Project Implementation & Navigation Guide

The purpose of this document is to help the readers of my code understand what on earth I have written and, most importantly, how I have gone around writing it

Overview

The project, movies reviewing site, is on the MERN stack. The project uses **bootstrap** (front-end), **ExpressJS** (middleware), **NodeJS** (server) and **MongoDB** (database). Moreover, a templating engine, called **handlebars**, has also been used (which introduces all the complexity in the project and compels me to document it in the first place)
Remember, **app.js** is the entry point for the project

ExpressJS – Serving Static Files

This part is very easy; using ExpressJS framework, we serve the bootstrap files. The basic syntax used is: **app.use('/css', express.static('./public/home/css'));** for the home and **app.use('/css', express.static('./public/admin/css'));** for the admin. Hence, we created a middleware and served all the files (css, js, img, icon and fonts) statically

Be noted that the middleware adds request for these files (in the request object) and then sends that request to the server for these static files, and the files get served automatically. You don't have to arrange any paths or anything else. Just put all the template files (js, fonts, css, img, icon) in /public/admin and in /public/home

At one point, the static files were not being served. This consistently happened for all sub-routes on a main route. For example, after going to **localhost:64000/admin** and **localhost:64000/home** files are served perfectly. But on the sub-routes, like **localhost:64000/admin/view-reviews** and **localhost:64000/admin/add-genre,** static files were not served. All you have to do here is add absolute path (not relative) request to the server. You do this by prepending just one, single forward slash ("/") to the lines, like **<script src = "js/main/jquery.3.5.min)>** so that they become **<script src = "/js/main/jquery.3.5.min)>.** Moreover, you do the same to the CSS file requests in the href of the same files (in the head of the html)

ExpressJS – Managing Routes

This one is relatively trickier but not so much. The complexity here is added by both ExpressJS as well as handlebars

Using ExpressJS, we make sure that home and admin routes are managed separately in separate files at **routes/admin/main.js** and **routes/home/main.js** by the help of Express router that can be included by writing **express.Router().** It is also exported from those files

Handlebars is a little tricky. The following points must be kept in mind while using handlebars

- **Views** – Handlebars framework always looks (by default) into views directory. So, we create that at first. Inside there, we have layouts directory (use exact same name), partials directory (use exact same name), admin directory and the home directory
- **Layouts** – Handlebars framework makes you create layouts. When creating routes, on top of all the route files **(app.js, routes/admin/main.js** and **routes/home/mian.js),** you have to inform what the default layout of all the routes (that follow in this file) is going to be. That layout is basically the code that will be hardcoded to all the files that are set up on this route

  For example, in the home side of the projects, all the files inherit navigation and footer. This is because navigation and footer are set-up in **views/layouts/index-home.handlebars.** In this file, where you see **{{> *file-name*}}** means that handlebars should peek into **views/partials/file-name** and include that file's code here (navigation code / footer code etc.). Whereas, if you see **{{{ body }}}** in this file, it means that handlebars should not hardcode anything here (like footer or navigation) from some file, rather code will be added here dynamically (as we will serve the file when somebody comes on this route). Hence, to add code dynamically in place of {{{ body }}}, we simply serve some file on this route

For example, on **localhost:64000/home,** we render **page-content.handlebars** file (from **views/home/page-content**) and on **localhost:64000/faq,** we render **faq.handlebars** file (from **views/home/faq**). The content dynamically rendererd on these files will be added in place of {{{ body }}} of the layouts file

Remember, the main file of the template **(index.html)** has to set-up in **views/layouts** folder for the admin as well as for the home

- **Partials** – These are the code snippets that you want to include (hardcode) in every file on a specific route. For example, **views/partials/admin-sidebar.handlebars** is added to all the files of the admin/ route

ExpressJS & Handlebars – Parameters to res.render( )

Parameters refer to the same thing as they do in React-Native: while moving from one "screen" to another, you simply pass some data in JSON format so that it can be accessed in the other file. Whatever we pass in parameters can be displayed easily using **{{ *something* }}** in the other file. In case it is an array, you can also run a loop in handlebars to go over all the values. **{{#each data}} *display data here* {{/each}}** can be used to use a for loop. Similary, conditional statements can also be used in handlebars using {{**#if**}} **….** {{**/if**}}

JavaScript & Handlebars – Helper Functions

Helper functions, set in helpers directory, are JavaScript functions that are exported from a file and imported in the server-containing file (**app.js**) and added to the **app.engine(….)** function. These functions allow us to add more functionality (logic, basically) in handlebars which it does not have already, as handlebars closely resembles HTML which itself is not a programming / scripting language in the first place

For example, in order to read values from an array inside a handlebars for loop (nested loop), we have to depend on these external helper functions, as nested loop gives issues in handlebars

Handlebars – lean( ) for Prototype Access

Sometimes, for security reasons, handlebars does not display data coming in as a parameter. Now, we have to add a “lean()” method before writing the “then()” handler, and the issue gets resolved. There is a longer, tedious work-around as well (that requires installing modules) which I obviously did not follow

This workaround allows us to access unprotected prototype. This can cause malicious code to be run on the server. But if we are 100% sure that we and we alone are writing code, then we can use this method to overcome the restriction that handlebars has put there for users

On the other hand, if we are taking code from our user and executing it in the server (which we are not doing), then we should never enable unprotected prototype access using this method, because then there will be issues and security vulnerabilities that we do not want

Uploading Files – Images and More (express-fileupload)

In order to upload files, a separate module has been used and set-up in server **(app.js).** It adds to the incoming request object the functionality to access files, and the files can be accessed later using **req.files.*object name*** in the specific route

ExpressJS & Handlebars – Flash Messages (connect-flash)

Here, we have installed a module for sending flash messages. Flash messages are the messages that are sent to a specific page after a user is redirected to it. For example, after updating the movie, you take your user to the **/admin/movies/view-all-movies** route. Here, you need to display confirmation messages for the movie being updated successfully. So, you take help from flash messages

Here, we have to set the flash() method in our middleware. Secondly, we have to add flash message to the outgoing response object (as the middleware has access to the incoming request object as well as the outgoing response object) by writing **app.use((req, res) => {*outgoing flash message*});** Finally, on the specific route from where we want to send the flash message, we have to write this: **req.flash('flash-message-name', 'flash-message-content');** Now, this message can be displayed in handlebars file using **{{ *flash-message-name* }}** syntax

Sessions (express-session)

We can also set sessions by setting them up in our middleware. Afterwards, our sessions will be available in the entire application via the request object (to which the sessions object was added)