

Introduction

In this tutorial, you will see how the EJS templating engine is brought into your Node.js application to create dynamic page content, and modularize the code base. You will be creating a simple application that shows a list of reminders. That list is rendered using EJS, to selectively display items in a list, and pass values dynamically to the display.

The important things to take from this tutorial are,

1. How to create HTML content with EJS
2. How to implement conditionals with EJS
3. How to implement loops with EJS
4. How to transfer content to EJS templates

EJS syntax takes a bit of getting used to, but the templating structure is not sophisticated. You will see

- `<%=`
- `%>`
- `<%-`

These form part of the protocol of transferring values to variables, and indicating the beginning and ending of a process (almost like a function).

1. Getting started – folder structure, package.json, installs

Project Initialization

Create a new folder, ejs-sampler, or other.

Use the ***npm init -y*** command to initialize the folder with a **package.json** file.

Remember, **package.json** is a key file in modern software development because it keeps track of all the packages that the application depends on, and stores other meta-data, like any repositories.

➤ **npm init -y**

Take note of the main file, like index.js, server.js. Here, we have used the title app.js.

Organization, Folder Structure

In any significant software development effort, content should be organized so that it can be easily maintained. Professional software developers expect a ‘folder protocol’ in which a key concept in software engineering is demonstrated. That key concept is **Separation of Concerns**. Creating separate folders for different parts of the application, supports Separation of Concerns.

Here, Separation of Concerns is about keeping presentation (the user interface) separate from any business logic, and separate from any routing logic.

Now create the following folder structure

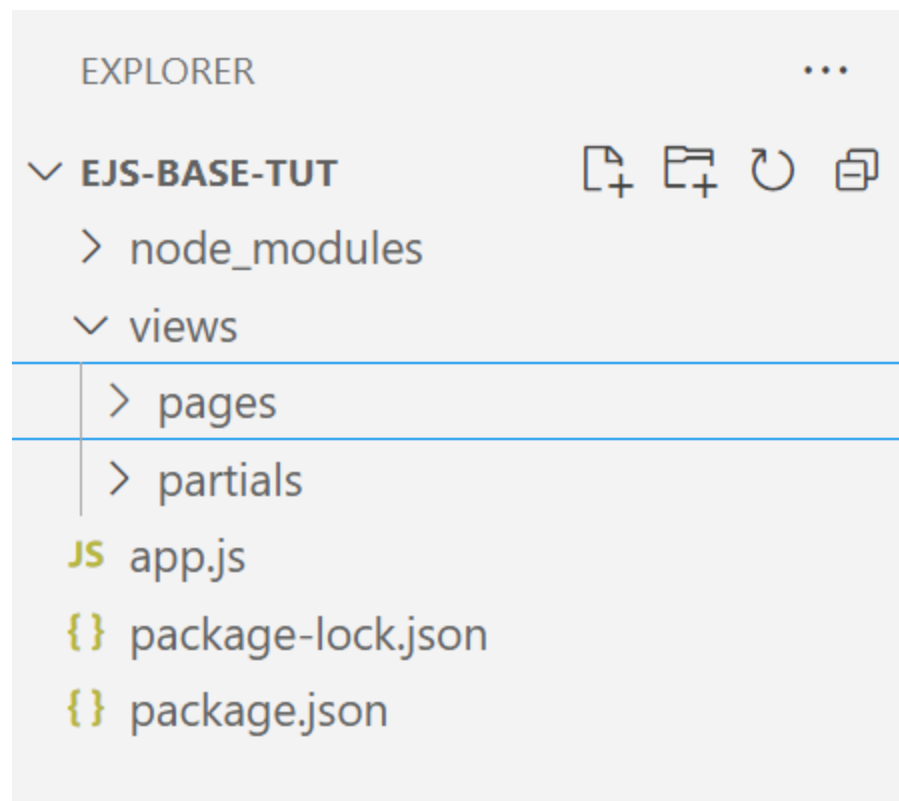
The **views** folder is one that keeps all of the content to be rendered and displayed on the client, here this is our browser, and the content at localhost. This folder has 2 sub-folders, the pages folder and the partials folder.

The **pages** folder will contain files with an extension of .ejs (Embedded JavaScript) – these files support the display of dynamic

content and are recognized once EJS is installed as a dependency.

The **partials** folder will contain segments of HTML with EJS, that appear in multiple pages, and that can be reused, like a function.

You will be adding files to this folder structure to create a simple application that lists notes, reminders about meetings (just like in the 212, the 215 and the 312 – I’m low on material...).



Install dependencies, support ES6

This application needs **express** and **ejs**, and both of these can be installed as dependencies with **npm install**. It is a good idea to use the **-save** flag, which ensures that the exact version of each is saved to package.json. Note, no need for comma between the packages, and multiple packages can be installed together – just separate with a space.

➤ **npm install express ej**s

Optionally, install **nodemon** (use the -g flag), and add start script,

“start” : “node app.js”

With the start script specified, you can use

➤ **npm start**

... to start your application.

Important! To add support for the ES6 import statements, add the “type” specification to your package.json file.



```
{ } package.json × <> index.ejs JS app.js <>
{ } package.json > ...
1 {
2   "name": "ejs-base-tut",
3   "type": "module",
4   "version": "1.0.0",
5   "description": "Starter EJS Tutorial",
6   "main": "app.js",
```

2. Server Code, Creation of ‘home page’ route

Create the server code, using ES6 protocols.

We want to render an **.ejs** file, not as yet created! This is what the **pages/index** reference is below, in the res.render call.

```

JS app.js > ...
1  import express from 'express'; // Bring in express, import (ES6)
2  const app = express();          // Create an instance of Express
3  const port = 3000;              // Use port 3000
4
5  // Set the view engine with app.set,
6  // Express loads the module internally and stores it in app reference
7  app.set('view engine', 'ejs')   //
8
9  // Set the landing page route
10 // Send the index.ejs file in the pages folder, to client
11 // File will have an extension of .ejs, Embedded JavaScript
12 app.get('/', (req, res) => {
13 |   res.render('pages/index')
14 | });
15
16 // Listen for requests
17 app.listen(port, () => {
18 |   console.log(`App listening at port ${port}`)
19 | });

```

This is a basic server setup with Node and Express.

Later, we will need to resolve path names in support of ES6. So, we need to add some middleware. May as well do this now. We will be creating a **public** folder when we get to the styling part of this tutorial.

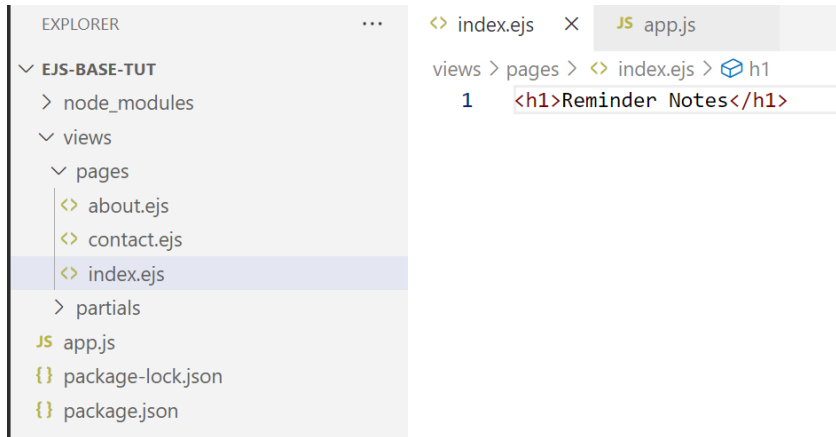
```

3  import express from "express";
4  // imports needed to resolve ES6 file and directory references
5  import path from "path";
6  import { fileURLToPath } from "url";
7
8  const app = express();
9  const port = 3000;
10
11 const __filename = fileURLToPath(import.meta.url); // get the resolved path to the file
12 const __dirname = path.dirname(__filename);        // get the name of the current directory
13 app.use(express.static(__dirname + "/public"));    // make the public folder the default
14
15 app.set("view engine", "ejs");

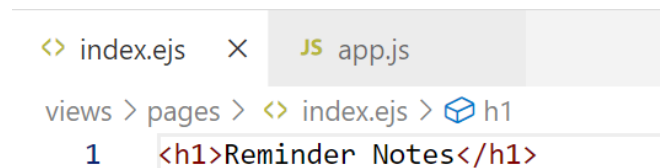
```

3. Create the Views

Now create and edit a file, **index.ejs**, in the **pages** folder – which is a subfolder of **views**. Create 2 others, **about.ejs** and **contact.ejs**, while you are doing this, we will add content later. Here is the folder structure you should have.



This `<h1>` tag is all we need in here for starters, we will change the content to include EJS.



These files are Embedded JavaScript files, and will contain the code we need to render the content. For now, the Express object will serve the `<h1>Reminder Notes</h1>` markup only.



Reminder Notes

This should be rendered then you start the server with,

➤ **node app.js**

... or your start script.

4. Passing data to the home page

Now let's pass data to the index.ejs file!

In the **app.js** file, modify the call to **res.render** for the index page. Create a JavaScript object containing the data to be passed to the .ejs component (index.ejs).

```
let dataToPass = { topic: "CIT Monthly Meeting",
  dateTime: "September 19th 2024, 2pm-3pm",
  location: "Knoy Hall West Lafayette"};
app.get('/', (req, res) => {
  res.render('pages/index', { data : dataToPass });
});
```

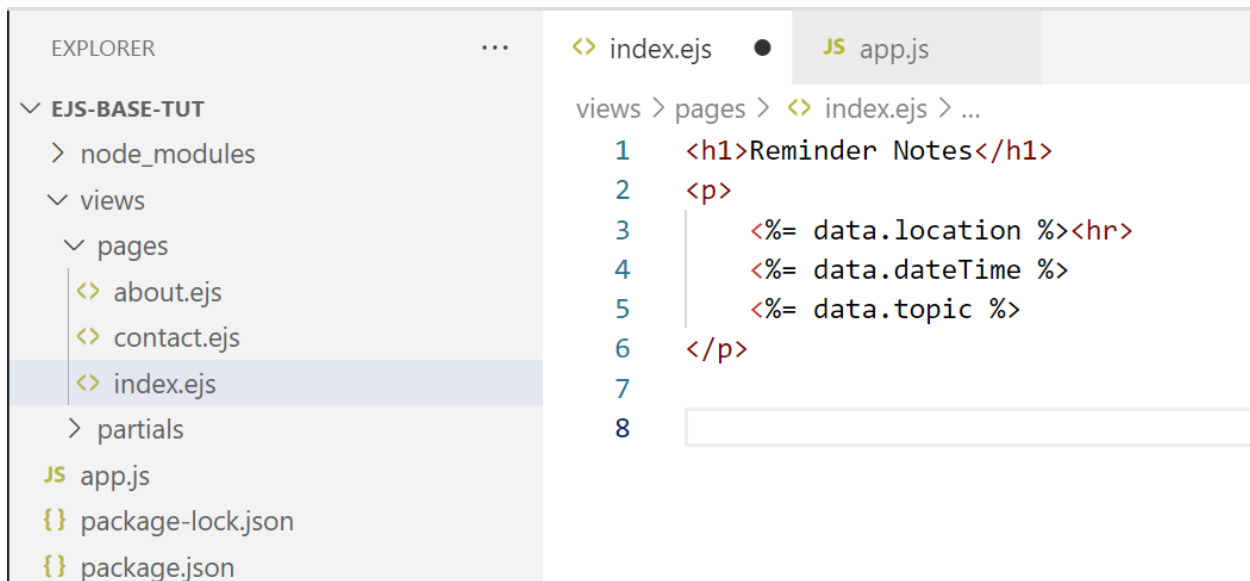
Now, the **render** method transfers the second parameter, which is an object,

{ data : dataToPass }

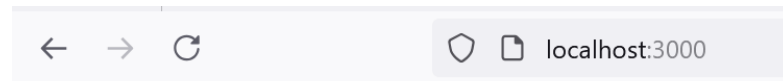
to the .ejs file.

Modify the **index.ejs** file to accept this data.

Below, you see the assignment part of EJS syntax, **<%= %>**



When this is added to the .ejs file, and the file is rendered, you should see the following.



Reminder Notes

Knoy Hall West Lafayette

September 19th 2024, 2pm-3pm CIT Monthly Meeting

This content has been passed from the object, directly to the ***index.ejs*** file.

That's just one object, now let's add a little more data!

We will create an array of objects, simulating a potential set of data delivered from an API, or retrieved from a database. We will modify the carrier object used in ***res.render***, to include an attribute and value pair with the array of objects.

Below, note the array of objects referenced by variable, ***meetings***, represented as JavaScript objects. Note also that another piece of data is passed to the .ejs file, in the form of an attribute and it's value again – this time title, and “Scheduled Meetings”.

```

15 let meetings = [
16     { topic: "CIT Monthly Meeting", dateTime: "September 19th 2024, 2pm-3pm", location: "Knoy Hall West Lafayette"},
17     { topic: "Research In Higher Level Ed", dateTime: "September 24th 2024, 1pm-5pm", location: "Beresford Building, Room 2, West Lafayette"},
18     { topic: "Curriculum Planning", dateTime: "October 19th 2024, 4pm-6pm", location: "I0240, Indianapolis"}
19 ];
20 app.get('/', (req, res) => {
21     res.render('pages/index', { data : meetings, title : "Scheduled Meetings"});
22 });
23

```

Transferring List Content To Views, Using EJS

The ***data*** attribute's value (from the carrier object) is an array, ***meetings***. The ***title*** attribute's value is a string.

EJS uses the ***<%= ... %>*** syntax to associate a value with a tag.

To **iterate** through items in the JavaScript array, we can use the JavaScript array method, **forEach**, with associated anonymous function to handle each of the array objects. Note that a regular line of JavaScript code is encased in `<% %>`

```
<% yourList.forEach( (item)=> { %>
```

```
  <p>
```

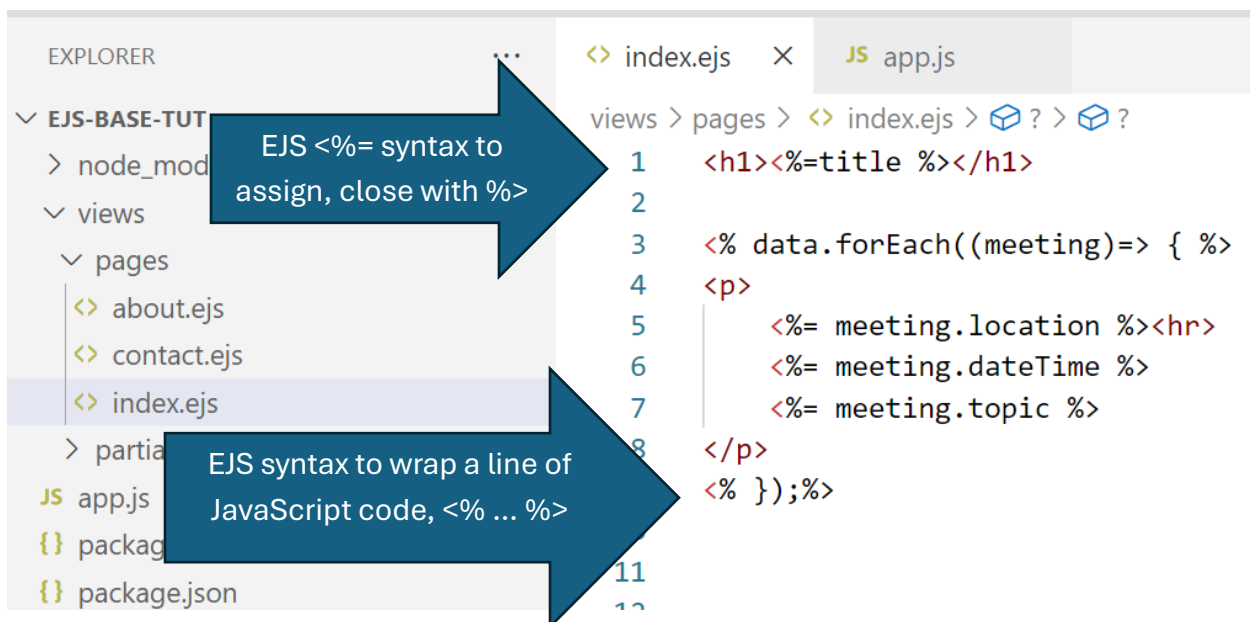
```
    .....
```

```
    <%= item.requiredAttribute %>
```

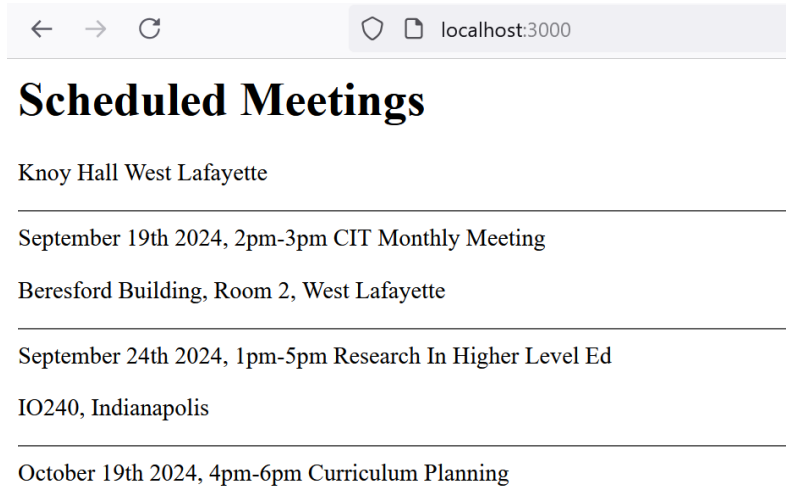
```
    .....
```

```
  </p>
```

```
<% }); %>
```



When you render the code so far, you should see this.



5. Handling Conditional Logic with EJS

Now let's add another field to the objects being passed to index.ejs, so that the meeting is only displayed if it is mandatory.

```
let meetings = [  
  { topic: "CIT Monthly Meeting", mandatory : true, dateTime: "September 19th 2024, 2pm-3pm", loca  
  { topic: "Research In Higher Level Ed", mandatory : false, dateTime: "September 24th 2024, 1pm  
  { topic: "Curriculum Planning", mandatory : true, dateTime: "October 19th 2024, 4pm-6pm", loca  
];
```

The **mandatory** attribute can have a Boolean value - true or false. Let's use this to show how conditionals are supported with EJS.

Modify **index.js** as follows, to include the conditional. Only if the meeting mandatory field is set to true, display the string "Attendance Mandatory!".

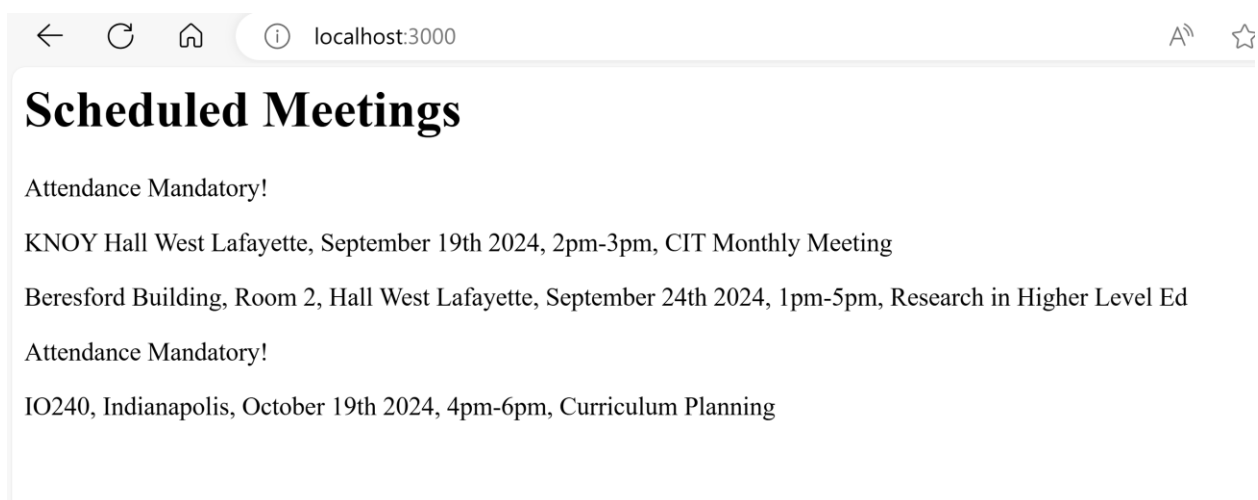
```
<h1><%= title %></h1>
<% data.forEach((meeting)=>{ %> <% if (meeting.mandatory) { %> <%
console.log(meeting.mandatory) %>
<span>Attendance Mandatory!</span>
<% } %>
<p><%= meeting.location %>, <%= meeting.dateTime %>, <%= meeting.topic %></p>
<% }) %>
```

<% if (condition) { %>

.....

<% } %>

You will see



6. Using Partial

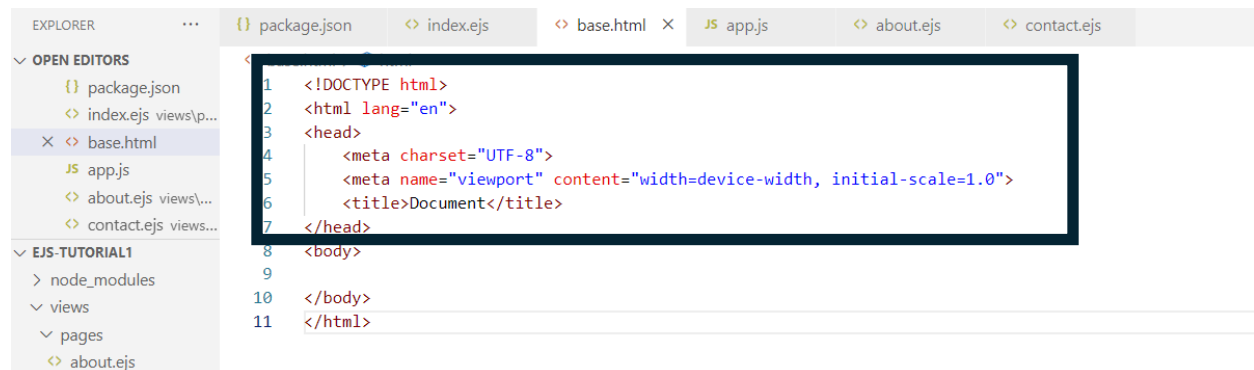
Partials represent EJS/HTML content that is common to several pages, and can be stored in a file and reused (included) across all of these pages. The content in a **partial** represents a segment of the markup in the application, and can be included in multiple files. This supports reuse, reduces the code base, and adds uniformity to the appearance of the application. It also means that the code base is easier to maintain, as only the code in the **partial** needs to be changed, if a change is required.

We left home page without a head, and without a footer – so let's use partials to include these.

In the partials folder, create some additional files:

- **head.ejs**, to contain the head part of the document
- **footer.ejs**, to contain the footer part of the document

Let's start with **head.ejs**, the head part of the document. This should be plain HTML, as would be populated if you used the Emmet abbreviations in VS Code.



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor view on the right. The Explorer sidebar shows the project structure with folders like 'node_modules', 'views', and 'pages'. The Editor view shows the content of 'head.ejs' with the following code:

```

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>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>

```

Set up **head.ejs** to have the head part of the document.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

```

Add content to the footer, via **footer.ejs**

```

{} package.json  <> index.ejs  <> base.html  <> head.ejs  <> menu.ejs  <> footer.ejs  X
views > partials > <> footer.ejs > footer
1  <footer>
2  <p>Purdue Pete Tutorials Inc</p>
3  <p><a href="mailto:purdueExample.com">purduepete@example.com</a></p>
4  </footer>

```

Add content to the menu, with **menu.ejs**, to navigate between the home, about and contact pages, below. We need to add the routes too!



```

{} package.json X  <> index.ejs  <> base.html  <> head.ejs
EXPLORED
OPEN EDITORS
{} package.json
<> index.ejs views\p...
<> base.html
<> head.ejs views\p...
X <> menu.ejs views\p...
<> footer.ejs views\...
JS app.js
<> about.ejs views\...
<> contact.ejs views...
views C:\Users\13179\Desktop\PurdueClasses\PurdueFall24\313\module4\ejs-tutorial1\p
1  <!-- @format -->
2
3  <nav>
4    <ul>
5      <li><a href="/">Home</a></li>
6      <li><a href="/about">About</a></li>
7      <li><a href="/contact">Contact</a></li>
8    </ul>
9  </nav>
10

```

These **partials** can be included in one or more pages, and used and reused across the application. How is this done with EJS? With the **include** directive.

Modify the content of **index.ejs** to include the partials, as below.

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

```
<%- include('../partials/menu.ejs') %>
```

The reference point is the views/pages folder, so we need to go up one folder

```
../
```

And then into the partials folder,

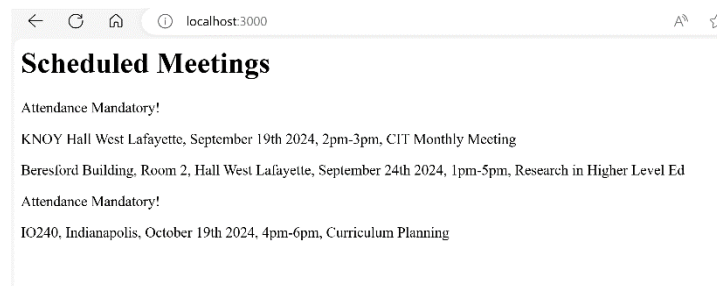
```
../partials
```

```

2  <%- include('../partials/head.ejs') %>
3
4  <%- include('../partials/menu.ejs') %>
5  <body>
6  <h1><%= title %></h1>
7  <% data.forEach((meeting)=>{ %> <% if (meeting.mandatory) { %> <%
8  console.log(meeting.mandatory) %>
9  <span>Attendance Mandatory!</span>
10 <% } %>
11 <p><%= meeting.location %>, <%= meeting.dateTime %>, <%= meeting.topic %></p>
12 <% } %>
13 <%- include('../partials/footer.ejs') %>
14 </body>
15 </html>

```

Now if you restart the app, you should see an un-styled home page, but now one with a head and footer.



You can modify the menu placement so that this is inside the body.

Let's finish by adding the other 2 routes, and reusing the partials to keep everything the same.

Modify the app.js code to include the other 2 routes, and pass the title of the page to the respective document.

```

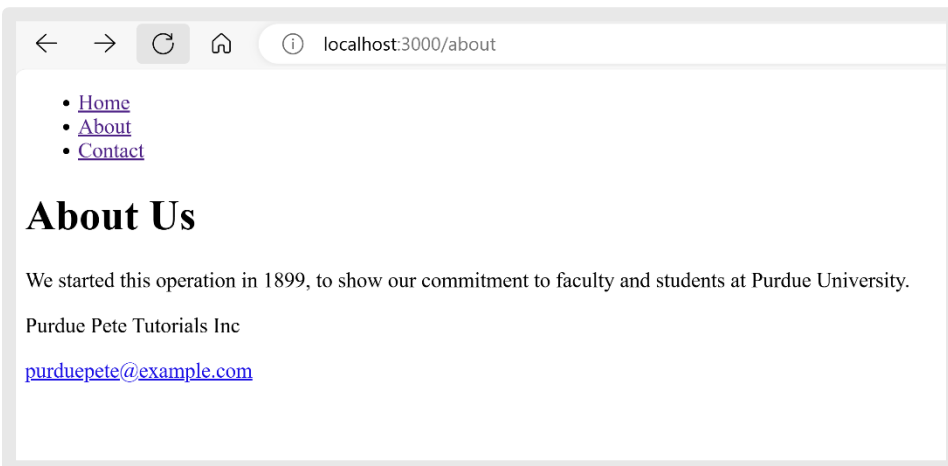
39  app.get("/", (req, res) => {
40    res.render("pages/index", { data: meetings, title: "Scheduled Meetings" });
41  });
42  app.get("/about", (req, res) => {
43    res.render("pages/about", { title: "About Us" });
44  });
45  app.get("/contact", (req, res) => {
46    res.render("pages/contact", { title: "Contact Us" });
47  });
48  app.listen(port, () => {
49    console.log(`Listening at port ${port}`);
50  });
51

```

```

<> about.ejs  x  {} package.json  <> index.ejs
views > pages > <> about.ejs > ? > ? > ? body
1  <!-- @format -->
2
3  <%- include("../partials/head") %> <%- include("../partials/menu") %>
4  <body>
5      <h1><%= title %></h1>
6      <p>
7          We started this operation in 1899, to show our commitment to faculty and
8          students at Purdue University.
9      </p>
10     <%- include("../partials/footer") %>
11 </body>
12 </html>

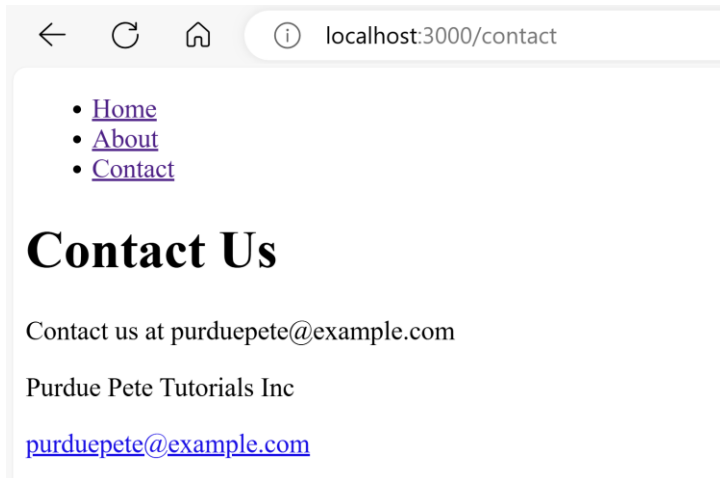
```



```

<> about.ejs  <> contact.ejs  x  {} package.json  <> index.ejs
views > pages > <> contact.ejs > ? > ?
1  <%- include("../partials/head") %>
2  <%- include("../partials/menu") %>
3  <body>
4  <h1><%= title %></h1>
5  <p> Contact us at purduepete@example.com</p>
6  |   <%- include("../partials/footer") %>
7  </body>
8  </html>

```



Now, you know how to

- create templating code using EJS
- include partials to promote reuse
- pass data from the server application to the pages that represent the views

Now, let's integrate some styling, this time with Bootstrap. Bootstrap is one of the most popular styling frameworks in Web development, but the code base is massive. In the next tutorial, we will use the CDN (content delivery network), even though we can also do an *npm install* on Bootstrap and add it to our project. We will avoid this, to minimize **bloat**.

Using Bootstrap

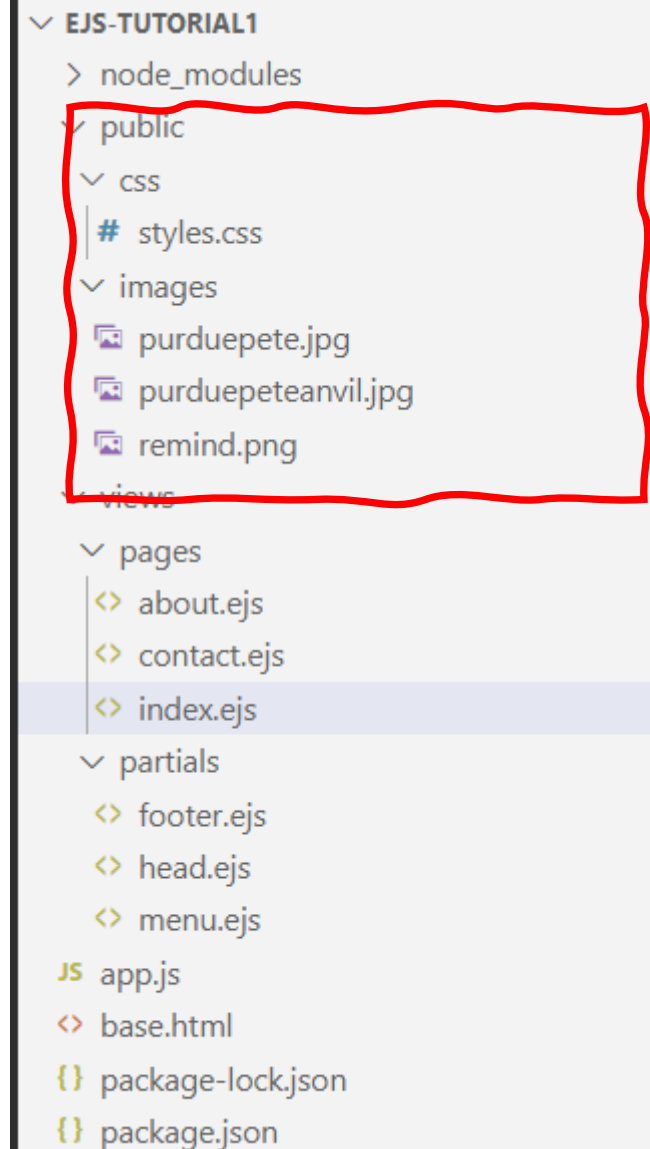
The Bootstrap library is very often used to spin off a professional user interface. However, it is not easy to use, and customization to suit specific needs can be tricky. As with anything, once you learn how to use it and practice creating your own designs using Bootstrap, you will become more fluent with Bootstrap.

Caveat, use of any third party code, Bootstrap or other, exposes the site to cybersecurity concerns.

In this section, we will tackle the menu bar, the footer and add a formatted heading that used to be termed a Jumbotron – however this exact control was deprecated. This will finish this EJS and Node.js tutorial, by adding some styling. Feel free to explore Bootstrap and add your own content here instead of what is in the tutorial. Also, feel free to add your own spin on the styling.

All of the Bootstrap styles can be copied from Bootstrap, at <https://getbootstrap.com/> .

Before adding the styling, create a public folder, with 2 sub-folders, **images** and **css**.
See the following overview:




You have already set the default folder to public, in app.js:

```
import path from "path";
import { fileURLToPath } from "url";
....
const __filename = fileURLToPath(import.meta.url); // get the resolved path to the file
const __dirname = path.dirname(__filename); // get the name of the current directory
app.use(express.static(__dirname + "/public")); // make the public folder the default one
.....
```

This is important because we are pulling in images, and adding a simple style file of our own.

Here are the images and the style rules. You should place them in their respective folders.

	<pre>.note { width: 400px; height: 200px; background-color: blanchedalmond; padding: 5px; border: 3px dashed goldenrod; } .note img { width: 50px; height: 50px; } .takenote { font-weight: bolder; background-color: black; color: goldenrod; }</pre>
--	---

The Menu Bar

<https://getbootstrap.com/docs/5.0/components/navbar/>

This format uses a hamburger style icon, and expands at wider screen breakpoints. The style is Bootstrap, and the branding icon is the one given, in the images folder.

```
<nav class="navbar navbar-expand-lg bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#"
      ></a>
    <button
      class="navbar-toggler"
      type="button"
      data-bs-toggle="collapse"
      data-bs-target="#navbarNav"
      aria-controls="navbarNav"
      aria-expanded="false"
      aria-label="Toggle navigation"
    >
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="/">Reminders</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/about">About</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/contact">Contact</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Replace the code in **menu.ejs** with this content, and modify to fit the application.

The Footer

The code for the footer is from the link below.

<https://getbootstrap.com/docs/5.3/examples/footers/>

Replace the code in **footer.ejs** with this content and modify to fit the application. Note, class of **fixed-bottom** places the footer at the end of the viewport.

```
<footer class="fixed-bottom">
  <nav class="navbar sticky-bottom bg-body-tertiary">
    <div class="container-fluid">
      <a class="navbar-brand" href="https://www.purdue.edu">
        
        Purdue Pete Reminders, Tutorials
      </a>
    </div>
  </nav>
</footer>
```

The Jumbotron

The official Bootstrap Jumbotron is deprecated because there are other styles and controls that give the same result, like the code below.

Add the following code **to *index.ejs*, *about.ejs* and *contact.ejs***, before or after the menu include.

```
<div class="mt-4 p-5 bg-warning text-white rounded">
  <h1><%= title %></h1>
</div>
```

The title is now set to the value passed in.

The Reminder Note

You should change the class of the paragraph, in ***index.ejs***, to “note” to pull in the style.

```
<p class="note">
|   | <% if (meeting.mandatory) { %> <%
console.log(meeting.mandatory) %>
<span class="takenote">Attendance Mandatory!</span>

<% } %><%= meeting.location %>, <%= meeting.dateTime %>, <%= meeting.topic %></p>
```

The image is pulled from the **images** folder.

Now, you should have an example that shows

- how to use EJS basic syntax
- how to pass values between controller code and view documents
- how to set user interface component values
- how to iterate using EJS
- how to use conditional logic with EJS
- how to integrate Bootstrap
- the importance of a folder structure
- how to establish a default folder or directory

