Because of formatting is not correct in ms word. See code in code folder.

77)Sharing Data across Requests and users

We are not working with data as of now. Its quite difficult to do now bcoz we are not using database. But we can use variables to store data and see if they are shared across incoming requests from different users. And that ofcousre will have some important learning because you often well don’t want to share such data.

Here what we are doing here we send data thorugh form, then we save it some variable(array) in nodejs code. After user submits that form, then we print that variable on console. Code-

In admin.js-

router.post('/product', (req, res, next) => {

products.push({title: req.body.title});

res.redirect('/');

});

In shop.js-

router.get('/', (req, res, next) => {

console.log(adminData.products);

// res.sendFile(path.join(\_\_dirname, '..', 'views', 'shop.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views', 'shop.html'));

res.sendFile(path.join(rootDir, 'views', 'shop.html'));

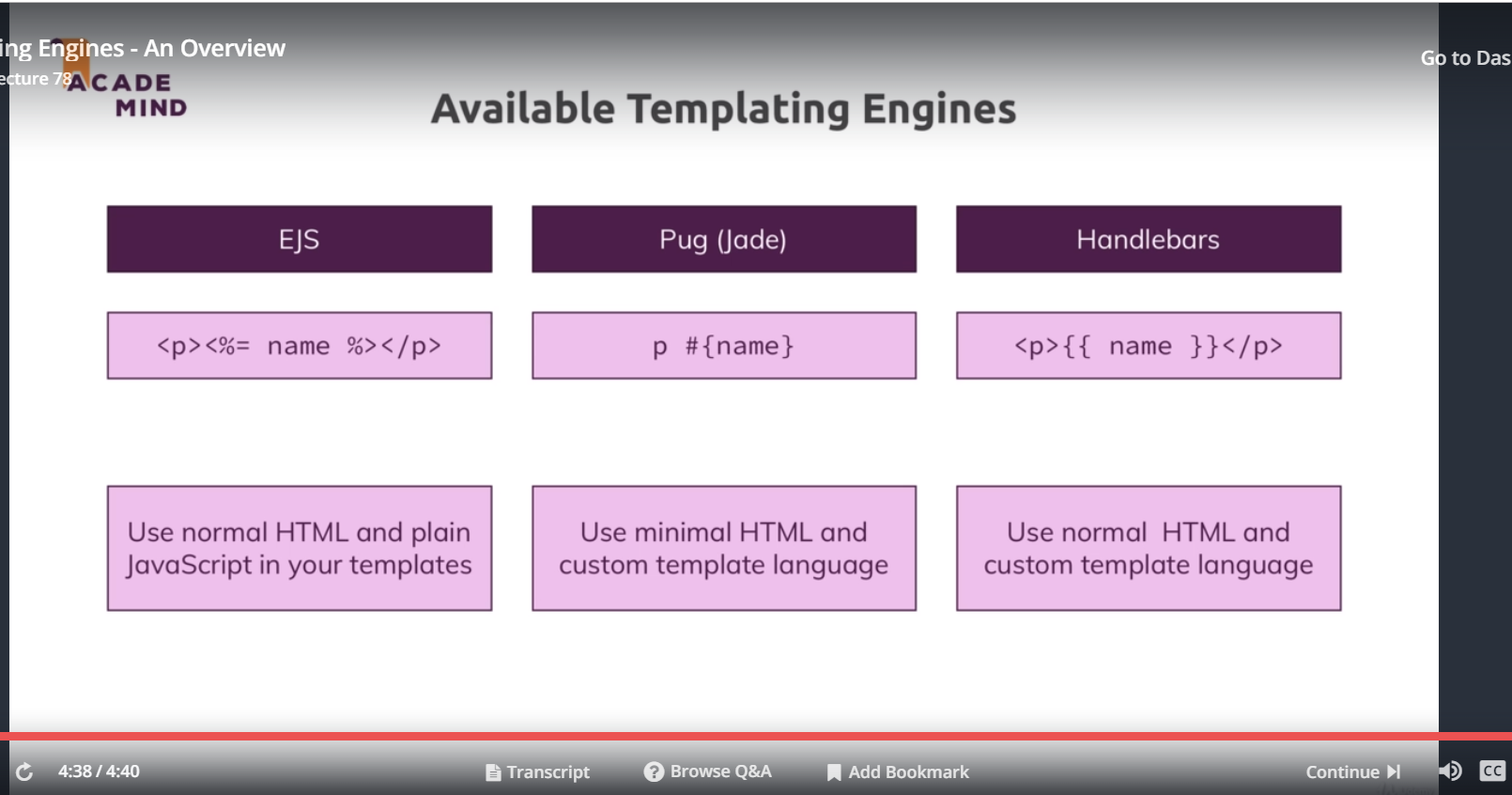
});

So this is interesting to see. So w ecan export something, some object, some array, a refrence type therefore and if we change it in other files. It also gives us update here(in shop.js file). this is one way of sharing data and to be honest we will later use different ways because this one disadvantage.

If we open totally different browser.lets say firefox and visist localhost:3000. So this is totally different browser and this is kind of like a brand new user. It does’nt share any cookies with the other browser, nothing of that kind. It uses same ip address but that does’nt matter here. so it’s brand new request as if it is made from different machine. We still see the data that we entered on chrome. This might be what you want to do, but ver very raraely. In most fo cases each request has different data for different users.

We will use this technique , which is fine for now but later we learn technique to share data in memeory here, in the node app across different request but only for one and same user and not across users because now we share data across request and users.

78)Tempalting engines –An overview



These makes our html content dyanimic. They have some placeholders which are filled with value. all this happens on server side, user(browser) gets pain html.

Here are 3 most popular options. First and third use normal html. Second one does’nt uses real html it replaces them with minimal version. In first one we can have js but other 2 have their own templating language. Language of secod one is extensible but generally offers only a set of things l, operations you could do but if statements and lists would be included . regarding third one it has limited set of features gain including common things like if statements or lists. We will use all of them ten we will pick fav and work with it in rest of course.

79)Installing & Implementing pug

Run this-

**npm install --save ejs pub express-handlebar**

we also have handlevars package but express-handlebars have built in integration into nodejs. Rest 2 have that in their core modules.

Lets use pug . to use that we need o go app.js and we need to tell express(and that is express feature not of node by the way ,another strong reason why we want to use express because with standalone node this will be harder,we have to do all that manually. Here we are simply telling express , hey we got templating engine that is express conforming and that is so please y=use it to render dynamic templates ) that we want to use pug.

We can do this by going to app.js and after we have created express app and stored it in app constant, we can set global configuration value. now what is that. App.set allows us to set any values globally on our express app and this can actually also be keys and configuration items express does not understand. In that case it just ignores them. But we can actually read them on app object with app.get. that would be another approach of sharing data across app but really not something I am interested in here. what we can do is we can use a couple reserved key names here, so configuration items we can set that do lead to expressjs behaving differently. You can see all that list docs-> apiRef of app.set(). There is a table.now most of them really does’nt matter for us here, but feel free to browse through them. Interesting for us is the view engine and views key. View engine tells express that for any dynamic templates we are trying to render(there will be special function for doing that) please use this engine, that we are registering here. views tells, where to find these dynamic views. So we do this-

app.set('view engine','pug');

now we cannot use any strng here. here w e used pug bcoz we have installed pug and this engine ships with built it express support and auto registers itself with express, so as to say. This is why this works. It does’nt work for all engines but you will find more in the links in last lecture, here it works, pug is supported out of the box. with that we are set to use pug.

Then we can tell express where to find our views but default setting for view is in main directory then in views folder. so we can skip it.. to show you, here will set views value.

Then lets create our templating files. Create shop.pug, then type html and select html5 option from ide. This gives you pug structure. Here we see minimal html syntax.we got no normal html tags but keep in mind that pug templaitng engine will complie our code to normal html in the end. Some tips to write pug files

shop.pug-

<!DOCTYPE html>

html(lang="en")

head

meta(charset="UTF-8")

meta(name="viewport", content="width=device-width, initial-scale=1.0")

meta(http-equiv="X-UA-Compatible", content="ie=edge")

title Document

link(rel="stylesheet", href="/css/main.css")

link(rel="stylesheet", href="/css/product.css")

body

header.main-header

nav.main-header\_\_nav

ul.main-header\_\_item-

li.main-header\_\_item

a.active(href="/") Shop

li.main-header\_\_item

a(href="/admin/add-product") Add Product

it is equivalent to shop.html , we w ejust do ot use <>. And attributes are enclosed in (). To see which attributes are supported, press tab after tag , you will see options. We use . to add classes.

Now we have template define but it is not used. Because we are not telling express to render it . We are told express that it should use this templating engine(by using app.set()) whenever we try to render a template .but now we are not trying to that.

So we change the response. Right now we are sending html. We use render method on response. It will use the default templating engine that we have defined. It will use that default templating engine and then return that template and since we have defined that all views are in views folder, we also don’t have to construct a path to that folder. we just mention shop. We also dnt need pug because we defined pug as the default tempaling engine. So it will look for .pug file. code-

Shop.js-

router.get('/', (req, res, next) => {

console.log(adminData.products);

// res.sendFile(path.join(\_\_dirname, '..', 'views', 'shop.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views', 'shop.html'));

// res.sendFile(path.join(rootDir, 'views', 'shop.html'));

res.render('shop');

});

Right now we have shop.pug file in views folder. lets ay we have shop.pug file in sumit folder in views. Then we will use this-

res.render(path.join('sumit', 'shop'));

80)Outputting dynamic content

Here we pass data to our templates and we use data. We also use loop and if , else conditions. If w ehave 2 proeprties on same tag then-

link(rel="stylesheet", href="/css/main.css")

not in pug we do not us estarting and ending tags. Indentation defines nesting.

If we have 2 classes o same tag then-

li.main-header\_\_item.class2

there is no space between tag name and class name(just . is there), same goes for attribute and tag name(only bracket is there). But text is always started after a space between tag name and text of tag. Example-

label.dummyclass.secodclass(for="title", secondattr="sumit") Title

shop.js-

router.get('/', (req, res, next) => {

const products = adminData.products;

// res.sendFile(path.join(\_\_dirname, '..', 'views', 'shop.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views', 'shop.html'));

// res.sendFile(path.join(rootDir, 'views', 'shop.html'));

res.render('shop', {prods: products, docTitle: 'Shop'});

});

So we pass data. This is how we use data in shop.pog-

title #{docTitle}

we used loop in shop.pug. here article tag is repeated. So nesting is defined by indentaion. We also used if else block.

shop.pug-

<!DOCTYPE html>

html(lang="en")

head

meta(charset="UTF-8")

meta(name="viewport", content="width=device-width, initial-scale=1.0")

meta(http-equiv="X-UA-Compatible", content="ie=edge")

title #{docTitle}

link(rel="stylesheet", href="/css/main.css")

link(rel="stylesheet", href="/css/product.css")

body

header.main-header

nav.main-header\_\_nav

ul.main-header\_\_item-

li.main-header\_\_item

a.active(href="/") Shop

li.main-header\_\_item

a(href="/admin/add-product") Add Product

main

if prods.length >0

div.grid

each product in prods

article.card.product-item

header.card\_\_header

h1.product\_\_title #{product.title}

div.card\_\_image

img(src="some data" alt="A Book")

div.card\_\_content

h2.product\_\_price $19.99

p.product\_\_description A very interesting book about so many even more interesting things!

div.card\_\_actions

button.btn Add to Cart

else

h1 NO Prodcts

If you change something in template, you do not need to restart the server and nomdemon will not do so because the templates are not part of you server side code, they are basically which are picked on the fly.

83)Adding layouts

Some portion of or template is repeated everywhere. So what can we do about it? We can create a layout for that. create a folder called layout. There create file called main-layout.pug. now copy some pg file here. now we have some elements which are dynamic now, which depend on page where we use them .question is how can we reuse this skeleton.

Well we can extend this layout from inside our pug views and we can define some placeholders some hooks, where other views can then enter their data. like we have links in header, some page may need additional link tag to import addtional css files.

We define such hooks by using block keyword then any name of our choice which pug understands.

Mainlayout.pub-

<!DOCTYPE html>

html(lang="en")

head

meta(charset="UTF-8")

meta(name="viewport", content="width=device-width, initial-scale=1.0")

meta(http-equiv="X-UA-Compatible", content="ie=edge")

title Page Not Found

link(rel="stylesheet", href="/css/main.css")

block styles

body

header.main-header

nav.main-header\_\_nav

ul.main-header\_\_item-

li.main-header\_\_item

a(href="/") Shop

li.main-header\_\_item

a(href="/admin/add-product") Add Product

block content

here we have 2 hooks. One for additional link tags and second one for content after head.

Also we do set active class on any a tag. We will set it dynamically.

Lets use it in 404.pug-

extends layouts/main-layout.pug

block content

h1 Page Not Found

here we import layout. Now for block cotent we define what we want there. We do not want to add additional styles so we do not mention anything in style sblock.

Do same for rest of pages.

Now only thing remaining is how we can set active class on a tags depending on which page we are.

84)Finishing the Pug template

We know what is remaining for last lecture. We render templates here-

router.get('/add-product', (req, res, next) => {

res.render('add-product', {pageTitle: 'Add Product'});

});

Now we add new filed to object that we pass to templaing engine path field.

router.get('/add-product', (req, res, next) => {

res.render('add-product', {pageTitle: 'Add Product', path: '/admin/add-product'});

});

You can pass anything you like. So now I pass this path into view so that view can find out what is the path for which it was loaded. I will get this path in main-layout. Now we use this path in condition like this-

Main-layout.pug-

<!DOCTYPE html>

html(lang="en")

head

meta(charset="UTF-8")

meta(name="viewport", content="width=device-width, initial-scale=1.0")

meta(http-equiv="X-UA-Compatible", content="ie=edge")

title #{pageTitle}

link(rel="stylesheet", href="/css/main.css")

block styles

body

header.main-header

nav.main-header\_\_nav

ul.main-header\_\_item-

li.main-header\_\_item

a(href="/",class=(path === '/' ? 'active' : '')) Shop

li.main-header\_\_item

a(href="/admin/add-product", class=(path === '/admin/add-product' ? 'active' : '')) Add Product

block content

Here we write js code which is enclosed in brackets her. in each time we pass this path variable to template. Now herew e have also made pageTitle dynamic. So if we use this payout I template1, than that page need to receive pagetItle property na dvalue will be set.

85)Working with handlebars

Handlebar does not follow such a minimal html approach but uses normal html mixed with some normal templating logic. Now go to app.js, here w eneed to change our view engine. We does install express-handlebars but this actually is a package that is not auto installed by express, so inetad we manually have to tell express that there is such an express handlebars engine available. To find out whether you need to do it for your fav engine, check the official docs of templating engine. So first we import express-handlebars and then w etell express that this exists, that this is engine we can use. For this we we use engine method on app, which registers a templaing engine that is not built into express.js, pug was kind of build into express js, express-handlebars is not. To register it we have to define name for our engine. You can use any name you want, ofcourse you should try not to clash with built-in engine name(you will names in end of this module. We use handlebars name, now we need to tell this is name what actual tool I should use. This is object that we imported from expressHbs package. It turns out to be function , so we execute it. This basically initializes this engine you could say. So this function returns the initialized view engine which we can assign to engine here. then w e switch the view engine to handlebars.

App.js-

// const http = require('http');

// const { requestHandler } = require("./routes");

const express = require('express');

const bodyParser = require('body-parser');

const expressHbs = require('express-handlebars');

const path = require('path');

const app = express();

app.engine('handlebars', expressHbs());

app.set('view engine', 'handlebars');

// app.set('view engine','pug');

app.set('views', 'views');

const adminData = require('./routes/admin');

const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/admin',adminData.routes);

app.use(shopRoutes);

app.use((req,res,next) => {

res.render('404', {pageTitle: 'Page Not Found'});

});

app.listen(3000);

console.log(process.mainModule.filename);

console.log(path.dirname(process.mainModule.filename));

console.log(\_\_dirname);

console.log('listening to 3000');

now w eare reday to use handlebars. How do we use it. Lets create files for it.first start with 404 page. So create page named 404.handlebars. we have to name to handlebars because it is default by handlebars and we defined this as engine name in app.js. if you do this-

app.engine('hbs', expressHbs());

app.set('view engine', 'hbs');

now we can have files with hbs extension.

Lets use hbs extension. Lets crete file , remmev=ber handle bar uses normal html with custom synyax. On ething o note is way you pass data into templating egine does not change with templating engine. We pass data like this-

res.render('404', {pageTitle: 'Page Not Found'});

this is how you use it in template , it differs from engine to engine.

In handlebar we have simle html. We define placeholders like this-

404.hbs-

<title>{{pageTitle}}</title>

86)Coverting our project into hbs

Add product was very straight forward. Now in shop.hbs w eneed to have if codrion and we need to have loop to replicate tags. We add if helper in {{}}, then we add a **#** for special block statements, block statements simply are statements which are not just outputting some text but which actually wraps some content that should be output conditionaaly or in loop. Then after {{}} is html code thatw ewant to diplsay codtionally, then we close that block with again with {{}}.

Shop.hbs-

<main>

{{#if prods.length > 0}}

<div class="grid">

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">Great Book</h1>

</header>

<div class="card\_\_image">

<img src="https://cdn.pixabay.com/photo/2016/03/31/20/51/book-1296045\_960\_720.png" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

</div>

{{/if}}

</main>

Here we also want to have else block. So lets have else block-

<main>

{{#if prods.length > 0}}

<div class="grid">

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">Great Book</h1>

</header>

<div class="card\_\_image">

<img src="https://cdn.pixabay.com/photo/2016/03/31/20/51/book-1296045\_960\_720.png" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

</div>

{{else}} <h1>No Products Found</h1>

{{/if}}

</main>

But we get error. The problem that we have here is that handlebars does’nt actually supports statements like this –

prods.length > 0

it just supports output of keys that yield true or false. Now this means that we have to move that logic from the template into our node express code and pass the result of this check into the template. So in shop.js we add a new property called hasProducts-

router.get('/', (req, res, next) => {

const products = adminData.products;

res.render('shop', {prods: products, pageTitle: 'Shop', path: '/', hasProducts: products.length >0});

});

This is core difference to pug already besides that html thing. Here in handlebars we cnt run any logic in our handlebars template. We can just output single property, single variables and their values and we can only use these in if blocks. This actually has positive side, this keeps your template lean . you knw that all logic will happen at express side.

Now we need to have loop that is done with **each block.** Now how do we access each elment of array in template? Here handlebars only give sus one way. It gives us **this**  keyword which refers to the element in array for current iteration. So to each product. So therefore we can output this title here.

Shop.hbs-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>{{pageTitle}}</title>

<link rel="stylesheet" href="/css/main.css">

<link rel="stylesheet" href="/css/product.css">

</head>

<body>

<header class="main-header">

<nav class="main-header\_\_nav">

<ul class="main-header\_\_item-list">

<li class="main-header\_\_item"><a class="active" href="/">Shop</a></li>

<li class="main-header\_\_item"><a href="/admin/add-product">Add Product</a></li>

</ul>

</nav>

</header>

<main>

{{#if hasProducts}}

<div class="grid">

{{#each products}}

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">{{this.title}}k</h1>

</header>

<div class="card\_\_image">

<img src="d=" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

{{/each}}

</div>

{{else}} <h1>No Products Found</h1>

{{/if}}

</main>

</body>

</html>

So philosipy of handlebars is , less logic in template and more logic in node-express code. So you have to prepare everythinh in js code, so that you dnt have to write js expressions in templates.

87)Adding layouts to handlebars

Handlebars supports layout but it works differently than pug. First of all in app.js where we register our handlebar engine we need to pass options. You can press ctrl+space to see what options we can pass. Here we want to tell, where our layouts live. So in which folder I can find my layouts.

app.engine('hbs', expressHbs({layoutsDir: 'views/layouts/'}));

app.set('view engine', 'hbs');

this is default value. you dnt need to set it. You need to set it if you have different path.or you have different views folder. the we can also define a default layout whch should be used for all files.

app.engine('hbs', expressHbs({layoutsDir: 'views/layouts/',defaultLayout: 'main-layout'}));

it means you need to have this file in your layouts folders. In pug we defined a block where we could dynamically add the content the should be rendered from inside the view that extended this layout. Hbs dnt have that feature. So we can’t do that , we cnt define block here. instead the only thing that we can do there is we can define a placeholder with three curlybraces, then adding body there and you have to exactly use this placeholder. This is understood by handlebars and you will then be able to target this in your views that extend the layout, automatically because you set it as default layout.

However if you have some part like you need to add some link(css imports) depending upon which page you are, you ahave to solve this differently, in a kind of similar approach we solved the active class problem in pug. You have to add an if statement here in your main layout and this ofcourse is abit more complex way, though you can do more with handlebars, you could define custom helprs to help you with that but this is beyod scope of this course. Let me show you easier way of doing this-

We handle the classes using if statement. Code-

Main layout.hbs-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>{{PageTitle}}</title>

<link rel="stylesheet" href="/css/main.css">

{{#if formsCSS}}

<link rel="stylesheet" href="/css/forms.css">

{{/if}}

{{#if productCSS}}

<link rel="stylesheet" href="/css/product.css">

{{/if}}

</head>

<body>

<header class="main-header">

<nav class="main-header\_\_nav">

<ul class="main-header\_\_item-list">

<li class="main-header\_\_item"><a class="{{#if activeShop}}active{{/if}}" href="/">Shop</a></li>

<li class="main-header\_\_item"><a class="{{#if activeProduct}}active{{/if}}"href="/admin/add-product">Add Product</a></li>

</ul>

</nav>

</header>

{{{body}}}

</body>

</html>

Now you have to pass additional variable that you are using here. if you do not pass them then they are treaded as false. Lets use it in shop.hbs-

First we pass additional data. now shop.hbs will use the layout by default. You could disable it by going to render function which renders a layout and adding a new property called layout to data object and setting it to false.

Shop.js-

router.get('/', (req, res, next) => {

const products = adminData.products;

// res.sendFile(path.join(\_\_dirname, '..', 'views', 'shop.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views', 'shop.html'));

// res.sendFile(path.join(rootDir, 'views', 'shop.html'));

res.render('shop', {

prods: products,

pageTitle: 'Shop',

path: '/',

hasProducts: products.length >0,

activeShop: true,

productCSS: true,

layout: false

});

});

Herew e want to use layouts, so do not disable it. This is special key understood by handlebars and it would not use default layout(if set to false) , otherwise it will.

Since it will use default layout, in shop,hbs we can get rid of all code. Only that should remain that we want to be injected into {{{body}}}, nothing else. Shop.hbs-

<main>

{{#if hasProducts}}

<div class="grid">

{{#each products}}

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">{{this.title}}k</h1>

</header>

<div class="card\_\_image">

<img src="d=" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

{{/each}}

</div>

{{else}} <h1>No Products Found</h1>

{{/if}}

</main>

If we run code now, we get-

Error: ENOENT: no such file or directory, open 'D:\Max's Courses\nodejs\views\layouts\main-layout.handlebars'

But we renamed our files to hbs. We need to tell handlebar that for layout too it needs to take hbs extension.

In app.js-

app.engine('hbs', expressHbs({

layoutsDir: 'views/layouts/',

defaultLayout: 'main-layout',

express: 'hbs'}));

app.set('view engine', 'hbs');

now everything works fine. Lets use layout in add-product.hbs. code-

add-product.hbs-

<main>

<form class="product-form" action="/admin/product" method="POST">

<div class="form-control">

<label for="title">Title</label>

<input type="text" name="title" id="title">

</div>

<button class="btn" type="submit">Add Product</button>

</form>

</main>

88)Working with EJS

This is my fav and we are going to use it for rest of course. This is like pug out of box supported. We dnt need to register it like we did with handlebars.lets remove handlebars code. Sow we remove app.engine from app.js we just set the view engine to ejs-app.js-

app.set('view engine', 'ejs');

niw ejs has very nice syntax in my opinion and a nice mixture of extended functionalities of pug, so not regarding the html but regarding js code you can use it in your template, that you can do things like , compaision in if statements(like we did in pug), which you could’nt do in handlebars, that is gain possible in ejs, this is thing I like about it and still it uses normal html like handlebars which I personally also like. So lets first prepare 404.rjs. copy and paste 404.html in 404.ejs.

now ejs do not support layouts but we will find some solution to have reusability. This is how you show a variable-

<title> <%= pageTitle% ></title>

The way we pass data remains same-

app.use((req,res,next) => {

// res.sendFile(path.join(\_\_dirname, 'views', '404.html'));

res.render('404', {pageTitle: 'Page Not Found'});

});

Similarly we prepare add-product.ejs. now most interting is shop.js, there we got if statement and loop.

We saw how we output a variable in ejs. Now we can have <%% > block and between it we can write vanilla js code. Now w e enclose code between curly braces that we want to display only on condition. Sow e write normal fi statement as if would in normal js file, just part inside {} of if is not js code it is html code. Shop.ejs-

<main>

<% if prods.length > 0 {%>

<div class="grid">

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">Great Book</h1>

</header>

<div class="card\_\_image">

<img src="https://cdn.pixabay.com/photo/2016/03/31/20/51/book-1296045\_960\_720.png" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

</div>

<% } %>

</main>

Tenw e add else condition-

<main>

<% if prods.length > 0 {%>

<div class="grid">

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">Great Book</h1>

</header>

<div class="card\_\_image">

<img src="https://cdn.pixabay.com/photo/2016/03/31/20/51/book-1296045\_960\_720.png" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

</div>

<% } else { %>

<h1>No products found</h1>

<% }%>

</main>

then we add else block –

<main>

<% if (prods.length > 0) { %>

<div class="grid">

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title">Great Book</h1>

</header>

<div class="card\_\_image">

<img src="https://cdn.pixabay.com/photo/2016/03/31/20/51/book-1296045\_960\_720.png" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

</div>

<% } else { %>

<h1>No products found</h1>

<% } %>

</main>

Now we want to loop over our array of prods and repeat the html. We do it in same way as if statement. We enclose the html that we want to repeat in {}. As we write vanilla js we can use forEach method on prods or we can use form loop, which ever method you like.

Then use variable of loop in html. Shop.ejs-

<main>

<% if (prods.length > 0) { %>

<div class="grid">

<% for (let product of prods) { %>

<article class="card product-item">

<header class="card\_\_header">

<h1 class="product\_\_title"><%= product.title %></h1>

</header>

<div class="card\_\_image">

<img src="https://cdn.pixabay.com/photo/2016/03/31/20/51/book-1296045\_960\_720.png" alt="A Book">

</div>

<div class="card\_\_content">

<h2 class="product\_\_price">$19.99</h2>

<p class="product\_\_description">A very interesting book about so many even more interesting things!</p>

</div>

<div class="card\_\_actions">

<button class="btn">Add to Cart</button>

</div>

</article>

<% } %>

</div>

<% } else { %>

<h1>No products found</h1>

<% } %>

</main>

89)Working on Layout with partials

ejs does not give functionality of layouts which pug and handlebar gives. But we can use so-called partials or includes(this feature is also supported by pug and handlebar) . the idea here is to have some code blocks which you reuse in different parts of your templates and you can share them across templates. So it’s a bit like opposite of layout. Instead of having one master layout where you put your individual view parts into, you have couple of separated view parts which you can merge into the views you are creating and for that I will create a new folder called includes in views, there will create couple of shred code blocks.

We use minus , if we want to output the unescaped html code, that by the way means, that default if you have this syntax with equal sign-

<% = %>

and you would render some variable that holds the string that holds the html code , it would not render that html code, but it will render it as text, to avoid cross-site scripting attacks. With minus you can avoid it and render the html code. we can do this combined with a keyword provided by ejs , include keyword which allows us to include a certain element into this page and then you close that with normal ejs tag. In paranethesis you add a string, that inlucdes the path of file that you want to include. This is relative path.

404.ejs-

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

</head>

<body>

<header class="main-header">

<nav class="main-header\_\_nav">

<ul class="main-header\_\_item-list">

<li class="main-header\_\_item"><a href="/">Shop</a></li>

<li class="main-header\_\_item"><a href="/admin/add-product">Add Product</a></li>

</ul>

</nav>

</header>

<h1>Page Not Found!</h1>

</body>

</html>

Head.ejs-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>

<%= pageTitle %>

</title>

<link rel="stylesheet" href="/css/main.css">

If we use = sign instead of -,our html will be interpreted as string. This is excaping the values. So if oyu have script tag or anything fishy over here it will not render it. It will display it here and will protect you. But if you knw what what yu are doing, we can include it as html.like we do here. similarly we define 3 other files in include.

In navigation, we want to attach class conditionally. So w ehave this in navigation.ejs(it is in includes folder)-

<header class="main-header">

<nav class="main-header\_\_nav">

<ul class="main-header\_\_item-list">

<li class="main-header\_\_item"><a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a></li>

<li class="main-header\_\_item"><a class="<%= path === '/admin/add-product' ? 'active' : '' %>" href="/admin/add-product">Add Product</a></li>

</ul>

</nav>

</header>

Here we used = sign in <% %> because ewe want to output a text in the end , we do not want to have to a block.

90)Wrap up

Lets clean up the code. As we will be using ejs. We do not need the layout folder. it was used for pug and handlebars. Delete all html, hbs and pug files.