502070

WEB APPLICATION DEVELOPMENT USING NODEJS

LESSON 03 – TEMPLATING

OUTLINE

1. Template Engine
2. Pug: A Different Approach
3. Handlebars Basics

OUTLINE

1. Template Engine
2. Pug: A Different Approach
3. Handlebars Basics

Recall

```
const express = require('express')
const hbs = require('express-handlebars')
const app = express()
// configure Handlebars view engine
app.engine('handlebars', hbs.engine({
    defaultLayout: 'main',
}))
app.set('view engine', 'handlebars')
```

Recall

```
app.get('/', (req, res) => res.render('home'))
app.get('/about', (req, res) => res.render('about'))
// custom 404 page
app.use((req, res) => {
    res.status(404)
    res.render('404')
})
// custom 500 page
app.use((err, req, res, next) => {
    console.error(err.message)
    res.status(500)
    res.render('500')
})
```

Choosing a Template Engine

- In the Node world, you have many templating engines to choose from. Here are some criteria:
  - *Performance:* Clearly, you want your templating engine to be as fast as possible
  - *Client, server, or both?*
  - *Abstraction:* Do you want something familiar (like normal HTML with curly brackets thrown in, for example)
- https://expressjs.com/en/resources/template-engines.html

OUTLINE

1. Template Engine
2. Pug: A Different Approach
3. Handlebars Basics

Pug: A Different Approach

- Pug stands out by abstracting the details of HTML away from you
- Pug homepage: https://pugjs.org/api/getting-started.html
- Let's take a look at what a Pug template looks like, along with the HTML it will output

Pug: A Different Approach

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script.
      if (foo) {
         bar(1 + 5)
      }
  body

    h1 Pug
    #container
      if youAreUsingPug
        p You are amazing

    else
      p Get on it!
    p.
      Pug is a terse and
      simple templating
      language with a
      strong focus on
      performance and
      powerful features.
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Pug Demo</title>
<script>
    if (foo) {
        bar(1 + 5)
    }
</script>
<body>
<h1>Pug</h1>
<div id="container">

<p>You are amazing</p>


<p>
Pug is a terse and
 simple templating
 language with a
 strong focus on
 performance and
 powerful features.
</p>
</body>
</html>
```

OUTLINE

1. Template Engine
2. Pug: A Different Approach
3. Handlebars Basics

Handlebars Basics

- *Handlebars* is an extension of Mustache, another popular templating engine
- The concepts we're discussing are broadly applicable to other templating engines
- The key to understanding templating is understanding the concept of *context*
- For example, if my *context* object is

```
{ customer: 'World' }
```

- and my template is

```
<p>Hello,{{customer}}!</p>
```

- then *{{customer}}* will be replaced with *World*

Handlebars Basics

- What if you want to pass HTML to the template? For example, if our context was instead

```
{ customer: '<b>World</b>' }
```

- then using the previous template will result in * Hello,<b>World<b> *
- To solve this problem, simply use three curly brackets instead of two: *{{{name}}}*

Handlebars Basics - Comments

- *Comments* in Handlebars look like *{{! comment goes here }}*.
- It's important to understand the distinction between Handlebars comments and HTML comments
- Consider the following template:

```
{{! super-secret comment }}
<!-- not-so-secret comment -->
```

- The Handlebars comment will never be sent to the browser, whereas the HTML comment will be visible if the user inspects the HTML source

Handlebars Basics - Blocks

- *Blocks* provide flow control, conditional execution, and extensibility
- Consider the following context object:

```
{
    currency: {
        name: 'United States dollars',
        abbrev: 'USD',
    },
    tours: [
        { name: 'Hood River', price: '99.95' },
        { name: 'Oregon Coast', price: '159.95' },
    ],
    specialsUrl: '/january-specials',
    currencies: [ 'USD', 'GBP', 'BTC' ],
}
```

Handlebars Basics - Blocks

- Now let's examine a template we can pass that context to:

```
<ul>
  {{#each tours}}
    {{! I'm in a new block...and the context has changed
}}
    <li>
      {{name}} - {{price}}
      {{#if ../currencies}}
        ({{../currency.abbrev}})
      {{/if}}
    </li>
  {{/each}}
</ul>
{{#unless currencies}}
  <p>All prices in {{currency.name}}.</p>
{{/unless}}
```

```
{{#if specialsUrl}}
  {{! I'm in a new block...but the context hasn't changed
(sortof) }}
  <p>Check out our <a href="{{specialsUrl}}">specials!</p>
{{else}}
  <p>Please check back often for specials.</p>
{{/if}}
<p>
  {{#each currencies}}
  <a href="#" class="currency">{{.}}</a>
  {{else}}
    Unfortunately, we currently only accept
{{currency.name}}.
{{/each}}
</p>
```

Handlebars Basics - Layouts

- You'll notice that when we created the view engine, we specified the name of the default layout. It will use *views/layouts/main.handlebars* as the layout:

```
app.engine('handlebars', hbs.engine({
    defaultLayout: 'main',
})
```

- If we want to use a different template, we can specify the template name:

```
app.get('/about', (req, res) => res.render('about', { layout: 'microsite' }))
```

- If we don't want to use a layout at all, you can specify *layout: null* in the context object:

```
app.get('/foo', (req, res) => res.render('about', { layout: null }))
```

Handlebars Basics - Helpers

- You can create your own helpers to perform complex logics using the expression system that Handlebars provides.
- There are two kinds of helpers: *function helpers* and *block helpers*.
- The first definition is meant for a single expression, while the latter is used for block expressions.

Handlebars Basics – Function Helpers

- When we instantiate the Handlebars object, we'll add a helper called *currency:*

```
app.engine('handlebars', hbs.engine({
    defaultLayout: 'main',
    helpers: {
        currency: function(price) {
            return '$' + price
        },
    },
}))
```

Handlebars Basics – Function Helpers

- In the view, the syntax for a function helper is {{helperName parameter1 parameter2 ...}}

```
<ul>
  {{#each tours}}
    <li>
      {{name}} - {{currency price}}
    </li>
  {{/each}}
</ul>
```

Handlebars Basics – Block Helpers

- Block helpers make it possible to define custom iterators and other functionality that can invoke the passed block with a new context
- To better illustrate the syntax, let's define another block helper that adds some markup to the wrapped text.

```
<ul>
  {{#each tours}}
    <li>
      {{#bold}}{{{name}}}{{/bold}} - {{currency price}}
    </li>
  {{/each}}
</ul>
```

Handlebars Basics – Block Helpers

- When we register a custom block helper, Handlebars automatically adds an options object as the last parameter to the callback function

```
app.engine('handlebars', hbs.engine({
    defaultLayout: 'main',
    helpers: {
        currency: function(price) {
            return '$' + price
        },
        bold: function(options) {
            return '<b>' + options.fn(this) + '</b>'
        },
    },
}))
```

Handlebars Basics – Block Helpers

- This *options* contains a function (*options.fn*) that behaves like a normal compiled Handlebars template.
- Specifically, the function will take a context and return a String.
- Handlebars always invokes helpers with the current context as *this*
- Let's change the example of the previous section by using a block helper named *tours*

Handlebars Basics – Block Helpers

```
<ul>
  {{#mytours tours}}
    <li>
      {{{name}}} - {{price}}
    </li>
  {{/mytours}}
</ul>
```

Handlebars Basics – Block Helpers

```
app.engine('handlebars', hbs.engine({
    defaultLayout: 'main',
    helpers: {
        mytours: function(tours, options) {
            var len = tours.length
            var returnData = ''
            for (var i=0; i<len; i++) {
                tours[i].name = '<b>' + tours[i].name + '</b>'
                tours[i].price = '$' + tours[i].price
                returnData = returnData + options.fn(tours[i])
            }
            return returnData
        },
    },
}))
```

Handlebars Basics – Partials

- First, we create a partial file, *views/partials/tours.handlebars*:

```
<ul>
  {{#each tours}}
    <li>
      {{name}} - {{price}}
    </li>
  {{/each}}
</ul>
```

- To put our partial on the about page, edit *views/about.handlebars*

```
<h1>About</h1>

{{> tours}}
```

Handlebars Basics – Partials

- Very often, you'll have components that you want to reuse on different pages
- One way to achieve that with templates is to use *partials*
- They are written as *{{> partialName}}*
- *express-handlebars* will know to look in *views/partials* for a view called *partialName.handlebars*
- Custom data can be passed to partials through parameters.
  - *{{> myPartial parameter=value }}*