**Lecture 77**

**Module Introduction**

* Refer notes

**Lecture 78**

**Sharing data across requests and users**

* Refer code 01-sharing-data-across-requests-and-users.
* Using global variables to share data across requests is not recommended. Because data will be shared across users also.

**Lecture 79**

**Templating engines – an overview**

* For generating dynamic content in view
* . Templating engines work like this, we got a html-ish template and with that I mean that you typically write some code, some file that contains a lot of html, your html structure and markup, your style and javascript imports, all of that is typically included but you have some blanks in there, some placeholders. And then you have your node express content in your app, like our dummy array, our products array we're currently using and you've got a templating engine which understands a certain syntax for which it scans your html-ish template and where it then replaces placeholders or certain snippets depending on the engine you're using with real html content but that content, this html content it uses there is generated on the fly, on the server by the templating engine taking that dynamic content into account.
* The three options I want to present to you are the three most popular ones, we've got ejs, pug formerly named jade and handlebars.
* Ejs looks something like this, you write normal html markup but then you also add like this smaller than percentage sign and then sometimes an equal sign.

*<p><%= name %></p> .*

Uses normal HTML and plain JS in your templates

* Pug uses a different syntax, it doesn't use real html, it replaces this with a minimized version or a minimal version and then it also allows you to output dynamic content with this syntax, the hashtag curly brace syntax for example

*p #{name}*

Use minimal HTML and custom template language.

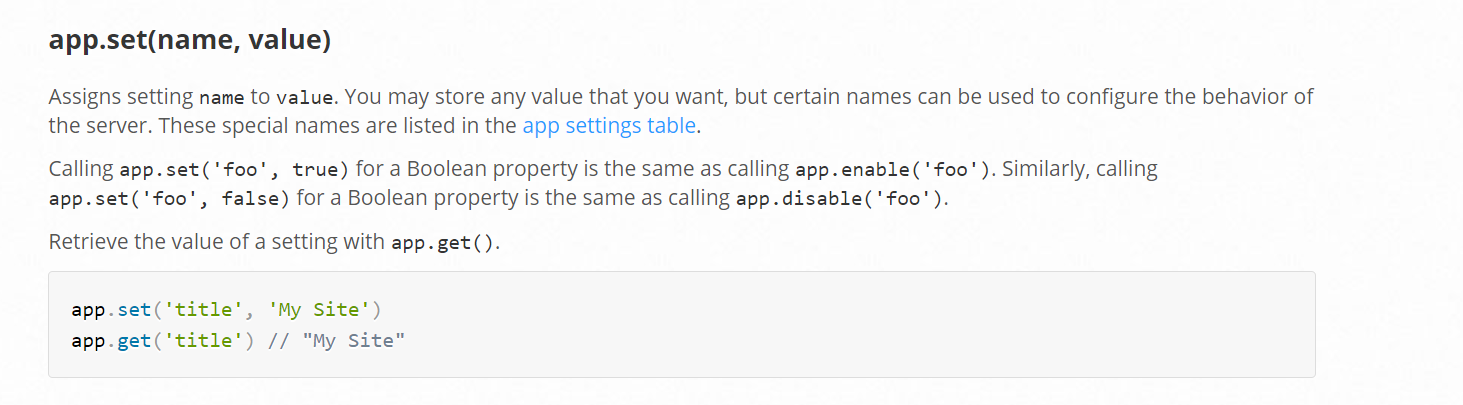
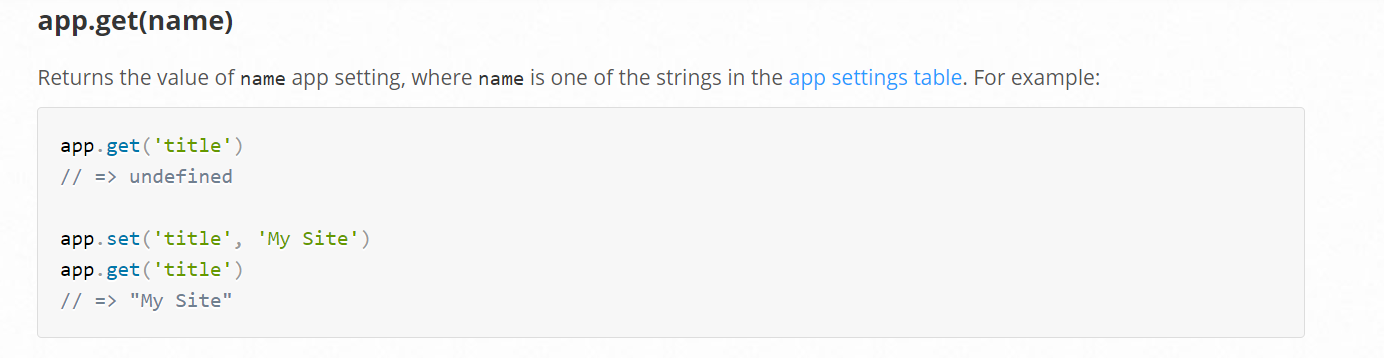
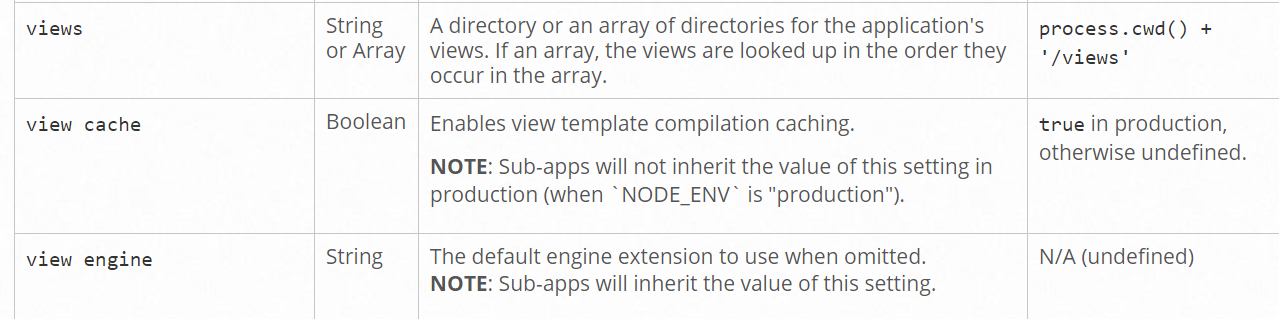
* Handlebars in turn uses html again but then you have the double curly brace placeholders for the dynamic content, so similar to ejs.

*<p>{{ name }}</p>*

Use normal HTML and custom template language

**Lecture 80**

**Installing and implementing pug**

* Refer code 02-added-pug
* Express-handlebar package has bultin integration with express.
* app.set() allows us to set any values globally on our express application and this can actually also be keys or configuration items express doesn't understand, in that case it just ignores them but we could actually read them from the app object with app get and this would be another way of sharing data across our application but not really something I'm interested in here.
* 
* 
* 
* <https://expressjs.com/en/4x/api.html>
* View engine allows us to tell express hey for any dynamic templates we're trying to render and there will be a special function for doing that, please use this engine we're registering here and views allows us to tell express where to find these dynamic views. So what we can do here is we can app set and set the view here, view engine to a string, pug. Now you can't enter anything here, we use pug here because we installed the pug templating engine and this engine actually ships with built in express support and auto registers itself with express so to say. So that is why this works, it doesn't work for all engines.
* . We can set an additional configuration though, we can let express know where to find our views, however the default setting here in this last column, the default setting for views already is our, basically our main directory and then the views folder, still I'll send it explicitly here to show you how this would work if you would store your views in another folder which is not called views but maybe templates or whatever it is, that you have to set this configuration item here, whoops this one and here, I will set it too even though it wouldn't be needed because views is the default.
* Res.render(‘shop’) used to send the html as response.
* So in shop.js where we do define what should be our response, we have to change the response because right now, we're sending the html file, now we want to do something else. We can use response and then there is a special render method, this is provided by expressjs and it will use the default templating engine which is why we had to define it here, it will use that default templating engine and then return that template. And now we defined that all the views are in the views folder, we also don't have to construct a path to that folder instead we can just say shop. We also don't need shop.pug because we defined pug as the default templating engine so it will look for .pug files. With this if we save and we reload the shop page, we see the header, we don't see the main content because we didn't add this but the rest is working just fine. And if you inspect this or view the page source, you'll see this is normal html code, so it's not our minimal version which the browser wouldn't be able to read anyways but it is the html code pug generated for us based on that minimal version.
* <https://pugjs.org/api/getting-started.html>

**Lecture 81**

**Outputting dynamic content**

* Refer code added-pug-shop-template
* <div class=”grid”> can be written in pug format as either div.grid or simply .grid.When the html element is not specified , pug assumes it be a ‘div’.
* Multiple classes should be concatenated using ‘.’

*<article class=”card product-item”>* is written as *article.card.product-item*

* Attributes are specified within parenthesis

*<img src=”test.jpg” alt=”image”>* is written as *img(src=”test.jpg”, alt=”image”)*

* Dynamic content added using #{product.title}
* Id property can be specified using #

*<input type=”text” id=”title”>* can be written as *input(type=”text”)#title*

**Lecture 83**

**Converting HTML Files to Pug**

* Nothing

**Lecture 84**

**Adding a Layout**

* one pug file can extend the behaviour of other pug files. Placeholders are created in the main pug file using ‘block’ keyword
* In the pug file that extends this main file,we add the content of these placeholders.

**Lecture 85**

**Finishing the pug template**

* Refer code 03-pug-finished

**Lecture 87**

**Working with Handlebars**

* Unlike pug which is built in into express , we have to manually import handlebar and set it as the template engine using app.engine() method.

**const expressHbs = require(‘express-handlebars’);**

**const app = express();**

**app.engine(‘handlebars’,expressHbs());**

**app.set(‘view engine’,’handlebars’);** //*the second argument should be same as the first argument of app.engine() method. Here it is ‘handlebars’; we can give any name*

**app.set(‘views’,’views’)**

* The way of passing data to pages doesnot change with templating engine.
* Refer 04-adding-a-layout-to-handlebars

**Lecture 88**

**Converting our project to handlebars**

* Refer code 04-adding-a-layout-to-handlebars
* Unlike pug , we cannot write *if products.length>0* in the handlebar template.
* We can check if a variable is true or false *{{#if hasProducts}}.* hasProducts has to be passed from express to the template.
* This approach forces us to write the logic in the express itself and not take it to the template.

**Lecture 89**

**Adding the layout to handlebars**

* Refer code 04-adding-a-layout-to-handlebars.
* Placeholder specified in main layout using *{{{ body }}}*

**Lecture 90**

**Working with EJS**

* EJS doesnot support layout by default
* EJS is a templating engine which is just like pug, supported out of the box so we don't need to register the engine as we did it with handlebar.
* EJS has a very nice syntax in my opinion and has a nice mixture of the extended functionalities of pug, so not regarding the html but regarding the javascript code you can use in your templates
* To output a variable value we use <%= <variable\_name>%>
* We can write normal JS code between enclosed within <% %>

E.g *<% if(prod.length>0) {%>*

*//html code*

*<% } %>*

**Lecture 91**

**Working on the layout with partials**

* Refer code 05-working-on-layout-with-partials.
* We use includes to add layout capability to ejs.