Beginning the Major Project -2

Introduction

This is the final part of creating the directory structure wherein we will be looking at partials and layouts, how to distribute the views folder into different smaller files so that they are easily manageable, and linking MongoDB with the project.

Understanding Partials in Views

- **Partials (Partial code put into another file) -** The advantage of EJS is that it combines data and a template to produce HTML. This makes our code scalable and manageable. One of the most important features of EJS is its use of partials. Using partials, you may write a piece of code just once, and use it at many places as and when required.
- Whenever we have some piece of code that is either very long or a piece of code that is used again and again in different files, we may store that in a separate file, and then re-use it wherever it needs to be added. One such example would be the creation of a NavBar.

Understanding Layout in Views

- **Layouts -** Layouts enable us to dynamically fix content or elements to a page in such a way that even if a different page is requested, the content remains but the page is successfully accessed.

- We can put in any layout that we want the website to look like into one file & whatever variables need to be put in or if the central content needs to be changed, we can just tell it using our view engine that this is the layout of the website wherein some part needs to be changed and can be filled in as in when needed.

Implementing Partials

Setting up our partials -

- Create header and footer sections in **{ home.ejs }**.
- Copy that same section in { **user_profile.ejs** } too.
- The content that is present in both the ejs files [**{ home.ejs** } and { **user_profile.ejs** }] are the variable part.
- We have to create two partials that are {_header.ejs } & { _footer.ejs }.
- Put that header section in {_header.ejs } & footer section in { _footer.ejs }.
- In the {user_profile.ejs } and {home.ejs}, we don't need the header and footer section. Hence we have to remove both the sections from both the files.
- Include that separate file content of header and footer section in {user_profile } & {home.ejs} using following code .

```
<body>
    <%- include('_header');%>
   <!--Variable content that we want to show-->
    <%- include('_footer');%>
</body>
```

NOTE **-** We are using different naming conventions for partial files to make them separate from all the other ejs files. Hence we have used (_filename.ejs }.

Extra - For further read, you can refer to the following link

https://ejs.co/

Creating a Layout

Generally, a website can have multiple different layouts but for now, we are focusing on building a common layout for our whole website.

We are going to use the library for layout that is { **express-ejs-layout** }.

- Install the library { npm install express-ejs-layouts }.
- We have to create a file in the views folder { **layout.ejs** }.
- In index.js { entry point } we need to **require** the library we have installed.
- We have to tell our app to use that library before requiring the routes in an index.js file { entry point }.

```
const expressLayouts = require('express-ejs-layouts');

app.use(expressLayouts);
```

- In layout.ejs copy the same code of { **home.ejs** } or { **user_profile.ejs** } and in the variable part do as follows-

```
<body>
    <%- include('_header');%>
   <!--Variable content that we want to show-->
   <%- body %>
    <%- include('_footer');%>
</body>
```

- Remove everything from { **user_profile.ejs** } & { **home.ejs** } except the variable part in both the files.
- So Layout gets rendered, body gets filled with whatever it is there in { **user_profile.ejs** } & { **home.ejs** }.

    - Express ejs layouts is being used by the app it finds a layout that should be a wrapper that should be covering the { **user_profile.ejs**}, so the wrapper is rendered also the { **user_profile.ejs**} is rendered together with it and it is sent it to the browser.
    - Combining { **user_profile** } and layout, filling the { **user_profile** }in the place of the body and it is sent back to the browser.

Extra - For further read, you can refer to the following link

https://www.npmjs.com/package/express-ejs-layouts

Setting Up Static Files Access

- Firstly we need to tell in which folder the app { **index.js** } should look out for static files.

```
app.use(express.static('./assets'));
```

    - Create a new folder **{ assets }**. In the assets folder ,create three more folders named as **{ CSS , js, images }**.
    - Create a css file **{ layout.css }** and create some basics styles for **{ layout.ejs }** to use.

- Include that style in **{ layout.ejs }**.

Static Files For Pages

The app should automatically render the link tag into the head of the layout so to do that -

- Just below **app.use(express.static ('./assets'))** in **{ index.js** } {Entry point } - Extract style and scripts from the sub pages into the layout using -

```
app.use(express.static('./assets'));
app.set('layout extractStyles', true);
app.set('layout extractScripts', true);
```

- Go to **{ layout. ejs }** wherever we need to put up the style tag we just need to do -

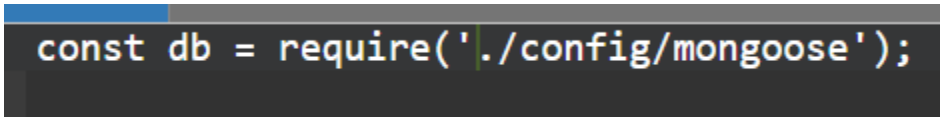
```
<head>
<%= style %>
</head>
<body>
<%= script %>
</body>
```

- Whenever there is something termed as a **link tag** or a **script tag** in the body, they are automatically put on the top, at their correct positions.

Linking our MongoDB using Mongoose

We need to set up our database but before that commit all the changes that have been done till now.

- Inside the config folder create a new file **{ mongoose.js }**.
- Install mongoose **{ npm install mongoose }**.
- Require mongoose in **{ mongoose.js }**.
- We need to provide a connection to the database
- Whenever there is an error while connecting to the database we need to console that, This will display the **console.log** like an error.
- If the database is connected properly one callback function will be called.
- To make that **{mongoose.js }** file usable we need to export it.
- Require mongoose in **{ inde.js }** [Entry point].



```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/codeial_development');

const db = mongoose.connection;

db.on('error', console.error.bind(console, "Error connecting to MongoDB"));


db.once('open', function(){
    console.log('Connected to Database :: MongoDB');
});


module.exports = db;
```

Summary

- Created **{ index.js }** and require the express library.
- We ran the server on **port 8000**.
- Created different folders for **controllers, models & views.**
- We created routes **{ index.js }** centralized path for all other routes.
- We created different controllers for different routes.
- We render our views using the ejs view engine.
- Partials and Layout introduction
- Common Layout creation
- We set up the static files
- For models, we set up the database.
- Install mongoose library