**Lec-21-24 EJS(Embedded JavaScript)**

## **Template engines**

## **How to set up EJS in a Node.js application using Express**

## **Passing data to render**

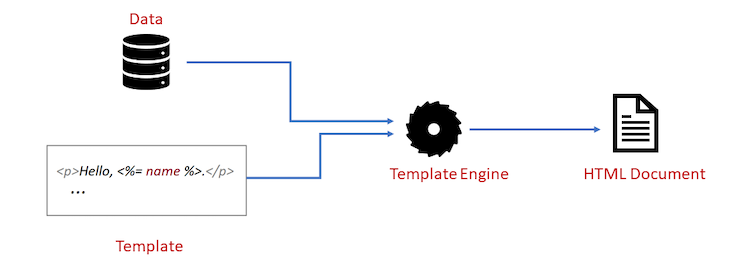
## **Looping through data**

## **EJS partials**

EJS is a simple and direct templating language that enables JavaScript to generate HTML. It's frequently used in Express.js, a Node.js framework for creating web applications, but it can also be used in other JavaScript environments.

## **Template engines**

According to Wikipedia’s definition, a template engine is software designed to combine templates with a data model to produce, in our case, real HTML code.



## Template engines handle the task of interpolating data into HTML code while providing some features (like partials in EJS) that would have been difficult to replicate by concatenating strings.

## **How to set up EJS in a Node.js application using Express**

## Create a new folder where you want to put the project files.

## Step 1: Initialize a new Node project in the folder by running:

## **npm init -y** in the terminal

## step 2: Then to install Express and EJS, run:

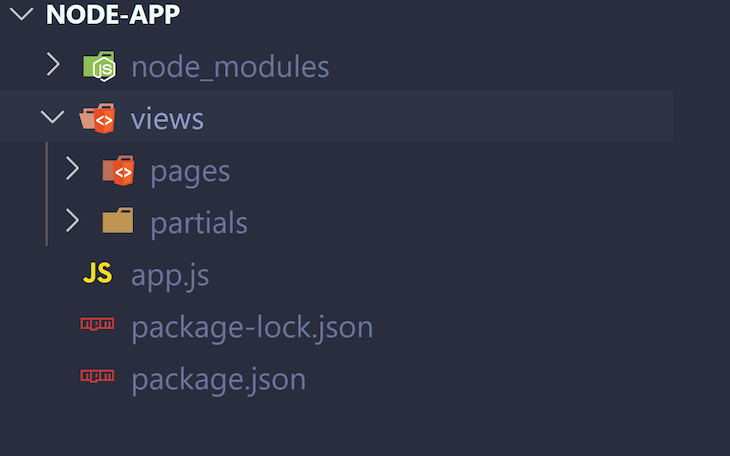
## **npm install -S express ejs**

step 3:After installation, create an

**app.js** file and a **views folder** in the root folder.

Step 4: Inside the views folder,

create **two folders pages and partials.**



Step 5: Now, inside **views/pages** folder

create a file called **index.ejs.**

**app.js**

**const express = require('express')**

**const app = express()**

**const port = 3000**

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

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

**res.render('pages/index')**

**})**

**app.listen(port, () => {**

**console.log(`App listening at port ${port}`)**

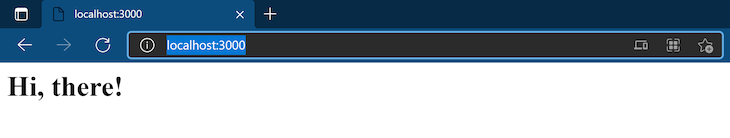
**})**

**Index.ejs**

**<h1>Hi, there!</h1>**

**RUN:-** node app.js on the terminal from the root folder.

**OUTPUT:**



**Explanation:-**

**app.set('view engine', 'ejs')** is self-explanatory. We are setting EJS as the Express app view engine. By default, Express will look inside of a views folder when resolving the template files, which is why we had to create a views folder.

In **res.render('pages/index'),** we are calling the render method on the response object. This renders the view provided (pages/index in this case) and sends back the rendered HTML string to the client.

Note that we didn’t have to provide the file extension because Express resolves it automatically; it knows the view engine we are using from **app.set('view engine', 'ejs').** We also didn’t have to write the path as views/pages/index because the views folder is used by default.

## **Passing data to render**

our aim is to combine data with templates. We can do that by passing a second argument to **res.render**. This second argument must be an object, which will be accessible in the EJS template file. **This start tag <%= is called the “escape output”** tag because if the string in the content has forbidden characters like > and &, the characters will be escaped (replaced by HTML codes) in the output string.

**Update app.js like so:**

**const express = require('express')**

**const app = express()**

**const port = 3000**

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

**const user = {**

**firstName: 'Preenu',**

**lastName: 'Mittan',**

**}**

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

**res.render('pages/index', {**

**user: user**

**})**

**})**

**app.listen(port, () => {**

**console.log(`App listening at port ${port}`)**

**})**

**Update index.ejs too:**

**<h1>Hi, I am <%= user.firstName %></h1>**

**Run node app.js**

**update app.js:**

**const express = require('express')**

**const app = express()**

**const port = 3000**

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

**const user = {**

**firstName: 'Preenu',**

**lastName: 'Mittan',**

**admin: true,**

**}**

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

**res.render('pages/index', {**

**user**

**})**

**})**

**app.listen(port, () => {**

**console.log(`App listening at port ${port}`)**

**})**

**Then update index.js:**

**<h1>Hi, I am <%= user.firstName %></h1>**

**<% if (user.admin) { %>**

**<p>Let me tell you a secret: <b>I am an admin</b></p>**

**<% } %>**

## **Looping through data**

Because the **<% tag** can contain any valid JavaScript code, we can also loop through and display data in EJS. Create a new file inside the **views/pages named articles.ejs.**

**update app.js:**

**const express = require('express')**

**const app = express()**

**const port = 3000**

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

**const posts = [**

**{title: 'Title 1', body: 'Body 1' },**

**{title: 'Title 2', body: 'Body 2' },**

**{title: 'Title 3', body: 'Body 3' },**

**{title: 'Title 4', body: 'Body 4' },**

**]**

**const user = {**

**firstName: 'Tim',**

**lastName: 'Cook',**

**}**

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

**res.render('pages/index', {**

**user**

**})**

**})**

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

**res.render('pages/articles', {**

**articles: posts**

**})**

**})**

**app.listen(port, () => {**

**console.log(`App listening at port ${port}`)**

**})**

**updated articles.ejs**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

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

**<title>Articles</title>**

**</head>**

**<body>**

**<ul>**

**<% articles.forEach((article)=> { %>**

**<li>**

**<h2>**

**<%= article.title %>**

**</h2>**

**<p>**

**<%= article.body %>**

**</p>**

**</li>**

**<hr />**

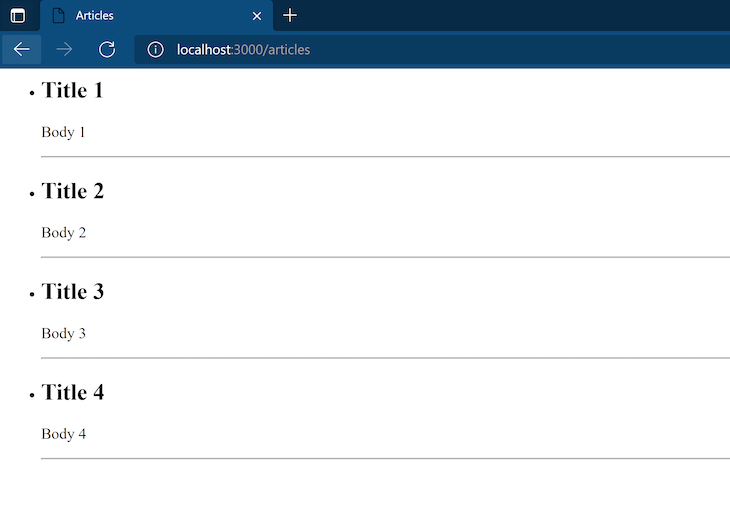
**<% }) %>**

**</ul>**

**</body>**

**</html>**

**Run the app, : node app.js & visit http://localhost:3000/articles**



**Explanation:-**

We passed articles which is an array of post objects containing a title and a body to the **articles.ejs template**. Then, in the template, we loop through the array using forEach to render each post object as a list item.

Notice that the article variable that references each item of the array on each iteration of the loop **<% articles.forEach((article)=> { %>** is accessible in other portions of the template code until we reach the end of the closing brackets **<% }) %>.**

## **EJS partials**

Some parts of websites stay the same across different pages, like the header, footer, and sidebar. EJS provides us with partials that allow us to reuse views.

Recall that we **created the views/partials folder** earlier. **Create two new files named head.ejs and footer.ejs in this folder.**

**head.ejs should be the following:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

<title>Articles</title>

</head>

**update footer.ejs like so:**

</html>

<div class="container">

<footer class="d-flex flex-wrap justify-content-between align-items-center py-3 my-4">

<p class="col-md-4 mb-0 text-muted">&copy; 2021 Simple Blog</p>

<ul class="nav col-md-4 justify-content-end">

<li class="nav-item"><a href="/" class="nav-link px-2">Home</a></li>

<li class="nav-item"><a href="/articles" class="nav-link px-2">Articles</a></li>

<li class="nav-item"><a href="#" class="nav-link px-2">About</a></li>

</ul>

</footer>

</div>

**articles.ejs:**

**<%- include('../partials/head') %>**

**<body><ul>**

**<% articles.forEach((article)=> { %>**

**<li>**

**<h2>**

**<%= article.title %>**

**</h2>**

**<p>**

**<%= article.body %>**

**</p>**

**</li>**

**<hr />**

**<% }) %>**

**</ul></body>**

**<%- include('../partials/footer') %>**

**Explanation:**

I have included **the head.ejs and footer.ejs partials** using the include function. It takes in the relative path to the file as an argument. Because pages and partials are in the same folder, to access partials from pages, we have to first go out of the pages folder **(../partials/head').**

Also, take note of the EJS tag used **(<%-)** instead of the escaped output tag mentioned above **(<%=). <%- is called the “unescaped output” tag**, and is used when you want to output raw HTML.

Make sure to use it with care. Don’t use it with user input, because it can expose your application to attacks.

**Let’s create another partial named header.ejs inside of views/partials**

**header.ejs:**

**<header class="p-3 bg-dark text-white mb-4">**

**<div class="container">**

**<div class="d-flex flex-wrap align-items-center justify-content-center justify-content-lg-start">**

**<ul class="nav col-12 col-lg-auto me-lg-auto mb-2 justify-content-center mb-md-0">**

**<li><a href="/" class="nav-link px-2 text-white">Home</a></li>**

**<li><a href="/articles" class="nav-link px-2 text-white">Articles</a></li>**

**<li><a href="#" class="nav-link px-2 text-white">About</a></li>**

**</ul>**

**<form class="col-12 col-lg-auto mb-3 mb-lg-0 me-lg-3">**

**<input type="search" class="form-control form-control-dark" placeholder="Search..." aria-label="Search">**

**</form>**

**<div class="text-end">**

**<button type="button" class="btn btn-outline-light me-2">Login</button>**

**<button type="button" class="btn btn-warning">Sign-up</button>**

**</div>**

**</div>**

**</div>**

**</header>**

**Update articles.ejs to this:**

**<%- include('../partials/head') %>**

**<%- include('../partials/header') %>**

**<body>**

**<ul>**

**<% articles.forEach((article)=> { %>**

**<li>**

**<h2>**

**<%= article.title %>**

**</h2>**

**<p>**

**<%= article.body %>**

**</p>**

**</li>**

**<hr />**

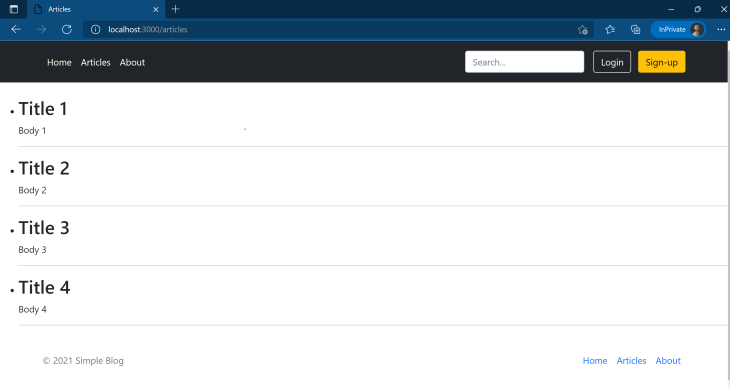
**<% }) %>**

**</ul>**

**</body>**

**<%- include('../partials/footer') %>**

**Run :- node app.js & visit http://localhost:3000/articles and you should see this:**



**update index.ejs to include the partials:**

**<%- include('../partials/head') %>**

**<%- include('../partials/header') %>**

**<div class="container">**

**<h1>Hi, I am <%= user.firstName %> <%= user.lastName %></h1>**

**<h2>Welcome to my Blog</h2>**

**</div>**

**<%- include('../partials/footer') %>**

**Run the app again and you can see the new index page at** [**http://localhost:3000/**](http://localhost:3000/)

