**Node.js Workshop 5: Templates and views**

**After completing this workshop, student knows how to:**

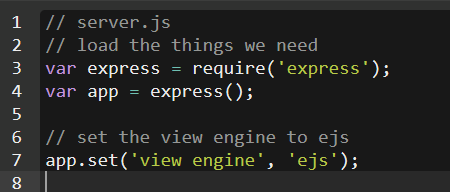
* Utilize EJS template engine with Express.js
* Parse JSON files using EJS templates
* Independent Assignments 1 and 2

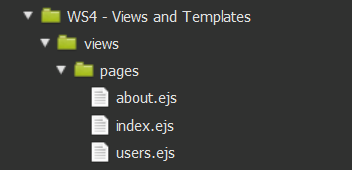
*Create a new folder called WS5 for these assignments. Place all your code there.*

**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.

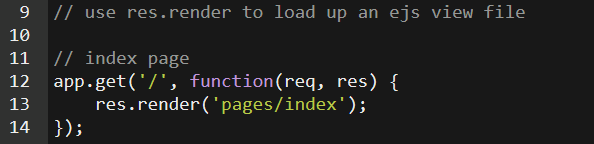
****

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

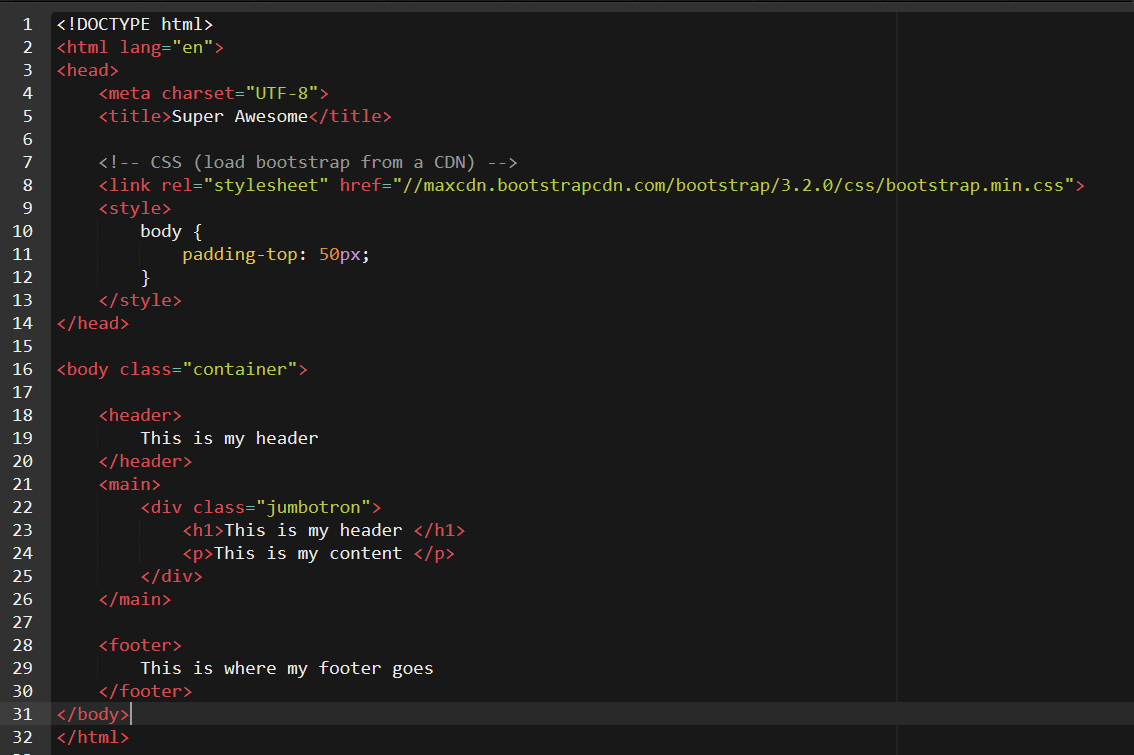
Below we’re creating a route (“/”) which returns a rendered template to the browser using a file called index under views/pages subdirectory. Add this code before the app.listen –directive.



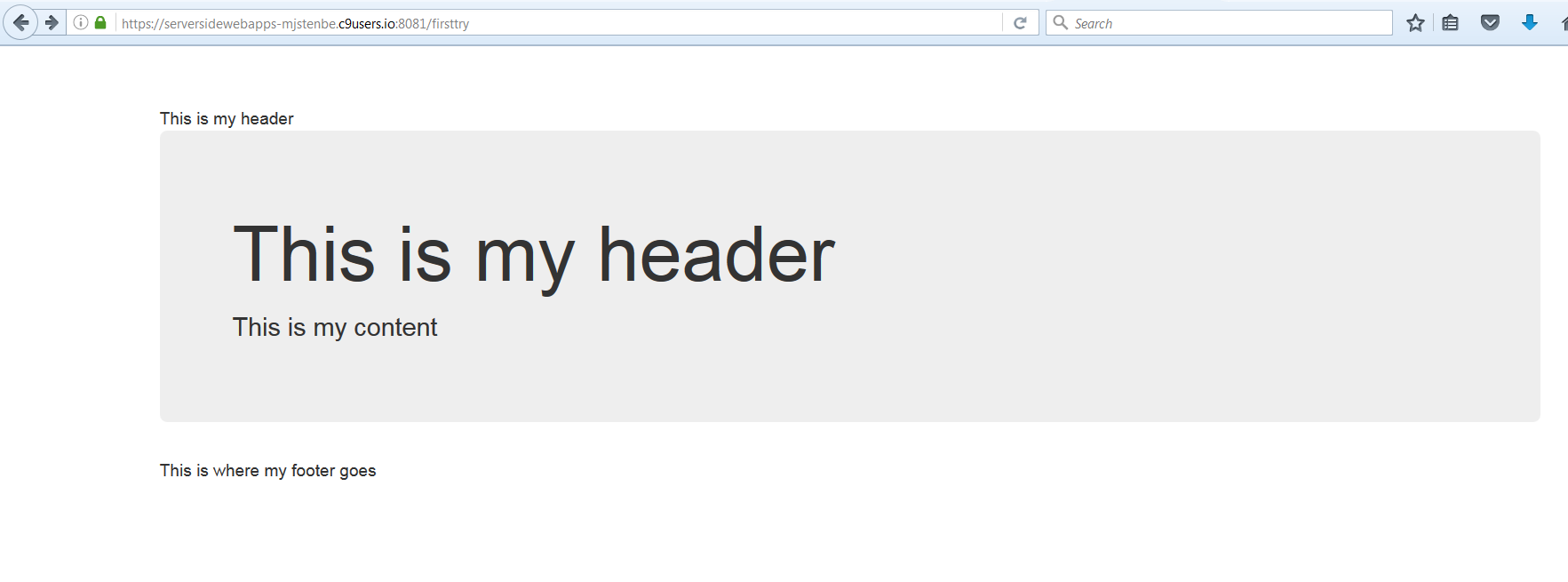
**Create templates with EJS**

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>



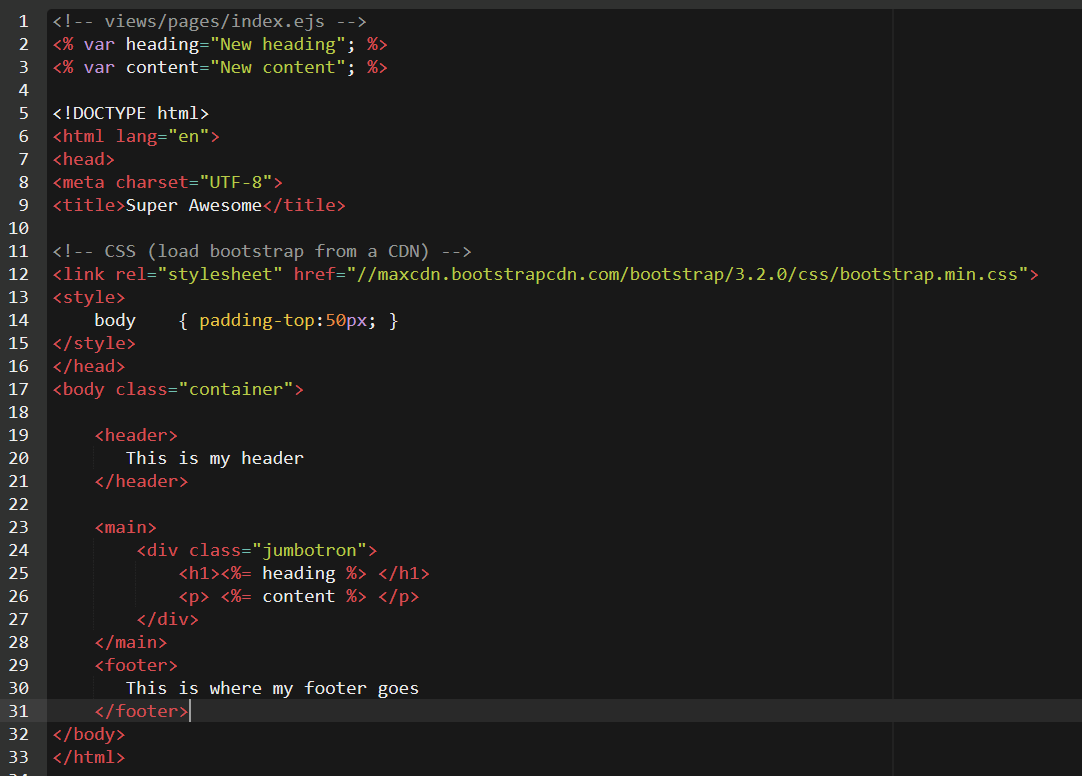
Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser.



**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.

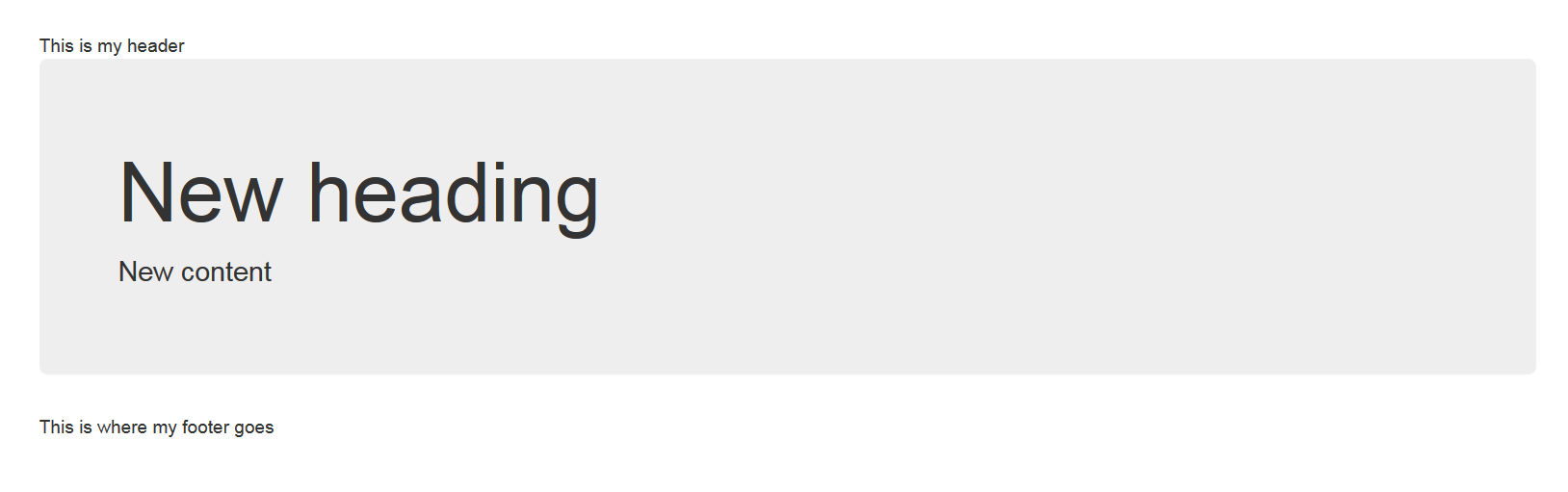
However, lets edit the template just created a bit. We will add some variables here and there and see how we can populate them within the HTML. Note that the JavaScript –variables set in the first lines of the page should be rendered at place where <%= variable\_name %> is used.



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

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

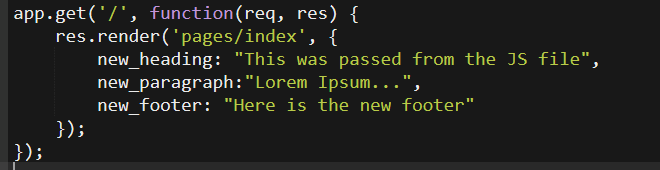
Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser. Note that the contents of the variables are now in use.



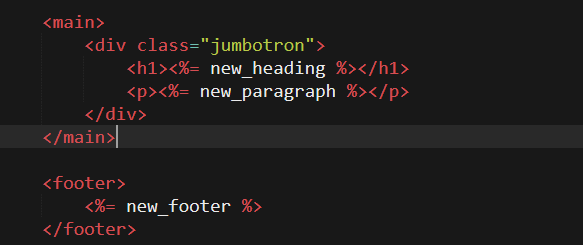
**Sending variables as parameters**

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.

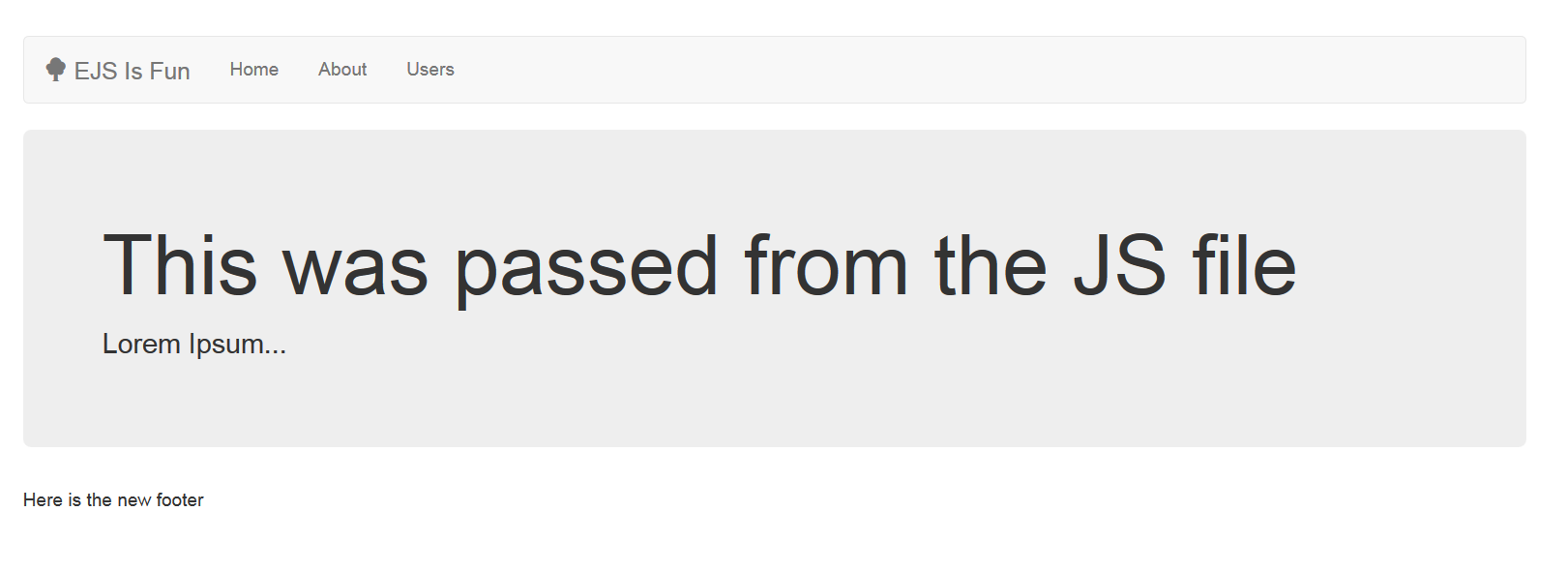
Modify the route we created earlier like shown below. Here we are sending a bunch of data as parameters for the renderer to work with.



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.*

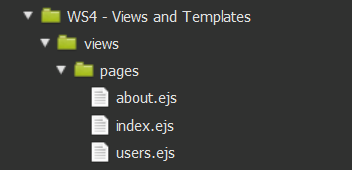


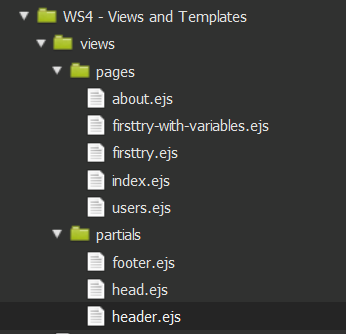
Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser. Note that the contents of the variables are now in use.



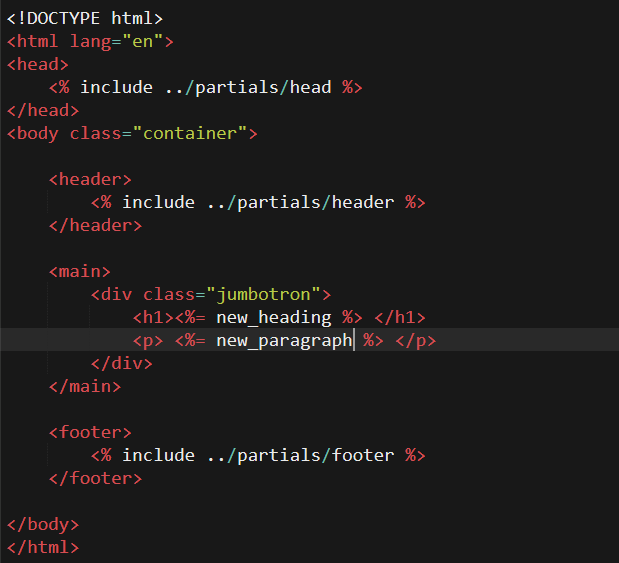
**Using partials**

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”.



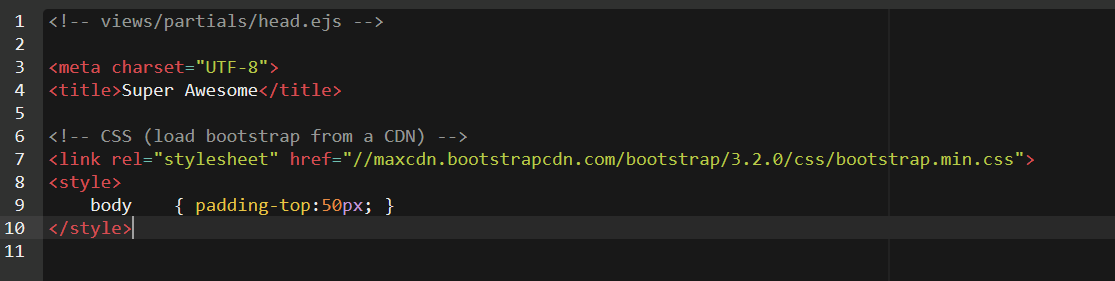


Below I have splitted the original HTML-template into three different parts and I’m using includes to combine them into one.

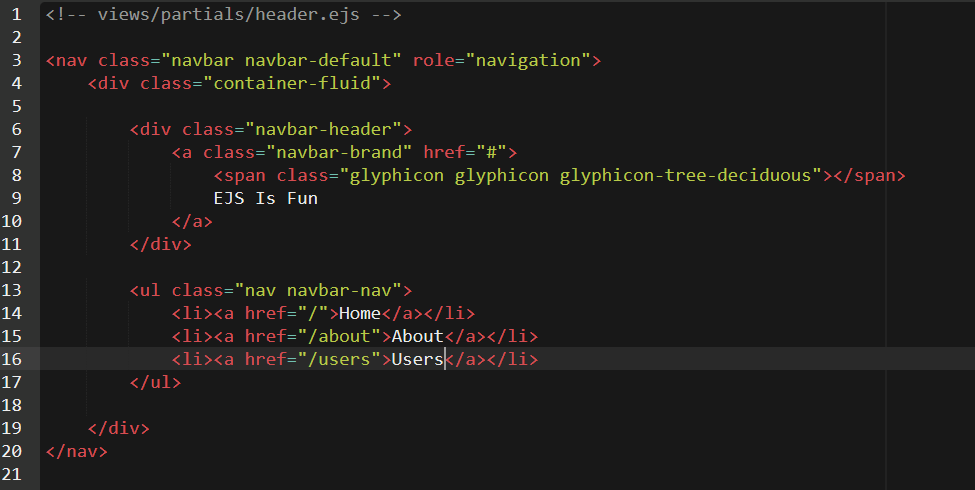


The contents of each partial file are as follows:

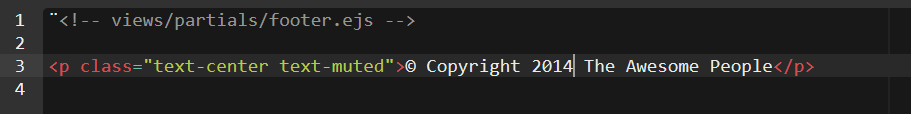
Head.ejs (same as in the HTML file before)



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



Footer.ejs (some new items)

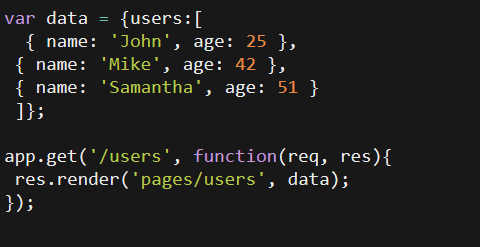


Now run the Node.js code and see what happens. You should be able to see the HTML-file rendered in your browser.

**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.

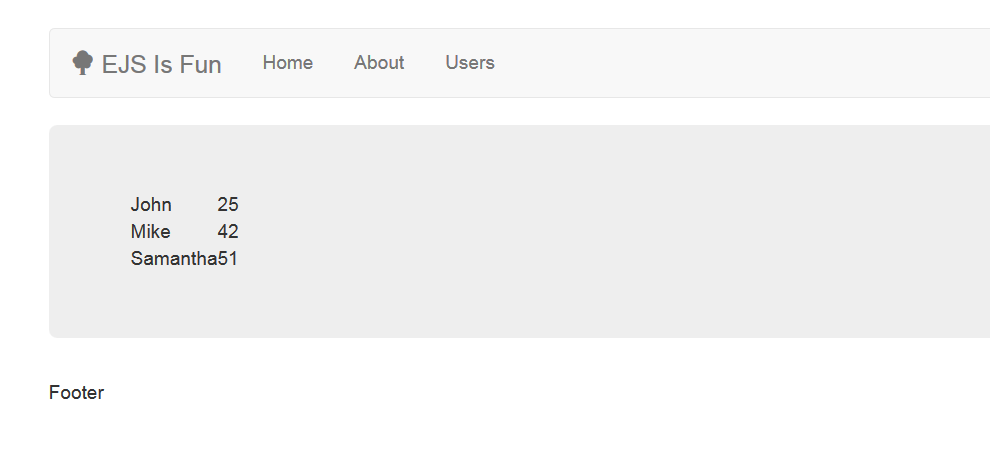
First define a dataset with some users and values. Then create a new route in Node.js called users. Pass on the dataset as a parameter to the users.ejs template.



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.



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>.



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

