Beginning the Major Project 1

Introduction

We are beginning with our major project and for that, we have to build our directory structure which should be very scalable, which means large files should be split into multiple small files and placed in the right folder. We should have distributed code that is centrally managed by express.

The steps we are going to follow for beginning the major project are-

- Set up directory structure
- Set up express.js
- Initialize Git onto it
- Set up Routers & Controllers
- Set up view Engine
- Setting up the Partials and Layouts
- Create static files
- Connect to MongoDB {Database}

Setting Up Directory Structure

Follow along with the steps -

- Create a new folder in your workstation {Codeial}.
- Create a new file which is the entry point **{index.js}**.
- Initialize the project using {npm init}

- Create multiple directories for a scalable project using terminal {mkdir routes controllers models views config}
- Routes will be storing all the paths
- According to routes, which function would be called routes will be stored in the controllers folder mapped into different files.
- Models will contain different schemas.
- Views will contain different HTML files.

Starting Express

To fire up the server we should follow the steps-

- Install express using {npm install express}.
- Require express in the entry point file {index.js}.

  Note - *By default, websites run on port no 80.*

- Interpolation - Instead of using (,) or (+) for concatenating two strings, we can embed our variable inside {} the string using backticks (` `).
- To include the variable inside the string this should be done "${` `}". So Instead of those [console.log("Error: ", err)], we can do [console.log(`Error: ${err}`)].
- Anything inside [${}] is to be evaluated.
- Run command {nodemon index.js} to run the server.

```
const express = require('express');
const app = express();
const port = 8000;


app.listen(port, function(err){
    if (err){
        console.log(`Error in running the server: ${err}`);
    }

    console.log(`Server is running on port: ${port}`);
});
```

Adding Git & NPM Start

To make the project scalable we need to take care of some important points -

- We need to initiate the project as a git repository to track all of the changes that were done.
- Different branches for different features.

To Get rid of the repeated command that we were using to start the server {nodemon index.js}

- Go to package.json which has scripts part {Which gives us the option to run some predefined commands}.
- Give it a key name “start” and in the value give the command used for starting the server{nodemon index.js}.
- To run the server we have to write the command only once that is {npm start}.

To initialize as a git repository { We need to track all the changes and in case anything breaks down we can come back to it. Following are the steps -

- In the terminal do {git init}.
- To add remote origin { git remote add origin “URL” }.
- Git add. {Add everything that is committable}.
- Git commit -m “message”.
- To check the status of the git commit we can do a git log.

There is a folder, named **git_modules** which is quite large due to the installed libraries present over there. Therefore, we need to ignore committing the **node_modules** folder for that -

- Create a new file named as { .gitignore } .
- In that file write **node_modules/ {This folder is ignored now}**
- You can check this using the git status command which will not show the **node_modules** folder.

NOTE - System that records changes to a file in a project over time is called **version control.**

Setting up the Express Router

Routes are the entry point for all the requests from the browser.

Setting up your basic routes -

- In the routes, folder create a new file **{index.js}**, which is the entry point for all the routes.
- The App index.js file will send in the request to the routes/index and this will further route us to all the different routes files that will be present in the website code.
- Express contains a module, express. router, which will help in separating the app routes and the controller.
- To check whether the routes are loaded, we will use the console.log in routes **{index.js}.**

Routes**{index.js}**

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

const router = express.Router();

console.log('router loaded');


module.exports = router;
```

Index.js

```
const express = require('express');
const app = express();
const port = 8000;

// use express router
app.use('/', require('./routes'));


app.listen(port, function(err){
    if (err){
        console.log(`Error in running the server: ${err}`);
    }

    console.log(`Server is running on port: ${port}`);
});
```

NOTE - Every time we require express no new instance of express is created instead it will fetch the existing instance

Create a Controller

After route setup, it's time to set up the controller {Action that is taken up for any route}. A group of actions bundled together is known as a controller.

- Create a new inside controller **{home_controller.js}**.
- **{home_controller}** has an exported function that is home.
- Create a function that will take two parameters request and response by default.
- In the request-response, we have to return something {It can be HTML also}.
- We need to access that function in routes.
- Require the **{home_controller }** inside index.js of the route folder.

NOTE **-** {../} To go a step above and move parallelly and get inside the folder.

- To access the home_controller action in routes **{index.js}** file we need to do-

index.js file -

```
router.get('/' , homeController.home);
```

Controller**(home_contoller)**

```
module.exports.home = function(req, res){
    return res.end('<h1>Express is up for Codeial!</h1>')
}
```

Create another Controller & Router

Create more routes and more controllers and commit the changes till now.

Steps

- Git add.
- Git commit -m “message”
- Create a file in controller**{ users_controller }**.
- Export an action corresponding to **{ users_controller }** that is profile.
- Return HTML
- Create a route for the controller to be accessible inside the route folder named as **{users}**.
- Repeat the same steps that we used to create the previous route **{index.js}**.
- We need to map a route to the **{users_controller}** profile action.

  Users.js file -

```
router.get('/' , usersController.profile);
```

  - Index router was accessing the **home_controller**. Considering index.js as the index or root of the route, we want this route to be controlling all the other routes or having a list of all the other routes. For that, we need to do -

    Index.js file -

```
router.get('/' , homeController.home);
route.use('/users',require('./users'));
```

- Any request to the home page goes to homeController.home, any request to /users goes the users route and where further mapping is done.

ASSIGNMENT - You need to set up another controller **{posts_controller}** which will be for users post, add one more route, one more action in the **users_controller,home_controller,** and render it.

Installing EJS & Set Up the View Engine

We need to send something back to the browser in HTML format from an HTML file from the Views folder using View Engine.

App. set is an object where different properties are predefined {keys are present}. Whenever we put a value of those keys, the express app takes up those values and does something with those values. The view and view engine property is also defined here.

NOTE - By default, express routing is not case-sensitive.

- Install ejs using {npm install ejs}.
- Tell the app to use ejs as the view engine using the below line code.

```
app.set('view engine','ejs');
app.set('views','./views');
```

NOTE - Try to commit after every lecture.

Create a view for home

We will now render the view from the homeController.

- Create a new file inside the views folder **{home.ejs}**.
- Create a normal HTML file.
- In homeController we will send the response by rendering the ejs file **{home.ejs}.**

```
module.exports.home = function(req,res) {
    return res.render('home' ,{
        title : "home"
    });
}
```

ASSIGNMENT - Go Ahead and create the same view file for usersController. Access it via your action inside the controller and send it back to the browser.

- For further understanding, you can refer to the following link and study blocks and layout.

  https://ejs.co/

Summary

We have distributed our directory structure and also created our views from the template engine for this project.