Node.js Workshop 5: Templates and views

After completing this workshop the student knows how to:

- Utilize EJS template engine with Express.js
- Parse JSON files using EJS templates
- Independent Assignments 1 and 2

Create a new folder called WS5 for these assignments. Place all your code there.

Utilize EJS template engine with Express.js

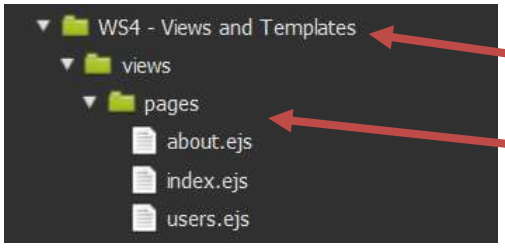
EJS template engine should be bundled with Express.js. Just to make sure, open a new terminal and type in "npm install ejs".

Below I have created a file called server.js. We need to apply the template engine to our code before we can use it. This is done using the app.set() -function.

```
1  // server.js
2  // load the things we need
3  var express = require('express');
4  var app = express();
5
6  // set the view engine to ejs
7  app.set('view engine', 'ejs');
8
9  app.listen(8081);
10 console.log('8081 is the magic port');
```

Previously we have used Node to send text responses and text/json files as a response to the browsers request. However, after the template engine is set, we can tell Node.js to render a response using different templates.

As a default, all the template files are placed under the views directory. EJS template files are commonly named with .ejs -suffix. **NOTE: our server.js should be located in the root folder (WS4).**



server.js goes to the root folder (WS4)

template-files under the views/pages -folder

Below we're creating a route ("/") which returns a rendered template to the browser using a file called index under views/pages subdirectory. Add this code before the app.listen –directive.

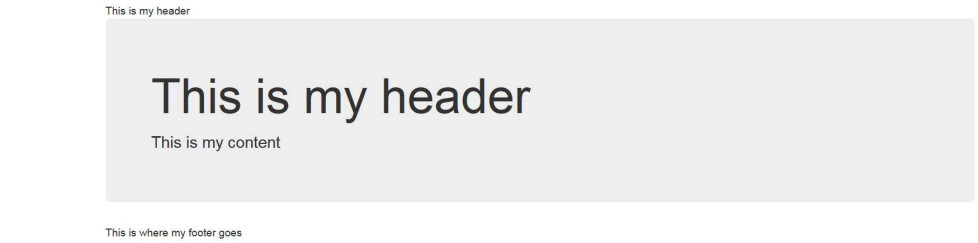
```
// index page
app.get('/', function(req, res) {
    res.render('pages/index');
});
```

Create templates with EJS

Before running the code, we need to create the template file(s) ofcourse. So create views/pages subdirectories and under it save a file called "index.ejs". The copy this content to the file: http://pastebin.com/kYShxdrz

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Super Awesome</title>

    <!-- CSS (load bootstrap from a CDN) -->
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <style>
        body {
            padding-top: 50px;
        }
    </style>
</head>

<body class="container">

    <header>
        This is my header
    </header>
    <main>
        <div class="jumbotron">
            <h1>This is my header </h1>
            <p>This is my content </p>
        </div>
    </main>

    <footer>
        This is where my footer goes
    </footer>
</body>
</html>
```

Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser.



Create variables within templates

The templates are meant to be used in scenarios where the content changes dynamically. Now on our previous demo the template file was completely static, thus nothing changes when the page is rendered and we could just as well send the file as such without using a template engine at all.

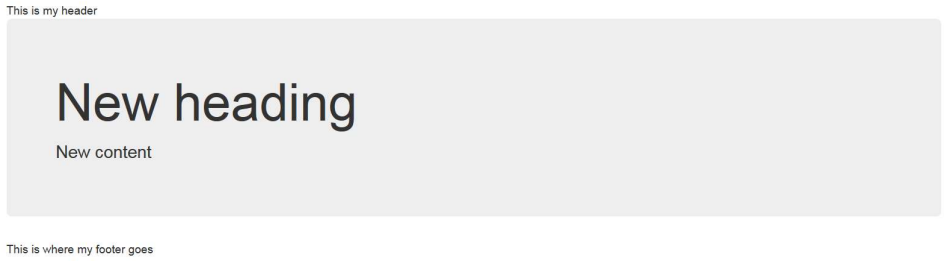
However, lets edit the template just created a bit. We will add some variables here and there and see how we can populate them within the HTML. Note that the JavaScript -variables set in the first lines of the page should be rendered at place where `<%= variable_name %>` is used.

```
1  <!-- views/pages/index.ejs -->
2  <% var heading="New heading"; %>
3  <% var content="New content"; %>
4
5  <!DOCTYPE html>
6  <html lang="en">
7  <head>
8  <meta charset="UTF-8">
9  <title>Super Awesome</title>
10
11 <!-- CSS (load bootstrap from a CDN) -->
12 <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
13 <style>
14     body    { padding-top:50px; }
15 </style>
16 </head>
17 <body class="container">
18
19     <header>
20        This is my header
21     </header>
22
23     <main>
24         <div class="jumbotron">
25             <h1><%= heading %> </h1>
26             <p> <%= content %> </p>
27         </div>
28     </main>
29     <footer>
30        This is where my footer goes
31     </footer>
32 </body>
33 </html>
```

Introducing variables. Note that everything written between <% and %> are treated as JavaScript

I'm placing the contents of the variables within the HTML code by using <%= syntax. This outputs the contents at place.

Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser. Note that the contents of the variables are now in use.



Sending variables as parameters

The idea of setting the variables in the beginning of the template page is far from being useful. So in real life, the variables should be passed on to the template from code when we are sending the page to the renderer using res.render() –function.

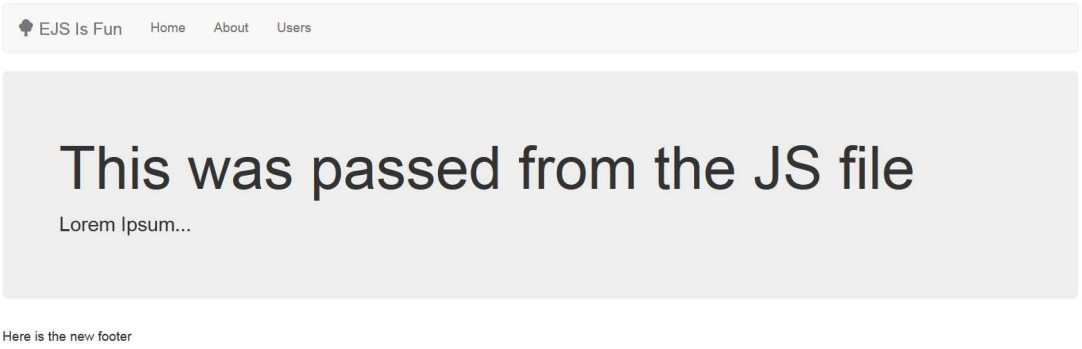
Modify the route we created earlier like shown below. Here we are sending a bunch of data as parameters for the renderer to work with.

```
app.get('/', function(req, res) {
    res.render('pages/index', {
        new_heading: "This was passed from the JS file",
        new_paragraph:"Lorem Ipsum...",
        new_footer: "Here is the new footer"
    });
});
```

All the three parameters (new_heading, new_paragraph, new_footer) are available to our template for output. *Note that you can remove or comment out the variables in the beginning of the template file since they are no longer used.*

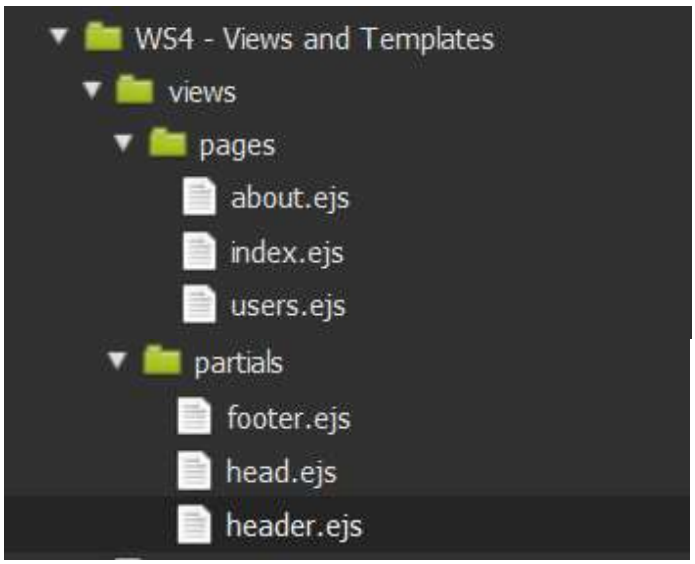
```
    <main>
        <div class="jumbotron">
            <h1><%= new_heading %></h1>
            <p><%= new_paragraph %></p>
        </div>
    </main>

    <footer>
        <%= new_footer %>
    </footer>
```

Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser. Note that the contents of the variables are now in use.



Using partials

Templates can also be split into multiple parts and combined during runtime. This can be done using the <% include %> -directive. These parts are called “partials” and they are usually placed in a separate directory called “partials”.



Below I have splitted the original HTML-template into three different parts and I'm using includes to combine them into one.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <% include ../partials/head %>
</head>
<body class="container">

    <header>
        <% include ../partials/header %>
    </header>

    <main>
        <div class="jumbotron">
            <h1><%= new_heading %> </h1>
            <p> <%= new_paragraph %> </p>
        </div>
    </main>

    <footer>
        <% include ../partials/footer %>
    </footer>

</body>
</html>
```

The contents of each partial file are as follows:

Head.ejs (same as in the HTML file before)

```
1  <!-- views/partials/head.ejs -->
2
3  <meta charset="UTF-8">
4  <title>Super Awesome</title>
5
6  <!-- CSS (load bootstrap from a CDN) -->
7  <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
8  <style>
9      body    { padding-top:50px; }
10 </style>
11
```

Header.ejs (some new items) Code can be found at: http://pastebin.com/jM0ymqtj

```
<!-- views/partials/header.ejs -->

<nav class="navbar navbar-default" role="navigation">
    <div class="container-fluid">

        <div class="navbar-header">
            <a class="navbar-brand" href="#">
                <span class="glyphicon glyphicon glyphicon-tree-deciduous"></span>
                EJS Is Fun
            </a>
        </div>

        <ul class="nav navbar-nav">
            <li><a href="/">Home</a></li>
            <li><a href="/about">About</a></li>
            <li><a href="/users">Users</a></li>
        </ul>

    </div>
</nav>

```

Footer.ejs (some new items)

```
¨<!-- views/partials/footer.ejs -->

<p class="text-center text-muted">© Copyright 2014 The Awesome People</p>

```

Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser.

Using loops in templates

Templates are a powerful tool when combined with programming logic. For example, one could pass in a JSON data or an array and parse the contents within a template. Lets try this in action.

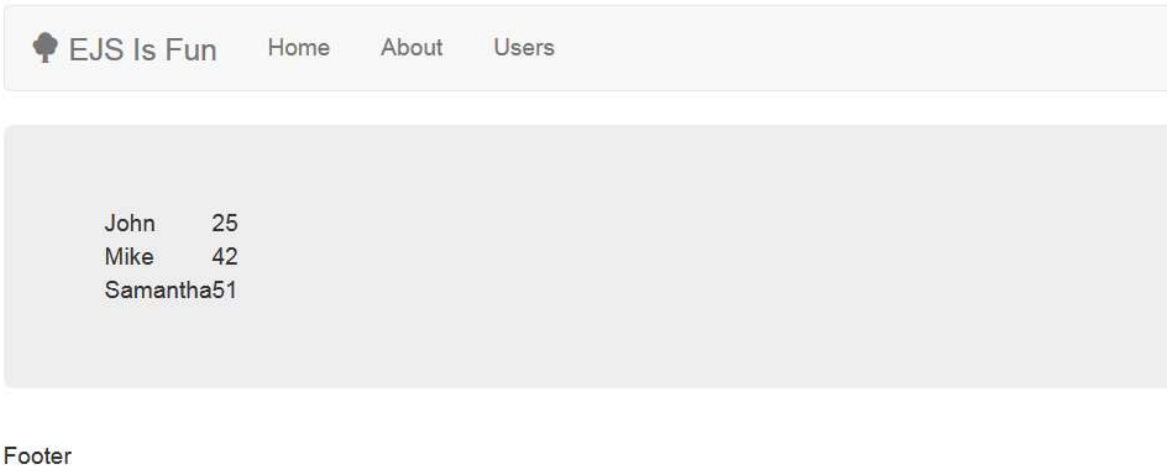
First define a dataset with some users and values. Then create a new route in Node.js called users. Pass on the dataset as a parameter to the users.ejs template.

```
var data = {users:[
  { name: 'John', age: 25 },
 { name: 'Mike', age: 42 },
 { name: 'Samantha', age: 51 }
 ]};

app.get('/users', function(req, res){
 res.render('pages/users', data);
});
```

Then we need to create users.ejs file. Within the template, we can define the loop which will walk through the data which was passed in and output it to the browser as HTML.

```
1  <!-- views/pages/index.ejs -->
2
3  <!DOCTYPE html>
4  <html lang="en">
5  <head>
6      <% include ../partials/head %>
7  </head>
8  <body class="container">
9
10     <header>
11         <% include ../partials/header %>
12     </header>
13
14     <main>
15         <div class="jumbotron">
16
17         <table>
18         <% users.forEach(function(user){ %>
19           <tr><td><%= user.name %></td>
20           <td><%= user.age %></td>
21         </tr>
22         <% })%>
23         </table>
24
25         </div>
26     </main>
27
28     <footer>
29       Footer
30     </footer>
31
32 </body>
33 </html>
```

Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser.



Using if statement in templates

Furhermore we could add an if statement within the loop. This would only print out the users who are NOT named John and who are under 50. For more features and functionality, see https://www.npmjs.com/package/ejs.

```
<main>
    <div class="jumbotron">

    <table>
    <% users.forEach(function(user){ %>
        <% if (user.name != "John" && user.age < 50) { %>

        <tr>
          <td><%= user.name %></td>
          <td><%= user.age %></td>
        </tr>

        <% } %>

    <% })%>
    </table>
```

Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser.