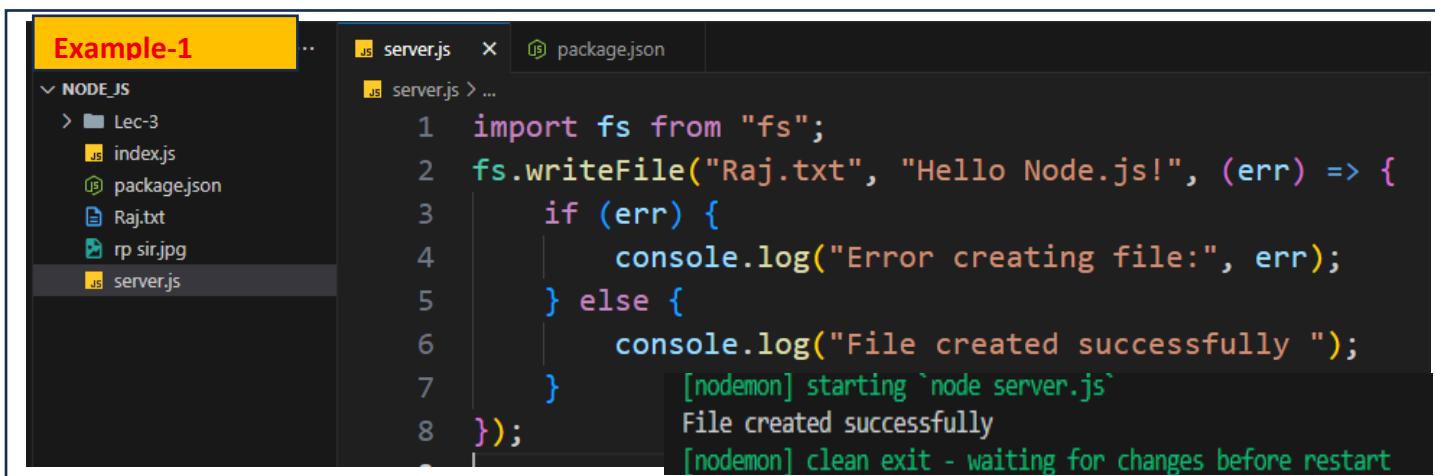
Note-1: "File content here" you can pass a variable instead of a string directly for the file content in fs.writeFile()

Note-2:

- Node.js does not restrict file types; you can create .txt, .js, .py, .html, .css, .json, etc.
- content inside the file should match the type for it to work correctly (JS code inside .js, HTML inside .html, etc.).

Step: -1: Importing the File System Module

Ex: import fs from "fs" or → const fs = require("fs");



```
Example-1
...
server.js  x  package.json
server.js > ...
1 import fs from "fs";
2 fs.writeFile("Raj.txt", "Hello Node.js!", (err) => {
3   if (err) {
4     console.log("Error creating file:", err);
5   } else {
6     console.log("File created successfully ");
7   }
8 });
[nodemon] starting `node server.js`
File created successfully
[nodemon] clean exit - waiting for changes before restart
```



Notes on fs.writeFile (Node.js)

2. Creates a new file with given content.
3. If file exists → overwrites content.
4. If file doesn't exist → creates it.
5. Works asynchronously (non-blocking).
6. Can also create inside folders ("myFolder/file.txt").

6. Can throw errors if:

- Folder doesn't exist
- Permission denied
- Invalid filename
- Disk full or read-only file system

Example-2

```

1 import fs from "fs";
2 // const fs = require("fs");
3 const data = "welcome to Node js class";
4 fs.writeFile("skills.txt", data, (err) => {
5     if (err) {
6         console.log("Error creating file:", err);
7     } else {
8         console.log("File created successfully");
9     }
10 });

```

Example-3

```

1 // Example-1: Create a simple file (raj1.txt) in newfolder that is not exit
2 import fs from 'fs'
3 fs.writeFile("newfolder/raj1.txt", "Hello! This is raj1 file.", (err) => {
4     if (err) {
5         console.log("Error creating file:", err);
6     } else {
7         console.log("raj1.txt created successfully!");
8     }
9 });

```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\hp\OneDrive\Desktop\Node_js> node "c:\Users\hp\OneDrive\Desktop\Node_js\Lect-4\demo1.js" Error creating file: [Error: ENOENT: no such file or directory, open 'C:\Users\hp\OneDrive\Desktop\Node_js\newfolder\raj1.txt'] { errno: -4058, code: 'ENOENT', syscall: 'open',

Note: if the folder does not exit it will throw the error

Example-4

```

1 // Example-1: Create a simple file (raj1.txt) in newfolder
2 import fs from 'fs'
3 fs.writeFile("newfolder/raj1.txt", "Hello! This is raj1 file.", (err) => {
4     if (err) {
5         console.log("Error creating file:", err);
6     } else {
7         console.log("raj1.txt created successfully!");
8     }
9 });

```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\hp\OneDrive\Desktop\Node_js\Lect-4> node demo1.js raj1.txt created successfully!



2. How to Read a File

- **Syntax:** `fs.readFile("filename.txt", "utf8", callback);`
- **Example:**

```
import fs from "fs";
fs.readFile("raj.txt", "utf8", (err, data) => {
  if (err) {console.log("Error reading file:", err)}
  else {console.log("File content:", data);}
});
```

❖ Parameters:

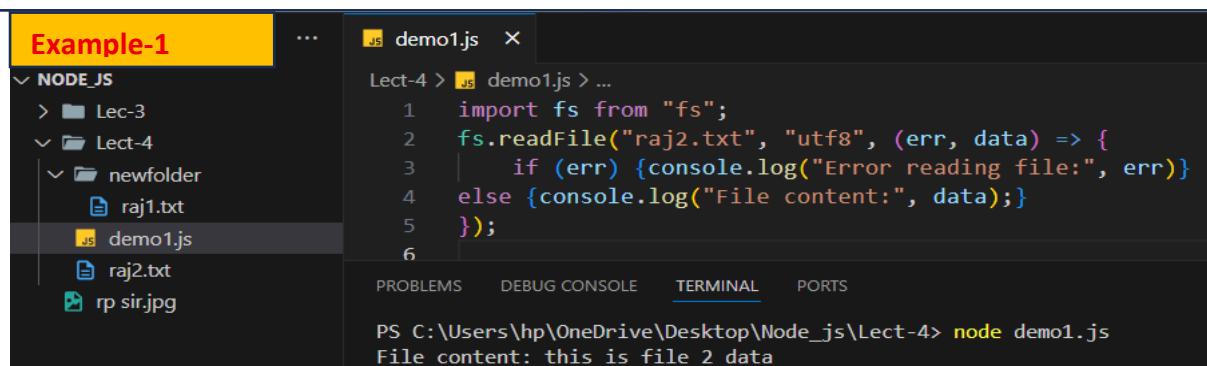
Parameter	Mandatory / Optional	Description
1."filename.txt"	Mandatory	The name (and path) of the file to read
2."utf8"	Optional	Encoding format (to read as text)
3.callback	Mandatory	Function to handle success or error after reading

- **Notes:**

1. Can read any file type: .txt, .js, .py, .html, .json, etc.
2. Use "utf8" for text files; omit for binary files.
3. Errors occur if file doesn't exist, path is wrong, or permission is denied.
4. Works asynchronously (non-blocking).

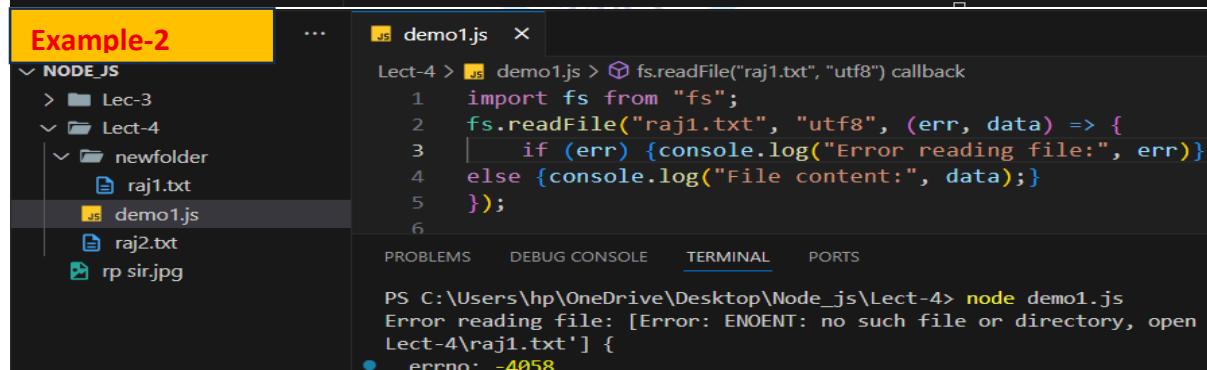
- **Step-1:** Import the File System module: **Note:** `import fs from "fs"; // or const fs = require("fs");`

Example-1



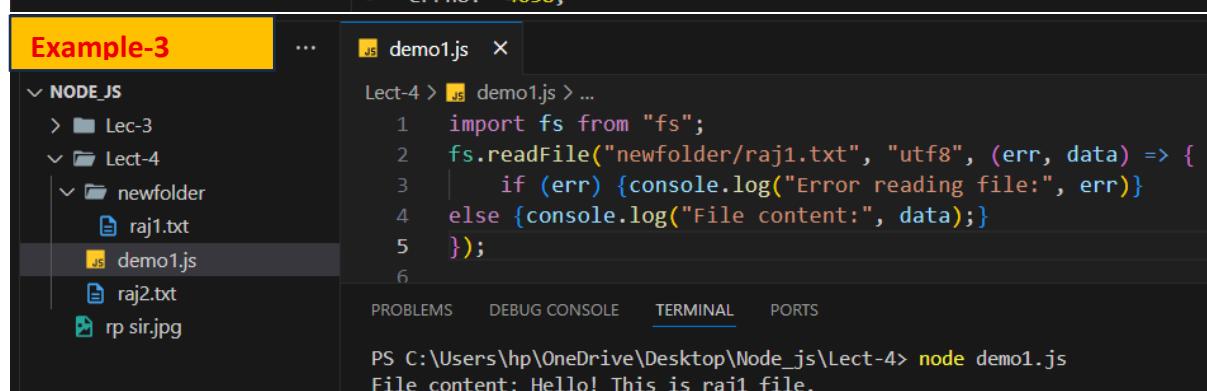
```
import fs from "fs";
fs.readFile("raj2.txt", "utf8", (err, data) => {
  if (err) {console.log("Error reading file:", err)}
  else {console.log("File content:", data);}
});
```

Example-2



```
import fs from "fs";
fs.readFile("raj1.txt", "utf8", (err, data) => {
  if (err) {console.log("Error reading file:", err)}
  else {console.log("File content:", data);}
});
```

Example-3



```
import fs from "fs";
fs.readFile("newfolder/raj1.txt", "utf8", (err, data) => {
  if (err) {console.log("Error reading file:", err)}
  else {console.log("File content:", data);}
});
```



3. How to Update a File

File("filename.txt", "Additional content", callback);

- **Example:**

```
import fs from "fs";
fs.appendFile("raj.txt", "\nThis is new content.", (err) => {
  if (err) console.log("Error updating file:", err);
  else console.log("File updated successfully");
});
```

- **Parameters:**

Parameter	Mandatory / Optional	Description
1. "filename.txt"	Mandatory	Name (and path) of the file to update
2. "Additional content"	Mandatory	Content to append to the file
3. callback	Mandatory	Function to handle success or error

Notes:

1. Can update any file type: .txt, .js, .py, .html, .json, etc.
2. Appends content at the **end of the file**.
3. Errors occur if file doesn't exist, path is wrong, or permission is denied.
4. Works asynchronously (non-blocking).

Note: You want to add content at the beginning or anywhere in a file instead of just appending at the end. Here's a short note on that: **Adding Content at Beginning or Anywhere in a File**

- fs.appendFile() always adds content at the end.
- To add content at the beginning or anywhere, you need to:
 1. Read the file first using fs.readFile().
 2. Modify the content (add text at the start, middle, or specific position).
 3. Write the updated content back using fs.writeFile().

- **Example – Add at the beginning:**

```
import fs from "fs";
fs.readFile("raj.txt", "utf8", (err, data) => {
  if (err) return console.log("Error reading file:", err);
  const updatedContent = "New content at beginning\n" + data;
  fs.writeFile("raj.txt", updatedContent, (err) => {
    if (err) console.log("Error updating file:", err);
    else console.log("Content added at the beginning successfully!");
  });
});
```

- **Example – Add anywhere (after line 2):**

```
fs.readFile("raj.txt", "utf8", (err, data) => {
  if (err) return console.log("Error reading file:", err);
  let lines = data.split("\n");
  lines.splice(2, 0, "New content in the middle"); // insert at 3rd line
  const updatedContent = lines.join("\n");
  fs.writeFile("raj.txt", updatedContent, (err) => {
    if (err) console.log("Error updating file:", err);
    else console.log("Content added in the middle successfully!");
  });
});
```

Notes:

1. Can update any file type: .txt, .js, .py, .html, .json, etc.
2. Use fs.readFile() → modify → fs.writeFile() to add content anywhere.
3. Works asynchronously (non-blocking).
4. Errors occur if file doesn't exist, path is wrong, or permission denied.



4. How to Create a Folder

- **Syntax:** fs.mkdir("folderName", callback);
- **Example:**

```
import fs from "fs";
fs.mkdir("myFolder", (err) => {
  if (err) console.log("Error creating folder:", err);
  else console.log("Folder created successfully");
});
```

Parameters:

Parameter	Mandatory / Optional	Description
1. "folderName"	Mandatory	Name (and path) of the folder to create
2. callback	Mandatory	Function to handle success or error

Notes:

1. Can create single or nested folders using { recursive: true }.
2. Errors occur if folder already exists (without recursive) or permission is denied.
3. Works asynchronously (non-blocking).

5. How to Delete a File and a Folder

1. Delete a File Syntax: fs.unlink("filename.txt", callback);

Example:

```
import fs from "fs";
fs.unlink("raj.txt", (err) => {
  if (err) console.log("Error deleting file:", err);
  else console.log("File deleted successfully");});
```

Parameters:

Parameter	Mandatory / Optional	Description
1. "filename.txt"	Mandatory	Name (and path) of the file to delete
2. callback	Mandatory	Function to handle success or error

Notes:

1. Deletes a file permanently.
2. Errors occur if file doesn't exist or permission is denied.
3. Works asynchronously (non-blocking).

2. Delete a Folder

- **Syntax:** fs.rmdir("folderName", callback);

- **Example:**

```
import fs from "fs";
fs.rmdir("myFolder", (err) => {
  if (err) console.log("Error deleting folder:", err);
  else console.log("Folder deleted successfully");});
```

Parameters:

Parameter	Mandatory / Optional	Description
1. "folderName"	Mandatory	Name (and path) of the folder to delete
2. callback	Mandatory	Function to handle success or error

Notes:

1. Deletes an **empty folder** only.
2. Errors occur if folder is not empty, doesn't exist, or permission is denied.
3. Works asynchronously (non-blocking).

Path Module in Node.js

Path Module in Node.js

- The path module is a built-in module in Node.js.
- It is used to work with file and directory paths.
- It helps in joining, resolving, and normalizing paths so your code works across all operating systems.
- Path module makes it easy to handle file & folder paths.

path module methods in Node.js

- **basename()** → file name : Use when you need only the **file name** from a complete path.
- **dirname()** → folder name : Use when you need only the **directory/folder path**.
- **extname()** → file extension : Use when you need the **file extension** (like .js, .html, .css, .pdf.jpg).
- **join()** → combine paths : use when Safely combine multiple folders/files into one full path
- **resolve()** → absolute path : Converts a relative path into an **absolute path**. Very useful in projects.

Example-1

```
Lect-5 > demo2.js > ...
1 import path from "path";
2 // Join paths
3 const fullPath = path.join("/path", "home.html", "raj.css");
4 console.log("Full Path =", fullPath);
5 // How to find the actual or absolute path
6 const currentPath = path.resolve();
7 console.log("Current Path =", currentPath);
8 // Get file extension
9 const fileExt = path.extname("raj.css");
10 console.log("File Extension =", fileExt);
11 // Get base name (file name)
12 const fileName = path.basename("home.html");
13 console.log("File Name =", fileName);
```

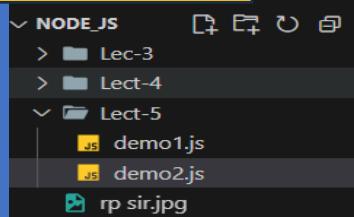
PROBLEMS DEBUG CONSOLE TERMINAL PORTS

Full Path = \path\home.html\raj.css
 Current Path = C:\Users\hp\OneDrive\Desktop\Node_js
 File Extension = .css
 File Name = home.html

Getting the actual folder of the file (ES Modules)

- In **ES Modules (import)**, `__filename` and `__dirname` are not available.
- Use `fileURLToPath(import.meta.url)` to get the **full file path**.
- Then use `path.dirname()` to get the **folder containing the file**.

Example-2



demo2.js

```
1 //How to get the actual folder of the file
2 import path from "path";
3 import { fileURLToPath } from "url";
4 const filename = fileURLToPath(import.meta.url)
5 const foldername = path.dirname(filename);
6 console.log("File Path =", filename);
7 console.log("Folder Path =", foldername);
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

- PS C:\Users\hp\OneDrive\Desktop\Node_js\Lect-5> `node demo2.js`
 File Path = C:\Users\hp\OneDrive\Desktop\Node_js\Lect-5\demo2.js
- Folder Path = C:\Users\hp\OneDrive\Desktop\Node_js\Lect-5



Example-3

```
Lect-5 > [js] demo3.js > ...
1 import path from "path";
2 const v = "myimage.jpg";
3 // Get the file extension
4 const getFileType = path.extname(v);
5 console.log("File Extension =", getFileType);
6 // Check if the file is a .jpg
7 if (getFileType === '.jpg') {
8     console.log("Your photo uploaded successfully");
9 } else {
10     console.log("Please upload a file with .jpg extension");
11 }
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

Code - Lect-5 +

```
PS C:\Users\hp\OneDrive\Desktop\Node_js\Lect-5> node demo3.js
File Extension = .jpg
Your photo uploaded successfully
```

Example-4

```
Lect-5 > [js] demo3.js > ...
1 import path from "path";
2 // 1. extname() → Check file type
3 const uploadedFile = "marksheet.pdf";
4 const fileExt = path.extname(uploadedFile);
5 console.log("File Extension =", fileExt);
6 if (fileExt === ".pdf") {
7     console.log("File uploaded successfully\n");
8 } else {console.log("Please upload a .jpg file\n");}
9 // 2. join() → Combine folder and file paths
10 const folder = "/user";
11 const subFolder = "images";
12 const file = "profile.jpg";
13 const fullPath = path.join(folder, subFolder, file);
14 console.log("Joined Full Path =", fullPath);
15 // 3. resolve() → Get absolute path
16 const absPath = path.resolve(file);
17 console.log("Absolute Path =", absPath);
```



How to Build Server in Node JS

❖ Server:

- server is a **dedicated computer or a computer program** that **accepts requests from clients** (like browsers) and **sends back responses** over a network
- It is a program or computer that **listens for requests** from clients (like web browsers) and **sends back responses**.

❖ Real Life Example of server:

- A **restaurant waiter** acts like a server: the customer (client) places an order (request), and the waiter (server) brings back the food (response).
- Similarly, a **web server** receives requests from your browser and sends back web pages or data.

❖ Examples of servers:

- Web server (sends HTML, CSS, JS files) API server (sends JSON data)

❖ Note:

- It can be **hardware** (a powerful computer) or **software** (a program running on a computer). Its main job is to **listen for requests and respond** with data, files, or services.

❖ Step to create the Server in node JS:

- Node.js makes it easy to build a simple server using its built-in http module.

Step 1: create file with server.js(recommended)

Step 2: run npm init -y (for creating the json file “project setting”)

- npm init -y automatically creates a **package.json** with default values.
- This file stores **project information and dependencies**.

Step3: Import the http module

- Ex:** import **http** from "http";
- ✓ **http** is built into Node.js, no need to install anything.
- ✓ It allows us to create a web server.

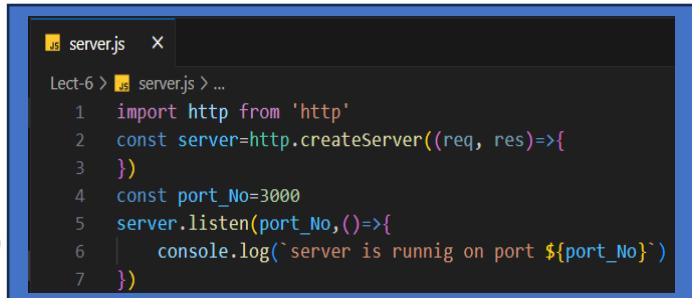
web is a system of interconnected **websites and web pages** that can be accessed over Internet using a browser.

web server: - it is a computer or software that **delivers web pages and content** to clients (browsers) over the Internet.

Step-4: Create a server

Example-1:

```
import http from 'http'  
const server=http.createServer((req, res)=>{  
})  
const port_No=3000  
server.listen(port_No, ()=>{  
    console.log(`server is runnig on port ${port_No}`)})
```



```
server.js  X  
Lect-6 > server.js > ...  
1 import http from 'http'  
2 const server=http.createServer((req, res)=>{  
3 })  
4 const port_No=3000  
5 server.listen(port_No, ()=>{  
6     console.log(`server is runnig on port ${port_No}`)  
7 })
```

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

- Creates a **new server** using **http.createServer()**.
- Takes a **callback function** with two parameters:
 - ✓ **req** → represents the **client request** (e.g., browser asking for a page).
 - ✓ **res** → represents the **response** that the server will send back.

➤ const port_No = 3000

- Defines the port number where the server will listen.
- ✓ 3000 is commonly used for local development.

➤ server.listen(port_No, () => {console.log(`server is runnig on port \${port_No}`)})

- Starts the server and makes it listen on the specified port.
- The callback function runs once the server is successfully running.
- console.log() prints a message confirming the server is running.



```

server.js  Example-2
Lect-6 > server.js > ...
1 // Example for creating the server in Node js
2 import http from 'http';
3 const server = http.createServer((req, res) => { // Create a server
4   // Send response to the client
5   res.writeHead(200, { "Content-Type": "text/plain" });
6   res.end("Your request was accepted successfully");
7 }); // Define the port number
8 const port_No = 3000; // Start the server and listen on the specified port
9 server.listen(port_No, () => {
10   console.log(`Server is running on http://localhost:\${port\_No}`);
11 });

```

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

it is used to set the HTTP response headers before sending the actual content. Let me break it down:

1. **res.writeHead(200, ...)**

200 → HTTP status code 200 means the request was successful.

Other examples: 404 = Not Found, 500 = Server Error.

2. **{ "Content-Type": "text/plain" }**

This is a header object that tells the browser what kind of content the server is sending.

"text/plain" → the response is plain text.

Other examples: "text/html" → HTML content "application/json" → JSON data

Common HTTP status codes you can use in Node.js:

1. Successful Responses:

200 → OK (request successful),

201 → Created (resource successfully created)

2. Client Errors:

400 → Bad Request (invalid request from client),

401 → Unauthorized (authentication needed)

403 → Forbidden (access denied) 404 → Not Found (resource not found)

3. Server Errors

500 → Internal Server Error (something went wrong on server)

502 → Bad Gateway (server received invalid response)

503 → Service Unavailable (server temporarily unavailable)

Note: In short: 2xx → Success, 4xx → Client errors, 5xx → Server errors

❖ What **res.end()** does

- **res.end()** sends the response to the client and closes the connection.
- Whatever you pass inside **res.end()** is the **actual content** that the browser (or client) will receive.

2. Relation to **res.writeHead()**

- **res.writeHead(200, { "Content-Type": "text/plain" })** → sets the **status code** and **headers** (metadata about the response).
- **res.end("...")** → sends the **body of the response** (the actual message or data).

res.end() can send:

- **Text** → "Hello World"
- **HTML** → "<h1>Hi</h1>"
- **JSON** → `JSON.stringify({...})`
- **Binary data** → file buffers like images, PDFs, etc.



res.end() method in Node.js can send different types of data as the response.

1. Plain Text Example:

```
res.writeHead(200, { "Content-Type": "text/plain" });
res.end("Hello, this is plain text!");
```

- Content type: text/plain
- Browser displays it as simple text.

```
import http from 'http';
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end(`Hello this is node js class `);
});
const port_No = 3000;
server.listen(port_No, () => {
  console.log(`Server is running on http://localhost:\${port\_No}`);
});
```

2. HTML Example:

```
res.writeHead(200, { "Content-Type": "text/html" });
res.end(`<h1>Hello, this is HTML content!</h1>`);
```

- Content type: text/html
- Browser renders it as a formatted webpage.

```
import http from 'http';
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end(`<h1>Hello this is HTML Content </h1>`);
});
const port_No = 3000;
server.listen(port_No, () => {
  console.log(`Server is running on http://localhost:\${port\_No}`);
});
```

3. JSON Example:

```
res.writeHead(200, { "Content-Type": "application/json" });
res.end(JSON.stringify({ name: "Sangam", course: "Node.js" }));
```

- Content type: application/json
- Browser or client receives it as JSON data.

```
import http from 'http';
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end(JSON.stringify({ name: "Sangam", course: "Node.js" }));
  // res.end(`<h1>Name: Sangam</h1><p>Course: Node.js</p>`);
});
const port_No = 3000;
server.listen(port_No, () => {
  console.log(`Server is running on http://localhost:\${port\_No}`);
});
```

Optional

4. Binary / Files Example

```
res.writeHead(200, { "Content-Type": "image/png" });
// Here you would send binary data like image or file buffer
res.end(fileBuffer);
```

- Content type depends on the file (image/png, application/pdf, etc.)
- Used to send **images, PDFs, videos, or any file**.

```
//common types, to use in res.writeHead():
// For Images
res.writeHead(200, { "Content-Type": "image/jpeg" }); // .jpg
res.writeHead(200, { "Content-Type": "image/png" }); // .png
res.writeHead(200, { "Content-Type": "image/gif" }); // .gif
//For PDF
res.writeHead(200, { "Content-Type": "application/pdf" });
// HTML
res.writeHead(200, { "Content-Type": "text/html" });
//For JSON
res.writeHead(200, { "Content-Type": "application/json" });
//For Video
res.writeHead(200, { "Content-Type": "video/mp4" }); // .mp4
res.writeHead(200, { "Content-Type": "video/webm" }); // .webm
res.writeHead(200, { "Content-Type": "video/ogg" }); // .ogg
//For Audio
res.writeHead(200, { "Content-Type": "audio/mpeg" }); // .mp3
res.writeHead(200, { "Content-Type": "audio/wav" }); // .wav
//Note: Always set the correct Content-Type before res.end() when sending the file.
```

Types for send the data

- For **images** → "image/jpeg" / "image/png"
- For **PDFs** → "application/pdf"
- For **HTML** → "text/html"
- For **JSON** → "application/json"

video content types:

- .mp4 → "video/mp4"
- .webm → "video/webm"
- .ogg → "video/ogg"



Example-1

... server.js rpsir.jpg

Lect-6 > server.js > server > http.createServer() callback > fs.readFile() callback

```

1 import http from 'http';
2 import fs from 'fs';
3 import path from 'path';
4 const server = http.createServer((req, res) => {
5     // Set file path (uncomment the file you want to serve)
6     // const filePath = path.join(process.cwd(), 'allimg/rpsir.jpg'); //for Image
7     // const filePath = path.join(process.cwd(), 'allpdf/nodejsebbok.pdf'); //for PDF
8     const filePath = path.join(process.cwd(), 'allvideo/myvideo.mp4'); // for Video
9     // Read the file
10    fs.readFile(filePath, (err, fileBuffer) => {
11        if (err) {
12            res.writeHead(404, { "Content-Type": "text/plain" });
13            res.end("File not found");
14        } else {
15            // Set correct Content-Type for the file
16            // res.writeHead(200, { "Content-Type": "image/jpeg" }); // For Image
17            // res.writeHead(200, { "Content-Type": "application/pdf" }); // For PDF
18            res.writeHead(200, { "Content-Type": "video/mp4" }); // For Video
19            // Send file to browser
20            res.end(fileBuffer);
21        }
22    });
23    const port_No = 3000;
24    server.listen(port_No, () => {
25        console.log(`Server is running on http://localhost:\${port\_No}`);
    });

```

NODE JS

- Lect-4
- Lect-5
- Lect-6
 - allimg
 - rpsir.jpg
 - allpdf
 - nodejsebbok.pdf
 - allvideo
 - myvideo.mp4
- package.json
- server.js

OUTPUT

WORKSPACE ENVIRONMENT: WO...

ENVIRONMENT MANAGER: GLOB...

OUTLINE

- server
 - http.createServer() callback
 - filePath
 - fs.readFile() callback
 - Content-Type

TIMELINE

Example-2

server.js

Lect-6 > server.js > server > http.createServer() callback

```

1 import http from 'http';
2 const server = http.createServer((req, res) => { // Create a server
3     // Set status code and content type to HTML
4     res.writeHead(200, { "Content-Type": "text/html" });
5     // Send HTML response
6     res.end(`
7         <html>
8             <head>
9                 <title>My First Node Server</title>
10            </head>
11            <body>
12                <h1>Your request was accepted successfully</h1>
13                <p>Welcome to my Node.js server!</p>
14            </body>
15        </html>
16    `);
17 });
18 // Define the port number
19 const port_No = 3000;
20 // Start the server
21 server.listen(port_No, () => {
22     console.log(`Server is running on http://localhost:\${port\_No}`);
23 });

```

How to send external html as response in node using res.end()

NODE JS

- > Lec-3
- > Lect-4
- > Lect-5
- ✓ Lect-6
 - > allimg
 - > allpdf
 - > allvideo
 - ✓ htmlfile
 - Home.html
 - package.json
 - server.js

OUTPUT

WORKSPACE ENVIRONMENT: WO...

ENVIRONMENT MANAGER: GLOB...

EXPLORER

- > NODE JS
- > Lec-3
- > Lect-4
- > Lect-5
- ✓ Lect-6
 - > allimg
 - > allpdf
 - > allvideo
 - ✓ htmlfile
 - Home.html
 - package.json
 - server.js

OUTLINE

- ✓ server
- ✓ http.createServer() callback
 - ✓ filePath
 - ✓ fs.readFile() callback
 - ✓ "Content-Type"

TIMELINE

Lect-6 > `server.js` > `server` > `http.createServer()` callback

```

1 import http from 'http';
2 import fs from 'fs';
3 import path from 'path';
4 const server = http.createServer((req, res) => {
5     // Get the path to data.html in the same folder external htmlfile
6     const filePath = path.join(process.cwd(), 'htmlfile/Home.html');
7     // Read the HTML file
8     fs.readFile(filePath, (err, data) => {
9         if (err) {res.end("Error loading file");}
10        else {res.end(data);}
11    });
12 });// Define the port number
13 const port_No = 3000;
14 server.listen(port_No, () => {
15     console.log(`Server is running on http://localhost:\${port\_No}`);
16 });

```

server.js X Home.html

Lect-6 > `server.js` > ...

```

1 import http from 'http';
2 import fs from 'fs';
3 import path from 'path';
4 const server = http.createServer((req, res) => {
5     // Get the path to data.html in the same folder
6     const filePath = path.join(process.cwd(), 'htmlfile/Home.html');
7     // Read the HTML file
8     fs.readFile(filePath, (err, data) => {
9         if (err) {
10             // If error, send 500 response
11             res.writeHead(500, { "Content-Type": "text/plain" });
12             res.end("Error loading file");
13         } else {
14             // Send HTML content
15             res.writeHead(200, { "Content-Type": "text/html" });
16             res.end(data);
17         }
18     });
19 });
20 // Define the port number
21 const port_No = 3000;
22 server.listen(port_No, () => {
23     console.log(`Server is running on http://localhost:\${port\_No}`);
24 });

```

Optional

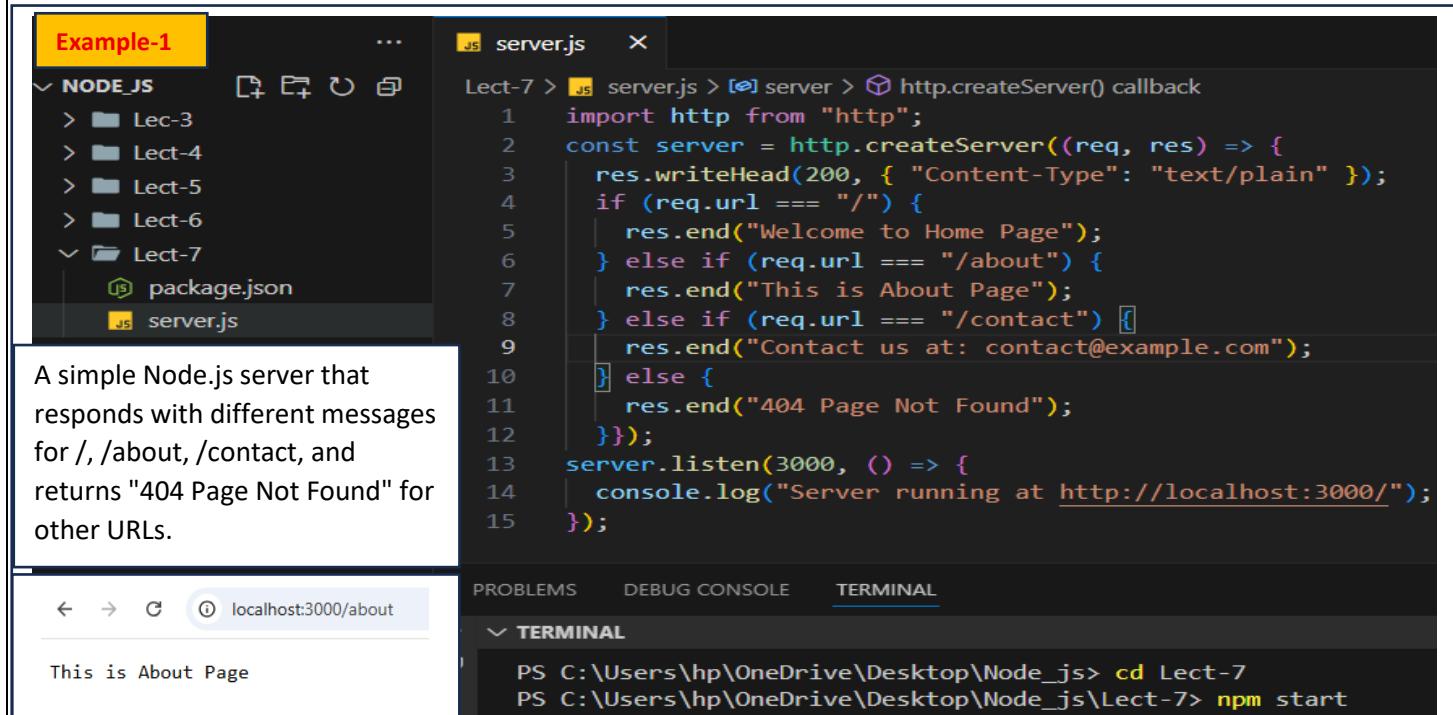
Route in Node JS

A route is a rule: (HTTP method + URL path) → handler function → response.

A Route is simply a way to tell your server: "When someone visits a particular URL, show them the correct response."

1. **HTTP Method** → Action type (like asking, sending, updating, deleting).
 2. **URL Path** → The address (like /, /about, /contact).
 3. **Handler Function** → The code that runs when that address is visited.
 4. **Response** → The answer your server sends back to the client.
- You **match** the request, **extract** inputs, **run** logic, and **respond**.
 - Use **Express** to define routes declaratively; use **http** for learning or very simple servers.

Example-1



```

server.js  x
Lect-7 > server.js > server > http.createServer() callback
1 import http from "http";
2 const server = http.createServer((req, res) => {
3   res.writeHead(200, { "Content-Type": "text/plain" });
4   if (req.url === "/") {
5     res.end("Welcome to Home Page");
6   } else if (req.url === "/about") {
7     res.end("This is About Page");
8   } else if (req.url === "/contact") {
9     res.end("Contact us at: contact@example.com");
10 } else {
11   res.end("404 Page Not Found");
12 });
13 server.listen(3000, () => {
14   console.log("Server running at http://localhost:3000/");
15 });

```

PROBLEMS DEBUG CONSOLE TERMINAL

TERMINAL

```

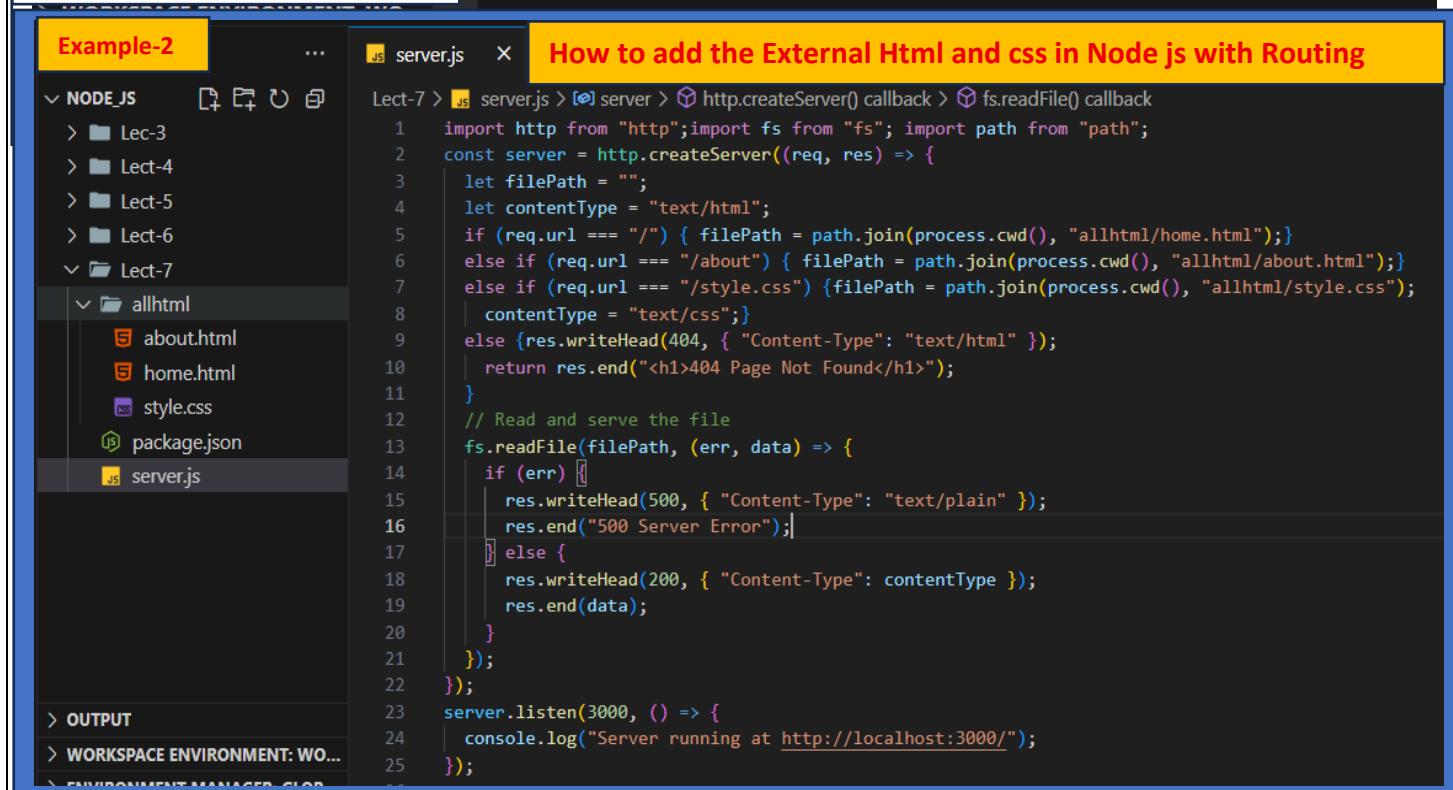
PS C:\Users\hp\OneDrive\Desktop\Node_js> cd Lect-7
PS C:\Users\hp\OneDrive\Desktop\Node_js\Lect-7> npm start

```

This is About Page

Example-2

How to add the External Html and css in Node js with Routing



```

server.js  x
Lect-7 > server.js > server > http.createServer() callback > fs.readFile() callback
1 import http from "http"; import fs from "fs"; import path from "path";
2 const server = http.createServer((req, res) => {
3   let filePath = "";
4   let contentType = "text/html";
5   if (req.url === "/") { filePath = path.join(process.cwd(), "allhtml/home.html"); }
6   else if (req.url === "/about") { filePath = path.join(process.cwd(), "allhtml/about.html"); }
7   else if (req.url === "/style.css") { filePath = path.join(process.cwd(), "allhtml/style.css"); }
8   else {contentType = "text/css";}
9   else {res.writeHead(404, { "Content-Type": "text/html" });
10  return res.end("<h1>404 Page Not Found</h1>");}
11  // Read and serve the file
12  fs.readFile(filePath, (err, data) => {
13    if (err) {
14      res.writeHead(500, { "Content-Type": "text/plain" });
15      res.end("500 Server Error");
16    } else {
17      res.writeHead(200, { "Content-Type": contentType });
18      res.end(data);
19    }
20  });
21 });
22 });
23 server.listen(3000, () => {
24   console.log("Server running at http://localhost:3000/");
25 });

```

Express JS

What is Express.js?

- Express.js is a lightweight and flexible **web application framework** for Node.js.
- It simplifies handling **routes, requests, responses, and middleware** compared to writing everything with Node's built-in http module.

Why are we using it?

servers and APIs quickly.

routing system (GET, POST, PUT, DELETE).

middleware (e.g., authentication, logging).

for REST APIs, websites, and backend services.

How to Use Express.js in Node.js

Step-1: Make a new folder

Example: mkdir myapp then cd myapp

Step-2: create the file with the .js extension

Example: server.js or index.js

Step-3: Initialize project

Example: npm init -y

Step-4: Install Express

Example: npm install express or npm i express

Step-5: Create the server

```
server.js
import express from "express";
// or: const express = require("express")
const app = express();
// Start the server only
const port=3000
app.listen(port, () => {
  console.log(`Server running
  at http://localhost:${port}`);
});
```

```
package.json
{
  "name": "lect-8",
  "version": "1.0.0",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "test": "echo \\"$Error: no test specified\\"",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^5.1.0"
  }
}
```

Step-6: Run the server
Example: nodemon server.js

Step-7: Open in browser
Go to <http://localhost:3000>

How to create server and send response on browser in express JS

- **get(path, callback)** → Handles GET requests for given path.
- **req** → Request object (information from client).
- **res** → Response object (used to send data back).
- **res.send()** → Sends a simple text/HTML response.
- **res.send()** automatically sets the correct Content-Type.
- For JSON specifically, you can also use **res.json()** (better practice for APIs).
- **send()** can handle **strings, HTML, arrays, objects, or buffers**. like files, images, video or any binary content

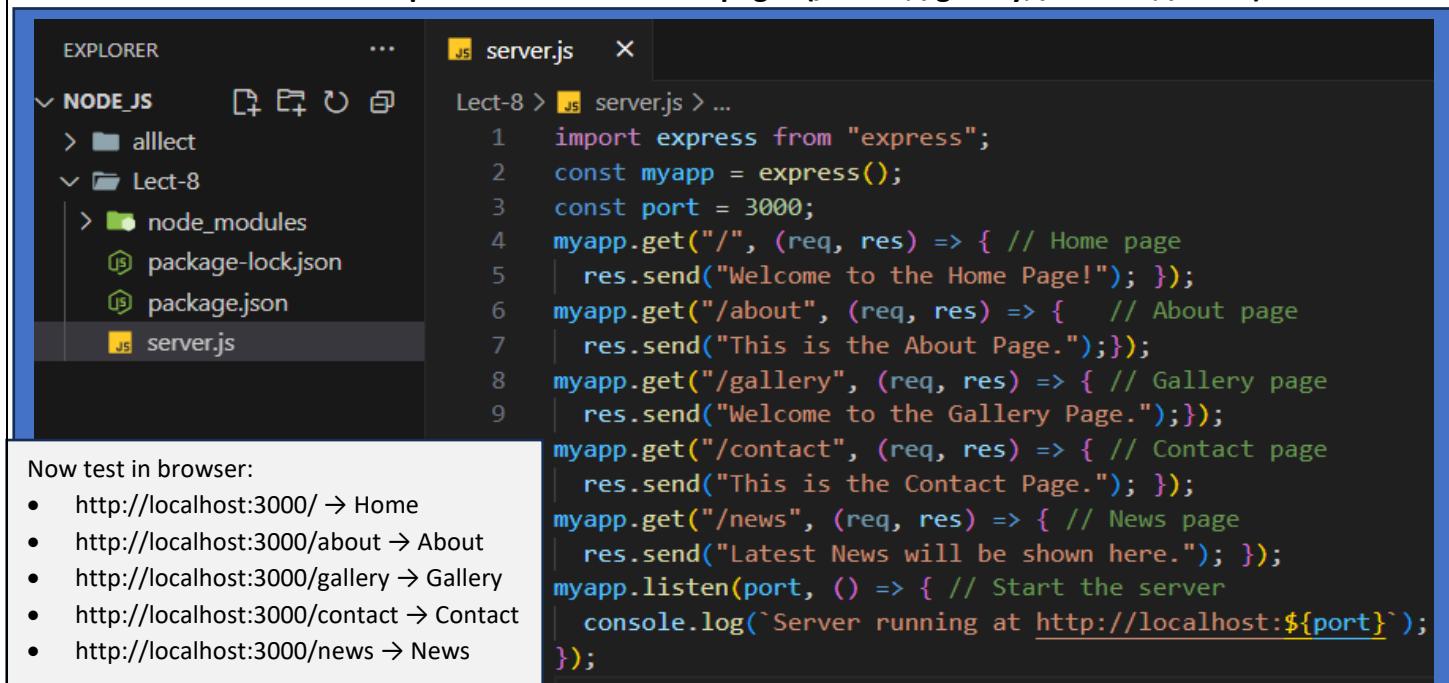
```
index.js
import express from 'express'
const app=express()
app.get("/", (req, res)=>{
  res.send("this is first server")
})
const port=3000
app.listen(port, ()=>{
  console.log(`server running on http://localhost:${port}`)
})
```

How it works:

1. **app.get("/", ...)** → listens for GET requests at / (home page).
2. **res.send("...")** → sends a plain text response back to the client.
3. Now if you open <http://localhost:3000>

How to send multiple response on browser

Need to define multiple routes for different pages (/about, /gallery, /contact, /news).



The screenshot shows a code editor with an 'EXPLORER' sidebar on the left. The 'NODE JS' section shows a project named 'Lect-8' with subfolders 'alllect' and 'server.js'. The 'server.js' file is selected and open in the main editor area. The code in 'server.js' is as follows:

```

1 import express from "express";
2 const myapp = express();
3 const port = 3000;
4 myapp.get("/", (req, res) => { // Home page
5   res.send("Welcome to the Home Page!"); });
6 myapp.get("/about", (req, res) => { // About page
7   res.send("This is the About Page."); });
8 myapp.get("/gallery", (req, res) => { // Gallery page
9   res.send("Welcome to the Gallery Page."); });
myapp.get("/contact", (req, res) => { // Contact page
  res.send("This is the Contact Page."); });
myapp.get("/news", (req, res) => { // News page
  res.send("Latest News will be shown here."); });
myapp.listen(port, () => { // Start the server
  console.log(`Server running at http://localhost:\${port}`);
});

```

Now test in browser:

- <http://localhost:3000/> → Home
- <http://localhost:3000/about> → About
- <http://localhost:3000/gallery> → Gallery
- <http://localhost:3000/contact> → Contact
- <http://localhost:3000/news> → News

How to send image, video & video response on browser

Folder Structure

```

myapp/                                <-- Project root
|
|   node_modules/                      <-- Installed npm packages (auto-created)
|   package.json                        <-- NPM config file
|
|   server.js                           <-- Main Express server file
|
|   public/                             <-- Static files (CSS, JS, images)
|       |   css/                         <-- CSS files
|           |       style.css
|       |   js/                          <-- JavaScript files
|           |       script.js
|       |   images/                      <-- Image files
|           |       rpsir.jpg
|
|   htmlfile/                          <-- External HTML files
|       |   index.html

```

How it works

1. **server.js** → Creates the Express server and serves routes.
2. **public/** → Use `express.static()` to serve CSS, JS, and images.
3. **htmlfile/** → Use `res.sendFile()` to send HTML files to the browser

How to send External multiple response on browser

This all methods we used in Node.js + Express.js for sending external HTML, CSS, images, PDFs, and videos

Step-11. import express from "express"

- Imports the Express framework. Allows you to create an Express server and routes.

Step-2. import path from "path"

- Node.js module to handle file and folder paths. Helps create absolute paths with path.join().

Step 3. import { fileURLToPath } from "url"

- Converts ES Module import.meta.url to a normal file path. Needed because __dirname is not available in ES Modules.

Step-4. const __filename = fileURLToPath(import.meta.url) const any_v_name= fileURLToPath(import.meta.url)

- Gets the absolute path of the current file. **absolute path** means **full path from the root of the file system to a specific file or folder**, showing its exact location.

Step-5. const __dirname = path.dirname(__filename) or const anyvaraiable= path.dirname(__filename)

- Gets the directory of the current file. Useful for building paths to HTML, CSS, or other files.

Step-6. express(): This is used to create an Express application (server). You usually store it in a variable.

Example: const app = express();

Step-7. app.use() Middleware to process requests.

Node.js and Express.js, a middleware is a function that runs during the request-response cycle.

- Purpose: It can process requests, modify request/response, or serve files before sending a response.
- Why we use it: To handle tasks like serving static files, parsing data, authentication, logging, etc.
- Example: app.use('/css', express.static(path.join(__dirname, 'public/css')));

Note: express.static() serves static files like CSS, JS, images, PDFs, videos.

Example: app.use('/images', express.static(path.join(__dirname, 'allimg')));

8. path.join() :- Joins multiple path segments into a single platform-independent path.

- Example: path.join(__dirname, 'htmlfile', 'index.html')

9. res.sendFile() :- Sends an external HTML or other file to the client.

- Example: res.sendFile(path.join(__dirname, 'htmlfile', 'index.html'));

10. res.send():- Sends a simple text, HTML string, JSON, array, or buffer to the client.

- Example: res.send("This is Home Page");

11. app.get(path, callback)

- Creates a GET route.
- path → URL path (e.g., / or /about)
- callback(req, res) → Function executed when route is accessed.

12. app.listen(port, callback)

- Starts the Express server and listens on the given port.
- Example: app.listen(3000, () => console.log("Server running"));

- Middleware = function that processes requests before final response.

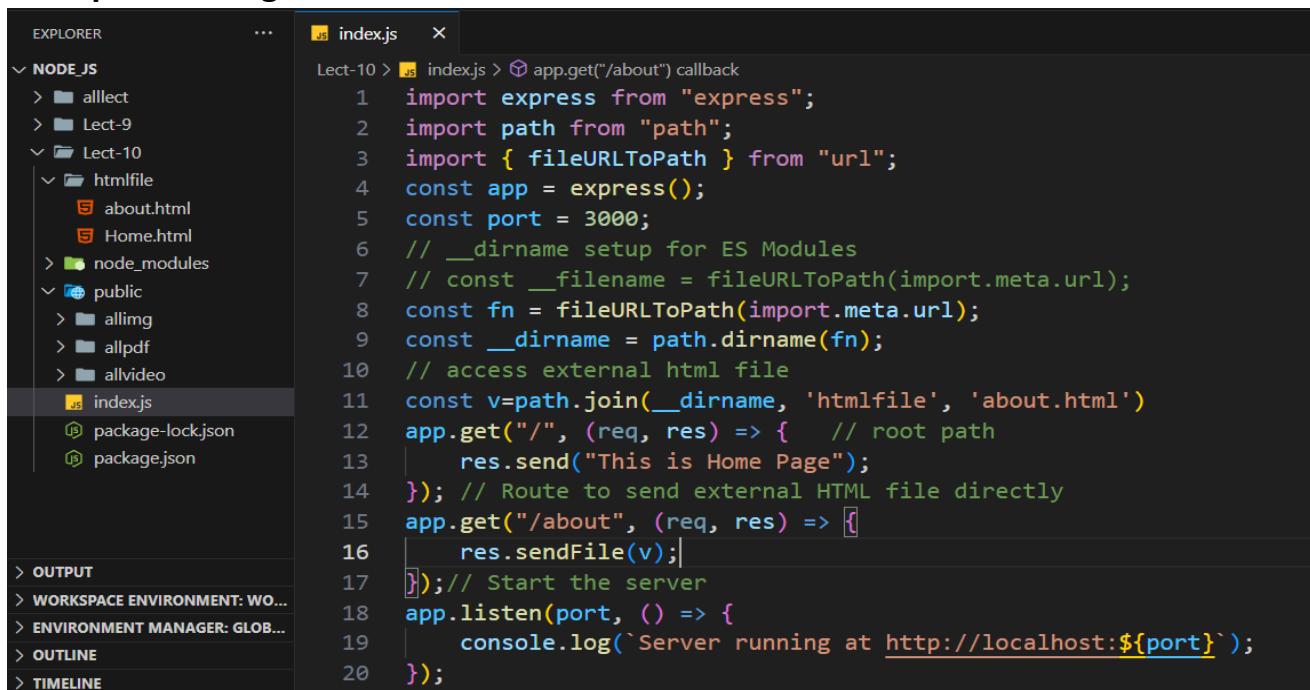
- Absolute path = full path from root to file/folder.

- Static files = CSS, JS, images, PDFs, videos served with express.static().

- Use res.sendFile() for HTML files, res.send() for text/JSON.



Example: sending external html



EXPLORER

- NODE_JS
 - allct
 - Lect-9
 - Lect-10
 - htmlfile
 - about.html
 - Home.html
 - node_modules
 - public
 - allimg
 - allpdf
 - allvideo
 - index.js
 - package-lock.json
 - package.json
- OUTPUT
- WORKSPACE ENVIRONMENT: WO...
- ENVIRONMENT MANAGER: GLOB...
- OUTLINE
- TIMELINE

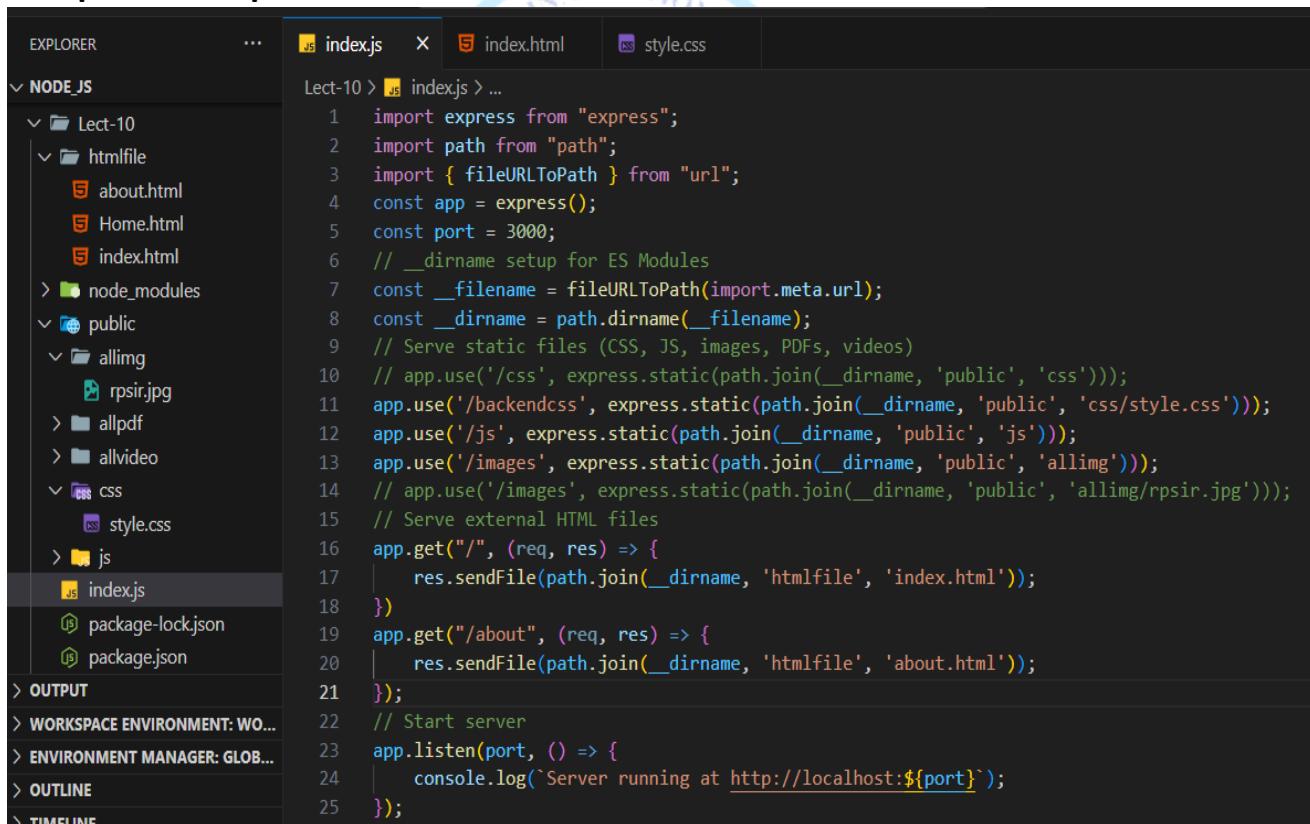
index.js

```

1 import express from "express";
2 import path from "path";
3 import { fileURLToPath } from "url";
4 const app = express();
5 const port = 3000;
6 // __dirname setup for ES Modules
7 // const __filename = fileURLToPath(import.meta.url);
8 const fn = fileURLToPath(import.meta.url);
9 const __dirname = path.dirname(fn);
10 // access external html file
11 const v=path.join(__dirname, 'htmlfile', 'about.html')
12 app.get("/", (req, res) => { // root path
13     res.send("This is Home Page");
14}); // Route to send external HTML file directly
15 app.get("/about", (req, res) => [
16     res.sendFile(v),
17 ]); // Start the server
18 app.listen(port, () => {
19     console.log(`Server running at http://localhost:\${port}`);
20 });

```

Complete Example



EXPLORER

- NODE_JS
 - Lect-10
 - htmlfile
 - about.html
 - Home.html
 - index.html
 - node_modules
 - public
 - allimg
 - rpsir.jpg
 - allpdf
 - allvideo
 - css
 - style.css
 - js
 - index.js
 - OUTPUT
 - WORKSPACE ENVIRONMENT: WO...
 - ENVIRONMENT MANAGER: GLOB...
 - OUTLINE
 - TIMELINE

index.js

index.html

style.css

```

1 import express from "express";
2 import path from "path";
3 import { fileURLToPath } from "url";
4 const app = express();
5 const port = 3000;
6 // __dirname setup for ES Modules
7 const __filename = fileURLToPath(import.meta.url);
8 const __dirname = path.dirname(__filename);
9 // Serve static files (CSS, JS, images, PDFs, videos)
10 // app.use('/css', express.static(path.join(__dirname, 'public', 'css')));
11 app.use('/backendcss', express.static(path.join(__dirname, 'public', 'css/style.css')));
12 app.use('/js', express.static(path.join(__dirname, 'public', 'js')));
13 app.use('/images', express.static(path.join(__dirname, 'public', 'allimg')));
14 // app.use('/images', express.static(path.join(__dirname, 'public', 'allimg/rpsir.jpg')));
15 // Serve external HTML files
16 app.get("/", (req, res) => {
17     res.sendFile(path.join(__dirname, 'htmlfile', 'index.html'));
18 })
19 app.get("/about", (req, res) => {
20     res.sendFile(path.join(__dirname, 'htmlfile', 'about.html'));
21 })
22 // Start server
23 app.listen(port, () => {
24     console.log(`Server running at http://localhost:\${port}`);
25 });

```

Index.html

index.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Home Page</title>
5  |   <!-- <link rel="stylesheet" href="/css/style.css"> -->
6  |   <!-- <link rel="stylesheet" href="/css/style.css"> -->
7  |   <link rel="stylesheet" href="/backendcss">
8  </head>
9  <body>
10 |     <h1 id="d1">Welcome to Home Page</h1>
11 |     
12 |     <!--  -->
13 |     <h1>this is backen code </h1>
14 </body>
15 </html>
16

```

style.css

```

1  #d1{
2      color: red;
3      background-color: green;
4      font-size: 40px;
5  }

```

index.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Home Page</title>
5  |   <!-- <link rel="stylesheet" href="/css/style.css"> -->
6  |   <!-- <link rel="stylesheet" href="/css/style.css"> -->
7  |   <link rel="stylesheet" href="/backendcss">
8  </head>
9  <body>
10 |     <h1 id="d1">Welcome to Home Page</h1>
11 |     
12 |     <!--  -->
13 |     <h1>this is backen code </h1>
14 </body>
15 </html>
16

```

Simple example for send external html file as response

server.js

```

1  import express from 'express'
2  import path from 'path'
3  const app=express()
4  app.get("/",(req, res)=>{
5      const dirname=path.resolve()
6      console.log("path=",dirname)
7      const path_urls=path.join(dirname, '/allhtml/index.html')
8      console.log("fullpath=",path_urls)
9      res.sendFile(path_urls)
10 })
11 const port=3000
12 app.listen(port, ()=>{
13     console.log(`server runing one "http://localhost:${port}"`)
14 })

```

Template Engines in Node.js/Express.js (EJS)

- ❖ **What is a Template Engine?**
 - template engine is a tool that allows you to build **HTML templates with dynamic content**. Means we can write js variable, condition and looping directly in html file
- ❖ **Embedded JavaScript (EJS):**
 - ejs means **writing JavaScript code directly inside HTML templates** to generate dynamic content on server.
- ❖ **It enables you to:**
 - Inject variables into HTML pages.
 - Use programming logic such as **loops** and **conditionals** directly inside templates.
 - Generate the final HTML page on the **server side** before sending it to the client's browser.
- ❖ **Why use one?**
 - **Clean separation:** keep your HTML in .ejs files, logic in routes/controllers.
 - **Dynamic content:** print variables, loop over lists, use if/else.
 - **Reuse:** include headers/footers (partials) and avoid copy-paste.

How to create Templates Engine or (EJS)

Step-1 Create Folder structure

```
my-app/
  ├── package.json
  ├── server.js      # Main server file
  ├── views/          # EJS templates
  │   ├── index.ejs
  │   ├── media.ejs
  │   ├── header.ejs
  │   └── footer.ejs
  └── public/          #
    ├── css/
    │   └── style.css
    ├── images/
    ├── pdfs/
    └── videos/
  └── node_modules/
```

Step 2: Initialize the project

- mkdir my-app
- cd my-app
- npm init -y
- npm install express
- npm install ejs

Step-3: Create server.js and set up Express server.

Step-4: Create index.ejs file inside views folder

Step-5: Configure Template Engine (set view engine to ejs and define views directory).

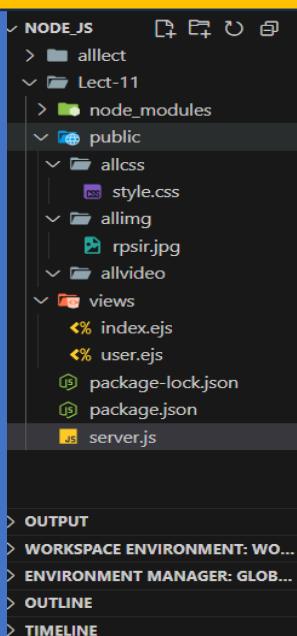
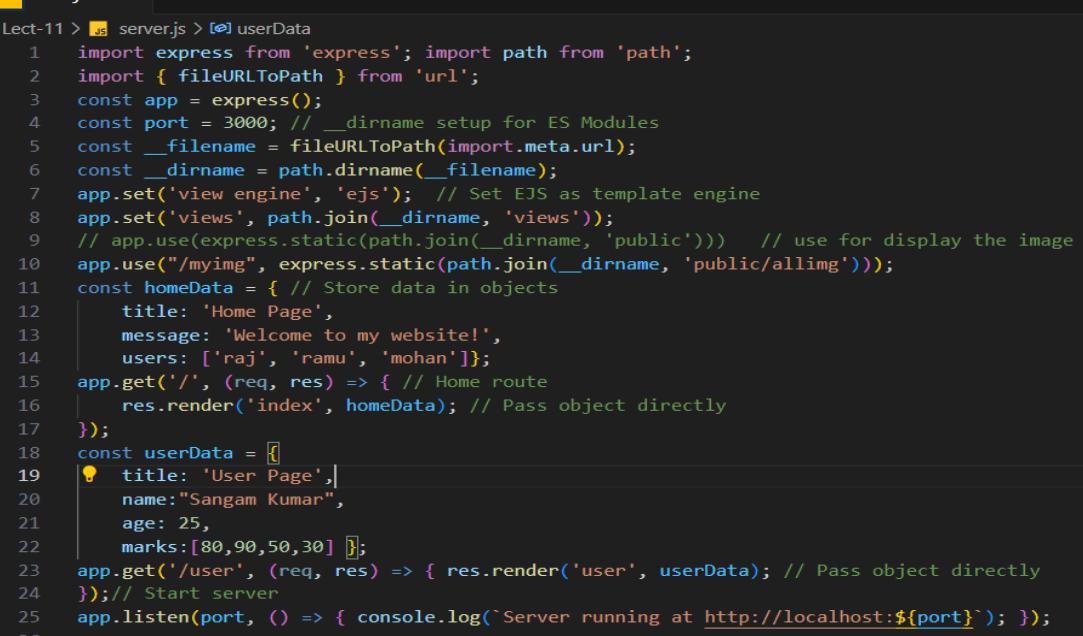
Step-6: Create Routes (/ , /user, etc.) and render .ejs templates with data.

Step-7: Start the Server and access templates in the browser (<http://localhost:3000>).



```
package.json
{
  "name": "lect-11",
  "version": "1.0.0",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "test": "echo \\"Error: no start script defined\\"",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "ejs": "^3.1.10",
    "express": "5.1.0"
  }
}
```

Step-3: Create server.js

```
server.js
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
const app = express();
const port = 3000; // __dirname setup for ES Modules
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
app.set('view engine', 'ejs'); // Set EJS as template engine
app.set('views', path.join(__dirname, 'views'));
// app.use(express.static(path.join(__dirname, 'public'))) // use for display the image
app.use("/myimg", express.static(path.join(__dirname, 'public/allimg')));
const(userData = { // Store data in objects
  title: 'Home Page',
  message: 'Welcome to my website!',
  users: ['raj', 'ramu', 'mohan']},
  app.get('/', (req, res) => { // Home route
    res.render('index', userData); // Pass object directly
  });
const(userData = { // User Page
  title: 'User Page',
  name: "Sangam Kumar",
  age: 25,
  marks: [80, 90, 50, 30];
  app.get('/user', (req, res) => { res.render('user', userData); // Pass object directly
}); // Start server
app.listen(port, () => { console.log(`Server running at http://localhost:${port}`); });
});
```

Explanation of the important points in server.js

1. Imports

```
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
• express → framework for server.
• path → handles file paths.
• fileURLToPath → converts ES module URL to file path.
```

2. __dirname setup (needed in ES Modules)

```
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
• Gets the current file's directory path (since ES modules don't have __dirname by default).
```

3. EJS setup

```
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
• view engine = ejs → Tells Express to render .ejs templates.
• views → Folder where EJS files are stored.
```

4. Static files (images, CSS, JS)

```
// app.use(express.static(path.join(__dirname, 'public')));
app.use("/myimg", express.static(path.join(__dirname, 'public/allimg/rpsir.jpg')));
• Normally: app.use(express.static(...)) → makes everything in public/ accessible.
• Your line (/myimg) → only serves that single image at http://localhost:3000/myimg.
```

Note: (Better to keep the first line if you want all images, CSS, JS accessible.)

5. Data objects

```
const homeData = { title: 'Home Page', message: 'Welcome to my website!', users: ['raj', 'ramu', 'mohan']};
const userData = { title: 'User Page', age: 25, isAdmin: false };
• homeData → passed to index.ejs.
• userData → passed to user.ejs.
```

6. Routes

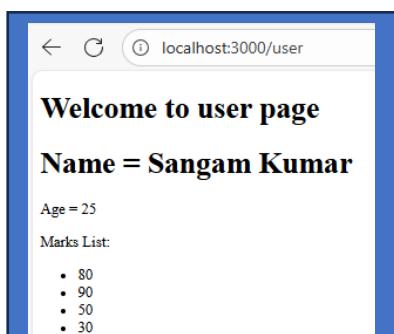
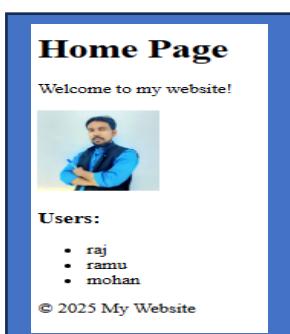
```
app.get('/', (req, res) => { res.render('index', homeData); });
app.get('/user', (req, res) => { res.render('user', userData); });
• / → loads index.ejs with homeData.
• /user → loads user.ejs with userData.
```

7. Start server

```
app.listen(port, () => { console.log(`Server running at http://localhost:\${port}`); });
• Starts server on port 3000.
```

Note:

- **Express** server with **EJS templates**.
- **Home (/)** → index.ejs with homeData.
- **User (/user)** → user.ejs with userData.
- **Static files** served (currently only one image via /myimg).



Step-3,4: Create index.ejs and user.ejs

```
Lect-11 > views > index.ejs > html > body > main > ul > ? > li
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>%= title %</title>
6      <link rel="stylesheet" href="/css/style.css">
7  </head>
8  <body>
9      <header>
10         <h1>%= title %</h1>
11     </header>
12     <main>
13         <p>%= message %</p><!-- Correct way: -->
14     <!-- 
15     
16     <h3>Users:</h3>
17     <ul>
18         <% users.forEach(user => { %>
19             <li>%= user %</li>
20         <% }) %>
21     </ul>
22 </main>
23 <footer>
24     <p>&copy; <%= new Date().getFullYear() %> My Website</p>
25 </footer>
```

```
Lect-11 > views > user.ejs > html > body > ul > ?
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>%= title %</title>
6  </head>
7  <body>
8
9      <h1>Welcome to user page</h1>
10     <h1>Name = <%= name %></h1>
11     <p>Age = <%= age %></p>
12
13     <p>Marks List:</p>
14     <ul>
15         <% marks.forEach(function(m) { %>
16             <li>%= m %</li>
17         <% }) %>
18     </ul>
19
20 </body>
21 </html>
```

Expiations

1. <%= title %> & <%= message %> → Inserts dynamic values from server (coming via res.render).
2. <link rel="stylesheet" href="/css/style.css"> → Loads external CSS from public/css/style.css.
3. Image tag () → Displays image served from Express static route /myimg.
4. users.forEach loop → Iterates over the array of users and displays each one as a list item.
5. <%= new Date().getFullYear() %> → Prints the current year automatically in footer.

Note: - This template dynamically shows **title, message, image, list of users, and current year**, all styled with external CSS.

Why we use <%= %> in EJS?

- <%= %> → Used to **output dynamic values as HTML (escaped)**.
Example: <%= title %> → prints the value of title sent from server.
- <% %> → Used for **JavaScript logic only** (loops, conditions). It does not print anything directly.
Example: <% users.forEach(...) %> → runs the loop but doesn't output itself.



- Spaces inside JS (if, for, foreach) are fine → same as normal JS.
 - After <%= keep variable name immediately → <%= user %>.
 - Don't add random spaces before closing %>.
1. **Inside <% %> (logic only) Spaces don't matter inside <% %>**
 - You can give spaces freely (like normal JS)

```
<% if (isAdmin) { %>          → ok
<% if(isAdmin){ %>          → ok
<% for (let i = 0; i < 5; i++) { %>  → ok
```
 2. **Inside <%= %> (output) No space after = before variable name.**

```
<%= user %>          → correct
<%=user%>          → also correct
<%= user %>          → avoid extra spaces at the end
```
 3. **Closing Tags** Always write <% } %> without putting JS code after %>.

```
<% } %>          → correct
<% }%>          → correct
<% } %>          → avoid extra spaces before '>'
```

for loop & conditions in EJS:

1. Loop in EJS

```
<ul>
<% users.forEach(user => { %>
    <li>%= user %</li>
<% }) %>
</ul> Iterates over array users and prints each user.
```

2. If Condition

```
<% if (isAdmin) { %>
    <p>Welcome Admin!</p>
<% } else { %>
    <p>Welcome User!</p>
<% } %>
Checks isAdmin (boolean) and shows message.
```

3. For Loop

```
<% for(let i = 1; i <= 5; i++) { %>
    <p>Number: <%= i %></p> <% } %>
    • <% %> = run JavaScript logic (loop/if).
    • <%= %> = print variable value inside HTML.
```



Examples of loops & conditions in EJS

1. Check Even/Odd Number

```
<% for(let i = 1; i <= 5; i++) { %>
<% if(i % 2 === 0) { %>
<p><%= i %> is Even</p>
<% } else { %>
<p><%= i %> is Odd</p>
<% } %>
<% } %>
```

→ Prints numbers 1–5 with Even/Odd check.

2. Show Marks with Pass/Fail

```
<% marks.forEach(m => { %>
<p>
Marks: <%= m %> -
<% if(m >= 40) { %>
Pass
<% } else { %>
Fail
<% } %>
</p>
<% }) %>
```

→ Loops through marks array, shows Pass/Fail.

6. Show First 3 Users Only

```
<% for(let i = 0; i < 3; i++) { %>
<p>User: <%= users[i] %></p>
<% } %>
```

→ Displays only first 3 users.

7. Conditional Message

```
<% if(users.length === 0) { %>
<p>No users found!</p>
<% } else { %>
<p>Total Users: <%= users.length %></p>
<% } %>
```

→ Checks if array is empty.

8. Nested Loop Example

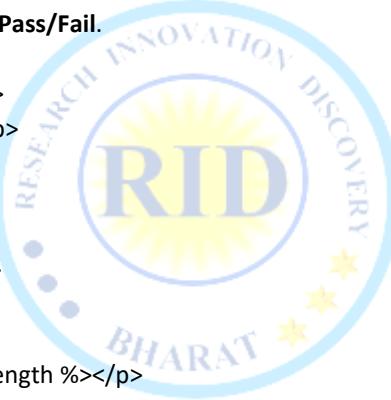
```
<% classes.forEach((cls, i) => { %>
<h3>Class <%= i+1 %></h3>
<ul>
<% cls.forEach(student => { %>
<li><%= student %></li>
<% } %>
</ul>
<% }) %>
```

→ Loops through **array of arrays** (like students in multiple classes).

9. Ternary Operator (inline condition)

```
<p>Status: <%= isAdmin ? "Admin" : "User" %></p>
```

→ One-line condition using ternary operator.



How to add header and footer on all EJS pages and route the route in EJS(HTML file)

Home Gallery Contact Service News

Login Signup

Welcome

This is a simple navbar page with a footer.

- Routes are written in Express (server), not in EJS.
- EJS only uses `` or `<form action="/route">` to connect to routes.
- Use GET routes to render pages (e.g., `/home`, `/news`).
- Use POST routes to handle form submissions (e.g., `signup`, `login`).
- Dynamic routes use parameters (e.g., `/news/:id`).
- Pass data from routes → EJS with `res.render("file", { data })`.
- Include partials for header/footer with `<% include("partials/header") %>`.

Step-1: - Create project structure

```
Lect-11/
  ├── app.js (or server.js)
  └── package.json

  └── public/
      └── css/
          └── index.css
          └── nav.css

  └── views/
      ├── index.ejs
      ├── gallery.ejs
      ├── contact.ejs
      ├── service.ejs
      ├── news.ejs
      ├── login.ejs
      ├── signup.ejs
      └── partials/
          ├── header.ejs
          └── footer.ejs
```

```
└── Lect-11
    ├── node_modules
    └── public
        ├── allimg
        └── allvideo
            ├── css
            │   ├── index.css
            │   └── style.css
            └── views
                └── partials
                    ├── footer.ejs
                    ├── header.ejs
                    ├── Contact.ejs
                    ├── Gallery.ejs
                    ├── index.ejs
                    ├── Login.ejs
                    ├── News.ejs
                    ├── Service.ejs
                    ├── Signup.ejs
                    └── package-lock.json
                    └── package.json
                    └── server.js
```

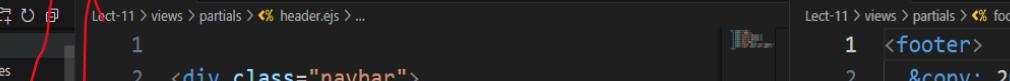
Step-2: -Configure Express (myserver.js)

```
Lect-11 > server.js > data > "Name"
1  // server.js
2  import express from "express";
3  import path from "path";
4  import { fileURLToPath } from "url";
5  const app = express();
6  const port = 3000;// setup __dirname for ES Modules
7  const __filename = fileURLToPath(import.meta.url);
8  const __dirname = path.dirname(__filename)// Set EJS as template engine
9  app.set("view engine", "ejs");
10 app.set("views", path.join(__dirname, "views"));
11 // Serve static files (CSS, Images, JS)
12 app.use("/myimg", express.static(path.join(__dirname, "public/allimg")));
13 app.use(express.static(path.join(__dirname, "public"))); // for CSS, JS
14 const data={ "Name": "Sangam Kumar", "Age": 18}
15 app.get("/", (req, res) => {res.render("index", {data});});// index route
16 app.get("/Gallery", (req, res) => {res.render("Gallery");}); // Gallery route
17 app.get("/Contact", (req, res) => {res.render("Contact");}); // Gallery route
18 app.get("/Service", (req, res) => {res.render("Service");});// Service route
19 app.get("/News", (req, res) => {res.render("News");});// News route
20 app.get("/Signup", (req, res) => {res.render("Signup");});// Signup route
21 app.get("/Login", (req, res) => {res.render("Login");});// Login route
22 app.listen(port, () => {console.log(`Server running at http://localhost:${port}`);});
```



3) Create the partials views/partials/header.ejs

“<%- %> for includes”



The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure. The **views** folder contains **partials**, which includes **header.ejs** and **footer.ejs**. Other files in **views** include **Contact.ejs**, **index.ejs**, and **style.css**. The **public** folder contains **allimg** and **allvideo**.
- header.ejs** file content (10 lines):

```
1 <div class="navbar">
2   <a href="/">Home</a>
3   <a href="/Gallery">Gallery</a>
4   <a href="/Contact">Contact</a>
5   <a href="/Service">Service</a>
6   <a href="/News">News</a>
7   <a href="/Signup" class="right">Signup</a>
8   <a href="/Login" class="right">Login</a>
9
10 </div>
```
- footer.ejs** file content (4 lines):

```
1 <footer>
2   &copy; 2025 My Website
3 </footer>
4
```

4) Use the partials in every page

- Top of each page: include header and pass the title (already sent from the route).
Bottom: include footer.

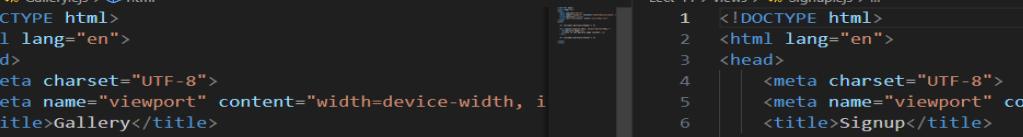
Syntax:

```
<%- include("partials/header") %>
<h2>Welcome to Home</h2>
<p>This is the homepage content. </p>
<%- include("partials/footer") %>
```

Common mistakes (to avoid that “Unexpected identifier ‘partials’” error)
Use `<%- include("partials/header") %>` (with quotes and parentheses).
Don’t write partial/header or partials/header without `include()`.
Keep partial paths relative to views/ (no leading slash).
Use `<%- ... %>` (not `<%= ... %>`) for includes.



```
Lect-11 > views > <% index.ejs > <% Contact.ejs > <% html > <% body >
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <title>Home</title>
6  |   <link rel="stylesheet" href="/css/index.css">
7  </head>
8  <body>
9  |   <%- include('partials/header') %>
10 <div style="padding:20px; margin-bottom:60px;">
11 |   <h1>Welcome</h1>
12 |   <p>This is a simple navbar page with a footer.</p>
13 </div>
14 <%- include('partials/footer') %>
15 </body>
16 </html>
17
18
19
Lect-11 > views > <% Contact.ejs > <% html >
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device"
6  |   <title>Contact</title>
7  |   <link rel="stylesheet" href="/css/index.css">
8  </head>
9  <body>
10 |   <%- include('partials/header') %>
11 <div style="padding:20px; margin-bottom:60px;">
12 |   <h1>Contact Page</h1>
13 |   <p>This is the contact page content.</p>
14 </div>
15
16 <%- include('partials/footer') %>
17
18 </body>
19 </html>
```



```
<% Gallery.ejs %> Lect-11 > views > <% Gallery.ejs > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, i
6      <title>Gallery</title>
7      <link rel="stylesheet" href="/css/index.css">
8  </head>
9  <body>
10     <%- include('partials/header') %>
11
12     <div style="padding:20px; margin-bottom:60px;">
13         <h1>Gallery Page</h1>
14         <p>This is the gallery page content.</p>
15     </div>
16
17     <%- include('partials/footer') %>
18
19 </body>
20 </html>| Lect-11 > views > <% Signup.ejs > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=de...
6      <title>Signup</title>
7      <link rel="stylesheet" href="/css/index...
8  </head>
9  <body>
10     <%- include('partials/header') %>
11     <div style="padding:20px; margin-bottom:60px;">
12         <h1>Signup Page</h1>
13         <p>This is the singup page content.</p>
14     </div>
15
16     <%- include('partials/footer') %>
17
18 </body>
19 </html>
```

Form handling (How to get data from the form means front to backend)

Rules for Form Handling

1. Create Form in EJS/HTML

- **Rule:** form tag must have method="post" and action="/your-route".
- **Syntax:**

```
<form action="/route-name" method="post">
  <input type="text" name="fieldName">
  <button type="submit">Submit</button> or
  <input type="submit" value="submit">
</form>
```

2. Enable Body Parsing Middleware in Express

- **Rule:** Without this, req.body will be undefined.
- **Syntax:**

```
app.use(express.urlencoded({ extended: true })); // for form data
app.use(express.json()); // for JSON data (API/AJAX)
```

3. Define Backend Route for Form Submission

- **Rule:** Use app.post() (not app.get()) for form submission.
- **Syntax:**

```
app.post("/route-name", (req, res) => {
  const data = req.body; // access form data
  console.log(data); // check in console
  res.send("Form submitted!"); or
  res.render("ejs_file name")
});
```

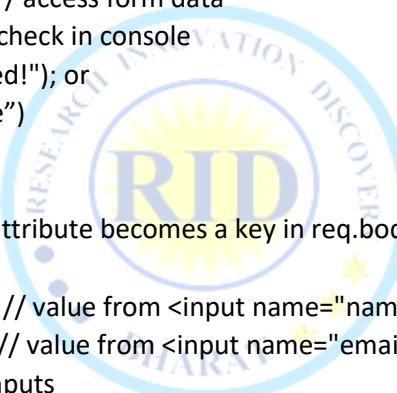
4. Access Data in Backend

- **Rule:** Each input's name attribute becomes a key in req.body.
- **Syntax:**
 - **req.body.name;** // value from <input name="name">
 - **req.body.email;** // value from <input name="email">
 - **req.body** → text inputs
 - **req.file** → single uploaded file
 - **req.files** → multiple uploaded files
 - **enctype="multipart/form-data"** → required for file uploads

5. Send Response

- **Rule:** Always send a response (render a page or send message).
- **Syntax:**

```
res.send("Form received!");
// or
res.render("success", { user: req.body });
```



EXPLORER ...

✓ NODE_JS

- ✓ Lect-11
- > node_modules
- ✓ public
- > allimg
- > allvideo
- ✓ css
- ✓ index.css
- ✓ style.css
- ✓ views
- ✓ partials
- ✓ footer.ejs
- ✓ header.ejs
- ✓ Contact.ejs
- ✓ Gallery.ejs
- ✓ index.ejs
- ✓ Login.ejs
- ✓ News.ejs
- ✓ Service.ejs
- ✓ Signup.ejs
- ✓ package-lock.json
- ✓ package.json
- ✓ server.js

↳ Signup.ejs x

Lect-11 > views > Signup.ejs > html > body

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  | <title>Signup</title>
5  | <link rel="stylesheet" href="/css/index.css">
6  </head>
7  <body>
8  <%- include('partials/header') %>
9  <h2>Signup Form</h2>
10 <form action="/req-signup" method="post">
11 | Name:<input type="text" name="name" required><br><br>
12 | Email:<input type="email" name="email" required><br><br>
13 | Mobile No:<input type="text" name="mobile" required><br><br>
14 | Password:<input type="password" name="password" required><br><br>
15 | Confirm Password:<input type="password" name="confirm" required><br><br>
16 | <button type="submit">Sign Up</button>
17 </form>
18 <%- include('partials/footer') %>
19 </body>
20 </html>
21

```

js server.js x

Lect-11 > server.js > app.post("/req-signup") callback > user

```

1 import express from "express"; import path from "path"; import { fileURLToPath } from "url"
2 const app = express(); const port = 3000; // Setup __dirname for ES Modules
3 const __filename = fileURLToPath(import.meta.url);
4 const __dirname = path.dirname(__filename); // Set EJS as template engine
5 app.set("view engine", "ejs"); app.set("views", path.join(__dirname, "views"));
6 // Middleware to parse form data from front end to backend
7 app.use(express.urlencoded({ extended: true })); // Serve static files (CSS, Images, JS)
8 app.use("/myimg", express.static(path.join(__dirname, "public/allimg")));
9 app.use(express.static(path.join(__dirname, "public"))); // for CSS, JS
10 const data = { "Name": "Sangam Kumar", "Age": 18 };
11 app.get("/", (req, res) => { res.render("index", data); }); // index route
12 app.get("/Gallery", (req, res) => { res.render("Gallery"); });
13 app.get("/Contact", (req, res) => { res.render("Contact"); });
14 app.get("/Service", (req, res) => { res.render("Service"); });
15 app.get("/News", (req, res) => { res.render("News"); });
16 app.get("/Signup", (req, res) => { res.render("Signup"); });
17 app.post("/req-signup", (req, res) => {
18   const user = [
19     name: req.body.name,
20     email: req.body.email,
21     mobile: req.body.mobile,
22     password: req.body.password,
23     confirm: req.body.confirm
24   ]; console.log("User data:", user); // res.send("Signup form submitted successfully!");
25   res.render("Login.ejs");
26   app.get("/Login", (req, res) => { res.render("Login"); });
27   app.listen(port, () => { console.log(`Server running at http://localhost:${port}`); });
28

```

Example student registration form

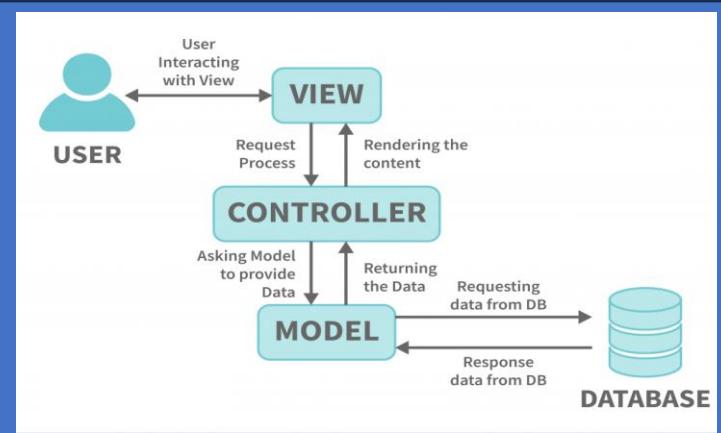
```
<!DOCTYPE html>
<html>
<head>
<title>Student Registration</title>
</head>
<body>
<h2>Student Registration Form</h2>
<form action="/register" method="post" enctype="multipart/form-data">
<!-- Basic Info -->
<label>Name:</label>
<input type="text" name="name" required><br><br>
<label>Email:</label>
<input type="email" name="email" required><br><br>
<label>Mobile No:</label>
<input type="text" name="mobile" required><br><br>
<label>Date of Birth:</label>
<input type="date" name="dob" required><br><br>
<!-- Gender -->
<label>Gender:</label>
<input type="radio" name="gender" value="Male" required> Male
<input type="radio" name="gender" value="Female" required> Female
<input type="radio" name="gender" value="Other" required> Other<br><br>
<!-- File Uploads -->
<label>Profile Image:</label>
<input type="file" name="image" accept="image/*" required><br><br>
<label>Marksheet (PDF):</label>
<input type="file" name="marksheet" accept="application/pdf" required><br><br>
<label>Short Video:</label>
<input type="file" name="video" accept="video/*" required><br><br>
<!-- Branch -->
<label>Branch:</label>
<select name="branch" required>
<option value="">Select Branch</option>
<option value="CSE">CSE</option>
<option value="ECE">ECE</option>
<option value="ME">ME</option>
<option value="CE">CE</option>
</select><br><br>
<button type="submit">Register</button>
</form>
</body>
</html>
```

Key Points:

- `enctype="multipart/form-data"` is **required** for file uploads.
- name attribute for each input is used to access the data in backend (req.body for text, req.file/req.files for files).
- Simple dropdown for branch, radio buttons for gender.



MVC Model in Node JS/ Express JS

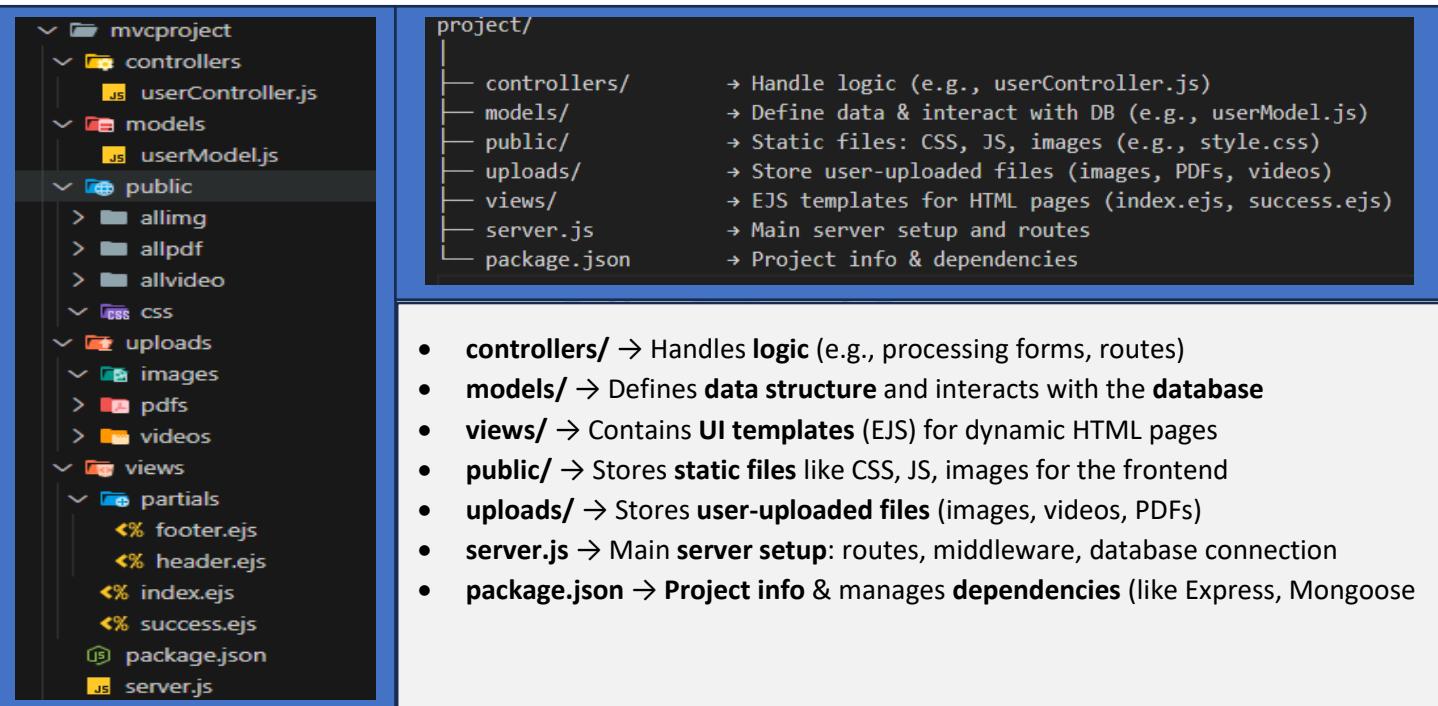


MVC stands for Model-View-Controller, is way to organize code in a clean and scalable way.

- 1) **Model:** - Handles **data** and **database logic**. Example: Defines what a "User" is and talks to the database.
- 2) **View:** - Handles the **UI (User Interface)**. What the user sees (HTML, templates like EJS, etc.).
- 3) **Controller:** - Handles **requests & business logic**. Connects Model & View. Gets data from model and sends to view.

Why use MVC? - Keeps code organized. Easy to maintain and scale.

Step-1: Create Folder Structure



Step-2: Initialize Project & Install Packages

- ✓ mkdir student-register
- ✓ cd student-register
- ✓ npm init -y
- ✓ **npm install express mongoose ejs body-parser multer**

```

{
  "dependencies": {
    "body-parser": "^2.2.0",
    "ejs": "^3.1.10",
    "express": "^5.1.0",
    "mongoose": "^8.18.0",
    "multer": "^2.0.2"
  }
}
  
```

- 1) **npm init -y** → Creates package.json with default settings to manage project and dependencies.
- 2) **npm install express** → Installs Express, a web framework to create servers and handle routes.
- 3) **npm install mongoose** → Installs Mongoose to define schemas and interact with MongoDB.
- 4) **npm install ejs** → Installs EJS template engine to render dynamic HTML pages.
- 5) **npm install body-parser** → Installs middleware to parse incoming form data (JSON or URL-encoded).
- 6) **npm install multer** → Installs middleware to handle file uploads like images, PDFs, and videos.

Step-3. Connect to MongoDB: - Use MongoDB Atlas or local MongoDB. Replace <your_mongo_uri> in server.js.

How to setting up MongoDB Atlas and MongoDB Compass/Shell:

Step 1: Create MongoDB Atlas Account

1. Go to <https://www.mongodb.com/cloud/atlas>
2. Click **Sign Up** and create an account (or login if you have one)
3. Verify your email and complete signup

Step 2: Create a Cluster

1. Click **Build a Cluster** → Choose **Free Tier**
2. Select **Cloud Provider & Region** (default is fine)
3. Click **Create Cluster** → Wait a few minutes

Step 3: Create Database User

1. Go to **Database Access** → Click **Add New Database User**
2. Set **Username & Password** → Give **Read and Write** access
3. Save the user

Step 4: Allow IP Access

1. Go to **Network Access** → Click **Add IP Address**
2. Choose **Allow Access from Anywhere (0.0.0.0/0)** → Save

Step 5: Connect Cluster

1. Click **Connect** → Choose **Connect Your Application**
2. Copy **MongoDB URI** → Replace <username> and <password> with your database user credentials

Step 6: Download MongoDB Compass & Shell (Optional)

1. Go to <https://www.mongodb.com/try/download>
2. Download **MongoDB Compass** (GUI to manage DB)
3. Download **Mongo Shell** (CLI to run commands)
4. Install both

The screenshot shows the MongoDB Atlas interface. At the top, there is a navigation bar with the 'Atlas' logo, a dropdown for 'Rajesh's Org...', a gear icon for 'Access Manager', and a 'Billing' link. Below the navigation bar, there is a dropdown for 'Project 0' and a 'Data Services' tab which is currently selected. A search bar says 'Search for a Project...'. Below the search bar, there is a list of projects, with 'Project 0' being the active one. At the bottom of the list, there are two buttons: 'View All Projects' and '+ New Project'. To the right of the project list, there is a sidebar with some status messages. At the bottom of the screen, there is a large 'Overview' section.

← → ⌂ cloud.mongodb.com/v2#/org/67dac95bf9bb834fac7062bd/projects/create

Atlas Rajesh's Org... Access Manager Billing

RAJESH'S ORG - 2025-03-19 > PROJECTS

Create a Project

Projects

Alerts 0

Activity Feed

Settings

Integrations

Access Manager

Resource Policies

Billing

Support

Live Migration

Name Your Project Add Members

Name Your Project

Project names have to be unique within the organization (and other restrictions).

newproject

Add Tags (Optional)

Use tags to efficiently label and categorize your projects. A project can have a maximum of 50 tags. You can modify tags for the project later. [Learn more](#)

Key	Value	Actions
Select a key or enter your own	Select a value or enter your own	
+ Add tag		0 TAGS

Cancel Next

Click to go back, hold to see history Access Manager Billing

Create a Project

Projects

Alerts 0

Activity Feed

Settings

Integrations

Access Manager

Resource Policies

Billing

Support

Live Migration

Name Your Project Add Members

Add Members and Set Permissions

Invite new or existing users via email address...

Give your members access permissions below.

rojeshpd339@gmail.com (you) Project Owner

Back Cancel Create Project

Create cluster: - cluster means database(table)

Your organization does not have a designated security contact. Add an Atlas Security Contact in [Organization Settings](#) to receive security-related notifications.

RAJESH'S ORG - 2025-03-19 > NEWPROJECT

Overview

Create a cluster

Choose your cloud provider, region, and specs.

+ Create

It will take some time then it will redirect to new page

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10 \$0.08/hour

Dedicated cluster for development environments and low-traffic applications.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

Flex From \$0.011/hour
Up to \$30/month

For development and testing, with on-demand burst capacity for unpredictable traffic.

STORAGE	RAM	vCPU
5 GB	Shared	Shared

Free

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

Free forever! Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Configurations

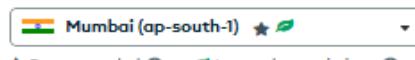
Name
You cannot change the name once the cluster is created.

Cluster0

Provider



Region



I'll do this later

Quick setup

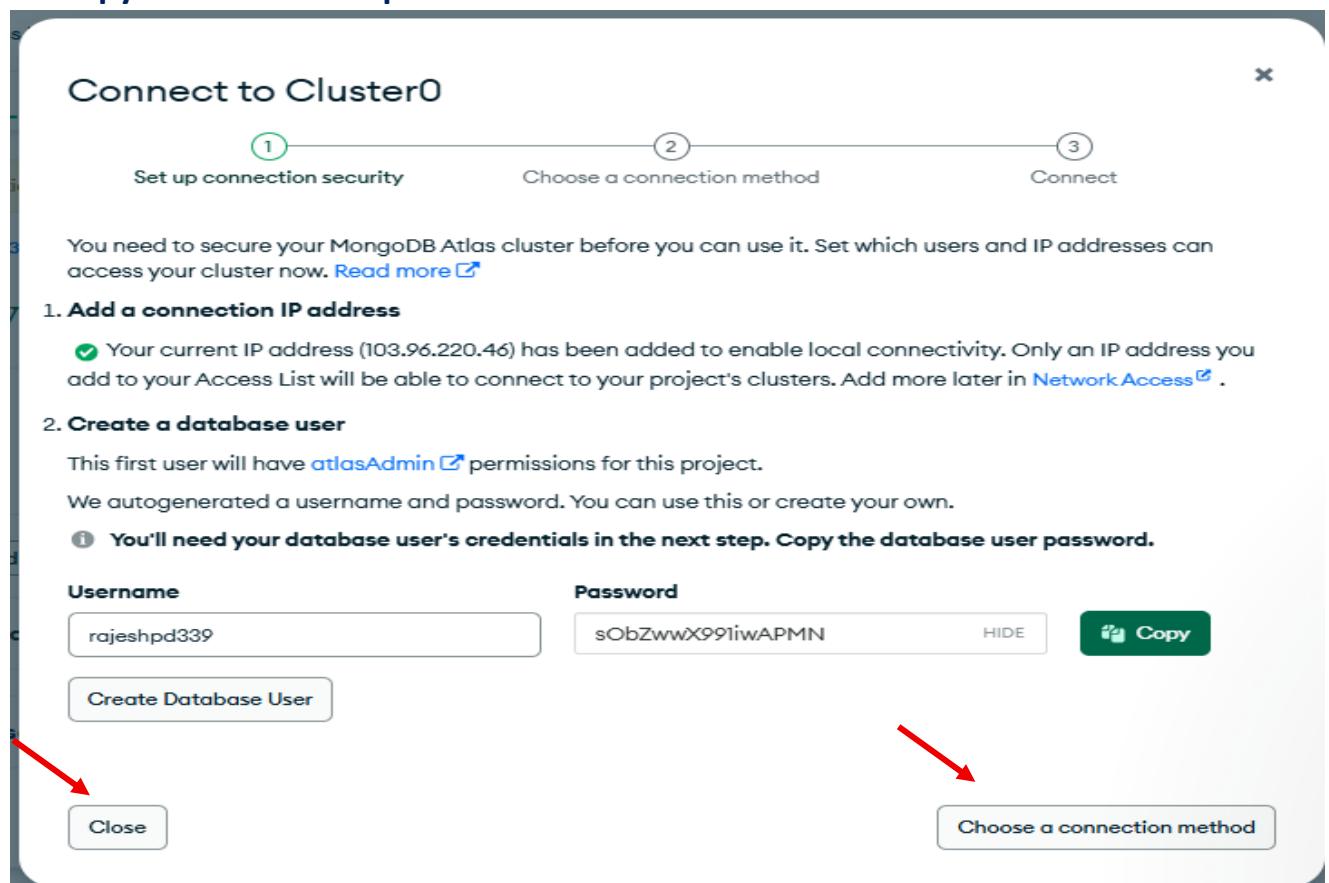
- Automate security setup (i)
- Preload sample dataset (i)

[Go to Advanced Configuration](#)

[Create Deployment](#)

→ Select free and aws to deploy your cluster (database to cloud)

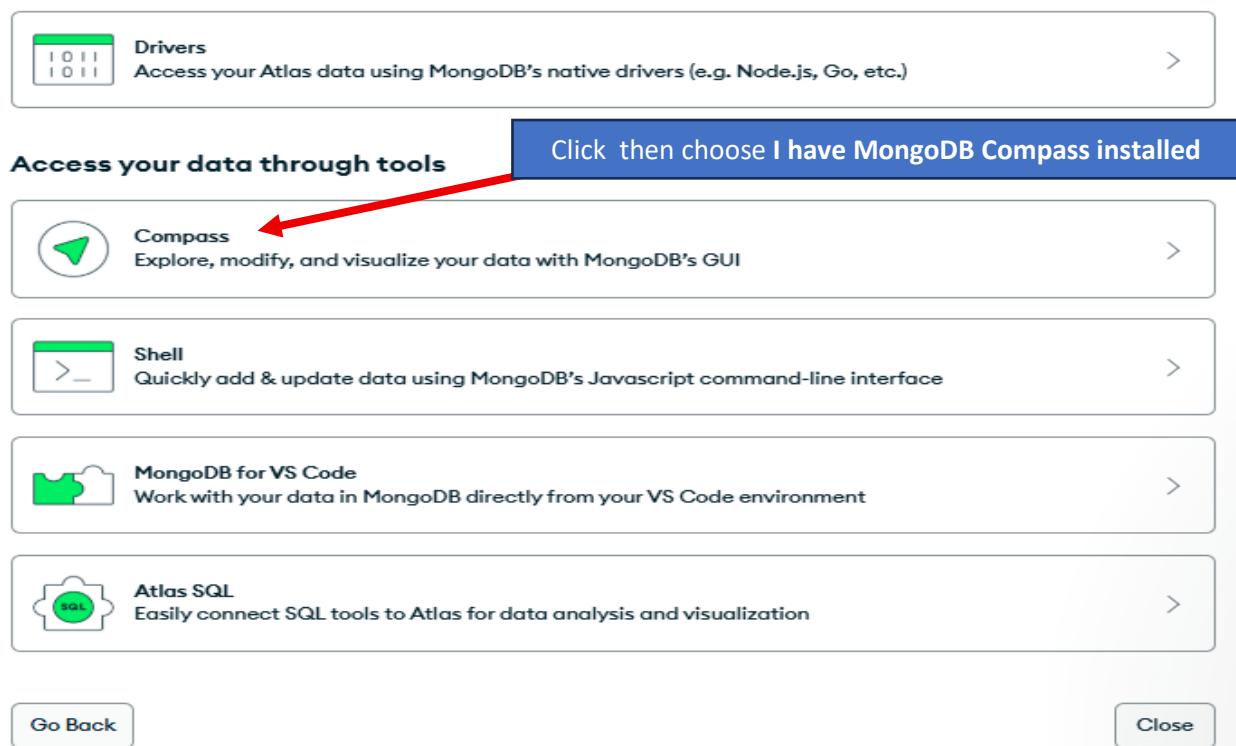
→ Copy user name and password



User name: - rajeshpd339 Password: - sObZwwX991iwAPMN → sObZwwX991iwAPMN

→ click create database user and click choose connection method

Connect to your application



→ Copy this created urls for connection with mongodb compass to see data

Connect to Cluster0

Set up connection security Choose a connection method Connect

Connecting with MongoDB Compass

I don't have MongoDB Compass installed I have MongoDB Compass installed

1. Choose your version of Compass

1.38 or later

See your Compass version in "About Compass"

2. Copy the connection string, then open MongoDB Compass

Use this connection string in your application

mongodb+srv://rajeshpd339:<db_password>@cluster0.4d5kdsy.mongodb.net/

Replace <db_password> with the password for the **rajeshpd339** user. Ensure any options are **URL encoded**. You can edit your database user password in [Database Access](#).

Urls: `mongodb+srv://rajeshpd339:<db_password>@cluster0.4d5kdsy.mongodb.net/`

→ Replace <username> and <password> with your DB user details. Keep /test or replace with your database name.

Url → `mongodb+srv://rajeshpd339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/test`

→ Open your mongo db compass in local system and then remove local urls and paste this urls
Remove this urls : `mongodb://localhost:27017`

New Connection

Manage your connection settings

URI [Edit Connection String](#)

mongodb://localhost:27017

Name

Color

Favorite this connection

Favoriting a connection will pin it to the top of your list of connections

Advanced Connection Options

[Cancel](#) [Save](#) [Connect](#) [Save & Connect](#)

URI [Edit Connection String](#)

mongodb+srv://rajeshpd339:*****@cluster0.4d5kdsy.mongodb.net/

How do I find my connection string in Atlas?

If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect. See example

How do I format my connection string?

See example

- If any error is coming then go to Network access and allow for anywhere

The screenshot shows the MongoDB Atlas interface. The left sidebar has a 'Network Access' link, which is highlighted with a red arrow. A modal window titled 'Edit IP Access List Entry' is open, containing a button labeled 'ALLOW ACCESS FROM ANYWHERE' with a red arrow pointing to it. The 'Confirm' button at the bottom right of the modal is also highlighted with a red arrow.

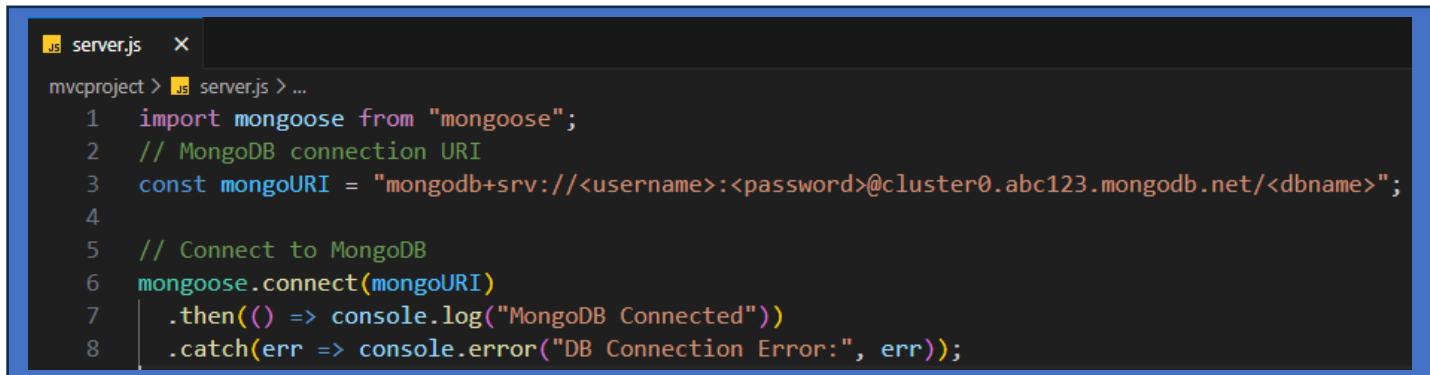
- Now your mongo db. connection will be done now we can send the data from express.

The screenshot shows the MongoDB Compass application interface. The top bar has 'MongoDB Compass' and a 'Welcome' message. The main area is titled 'Compass' and shows a 'My Queries' section. Below it is a 'CONNECTIONS (1)' section with a search bar and a list of connections. One connection is expanded, showing the 'admin', 'config', and 'local' databases.

After connection successful we see admin, config, local now move to next step-4 for model design.

Step-4: - Connect to MongoDB with express js(Backend to Database)

- Install mongosh for connect the **backend to MongoDB**
- **Example:** npm install -g mongosh check install or not:-> mongosh –version
- Connect to MongoDB use this urls
Mongosh"mongodb+srv://rajeshpd339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/test"



```
js server.js x
mvcproject > js server.js > ...
1 import mongoose from "mongoose";
2 // MongoDB connection URI
3 const mongoURI = "mongodb+srv://<username>:<password>@cluster0.abc123.mongodb.net/<dbname>";
4
5 // Connect to MongoDB
6 mongoose.connect(mongoURI)
7   .then(() => console.log("MongoDB Connected"))
8   .catch(err => console.error("DB Connection Error:", err));
```

- Replace <username> and <password> with your MongoDB credentials.
- Replace <dbname> with your database name.
- In Mongoose 6+ version, useNewUrlParser and useUnifiedTopology are optional

How to give the database name in Express.js with Mongoose

Step 1: Install Required Packages → npm install express mongoose

Step 2: Import Packages → import mongoose from "mongoose";

Step 3: Create Express App → const app = express();

Step 4: Add Database Name in Connection URI

→ **Database name is "mydatabase"**

const mongoURI = "mongodb+srv://<username>:<password>@cluster0.abc123.mongodb.net/mydatabase";

Note: Replace <username> and <password> with your MongoDB Atlas credentials.

- mydatabase at the end is the **database name**.

Step 5: Connect to MongoDB

```
mongoose.connect(mongoURI)
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.error("DB Connection Error:", err));
```

Step 6: Start Express Server

```
app.get("/", (req, res) => {
  res.send("Hello, MongoDB is connected!");
  app.listen(port, () => { console.log(`Server is running on ${port}`);});
```

Note: Database name is always in the URI: ...mongodb.net/mydatabase

- No need to give dbName or ddbname in Mongoose options.
- If the database doesn't exist, MongoDB will create it automatically when you add data.

❖ How to check the database is created or not

In Atlas

- Go to Cluster → Collections.
- Check if userdatabase exists.
- If not, click Create Database → Name: userdatabase, Collection: users.

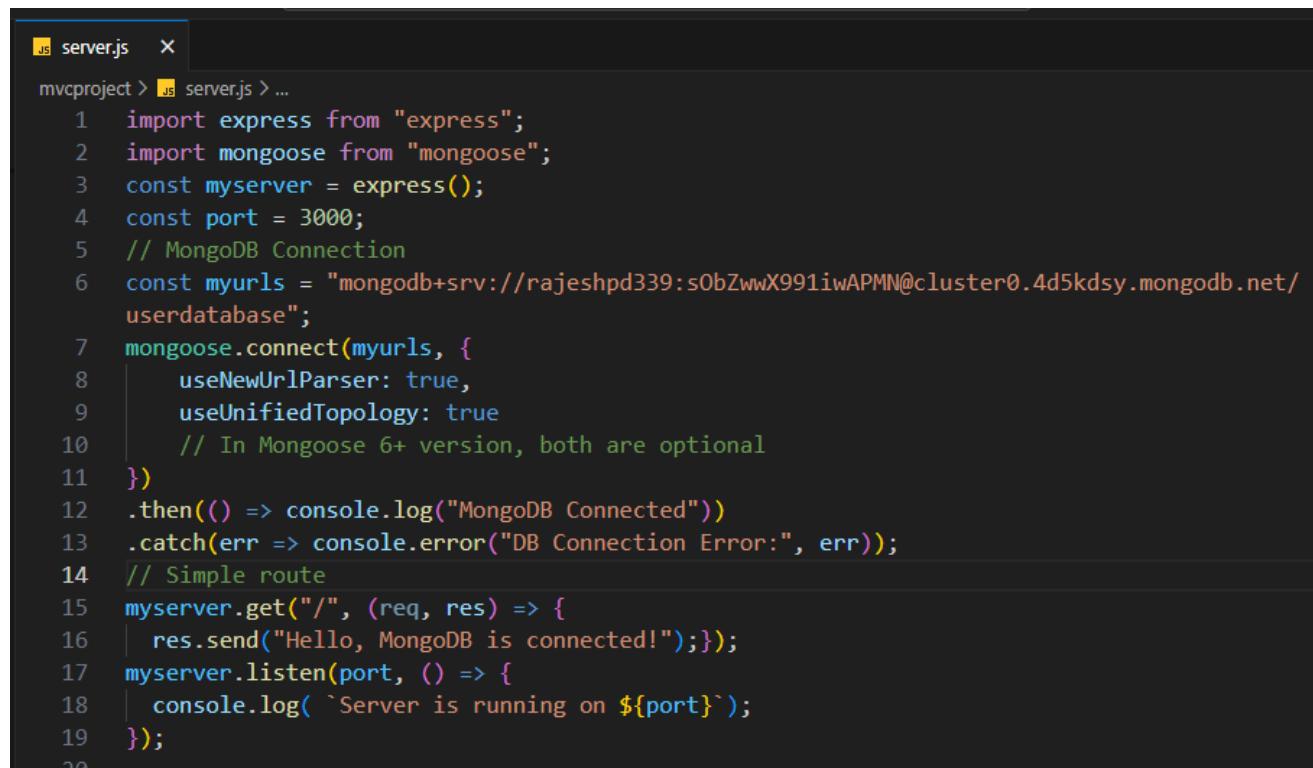
In Compass Paste connection string:

mongodb+srv://rajeshpd339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/userdatabase

- Click Connect → check left sidebar for userdatabase.
- If missing, click Create Database → Name: userdatabase, Collection: users.



Complete code for connecting the MongoDB database

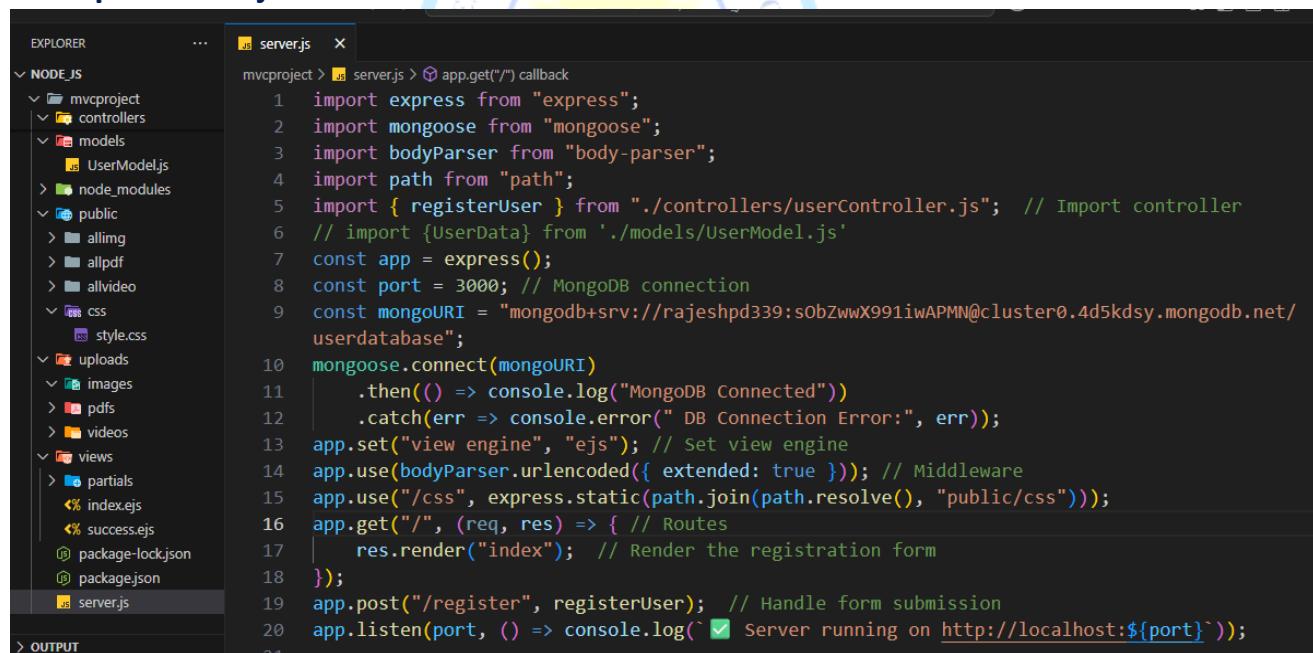


```

js server.js x
mvcproject > js server.js > ...
1 import express from "express";
2 import mongoose from "mongoose";
3 const myserver = express();
4 const port = 3000;
5 // MongoDB Connection
6 const myurls = "mongodb+srv://rajeshpd339:s0bZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/
7 userdatabase";
8 mongoose.connect(myurls, {
9   useNewUrlParser: true,
10  useUnifiedTopology: true
11  // In Mongoose 6+ version, both are optional
12 })
13 .then(() => console.log("MongoDB Connected"))
14 .catch(err => console.error("DB Connection Error:", err));
15 // Simple route
16 myserver.get("/", (req, res) => {
17   res.send("Hello, MongoDB is connected!");
18 })
19 myserver.listen(port, () => {
20   console.log(`Server is running on ${port}`);
21 });

```

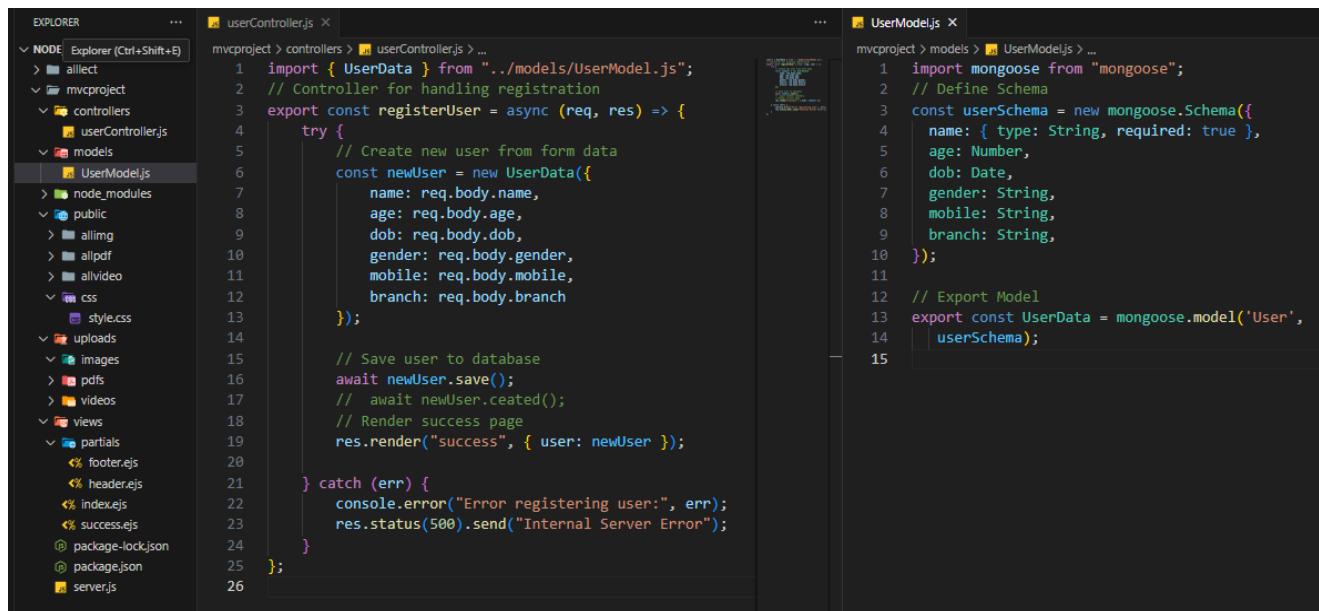
Example server.js



```

EXPLORER ... js server.js x
mvcproject > js server.js > app.get("/") callback
1 import express from "express";
2 import mongoose from "mongoose";
3 import bodyParser from "body-parser";
4 import path from "path";
5 import { registerUser } from "./controllers/userController.js"; // Import controller
6 // import {UserData} from './models/UserModel.js'
7 const app = express();
8 const port = 3000; // MongoDB connection
9 const mongoURI = "mongodb+srv://rajeshpd339:s0bZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/
10 userdatabase";
11 mongoose.connect(mongoURI)
12   .then(() => console.log("MongoDB Connected"))
13   .catch(err => console.error(" DB Connection Error:", err));
14 app.set("view engine", "ejs"); // Set view engine
15 app.use(bodyParser.urlencoded({ extended: true })); // Middleware
16 app.use("/css", express.static(path.join(path.resolve(), "public/css")));
17 app.get("/", (req, res) => { // Routes
18   res.render("index"); // Render the registration form
19 })
20 app.post("/register", registerUser); // Handle form submission
21 app.listen(port, () => console.log(` Server running on http://localhost:${port}`));

```



```

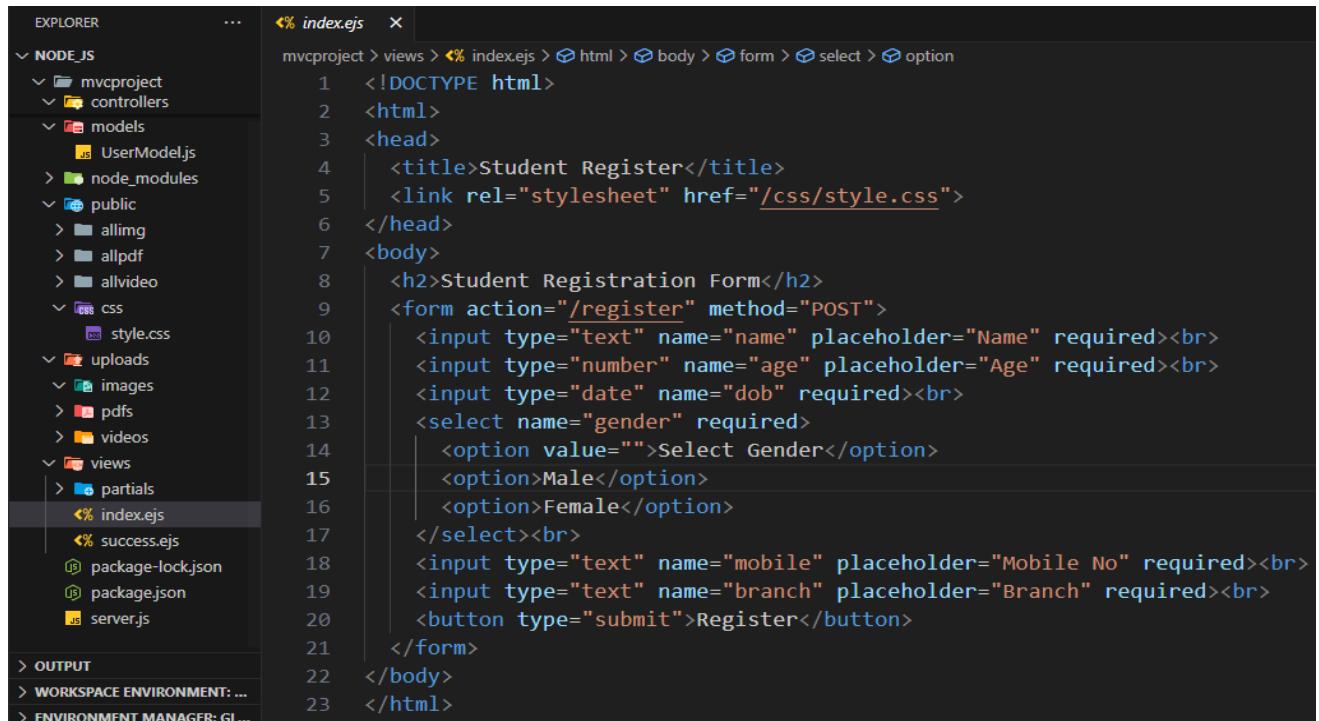
EXPLORER
NODE_Explorer (Ctrl+Shift+E)
> allct
mvproject
  controllers
    UserController.js
  models
    UserModel.js
  node_modules
  public
    allimg
    alppdf
    allvideo
  css
    style.css
  uploads
    images
    pdfs
    videos
  views
    partials
      footer.ejs
      header.ejs
      index.ejs
      success.ejs
    package-lock.json
    package.json
    server.js

userController.js
1 import { UserData } from "../models/UserModel.js";
2 // Controller for handling registration
3 export const registerUser = async (req, res) => {
4   try {
5     // Create new user from form data
6     const newUser = new UserData({
7       name: req.body.name,
8       age: req.body.age,
9       dob: req.body.dob,
10      gender: req.body.gender,
11      mobile: req.body.mobile,
12      branch: req.body.branch
13    });
14
15    // Save user to database
16    await newUser.save();
17    // await newUser.reated();
18    // Render success page
19    res.render("success", { user: newUser });
20
21  } catch (err) {
22    console.error("Error registering user:", err);
23    res.status(500).send("Internal Server Error");
24  }
25}
26

UserModel.js
1 import mongoose from "mongoose";
2 // Define Schema
3 const userSchema = new mongoose.Schema({
4   name: { type: String, required: true },
5   age: Number,
6   dob: Date,
7   gender: String,
8   mobile: String,
9   branch: String,
10 });
11
12 // Export Model
13 export const UserData = mongoose.model('User',
14   userSchema);
15

```

Html file



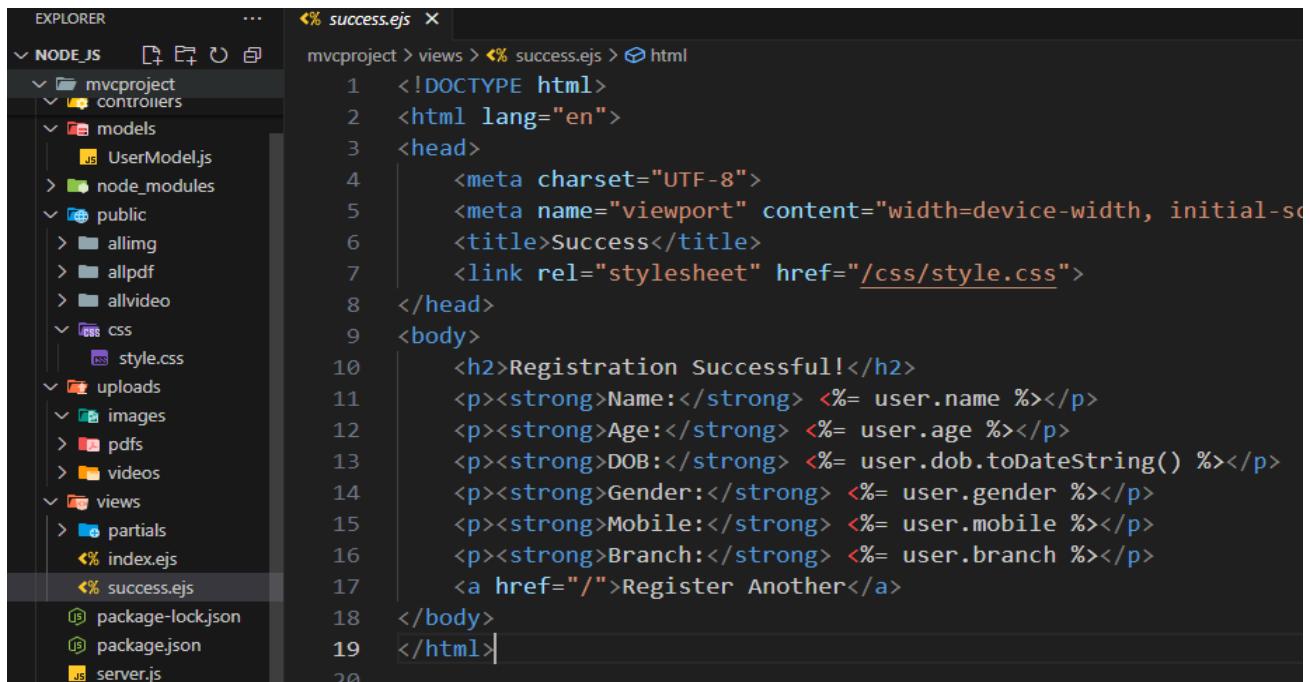
```

EXPLORER
NODE_JS
  mvcproject
    controllers
      models
        UserModel.js
      node_modules
    public
      allimg
      alppdf
      allvideo
    css
      style.css
    uploads
    images
    pdfs
    videos
    views
      partials
        index.ejs
        success.ejs
      package-lock.json
      package.json
      server.js

  OUTPUT
  WORKSPACE ENVIRONMENT: ...
  ENVIRONMENT MANAGER: GL...

index.ejs
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Student Register</title>
5   <link rel="stylesheet" href="/css/style.css">
6 </head>
7 <body>
8   <h2>Student Registration Form</h2>
9   <form action="/register" method="POST">
10    <input type="text" name="name" placeholder="Name" required><br>
11    <input type="number" name="age" placeholder="Age" required><br>
12    <input type="date" name="dob" required><br>
13    <select name="gender" required>
14      <option value="">Select Gender</option>
15      <option>Male</option>
16      <option>Female</option>
17    </select><br>
18    <input type="text" name="mobile" placeholder="Mobile No" required><br>
19    <input type="text" name="branch" placeholder="Branch" required><br>
20    <button type="submit">Register</button>
21  </form>
22 </body>
23 </html>

```



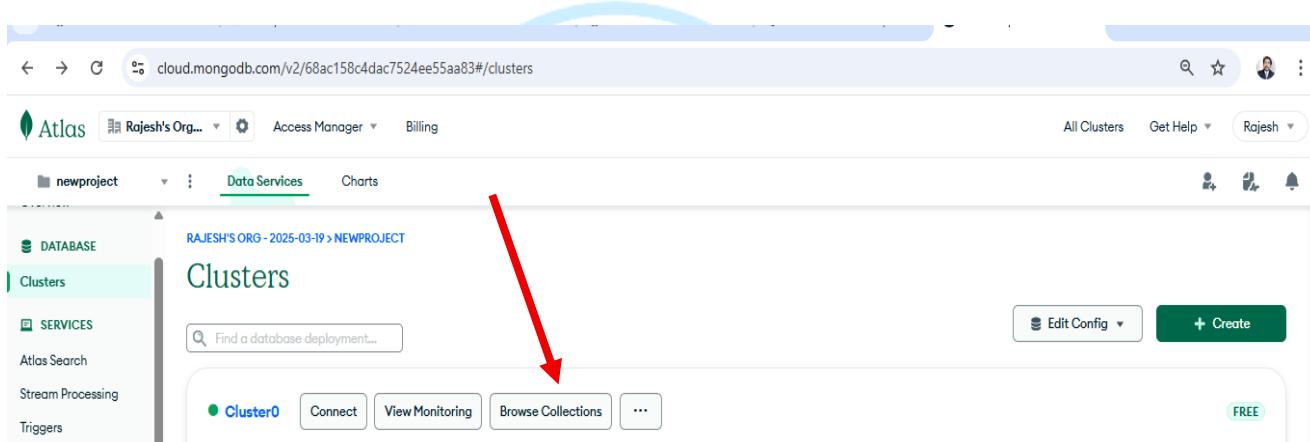
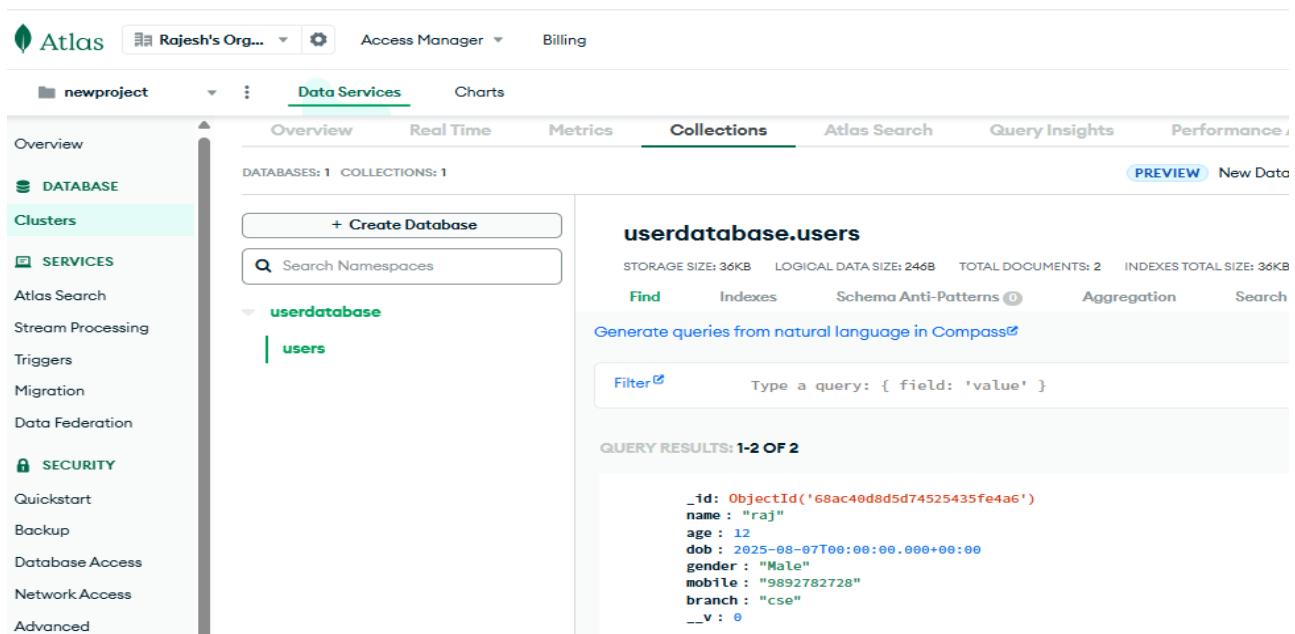
```

EXPLORER
  NODE_JS
    mvcproject
      controllers
        models
          UserModel.js
        node_modules
      public
        allimg
        allpdf
        allvideo
      css
        style.css
    uploads
    images
    pdfs
    videos
    views
      partials
        index.ejs
        success.ejs
      package-lock.json
      package.json
      server.js

EDITOR
  %% success.ejs x
  mvcproject > views > %% success.ejs > html
  1  <!DOCTYPE html>
  2  <html lang="en">
  3  <head>
  4    <meta charset="UTF-8">
  5    <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
  6    <title>Success</title>
  7    <link rel="stylesheet" href="/css/style.css">
  8  </head>
  9  <body>
 10    <h2>Registration Successful!</h2>
 11    <p><strong>Name:</strong> <%= user.name %></p>
 12    <p><strong>Age:</strong> <%= user.age %></p>
 13    <p><strong>DOB:</strong> <%= user.dob.toDateString() %></p>
 14    <p><strong>Gender:</strong> <%= user.gender %></p>
 15    <p><strong>Mobile:</strong> <%= user.mobile %></p>
 16    <p><strong>Branch:</strong> <%= user.branch %></p>
 17    <a href="/">Register Another</a>
 18  </body>
 19  </html>
 20

```

In mongo db click on Browser Collections

userdatabase.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 246B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search

Generate queries from natural language in Compass

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

QUERY RESULTS: 1-2 OF 2

```

_id: ObjectId('68ac40d8d5d74525435fe4a6')
name: "raj"
age: 12
dob: 2025-08-07T00:00:00.000+00:00
gender: "Male"
mobile: "9892782728"
branch: "cse"
__v: 0

```

How to add plain text, image, video& pdf from Front to Backend to Database

1) Install & Why

```
npm init -y
npm i express mongoose ejs multer body-parser
# optional
npm i -D nodemon
```

- **express** – web server & routing
- **mongoose** – MongoDB ODM (schemas/models)
- **ejs** – server-side templates (views)
- **multer** – parses multipart/form-data and saves uploaded files
- **body-parser** – parses URL-encoded form fields (text inputs)

2) Folder Structure

```
project/
  controllers/
    userController.js
  models/
    UserModel.js
  public/
    css/
      style.css
  uploads/  multer saves files here
  views/
    index.ejs
    success.ejs
  server.js
```

Add "type": "module" so import works: // package.json

```
{ "type": "module",
  "scripts": { "start": "node server.js", "dev": "nodemon server.js" } }
```

"form → multer → controller → MongoDB → success view"

Create the uploads/ folder (multer needs it).

3) Model (MongoDB) models/UserModel.js

```
 UserModel.js X
mvcproject_2 > models > UserModel.js > default
1  import mongoose from "mongoose";
2  const userSchema = new mongoose.Schema({
3    name: String,
4    age: Number,
5    dob: Date,
6    gender: String,
7    mobile: String,
8    branch: String,
9    image: String,      // filename for image
10   marksheets: String, // filename for pdf
11   video: String      // filename for video
12 });
13 const User = mongoose.model("User", userSchema);
14 export default User;
```

- Schema is a **class constructor** in Mongoose.
- new creates a **schema instance** from that class.
- The schema defines the **structure and rules** of documents in a collection.
- Without new, Mongoose still allows it (shortcut), but **best practice** is to use new for clarity and correctness.

```
login.js > routes > login.js > ...
import User from "../models/Usermodels.js";
export const registerUser=async (req,res)=>{
try{
const user=new User({
  name:req.body.name,
  age:req.body.age_1,
  dob:req.body.dob,
  gender:req.body.gender ,
  branch:req.body.branch,
  image:req.files?.image?.[0]?.filename,
  marksheets:req.files?.marksheets?.[0]?.filename ,
  video:req.files?.video?.[0]?.filename
})
await user.save()
res.render("success")
catch(error){console.log("form not submitted",error)}
}
```

- **import User** → brings in the Mongoose model for users.
- **registerUser function** → handles user registration.
- **req.body** → gets form data (name, age, etc.).
- **req.files** → gets uploaded files (image, marksheets, video).
- **new User({...})** → creates a new user document.
- **user.save()** → saves the document to MongoDB.
- **res.render("success", { user })** → shows success page with user data.
- **try...catch** → handles errors safely.



```

server.js
1 import express from "express";
2 import mongoose from "mongoose";
3 import bodyParser from "body-parser";
4 import path from "path";
5 import { fileURLToPath } from "url";
6 import multer from "multer";
7 import { registerUser } from "./controllers/userController.js";
8 // Create app
9 const app = express();
10 const port = 3000;
11 // __dirname setup for ES Modules
12 const __filename = fileURLToPath(import.meta.url);
13 const __dirname = path.dirname(__filename);
14 // MongoDB connection
15 const mongoURI = "mongodb+srv://rajeshpd339:s0bZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/userdatabase";
16 mongoose.connect(mongoURI)
17 .then(() => console.log("MongoDB Connected"))
18 .catch(err => console.error("DB Connection Error:", err));
19 // Middleware
20 app.set("view engine", "ejs");
21 app.set("views", path.join(__dirname, "views"));
22 app.use(bodyParser.urlencoded({ extended: true }));
23 app.use(express.static(path.join(__dirname, "public")));
24 // Configure Multer for file uploads

```

```

server.js
24 // Configure Multer for file uploads
25 const storage = multer.diskStorage({
26   destination: (req, file, cb) => {
27     cb(null, "uploads/"); // Save files in /uploads folder
28   },
29   filename: (req, file, cb) => {
30     cb(null, Date.now() + "-" + file.originalname);
31   },
32 });
33 const upload = multer({ storage: storage });
34 // Routes
35 app.get("/", (req, res) => {
36   res.render("index"); // Registration form
37 }); // Handle form submission (upload multiple files: image, marksheets, video)
38 app.post(
39   "/register",
40   upload.fields([
41     { name: "image", maxCount: 1 },
42     { name: "marksheets", maxCount: 1 },
43     { name: "video", maxCount: 1 },
44   ]),
45   registerUser
46 ); // Start server
47 app.listen(port, () => console.log(`Server running on http://localhost:${port}`));

```

- **express** → framework to build the web server. **mongoose** → connect and interact with MongoDB.
- **body-parser** → parse form data from requests (req.body).path & **fileURLToPath** → handle file paths and set __dirname in ES modules. **multer** → handle file uploads (image, pdf, video).
- **registerUser (controller)** → handles saving user data and files to DB.
- **const app = express()** → create an Express app instance.
- **app.set("view engine", "ejs")** → use EJS templates for rendering views.
- **app.set("views", path.join(__dirname, "views"))** → set the views folder path.
- **app.use(bodyParser.urlencoded({ extended: true }))** → allow form submissions to be parsed.
- **app.use(express.static(path.join(__dirname, "public")))** → serve static files (CSS, JS, images).
- **mongoose.connect(mongoURI)** → connect to MongoDB database.
- **multer.diskStorage({...})** → configure where and how files are saved.
- **upload.fields([...])** → allow uploading multiple files (image, marksheets, video).
- **app.get("/")** → route to display registration form.
- **app.post("/register", ...)** → route to handle form submission + file uploads.
- **app.listen(port, ...)** → start the server at http://localhost:3000.

Views(index.ejs)

```
mvcproject-1 > views > %% index.ejs > html > body
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Student Register</title>
5  <link rel="stylesheet" href="/css/style.css">
6  </head>
7  <body>
8  <h2>Student Registration Form</h2>
9  <form action="/register" method="POST">
10 <input type="text" name="name" placeholder="Name" required><br>
11 <input type="number" name="age" placeholder="Age" required><br>
12 <input type="date" name="dob" required><br>
13 <select name="gender" required>
14 <option value="">Select Gender</option>
15 <option>Male</option>
16 <option>Female</option>
17 </select><br>
18 <input type="text" name="mobile" placeholder="Mobile No" required>
19 <input type="text" name="branch" placeholder="Branch" required><br>
20 <button type="submit">Register</button>
21 </form>
22 </body>
23 </html>
```

Views(success.ejs)

```
mvcproject-1 > views > %% success.ejs > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="UTF-8">
5  <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  <title>Success</title>
7  <link rel="stylesheet" href="/css/style.css">
8  </head>
9  <body>
10 <h2>Registration Successful!</h2>
11 </body>
12 </html>
```

Student Registration Form

sangam kumar

20

Date of Birth:

09-08-2025

Gender:

Male

9892782728

aiml

Upload Profile Image:

rpsir.jpg

Upload Marksheets (PDF only):

oops.pdf

Upload Video: START VIDEO CLIP.mp4

Register

Cluster0

RAJESH'S ORO - 2025-03-19 > NEWPROJECT > DATABASES

Overview Real Time Metrics Collections Atlas Search Query Insights Performance Advisor

DATABASES: 1 COLLECTIONS: 1

+ Create Database

Search Namespaces

userdatabase

users

userdatabase.users

STORAGE SIZE: 34KB LOGICAL DATA SIZE: 1.0KB TOTAL DOCUMENTS: 5 INDEXES TOTAL SIZE: 34KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass!

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

...
`_id: ObjectId('68ad5f55ed9662d86895be27')
name : "sangam kumar"
age : 20
dob : 2025-08-09T00:00:00.000+00:00
gender : "Male"
mobile : "9892782728"
branch : "aiml"
image : "1756192596999-Screenshot_2025-03-11_174821-removebg-preview.png"
marksheets : "1756192597009-oops.pdf"
video : "1756192597084-START VIDEO CLIP.mp4"
... : 0`

...
`_id: ObjectId('68ad6772361119f09c07af90')
name : "sangam kumar"`

%% success.ejs

```
mvcproject_2 > views > %% success.ejs > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="UTF-8">
5  <title>Registration Success</title>
6  <link rel="stylesheet" href="/css/style.css">
7  </head>
8  <body>
9  <h2>Registration Successful!</h2>
10 <p><strong>Name:</strong> <%= user.name %></p>
11 <p><strong>Age:</strong> <%= user.age %></p>
12 <p><strong>DOB:</strong> <%= user.dob ? new Date(user.dob) : '' %></p>
13 <p><strong>Gender:</strong> <%= user.gender %></p>
14 <p><strong>Mobile:</strong> <%= user.mobile %></p>
15 <p><strong>Branch:</strong> <%= user.branch %></p>
16 </body>
17 </html>
```



7. Create View views/index.ejs

```
<!DOCTYPE html>
<html>
<head>
<title>Student Register</title>
<link rel="stylesheet" href="/css/style.css">
</head>
<body>
<h2>Student Registration Form</h2>
<form action="/register" method="POST" enctype="multipart/form-data">
<input type="text" name="name" placeholder="Name" required><br>
<input type="number" name="age" placeholder="Age" required><br>
<input type="date" name="dob" required><br>
<select name="gender" required>
<option value="">Select Gender</option>
<option>Male</option>
<option>Female</option>
</select><br>
<input type="text" name="mobile" placeholder="Mobile No" required><br>
<input type="text" name="branch" placeholder="Branch" required><br>
<label>Upload Profile Image:</label><br>
<input type="file" name="image" accept="image/*" required><br>
<label>Upload Marksheets (PDF):</label><br>
<input type="file" name="marksheets" accept=".pdf" required><br>
<label>Upload Short Video:</label><br>
<input type="file" name="video" accept="video/*" required><br>
<button type="submit">Register</button>
</form>
</body>
</html>
```

views/success.ejs

```
<h2>Registration Successful!</h2>
<p>Name: <%= user.name %></p>
<p>Branch: <%= user.branch %></p>
<p>Image: <%= user.image %></p>
<p>Marksheets: <%= user.marksheets %></p>
<p>Video: <%= user.video %></p>
<a href="/">Register Another</a>
```

8. Static CSS public/css/style.css

```
body { font-family: sans-serif; padding: 20px; }
input, select {
  display: block;
  margin: 10px 0;
  padding: 8px;
  width: 300px;
}
button {
  padding: 10px 20px;}
```

9. Main Server File server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const path = require('path');
const multer = require('multer');
const userController = require('./controllers/userController');
const app = express();
// MongoDB Connection
mongoose.connect('your_mongo_uri_here', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log('MongoDB connected'))
  .catch(err => console.error(err));
// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static('public'));
app.use('/uploads', express.static('uploads'));
app.set('view engine', 'ejs');
// Multer Config
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    if (file.fieldname === 'image') cb(null, 'uploads/images');
    else if (file.fieldname === 'marksheets') cb(null, 'uploads/pdfs');
    else if (file.fieldname === 'video') cb(null, 'uploads/videos');
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + '-' + file.originalname);
  }
});
const upload = multer({ storage: storage });
const cpUpload = upload.fields([
  { name: 'image', maxCount: 1 },
  { name: 'marksheets', maxCount: 1 },
  { name: 'video', maxCount: 1 }]);
// Routes
app.get('/', (req, res) => res.render('index'));
app.post('/register', cpUpload, userController.registerUser); // Start Server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

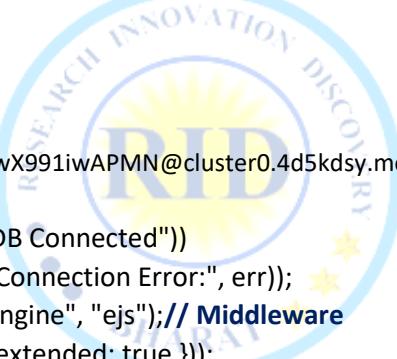
Controls.js

```
import { UserData } from "../models/UserModel.js";
import path from "path"; // Controller for handling registration
export const registerUser = async (req, res) => {
  try {
    // Save uploaded files paths
    const imagePath = req.files.image ? `/uploads/images/${req.files.image[0].filename}` : "";
    const marksheetsPath = req.files.marksheets ? `/uploads/pdfs/${req.files.marksheets[0].filename}` : "";
    const videoPath = req.files.video ? `/uploads/videos/${req.files.video[0].filename}` : "";
    // Create new user document
    const newUser = new UserData({
      name: req.body.name,
```

```
age: req.body.age,
dob: req.body.dob,
gender: req.body.gender,
mobile: req.body.mobile,
branch: req.body.branch,
image: imagePath,
marksheet: marksheetPath,
video: videoPath});
await newUser.save(); // Render success page
res.render("success", { user: newUser });
} catch (err) {
  console.error("Error registering user:", err);
  res.status(500).send("Internal Server Error");}}
```

Server.js

```
import express from "express";
import mongoose from "mongoose";
import bodyParser from "body-parser";
import multer from "multer";
import path from "path";
import { registerUser } from "./controllers/userController.js";
const app = express();
const port = 3000;
// MongoDB connection
const mongoURI =
"mongodb+srv://rajeshpd339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/userdatabase";
mongoose.connect(mongoURI)
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.error("DB Connection Error:", err));
// Set view engineapp.set("view engine", "ejs");// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use("/css", express.static(path.join(path.resolve(), "public/css")));
app.use("/uploads", express.static(path.join(path.resolve(), "uploads")));
// Multer Storage Configuration
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    if(file.fieldname === "image") cb(null, "uploads/images");
    else if(file.fieldname === "marksheets") cb(null, "uploads/pdfs");
    else if(file.fieldname === "video") cb(null, "uploads/videos");
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + "-" + file.originalname);}}
```



```
  },
  const upload = multer({ storage });
// Routesapp.get("/", (req, res) => res.render("index"));
// Form submission route
app.post("/register", upload.fields([
  { name: "image", maxCount: 1 },
  { name: "marksheets", maxCount: 1 },
  { name: "video", maxCount: 1 }
]), registerUser);
app.listen(port, () => console.log(`Server running on ${port}`));
```

Front End +Backend+ Database Connection in Node js/Express

Home Gallery Contact Service News [Login](#) [Signup](#)

Signup Form

Name:

Email:

Mobile No:

Password:

Confirm Password:

have an account? [Login](#)

© 2025 My Website. All rights reserved.

```
project/
  -- models/
    -- signupmodel.js

  -- controllers/
    -- signupc.js

  -- routes/
    -- signup.js

  -- views/
    -- index.ejs
    -- Signup.ejs
    -- Login.ejs
    -- partials/
      -- header.ejs
      -- footer.ejs

  -- public/
    -- css/
      -- style.css

  -- server.js
```



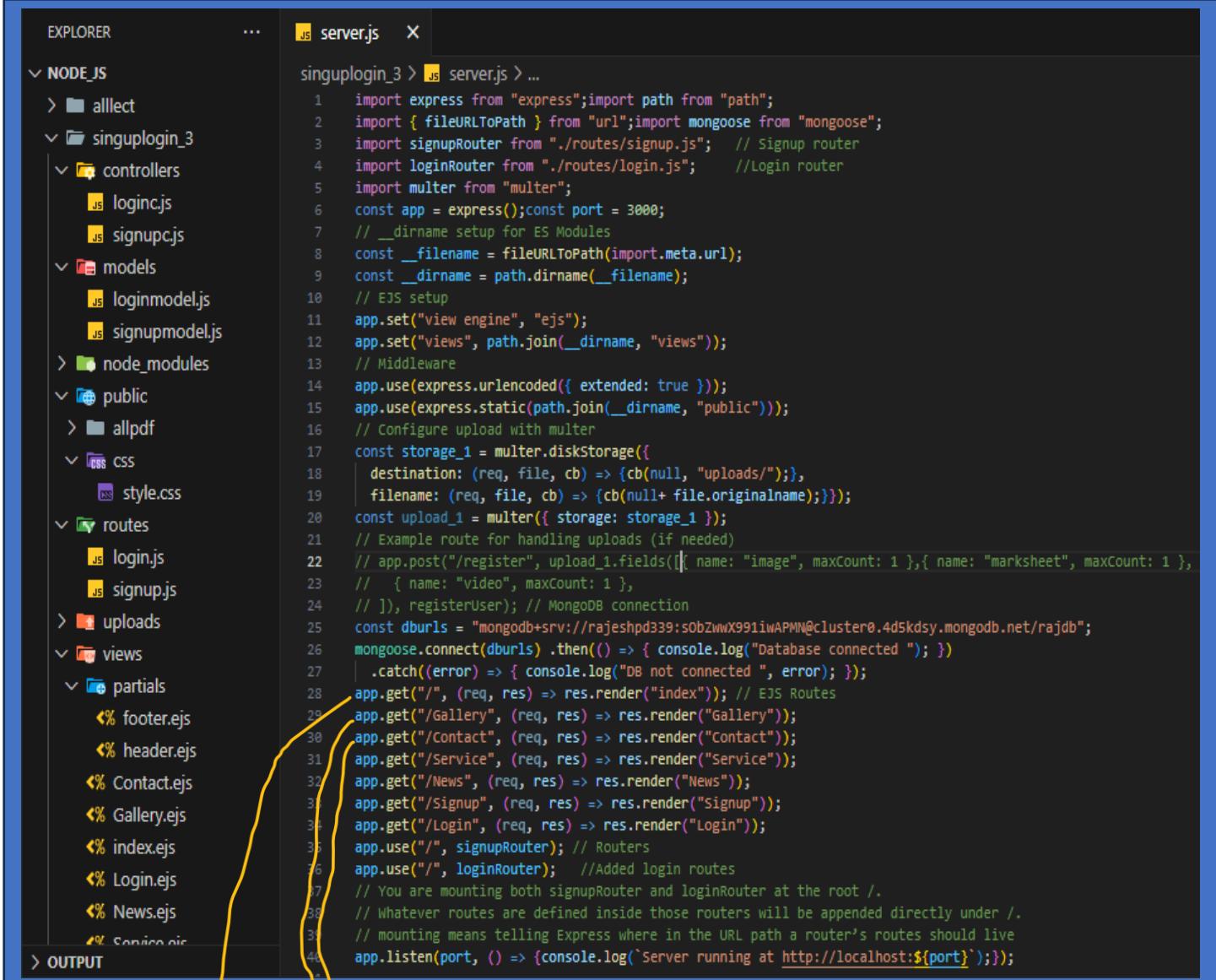
```
  -- singuplogin_3
  -- controllers
    -- loginc.js
    -- signupc.js
  -- models
    -- loginmodel.js
    -- signupmodel.js
  -- node_modules
  -- public
    -- allpdf
    -- css
      -- style.css
  -- routes
    -- login.js
    -- signup.js
  -- uploads
  -- views
    -- partials
      -- footer.ejs
      -- header.ejs
      -- Contact.ejs
      -- Gallery.ejs
      -- index.ejs
      -- Login.ejs
      -- News.ejs
      -- Service.ejs
      -- Signup.ejs
    -- package-lock.json
    -- package.json
  -- server.js
```

Install → `npm install express ejs mysql2 mongoose multer body-parser path`

Package	Purpose
• express	→ Backend framework for routes and server.
• ejs	→ Template engine for frontend rendering.
• mysql2	→ Connect & query MySQL database.
• Mongoose	→ Connect & manage MongoDB database.
• Multer	→ Handle file uploads (images, PDFs, videos).
• body-parser	→ Parse POST form data (req.body).



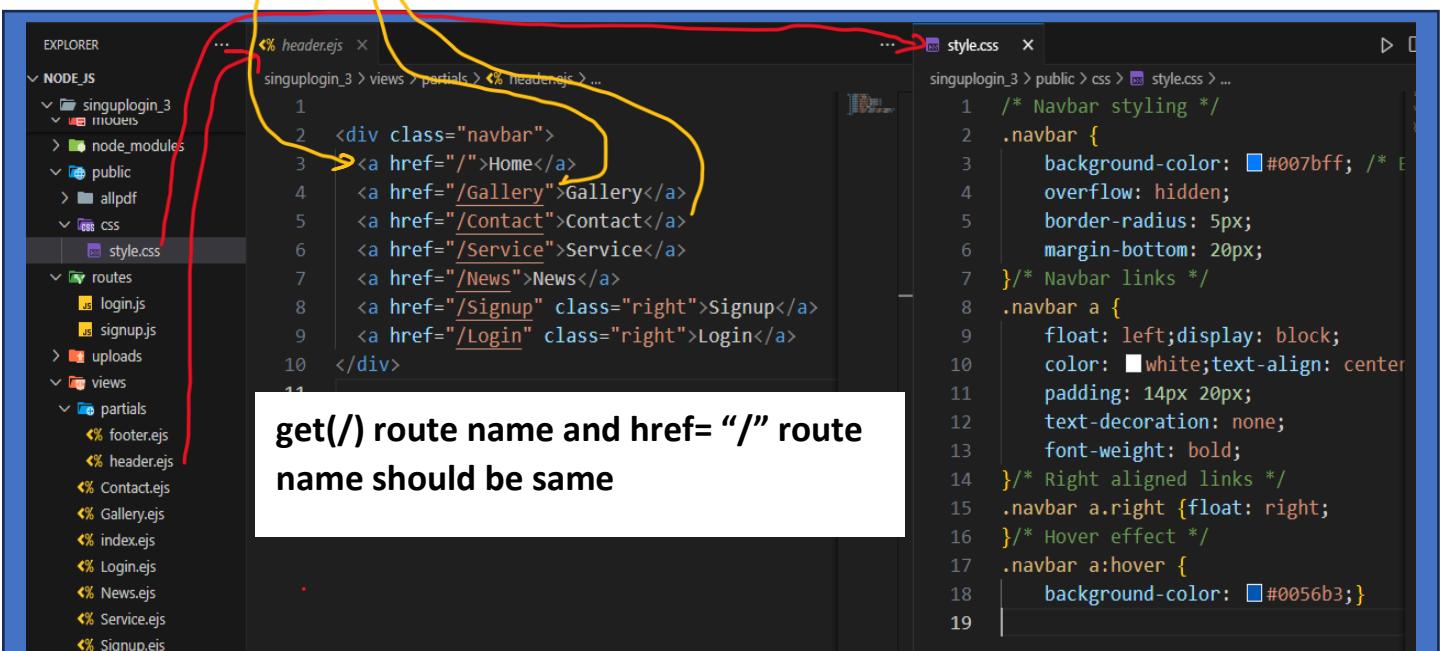
Server.js



```

singuplogin_3 > server.js > ...
1  import express from "express";import path from "path";
2  import { fileURLToPath } from "url";import mongoose from "mongoose";
3  import signupRouter from "./routes/signup.js"; // Signup router
4  import loginRouter from "./routes/login.js"; // Login router
5  import multer from "multer";
6  const app = express();const port = 3000;
7  // __dirname setup for ES Modules
8  const __filename = fileURLToPath(import.meta.url);
9  const __dirname = path.dirname(__filename);
10 // EJS setup
11 app.set("view engine", "ejs");
12 app.set("views", path.join(__dirname, "views"));
13 // Middleware
14 app.use(express.urlencoded({ extended: true }));
15 app.use(express.static(path.join(__dirname, "public")));
16 // Configure upload with multer
17 const storage_1 = multer.diskStorage({
18   destination: (req, file, cb) => {cb(null, "uploads/")},
19   filename: (req, file, cb) => {cb(null+ file.originalname);});
20 const upload_1 = multer({ storage: storage_1 });
21 // Example route for handling uploads (if needed)
22 // app.post("/register", upload_1.fields([
23 //   { name: "image", maxCount: 1 },
24 //   { name: "video", maxCount: 1 },
25 // ]), registerUser); // MongoDB connection
26 const dburls = "mongodb+srv://rajeshpd339:s0bZwwX991iWAPMN@cluster0.4d5kdsy.mongodb.net/rajdb";
27 mongoose.connect(dburls).then(() => { console.log("Database connected"); })
28 .catch((error) => { console.log("DB not connected", error); });
29 app.get("/", (req, res) => res.render("index")); // EJS Routes
30 app.get("/Gallery", (req, res) => res.render("Gallery"));
31 app.get("/Contact", (req, res) => res.render("Contact"));
32 app.get("/Service", (req, res) => res.render("Service"));
33 app.get("/News", (req, res) => res.render("News"));
34 app.get("/Signup", (req, res) => res.render("Signup"));
35 app.get("/Login", (req, res) => res.render("Login"));
36 app.use("/", signupRouter); // Routers
37 app.use("/", loginRouter); // Added login routes
38 // You are mounting both signupRouter and loginRouter at the root /.
39 // Whatever routes are defined inside those routers will be appended directly under /.
40 // mounting means telling Express where in the URL path a router's routes should live
41 app.listen(port, () => {console.log(`Server running at http://localhost:\${port}`);});

```



get() route name and href= "/" route name should be same

```

singuplogin_3 > views > partials > % header.ejs > ...
1 <div class="navbar">
2   <a href="/">Home</a>
3   <a href="/Gallery">Gallery</a>
4   <a href="/Contact">Contact</a>
5   <a href="/Service">Service</a>
6   <a href="/News">News</a>
7   <a href="/Signup" class="right">Signup</a>
8   <a href="/Login" class="right">Login</a>
9 </div>
10
11
12
13
14
15
16
17
18
19

```

```

singuplogin_3 > public > css > style.css > ...
1 /* Navbar styling */
2 .navbar {
3   background-color: #007bff; /* E
4   overflow: hidden;
5   border-radius: 5px;
6   margin-bottom: 20px;
7 }/* Navbar links */
8 .navbar a {
9   float: left;display: block;
10  color: white;text-align: center;
11  padding: 14px 20px;
12  text-decoration: none;
13  font-weight: bold;
14 }/* Right aligned links */
15 .navbar a.right {float: right;
16 }/* Hover effect */
17 .navbar a:hover {
18   background-color: #0056b3;}
```

views (Login.ejs and Signup.ejs)

```
singuplogin_3 > views > Login.ejs > html
1 <!-- views/Login.ejs -->
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <title>Login</title>
6   <link rel="stylesheet" href="css/style.css">
7 </head>
8 <body>
9   <%- include('partials/header') %>
10  <h2 id="dd1">Login Form</h2>
11  <form action="/req-login" method="post">
12    Email: <input type="email" name="email" required><br><br>
13    Password: <input type="password" name="password"><br><br>
14    <button type="submit">Login</button>
15  </form>
16  <p>Don't have an account? <a href="/Signup">Sign up here</a>
17  <%- include('partials/footer') %>
18 </body>
19 </html>
20
```

```
%% Signup.ejs x
singuplogin_3 > views > Signup.ejs ...
1 <!-- views/Signup.ejs -->
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <title>Signup</title>
6   <link rel="stylesheet" href="css/style.css">
7 </head>
8 <body>
9   <%- include('partials/header') %>
10  <h2 id="dd1">Signup Form</h2>
11  <form action="/req-signup" method="post">
12    Name:<input type="text" name="name" required><br><br>
13    Email:<input type="email" name="email" required><br><br>
14    Mobile No: <input type="text" name="mobile" required><br><br>
15    Password:<input type="password" name="password" required><br><br>
16    Confirm Password: <input type="password" name="confirm" required><br><br>
17    <button type="submit">Sign Up</button>
18  <p>have an account? <a href="/Login">Login</a></p>
19 </form>
20 <%- include('partials/footer') %>
21 </body></html>
```

Signupmodel.js(models) and singupc.js(controllers)

```
EXPLORER ... singupmodel.js x
singuplogin_3 > models > singupmodel.js ...
1 import mongoose from "mongoose";
2 // Define schema
3 const signupSchema = new mongoose.Schema({
4   name: String,
5   email: String,
6   mobile: String,
7   password: String,
8   confirm: String,
9   createdAt: { type: Date, default: Date.now }
10  });
11 // Create collection (model)
12 const SignupUser = mongoose.model
13 ("SignupUser", signupSchema);
14 export default SignupUser;
```

```
signupc.js x
singuplogin_3 > controllers > signupc.js ...
1 import SignupUser from "../models/signupmodel.js"; // ✅ spelling correction
2 export const registerUser = async (req, res) => {
3   try {
4     const { name, email, mobile, password, confirm } = req.body;
5     if (password !== confirm) {
6       return res.status(400).send("Passwords do not match!");
7     }
8     const existingUser = await SignupUser.findOne({ email });
9     if (existingUser) {
10       return res.status(400).send("Email already registered!");
11     }
12     const newUser = new SignupUser({ name, email, mobile, password });
13     await newUser.save();
14     res.status(201).send("Signup successful! Please login.");
15   } catch (error) {
16     console.error("Error during signup:", error);
17     res.status(500).send("Internal Server Error");
18   }
19 }
```

```
loginc.js x
singuplogin_3 > controllers > loginc.js > loginUser
1 import SignupUser from "../models/signupmodel.js";
2 export const loginUser = async (req, res) => {
3   try {
4     const { email, password } = req.body;
5     // Step 1: Find user by email
6     const user = await SignupUser.findOne({ email });
7     if (!user) {return res.send("User not found.");}
8     // Step 2: Compare plain text passwords
9     if (user.password !== password) {return res.send("Invalid pwd.")}
10    // Step 3: If everything matches → redirect
11    console.log("Login Successful");
12    res.redirect("/");
13  } catch (error) {
14    console.error("Login Error:", error);
15    res.send("Something went wrong while logging in.");
16  }
17}
```

```
login.js x
singuplogin_3 > routes > login.js ...
1 import express from "express";
2 import { loginUser } from "../controllers/loginc.js";
3
4 const router = express.Router();
5 // POST route for login form
6 router.post("/req-login", loginUser);
7
8 export default router;
```

Route (Login.js and singup.js)

```
logins x
singuplogin_3 > routes > logins.js ...
1 import express from "express";
2 import { loginUser } from "../controllers/loginc.js";
3
4 const router = express.Router();
5 // POST route for login form
6 router.post("/req-login", loginUser);
7
8 export default router;
```

```
signup.js x
singuplogin_3 > routes > signup.js ...
1 import express from "express";
2 import { registerUser } from "../controllers/signupc.js"; // spelling correction
3 const router = express.Router();
4 // POST request for signup
5 router.post("/req-signup", registerUser);
6 export default router;
7 // import signupRouter from "./routes/signup.js";
8 // import loginRouter from "./routes/login.js";
9 // Now signupRouter is just the exported router object from signup.js
10 // Same with loginRouter.
```



How you can fix your signup and login logic using bcrypt

1. Install bcrypt → npm install bcrypt 2. Update your signupmodel.js

```
singuplogin_3 > models > signupmodel.js > signupSchema.pre("save") callback
1  import mongoose from "mongoose";
2  import bcrypt from "bcrypt";
3  // Define schema
4  const signupSchema = new mongoose.Schema({
5      name: String,
6      email: { type: String, unique: true },
7      mobile: String,
8      password: String,
9      confirm: String,
10     createdAt: { type: Date, default: Date.now }
11 });
12 // Middleware → Hash password before saving
13 signupSchema.pre("save", async function (next) {
14     if (!this.isModified("password")) return next();
15     try {
16         const salt = await bcrypt.genSalt(10); // generate salt
17         this.password = await bcrypt.hash(this.password, salt); // hash password
18         this.confirm = undefined; // remove confirm field (not needed in DB)
19         next();
20     } catch (err) {
21         next(err);
22     }
23 });
24 // Create collection (model)
25 const SignupUser = mongoose.model("SignupUser", signupSchema);
26 export default SignupUser;
```

1. Update your loginUser Controller

```
signupmodel.js  login.js  x
singuplogin_3 > controllers > login.js > ...
1  import SignupUser from "../models/signupmodel.js";
2  import bcrypt from "bcrypt";
3  export const loginUser = async (req, res) => {
4      try {
5          const { email, password } = req.body;
6          // Step 1: Find user by email
7          const user = await SignupUser.findOne({ email });
8          if (!user) {
9              return res.send("User not found.");
10         }
11         // Step 2: Compare hashed password
12         const isMatch = await bcrypt.compare(password, user.password);
13         if (!isMatch) {
14             return res.send("Invalid password.");
15         }
16         // Step 3: If everything matches → login successful
17         console.log("Login Successful");
18         res.redirect("/"); // redirect to homepage
19     } catch (error) {
20         console.error("Login Error:", error);
21         res.send("Something went wrong while logging in.");
22     }
23 };
```



How you can fix your **signup** and **login** logic using **bcrypt**:

1. Install bcrypt → npm install bcrypt

2. Update your signupmodel.js

Example:

```
import mongoose from "mongoose";
import bcrypt from "bcrypt";
```

// Define schema

```
const signupSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  mobile: String,
  password: String,
  confirm: String,
  createdAt: { type: Date, default: Date.now }});
```

// Middleware → Hash password before saving

```
signupSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next();
  try {
    const salt = await bcrypt.genSalt(10); // generate salt
    this.password = await bcrypt.hash(this.password, salt); // hash password
    this.confirm = undefined; // remove confirm field (not needed in DB)
    next();
  } catch (err) {
    next(err);
  }
});
```

// Create collection (model)

```
const SignupUser = mongoose.model("SignupUser", signupSchema);
export default SignupUser;
```

3. Update your loginUser Controller

```
import SignupUser from "../models/signupmodel.js";
import bcrypt from "bcrypt";
export const loginUser = async (req, res) => {
  try {
    const { email, password } = req.body;
    // Step 1: Find user by email
    const user = await SignupUser.findOne({ email });
    if (!user) { return res.send("User not found."); }
    // Step 2: Compare hashed password
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) { return res.send("Invalid password."); }
    // Step 3: If everything matches → login successful
    console.log("Login Successful");
    res.redirect("/"); // redirect to homepage
  } catch (error) {
    console.error("Login Error:", error);
    res.send("Something went wrong while logging in.");
  }
};
```

How to connect a Node.js/Express app with local MongoDB

Step 1: Install MongoDB locally

- **Windows:** Download from [MongoDB Community](#) and install.
- **Mac:** brew install mongodb-community
- **Linux:** sudo apt install mongodb (or use official docs).

Step 2: Start MongoDB server

```
# Windows (from Command Prompt or Services)  
mongod  
# Mac/Linux → sudo systemctl start mongod  
# or  
mongod  
MongoDB default connection: mongodb://127.0.0.1:27017
```

Step 3: Install required Node.js packages

```
npm install express mongoose
```

- express → web server
- mongoose → MongoDB ODM

Step 4: Import packages and connect

```
import express from "express";  
import mongoose from "mongoose";  
const app = express();  
const port = 3000;  
// Local MongoDB URL  
const dbURL = "mongodb://127.0.0.1:27017/databaseName";  
// Connect to MongoDB  
mongoose.connect(dbURL, { useNewUrlParser: true, useUnifiedTopology: true })  
.then(() => console.log("MongoDB connected"))  
.catch(err => console.error("MongoDB connection error:", err));  
// Start Express server  
app.listen(port, () => console.log(`Server running at http://localhost:${port}`));
```

Step 5: Optional - Define a Mongoose Schema & Model

```
import mongoose from "mongoose";  
const userSchema = new mongoose.Schema({  
  name: String,  
  email: String,  
  password: String  
});  
const User = mongoose.model("User", userSchema);  
export default User;
```

Step 6: Use the model in routes

Example: Create new user

```
app.post("/signup", async (req, res) => {  
  const user = new User(req.body);  
  await user.save();  
  res.send("User created");  
});
```

Summary

1. Install MongoDB locally.
2. Start MongoDB server.
3. Install mongoose in Node.js.
4. Connect using
mongoose.connect("mongodb://127.0.0.1:27017/dbname").
5. Define schema & model.

Use model to create/read/update/delete documents.



MongoDB local connection guide with Node.js

Folder Structure

```
myapp/
  ├── server.js      # Main Express server
  ├── models/
  |   └── user.js    # Mongoose schema & model
  ├── routes/
  |   └── auth.js    # Signup/Login routes
  ├── package.json
  └── node_modules/
```

Step 1: Install MongoDB locally

- **Windows:** Download from [MongoDB Community](#) and install.
- **Mac:** brew install mongodb-community
- **Linux:** sudo apt install mongodb

Step 2: Start MongoDB server

```
# Windows → mongod
# Mac/Linux → sudo systemctl start mongod
# or mongod → MongoDB default connection: mongodb://127.0.0.1:27017
```

Step 3: Install required Node.js packages

```
npm install express mongoose
```

express → web server mongoose → MongoDB ODM Optional dev tool: npm install --save-dev nodemon

Step 4: Create server.js

```
import express from "express";
import mongoose from "mongoose";
import authRoutes from "./routes/auth.js";
const app = express();
const port = 3000;
// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
// Local MongoDB URL
const dbURL = "mongodb://127.0.0.1:27017/mydatabase";
// Connect to MongoDB
mongoose.connect(dbURL, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.error("MongoDB connection error:", err));
// Routes
app.use("/", authRoutes);
app.listen(port, () => console.log(`Server running at http://localhost:${port}`));
```

Step 5: Create models/user.js

```
import mongoose from "mongoose";
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String
});
const User = mongoose.model("User", userSchema);
export default User;
```

Step 6: Create routes/auth.js

```
import express from "express";
import User from "../models/user.js";
const router = express.Router();

// Signup route
router.post("/signup", async (req, res) => {
  try {
    const user = new User(req.body);
    await user.save();
    res.send("User created");
  } catch (err) {
    res.status(500).send("Error: " + err.message);
  }
});

// Login route (simple example)
router.post("/login", async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
  if (!user) return res.status(404).send("User not found");
  if (user.password !== password) return res.status(401).send("Invalid password");
  res.send(`Welcome ${user.name}`);
});

export default router;
```

How to run

nodemon server.js

- Signup: POST /signup with JSON { "name": "John", "email": "john@example.com", "password": "1234" }
- Login: POST /login with JSON { "email": "john@example.com", "password": "1234" }



MySQL DB connection with Node JS/Express JS

Node.js + Express + MySQL project setup

Step-1: Install required packages

- npm init -y
- npm install express ejs body-parser mysql2 bcrypt

Step-2: MySQL table

- Login to MySQL and create a database + table:

```
CREATE DATABASE testdb;  
USE testdb;  
CREATE TABLE users (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100),  
    email VARCHAR(100) UNIQUE,  
    password VARCHAR(255)  
);
```

Project structure + working with MySQL

Step-3: Project Structure

```
myapp/  
    ├── server.js  
    ├── package.json  
    ├── views/  
    |    ├── signup.ejs  
    |    └── login.ejs  
    ├── node_modules/  
    └── public/    (optional: CSS, images, JS)
```

server.js

```
import express from "express";  
import bodyParser from "body-parser";  
import mysql from "mysql2";  
import bcrypt from "bcrypt";  
import path from "path";  
import { fileURLToPath } from "url";  
const app = express();  
const port = 3000;  
// Setup __dirname for ES modules  
const __filename = fileURLToPath(import.meta.url);  
const __dirname = path.dirname(__filename);  
// Middleware  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: true }));  
app.use(express.static(path.join(__dirname, "public")));  
// Set EJS as view engine  
app.set("view engine", "ejs");  
app.set("views", path.join(__dirname, "views"));  
  
// DB Connection
```



How to Create a Database

- **Syntax:** CREATE DATABASE database_name;
- **Example:** mysql> CREATE DATABASE riddb;

How to Show All Databases

- **Syntax:** SHOW databases;
- **Example:** mysql> SHOW databases

How to Change a Database

- **Syntax:** USE database_name;
- **Example:** mysql> USE riddb;

How to Check Current DB

- **Syntax:** SELECT DATABASE();
- **Example:** mysql> SELECT DATABASE();

How to Delete database

- **Syntax:** DROP DATABASE database_name;
- **Example:** mysql> DROP DATABASE riddb;

How to switch from one database to another,

- **Syntax:** USE new_database_name;
- **Example:** mysql> USE riddb2;

How to Check if a Database Exist

- **Syntax:** SHOW DATABASES LIKE 'database_name';
- **Example:** mysql> SHOW DATABASES LIKE 'rid_db2';

How to Show Database Information (Schema)

- **Syntax:** SHOW TABLE STATUS FROM database_name;
- **Example:** mysql> SHOW TABLE STATUS FROM rid_db;

How to Show Database Users

- **Ex:** mysql> SELECT User, Host FROM mysql.user;

Table query in MySQL

How to create tables

```
CREATE TABLE table_name( column1 datatype constraints)
```

How to show all tables :

- SHOW TABLES ;

How to describe the table structure

- DESCRIBE table_name ;

How to delete the table

- DESCRIBE table_name;

How to show the all data from a table

- SELECT * FROM table_name;



```
const db = mysql.createConnection({
  host: "localhost",
  user: "root",      // your mysql username
  password: "raj12345", // your mysql password
  database: "testdb"
});
db.connect((err) => {
  if (err) { console.error("DB connection failed:", err);
  } else { console.log("MySQL Connected!"); } });
// ----- Routes -----
// Show signup form
app.get("/signup", (req, res) => {
  res.render("signup");
});
// Handle signup form
app.post("/signup", async (req, res) => {
  const { name, email, password } = req.body;
  const hashed = await bcrypt.hash(password, 10);
  const sql = "INSERT INTO users (name, email, password) VALUES (?, ?, ?)";
  db.query(sql, [name, email, hashed], (err) => {
    if (err) return res.status(500).send("Error: " + err.message);
    res.send("Signup successful!");
  });
});
// Show login form
app.get("/login", (req, res) => {
  res.render("login");
});
// Handle login form
app.post("/login", (req, res) => {
  const { email, password } = req.body;
  const sql = "SELECT * FROM users WHERE email = ?";
  db.query(sql, [email], async (err, results) => {
    if (err) return res.status(500).send("Error: " + err.message);
    if (results.length === 0) return res.status(404).send("User not found");
    const user = results[0];
    const match = await bcrypt.compare(password, user.password);

    if (!match) return res.status(401).send(" Invalid password");
    res.send(`Welcome ${user.name}, Login successful!`);
  });
});
// Start server app.listen(port, () => { console.log(`Server running at http://localhost:\${port}`); });
views/signup.ejs
<!DOCTYPE html>
<html>
<head>
  <title>Signup</title>
</head>
<body>
```



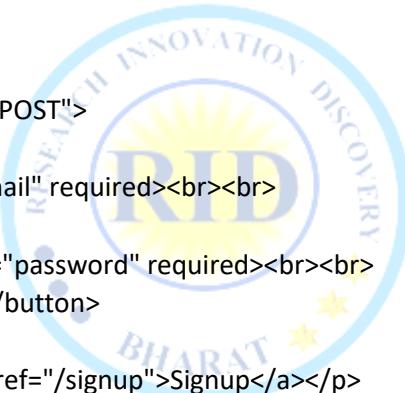
```
<h1>User Signup</h1>
<form action="/signup" method="POST">
  <label>Name:</label>
  <input type="text" name="name" required><br><br>
  <label>Email:</label>
  <input type="email" name="email" required><br><br>
  <label>Password:</label>
  <input type="password" name="password" required><br><br>
  <button type="submit">Signup</button>
</form>
<p>Already have an account? <a href="/login">Login</a></p>
</body>
</html>
```

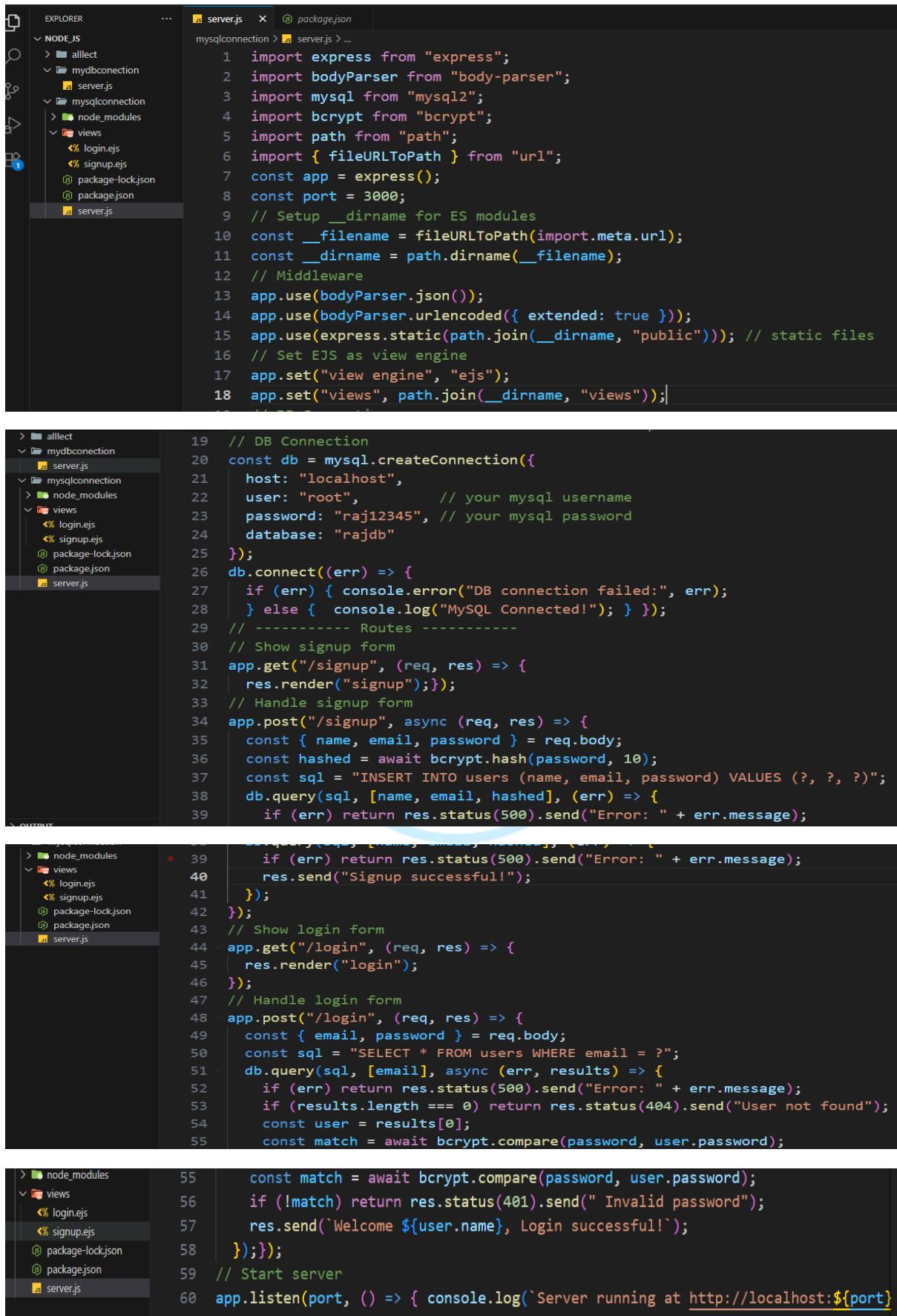
views/login.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
</head>
<body>
  <h1>User Login</h1>
  <form action="/login" method="POST">
    <label>Email:</label>
    <input type="email" name="email" required><br><br>
    <label>Password:</label>
    <input type="password" name="password" required><br><br>
    <button type="submit">Login</button>
  </form>
  <p>Don't have an account? <a href="/signup">Signup</a></p>
</body>
</html>
```

Now you can visit:

- <http://localhost:3000/signup> → register user
- <http://localhost:3000/login> → login user





```

EXPLORER
NODEJS
  > allect
  > mydbconnection
  > mysqlconnection
    > node_modules
    > views
      <% login.ejs
      <% signup.ejs
      package-lock.json
      package.json
    > server.js
  > server.js
  > package.json

server.js
  1 import express from "express";
  2 import bodyParser from "body-parser";
  3 import mysql from "mysql2";
  4 import bcrypt from "bcrypt";
  5 import path from "path";
  6 import { fileURLToPath } from "url";
  7 const app = express();
  8 const port = 3000;
  9 // Setup __dirname for ES modules
10 const __filename = fileURLToPath(import.meta.url);
11 const __dirname = path.dirname(__filename);
12 // Middleware
13 app.use(bodyParser.json());
14 app.use(bodyParser.urlencoded({ extended: true }));
15 app.use(express.static(path.join(__dirname, "public"))); // static files
16 // Set EJS as view engine
17 app.set("view engine", "ejs");
18 app.set("views", path.join(__dirname, "views"));
19
20 // DB Connection
21 const db = mysql.createConnection({
22   host: "localhost",
23   user: "root",           // your mysql username
24   password: "raj12345", // your mysql password
25   database: "rajdb"
26 });
27 db.connect((err) => {
28   if (err) { console.error("DB connection failed:", err);
29   } else { console.log("MySQL Connected!"); } });
30 // ----- Routes -----
31 // Show signup form
32 app.get("/signup", (req, res) => {
33   res.render("signup");
34 // Handle signup form
35 app.post("/signup", async (req, res) => {
36   const { name, email, password } = req.body;
37   const hashed = await bcrypt.hash(password, 10);
38   const sql = "INSERT INTO users (name, email, password) VALUES (?, ?, ?)";
39   db.query(sql, [name, email, hashed], (err) => {
40     if (err) return res.status(500).send("Error: " + err.message);
41     res.send("Signup successful!");
42   });
43 // Show login form
44 app.get("/login", (req, res) => {
45   res.render("login");
46 });
47 // Handle login form
48 app.post("/login", (req, res) => {
49   const { email, password } = req.body;
50   const sql = "SELECT * FROM users WHERE email = ?";
51   db.query(sql, [email], async (err, results) => {
52     if (err) return res.status(500).send("Error: " + err.message);
53     if (results.length === 0) return res.status(404).send("User not found");
54     const user = results[0];
55     const match = await bcrypt.compare(password, user.password);
56     if (!match) return res.status(401).send(" Invalid password");
57     res.send(`Welcome ${user.name}, Login successful!`);
58   });
59 // Start server
60 app.listen(port, () => { console.log(`Server running at http://localhost:${port}`) });
61
  
```

```

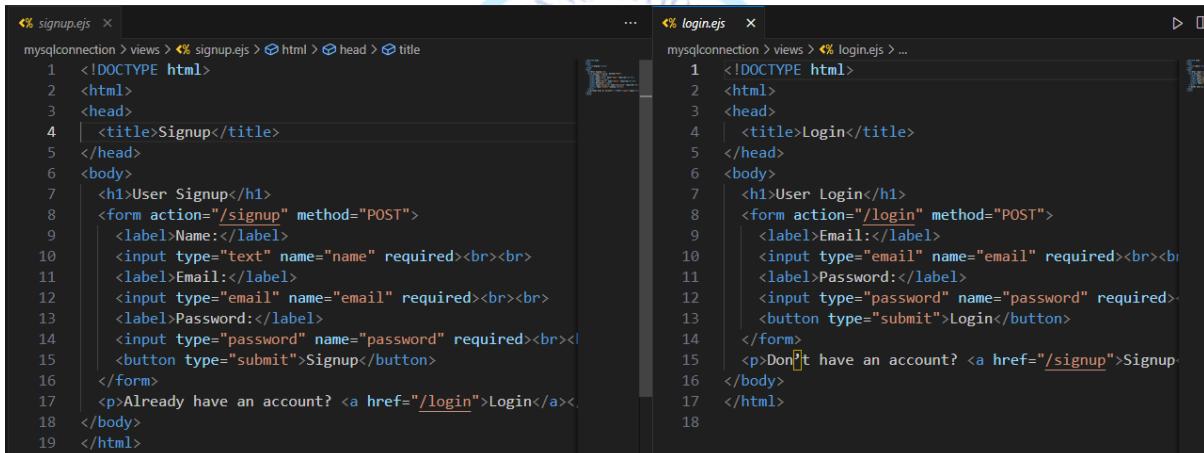
mysql> CREATE DATABASE rajdb;
Query OK, 1 row affected (0.12 sec)

mysql> use rajdb;
Database changed
mysql> CREATE TABLE users (
    ->     id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(100),
    ->     email VARCHAR(100) UNIQUE,
    ->     password VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.29 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_rajdb |
+-----+
| users           |
+-----+
1 row in set (0.24 sec)

mysql> SELECT * FROM users;
+----+----+----+----+
| id | name | email        | password          |
+----+----+----+----+
|  1 | raj  | raj@gmail.com | $2b$10$USHpWJClKvujkGoY8F60l.Vf18VLWEzf/0wPLLYLT.sEY43kq03/G |
+----+----+----+----+
1 row in set (0.00 sec)

```



```

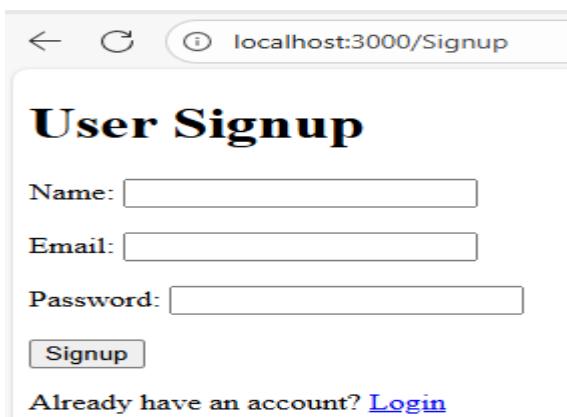
<% signup.ejs > ...
mysqlconnection > views > <% signup.ejs > <html> <head> <title>
1  <!DOCTYPE html>
2  <html>
3  <head>
4  | <title>Signup</title>
5  </head>
6  <body>
7  | <h1>User Signup</h1>
8  | <form action="/signup" method="POST">
9  | | <label>Name:</label>
10 | | <input type="text" name="name" required><br><br>
11 | | <label>Email:</label>
12 | | <input type="email" name="email" required><br><br>
13 | | <label>Password:</label>
14 | | <input type="password" name="password" required><br><br>
15 | | <button type="submit">Signup</button>
16 | </form>
17 | <p>Already have an account? <a href="/login">Login</a><
18 </body>
19 </html>
20

```

```

<% login.ejs > ...
mysqlconnection > views > <% login.ejs > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4  | <title>Login</title>
5  </head>
6  <body>
7  | <h1>User Login</h1>
8  | <form action="/login" method="POST">
9  | | <label>Email:</label>
10 | | <input type="email" name="email" required><br><br>
11 | | <label>Password:</label>
12 | | <input type="password" name="password" required>
13 | | <button type="submit">Login</button>
14 | </form>
15 | <p>Don't have an account? <a href="/signup">Signup</a>
16 </body>
17 </html>
18

```



localhost:3000/Signup

User Signup

Name:

Email:

Password:

Already have an account? [Login](#)

Here's a **short explanation** of your code:

1. Imports & Setup

- express → Web framework for routing.
- body-parser → Parses form/JSON data.
- mysql2 → Connects to MySQL database.
- bcrypt → Hashes passwords for security.
- path & fileURLToPath → Handle file paths in ES modules.

2. Server Setup

- app = express() → Initialize Express app.
- port = 3000 → Server will run on port 3000.
- __dirname setup → For ES module compatibility.

3. Middleware

- bodyParser.json() & urlencoded() → Parse incoming JSON and form data.
- express.static() → Serve static files from public folder.
- app.set("view engine", "ejs") → Use EJS templates.

4. Database Connection

- Connects to MySQL rajdb with root user and password.
- Logs success or error on connection.

5. Routes

- GET /signup → Shows signup form.
- POST /signup → Saves user with hashed password in users table.
- GET /login → Shows login form.
- POST /login → Checks user email, compares password with hashed password, and responds accordingly.

6. Start Server

- app.listen(port) → Runs server at <http://localhost:3000>.

💡 Summary:

This is a **basic Express + MySQL + EJS app** that allows **user signup and login with password hashing**.

If you want, I can also make a **tiny flow diagram** of how signup/login works—it'll be easier to visualize. Do you want me to do that?

Here's a clear explanation of your code:

Code Breakdown

```
app.post("/signup", async (req, res) => {
  const { name, email, password } = req.body; // 1
  const hashed = await bcrypt.hash(password, 10); // 2
  const sql = "INSERT INTO users (name, email, password) VALUES (?, ?, ?)"; // 3
  db.query(sql, [name, email, hashed], (err) => { // 4
    if (err) return res.status(500).send("Error: " + err.message); // 5
    res.send("Signup successful!"); // 6
  });
});
```

Step-by-Step Explanation

1. Extract form data

```
const { name, email, password } = req.body;
```

- Takes name, email, password from the signup form sent by the client.

2. Hash password

```
const hashed = await bcrypt.hash(password, 10);
```

- Uses bcrypt to securely hash the password before saving it.
- 10 → Number of salt rounds (strength of hashing).

3. SQL query with ? placeholders

```
const sql = "INSERT INTO users (name, email, password) VALUES (?, ?, ?);
```

- ? are **placeholders** for values.
- They prevent **SQL injection attacks** (safer than directly inserting variables into the query).

4. Execute query

```
db.query(sql, [name, email, hashed], (err) => { ... });
```

- [name, email, hashed] → These values replace the ? in the SQL query **in order**.
- db.query runs the SQL command in the database.

5. Error handling

```
if (err) return res.status(500).send("Error: " + err.message);
```

- Sends an error response if the database insertion fails.

6. Success response

```
res.send("Signup successful!");
```

- Sends a confirmation message to the user.

Key point about ? (placeholders)

• Why use ??

- Protects against **SQL injection**.
- Automatically escapes special characters in user input.
- Cleaner and safer than concatenating strings manually:

```
// Unsafe way
```

```
const sql = "INSERT INTO users (name, email, password) VALUES ('" + name + "','" + email + "','" + hashed + "')";
```

Here's a **line-by-line explanation** of your login code:

```
// Handle login form
```

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

- Creates a **POST route** for /login.
- This route will handle **form submissions** from the login page.
- req → Request from the client (contains form data).
- res → Response sent back to the client.

```
const { email, password } = req.body;
```

- Extracts email and password submitted by the user from the request body.

```
const sql = "SELECT * FROM users WHERE email = ?";
```

- Prepares a **SQL query** to find a user in the users table with the given email.
- ? is a **placeholder** to safely insert the email (prevents SQL injection).

```
db.query(sql, [email], async (err, results) => {
```

- Executes the SQL query.
- [email] replaces the ? placeholder in the SQL query.
- results contains the rows returned from the database.
- err will contain any database error.

```
if (err) return res.status(500).send("Error: " + err.message);
```

- Checks for **database errors**.
- If an error occurs, sends **HTTP 500** and the error message.

```
if (results.length === 0) return res.status(404).send("User not found");
```

- Checks if **no user** with that email exists.
- Sends **HTTP 404** if user is not found.



```
const user = results[0];
  • If a user is found, take the first row (the only user with that email).
const match = await bcrypt.compare(password, user.password);
  • Compares the entered password with the hashed password stored in the database using bcrypt.
  • match will be true if passwords match, false otherwise.
if (!match) return res.status(401).send(" Invalid password");
  • If password does not match, sends HTTP 401 Unauthorized with message.
res.send('Welcome ${user.name}, Login successful!');
  • If password matches, sends a success message including the user's name.
  •
```

Mysql connection with Node JS/Express JS

If you want to **switch from MongoDB to MySQL** and connect as user you need to use a MySQL driver for Node.js, such as **mysql2** or **sequelize**.

1. Install MySQL package

- In your project folder, run:
- `npm install mysql2`

2. Import and connect in server.js

- Replace your MongoDB code with MySQL connection:

```
import mysql from "mysql2";
// Create connection
const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "raj12345",
  database: "your_database_name" // replace with your DB name
}); // Connect
db.connect((err) => {
  if (err) {console.error("MySQL connection failed:", err);}
  } else { console.log("MySQL Database connected");}
}); // Export db if needed in routers/controllers
export default db;
```

3. Using MySQL in your login/signup controllers

- ➔ Instead of Mongoose `SignupUser.findOne()`, you'll run **SQL queries**:

```
import db from "../server.js"; // import the MySQL connection
export const loginUser = (req, res) => {
  const { email, password } = req.body;
  const sql = "SELECT * FROM users WHERE email = ?";
  db.query(sql, [email], (err, results) => {
    if (err) return res.send("DB Error");
    if (results.length === 0) { return res.send("User not found");}
    const user = results[0];
    if (user.password !== password) { return res.send("Invalid password");}
    console.log("Login successful");
    res.redirect("/"); // redirect to homepage
  });
};
```

- ➔ `users` is your MySQL table where you store user data.

Notes MongoDB and Mongoose are **NoSQL**, MySQL is **SQL**, so you need to adjust all database queries accordingly.

- Create a table in MySQL, for example:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    mobile VARCHAR(15),
    password VARCHAR(100),
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Project with MySQL

1. **Install required packages** npm install express ejs mysql2 multer body-parser path
2. **Create MySQL database and table**

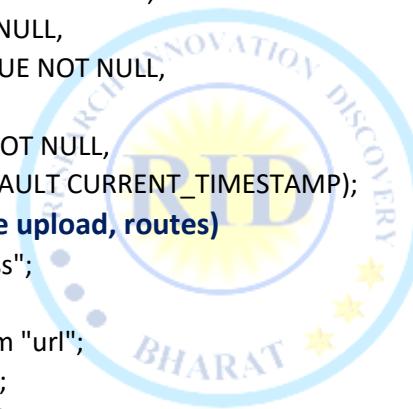
```
CREATE DATABASE rajdb;
```

```
USE rajdb;
```

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    mobile VARCHAR(15),
    password VARCHAR(100) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

3. **server.js (with MySQL, file upload, routes)**

```
import express from "express";
import path from "path";
import { fileURLToPath } from "url";
import mysql from "mysql2";
import multer from "multer";
import signupRouter from "./routes/signup.js";
import loginRouter from "./routes/login.js";
const app = express();
const port = 3000;
// __dirname setup
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
// EJS setup
app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));
// Middleware
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, "public")));
// MySQL connection
export const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "raj12345",
```



```
        database: "rajdb"
    });
    db.connect(err => {
        if (err) console.error("MySQL connection failed:", err);
        else console.log("MySQL Database connected");
    });
// Multer setup
const storage_1 = multer.diskStorage({
    destination: (req, file, cb) => cb(null, "uploads/"),
    filename: (req, file, cb) => cb(null, Date.now() + " " + file.originalname)
});
export const upload_1 = multer({ storage: storage_1 });
```

// Routes

```
app.get("/", (req, res) => res.render("index"));
app.get("/Gallery", (req, res) => res.render("Gallery"));
app.get("/Contact", (req, res) => res.render("Contact"));
app.get("/Service", (req, res) => res.render("Service"));
app.get("/News", (req, res) => res.render("News"));
app.get("/Signup", (req, res) => res.render("Signup"));
app.get("/Login", (req, res) => res.render("Login"));
app.use("/", signupRouter);
app.use("/", loginRouter);
// Start server app.listen(port, () => console.log(`Server running at http://localhost:${port}`));
```

4. controllers/signupc.js (Insert data into MySQL)

```
import { db } from "../server.js";
export const registerUser = (req, res) => {
    const { name, email, mobile, password } = req.body;
    const sql = "INSERT INTO users (name, email, mobile, password) VALUES (?, ?, ?, ?)";
    db.query(sql, [name, email, mobile, password], (err, result) => {
        if (err) { console.error("Signup Error:", err);
            return res.send("Error registering user. Maybe email already exists.");
        }
        console.log("User registered:", result.insertId);
        res.redirect("/Login"); // after signup, redirect to login
    });
};
```

5. controllers/logincontroller.js (Check user + password)

```
import { db } from "../server.js";
export const loginUser = (req, res) => {
    const { email, password } = req.body;
    const sql = "SELECT * FROM users WHERE email = ?";
    db.query(sql, [email], (err, results) => {
        if (err) return res.send("DB Error");
        if (results.length === 0) return res.send("User not found");
        const user = results[0];
        if (user.password !== password) return res.send("Invalid password");
        console.log("Login successful");
        res.redirect("/"); // redirect to homepage
    });
};
```



});};

6. **routes/signup.js**

```
import express from "express";
import { registerUser } from "../controllers/signupc.js";
const router = express.Router();
router.post("/req-signup", registerUser);
export default router;
```

7. **routes/login.js**

```
import express from "express";
import { loginUser } from "../controllers/logincontroller.js";
const router = express.Router();
router.post("/req-login", loginUser);
export default router;
```

Note: With this setup, your **signup/login flow works with MySQL**.

- Form data → req.body → saved in MySQL table users
- Login checks email/password from MySQL → redirects on success



What is API in node js/ Express js

❖ What is an API?

- API = Application Programming Interface
- Works like a **restaurant menu**:
 - Menu = API (tells what you can order).
 - Chef = Server (prepares the data).
 - Waiter = Response (brings the data).
- You don't care *how* it's cooked, you just get the result.

❖ API in Node.js / Express.js

- Express.js makes it easy to build APIs.
- APIs usually use **JSON** for data exchange.
- Clients (browser, mobile, other servers) talk to the API using **HTTP methods**:
 - GET → Read data
 - POST → Create data
 - PUT → Update data
 - DELETE → Remove data

2) Types of APIs In Node.js, common API types are:

1. **REST API** (Representational State Transfer)
 - Uses HTTP methods: GET, POST, PUT, DELETE.
 - Returns data usually in JSON.
 - Example: /api/books → get list of books.
2. **GraphQL API**
 - Uses queries instead of fixed routes.
 - Example: { books { id title } }.
3. **SOAP API** (less used in Node.js today)
 - XML-based.

Note: In practice, **REST API** is the most popular in Node.js apps.

3) Steps to build a REST API in Node.js + Express

Step 1: Create project

```
mkdir restful-api
cd restful-api
npm init -y
npm install express
npm install --save-dev nodemon
```

Step 2: Create index.js

```
import express from "express";
const app = express();
const port = 3000;
```

Middleware

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

Demo "database"

```
let books = [
  { id: 1, title: "Python", author: "Author RP sir" },
  { id: 2, title: "Node js", author: "Author RP sir" }
];
```

Routes, Root

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



```
    res.send("Welcome to Books REST API");
  });
}
```

GET all books

```
app.get("/api/books", (req, res) => {
  res.json(books);
});
```

GET one book

```
app.get("/api/books/:id", (req, res) => {
  const id = Number(req.params.id);
  const book = books.find(b => b.id === id);
  if (!book) return res.status(404).json({ error: "Book not found" });
  res.json(book);
});
```

POST create book

```
app.post("/api/books", (req, res) => {
  const { title, author } = req.body;
  if (!title || !author) {
    return res.status(400).json({ error: "Title and author required" });
  }
  const id = books.length ? books[books.length - 1].id + 1 : 1;
  const newBook = { id, title, author };
  books.push(newBook);
  res.status(201).json(newBook);
});
```

PUT update book

```
app.put("/api/books/:id", (req, res) => {
  const id = Number(req.params.id);
  const index = books.findIndex(b => b.id === id);
  if (index === -1) return res.status(404).json({ error: "Book not found" });
  const { title, author } = req.body;
  if (!title || !author) {
    return res.status(400).json({ error: "Title and author required" });
  }
  books[index] = { id, title, author };
  res.json(books[index]);
});
```

DELETE book

```
app.delete("/api/books/:id", (req, res) => {
  const id = Number(req.params.id);
  const index = books.findIndex(b => b.id === id);
  if (index === -1) return res.status(404).json({ error: "Book not found" });

  const removed = books.splice(index, 1)[0];
  res.json({ message: "Deleted", book: removed });
});
```

Error handler

```
app.use((err, req, res, next) => {
  console.error(err.stack);
```



```
        res.status(500).json({ error: "Server error" });
    });
    app.listen(port, () => {
        console.log(`Server running at http://localhost:${port}`);
    });
}
```

Step 3: Run the server

npm run dev

Output: Server running at http://localhost:3000

Step 4: Test API

1) GET all books

Browser or Postman:

GET http://localhost:3000/api/books

Output:

```
[  
  { "id": 1, "title": "Book One", "author": "Author A" },  
  { "id": 2, "title": "Book Two", "author": "Author B" }  
]
```

2) GET one book

GET http://localhost:3000/api/books/1

Output:

```
{ "id": 1, "title": "Book One", "author": "Author A" }
```

3) POST new book

Request body (JSON):

```
{ "title": "New Book", "author": "Sangam" }
```

Output:

```
{ "id": 3, "title": "New Book", "author": "sangam" }
```

4) PUT update book

Request:

PUT http://localhost:3000/api/books/2

Body:

```
{ "title": "Updated Title", "author": "Updated Author" }
```

Output:

```
{ "id": 2, "title": "Updated Title", "author": "Updated Author" }
```

5) DELETE book

DELETE http://localhost:3000/api/books/1

Output:

```
{  
  "message": "Deleted",  
  "book": { "id": 1, "title": "Book One", "author": "Author A" }  
}
```

Note: API in Node.js = interface for client apps (frontend, mobile, etc.) to interact with server.

- **Types of APIs:** REST, GraphQL, SOAP (REST is most common).
- **RESTful API** uses HTTP methods:
 - GET → Read
 - POST → Create
 - PUT → Update
 - DELETE → Remove
- **We built a Books API** step by step and tested it.



```
server.js  X
allnewproject > server.js > ...
1 import express from "express";
2 const app = express();
3 const port = 3000;
4 // Middleware
5 app.use(express.json());
6 // Demo "database"
7 let books = [
8   { id: 1, title: "Book One", author: "Author RP sir" },
9   { id: 2, title: "Book Two", author: "Author Sangam Kumar" }
10 ];
11 // Routes// Root
12 app.get("/", (req, res) => {res.send("Welcome to Books REST API");});
13 // GET all books
14 app.get("/api/books", (req, res) => {res.json(books);});
15 // GET one book
16 app.get("/api/books/:id", (req, res) => {
17   const id = Number(req.params.id);
18   const book = books.find(b => b.id === id);
19   if (!book) return res.status(404).json({ error: "Book not found" });
20   res.json(book);
21 });
22 app.listen(port, () => {
23   console.log(`Server running at http://localhost:\${port}`);
24 });
25
```

always use req.params to access :id from the URL

GET → <http://localhost:3000/api/books>

GET → <http://localhost:3000/api/books/1>

❖ Why do we use params (req.params.id)?

- :id in /api/books/:id is a **route parameter**.
 - It means: “Take the id from the URL”.
- Example: If you go to <http://localhost:3000/api/books/1>
- req.params.id = "1"
 - Code finds the book with id: 1 → returns **Book One**.
 - If you go to <http://localhost:3000/api/books/5>
 - Book doesn't exist → returns **404 error (Book not found)**.

Note: - params let us **capture dynamic values from the URL** (like book ID) instead of making separate routes for each book.



```
js server.js  X
allnewproject > js server.js > ...
1  import express from "express";
2  const app = express();
3  const port = 3000;
4  // Middleware
5  app.use(express.json());
6  // Demo "database" (projects)
7  let projects = [
8    { id: 1, title: "AI Chatbot", domain: "Artificial Intelligence", mentor: "Dr. R.P. sir" },
9    { id: 2, title: "Smart Irrigation System", domain: "IoT", mentor: "Prof. Neha Sharma" },
10   { id: 3, title: "Blockchain Voting App", domain: "Blockchain", mentor: "Dr. R.P. Singh" }
11 ];// Routes// Root
12 app.get("/", (req, res) => { res.send("Welcome to Engineering Projects REST API ");});
13 // GET all projects
14 app.get("/api/projects", (req, res) => {res.json(projects);});
15 // GET one project by id
16 app.get("/api/projects/:id", (req, res) => {
17   const id = Number(req.params.id);
18   const project = projects.find(p => p.id === id);
19   if (!project) return res.status(404).json({ error: "Project not found" });
20   res.json(project);
21 });

22 // Add a new project
23 app.post("/api/projects", (req, res) => {
24   const { title, domain, mentor } = req.body;
25   if (!title || !domain || !mentor) {
26     return res.status(400).json({ error: "title, domain, and mentor are required" });
27   const id = projects.length ? projects[projects.length - 1].id + 1 : 1;
28   const newProject = { id, title, domain, mentor };
29   projects.push(newProject);
30   res.status(201).json(newProject);});// Update a project
31 app.put("/api/projects/:id", (req, res) => {
32   const id = Number(req.params.id);
33   const index = projects.findIndex(p => p.id === id);
34   if (index === -1) return res.status(404).json({ error: "Project not found" });
35   const { title, domain, mentor } = req.body;
36   if (!title || !domain || !mentor) {
37     return res.status(400).json({ error: "title, domain, and mentor are required" });
38   projects[index] = { id, title, domain, mentor };
39   res.json(projects[index]);});// Delete a project
40 app.delete("/api/projects/:id", (req, res) => {
41   const id = Number(req.params.id);
42   const index = projects.findIndex(p => p.id === id);
43   if (index === -1) return res.status(404).json({ error: "Project not found" });
44   const removed = projects.splice(index, 1)[0];
45   res.json({ message: "Project deleted successfully", project: removed });
46   app.listen(port, () => {console.log(`Server running at http://localhost:\${port}`);});
47

let id;
if (projects.length === 0) {
  id = 1; } else {
  id = projects[projects.length - 1].id + 1;
const newProject = { id, title, domain, mentor };
projects.push(newProject);
res.status(201).json(newProject);});
```

How to create full flex RESTful API in Node JS

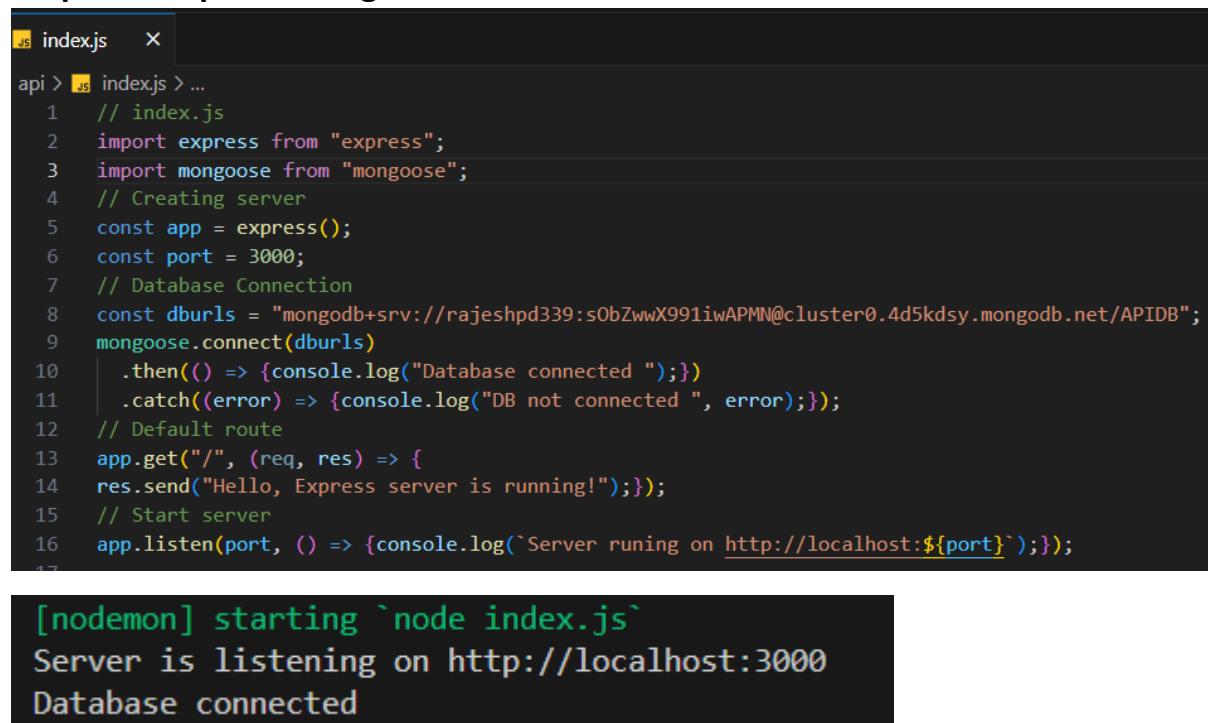
Step 1: Create project

```
mkdir restful-api
cd restful-api
npm init -y
npm install express mongoose bcryptjs
npm install --save-dev nodemon
```

Step 2: Create index.js(Server)

```
import express from "express";
const app = express();
const port = 3000;
Default route app.get("/", (req, res) => {res.send("Hello, Express server is running!");});
Start server app.listen(port, () => { console.log(`Server is listening on http://localhost:${port}`);});
```

Step-3: Setup the MongoDB connection



```
index.js  x
api > index.js > ...
1 // index.js
2 import express from "express";
3 import mongoose from "mongoose";
4 // Creating server
5 const app = express();
6 const port = 3000;
7 // Database Connection
8 const dburls = "mongodb+srv://rajeshpd339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/APIDB";
9 mongoose.connect(dburls)
10 .then(() => {console.log("Database connected ");}
11 .catch((error) => {console.log("DB not connected ", error);});
12 // Default route
13 app.get("/", (req, res) => {
14 res.send("Hello, Express server is running!");
15 // Start server
16 app.listen(port, () => {console.log(`Server runing on http://localhost:${port}`);});
```

```
[nodemon] starting `node index.js`
Server is listening on http://localhost:3000
Database connected
```

Database Connection

```
const dburls="mongodb+srv://rajeshpd339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/APIDB";
mongoose.connect(dburls)
.then(() => {console.log("Database connected ");}
.catch((error) => {console.log("DB not connected ", error);});
```

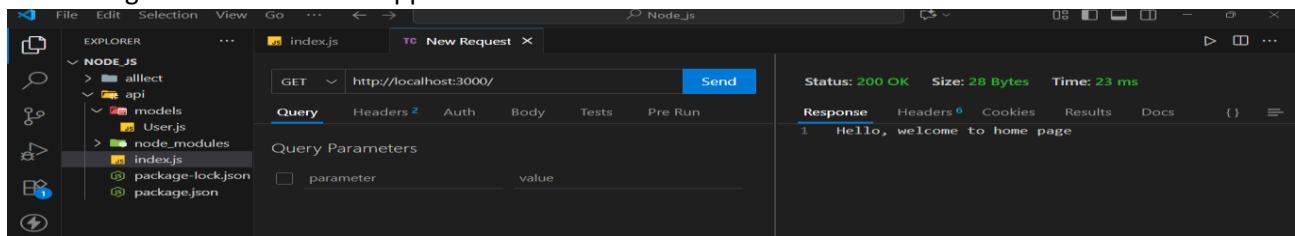
Note:

To test APIs without building a full frontend, you can use tools like **Postman** or **Thunder Client**.

- If you prefer working inside VS Code, install the Thunder Client extension from the Extensions Marketplace. After installation, open Thunder Client and click on "New Request".
- Select the HTTP method (e.g., GET, POST, PUT, DELETE) and enter your API route (e.g., <http://localhost:3000/api/users>).
- Add any required headers or request body (for POST/PUT).
- Click Send to send the request and view the response directly from your server.



This helps in quickly verifying whether your backend API endpoints are working correctly before connecting them to a frontend application.



Step-4: Create model (folder) store data

```

EXPLORER      ...
NODEJS
  > allect
  > api
    > models
      > User.js
    > node_modules
      > index.js
      > package-lock.json
      > package.json
  > index.js
  > package.json

index.js      User.js  X
api > models > User.js > ...
1 import mongoose from "mongoose";
2 // We are creating the user API // Step 1: Define the schema
3 const userSchema = new mongoose.Schema({
4   name: { type: String, required: true },
5   email: { type: String, required: true, unique: true },
6   password: { type: String, required: true },
7   createdAt: { type: Date, default: Date.now } // Auto-filled Date
8 });// Step 2: Create the model from schema
9 const User = mongoose.model("User", userSchema);
10 export default User;
11

```

Step-5: Write the route for user models and creating the api

Final Example for testing the api

```

EXPLORER      ...
NODEJS
  > allect
  > api
    > models
      > User.js
    > node_modules
      > index.js
      > package-lock.json
      > package.json
  > index.js
  > package.json

index.js      User.js  X
api > index.js > ...
1 import express from "express";
2 import mongoose from "mongoose"; // importing mongoose for db connection
3 import User from "./models/User.js"; // importing models for save data
4 const app = express(); // Creating server
5 const port = 3000;
6 app.use(express.json()); // Middleware (to read JSON data from client)
7 // Database Connection
8 const dburls = "mongodb+srv://rajeshpd339:s0bZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/APIDB";
9 mongoose.connect(dburls)
10 .then(() => console.log("Database connected"))
11 .catch((error) => console.log("DB not connected", error));
12 // Default route
13 app.get("/", (req, res) => {res.send("Hello, welcome to home page ");});
14 // User Register API @api desc: user register @api method: POST
15 // @api endpoint: /api/user/register
16 app.post('/api/user/register', async (req, res) => {
17   try { const { name, email, password } = req.body;
18         const newUser = new User({ name, email, password });
19         await newUser.save();
20         res.json({msg: "User registered successfully",user: newUser});
21       } catch (error) { console.error("Error saving user:", error);
22                     res.status(500).json({ error: "Failed to register user" });
23       }
24   app.listen(port, () => {console.log(`Server running on http://localhost:${port}`)});
25 }

```

Test API

POST http://localhost:3000/api/user/register

```

1  {
2      "msg": "getting data from client",
3      "data": {
4          "name": "Sangam Kumar",
5          "email": "sangam@gmail.com",
6          "password": "123"
7      }
8  }

```

PS C:\Users\hp\OneDrive\Desktop\Node_js\api> **node** index.js
[nodemon] starting `node index.js`
Server running on http://localhost:3000
Database connected
data { name: 'Sangam Kumar', email: 'sangam@gmail.com', password: '123' }

Storing data in database

APIDB.users

STORAGE SIZE: 20KB LOGICAL DATA SIZE: 122B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 40KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

Filter Type a query: { field: 'value' } **Reset** **Apply**

QUERY RESULTS: 1-1 OF 1

```

_id: ObjectId('68c536264b72a7b13e982cec')
name : "Sangam Kumar"
email : "sangam@gmail.com"
password : "123456"
createdAt : 2025-09-13T09:15:18.037+00:00
__v : 0

```

RESTful API in Node.js with MongoDB.

Folder Structure

```

project-folder/
|--- index.js      # Main server file
|--- package.json
|--- models/       # DB Schemas
|   |--- User.js
|--- routes/       # API routes
|   |--- userRoutes.js
|--- controllers/ # Business logic
|   |--- userController.js

```

Steps (with requirements)

Step 1: Initialize project

```
npm init -y
```

Step 2: Install required packages

```
npm install express mongoose nodemon
```

Step 3: Setup package.json

Add "type": "module" and nodemon script:

```
"scripts": {  
  "dev": "nodemon index.js"  
}
```

Step 4: Create index.js (server + DB connect)

- Import express, mongoose.
- Connect MongoDB (mongoose.connect).
- Add middleware (express.json()).
- Use routes (app.use('/api/users', userRoutes)).
- Start server.

Step 5: Create models/User.js

- Define schema (name, email, password, createdAt).
- Export model.

Step 6: Create controllers/userController.js

- **Functions:**
 - registerUser → POST /register
 - getUsers → GET /
 - getUserById → GET /:id
 - updateUser → PUT /:id
 - deleteUser → DELETE /:id

Step 7: Create routes/userRoutes.js

- Import express.Router().
- Map routes to controller functions.
- Export router.

Step 8: Test API with Postman / Thunder Client

- POST /api/users/register → add new user.
- GET /api/users → list all users.
- PUT /api/users/:id → update user.
- DELETE /api/users/:id → remove user.



Simple api for register the user in MongoDB through the Api

1) Folder structure (what you should have)

```
project-folder/  
  |--- index.js          # updated server file  
  |--- package.json  
  |--- models/  
  |   |--- User.js        # unchanged (your model)  
  |--- controllers/  
  |   |--- userController.js # NEW  
  |--- routes/  
  |   |--- userRoutes.js   # NEW
```

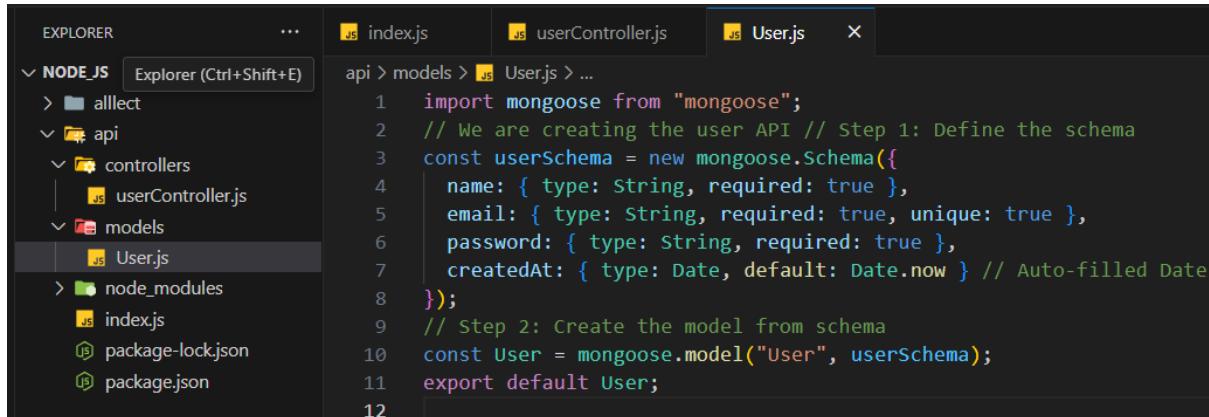
2) models/User.js

```
import mongoose from "mongoose"; // We are creating the user API // Step 1: Define the schema  
const userSchema = new mongoose.Schema({
```

```

name: { type: String, required: true },
email: { type: String, required: true, unique: true },
password: { type: String, required: true },
createdAt: { type: Date, default: Date.now } // Auto-filled Date when user is created);
const User = mongoose.model("User", userSchema); // Step 2: Create the model from schema
export default User;

```



The screenshot shows a code editor with the following structure:

- EXPLORER** pane: Shows the project structure with **NODEJS** expanded, containing **alllect**, **api** (expanded to **controllers** and **models**), **node_modules**, **index.js**, **package-lock.json**, and **package.json**.
- index.js**, **userController.js**, and **User.js** are listed in the main editor area.
- User.js** content:

```

1 import mongoose from "mongoose";
2 // We are creating the user API // Step 1: Define the schema
3 const userSchema = new mongoose.Schema({
4   name: { type: String, required: true },
5   email: { type: String, required: true, unique: true },
6   password: { type: String, required: true },
7   createdAt: { type: Date, default: Date.now } // Auto-filled Date
8 });
9 // Step 2: Create the model from schema
10 const User = mongoose.model("User", userSchema);
11 export default User;

```

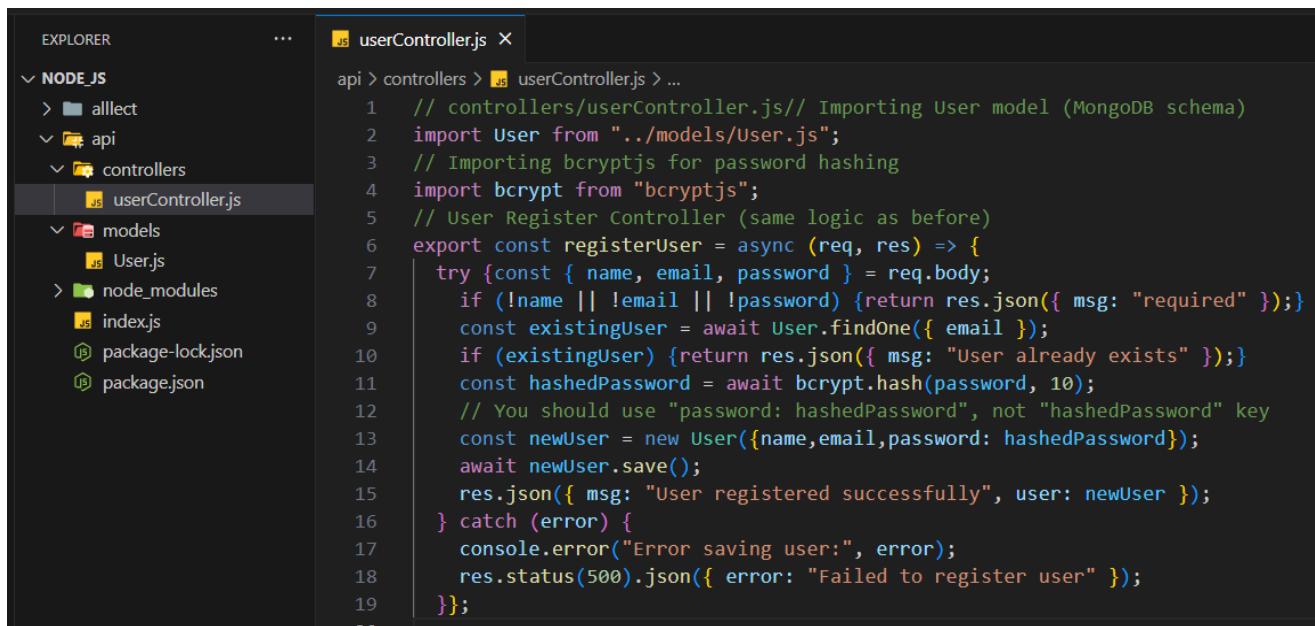
3) Create controllers/userController.js Paste this exact content: //controllers/userController.js

```

import User from "../models/User.js";
import bcrypt from "bcryptjs"; // User Register Controller (same logic as before)
export const registerUser = async (req, res) => {
  try {
    const { name, email, password } = req.body;
    if (!name || !email || !password) {
      return res.json({ msg: "All fields are required" });
    }
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.json({ msg: "User already exists" });
    }
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ name, email, password: hashedPassword });
    await newUser.save();
    res.json({ msg: "User registered successfully", user: newUser });
  } catch (error) {
    console.error("Error saving user:", error);
    res.status(500).json({ error: "Failed to register user" });
  }
}

```

Note: This file contains the **same logic and messages** you had in index.js — only moved into a controller function.



```

api > controllers > UserController.js > ...
1  // controllers/UserController.js // Importing User model (MongoDB schema)
2  import User from "../models/User.js";
3  // Importing bcryptjs for password hashing
4  import bcrypt from "bcryptjs";
5  // User Register Controller (same logic as before)
6  export const registerUser = async (req, res) => {
7      try {
8          const { name, email, password } = req.body;
9          if (!name || !email || !password) {return res.json({ msg: "required" })};
10         const existingUser = await User.findOne({ email });
11         if (existingUser) {return res.json({ msg: "User already exists" })};
12         const hashedPassword = await bcrypt.hash(password, 10);
13         const newUser = new User({name, email, password: hashedPassword});
14         await newUser.save();
15         res.json({ msg: "User registered successfully", user: newUser });
16     } catch (error) {
17         console.error("Error saving user:", error);
18         res.status(500).json({ error: "Failed to register user" });
19     }
20 }

```

4) Create routes/userRoutes.js → Paste this exact content:

routes/userRoutes.js

```

import express from "express";
import { registerUser } from "../controllers/userController.js"
const router = express.Router();

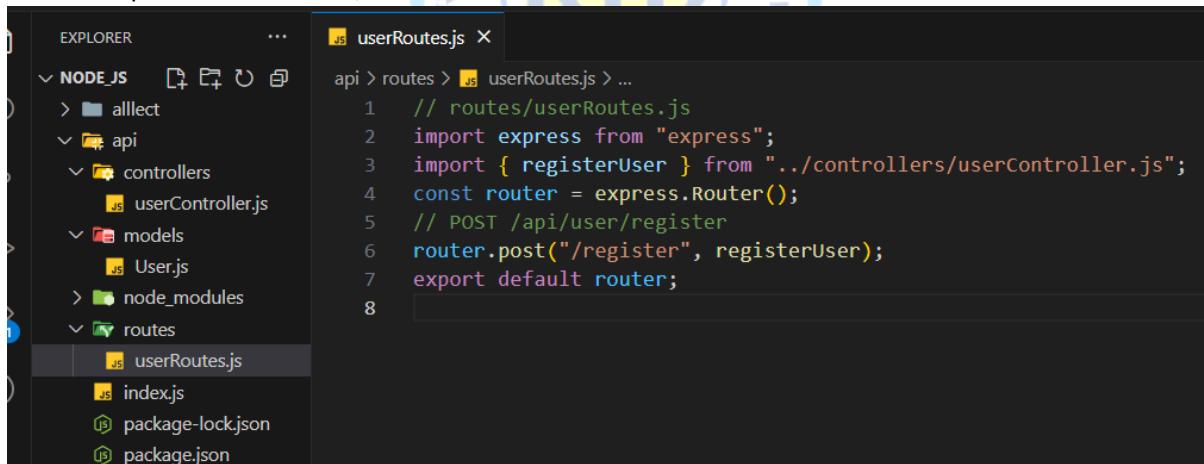
```

POST /api/user/register

```

router.post("/register", registerUser);
export default router;

```



```

api > routes > userRoutes.js > ...
1  // routes/UserRoutes.js
2  import express from "express";
3  import { registerUser } from "../controllers/userController.js";
4  const router = express.Router();
5  // POST /api/user/register
6  router.post("/register", registerUser);
7  export default router;
8

```

5) index.js (server file)

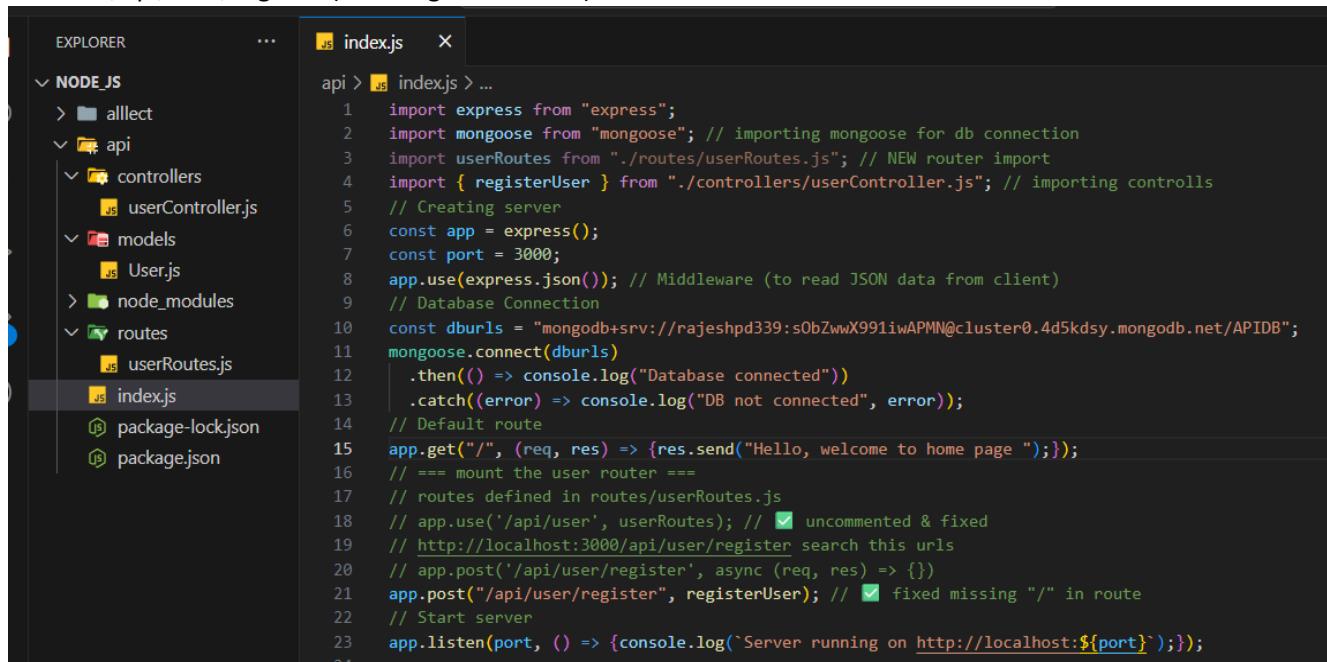
```

import express from "express";
import mongoose from "mongoose"; // importing mongoose for db connection
import User from "./models/User.js"; // importing models for save data
import bcrypt from "bcryptjs";
import userRoutes from "./routes/userRoutes.js"; // NEW router import
const app = express(); // Creating server
const port = 3000;
app.use(express.json()); // Middleware (to read JSON data from client)
// Database Connection

```

```
const dburls =
"mongodb+srv://rajeshp339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/APIDB";
mongoose.connect(dburls)
.then(() => console.log("Database connected"))
.catch((error) => console.log("DB not connected", error)); // Default route
app.get("/", (req, res) => {res.send("Hello, welcome to home page ");});
// === mount the user router === // routes defined in routes/userRoutes.js
app.use('/api/user', userRoutes); // Start server
app.listen(port, () => {
  console.log(`Server running on http://localhost:\${port}`);});
```

Note: You still have the same database connection and same default route. The register endpoint remains /api/user/register (unchanged behavior) via the mounted router.



```
EXPLORER      ...
NODEJS
  > alllect
  > api
    > controllers
      > userController.js
    > models
      > User.js
    > routes
      > userRoutes.js
    index.js
    package-lock.json
    package.json

index.js
api > index.js > ...
1  import express from "express";
2  import mongoose from "mongoose"; // importing mongoose for db connection
3  import userRoutes from "./routes/userRoutes.js"; // NEW router import
4  import { registerUser } from "./controllers/userController.js"; // importing controls
5  // Creating server
6  const app = express();
7  const port = 3000;
8  app.use(express.json()); // Middleware (to read JSON data from client)
9  // Database Connection
10 const dburls = "mongodb+srv://rajeshp339:sObZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/APIDB";
11 mongoose.connect(dburls)
12 .then(() => console.log("Database connected"))
13 .catch((error) => console.log("DB not connected", error));
14 // Default route
15 app.get("/", (req, res) => {res.send("Hello, welcome to home page ")}); // === mount the user router === // routes defined in routes/userRoutes.js
16 // app.use('/api/user', userRoutes); //  uncommented & fixed
17 // http://localhost:3000/api/user/register search this urls
18 // app.post('/api/user/register', async (req, res) => {})
19 // app.post("/api/user/register", registerUser); //  fixed missing "/" in route
20 // Start server
21 app.listen(port, () => {console.log(`Server running on http://localhost:\${port}`)});
```

6) Quick checklist & commands

1. **Create the folders:**
mkdir controllers routes
 2. **Create the files and paste the code above:**
 - controllers/userController.js
 - routes/userRoutes.js
 - update index.js as shown
 3. Restart server: npm run dev # or nodemon index.js / node index.js
 4. Test with Thunder Client / Postman:
 - **POST** <http://localhost:3000/api/user/register>
 - Body (JSON):


```
{ "name": "Rajesh",
        "email": "rajesh@example.com",
        "password": "123456"}
```
- } You should get the same response as earlier:
- ```
{"msg": "User registered successfully",
"user": { ... }}
```

And the user document will be saved in APIDB.users on MongoDB Atlas.

THUNDER CLIE... ॐ ...

New Request | ...

Activity Collections Env

filter activity

POST localhost:3000/... 13 mins ago

POST localhost:3000/... 7 days ago

localhost:3000/api/user/login

Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```

1 {
2 "msg": "Welcome to our API RD Sir",
3 "success": true
4 }

```

Status: 200 OK Size: 50 Bytes Time: 304 ms

Response Headers 6 Cookies Results Docs

### Complete Restful API with database connection Login Register and Login Step-1: Index.js

EXPLORER

NODE\_JS

- alllett
- api
  - controllers
    - UserController.js
  - models
    - User.js
  - routes
    - userRoutes.js
    - index.js
- node\_modules
- routes
- index.js
- package-lock.json
- package.json

index.js

```

api > index.js > ...
1 import express from "express";
2 import mongoose from "mongoose";
3 import userRoutes from "./routes/userRoutes.js";
4 const app = express();
5 const port = 3000;
6 app.use(express.json()); // Middleware to parse JSON
7 // Database connection
8 const dburls = "mongodb+srv://rajeshpd339:s0bZwwX991iwAPMN@cluster0.4d5kdsy.mongodb.net/APIDB";
9 mongoose.connect(dburls)
10 .then(() => console.log("Database connected"))
11 .catch((error) => console.log("DB not connected", error));
12 // Default route
13 app.get("/", (req, res) => {res.send("Hello, welcome to home page");});
14 // app.post('/api/user/register', async (req, res) => {})
15 // Mount user routes
16 app.use("/api/user", userRoutes);
17 // Start server
18 app.listen(port, () => {
19 | console.log(`Server running on http://localhost:${port}`);
20 });

```

EXPLORER

NODE\_JS

- alllett
- api
  - controllers
    - UserController.js
  - models
    - User.js
  - routes
    - userRoutes.js
    - index.js
- node\_modules
- routes
- index.js
- package-lock.json
- package.json

User.js

Model(user.js and routes userRoutes.js)

```

api > models > User.js > ...
1 import mongoose from "mongoose";
2 // We are creating the user API
3 // Step 1: Define the schema
4 const userSchema = new mongoose.Schema({
5 name: { type: String, required: true },
6 email: { type: String, required: true, unique: true },
7 password: { type: String, required: true },
8 createdAt: { type: Date, default: Date.now }
9 });
10 // Step 2: Create the model from schema
11 const User = mongoose.model("User", userSchema);
12 export default User;

```

userRoutes.js

```

api > routes > userRoutes.js > ...
1 // routes/userRoutes.js
2 import express from "express";
3 import { registerUser, login } from "../controllers/UserController.js";
4
5 const router = express.Router();
6
7 // Register user
8 // POST /api/user/register
9 router.post("/register", registerUser);
10 // Login user
11 // POST /api/user/login
12 router.post("/login", login);
13
14 export default router;

```

```
userController.js ×
api > controllers > userController.js > [o] login
1 // controllers/userController.js
2 import User from "../models/User.js";
3 import bcrypt from "bcryptjs"; // Register User
4 export const registerUser = async (req, res) => {
5 try {const { name, email, password } = req.body;
6 if (!name || !email || !password) {return res.json({ msg: "All fields are required" });}
7 const existingUser = await User.findOne({ email });
8 if (existingUser) {return res.json({ msg: "User already exists" });}
9 const hashedPassword = await bcrypt.hash(password, 10);
10 const newUser = new User({ name, email, password: hashedPassword });
11 await newUser.save();
12 res.json({ msg: "User registered successfully", user: newUser });
13 } catch (error) { console.error("Error saving user:", error);
14 res.status(500).json({ error: "Failed to register user" });}
15 // Login User
16 export const login = async (req, res) => {
17 try {const { email, password } = req.body;
18 if (!email || !password) { return res.json({ msg: "All fields are required" });}
19 // Use User (capital U)
20 const user = await User.findOne({ email });
21 if (!user) {return res.json({ msg: "User not found", success: false });}
22 // password is user define password and user.password is db password bcrypt and match
23 const validPass = await bcrypt.compare(password, user.password);
24 if (!validPass) {return res.json({ msg: "Invalid password", success: false });}
25 res.json({ msg: `Welcome to our API ${user.name}`, success: true });
26 } catch (error) {console.error("Login error:", error);
27 res.status(500).json({ error: "Login failed" });}
28 }
```



# Cookies Node js

## 1. What are Cookies in Node.js?

- **Cookies** are small pieces of data stored on the **client's browser**.
- The server sends cookies with the response → the browser saves them → they are automatically sent back with every request.
- **Useful for:** Storing login tokens (like JWT) Remembering sessions Keeping track of user preferences

## 2. What is jsonwebtoken (JWT)?

- **JWT** = JSON Web Token, a compact way to securely send information between server and client.
- **When user logs in:** Server verifies credentials Creates a JWT (signed with a secret key)
  - Sends it to the client (stored in cookie or localStorage)
- On future requests, the client sends back the token → server verifies it → allows access.

## 3. Install Required Packages

- Run this in terminal:
- **npm install jsonwebtoken cookie-parser**
  - jsonwebtoken → creates and verifies tokens
  - cookie-parser → helps read/write cookies in Node.js

## 4. Basic JWT Syntax

```
import jwt from "jsonwebtoken";
// Create token
const token = jwt.sign({ userId: 123 }, "SECRET_KEY", { expiresIn: "1h" });
// Verify token
const decoded = jwt.verify(token, "SECRET_KEY");
```

## 5. Work on controls parts

### → jwt.sign() Method

→ The **jwt.sign()** method is used to **create (sign)** a **JSON Web Token (JWT)**. It takes some information (called **payload**) and signs it with a secret key to ensure that the token cannot be tampered with.

**Syntax: jwt.sign(payload, secretOrPrivateKey, [options, callback])**

- **payload** → The data you want to store inside the token (e.g., user ID, role, email).
- **secretOrPrivateKey** → A secret string or private key used to sign the token (must be kept safe).
- **options** → (Optional) Settings like expiration time, algorithm, etc.
- **callback** → (Optional) If provided, the method works asynchronously.

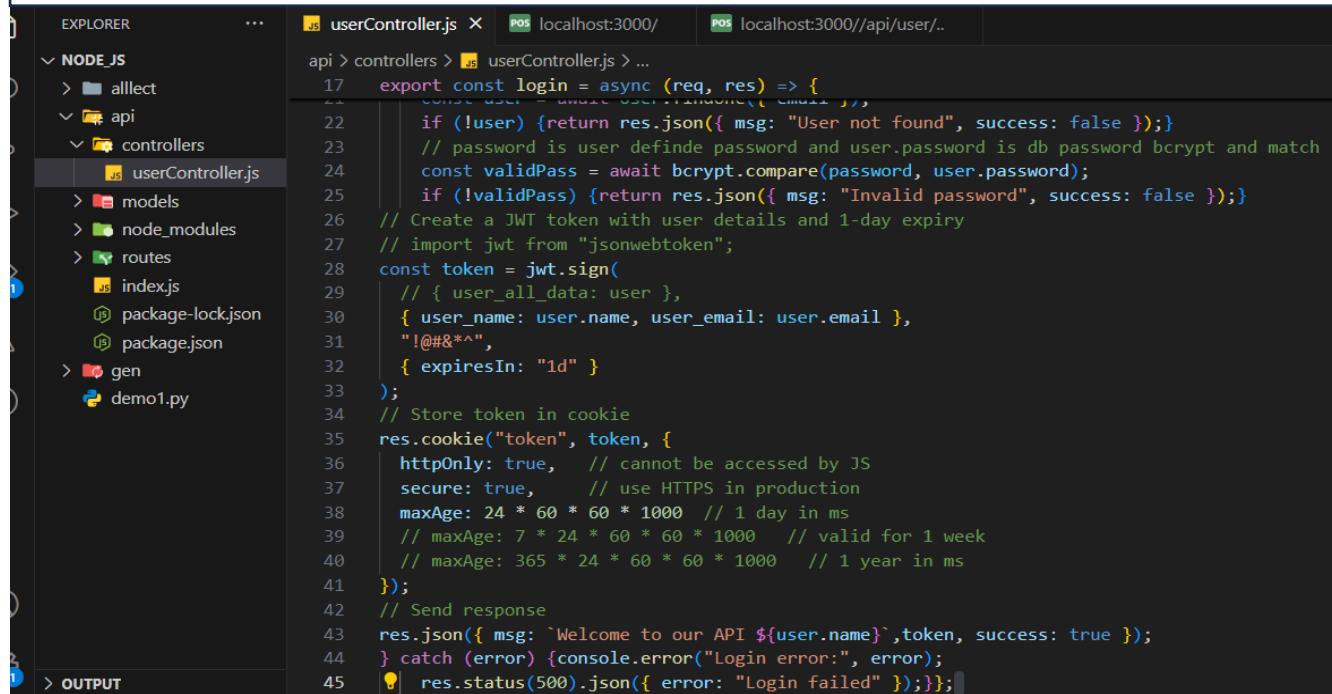
```
import jwt from "jsonwebtoken";
// Create a token
const token = jwt.sign(
 { userId: 123 }, // payload (data)
 "SECRET_KEY", // secret key
 { expiresIn: "1h" } // options (valid for 1 hour)
);
console.log(token);
Valid time formats you can use in expiresIn:
 • Seconds → "60"
 • Minutes → "10m"
 • Hours → "1h"
 • Days → "7d", "30d", "365d"
No expiry (not recommended)
```

### Explanation of Example

1. **Payload:**
  - 2. { userId: 123 }
  - Stores user ID inside the token.
  - This can be later extracted to identify the user.
3. **Secret Key:**
  - "SECRET\_KEY"
  - Used to **sign** the token.
  - Only the server knows this key, so it can later verify if the token is valid.
4. **Options (expiresIn):**
  - { expiresIn: "1h" }
  - Token will automatically expire after **1 hour**.
  - Common values: "60s", "10m", "1d", etc.



## Write the code for create the cookies in Controls.js



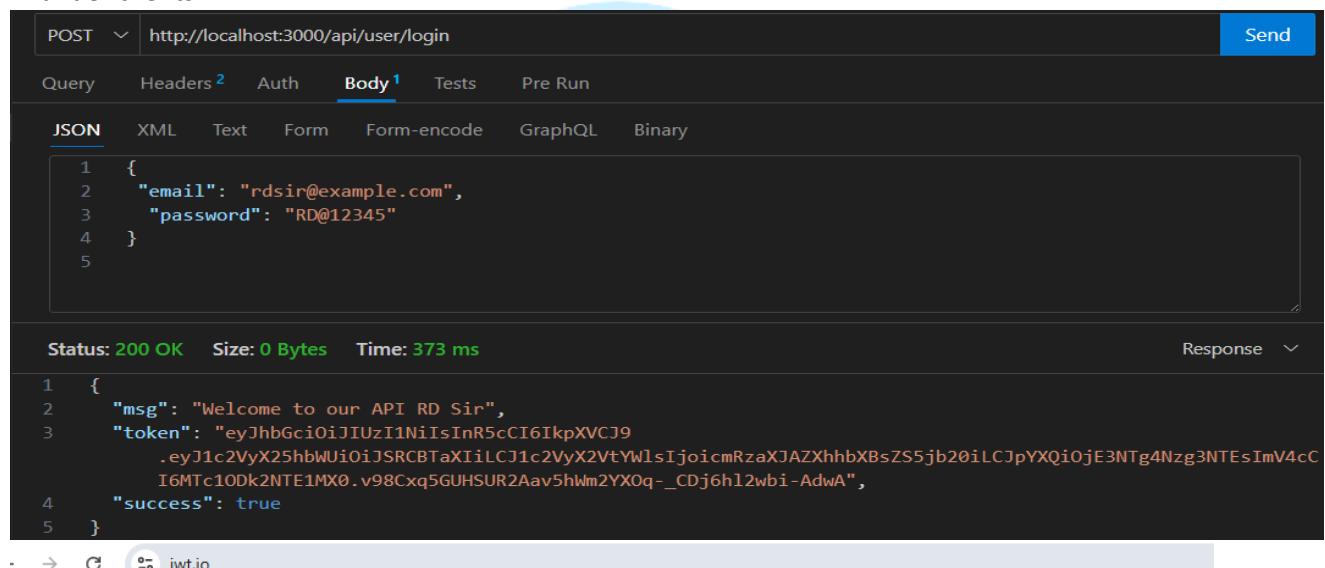
```

EXPLORER
NODE_JS
 allct
 api
 controllers
 UserController.js
 models
 node_modules
 routes
 index.js
 package-lock.json
 package.json
 gen
 demo1.py

userController.js
api > controllers > UserController.js ...
17 export const login = async (req, res) => {
18 const user = await User.findOne({ email: req.body.email });
19 if (!user) {return res.json({ msg: "User not found", success: false })};
20 // password is user define password and user.password is db password bcrypt and match
21 const validPass = await bcrypt.compare(password, user.password);
22 if (!validPass) {return res.json({ msg: "Invalid password", success: false })};
23 // Create a JWT token with user details and 1-day expiry
24 // import jwt from "jsonwebtoken";
25 const token = jwt.sign(
26 { user_all_data: user },
27 { user_name: user.name, user_email: user.email },
28 { expiresIn: "1d" }
29);
30 // Store token in cookie
31 res.cookie("token", token, {
32 httpOnly: true, // cannot be accessed by JS
33 secure: true, // use HTTPS in production
34 maxAge: 24 * 60 * 60 * 1000 // 1 day in ms
35 // maxAge: 7 * 24 * 60 * 60 * 1000 // valid for 1 week
36 // maxAge: 365 * 24 * 60 * 60 * 1000 // 1 year in ms
37 });
38 // Send response
39 res.json({ msg: `Welcome to our API ${user.name}`, token, success: true });
40 } catch (error) {console.error("Login error:", error)};
41 res.status(500).json({ error: "Login failed" });
42}
43
44
45

```

Thunder clients



POST <http://localhost:3000/api/user/login> Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

```

1 {
2 "email": "rdsir@example.com",
3 "password": "RD@12345"
4 }
5

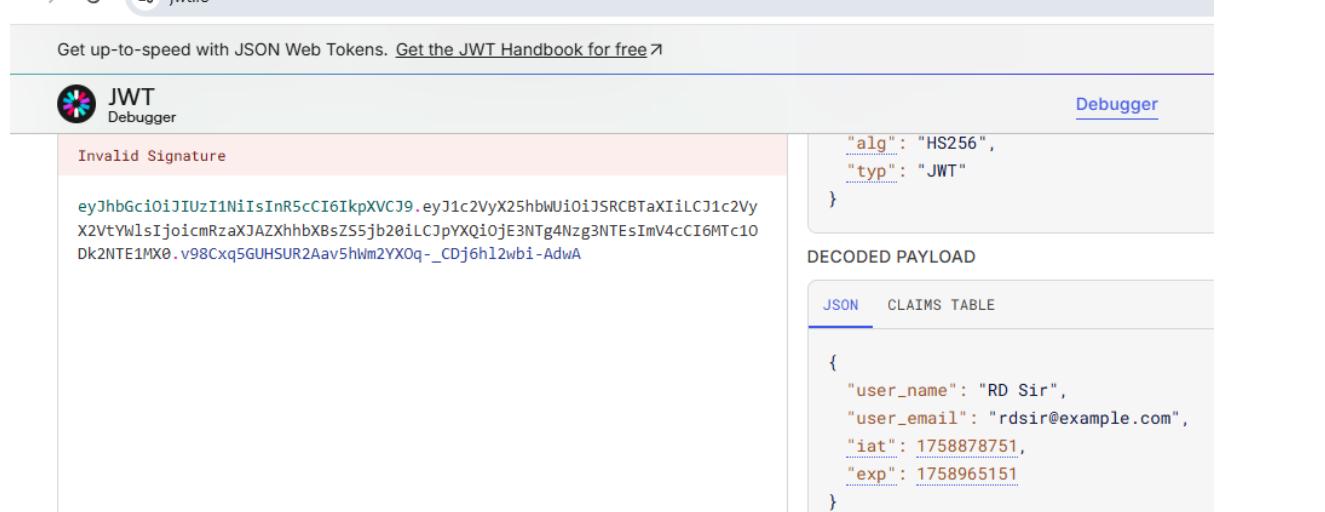
```

Status: 200 OK Size: 0 Bytes Time: 373 ms Response

```

1 {
2 "msg": "Welcome to our API RD Sir",
3 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJSRCBTaXiilCJ1c2VyX2VtYWlsIjoicmRzaXJAZXhhbXBsZS5jb20iLCJpYXQiOjE3NTg4Nzg3NTEsImV4cCI6MTc1ODk2NTE1MX0.v98Cxq5GUHSUR2Aav5hWm2YXOq-_CDj6h12wbi-AdwA",
4 "success": true
5 }

```



JWT Debugger

Invalid Signature

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJSRCBTaXiilCJ1c2VyX2VtYWlsIjoicmRzaXJAZXhhbXBsZS5jb20iLCJpYXQiOjE3NTg4Nzg3NTEsImV4cCI6MTc1ODk2NTE1MX0.v98Cxq5GUHSUR2Aav5hWm2YXOq-\_CDj6h12wbi-AdwA

Debugger

```

"alg": "HS256",
"typ": "JWT"
}

```

DECODED PAYLOAD

JSON CLAIMS TABLE

```

{
 "user_name": "RD Sir",
 "user_email": "rdsir@example.com",
 "iat": 1758878751,
 "exp": 1758965151
}

```



• RID BHARAT

Page. No:169 Website: [www.ridtech.in](http://www.ridtech.in)

**Example:**

```
// controllers/userController.js
import User from "../models/User.js";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
// Register User (same as before)
export const registerUser = async (req, res) => {
 try {
 const { name, email, password } = req.body;
 if (!name || !email || !password) {return res.json({ msg: "All fields are required" })};
 const existingUser = await User.findOne({ email });
 if (existingUser) {return res.json({ msg: "User already exists" })};
 const hashedPassword = await bcrypt.hash(password, 10);
 const newUser = new User({ name, email, password: hashedPassword });
 await newUser.save();
 res.json({ msg: "User registered successfully", user: newUser });
 } catch (error) {
 console.error("Error saving user:", error);
 res.status(500).json({ error: "Failed to register user" });
 }
// Login User with JWT + Cookie
export const login = async (req, res) => {
 try {
 const { email, password } = req.body;
 if (!email || !password) { return res.json({ msg: "All fields are required" })};
 const user = await User.findOne({ email });
 if (!user) { return res.json({ msg: "User not found", success: false }) };
 const validPass = await bcrypt.compare(password, user.password);
 if (!validPass) {return res.json({ msg: "Invalid password", success: false })};
 // Create JWT token
 const token = jwt.sign(
 { id: user._id, email: user.email }, // payload
 "MY_SECRET_KEY", // secret key
 { expiresIn: "1h" } // validity
);
 // Save token in cookie
 res.cookie("authToken", token, {
 httpOnly: true, // not accessible via JS (secure)
 secure: false, // set true if using https
 maxAge: 60 * 60 * 1000 // 1 hour
 });
 res.json({ msg: `Welcome ${user.name}`, success: true });
 } catch (error) {
 console.error("Login error:", error);
 res.status(500).json({ error: "Login failed" });
 }
};
```

## **6. Middleware to Protect Routes**

Later, if you want to protect routes, add:

```
import jwt from "jsonwebtoken";
export const verifyToken = (req, res, next) => {
 const token = req.cookies.authToken;
 if (!token) {return res.status(401).json({ msg: "Access denied, no token provided" });}
 try {
 const decoded = jwt.verify(token, "MY_SECRET_KEY");
 req.user = decoded; // store decoded user data
 next();
 } catch (err) { res.status(400).json({ msg: "Invalid token" });}
```

Use in routes:

```
router.get("/dashboard", verifyToken, (req, res) => {
 res.json({ msg: `Hello ${req.user.email}, welcome to dashboard` });
});
```

**In short:**

1. Login → create JWT → save in cookie
2. Future requests → read cookie → verify JWT → allow/deny



## ❖ What is Session Management in Node.js (Express)?

### What is Session Management in Node.js (Express)?

- **Session Management** is a way to store user-specific data on the server for use across multiple requests.
- For example, when a user logs in, you can store their user ID or name in a session so that they stay logged in across different pages until they log out.

### Why Session Management is Needed

- To **maintain user login state**.
- To **store temporary user data** (like shopping cart items).
- To **identify** which user is making the request.

### How Sessions Work in Express

1. When a user logs in → a **session** is created on the server.
2. A unique **Session ID** is sent to the user as a **cookie**.
3. On every next request, this **Session ID** is used to find the user's session data.

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

```
1 import express from "express";
2 import session from "express-session";
3 const app = express();
4 app.use(express.urlencoded({ extended: true })); // Configure Session Middleware
5 app.use(session({
6 secret: "mysecretkey", // Used to sign the session ID cookie
7 resave: false, // Don't save session if unmodified
8 saveUninitialized: true, // Save new sessions
9 cookie: { maxAge: 60000 } // Session expires after 1 minute
0 })); // Home Route
1 app.get("/", (req, res) => {
2 if (req.session.username) {
3 res.send(`<h2>Welcome back, ${req.session.username}!</h2>
4 Logout`);
5 } else { res.send(`
6 <h2>Login Form</h2>
7 <form method="POST" action="/login">
8 <input name="username" placeholder="Enter Username" required />
9 <button type="submit">Login</button>
0 </form>`);
1 })); // Login Route
2 app.post("/login", (req, res) => {
3 const { username } = req.body;
4 req.session.username = username; // Store username in session
5 res.redirect("/"); // Logout Route
6 app.get("/logout", (req, res) => {
7 req.session.destroy(err => {
8 if (err) { return res.send("Error logging out");}
9 res.clearCookie("connect.sid"); // Remove session cookie
0 res.redirect("/");
1 });}); // Start Server
2 app.listen(3000, () => {console.log("Server started on http://localhost:3000");});
```



## Import Interview Question and answer

### 1. What is Node.js?

Node.js is a **JavaScript runtime environment** built on Chrome's **V8 engine** that lets you run JavaScript code on the server side.

### 2. Why use Node.js for backend development?

Because it is **fast, scalable, non-blocking**, and supports **real-time applications** using asynchronous I/O.

### 3. What is the difference between JavaScript and Node.js?

- **JavaScript**: Runs in browsers (client-side).
- **Node.js**: Runs outside browsers (server-side).

### 4. What is the role of the V8 engine in Node.js?

V8 converts JavaScript code into **machine code**, enabling high performance and fast execution.

### 5. What is npm?

**npm (Node Package Manager)** manages dependencies (libraries) used in Node.js projects.

### 6. What is Express.js?

Express.js is a **web application framework for Node.js** used to build APIs and web servers easily.

### 7. What is middleware in Express.js?

Middleware are functions that execute **between request and response** — for logging, authentication, etc.

**Example:** `app.use((req, res, next) => { console.log('Middleware'); next(); })`

### 8. What are the core modules in Node.js?

Common modules: http, fs, path, url, events, os, crypto.

### 9. What is event-driven programming in Node.js?

Node.js uses an **event-driven, non-blocking architecture**, meaning actions trigger events handled by callbacks.

### 10. What is the difference between synchronous and asynchronous code?

- **Synchronous**: Executes one line at a time (blocking).
- **Asynchronous**: Doesn't block — other code runs while waiting.

### 11. What is a callback function?

A function passed as an argument to another function to be executed later.

### 12. What are Promises and async/await?

- **Promise**: Handles async tasks and returns success or error.
- **async/await**: Cleaner syntax to handle Promises.

### 13. How do you create a simple server in Node.js?

```
const http = require('http');
http.createServer((req, res) => res.end("Hello")).listen(3000);
```

### 14. What is package.json?

It contains project metadata and dependencies for a Node.js app.

### 15. What is the difference between require and import?

- **require**: CommonJS module system (Node.js default).
- **import**: ES6 module system (modern JS).

### 16. What is Nodemon?

A tool that automatically restarts your Node.js server when file changes are detected.

### 17. What is REST API?

REST (Representational State Transfer) is an architectural style that uses HTTP methods like GET, POST, PUT, DELETE for CRUD operations.

### 18. How to define routes in Express.js?

```
app.get('/users', (req, res) => res.send('User List'));
```

### 19. How to handle JSON data in Express.js?

Use built-in middleware:

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

**20. What is the difference between res.send() and res.json()?**

- res.send(): Sends any type of response (string, object, etc.)
- res.json(): Sends a JSON response.

**21. How to serve static files in Express.js?**

app.use(express.static('public'));

Used to serve HTML, CSS, JS, images, etc.

**22. How do you handle errors in Express.js?**

Use error-handling middleware:

```
app.use((err, req, res, next) => res.status(500).send(err.message));
```

**23. How to connect MongoDB with Node.js using Mongoose?**

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydb');
```

**24. What is Mongoose?**

Mongoose is an **ODM (Object Data Modeling)** library for MongoDB and Node.js — helps define schemas and interact with MongoDB easily.

**25. How to secure passwords in Node.js?**

Use **bcrypt** to hash passwords:

```
const hash = await bcrypt.hash(password, 10);
```

**26. What is JWT (JSON Web Token)?**

JWT is used for **authentication** — it securely transmits user data between client and server.

**27. How do you handle CORS in Express.js?**

Use CORS middleware:

```
const cors = require('cors');
app.use(cors());
```

**28. What is process.nextTick()?**

It defers the execution of a callback until the next iteration of the event loop.

**29. How to handle file uploads in Node.js?**

Use **Multer** middleware:

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });
app.post('/upload', upload.single('file'), (req, res) => res.send('Uploaded'));
```

**30. What is MVC architecture in Express.js?**

- **Model:** Handles data & database logic
- **View:** UI templates (EJS, Pug)
- **Controller:** Business logic & routing

**31. What is the Event Loop in Node.js?**

The **event loop** handles all asynchronous callbacks. It allows Node.js to perform non-blocking I/O operations even though JavaScript is single-threaded.

**32. What are Streams in Node.js?**

Streams are used to **read/write data continuously** (like handling large files).

Types: Readable, Writable, Duplex, Transform.

**33. What is the difference between readFile and createReadStream?**

- **readFile:** Loads the entire file into memory.
- **createReadStream:** Reads file chunk by chunk (memory efficient).

**34. How does Node.js handle child processes?**

Using the **child\_process** module:

```
const { exec } = require('child_process');
exec('dir', (err, stdout) => console.log(stdout));
```

**35. What are environment variables in Node.js?**

Used to store sensitive information like API keys or DB credentials.

Example: `process.env.PORT = 3000;`

**36. How to manage environment variables securely?**

Use `dotenv`:

```
require('dotenv').config();
```

```
const port = process.env.PORT;
```

**37. What is clustering in Node.js?**

Clustering allows Node.js to use **multiple CPU cores** for better performance using the `cluster` module.

**38. What is CORS and why is it used?**

**CORS (Cross-Origin Resource Sharing)** allows browsers to request resources from different origins (domains).

Handled via: `app.use(require('cors')());`

**39. Difference between PUT and PATCH in REST API?**

- **PUT:** Replaces the entire resource.
- **PATCH:** Updates only specific fields.

**40. What is rate limiting and why use it?**

To prevent API abuse by limiting requests per user/IP using packages like `express-rate-limit`.

**41. How do you handle authentication in Express.js?**

Using `JWT` or `Passport.js` for session or token-based authentication.

**42. How to implement session management?**

Using `express-session`:

```
const session = require('express-session');
```

```
app.use(session({ secret: 'secretKey', resave: false, saveUninitialized: true }));
```

**43. What are cookies and how to set them in Express.js?**

Cookies store small data on the client side.

```
res.cookie('username', 'Tanmay', { maxAge: 60000 });
```

**44. What are WebSockets in Node.js?**

Used for **real-time, two-way communication** (e.g., chat apps) using the `ws` or `socket.io` library.

**45. What are CommonJS and ES Modules?**

- **CommonJS:** Uses `require()` and `module.exports`.
- **ES Modules:** Uses `import` and `export`.

**46. How to handle 404 errors in Express.js?**

```
app.use((req, res) => res.status(404).send('Page Not Found'));
```

**47. What is the difference between `res.sendFile()` and `res.render()`?**

- `res.sendFile()`: Sends static files.
- `res.render()`: Renders templates (EJS, Pug, etc.) with dynamic data.

**48. How to validate user input in Node.js?**

Use validation libraries like `Joi` or `express-validator`:

```
const { body } = require('express-validator');
```

```
app.post('/user', body('email').isEmail(), (req, res) => {...});
```

**49. What is the purpose of `app.use()` in Express.js?**

It mounts middleware or routes at the application level — runs for every request that matches the path.

**50. How to deploy a Node.js application?**

- **Steps:**
  1. Push code to GitHub
  2. Use platforms like **Render**, **Vercel**, or **AWS EC2**
  3. Run `npm install` and `npm start`
  4. Use process managers like **PM2** for uptime management

# What is RID Organization (RID संस्था क्या है)?

- **RID Organization** यानि **Research, Innovation and Discovery Organization** एक संस्था हैं जो TWF (TWKSAA WELFARE FOUNDATION) NGO द्वारा RUN किया जाता है | जिसका मुख्य उद्देश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही इस **RID Organization** की स्थपना 30.09.2023 किया गया है | जो TWF द्वारा संचालित किया जाता है |
- TWF (TWKSAA WELFARE FOUNDATION) NGO की स्थपना 26-10-2020 में बिहार की पावन धरती सासाराम में Er. RAJESH PRASAD एवं Er. SUNIL KUMAR द्वारा किया गया था जो की भारत सरकार द्वारा मान्यता प्राप्त संस्था हैं |
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियों से मैं आवाहन करता हूँ की आप सभी इस **RID संस्था** से जुड़ें एवं अपने बुधिद्वय, विवेक एवं प्रतिभा से दुनियां को कुछ नई (**RID, PMS & TLR**) की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

| MISSION, VISSION & MOTIVE OF “RID ORGANIZATION” |                                                                        |
|-------------------------------------------------|------------------------------------------------------------------------|
| मिशन                                            | हर एक ONE भारत के संग                                                  |
| विजन                                            | TALENT WORLD KA SHRESHTM AB AAYEGA भारत में और भारत का TALENT भारत में |
| मक्षद                                           | NEW (RID, PMS, TLR)                                                    |

## MOTIVE OF RID ORGANIZATION NEW (RID, PMS, TLR)

### NEW (RID)

|          |            |           |
|----------|------------|-----------|
| R        | I          | D         |
| Research | Innovation | Discovery |

### NEW (TLR)

|                               |     |      |
|-------------------------------|-----|------|
| T                             | L   | R    |
| Technology, Theory, Technique | Law | Rule |

### NEW (PMS)

|                              |         |         |
|------------------------------|---------|---------|
| P                            | M       | S       |
| Product, Project, Production | Machine | Service |



RID रीड संस्था की मिशन, विजन एवं मक्षद को सार्थक हमें बनाना हैं |  
भारत के वर्चस्व को हर कोने में फैलना हैं |  
कर के नया कार्य एक बदलाव समाज में लाना हैं |  
रीड संस्था की कार्य-सिद्धांतों से ही, हमें अपनी पहचान बनाना हैं |

Er. Rajesh Prasad (B.E, M.E)  
Founder: TWF & RID Organization

React js , Node js, Express js के इस E-Book में अगर मिलती त्रुटी  
मिलती है तो कृपया हमें सूचित करें | WhatsApp's No: 9202707903 or  
Email Id: [ridorg.in@gmail.com](mailto:ridorg.in@gmail.com)

|| धन्यवाद ||

