My First Express App:: A List Of Contacts!!

Web Framework: Introduction

A web framework eases our way of coding, Let's see how.

Let's suppose we have a single file and there we write all our code. Now suppose that this file grows in size and gets difficult to manage. Hence as a solution, we divide that file into multiple files. even these files can grow in size. Hence we grow on dividing it further. It can be possible that some files may provide common functionality. Thus, we can separate this particular piece of code that provides common functionality and then import that file into all of the other files. We can't maintain a separate folder system which will become very messy, so here frameworks come to our rescue. What it does is, it makes your folder system structured {your application folder structured} and the web framework is watching over it by giving you access to all the common code that is present and also managing all the functionality.

Web Framework makes your life easier as a developer.

- **Note**: We should be very careful while choosing a web framework according to their use case.
- We will be using Express.js as our web framework.

Understanding MVC { Models,Views,Controller }

There are different ways of structuring the code, but the one that is quite popular now is the MVC structure {Models, Views, and Controller }. We will also be using the MVC structure to build websites in this course. {There are others also, for example - MVVM (Model, View, and ViewModel) }.

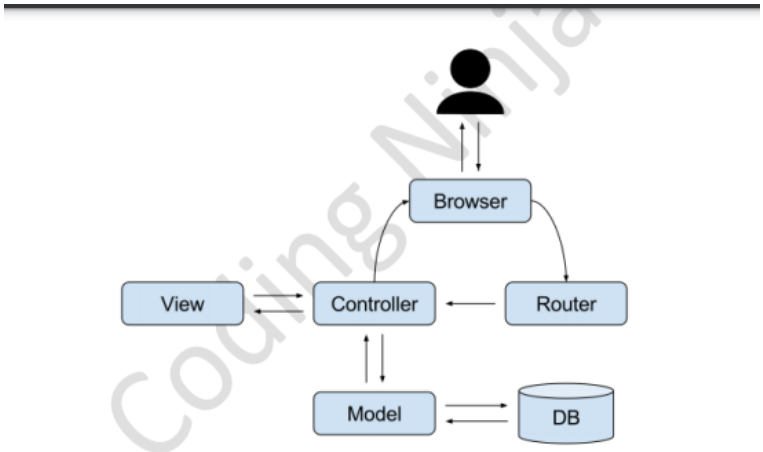
When we send a request, there is a set of functions that handles all the requests that are coming in from the user {for eg - home, user, profile}. Now, each of them is mapped to a function that function is written in a file or a set of files called controllers so whenever a request comes in it goes to the router where it goes into a map wherein {home is mapped against a function named as home controller, etc} that function is get called and that function has all the logic of what data should be sent back {Controller - It is a set of function or single function} and it finds out the HTML file {in our case} it could be JSON also depending upon different formats of the file and read that file and it sends back to the browser.

All your data should be stored permanently. Here, databases come into the picture. {File that holds all your data}. We need some functions to write and read from that database, such files are called models. {Models and Databases are interacting}.

We also need controllers. Controllers will ask models to fetch the file data which is present inside databases depending upon user requirement to fill that data into HTML and bring it back to the browser {Controller is the central part}.

A lot of frameworks use MVC structure like -

- ruby on rails
- Express.js,
- Sails
- python
- Django, etc.



Why Express.Js

- Express doesn't tell you to design your folder in a specific manner {As a beginner it may be possible that one can get into creating a wrong folder structure}.
- Express is a framework and many other frameworks are built on it. { sails.js}.
- Create a folder {contact_list} and then make a file inside it {index.js}.

- In the terminal, type the command to initialize the server (npm init) and fire up your server. {As discussed in previous modules}.
- To install express, type 'npm install express' in the terminal.
- Node_Modules folder will appear while installing express which has a lot of JS and JSON files; these are multiple NPM modules or NPM libraries, packages that are required to run express.js.
- When express is installed, it installs all the other dependencies in the node_modules folder. This folder is quite a heavy folder.
- In package.json you will see one dependency of the express. Package-lock.json consists of lists of further dependencies including express.

    - **Note**: node_modules folder contains libraries that node.js requires.

Setting Up Express Server

- We need to 'require' express just like any other library in {index.js} file
- Set up the port on which our application will run.
- Fire up the express.
- Run the server using {app. listen}. Then listen on a port and there will be a callback function that will take one argument as an error.f there is an error, the console will print the error. If there is no error, the console will print a successful message.
- Fire up the server on the terminal by the command {nodemon index.js}. As we have nothing to return, express will return a message {cannot get}. Previously, it was just loading without the presence of express.js.

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

const app = express();


app.listen(port,function(err){
    if(err) {
        console.log("error in returning the server",err);
    }
       console.log("Yupp !! my server is running on port",port);

});
```

***EXTRA**: You can read about Express from the given link below.*

https://expressjs.com/

Returning Response from the server

All the switch cases that were present earlier were now just removed using express and everything is modeled using express. Now we can easily define as many functions as we want to, without bearing the pain of writing a lot of code.

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

const app = express();


app.listen(port,function(err){
    if(err) {
        console.log("error in returning the server",err);
    }
       console.log("Yupp !! my server is running on port",port);

});
```

***EXTRA*: Practise rendering different pages as your mini assignment.**

HTTP Request

Let's get deep into the Request-Response cycle and find out the types of requests that your browser sends to the server and the different types of data that your server can send back.

Some Important part of requests

- Default Encoding - { Default encoding for express is **utf-8** }

  { Encoding - The way your binary data is encoded into appearing like "a" (alphabet or numbers or characters) }.

- Host - Whenever a request is coming in from the browser, the Host gives us information about who is sending us the request.
- User-agent - Which browser engine is sending the request.
- Accept - What all data can the browser accept back.

Different Types of Request

- Get - Whenever there is a server and one has requested some data that is already present that is known as getting the request.
- Post - Whenever we want to send some data, it makes some changes in the database and then gives us the response. {Server responds in some form of data but that data is the result of some changes done in the database}.

The requests listed below work only with AJAX:

- PUT - There is something already present but we are putting it somewhere else.

- PATCH - If we want to patch or change the existing data.
- DELETE - When we want to send some id to the server and delete it.

Note: Your HTML forms either use get or post request.

How to use -

app.get
app.post
app.delete

Template Engine: What?

Template Engine provides us the feature to create an HTML **template** with minimal code. Also, it can inject data into the HTML **template** on the client-side and produce the final HTML.

Template Engine: Some Examples

- Handlebars - using hbs - Handlebars is a simple templating language. It uses a template and an input object to generate HTML or other text formats. Handlebars templates look like regular text with embedded Handlebars expressions
- EJS - {Embedded JS} - EJS is a simple templating language that lets you generate HTML markup with plain JavaScript
- Pug - Pug.js is a template engine for Node.js and browsers to render dynamic reusable content

***EXTRA*: You can read more about Express.**

Setting Up Template Engine

- Install ejs {npm install ejs}.
- {App.set} will set up the value named ejs.
- The path is an inbuilt module in the node which we require to make our views folder
- {__dirname}: This will show the directory from where the server has started.
- Create a folder named views.
- Path. join to join the current directory with views dynamically
- Create a new file named as {home.ejs} to render the HTML file .

{ Index.js }

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

const app = express();

app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));


app.get('/', function(req, res){
    return res.render('home');
})


app.listen(port, function(err){
    if (err) {
        console.log("Error in running the server", err);
    }
    console.log('Yup!My Server is running on Port', port);
})
```

{ Home.ejs }

```
<html>
<head>

</head>


<body>
<h1> Rendered From EJS View !! </h1>
</body>

</html>
```

A Quick Revision

- Installed EJS template engine.
- App. set statement { 1 - setting up the view engine 2 - setting up the view path}.
- Set up our views directory and file.
- Rendered using response. render.
- The extension was ejs for view file

Rendering Our First Page

Whenever we want to display a variable just do { <%= variable_name %> }. If we want to show the title on the page dynamically.

```
<head>

<title>

<%= title %>

</title>
</head>
```

Some important syntax

- If we want to print something we can use this -> {<%= %>}.
- If we want to use a loop or we do not want to print anything use {<% %>}
- Loop statement

```
<ul>
<%
   for(let i = 1 ; i < 6; i++) {%>
     <li>
     <%= i%>
     </li>
    <%}%>
</ul>
```

- Conditional statement

```
<%
    if (10>12) {
        %> <p> Condition in True! </p> <%
    }
    else {
        %> <p> Condition is False! </p> <%
    }
%>
```

Locals In views

The variables that are being accessed in templates or views are known as **locals**.

Instead of writing {<% title%>}, we also write as {<% locals.title%>}.

Locals is a global object and whenever we are passing something from a template that is known as context like {title} then one of the keys present in response is locals, this key is an object, and when we pass {title} or the whole object gets substituted over there. Thus, these locals will be available globally to all your views whichever template engine you are using.

Summarizing

- We have set up our file and folder structure {index.js } which is used to send some contacts to the browser which is present in ram and is not permanent.