Advanced HANDLEBARS

We will convert the **res.get(__dirname + '/public/home.html')** of index.js file in handlebars itself and then we can remove the two html files (home and profile) in **'public'** folder. Let us see how to customize the extensions.

How to customize template engines?

It is tedious to type in the filename along with extension. In the documentation of express handlebars, it is good to go through it once and then start implementing it.

We need to tell the express that the engine we use would be express-handlebar itself but it needs to associate 'hbs' instead of handlebars extension and in app.set, we will tell engine what sort of an extension we want to use. We can provide an object as a parameter to expHbs(). This object is basically what we call as a conflict object. This will allow me specify some default behavior or override my default behavior. We can wrap it onto next line, or we can do it in one line as well.

```
app.engine('.hbs', express({ extname: '.hbs' }))
app.set('view engine', '.hbs')
```

With this configuration, if we click on submit button, then we would get an error, as we have not changed the filename extension. We need to rename the extensions from **.handlebar** to **.hbs** for all the files in the view folder. Now when we click on submit button, it will give us the output.

In the profile.hbs we do not have any html, head or body tag, but the browser itself would be adding it. If we remove the main.hbs file and click submit, it would show an error, that './main.hbs' is not configured. It is because we did not follow the structure. So, all we do is render the profile and send it to the client. From the documentation, for layouts, we see that we have to provide layout: false in the route itself. So, in index.js file, we would change,

`res.render('profile', user)` to `res.render('profile', {layout: false, user: user})`

We have to pass an object; we have to use it as a different property itself. The name property in the profile.hbs does not exit, it exits in the user.name, because we have another layout property defined. Another object in which user.name is a sub-object.

We do not need layout anymore, all we need is the profile.hbs and it gets rendered automatically as the browser add the html, head and body tags.

All these properties do not exit that is why all the parameters are also empty. So, in order to get the parameter to be displayed we have to write user.name, user.age and use the word 'user' to all the parameters.

```
<h1>User Profile</h1>
<h2>I have made handlebars working in my server</h2>
<h1>Your Name is {{ user.name }}</h1>
<h1>Your phone Number is {{ user.phone }}</h1>
<h1>Your website is {{ user.website }}</h1>
```

Now, if we don't want to use any layout in the entire project as it is getting tedious because we have to keep mentioning layout: false. So, we will define that configuration in the express-handlebar configuration in index.js file. There is a **defaultLayout** property which is existing in the handlebar. Similarly, if we have layouts in a different directory, that also we can provide using **layoutDir**.

In the default layout, since we don't want to use any layout all I say is false or any falsy value. We don't have to specify layout: false in every route, and the configuration is configured without any layout part.

```
app.engine('hbs', expHbs({ extname: 'hbs', defaultLayout: false }))
app.set('view engine', 'hbs')
```

and then, we would again change back,

`res.render('profile', {layout: false, user: user})` to `res.render('profile', user)`

Without the layout everything is working accordingly.

Since we have removed the layout, directly whatever is inside my home.hbs and profile.hbs is getting rendered in the specific routes. We need to include the CSS-bootstrap. Let us start with CSS-bootstrap.

In my home.html, we can have a linked tab. To do that we would add the basic html boilerplate and the CSS-bootstrap link. The same would be done with profile.hbs file. We are not sending the whole html page along with the handlebar syntax.

The issue is, we will have repeated-boilerplate code.

If we change the title in home.hbs to Amazon and refresh the page, the tab name would change to **Amazon**, but if we click on submit, it would again change to the title of the profile.hbs file, **Document**. If we have 15 files in view folder and we are supposed to change the name of the title, then we have to change the name of the title in each page. Furthermore, if we want to update the bootstrap-CSS version, then we have to change the link tags in all the 15 files of the view folder.

We can solve this issue using layouts. We will create a folder layout, pre-defined, pre-convention name, because if you change it then you have over-ride the path in the express-handlebar configuration. But if we are going with the default, we need to create a layout folder in the view folder and create a main.hbs file. If we change the name to something else, then we need to update it in the configuration part as well.

view → layout → main.hbs is the default

So, what do we put in layout/main.hbs? We put the html boilerplate and then add the link tag for bootstrap as well or any specific CSS that you have. In the main.hbs we have to put three **`{{{}}}`** and anything inside it will be understood as views, but for convention put body. So, for now will have **`{{{body}}}`**; if we write **`{{body}}`** then it will search for the body element and will show everything inside it on the browser along with the tags and attributes of the tag without rendering it.

Now we can remove the html-boilerplate from **home.hbs** and **profile.hbs** and everything that is common would be put inside the main.hbs file in layout folder. If we now change the title now, then once it is changed in the main.hbs, it would be reflected in all the pages.

Extra Features or More Functionality:

Let us add another file, features.hbs in the views/layout folder. We can add a new route for this file in the index.js file.

```
app.get('./features', (req, res) => {
    res.render('features')
})
```

Let's add something in the features.hbs file to see if it is working or not.

```
<h1>Features</h1>
```

Let us check this by using the URL, **`localhost:3000/features`** this will give us output as **Features**. The margin is at the left which means that bootstrap has been linked to it. We will put it inside a container to center everything.

```
<div class="container">
    <h1>Handlebar Features</h1>
</div>
```

In the actual handlebar, in API reference in the website, in the language guide there is something called as helpers. These provide something extra that we can see with the templating library which is handlerbar in our case. Every template engine will have this feature but named different or with different syntax.

Built-in Helper → https://handlebarsjs.com/guide/builtin-helpers.html#if

So, handlebar provided it under the name helpers. The first helper is **`if`**. We see a sort of if-block.

#if

```
<div class="entry">
    {{#if author}}
        <h1>{{firstName}} {{lastName}}</h1>
    {{/if}}
</div>
```

It has created an if block. In index.js

```
app.get('./features', (req, res) => {
    const featureObj = {
        title: "lorem ipsum"
    }
    res.render('features', featureObj)
})
```

And in features.hbs,

```
<div class="container">
    <h1>Handlebar Features</h1>

    <h2>{{title}}</h2>
</div>
```

If we change the key from title to name, then it would not show any update the value of title but will create a h2 tag in the html page. There are some cases were in you have to use logic inside template or html to handle such cases. So we will use handlebar syntax and will create if block in the template. In feature.html,

```
<div class="container">
    <h1>Handlebar Features</h1>

    {{#if title}}
    <h2>{{title}}</h2>
    {{/if}}
</div>
```

We have provided if block in the template with opening block as **{{#if}}** and closing block as **{{/if}}**. We can say that if title is passed inside the template, then execute the if block. Now if we refresh we will see that the h2 element is missing and only h1 is present. If we provide title in app.get of index.js file,

```
app.get('./features', (req, res) => {
    const featureObj = {
        name: "lorem ipsum",
        title: "it works"
    }
    res.render('features', featureObj)
})
```

Now if we refresh the page, we can see the h2 element being rendered. If we want to render a paragraph tag then if this is true then do this or else do something else.

```
<div class="entry">
    {{#if author}}
        <h1>{{firstName}} {{lastName}}</h1>
    {{else}}
        <h1>Unknown Author</h1>
    {{/if}}
</div>
```

Now, if we add the same to our feature.hbs file,

```
<div class="container">
    <h1>Handlebar Features</h1>

    {{#if title}}
        <h2> {{title}} </h2>
    {{else}}
        <p> {{name}} </p>
    {{/if}}
</div>
```

So, if title is not passed then the name is will be displayed. When we refresh the h2 is working, but if we comment the title in the index.js then the p is working. This is one of the helpers. The use cases would be to design an alert box.

The inverse of **`if`** is **`unless`**. Lets add few more values to the app.get

```
app.get('./features', (req, res) => {
    const featureObj = {
        name: "lorem ipsum",
        title: "it works",

        age: 123,
        ageValue: 431
    }
    res.render('features', featureObj)
})
```

#unless

Going back to the unless property, it will be rendered if the express returns any falsy values. In features.hbs

```
    {{#unless age}}
        <h3>I don't know your age</h3>
    {{/unless}}
```

Here it says, if the age is not passed then render this. Right now, we are sending the age value so the output would not show the h3 part of the html.

If we don't pass the value then the h3 would be shown.

We can have an else in the unless to do the default behavior.

```
<div class="container">
    <h1>Handlebar Features</h1>

    <h1>If Example</h1>
    {{#if title}}
        <h2> {{title}} </h2>
    {{else}}
        <p> {{name}} </p>
    {{/if}}

    <h1>Unless Example</h1>
    {{#unless age}}
        <h3>I don't know your age</h3>
        {{else}}
        <h3>Your age is {{age}}.</h3>
    {{/unless}}

</div>
```

The working is similar to if only the working is inversed. If we refresh now, we can see the h3 element being displayed.

#each

You can iterate over a list using the built-in each helper. Inside the block, you can use this to reference the element being iterated over.

This allows us to iterate over a collection. Rather than creating a html in our route, we pass the collection and based on the collection we can create our own html which keeps things separate. The html related stuff is getting handled in the template itself.

<ul class="people_list">
{{#each people}}
<li>{{this}}</li>
{{/each}}
</ul>

In index.js file, we can add the collection of stuff as an array,

```
app.get('./features', (req, res) => {
    const featureObj = {
        name: "lorem ipsum",
        title: "it works",

        age: 123,
        interests: ['food', 'reading', 'software', 'singing', 'games']
    }
    res.render('features', featureObj)
})
```

Let us create an un-ordered list. We would write <ul> and {{#each}} and tell the express-handlebar what is the collection name, So, {{#each interests}}.

```
<h1>Each Example</h1>
<ul>
    {{#each interest}}
    <li></li>
    {{/each}}
</ul>
```

And for each interest we would render the <li> tag. By default the handlebar provides the usage of `this` keyword. This is similar to the javascript but not the same. So we would add it inside the li tag.

```
<h1>Each Example</h1>
<ul>
    {{#each interest}}
    <li> {{this}} </li>
    {{/each}}
</ul>
```

- food
- reading
- software
- singing
- games

This also has an else block; in case we do not provide anything, we can see in html the ul is coming but it is empty. We can again have an if block, to read the li tag only if interest key is present. But we can do else in each as follows,

```
<h1>Each Example</h1>
<ul>
    {{#each interest}}
    <li> {{this}} </li>
    {{each}}
    <li>No interest</li>
    {{/each}}
</ul>
```

#with

The with helper allows you to change the evaluation context of template-part.

{{#with person}}
{{firstname}} {{lastname}}
{{/with}}

In the index.js file, let us add an address,

```
app.get('./features', (req, res) => {
    const featureObj = {
        name: "lorem ipsum",
        title: "it works",

        age: 123,
        interests: ['food', 'reading', 'software', 'singing', 'games'],
        address: {
            "street": "Kulas Light",
            "suite": "Apt. 556",
            "city": "Gwenborough",
            "zipcode": "92998-3874",
            "geo": {
                "lat": "-37.3159",
                "lng": "81.1496"
            }
        }
    }
    res.render('features', featureObj)
})
```

We have an address which is an object and that itself is another object and inside it we have another object. So, to access it,

```
<address>
    <p>{{address.street}}</p>
    <p>{{address.suite}}</p>
    <p>{{address.city}}</p>
    <p>{{address.zipcode}}</p>
    <p>{{address.geo.lat}}</p>
    <p>{{address.geo.lng}}</p>
</address>
```

We have to address to all the values and this causes repetition. We are getting the entire address in the output. Using with helper,

```
<address>
    {{#with address}}
        <p>{{street}}</p>
        <p>{{suite}}</p>
        <p>{{city}}</p>
        <p>{{zipcode}}</p>
        <p>{{geo.lat}}</p>
        <p>{{geo.lng}}</p>
    {{/with}}
</address>
```

We have **geo** object as well. So, we can have a nested with block,

```
<address>
    {{#with address}}
        <p>{{street}}</p>
        <p>{{suite}}</p>
        <p>{{city}}</p>
        <p>{{zipcode}}</p>
        {{#with geo}}
            <span>{{lat}},</span>
            <span>{{lng}}</span>
        {{/with}}
    {{/with}}
</address>
```

These are all the in-built helpers. There is a way to define our own helpers as well.

Partials:

Partials are normal Handlebars templates that may be called directly by other templates. We can say small blocks or bits here-n-there for the whole system. Similar type of idea we can use here as well.

Let's say we are looking for a header & footer. We can add them in the main.hbs file directly.

```
<body>
    <header>This is my header</header>
    {{{body}}}
    <footer>This is my footer</footer>
</body>
```

We can do this in the layout/main.hbs, but let's try to create this using partial. Because sometime the header itself would be quite huge; ex: Amazon.

We need to new folder called partials in the views folder and inside it we will create another template as header.hbs file. In this file we will add,

```
<header>
    <nav>
        <a href="./">Home</a>
        <a href="/profile">Profile</a>
        <a href="/features">Features</a>
    </nav>
</header>
```

Let us create another file as footer.hbs in partials folder.

```
<footer>
    <h1>My Footer</h1>
    <p>Best Website in the World</p>
</footer>
```

We have created partials, but we don't know how to use it yet. Even though we have created one it is not linked. First, we'll see in the home.hbs itself.

To include the partial, we just need to add the file name in the 2 curly braces in the home.hbs file. We will provide > as well, this will tell handlebar that the requirement is to fetch whatever is in the header partial and put that entire content in the header.

```
{{> header }}
<div class="container">
  <h1 class="text-center display-5">Login</h1>
  <div class="d-flex justify-content-center align-items-center">
    <form action="/profile" method="POST">
      <div class="mb-3">
       <label for="emailId" class="form-label">Email address</label>
       <input type="email" name="email" value="Karley_Dach@jasper.info" class="form-control" id="emailId">
       <div id="emailHelp" class="form-text">We'll never share your email with anyone else.</div>
      </div>
      <div class="mb-3">
        <label for="password" class="form-label">Password</label>
        <input type="password" name="password" value="123abc" class="form-control" id="password">
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
    </form>
  </div>
</div>
{{> footer }}
```

It is only included in our home page, but to have it in all the pages, then we need to put the header and footer in the main.hbs file in the layout folder.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    {{> header }}
    {{{ body }}}
    {{> footer }}
</body>
</html>
```

To use age in the feature part, then the features link itself is gone, because the link is there but the value is gone.