## Page:1

# Server-Side Rendering

**What is Server-Side Rendering (SSR)?**

**Server-Side Rendering (SSR)** is the process where the server generates the full HTML for a webpage and sends it to the client (browser). The browser then renders the HTML and displays the page to the user. In Node.js, SSR is commonly used to pre-render web pages before sending them to the client.

**Server-Side Rendering in Node.js**

In **Node.js**, SSR can be done using templating engines like **EJS**, **Pug**, or without any templating engine by manually rendering HTML.

**What is EJS?**

**EJS (Embedded JavaScript Templates)** is a simple templating engine that lets you generate HTML markup with plain JavaScript. It is often used in Node.js for rendering HTML pages with dynamic content on the server side.

**How to Set Up EJS in Express.js**

1. **Install Express and EJS**:
2. bash
3. Copy code
4. npm install express ejs
5. **Set Up Express with EJS**:
6. js
7. Copy code
8. const express = require('express');
9. const app = express();
10. // Set the view engine to EJS
11. app.set('view engine', 'ejs');
12. // Define a route
13. app.get('/', (req, res) => {
14. const data = { title: 'Home Page', message: 'Welcome to Server-Side Rendering with EJS!' };

## Page:2

1. res.render('index', data); // Render the 'index.ejs' file with data
2. });
3. app.listen(3000, () => {
4. console.log('Server is running on port 3000');
5. });
6. **Create an EJS View File (index.ejs)**: Create an index.ejs file inside the views directory:
7. html
8. Copy code
9. <!DOCTYPE html>
10. <html lang="en">
11. <head>
12. <meta charset="UTF-8">
13. <meta name="viewport" content="width=device-width, initial-scale=1.0">
14. <title><%= title %></title>
15. </head>
16. <body>
17. <h1><%= message %></h1>
18. </body>
19. </html>

To access the message variable in your index.ejs file without explicitly doing data.message, you need to pass the object properties directly in your res.render() method. When you pass the object in res.render(), the properties of that object will be directly accessible in the EJS template.

**Example: Passing Data Without data.message**

1. **Express Code (index.js)**: Instead of passing an object like { data: data }, you can pass the individual properties of the object directly to res.render().
2. js
3. Copy code

## Page:3

1. const express = require('express');
2. const app = express();
3. app.set('view engine', 'ejs');
4. // Route with dynamic data
5. app.get('/', (req, res) => {
6. const title = 'EJS Example';
7. const message = 'Welcome to Server-Side Rendering with EJS!';
8. // Pass individual properties instead of a single object
9. res.render('index', { title, message });
10. });
11. app.listen(3000, () => {
12. console.log('Server is running on port 3000');
13. });
14. **EJS Template (index.ejs)**: Now you can directly access message and title in the template without having to do data.message or data.title.
15. html
16. Copy code
17. <!DOCTYPE html>
18. <html lang="en">
19. <head>
20. <meta charset="UTF-8">
21. <meta name="viewport" content="width=device-width, initial-scale=1.0">
22. <title><%= title %></title>
23. </head>

## Page:4

1. <body>
2. <h1><%= message %></h1>
3. </body>
4. </html>

**Explanation:**

* In the res.render('index', { title, message }) part, we are passing title and message as separate properties.
* In the EJS file, you can access them directly as <%= title %> and <%= message %>.

By passing the properties individually, you don't have to access them as data.message or data.title. Instead, they are directly available in your EJS file.

**Example of Server-Side Rendering**

**Without EJS (Manual HTML Rendering):**

js

Copy code

const express = require('express');

const app = express();

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

const html = `

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home Page</title>

</head>

<body>

<h1>Welcome to Server-Side Rendering without EJS!</h1>

## Page:5

</body>

</html>

`;

res.send(html); // Send raw HTML to the client

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

**With EJS (Templating Engine Rendering):**

js

Copy code

const express = require('express');

const app = express();

// Set view engine to EJS

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

// Route with dynamic data

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

const data = {

title: 'EJS Example',

message: 'Welcome to Server-Side Rendering with EJS!'

};

res.render('index', data); // Render 'index.ejs' with dynamic data

});

app.listen(3000, () => {

## Page:6

console.log('Server is running on port 3000');

});

**Advantages of Server-Side Rendering**

1. **SEO Optimization**: Since the HTML is rendered on the server and sent to the browser, it is more accessible to search engines, improving SEO.
2. **Faster Initial Load**: Users receive a fully rendered page from the server, making the initial page load faster, especially for users on slow devices or connections.
3. **No JavaScript Required on Client**: Even if the client’s browser does not support JavaScript, they can still view the fully rendered HTML content.
4. **Better User Experience**: Users see content sooner, improving their perception of site performance.
5. **Easier to Debug**: Server-side rendering ensures that you can easily inspect the generated HTML, which can simplify debugging.

In conclusion, **SSR** is useful for performance, SEO, and better user experience. **EJS** makes SSR in Node.js convenient by offering a clean way to manage dynamic content in HTML templates.

In EJS (Embedded JavaScript Templates), you can write JavaScript code directly within your HTML file using special tags. Below are the different ways you can include JavaScript inside an .ejs file:

**1. Output Values in EJS**

You can output JavaScript variables or expressions using <%= %>. This will insert the result of the expression into the HTML.

html

Copy code

<h1>Hello, <%= name %>!</h1>

This will render the value of name.

**2. Execute JavaScript Logic**

You can run JavaScript logic using <% %>. This will allow you to execute code without rendering anything into the HTML.

html

Copy code

## Page:7

<% if (loggedIn) { %>

<h1>Welcome back, <%= username %>!</h1>

<% } else { %>

<h1>Welcome, guest!</h1>

<% } %>

In this example, different HTML content is rendered based on whether the user is logged in.

**3. Looping in EJS**

You can use JavaScript loops in EJS to render repeated HTML content.

html

Copy code

<ul>

<% items.forEach(function(item) { %>

<li><%= item %></li>

<% }); %>

</ul>

This will create a list where each item from the items array is rendered as a list element.

**4. Including External JavaScript in EJS**

If you want to include external JavaScript files (e.g., linking a JavaScript file for client-side execution), you can do so as usual in the <script> tag.

html

Copy code

<script src="/path/to/your/script.js"></script>

**5. Writing Inline JavaScript**

You can also write inline JavaScript directly in the <script> tag, which will be executed on the client side.

html

Copy code

## Page:8

<script>

let username = '<%= username %>';

console.log("Welcome " + username);

</script>

**Full Example:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title><%= title %></title>

</head>

<body>

<% if (loggedIn) { %>

<h1>Welcome, <%= username %>!</h1>

<% } else { %>

<h1>Welcome, Guest!</h1>

<% } %>

<ul>

<% items.forEach(function(item) { %>

<li><%= item %></li>

<% }); %>

</ul>

## Page:9

<script>

let username = '<%= username %>';

console.log("Welcome " + username);

</script>

</body>

</html>

**Summary of Tags:**

* <%= %>: Output a value into HTML (with HTML escaping).
* <%- %>: Output unescaped value (without HTML escaping).
* <% %>: Run JavaScript logic without outputting to HTML.

With these techniques, you can effectively include JavaScript in your EJS templates!