Intro routing and listening & middlewares

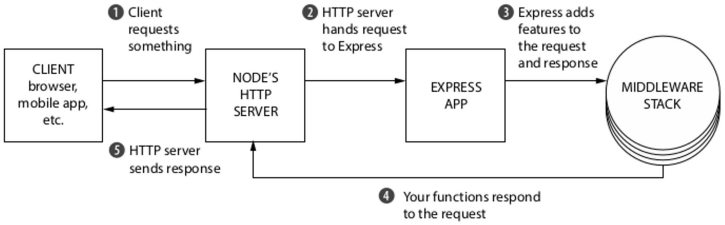
Monday, January 31, 2022 9:44 AM

Routing level web framework

Expressjs is not a solution for everything in nodejs
Express gives minimalist functionality to node js
Express js is a bunch of middle ware framework.

What is express and its use?
It is web framework of nodejs and it provides minimalist functionality for nodejs

Expressjs is a collection of middlewares.
Eg: auth, parse, body, cookie, session



(Sir dictates)

Express def

Express is a minimal and flexible node.js web application framework that provides a robust set of feautures for web and mobile applications.

Alternative node js framework
Featherjs
Locomotive js
Nestjs
Salesjs
Loopbackjs
Keystonejs

How to use express js (in cmd)

1. Create package.json file
   npm init -y
2. Install express js
   npm install express
3. You will notice dependency in package.json

Build an express app

Step 1.

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

Step 2. creating express application using top level function

```
Creates an Express application. The express() function is a top-level function exported by the express module.

const app = express();
```

Step 3. Basic routing

```
//basic express routing
app.get('/', (req, res) => {
    res.send("ok this is express");
})
```

npm start on command prompt

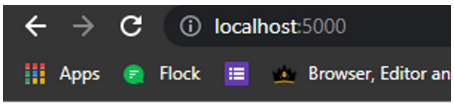
```
C:\Users\bhuvan\Desktop\MERN\Express\Express app>npm start

> express-app@1.0.0 start C:\Users\bhuvan\Desktop\MERN\Express\Express app
> nodemon server.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
server is online on port no 5000
```

Complete code basic

```
const express = require("express");
//Creates an Express application. The express() function is a top-level function exported by the express module.
const app = express();
//basic express routing
app.get('/', (req, res) => {
    res.send("ok this is express");
})
//listen a port
app.listen(5000, err => {
  if (err) throw err;
  console.log("server is online on port no 5000");
});
```



ok this is express

Middleware

Expressjs is a middleware framework.
Before request object goes to server we can inject or trigger middlewares (or additional functionalities)

Middllewares are objects or functions which add functionalities.

Sir dictates
Express Middleware are functions that execute during the lifecycle of a request to the express server.
Each middleware has access to the HTTP request and response for each route (or path) its attached to. This "chaining" of middleware allows you to compartmentalize your code and create reuseable middleware.

You create a middleware using app.use() method

```
app.use(function (req, res, next){
```

Use method takes three parameters request, respone, next
Next is very important otherwise the middleware is stuck and response won't end.

Sample code

```
//adding middleware code
const express = require("express");
const fs = require("fs");
const app = express();
//add middleware to request
app.use(function (req, res, next){
    let date = new Date().toLocaleTimeString();
    console.log(date);
    let text = fs.readFileSync("./text.html", "utf-8");
    console.log(text);
    next();
});
//basic express routing
app.get('/', (req, res) => {
    res.send("ok this is express");
})
//listen a port
app.listen(5000, err => {
  if (err) throw err;
  console.log("server is online on port no 5000");
});
```

Cmd o/p

```
C:\Users\bhuvan\Desktop\MERN\Express\Express app>npm start

> express-app@1.0.0 start C:\Users\bhuvan\Desktop\MERN\Express\Express app
> nodemon server.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
server is online on port no 5000
11:54:00 AM
<!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>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipisicing elit. Doloribus
      debitis quia non, doloremque corrupti minus incidunt quod quo eligendi
      reiciendis nesciunt facere fuga vero qui ratione! Ex ipsam modi sunt!
    </p>
  </body>
</html>
```

IMP	Middleware functions are functions that have access to the request object (req) and the response object(res) and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next. Middleware functions can perform the following tasks or uses of middleware: 1. Execute any code. 2. Make changes to the request and the response objects. 3. End the request- repsonse cycle. 4. Call the next middleware function in the stack.

If the current middleware function does not end the request -response cycle, it must call next( ) to pass control to the next middleware function. Otherwise the request will be left hanging.

An express application can use the following types of middleware.

- Application - level middleware
- Router-level middleware
- Error- handling middleware
- Built - in middleware
- Third - party middleware

For rendering html files

```
app.get("/", (req, res) => {
  res.sendFile(__dirname + "/index.html");
});
```

sendFile(path+"filename");
Automatically sets the Content-Type response header field. The callback fn(err) is invoked when the transfer is complete or when an error occurs. Be sure to check res.headersSent if you wish to attempt responding, as the header and some data may have already been transferred.

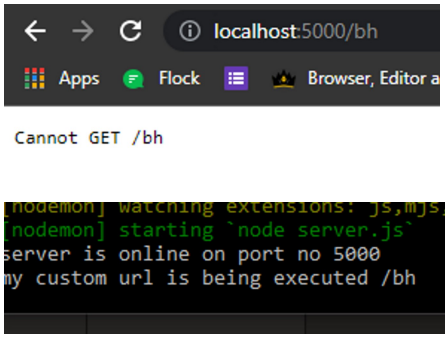
Built-in middleware

```
//built-in middleware
express.static()
express.json()
express.urlencoded()
```

Static middleware
To serve files such as images, CSS files, and JS files, use the express, static built in middleware function in Express. Express looks up the files relative to the static directory, so the name of the static directory Is not a part of the URL…

```
//apply built-in middleware express.static()
app.use(express.static(__dirname + "/public"));
```

```
//custom middleware
app.use(function (req, res, next) {
    if (req.url === "/bh") {
        console.log("my custom url is being executed /bh");
    }
    next();
})
```



Dynamic server side rendering For multipage applications

MVC- model view controller
Template engine - provides dynamic functionality

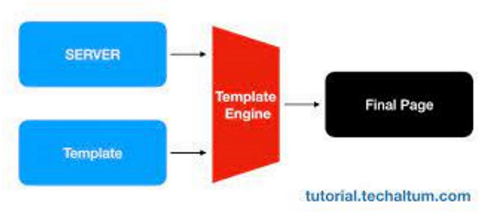
A template engine enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transfprms the template into an HTML file sent ot the client .

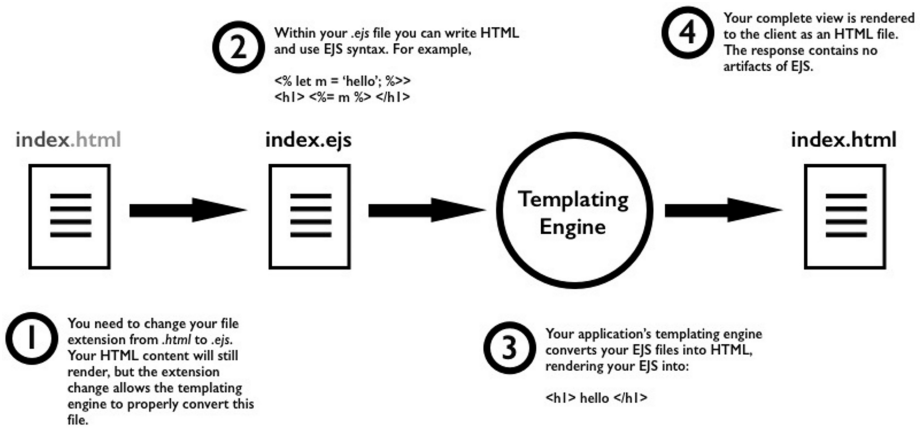
Template engine takes logic from server and sending it as a static file to browser.
Template Engine is used only for DYNAMIC PAGES
Model - database
View- template engine
Control - Server

We do not use MVC for single page applications (but we use only MV for single page.)

Top 5 JS Template engines

- Mustache
- Underscore templates
- Embedded JS templates or EJS
- Handlebars JS
- Pug





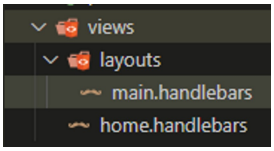
https://www.section.io/engineering-education/express/

How to use
Install third party middlewares

npm install express-handlebars
https://www.npmjs.com/package/express-handlebars

Handlebar folder structure
Files in blue

1. Root level create folder "views"
2. Inside "views" folder create one folder called "layouts"
3. Inside "views" create a file called "Home.handlebars"
4. Inside "layouts" create a file called "Main.handlebars"



Implementations in code
Step 1:

```
const { engine } = require('express-handlebars');
```

Step 2:

```
app.engine("handlebars", engine());
```

Step 3:

```
app.set('view engine', 'handlebars');
```

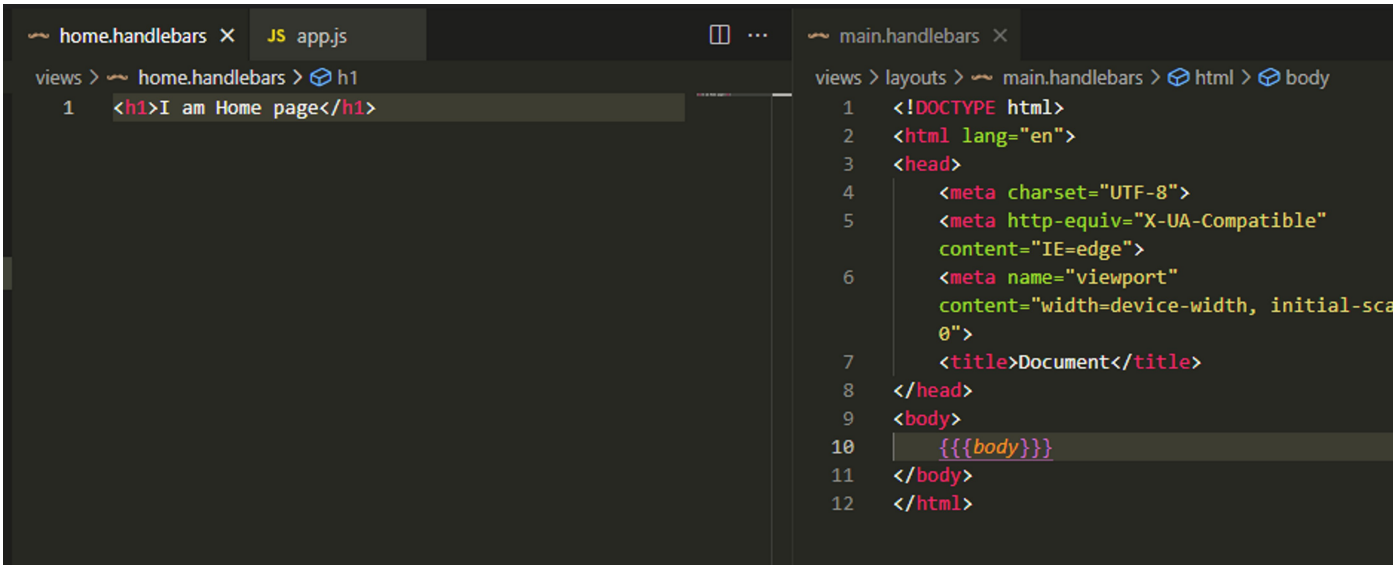
Step 4: optional step, automatically happens

```
app.set("views", "./views");
```

Step 5: go to main.handlebars in views/layout and go to home.handlebars
Main.handlebars is the root file
Use triple curly braces

```
    {{{body}}}
```

Content from home.handlebars goes to main.handlebars



Server.js code

```
// dynamic template for multi page
const express = require("express");
const app = express();
const { engine } = require('express-handlebars');
app.engine("handlebars", engine());
app.set('view engine', 'handlebars');
app.set("views", "./views");
app.get('/', (req, res) => {
    res.render("home.handlebars");
})
//listen a port
app.listen(5000, err => {
```

```
  if (err) throw err;
  console.log("server is online on port no 5000");
});
```

Ma'am notes
Day 1 :
Express JS

Express:

-> express is a minimalistic functionality middleware

Express Introduction:

basic example:
server1.js:

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

//its a top level function
const app = express();

// basic routing
app.get("/", (req, res) => {
res.send(`<h1>Hello Express JS</h1>`);
});

app.listen(5000, err => {
if (err) throw err;
console.log('server running in port 5000');
});
```

Middlewares:

-> these are the plain Js objects.
-> these is injected before sending the request to server.
ex:
before going to office , biometric is important. so biometric is a middleware function.

-> Express middleware are functions that execute during the lifecycle of a request to the Express server.
-> Each middleware has access to the HTTP request and response for each route (or path) it's attached to.
This “chaining” of middleware allows you to compartmentalize your code and create reusable middleware.

-> its like a broker
-> diagram 1

Adding middleware to request: (to inject the middleware)
---
app.use()

sytx: app.use(req,res,next)

to send request for next middleware:
---

next()

ex 1:

```
const express = require("express");
const app = express();

// add middleware to request
app.use((req, res, next) => {
let date = new Date().toLocaleTimeString();
```

```
console.log(date);

// passing the middleware for next request
next();
});

// basic routing
app.get("/", (req, res) => {
res.send("helloo world!!!");
});

app.listen(5000, err => {
if (err) throw err;
console.log("server running in port 5000");
});
```

ex 2: to read a text using fs and date middleware

```
const express = require("express");
const fs= require("fs");
const app = express();

// add middleware to request
app.use((req, res, next) => {
let date = new Date().toLocaleTimeString();
console.log(date);

let text = fs.readFileSync("./index.html", "utf-8");
console.log(text);
next();
});

// basic routing
app.get("/", (req, res) => {
res.send("helloo world!!!");
});

app.listen(5000, err => {
if (err) throw err;
console.log("server running in port 5000");
});
```

Definition of middleware: (imp)

Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.The next middleware function is commonly denoted by a variable named next.

Middleware functions can perform the following tasks:

-> Execute any code.
-> Make changes to the request and the response objects.
-> End the request-response cycle.
-> Call the next middleware function in the stack.

note:
If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

An Express application can use the following types of middleware:

1. Application-level middleware
2. Router-level middleware
3. Error-handling middleware
4. Built-in middleware
5. Third-party middleware

ex 1: Transfer the file at the given path.

sendFile(dirname,path)

soln:

```
// to send the file and image as a response
// sendFile(dirname,path of file)

const express = require("express");
const app = express();

// basic routing
app.get("/", (req, res) => {
//!to send a file
// res.sendFile(__dirname + "/allmiddlware.html");
//!to send a image
//res.sendFile(__dirname + "/tyimage.jpg")
});

app.listen(5000, err => {
if (err) throw err;
console.log("server running in port 5000");
});
```

Built-in middleware:

Express no longer depends on Connect.

Express has the following built-in middleware functions:

1.express.static()
2.express.json()
3.express.urlencoded()

note:

1. express.static : serves static assets such as HTML files, images,css,js(dom) and so on.

2. express.json: parses incoming requests with JSON payloads. NOTE: Available with Express 4.16.0+

3. express.urlencoded : parses incoming requests with URL-encoded payloads. NOTE: Available with Express 4.16.0+

1. Static() middleware:
*) static built-in middleware function in Express. Express looks up the files relative to the
static directory(public folder)

ex 1: // to send the static files to browser

```
const express = require("express");
const app = express();

// apply the built-in middleware
app.use(express.static(__dirname + "/public"));

// basic routing
app.get("/", (req, res) => {
res.sendFile(__dirname + "/index.html");
});

app.listen(5000, err => {
if (err) throw err;
console.log("server running in port 5000");
});
```

ex 2: with custom and built in static middleware

```
const express = require("express");
const app = express();

// to apply custom middleware
app.use(function (req, res, next) {
if (req.url === "/custom") {
console.log("Im a custom middleware");
res.send(`<h2>"Im a custom middleware"</h2>`);
}
next();
});
```

```
// apply the built-in middleware
app.use(express.static(__dirname + "/public"));

// basic routing
app.get("/", (req, res) => {
res.sendFile(__dirname + "/index.html");
});

app.listen(5000, err => {
if (err) throw err;
console.log("server running in port 5000");
});
```

---

ex 3: use one html template and render using express app

paste all the templates in public folder and call it using static()

Server side rendering of contents:
---
jsp -- java
php
asp -- dotnet

*) all acts like templating engine.
*) applications are of two types: SPA and MPA

---
to render dynamic html template to browser from server: (SSR folder)

-> need to use template engine
-> It provides to render contents dynamically..
ex: react uses jsx

note;
*) template engine is old way, currently exposing api is used.
*) template engines is not required for the SPA,
*) template engines(we have some middleware: EJS, pug, express handlebars)

MPA:
That is within server only we can create view(template engines), model(database), controller(server)
are written.

SPA: we make an api call, to render the data on browser we use react,angular,vue js

note: to render the static pages,we can use the built in middleware.

Template Engines:
---

*) A template engine enables you to use static template files in your application. At runtime, the template
engine replaces variables in a template file with actual values, and transforms the template into
an HTML file sent to the client.

*) Some popular template engines that work with Express are Pug/Jade, Mustache, and EJS(embedded JS templates)

The top 5 JavaScript templating engines
1. Mustache.
2. Underscore Templates.
3. Embedded JS Templates.
4. HandlebarsJS.
5. Jade/Pug templating

note: better to use ejs or handlebars

Handlebars JS:

*) Its a third party middleware.
*) Its a view engine for express

search for: npm express handlebars

to install: npm i express-handlebars

---

handlebars folder structure:

---

views
-> Home.handlebars (file)
-> layouts
-> Main.handlebars

2 directories, 3 files

---

to utilise the engine:

---

step 1: const { engine } = require("express-handlebars");

step 2: app.engine("handlebars", engine());

step 3:
syntax: app.engine(extension, engine())
app.engine("handlebars", engine());

syntax: app.set()
app.set("view engine", "handlebars");

step 4: app.set("views", "/views"); //optional

step 5: Go for the root handlebar files (to call bootstrap, custom js, tailwind everything can be called here)
dynamic render all the html body elements.

Main.handlebars:
{{{body}}}

step 6: to render the contents

home.handlebars: only the content can be rendered

<h1>Im a Home Page</h1>

step 7: Render view with the given options and optional callback fn. When a callback function is given a response will not be made automatically, otherwise a response of 200 and text/html is given.

// to render the template engine
res.render('./Home.handlebars')

ex:

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

//Loads the handlebars module
const { engine } = require("express-handlebars");
const app = express();

//console.log(app); //app provides one engine

app.engine("handlebars", engine());

//Sets our app to use the handlebars engine
app.set("view engine", "handlebars");

app.set("views", "./views"); //optional

app.get("/", (req, res) => {
// to render the template engine
res.render("./Home.handlebars");
});

app.listen(4000, err => {
if (err) throw err;
```

```
console.log("server running in 4000...");
});
```

Day2 parser

Tuesday, February 1, 2022 11:13 AM

Cli

Client side rendering	Server side rendering
React	Pug
angular	Ejs
Vue js	Handle bars

Interpolation or data expression

`<h1>{{title}}</h1>`
In home.handlebars
	`app.get("/", (req, res) => {`
	`  res.render("home",{title:"student app"});`
	`});`
	In server.js

1. npm init -y
2. npm install esm express express-handlebars
3. Change in package in json

```
"start": "nodemon -r esm server.js"
```

4. Create folder structure views/ layout
5. Create home.handlebars in views
6. Create main.handlebars in layout
7. main.handlebars

```
<!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>
    {{{body}}}
</body>
</html>
```

8. Home.handlebars

```
<p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Id sit, iste placeat dolor aperiam ex, unde
vel sequi repellendus, repellat assumenda. Dolores tenetur officiis dignissimos deserunt omnis,
laudantium odio! Fuga?</p>
<h1>bist du da?</h1>
<h1>{{title}}</h1>
```

9. Server.js

```
const express = require("express");
const app = express();
const { engine } = require("express-handlebars");
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views")
app.get("/", (req, res) => {
  res.render("home",{title:"student app"});
});
app.listen(5000, err => {
    if (err) throw err;
    console.log("server is online");
})
```

Database connection
Mongodb is a schema less database
It does not have any structure
It is object based or json
ODM object data modelling---> mongoose

What is mongoose in Mongodb?

Mongoose is a node.js based Object Data Modelling (ODM) library for MongoDB. It is akin an Object Relational Mapper (ORM) such as SQLAlchemy for traditional SQL databases. The Problem that Mongoose aims to solve is allowing developers to enforce a specific schema(structure) at the application layer.

Npm install mongoose

```
//set database connection
let mongodbURl = "mongodb://localhost:27017";
mongoose.connect(mongodbURl, err => {
    if (err) throw err;
    console.log("database connected");
})
//connection ends here
```

Middleware- used to render css or static files

```
const path = require("path");

/*serve static files to or middleware block */
app.use(express.static(path.join(__dirname, "public")));
//end of middleware block
```

1. Create public folders
2. Inside public folder create folder create css js images
3. Inside css folder create css file
4. Inside server.js use app.use

```
app.use(express.static(path.join(__dirname, "public")));
```

5. Go to main.hadnlebars in views/layout
6. Attach using link tag
7. In the link tag get rid of public
8. After body call js file using script
9. Remove public again from script src

10. Inside views create folder partials, inside partials create _navbar.handlebars file

11. Inside main.handlebars

```
{{!-- injecting partial files which must exist in all pages of a MPA --}}
      {{> _navbar}}
In body
```

```
li*4>a{home}
```

Express body parser (interview question)

It is an NPM library used to process data sent through an HTTP request body. It exposes four express middlewares for parsing text, JSON, url-encoded and raw data set through an HTTP request body. These middlewares are functions that process incoming request before they reach the target controller.

//middleware body parser

```
app.use(express.urlencoded({extended:true}))//for catching post data
```

Extended false takes only strings

Extended true takes any type of data

Complete code

```
const express = require("express");
const app = express();
const { engine } = require("express-handlebars");
const mongoose = require('mongoose');
const path = require("path");
//set template engine
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views")
//set database connection
let mongodbURl = "mongodb://localhost:27017";
mongoose.connect(mongodbURl, err => {
    if (err) throw err;
    console.log("database connected");
})
//connection ends here
/*serve static files to or middleware block */
app.use(express.static(path.join(__dirname, "public")));
app.use(express.urlencoded({extended:true}))//for catching post data
//end of middleware block
/*================all post starts here=============*/
app.post("/contact", (req, res) => {
    res.send("hey everything is okay");
    console.log(req.body);
})
/*================all post ends here=============*/
//basic route
app.get("/", (req, res) => {
  res.render("home",{title:"student app"});
});
app.get("/contact", (req, res) => {
  res.render("contact", { title: "submit name" });
```

```
});
//listen port
app.listen(5000, err => {
    if (err) throw err;
    console.log("server is online");
})
```

MA"AM notes
day 2:

Student App:

steps:

1.template engine setup
2. database connect (using mongoose)

ex: let mongodbURL = "mongodb://localhost:27017";
mongoose.connect(mongodbURL, err => {
if (err) throw err;
console.log("database connected....");
});
3. add the css from server
using the built in middleware

4. add js file

5. Adding the navigation
views --> partials folder (naming convention : _filename.handlebars)

to injecting the partials file in root handlebar file : {{> _filename}}

6. Designing the app with navbar,forms and css &

7.To capture data store in db:

note:
to parse the data from the form: body parser before express 4 is TPM, after express 4 its built in middleware
app.use(express.urlencoded({extended:true}))

extended:true ===> can take any data

extended:false ===> takes only string

8. to add data in db using mongoose: We need to create a Model along with Schema

to create a Model: Model folder ->

1. import {Schema, model} from 'mongoose'

2. create a new instance:
new Schema({})

3. Add the properties: match all the values with name properties
{firstname: {}, lastname: {},phone:{}}

4. schema validation:
{firstname: {
required: true,
type:string },

lastname: {},phone:{}}

5. to specify timestamp

Adding as a new object:
new Schema ({}, { timestamps:true})

6. export schema as a model:

//model is a schema name
model('contact',ContactSchema)

export default ContactModel;

7. Import the Model to server.js

8. After Importing, Now Connecting to database

within post req, gathering the req.body as payload. its asynchronous using async await

```
const payload = req.body;
ContactModel.create(payload)
res.end('ok')
```

9. Adding the sent mail for the app using nodemailer

10. to fetch details from database: used is get()
app.get()
ex: app.get(/fetchstudents,() => {})

within get() method: to connect to database using the mongoose model
ContactModel.find({}).lean()

lean(): converts the bson into the js objects

==========(or one more way: using promises)==========

```
ContactModel.find({}).lean().then(
students => { clg(students) }
).catch(err => { clg(err)})
```

==========(or one more way: using try catch)==========

```
try{
let students = ContactModel.find({}).lean();
clg (students);
res.send(students)
}
catch (err) {
clg(err)
}
```

==========(using try catch with render)==========

```
try{
let students = ContactModel.find({}).lean();
clg (students);
res.render('all-students',{students}); //same key and value name use only one key {students:students}
}
catch (err) {
clg(err)
}
```

Views:

To display the fetched students data on the Student App, creating view.
creating all-Students.handlebars:

*) Since the students response is an array of objects, we need to iterate using forEach()

In all-Students.handlebars:

```
<h1>List of Students</h1>
{{#each students}}

<h2> {{firstname}} {{lastname}} </h2>
<p> Email: {{email}}

{{/each}}
```

note:
-> #each is way of using forEach()
-> {{#each}} == start of each loop
-> {{/each}} == end of each loop

---

body parser:

*)Express body-parser is an npm library used to process data sent through an HTTP request body.
*)It exposes four express middlewares for parsing text, JSON, url-encoded and raw data set through an HTTP request body.
*)These middlewares are functions that process incoming requests before they reach the target controller.

Initially it was third party library, now its a built in middleware

---

---

note:
1. res.render(path, {property})
ex:

```
app.get("/", (req, res) => {
// binding variables: we use {{}}
// title is dynamic templating
res.render("home",{title:'welcome to jspiders student app'});
});
```

hbs file: {{title}}

2. process.env:

Mongoose (ODM driver):
---
*) Its a library of mongoDB
*) Mongoose provides the schema for mongoDB, (basically mongoDB is schemaless)
*) It is also a middleware.

definition:
-> Mongoose is a Node. js-based Object Data Modeling (ODM) library for MongoDB.
-> It is akin to an Object Relational Mapper (ORM) such as SQLAlchemy for traditional SQL databases.
-> The problem that Mongoose aims to solve is allowing developers to enforce a specific schema at the application layer.

npm i mongoose

mongoose.connect()

Day 3 Mongoose

Wednesday, February 2, 2022 1:50 PM

Mongoose is a middleware
It provides schema in the application layer
Schema also gives validation
Model is used for specifiying model name and to hold schema

Step 1.
Contact.js

```
const { Schema, model } = require("mongoose");
const ContactSchema = new Schema(
  {
    firstname: {
      type: String,
      required: true,
    },
    lastname: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
    },
    Phno: {
      type: Number,
      required: true,
    },
    description: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);
// export default schema
export default model("contact", ContactSchema);
```

Step 2:

```
import ContactModel from "./Model/Contact";
```

Step 3:

```
/*================all post starts here=============*/
app.post("/contact", async (req, res) => {
  let payload = await req.body;
  let data = await ContactModel.create(payload);
  await res.send({ data, text: "successfully created " });
});
/*================all post ends here=============*/
```

Whole code

```
const express = require("express");
const app = express();
const { engine } = require("express-handlebars");
const mongoose = require("mongoose");
const path = require("path");
// const ContactModel = require("ContactModel");
import ContactModel from "./Model/Contact";
//set template engine
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views");
//set database connection
let mongodbURl = "mongodb://localhost:27017/students";
mongoose.connect(mongodbURl, err => {
  if (err) throw err;
  console.log("database connected");
});
//connection ends here
/*serve static files to or middleware block */
app.use(express.static(path.join(__dirname, "public")));
app.use(express.urlencoded({ extended: true })); //for catching post data
//end of middleware block
/*================all post starts here=============*/
app.post("/contact", async (req, res) => {
  let payload = await req.body;
  let data = await ContactModel.create(payload);
  await res.send({ data, text: "successfully created " });
});
/*================all post ends here=============*/
//basic route
app.get("/", (req, res) => {
  res.render("home", { title: "student app" });
});
app.get("/contact", (req, res) => {
  res.render("contact", { title: "submit name" });
});
//listen port
app.listen(5000, err => {
  if (err) throw err;
  console.log("server is online");
});
```

1. Create the Schema and Export the Model
2. After Importing the Model, Connect it to database.
3. MeanWhile Try to add a nodemailer
4. Now to fetch details of students from database
i.e use with app.get()
5. After getting data, now to connect to database using the mongoose model to find the students details
ContactModel.find({}).lean()

6. Now to display the students details in the Student App,
To display the fetched students data on the Student App, creating view.
creating all-Students.handlebars:

7. Since the students response is an array of objects, we need to iterate using forEach()
{{#each }}

{{/each}}

Entire code

```
const express = require("express");
const app = express();
const { engine } = require("express-handlebars");
const mongoose = require("mongoose");
const path = require("path");
// const ContactModel = require("ContactModel");
import ContactModel from "./Model/Contact";
const nodemailer = require("nodemailer");
//set template engine
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views");
//set database connection
let mongodbURl = "mongodb://localhost:27017/students";
mongoose.connect(mongodbURl, err => {
  if (err) throw err;
  console.log("database connected");
});
//connection ends here
/*serve static files to or middleware block */
app.use(express.static(path.join(__dirname, "public")));
app.use(express.urlencoded({ extended: true })); //for catching post data
//end of middleware block
/*================all post starts here=============*/
app.post("/contact", async (req, res) => {
  //save incoming request inot mongo database
  let payload = await req.body;
  //node mailer block
  // nodemailer
  //   .createTransport({
  //     service: "gmail",
  //     auth: {
  //       user: "hypertasker98@gmail.com",
  //       pass: ",.",
  //     },
  //   })
  //   .sendMail({
  //     from: "hypertasker98@gmail.com",
  //     to: [req.body.email, "priyanka.km@testyantra.com"],
  //     subject: "Bhuvan Contact form",
  //     html: `<h1>${req.body.firstname}${req.body.lastname}</h1>
  //   <p>Email:${req.body.email}</p>
  //   <p>Phno:${req.body.phno}</p>
  //   <p>Comments:${req.body.description}</p>
  //   `,
  //   });
  //end of node mailer
  let data = await ContactModel.create(payload);
  res.send({ data, text: "successfully created " });
});
/*================all post ends here=============*/
//basic route
//fetching from database
app.get("/all-students", async (req, res) => {
  let dev = await ContactModel.find({}).lean();
  console.log(dev);
  res.render("all-students", { dev });
```

```
});
//end of fetching from database
app.get("/", (req, res) => {
  res.render("home", { title: "student app" });
});
app.get("/contact", (req, res) => {
  res.render("contact", { title: "submit name" });
});
//listen port
app.listen(5000, err => {
  if (err) throw err;
  console.log("server is online");
});
```

For each
All-students.handlebars

```
<h1>list of students</h1>
{{#each dev}}
  <h1>{{firstname}}</h1>
  <p>{{lastname}}</p>
  <p>{{email}}</p>
  <p>{{phno}}</p>
  <p>{{description}}</p>
{{/each}}
```

Contact.js

```
const { Schema, model } = require("mongoose");
const ContactSchema = new Schema(
  {
    firstname: {
      type: String,
      required: true,
    },
    lastname: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
    },
    Phno: {
      type: Number,
      required: true,
    },
    description: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);
// export default schema
export default model("contact", ContactSchema);
```

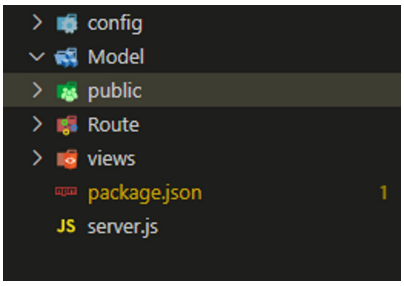
Day 4 HRM app

Thursday, February 3, 2022 11:21 AM

Step 1: yarn init -y
Step 2 go to package.json and add

```
"main": "server.js",
  "scripts":{
    "start":"nodemon server.js"
  },
```

Step 3:create folders Model, public , Route, config
Step 4: create files server.js



Config for configuration files
Model folder for schema structure
Public folder all static assets
Route folder express routing and logic or controller
Views folder all view templates

Step 5 yarn add express mongoose dotenv express-handlebars
Step 6 yarn add bootstrap@4.6 jquery popper.js

Final package.json file

```
{
  "name": "hrmapp",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "nodemon server.js"
  },
  "license": "MIT",
  "dependencies": {
    "bootstrap": "4.6",
    "dotenv": "^16.0.0",
    "express": "^4.17.2",
    "express-handlebars": "^6.0.2",
    "jquery": "^3.6.0",
    "mongoose": "^6.2.0",
    "popper.js": "^1.16.1"
  }
}
```

Dotenv

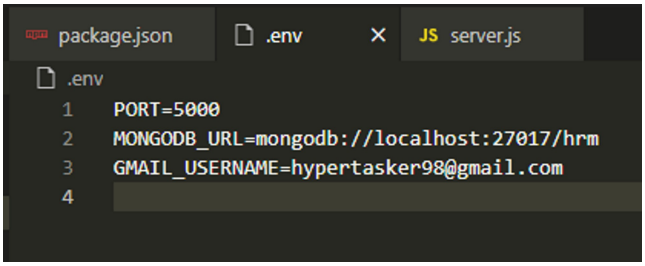
To protect data we use dot env, like passwords.
dotenv is a zero-dependency module that loads environment variables for a .env file inot process.env. Stroting configuration in the enivronment separate from code is based on the Twelve- factor app methodology.

Mandatory module in all projects

Step 1:
In root level create file .env (it is a hidden file)
Note: it does not have any extension

Do not use semicolons or string like conventions in this file
Step 2: create three variables

PORT=5000
MONGODB_URL=mongodb://localhost:27017/hrm
GMAIL_USERNAME=hypertasker98@gmail.com

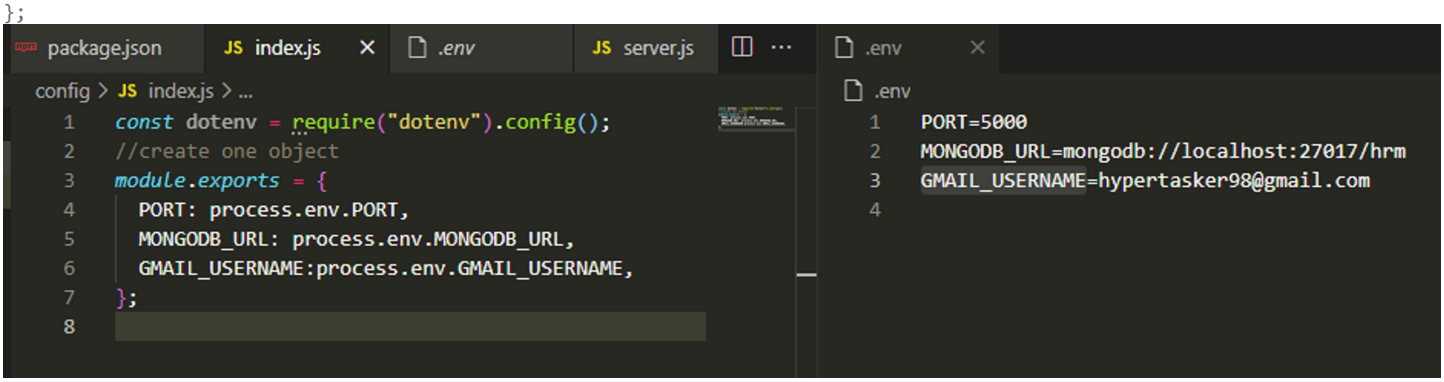


Step 3: inside config folder create a folder called index.js and type following code

```
const dotenv = require("dotenv").config();
```

Step 4:
Keys in index.js object can be anything but value must be same as .env file

```
const dotenv = require("dotenv").config();
//create one object
module.exports = {
  PORT: process.env.PORT,
  MONGODB_URL: process.env.MONGODB_URL,
  GMAIL_USERNAME:process.env.GMAIL_USERNAME,
};
```



Step 5: in server.js

```
const express = require("express");
const {PORT,MONGODB_URL } = require('./config');
```

Step 6: create top level function

```
const express = require("express");
const { PORT, MONGODB_URL } = require('./config');
//create top level function
const app = express();
//listen a port
app.listen(PORT, e => {
    if (err) throw err;
    console.log(`HRM app is running on port number ${PORT}`);
})
```

Step 7:

```
const { connect } = require('mongoose');
//!=====================database connection STARTS here===============
let DatabaseConnection = async () => {
    await connect(MONGODB_URL);
    console.log("database connected");
}
DatabaseConnection();
//!=====================database connection ENDS here===============
```

Entire code

```
const express = require("express");
const { PORT, MONGODB_URL } = require('./config');
//create top level function
const app = express();
const { connect } = require('mongoose');
//!=====================database connection STARTS here===============
let DatabaseConnection = async () => {
```

```
    await connect(MONGODB_URL);
    console.log("database connected");
}
DatabaseConnection();
//!=====================database connection ENDS here===============
//listen a port
app.listen(PORT, err => {
    if (err) throw err;
    console.log(`HRM app is running on port number ${PORT}`);
})
```

NOW HANDLE BARS
step 1: go to views folder, create partials folder and also create home.handlebars
Step 2: Then go to partials folder create _navbar.handlebars
Step 3: now inject tempplate engine middleware

```
const {engine} = require("express-handlebars");
//?=================to do template engine middleare STARTS here==================
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views");
//?=================to do template engine middleare ENDS here==================
```

Entire code with handlebars

```
const express = require("express");
const { PORT, MONGODB_URL } = require('./config');
//create top level function
const app = express();
const { connect } = require('mongoose');
const {engine } = require("express-handlebars");
//!=====================database connection STARTS here===============
let DatabaseConnection = async () => {
    await connect(MONGODB_URL);
    console.log("database connected");
}
DatabaseConnection();
//!=====================database connection ENDS here===============
//?=================to do template engine middleare STARTS here==================
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views");
//?=================to do template engine middleare ENDS here==================
//listen a port
app.listen(PORT, err => {
    if (err) throw err;
    console.log(`HRM app is running on port number ${PORT}`);
})
```

NOW ROUTER Setting

Step1: inside route folder create a file called employee.js

```
//router level middleware
const {Router} = require("express");
```

What is router level middleware Express?
Router level middleware work just like application level middleware except they are bound to an instance of express. Router().

Create top level function of router and export it

```
//router level middleware
const { Router } = require("express");
//creating router top level function
const router = Router();
//export it to use in server.js
module.exports = router;
```

Final code of employee.js

```
//router level middleware
const { Router } = require("express");
const router = Router();
/*@GET METHOD
    @ACCESS PUBLIC
 */
router.get("/home", (req, res) => {
  res.render("../views/home", { title: "Home page" });
});
/*====================END OF ALL GET METHODS==========================*/
```

```
module.exports = router;
```

Now Route setting in Server.js

Step 1:
```
const EmployeeRoute = require("./Route/employee");
```
Step2:
Always routing path should above listen section and after middleware injecting section

```
//route setting for employees
app.use("/employee", EmployeeRoute);
```

Now adding custom css, bootstrap

Step 1: go to public folder create three other folders css, images and js (create files inside hrm.css, client.js)
Step 2:
```
const { join } = require("path");
```

Step 3:for CSS, JS etc
```
app.use(express.static(join(__dirname + "public")));
```

Step 4: for BOOTSTRAP
```
app.use(express.static(join(__dirname + "node_modules")));
```

NOW link all files to main.handlebars

```
    <title>Document</title>
    <link rel="stylesheet" href="../../bootstrap/dist/css/bootstrap.css">
    <link rel="stylesheet" href="../../css/hrm.css">
  </head>
  <body>
    {{{body}}}
    <script src="../../jquery/dist/jquery.js"></script>
    <script src="../../node_modules/popper.js/dist/popper.js"></script>
    <script src="../../node_modules/bootstrap/dist/js/bootstrap.js"></script>
    <script src="../../public/js/client.js"></script>
  </body>
</html>
```

Remove all node_modules and public from folders in link tags and script tags

main.handlebars

```
<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>
    <link rel="stylesheet" href="../../bootstrap/dist/css/bootstrap.css">
    <link rel="stylesheet" href="../../css/hrm.css">
  </head>
  <body>
    {{{body}}}
    <script src="../../jquery/dist/jquery.js"></script>
    <script src="../../popper.js/dist/popper.js"></script>
    <script src="../../bootstrap/dist/js/bootstrap.js"></script>
    <script src="../../js/client.js"></script>
  </body>
</html>
```

```
div.col-3*6>h2{create employee}+form>div.form-group*6>label{emp name}+input.form-control
```

```
main.employeeBlock>h2.h2.text.uppercase{create employee}+form>div.row>div.col-3.form-group*6>label{emp name}+input.form-control
```
Now create form using bootstrap

Create-emp.handlebars

```
<section class="container bg-white my-4">
  <article>
    <main class="employeeBlock">
```

```
<h2 class="h2 text-uppercase py-3 font-weight-bold text-success">create
  employee</h2>
<form action="/employee/create-emp" method="post">
  <div class="row">
    <div class="col-3 form-group">
      <label for="emp_name">emp name</label><input
        type="text"
        name="emp_name"
        id="emp_name"
        placeholder="enter employee name"
        class="form-control"
      /></div>
    <div class="col-3 form-group">
      <label for="emp_id">emp id</label><input
        type="text"
        name="emp_id"
        id="emp_id"
        placeholder="enter employee id"
        class="form-control"
      /></div>
    <div class="col-3 form-group">
      <label for="emp_salary">emp salary</label><input
        type="text"
        name="emp_salary"
        id="emp_salary"
        placeholder="enter employee salary"
        class="form-control"
      /></div>
    <div class="col-3 form-group">
      <label for="emp_edu">emp education</label>
      <select name="emp_edu" id="emp_edu" class="form-control">
        <option value="be">BE</option>
        <option value="btech">BTECH</option>
        <option value="bca">BCA</option>
        <option value="mca">MCA</option>
        <option value="msc">MSC</option>
      </select>
    </div>
    <div class="col-3 form-group">
      <label for="emp_gender">emp gender</label>
      <input
        type="radio"
        name="emp_gender"
        id="emp_gender"
        placeholder="male"
        value="male"
        class="form-check-input"
      />male
      <input
        type="radio"
        name="emp_gender"
        id="emp_gender"
        value="female"
        placeholder="female"
        class="form-check-input1"
      />female
    </div>
    <div class="col-3 form-group">
      <label for="emp_perc">emp percentage</label>
      <input
        type="text"
        name="emp_perc"
        id="emp_perc"
        placeholder="enter employee percentage"
        class="form-control"
      /></div>
    <div class="col-3 form-group">
      <label for="emp_loc">emp location</label>
      <input
        type="text"
        name="emp_loc"
        id="emp_loc"
        placeholder="enter employee location"
        class="form-control"
      /></div>
    <div class="col-3 form-group">
      <label for="emp_exp">emp experience</label>
      <input
        type="text"
        name="emp_exp"
        id="emp_exp"
```

```
              placeholder="enter employee experience"
              class="form-control"
            /></div>
          <div class="col-6 form-group">
            <label for="emp_skills">emp skills</label>
            <div>
              <input type="checkbox" name="emp_skills" value="JS" />JS
              <input type="checkbox" name="emp_skills" value="react" />react
              <input type="checkbox" name="emp_skills" value="express" />express
            </div>
          </div>
          <div class="col-3 form-group">
            <label for="emp_phone">emp phone number</label>
            <input
              type="text"
              name="emp_phone"
              id="emp_phone"
              placeholder="enter employee phone"
              class="form-control"
            /></div>
          <div class="col-3 form-group">
            <label for="emp_email">emp email</label>
            <input
              type="text"
              name="emp_email"
              id="emp_email"
              placeholder="enter employee email"
              class="form-control"
            /></div>
          <div class="col-3 form-group">
            <label for="emp_des">emp designation</label>
            <input
              type="text"
              name="emp_des"
              id="emp_des"
              placeholder="enter employee designation"
              class="form-control"
            /></div>
        </div>
        <button type="submit" class="btn btn-success">submit</button>
      </form>
    </main>
  </article>
</section>
{{! emp gender
emp percentage
emp location
emp experience
emp skills- check box
emp phno
emp email id
emp designation}}
```

Now create get reponse for the same form

employee.js present in Route folder

```
/*@HTTP GET METHOD
    @ACCESS PUBLIC
    @URL employee/create-emp
 */
router.get("/create-emp", (req, res) => {
  res.render("../views/employees/create-emp", { title: "create employee" });
});
```

We need to collect data in form now, we need to give Schema and model

Employee.js in Model folder

```
const { Schema, model } = require("mongoose");
const EmpSchema = new Schema(
  {
    emp_name: {
      type: String,
      required: true,
    },
    emp_id: {
      type: String,
      required: true,
    },
    emp_salary: {
      type: Number,
      required: true,
```

```
    },
    emp_edu: {
      type: String,
      required: true,
    },
    emp_gender: {
      type: String,
      required: true,
      enum: ["male", "female", "others"],
    },
    emp_exp: {
      type: Number,
      required: true,
    },
    emp_des: {
      type: String,
      required: true,
    },
    emp_location: {
      type: String,
      required: true,
    },
    emp_email: {
      type: String,
      required: true,
    },
    emp_phone: {
      type: Number,
      required: true,
    },
    emp_skills: {
      type: [" "],
      required: true,
    },
  },
  { timestamps }
);
```

Now got to server.js and call the parse method

Server.js

```
const { join } = require("path");

app.use(express.urlencoded({ extended: true }));
```

Add post method to catch data from form

Employee.js in Route folder

```
router.post("/create-emp", (req, res) => {
  res.send("ok");
  console.log(req.body);
});
module.exports = router;
```

So entire server.js file

```
const express = require("express");
const { PORT, MONGODB_URL } = require("./config");
//create top level function
const app = express();
const { connect } = require("mongoose");
const { engine } = require("express-handlebars");
//importing all routing module
const EmployeeRoute = require("./Route/employee");
const { join } = require("path");
//!=====================database connection STARTS here===============
let DatabaseConnection = async () => {
  await connect(MONGODB_URL);
  console.log("database connected");
};
DatabaseConnection();
//!=====================database connection ENDS here===============
//?=================to do template engine middleare STARTS here==================
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views");
//?=================to do template engine middleare ENDS here==================
//*=================BUILT in middlewares STARTS Here====================
app.use(express.static(join(__dirname, "public")));
app.use(express.static(join(__dirname, "node_modules")));
```

```
app.use(express.urlencoded({ extended: true }));
//*=================BUILT in middlewares ENDS Here====================
//route setting for employees
app.use("/employee", EmployeeRoute);
//listen a port
app.listen(PORT, err => {
  if (err) throw err;
  console.log(`HRM app is running on port number ${PORT}`);
});
```

Day 5 upload files and pics

Friday, February 4, 2022 11:51 AM

Hrm app continued

Upload file or pics
Insert snippet into create-emp.handlerbars in views/employes folder

Create-emp.handlebars

```
<div class="col-3 form-group">
            <label for="emp_photo">emp photo</label>
            <input
              type="file"
              name="emp_photo"
              id="emp_photo"
              class="form-control"
            /></div>
```

Insert schema for the same in Employee.js from Model folder

Employee.js

```
emp_photo: {
      type: [""],
      required: true,
      default: [
        "https://cdn-icons.flaticon.com/png/512/1144/premium/1144709.png?token=exp=1643956906~hmac=
73efdb48532ec4a91d1dadf1c2682d1b",
      ],
    },
```

TO UPLOAD FILES IN FORM
Add this attribute in form tag for front end

```
enctype="multipart/form-data"
```

What is multipart form data?
Mulitpart form data is one of the values of enctype attribute, which is used in form element that have a file upload. Mulitpart means form data divides into multiple parts to send to server

Middleware for backend
Multer is a nodesjs middleware for hndling multipart/form -data, which is primarily used for uploading files, it is written on top of busboy for maximum efficiency.
Step 1:

Cmd	yarn add multer

Step 2. Now create a new folder called middleware
Step 3: inside middleware create a file called multer.js
Step 4:
Import multer
Use a method called diskstorage which takes two properties({destination},{filename})
Step 5: now for destination and filename create a function with three parameter
Req, file , cb
Step 6: cb()

Multer.js in middleware folder

```
const multer = require("multer");
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "public/uploads");
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + file.originalname);
  },
});
  1.module.exports = { storage };
```

Now in employee.js add to routing

Load middlewares

```
const multer = require("multer");

//?load multer middlewares
let { storage } = require("../middleware/multer");
const upload = multer({ storage: storage });
```

Add upload in post method in employee.js in Route folder

Post

```
router.post("/create-emp", upload.single("emp_photo"), async (req, res) => {
  let payload = {
    emp_photo: req.file,
    emp_name: req.body.emp_name,
    emp_id: req.body.emp_id,
    emp_salary: req.body.emp_salary,
    emp_edu: req.body.emp_edu,
    emp_des: req.body.emp_des,
    emp_phone: req.body.emp_phone,
    emp_loc: req.body.emp_loc,
    emp_email: req.body.emp_email,
    emp_skills: req.body.emp_skills,
    emp_gender: req.body.emp_gender,
    emp_exp: req.body.emp_exp,
  };
  res.redirect("/employee/home", 302, {});
  console.log(req.body);
  console.log(req.file);
  // await EMPLOYEE.create(payload);
  await new EMPLOYEE(payload).save();
});
module.exports = router;
```

Day 6

Saturday, February 5, 2022 10:48 AM

To use handlebar helper classes install handlebar library
STEP1:
yarn add handlebars

Why helper classes?
We cannot write js code in template handlebars

STEP2:

```
const Handlebars = require("handlebars");
```

STEP3:

```
//*HANDLEBARS HELPER CLASSES
Handlebars.registerHelper("trimString", function (passedString) {
  var theString = passedString.slice(6);
  return new Handlebars.SafeString(theString);
});
```

STEP4:
Create table in home.handlebars and for the body of the table add for each loop and

Home.handlebars in views	

```
<table class="table table-bordered table-hover text-uppercase">
            <thead class="table-dark">
                <th>photo</th>
                <th>id</th>
                <th>name</th>
                <th>salary</th>
                <th>designation</th>
                <th>exp</th>
                <th>edu</th>
                <th>email</th>
                <th>skills</th>
                <th>phone</th>
                <th>gender</th>
                <th>location</th>
            </thead>
            <tbody class="bg-white">
                <tr>
                {{#each payload}}
                <td><img
                class="img-thumbnail"
                src="{{trimString emp_photo.0.path}}"
                alt="{{emp_name}}"></td>
                <td>{{emp_id}}</td>
                <td>{{emp_name}}</td>
                <td>{{emp_salary}}</td>
                <td>{{emp_des}}</td>
                <td>{{emp_exp}}</td>
                <td>{{emp_edu}}</td>
                <td>{{emp_email}}</td>
                <td>{{emp_skills}}</td>
                <td>{{emp_phone}}</td>
                <td>{{emp_gender}}</td>
                <td>{{emp_loc}}</td>
              </tr>
            </tbody>
```

```
                {{/each}}
            </table>
```

STEP 5:
Change the get request for home page by passing payload information to handlebar page

```
router.get("/home", async (req, res) => {
  let payload = await EMPLOYEE.find({}).lean();
  res.render("../views/home", { title: "Home page", payload });
});
```

Day 7, PUT method

Monday, February 7, 2022 11:22 AM

URL Slug

http://www.findcoder.io ----> this is base URL
http://www.findcoder.io/shashi ----> /shashi is path parameter

Which is dynamically changing

Dynamic http://www.findcoder.io/:name
: maen

```
router.get("/:id", async (req, res) => {
  let payload = await EMPLOYEE.findOne({ _id: req.params.id }).lean();
  res.render("../views/employees/employeeProfile", { payload });
  console.log(payload);
});
```

Add this field in home.handlers

```
 <td>
   <a href="/employee/{{_id}}"><button class="btn btn-primary btn-sm btn-block">View Profile</button></a>
                </td>
```

`													

Update notes

To make changes and update data
We use method override which modifies form method to put and delete to at the client end using middlewares.
Step 1
npm install method-override

Step2 in server.js

```
const methodOverride = require("method=override");
```

Step3 in server.js

```
app.use(methodOverride("_method"));
```

Step4 in route folder employee.js

```
//?=============edit data of emp===============
router.get("/edit-emp/:id", async (req, res) => {
  let editPayload = await EMPLOYEE.findOne({ _id: req.params.id }).lean();
  res.render("../views/employees/editEmp", { editPayload });
});
```

PUT method (interview question)
HTTP put request method creates a new resource or replaces a representation of the target resource with the reequest payload.
The difference between PUT and POST is that PUT is idempotent calling it once or several times successively has the same effect (that is no side effect), where as successive identical.

Request has a body	Yes
Successful response has body	No
safe	No
idempotent	Yes
cacheable	No
Allowed in html forms	No

Step 5: in route folder employee.js

```
//*============put request starts here======================
router.put("/edit-emp/:id", upload.single("emp_photo"), (req, res) => {
  // res.send("ok");
  EMPLOYEE.findOne({ _id: req.params.id })
    .then(editEmp => {
      //old             new
      (editEmp.emp_photo = req.file),
        (editEmp.emp_name = req.body.emp_id),
        (editEmp.emp_salary = req.body.emp_salary),
        (editEmp.emp_edu = req.body.emp_edu),
        (editEmp.emp_exp = req.body.emp_exp),
        (editEmp.emp_email = req.body.emp_email),
        (editEmp.emp_phone = req.body.emp_phone),
        (editEmp.emp_gender = req.body.emp_gender),
        (editEmp.emp_des = req.body.emp_des),
        (editEmp.emp_skills = req.body.emp_skills),
        (editEmp.emp_loc = req.body.emp_loc);
      //update in database
      editEmp.save().then(_ => {
        res.redirect("/employee/home", 302, {});
      });
    })
    .catch(err => {
      console.log(err);
    });
});
```

Step 6: in editEmp.handlbars file

```
<form
        action="/employee/edit-emp/{{editPayload._id}}?_method=PUT"
        method="post"
        enctype="multipart/form-data"
      >
        <input type="hidden" name="_method" value="PUT" />
```

Day 8 toast msg,authentication

Tuesday, February 8, 2022 11:30 AM

To set toast messages or snack bar messages
We have a module called connect-flash

The flash is a special area of the session used for storing messages. Messages are written to the flash and cleared after being displayed to the user. The flash is typically used in combination with redirects, ensuring that the message is available to the next page that is to be rendered.

From <https://www.npmjs.com/package/connect-flash>

Step 1:

```
$ npm install connect-flash
```

From <https://www.npmjs.com/package/connect-flash>

Step2:
Connect-flash depends on session handling so we need to install another module called session

Npm install express-session

Step 3: in server.js

```
const flash = require("connect-flash");
const session = require("express-session");
```

Step4:in server.js

```
//!session middlewares
app.use(
  session({
    secret: "keyboard cat",
    resave: true,
    saveUninitialized: true,
    //cookie: { secure: true },
  })
);
//*connect flash middlewares
app.use(flash());
```

Step 5: set global variables in server.js

```
//?==============set global variables================
app.use(function (req, res, next) {
  app.locals.SUCCESS_MESSAGE = req.flash("SUCCESS_MESSAGE");
  app.locals.ERROR_MESSAGE = req.flash("ERROR_MESSAGE");
  app.locals.errors = req.flash("errors");
  next();
});
```

Step6: in partials folder create a new file _messages.handlebars
Inside _message.handlebars

```
{{#if SUCCESS_MESSAGE}}
  <div class="alert alert-success">
    {{SUCCESS_MESSAGE}}
  </div>
{{/if}}
{{#if ERROR_MESSAGE}}
  <div class="alert alert-danger">
    {{ERROR_MESSAGE}}
  </div>
{{/if}}
```

Step 7:Client.js for popup animation

```
let alert = document.querySelector(".alert");
setTimeout(function () {
  alert.style.transform = "translate(500px)";
  alert.style.transition = "ease all 0.7s";
  alert.style.position = "fixed";
}, 5000);
{
  alert.style.transform = "translate(0px)";
  alert.style.transition = "ease all 0.7s";
}
```

Step 8: Update notes update req.flash for employee.js

```
//*============delete request starts here====================
```

```
router.delete("/delete-emp/:id", async (req, res) => {
  await EMPLOYEE.deleteOne({ _id: req.params.id });
  req.flash("SUCCESS_MESSAGE", "Profile deleted Successfully");
  res.redirect("/employee/home", 302, {});
});
```

What is authentication and authorization?

Authentication is the process of verifying who someone is,
whereas authorization is the process of verifying what specific applications, files, data a user has access to.

Here's a quick overview of the differences between authentication and authorization:

Authentication	Authorization
Determines whether users are who they claim to be	Determines what users can and cannot access
Challenges the user to validate credentials (for example, through passwords, answers to security questions, or facial recognition)	Verifies whether access is allowed through policies and rules
Usually done before authorization	Usually done after successful authentication
Generally, transmits info through an ID Token	Generally, transmits info through an Access Token
Generally governed by the OpenID Connect (OIDC) protocol	Generally governed by the OAuth 2.0 framework
Example: Employees in a company are required to authenticate through the network before accessing their company email	Example: After an employee successfully authenticates, the system determines what information the employees are allowed to access

In short, access to a resource is protected by both authentication and authorization. If you can't prove your identity, you won't be allowed into a resource. And even if you can prove your identity, if you are not authorized for that resource, you will still be denied access.

https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization

Authentication has two middlewares

- Passport JS
- Token based Authentication (JWT - Json Web Token)

What is JWT ?
Is a means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is digitally signed usiing JSON Web Signatute (JWS) and/or encoded using JSON web encryption. (JWE).
Jwt.io
JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

From <https://jwt.io/introduction>

When should you use JSON Web Tokens?

Here are some scenarios where JSON Web Tokens are useful:

- **Authorization**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

- **Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

From <https://jwt.io/introduction>

Real-time examples:

when you go through security in an airport, you show your ID to authenticate your identity. Then, when you arrive at the gate, you present your boarding pass to the flight attendant, so they can authorize you to board your flight and allow access to the plane.

What is the JSON Web Token structure?

In its compact form, JSON Web Tokens consist of three parts separated by dots (`.`), which are:

- Header
- Payload
- Signature

From <https://jwt.io/introduction>

PASSPORT.JS

What is Passport.js?

Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.

From <https://www.passportjs.org/>

Search for local strategy and install it

Step1: create a file called auth.js in route folder.

Step2:

```
const { Router } = require("express");
const router = Router();
module.exports = { router };
```

Step3:

```
const AuthRoute = require("./Route/auth");
```

Step4:

```
app.use("/auth", AuthRoute);
```

Step5:
Inside views create folder auth and then create two new files login and register.handlebars

Step6: inside model create a new file called Auth.js

```
const { model, Schema } = require("mongoose");
const UserSchema = new Schema(
  {
    username: {
      type: String,
      required: true,
      // trim: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: { user, employee },
    default: "user",
  },
  { timestamps: true }
);
module.exports = model("user",UserSchema);
```

Step7: build password form in register.handlebars

```
<section id="authBlock my-4">
  <article class="col-md-3 mx-auto bg-white my-2">
    <h2 class="h2 font-weight-bold text-secondary text-uppercase py-2">
      Register
    </h2>
    <form action="/auth/register" method="post">
      <div class="form-group">
        <label for="username">username</label><input
          type="text"
          class="form-control"
          name="username"
          id="username"
          placeholder="username"
        /></div>
      <div class="form-group">
        <label for="email">email</label><input
          type="email"
          class="form-control"
          name="email"
          id="email"
          placeholder="email"
        /></div>
      <div class="form-group">
        <label for="password">password</label><input
          type="password"
          class="form-control"
          name="password"
          placeholder="password"
          id="password"
        /></div>
      <div class="form-group">
        <label for="confirmmpassword">confirm password</label><input
          type="password"
          class="form-control"
          name="password1"
          placeholder="confirm password"
          id="confirmpassword"
        /></div>
      <div class="form-group">
        <button
          class="btn btn-primary btn-block text-uppercase font-weight-bold my-2"
        >Register
        </button>
      </div>
    </form>
  </article>
</section>
```

Step8: now we need to set the route in Route folder and auth.js filename

```
const { Router } = require("express");
const router = Router();
/*
@HTTP GET REQUEST
@ACCESS PUBLIC
@URL /auth/register
*/
router.get("/register", (req, res) => {
  res.render("../views/auth/register", {});
});
router.post("/register", (req, res) => {
  let { username, email, password, password1 } = req.body;
  let errors = [];
  if (!username) {
    errors.push({ text: "username is required" });
  }
  if (!email) {
    errors.push({ text: "email is required" });
  }
  if (!password) {
    errors.push({ text: "password is required" });
  }
  if (password !== password1) {
    errors.push({ text: "password mismatch" });
  }
  if (errors.length > 0) {
    req.flash("ERROR_MESSAGE", `failed to register`);
    res.redirect("/auth/register", 302, {
      username,
```

```
      email,
      password,
      password1,
    });
  } else {
    req.flash("SUCCESS_MESSAGE", "successfully registered");
    res.redirect("/employee/home", 302, {});
  }
});
module.exports = router;
```

Day 9

Wednesday, February 9, 2022 12:02 PM

User update code

```
const USERSCHEMA = require("../Model/Auth");
```

```
const { Router } = require("express");
const router = Router();
const USERSCHEMA = require("../Model/Auth");
const bcrypt = require("bcryptjs");
/*
@HTTP GET REQUEST
@ACCESS PUBLIC
@URL /auth/register
*/
router.get("/register", (req, res) => {
  res.render("../views/auth/register", {});
});
router.post("/register", async (req, res) => {
  let { username, email, password, password1 } = req.body;
  let errors = [];
  if (!username) {
    errors.push({ text: "username is required" });
  }
  if (username.length < 6) {
    errors.push({ text: "username minimum 6 characters" });
  }
  if (!email) {
    errors.push({ text: "email is required" });
  }
  if (!password) {
    errors.push({ text: "password is required" });
  }
  if (password !== password1) {
    errors.push({ text: "password does not match" });
  }
  if (errors.length > 0) {
    // req.flash("ERROR_MESSAGE", `failed to register`);
    res.render("../views/auth/register", {
      errors,
      username,
      email,
      password,
      password1,
    });
  } else {
    let user = await USERSCHEMA.findOne({ email: email });
    if (user) {
      req.flash(
        "ERROR_MESSAGE",
        "email already exists please add new email address"
      );
      res.redirect("/auth/register", 302, {});
    } else {
      let newUser = new USERSCHEMA({
        username,
        email,
        password,
      });
           await newUser.save();
      req.flash("SUCCESS_MESSAGE", "successfully registered");
      res.redirect("/auth/login", 302, {});
    }
    // req.flash("SUCCESS_MESSAGE", "successfully registered");
    // res.redirect("/employee/home", 302, {});
  }
});
```

what is bcrypt in node js

Bcrypt is a popular and trusted method for salt and hashing passwords.

What is bcrypt used for?
The bcrypt hashing function allows us to build a password security platform that scales with computation power and always hashes every password with a salt.

Optimized bcrypt in JavaScript with zero dependencies. Compatible to the C++ bcrypt binding on node.js and also working in the browser.
step 1: npm install bcryptjs

From <https://www.npmjs.com/package/bcryptjs>

Step2:

```
const bcrypt = require("bcryptjs");
```

Step 3: in auth.js in route folder

```
bcrypt.genSalt(12, (err, salt)=>{
        if (err) throw err;
        console.log(salt);
        bcrypt.hash(newUser.password, salt, (err, hash) => {
          console.log(hash);
          if (err) throw err;
          newUser.password = hash;
        })
      })
```

auth.js route folder entire code

```
const { Router } = require("express");
const router = Router();
const USERSCHEMA = require("../Model/Auth");
const bcrypt = require("bcryptjs");
/*
@HTTP GET REQUEST
@ACCESS PUBLIC
@URL /auth/register
*/
router.get("/register", (req, res) => {
  res.render("../views/auth/register", {});
});
router.post("/register", async (req, res) => {
  let { username, email, password, password1 } = req.body;
  let errors = [];
  if (!username) {
    errors.push({ text: "username is required" });
  }
  if (username.length < 6) {
    errors.push({ text: "username minimum 6 characters" });
  }
  if (!email) {
    errors.push({ text: "email is required" });
  }
  if (!password) {
    errors.push({ text: "password is required" });
  }
  if (password !== password1) {
    errors.push({ text: "password does not match" });
  }
  if (errors.length > 0) {
    // req.flash("ERROR_MESSAGE", `failed to register`);
    res.render("../views/auth/register", {
      errors,
      username,
      email,
      password,
      password1,
    });
  } else {
    let user = await USERSCHEMA.findOne({ email: email });
    if (user) {
      req.flash(
        "ERROR_MESSAGE",
        "email already exists please add new email address"
      );
      res.redirect("/auth/register", 302, {});
    } else {
      let newUser = new USERSCHEMA({
        username,
        email,
        password,
      });
      bcrypt.genSalt(12, (err, salt) => {
        if (err) throw err;
        console.log(salt);
        bcrypt.hash(newUser.password, salt, async (err, hash) => {
          console.log(hash);
```

```
          if (err) throw err;
          newUser.password = hash;
          await newUser.save();
          req.flash("SUCCESS_MESSAGE", "successfully registered");
          res.redirect("/auth/login", 302, {});
        });
      });
    }
    // req.flash("SUCCESS_MESSAGE", "successfully registered");
    // res.redirect("/employee/home", 302, {});
  }
});
module.exports = router;
```

Login.handlebars

```
<section id="authBlock my-4">
  <article class="col-md-3 mx-auto bg-white my-2">
    <h2 class="h2 font-weight-bold text-secondary text-uppercase py-2">
      LOGIN
    </h2>
    <form action="/auth/register" method="post">
      <div class="form-group">
        <label for="email">email</label><input
          type="email"
          class="form-control"
          name="email"
          id="email"
          placeholder="email"
        /></div>
      <div class="form-group">
        <label for="password">password</label><input
          type="password"
          class="form-control"
          name="password"
          placeholder="password"
          id="password"
        /></div>
      <div class="form-group">
        <a
          href="/auth/register"
          class="badge badge-sm badge-secondary float-right my-2 p-2"
        >register</a>
        <button
          class="btn btn-primary btn-block text-uppercase font-weight-bold my-2"
        >Login
        </button>
      </div>
    </form>
  </article>
</section>
```

Authorization

Go with passport js
Step1: npm install passport-local
Step2:npm install passport
Step3:go to server.js and import passport

```
const passport = require("passport");
```

What is session in Nodejs?

Session management can be done in node. js by using the express-session module. It helps **in saving the data in the key-value form**.
Session storage is temprovary in nature.

By using express-session only we can use session.
HTTP is a stateless protocol which means at the end of every request and response cycle, the client and the server forget about each other.

From <https://www.section.io/engineering-education/session-management-in-nodejs-using-expressjs-and-express-session/>

How sessions works (imp)

When the client makes a login request to the server, the server will create a session and store it on the server-side. When the server responds to the client, it sends a cookie. This cookie will contain the session's unique id stored on the server, which will now be stored on the client. This cookie will be sent on every request to the server.
We use this session ID and look up the session saved in the database or the session store to maintain a one-to-one match between a session and a cookie. This will make HTTP protocol connections stateful.

From <https://www.section.io/engineering-education/session-management-in-nodejs-using-expressjs-and-express-session/>

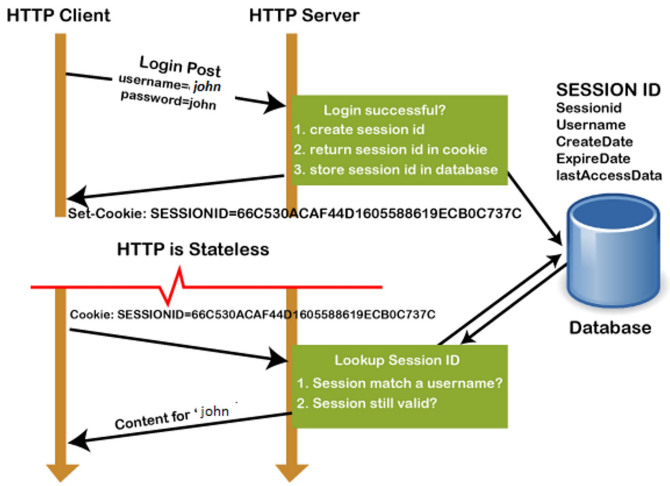
Cookies is stored in both client and server side.

What is cookie

A cookie is a key-value pair that is stored in the browser. The browser attaches cookies to every HTTP request that is sent to the server

Cookie capcity is 5kb.

In a cookie, you can't store a lot of data. A cookie cannot store any sort of user credentials or secret information. If we did that, a hacker could easily get hold of that information and steal personal data for malicious activities.



The **sessionID** is created on the server, and it saves the sessionID into the database. It returns the sessionId with a cookie as a response to the client.

From <https://www.javatpoint.com/session-vs-cookies>

Step 4:

```
//!session middlewares
app.use(
  session({
    secret: "keyboard cat",
    resave: true,
    saveUninitialized: true,
    // cookie: { secure: true },
  })
);//already written in application level middleware
app.use(passport.initialize);
app.use(passport.session());
```

```
Step 5: go to auth.js in route folder

const passport = require("passport");
Step6:
```

In auth.js under route folder

```
router.post("/login", (req, res, next) => {
  passport.authenticate("local", {
    successRedirect: "/employee/home",
    failureRedirect: "/auth/login",
    failureFlash: true,
  })(req, res, next);
});
```

Step7: under middlewares folder create a file called passport.js

Step8: inside passport.js

```
const strategy = require("passport-local").Strategy;
const bcrypt = require("bcryptjs");
const USERSCHEMA = require("../Model/Auth");
```

Step 9: same file

```
const Localstrategy = require("passport-local").Strategy;
const bcrypt = require("bcryptjs");
const USERSCHEMA = require("../Model/Auth");
module.exports = (passport) => {
    passport.use(new Localstrategy({ usernameField: "email" }, async (email, password, done) => {
        let user = await USERSCHEMA.findOne({ email: email });
        if (!user) {
            done(null, false,"User does not exists");
        } else {
            done(null, user, "successfully logged in");
        }
    }));
}
```

Step 10: same file

```
const Localstrategy = require("passport-local").Strategy;
const bcrypt = require("bcryptjs");
const USERSCHEMA = require("../Model/Auth");
module.exports = passport => {
  passport.use(
    new Localstrategy(
      { usernameField: "email" }, //optional field
      async (email, password, done) => {
        let user = await USERSCHEMA.findOne({ email: email });
        //checking username
        if (!user) {
          done(null, false, "User does not exists");
          //1st argument for error handling
          //2nd argument user or false
          //3rd argument is a message
        }
        //match password
        bcrypt.compare(password, user.password, (err, isMatch) => {
          if (err) throw err;
          if (!isMatch) {
            done(null, false, { message: "Password is not match" });
          }
        });
      }
    )
  );
};
```

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

Final working version of passport.js

```
const Localstrategy = require("passport-local").Strategy;
const bcrypt = require("bcryptjs");
const USERSCHEMA = require("../Model/Auth");
module.exports = passport => {
  passport.use(
    new Localstrategy(
      { usernameField: "email" },
      async (email, password, done) => {
        let user = await USERSCHEMA.findOne({ email: email });
        //checking username
        if (!user) {
          return done(null, false, "User does not exists");
          //1st argument for error handling
          //2nd argument user or false
          //3rd argument is a message
        }
        //match password
        bcrypt.compare(password, user.password, (err, isMatch) => {
          if (err) throw err;
          if (!isMatch) {
            return done(null, false, { message: "Password is not match" });
          } else {
            return done(null, user);
          }
        });
      }
    )
  );
  //used to serialize the user for the session
  passport.serializeUser(function (user, done) {
    done(null, user.id);
  });
  passport.deserializeUser(function (id, done) {
    USERSCHEMA.findById(id, function (err, user) {
      done(err, user);
    });
  });
};
```

Final version of server.js

```
const express = require("express");
const { PORT, MONGODB_URL } = require("./config");
//create top level function
const app = express();
const { connect } = require("mongoose");
const { engine } = require("express-handlebars");
//importing all routing module
const EmployeeRoute = require("./Route/employee");
const { join } = require("path");
```

```
const Handlebars = require("handlebars");
const methodOverride = require("method-override");
const flash = require("connect-flash");
const session = require("express-session");
const AuthRoute = require("./Route/auth");
const passport = require("passport");
require("./middleware/passport")(passport);
//!======================database connection STARTS here===============
let DatabaseConnection = async () => {
  await connect(MONGODB_URL);
  console.log("database connected");
};
DatabaseConnection();
//!======================database connection ENDS here===============
//?=================to do template engine middleare STARTS here==================
app.engine("handlebars", engine());
app.set("view engine", "handlebars");
app.set("views", "./views");
//?=================to do template engine middleare ENDS here==================
//*================BUILT in middlewares STARTS Here====================
app.use(express.static(join(__dirname, "public")));
app.use(express.static(join(__dirname, "node_modules")));
app.use(express.urlencoded({ extended: true }));
app.use(methodOverride("_method"));
//!session middlewares
app.use(
  session({
    secret: "secret",
    resave: true,
    saveUninitialized: true,
    // cookie: { secure: true },
  })
);
app.use(passport.initialize());
app.use(passport.session());
//*connect flash middlewares
app.use(flash());
//*================BUILT in middlewares ENDS Here====================
//*HANDLEBARS HELPER CLASSES
Handlebars.registerHelper("trimString", function (passedString) {
  var theString = passedString.slice(6);
  return new Handlebars.SafeString(theString);
});
//?==============set global variables================
app.use(function (req, res, next) {
  app.locals.SUCCESS_MESSAGE = req.flash("SUCCESS_MESSAGE");
  app.locals.ERROR_MESSAGE = req.flash("ERROR_MESSAGE");
  app.locals.errors = req.flash("errors");
  app.locals.error = req.flash("error");
  next();
});
//!route setting for employees---application level middleware
app.use("/employee", EmployeeRoute);
app.use("/auth", AuthRoute);
//listen a port
app.listen(PORT, err => {
  if (err) throw err;
  console.log(`HRM app is running on port number ${PORT}`);
});
```

Day 10

Thursday, February 10, 2022 11:34 AM

auth.js in route folder setting route for logout

```
router.get("/logout", async (req, res) => {
  req.logOut();
  req.flash("SUCCESS_MESSAGE", "Successfully logged out");
  res.redirect("/auth/login", 302, {});
});
```

Complete authentication is completed

On root folder create a new folder called helper and a file inside auth_helper.js

Inside auth_helper.js

```
module.exports = {
  ensureAuthenticated: function (req, res, next) {
    if (req.isAuthenticated()) {
      next();
    }
    req.flash("ERROR_MESSAGE", "you are not authenticated user");
    res.redirect("/auth/login", 302, {});
  },
};
```

In employee.js in route folder

```
//custom middleware
const { ensureAuthenticated } = require("../helper/auth_helper");
```

And enusreAuthenticates for all private methods as a second parameter for router.get/ post

NOW ADDING IMPLEMENTATION of loggined in personal name next to logout button

```
//?==============set global variables================
app.use(function (req, res, next) {
  app.locals.SUCCESS_MESSAGE = req.flash("SUCCESS_MESSAGE");
  app.locals.ERROR_MESSAGE = req.flash("ERROR_MESSAGE");
  app.locals.errors = req.flash("errors");
  app.locals.error = req.flash("error");
  app.locals.user = req.user || null;
  let userData = req.user || null;
  app.locals.finalData = Object.create(userData);
  app.locals.username = res.locals.finalData.username;
  next();
});
```

Go to employee schema and add a field

```
User:{
}
```

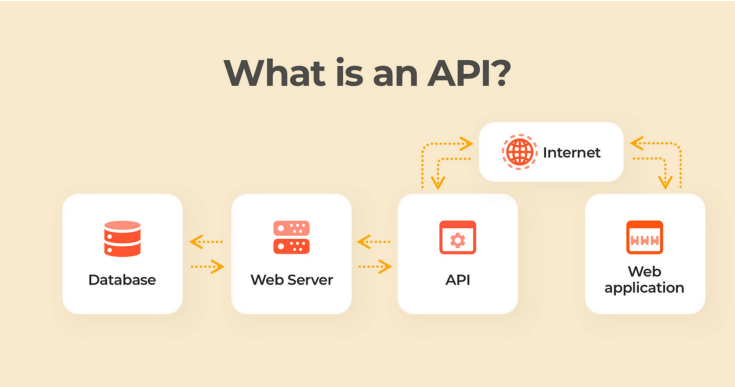
Day 10 API

Thursday, February 10, 2022 2:39 PM

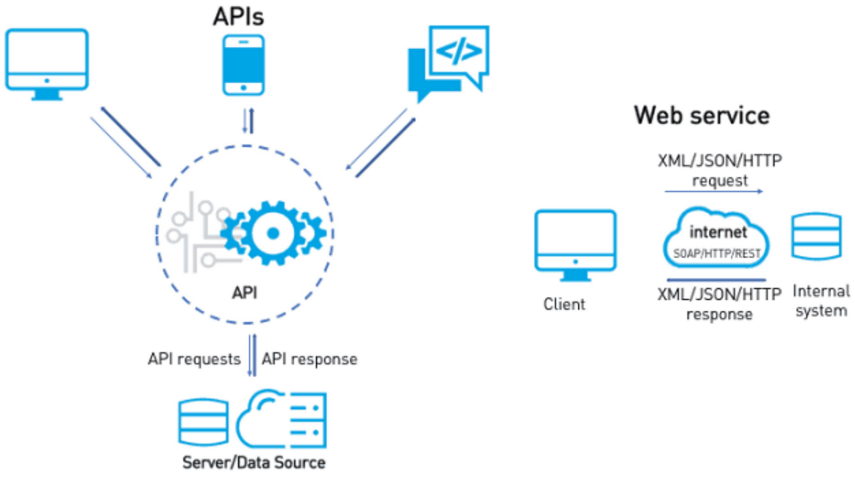
API exchange data b/w servers or client and server

ALL web services are API but all API's are not websevices

An Application Programmign Interface (API) is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software.



Difference between webservices and API
API works without internet
Web services is an Api on internet



HOW TO do CRUD operations using API
Step1: create package.json and configure it with start
Step2: yarn add express mongoose cors dotenv
In dotenv file do not give white spaces
Step3:
Configure .env files and configure it in index.js

```
require("dotenv").config();
module.exports = {
  MONGODB_URL: process.env.MONGODB_URL,
  PORT: process.env.PORT,
};
```

Step4: env-> index->DB-> Server
Inside configure folder create a new file db.js

```
const { connect } = require("mongoose");
const { MONGODB_URL } = require("./index");
exports.dbConnection = () => {
  connect(MONGODB_URL, err => {
    if (err) throw err;
    console.log("mongodb connected");
  });
```

```
};
```

Step5: in server.js

```
const express = require("express");
const { dbConnection } = require("./config/DB.JS");
const app = express();
/*=========database starts connection===============*/
dbConnection();
/*=========database ends here================*/
/*!=========middleware section starts here==============*/
app.use(express.json());
/*!=========middleware section ends here==============*/
```

Step 6: create a new folder routes and controller
In both the folders create a new folder called post.js

Step 7:in routes folder=> post.js

```
const { Router } = require("express");
const router = Router();

module.exports = router;
```

Step8: in server.js

```
/*==========load routes starts here=====*/
app.use("/api/posts/", PostRoute);
/*===========load routes ends here=====*/
```

Step9: in controller=> post.js

```
/*HTTP GET REQUEST
#ACCESS public
@URL /api/post/allposts
*/
exports.getAllPosts = (req, res) => {
  let data = [
    {
      id: 1,
      title: "REACTjs",
      body: "frontend",
    },
    {
      id: 2,
      title: "EXPRESSjs",
      body: "backend",
    },
  ];
  res.status(200).json(data);
};
```

Step 10: in routes folder post.js
Use the exportedd data from db.js and route it here to display on UI

```
const { Router } = require("express");
const router = Router();
const { getAllPosts } = require("../controllers/post");
router.route("/").get(getAllPosts);
module.exports = router;
```

Step11: set a port to listen in server.js and also add custom middleware for api creation
Final code of server.js

```
/*==========load routes starts here=====*/
app.use("/api/posts/", PostRoute);
/*===========load routes ends here=====*/
//listen port
app.listen(PORT, err => {
  if (err) throw err;
  console.log("server is running");
});
```

final code server.js	Final code routes folder post.js

	`const { Router } = require("express");`
	`const router = Router();`
	`const { getAllPosts } = require("../controllers/post");`
	`router.route("/").get(getAllPosts);`
	`module.exports = router;`

Final code config folder db.jd

```
const { connect } = require("mongoose");
const { MONGODB_URL } = require("./index");
exports.dbConnection = () => {
  connect(MONGODB_URL, err => {
    if (err) throw err;
    console.log("mongodb connected");
  });
};
```

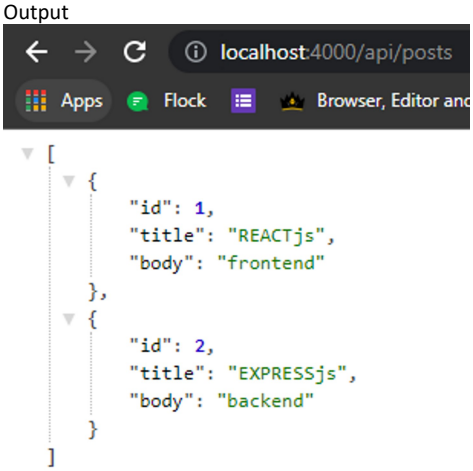
Final code config folder index.js

```
require("dotenv").config();
module.exports = {
  MONGODB_URL: process.env.MONGODB_URL,
  PORT: process.env.PORT,
};
```

Final code controller post.js

```
/*HTTP GET REQUEST
#ACCESS public
@URL /api/post/allposts
*/
exports.getAllPosts = (req, res) => {
  let data = [
    {
      id: 1,
      title: "REACTjs",
      body: "frontend",
    },
    {
      id: 2,
      title: "EXPRESSjs",
      body: "backend",
    },
  ];
  res.status(200).json(data);
};
```

Output



Day 11

Friday, February 11, 2022 9:54 AM

Postman
POStman is an API platofrom for building and using API's.
REST- representation Stateful transform
Inpostman app

1. Create a new workspace
2. Click on eye symbol create environmental name
3. Name it development_ENV
4. Give variable name BASE_URL
5. Make type default
6. Now set current and initial value as http://localhost:4000
7. Now create a new request and give url {{BASE_URL}}/api/students
8. Now save it

Ma'am notes

POSTMAN WorkSpace Creation Steps:
_____

*) create an account, create workspace (name,description,personal)
*) set the variables, there are 3 types of variables Scope: global variables
Environmental variables
Collection level variables (Development & Production)

*) set collection level variables: eye icon -> add environment(dev_ENV) -> add variables (BASE_URL) ->
initial value & current value (http://localhost:5000) -> save

*) Now switch to dev_ENV from dropdown

*) Now left we have create collection (STUDENTS)
note: all the routes of students (CRUD) can be written within the STUDENTS collections

*) Now inside students add the request, we used as GET

*) Now add the url in the below form,

{{BASE_URL}}/api/students

*) And now make a Request

Morgan npm
A tool for loggers only in development.
Morgan is a node js and express middleware to log HTTP request and errors.
Yarn add morgan

Go to .env file and add
NODE_ENV=development

Go to config index.js

```
require("dotenv").config();
module.exports = {
  PORT: process.env.PORT,
    MONGODB_URL: process.env.MONGODB_URL,
    NODE_ENV: proces.env.NODE_ENV,
};
```

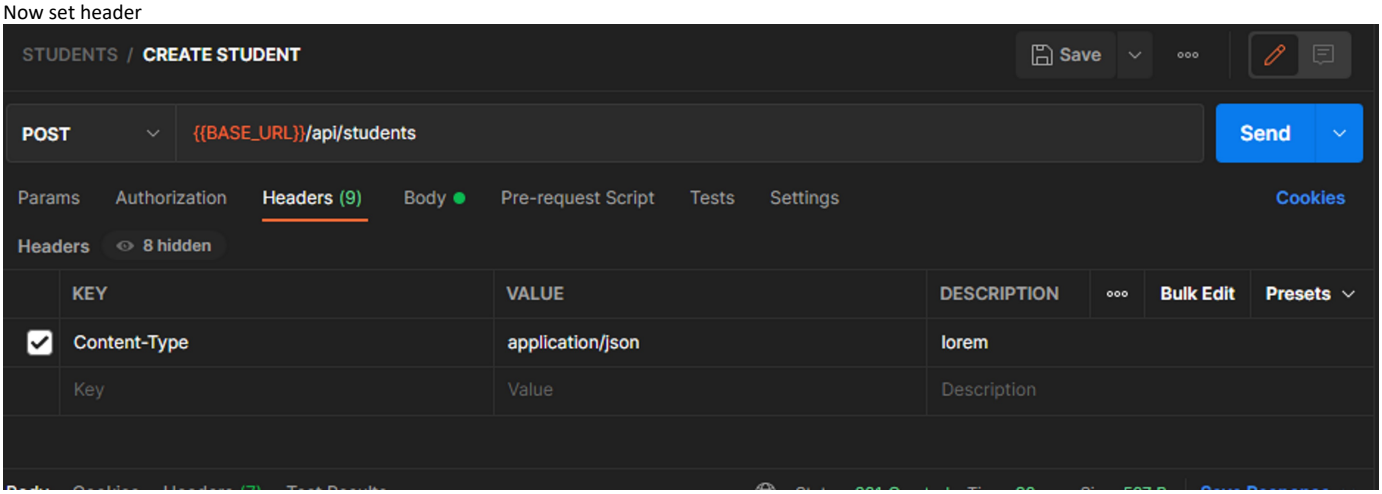
Go to server.js and load morgan

```
const morgan = require("morgan");
const { PORT, MONGODB_URL, NODE_ENV } = require("./config");
 /*-----------------Middleware section start here --------------*/
  if (NODE_ENV === "development") {
    app.use(morgan("dev"));
  }
  app.use(express.json());
  /*-----------------Middleware section end here --------------*/
```

Fill up code from vscode

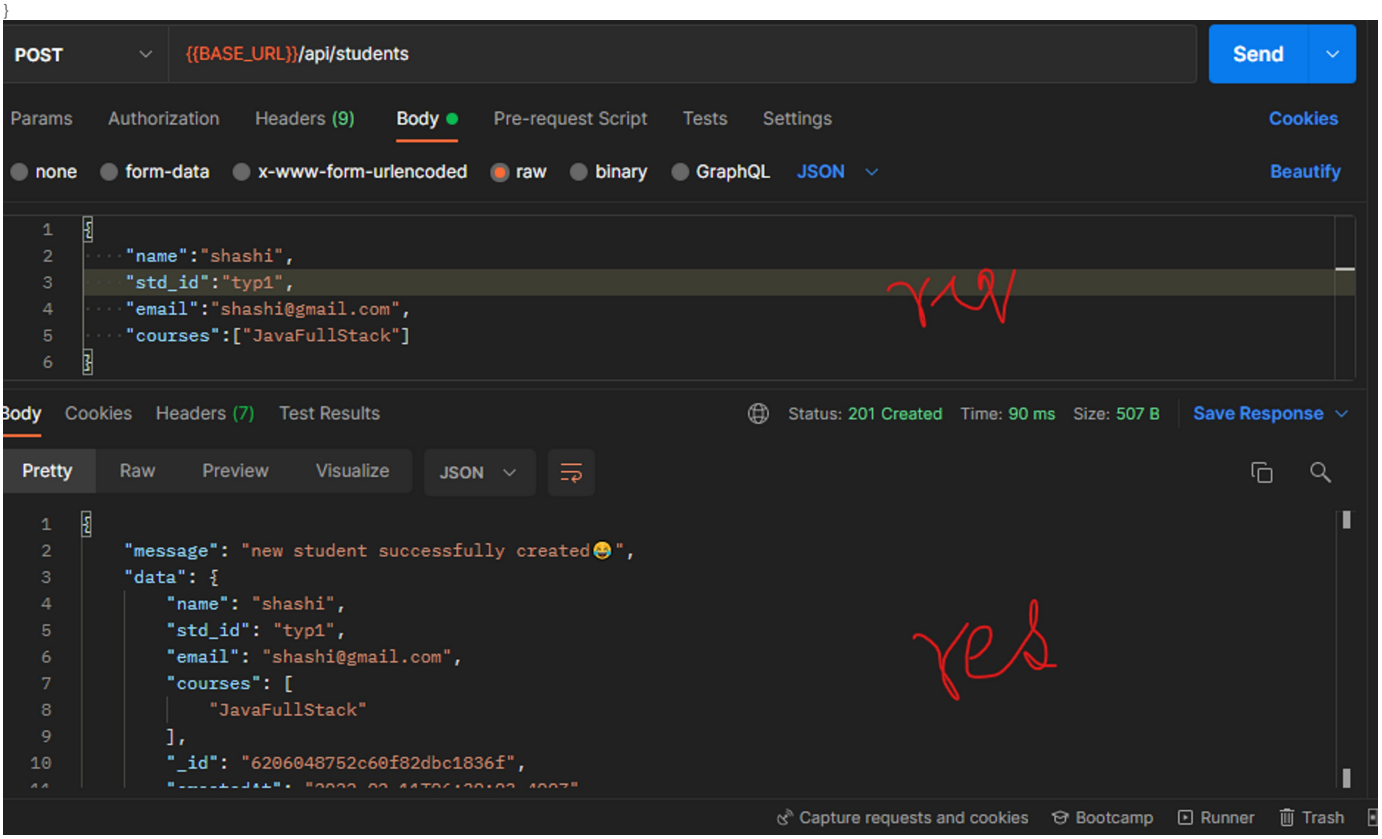
Go to postman

Create a new request by hovering on workspace
Set the post request and give url
Now set header



Now give object in post request by selecting body

```
{
  "name":"shashi",
  "std_id":"typ1",
  "email":"shashi@gmail.com",
  "courses":["JavaFullStack"]
}
```



Retry
In controller folder Student.js

```
const StudentSchema = require("../models/Student");
exports.getAllStudents = () => {};
exports.getStudent = () => {};
exports.createStudent = () => {};
exports.deleteStudent = () => {};
```

Step 2

```
const StudentSchema = require("../models/Student");
exports.getAllStudents = () => {};
exports.getStudent = () => {};
exports.createStudent = async (req, res) => {
  try {
```

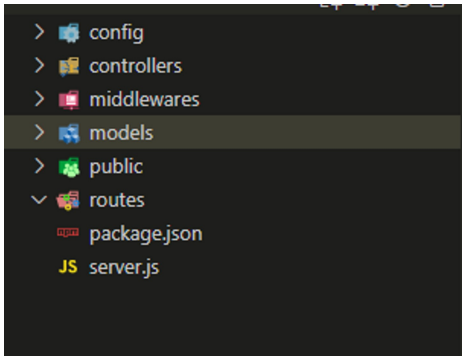
```
    let { name, std_id, email, courses } = req.body;
    let payload = {
      name,
      std_id,
      email,
      courses,
    };
    //save payload into database
    let data = await StudentSchema.create(payload);
    res.status(201).json({ message: "successfully student created", data });
  } catch (err) {
    console.log(err);
    res.status(501).json({ message: "SERVER ERROR" });
  }
};
exports.updateStudent = () => {};
exports.deleteStudent = () => {};
```

What is cors in webapi?

CROSS ORIGIN RESOURCE SHARING is a W3C standard that allows a server to relax the same-origin policy. USIng cors, a server can explicitly allow some cross-origin requests while rejecting other.

New project

Monday, February 14, 2022 12:54 PM



Folder structure as above

yarn add express dotenv mongoose cors morgan jsonwebtoken bycryptjs multer consola colors
Everything to install

Go to mongodb online
Create accoutn
Give database access and network access

Create a new project give password and username
And give it in clouldurl
In .env file

PORT=5000
LOCAL_MONGODB_URL=mongodb://localhost:27017/mern2expressapi
CLOUD_MONGODB_URL=mongodb+srv://findcoder:@cluster0.tf3oi.mongodb.net/myFirstDatabase?retryWrites=true&w=majority;
NODE_ENV=development

Config folder index.js

```
require("dotenv").config();
module.exports = {
  PORT: process.env.PORT,
  LOCAL_MONGODB_URL: process.env.LOCAL_MONGODB_URL,
  CLOUD_MONGODB_URL: process.env.CLOUD_MONGODB_URL,
  NODE_ENV: process.env.NODE_ENV,
};
```

Db.js in config folder

```
const { connect } = require("mongoose");
const { NODE_ENV, LOCAL_MONGODB_URL, CLOUD_MONGODB_URL } = require("./index");
const { success, error } = require("consola");
exports.DBConnection = async () => {
  try {
    if (NODE_ENV === "development") {
      await connect(LOCAL_MONGODB_URL);
      success("LOCAL instance database connected");
    } else {
      await connect(CLOUD_MONGODB_URL);
      success("CLOUD  instance database connected");
    }
  } catch (err) {
    error(err);
  }
};
```

Server.js

```
const express = require("express");
const { success, error } = require("consola");
const colors = require("colors");
const morgan = require("morgan");
const { NODE_ENV, PORT } = require("./config");
const { DBConnection } = require("./config/db");
const app = express();
let StartServer = async () => {
  try {
```

```
    //?==================DATABASE CONNECTION STARTS HERE===========
    DBConnection();
    //?==================DATABASE CONNECTION ENDS HERE===========
    //*================MIDDLEWARE SECTION STARTS HERE=========
    app.use(express.json());
    if (NODE_ENV === "development") {
      app.use(morgan("dev"));
    }
    //*================MIDDLEWARE SECTION ends HERE=========
    //!==============listen port start==================
    app.listen(PORT, err => {
      if (err) {
        error(err);
      } else {
        success(`Server is listening on port number 5000`.yellow.bold);
      }
    });
    //!==============listen port ends==================
  } catch (err) {
    error(`${err}`.red.bold);
  }
};
StartServer();
```

Model => Controller => router =>Root file(server.js)=> JSON (api) =>Frontend

Model	Controller	Router
Auth.js	auth.js	auth.js

```
      select: false,
```

Does not allow to fetch data from database if any schema field is declared select:false

Auth.js models folder

```
const { Schema, model } = require("mongoose");
const AuthSchema = new Schema(
  {
    username: {
      type: String,
      required: [true, "Please add username"],
      minlength: [6, "username should be minimum 6 characters"],
    },
    email: {
      type: String,
      unique: true,
      required: [true, "Please add email address"],
      match:
        /^(?:[\w\!\#\$\%\&\'\\+\-\/\=\?\^\`\{\|\}\~]+\.)[\w\!\#\$\%\&\'\*\+\-\/\=\?\^\`\{\|\}\~]+@(?:(?:(?:[a-zA-Z0-9]
(?:[a-zA-Z0-9\-](?!\.)){0,61}[a-zA-Z0-9]?\.)+[a-zA-Z0-9](?:[a-zA-Z0-9\-](?!$)){0,61}[a-zA-Z0-9]?)|(?:\[(?:(?:[01]?\d{1,2}|
2[0-4]\d|25[0-5])\.){3}(?:[01]?\d{1,2}|2[0-4]\d|25[0-5])\]))$/,
    },
    password: {
      type: String,
      required: [true, "Please add password"],
      minlength: 6,
      select: false,
    },
    role: {
      type: String,
      enum: ["user", "publisher"],
      default: user,
    },
  },
  { timestamps: true }
);
module.exports = model("user", AuthSchema);
```

Controller folder auth.js

```
const AuthSchema = require("../models/Auth");
/*
@Access public
@http request post
@url api/auth/signup
*/
exports.Signup = async (req, res) => {
    res.send("ok");
};
```

Routes folder auth.js

```
const { Router } = require("express");
const { Signup } = require("../controllers/auth");
const router = Router();
router.route("/signup").post(Signup);
module.exports = router;
```

Day 13

Tuesday, February 15, 2022 1:40 PM

JWT.io
Token based authentication

In Auth.js

```
const jwt = require("jsonwebtoken");
//way to create schema custom method
AuthSchema.methods.getJWTtoken = function () {
  return jwt.sign({ id: this._id }, JWT_SECRET, { expiresIn: JWT_EXPIRE });
};
```

Sign method has three parameters

1. Payload
2. Secret
3. Optoinal --expiry date

auth.js controller folder

```
    //save into database
    let data = await AuthSchema.create(payload);
    let TOKEN = data.getJWTtoken();
    console.log(TOKEN);
    //after JWT payload is encoded in the form of TOKEN
    res.status(201).json({ message: "Successfully user registered", TOKEN });
```

How to use JSON Web Token (JWT) for authentication in Node.js?
JSON Web Token (JWT) is an open standard that defines a compact and self-contained way of securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.
There are some advantages of using JWT for authorization:Purely stateless. No additional server or infra required to store session information.It can be easily shared among services.
JSON Web Tokens consist of three parts separated by dots (.), which are:
jwt.sign(payload, secretOrPrivateKey, [options, callback])
Header - Consists of two parts: the type of token (i.e., JWT) and the signing algorithm (i.e., HS512)
Payload - Contains the claims that provide information about a user who has been authenticated along with other information such as token expiration time.
Signature - Final part of a token that wraps in the encoded header and payload, along with the algorithm and a secret
Installationnpm install jsonwebtoken bcryptjs --save

Example:
 AuthController.js:

```
var express = require('express');
var router = express.Router();
var bodyParser = require('body-parser');
var User = require('../user/User');
var jwt = require('jsonwebtoken');
var bcrypt = require('bcryptjs');
var config = require('../config');

router.use(bodyParser.urlencoded({ extended: false }));
router.use(bodyParser.json());
router.post('/register', function(req, res) {
var hashedPassword = bcrypt.hashSync(req.body.password, 8);
User.create({
name : req.body.name,
email : req.body.email,
password : hashedPassword
},
function (err, user) {
if (err) return res.status(500).send("There was a problem registering the user.")
// create a token
var token = jwt.sign({ id: user._id }, config.secret, {
expiresIn: 86400 // expires in 24 hours
});
res.status(200).send({ auth: true, token: token });
});
});
config.js//
 config.js:
module.exports = {
'secret': 'supersecret'
};
```

The jwt.sign() method takes a payload and the secret key defined in config.js as parameters. It creates a unique string of characters representing the payload. In our case, the payload is an object containing only the id of the user.

Day14

Wednesday, February 16, 2022 9:43 AM

Install yarn add cookie-parser

Go to server.js

```
const cookie = require("cookie-parser");
```

In auth controller folder

```
function sendTokenResponse(user, statusCode, res) {
  let token = user.getJWTtoken();
  const options = {
    expires: new Date(Date.now() + JWT_COOKIE_EXPIRE * 24 * 60 * 60 * 1000),
    httpOnly: true,
  };
  res
    .status(statusCode)
    .cookie("token", token, options)
    .json({ message: "sucessfully stored", token });
}
```

What is Bearer token?

The name “Bearer authentication” can be understood as “give access to the bearer of this token.” The bearer token is a cryptic string, usually generated by the server in response to a login request.

What is difference between Bearer Token and JWT?

JWTs are **a convenient way to encode and verify claims**. A Bearer token is just string, potentially arbitrary, that is used for authorization

JWT is for authentication
Barrier token is for authorization

What is JWT Bearer Token?

JSON Web Token (JWT, RFC 7519) is **a way to encode claims in a JSON document that is then signed**. JWTs can be used as OAuth 2.0 Bearer Tokens to encode all relevant parts of an access token into the access token itself instead of having to store them in a database.

(Important)

==========HOW TO TEST USING POSTMAN for TOKENS==========

flow one: (generating the tokens by ourself)

1. signUp ---> (token generated)

2. signIn ---> (token generated)

both tokens are different, we need only signIn token

3. copy the token from signin

4. go to '/me' path

5. through Authorization

type: bearer-token
token: paste the tokenn here.....

6. save and request for me object

---

flow two: (generating the random tokens)

1. firts signUp and add the test value,
tests: pm.environment.set("TOKEN",pm.response.json().token)
---> (token generated)

2. signIn, and add the test value,
tests: pm.environment.set("TOKEN",pm.response.json().token)

```
pm.environment.set("TOKEN", pm.response.json().token)
```

---> (token generated)

both tokens are different, we need only signIn token

4. go to '/me' path

5. through Authorization

type: bearer-token
token: {{TOKEN}}

6. Now check in the eye symbol, see whether the TOKEN is generated

7. save and request for me object

process.env & questions

Friday, February 4, 2022 11:35 AM

1. Environmental variables are the variables that are set by the operating system
2. They are decoupled from application logic
3. they can be accessed from applications and programs through various API's
4. There is a nodejs library called env, that helps you to manage and load environmental variables

process.env
The process.env global variable is injected by the node at run time for your application to use and it represents the state of the system environment for your application, is when it starts.

Dev dependencies
These are the dependencies which is used by the developer.
It is like tools dependencies which is used in development not in production.

command	npm install jshint --save -dev

```
"dependencies": {
    "jshint": "^2.13.4"
  }
```

Cmd	npm un jshint

MVC is bidirectional
Redux is unidirectional

INTERVIEW QUESTIONS

1. Simple CRUD operations/
2. How to upload a file?
3. How to connect to a DB(mysql and mongodb)?
4. How to integrate from backend to frontend
5. How to add an email invite to a user
6. Write your own NodeJS module and publish it in npm
7. For username and password use token and session based authentication
8. How can I authenticate the users
9. How to implement JWT authentication in express app
10. How to allow cors in expressjs
11. How to configure properties in express (.env)
12. How to download a file
13. How to upload and download a file together in one snippet.
14. How to connect to mysql database
15. How to make a request for external API from express
16. Create your own API and fetch the details with the help of any template engine
17. Write the simple schema structure or basic level or sceanrio based
18. Create a flag for an employee has not joined and on a email invite joining for a employee change the flag value to joined.
19. How to separate express "app" and server.
20. Write a program to use async await or promises for async error handling
21. How to zip and unzip the file using node and express?