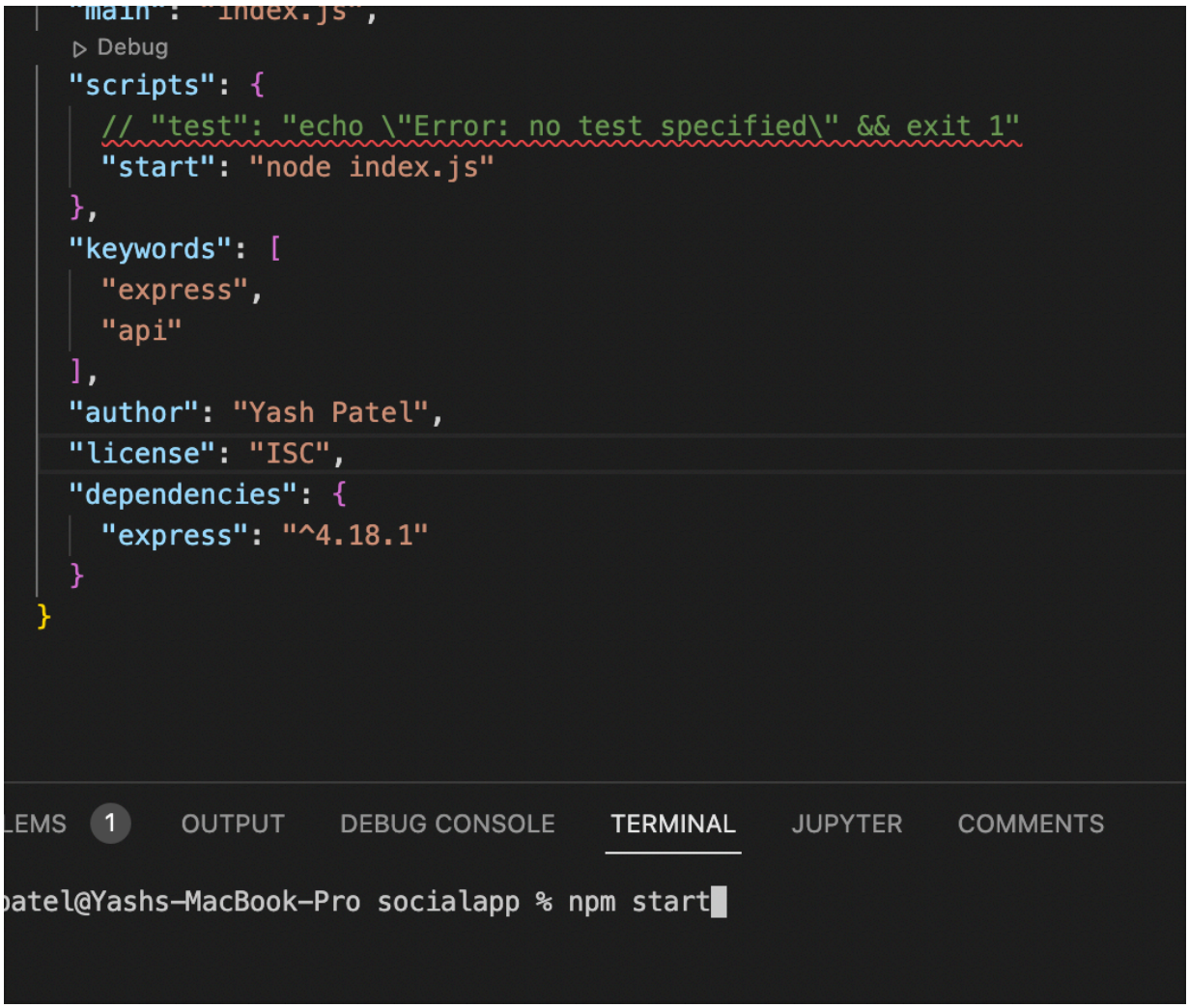
Pro Backend Developer

Section 2 - Take it up to Heroku

Lecture 12 - Your First Express App

First thing to do when use start or receive a project

Go to package.json and look for the scripts



When we write nom start in command line this will look for scripts in package.json and will run node index.js

Now the thing is we need to run node index.js again again to refresh so we install nodemon using **npm i nodemon -D**

Also there are two things

1. Dependencies
2. Dev dependencies

Dev dependencies do not go when the product is being deployed for production.

Change node index.js to nodemon index.js

```
JS index.js > ...
1   const express = require('express');
2   const app = express();
3
4   const PORT = 4000 || process.env.PORT
```

https://expressjs.com/en/starter/hello-world.html

We define here PORT with 4000 or for production purpose process.env.PORT <- this we will learn later

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

const PORT = 4000 || process.env.PORT;

app.get("/", (req, res) => {
  res.send("<h1>Hello from Yash</h1>");
});

app.listen(PORT, () => {
  console.log(`Server is running at ${PORT}`);
});
```

Lecture 13 - Request Response and Status Code

Now we can set many type of request with app

Major 4 Types of request (req) :

1. GET
2. POST
3. PUT
4. DELETE

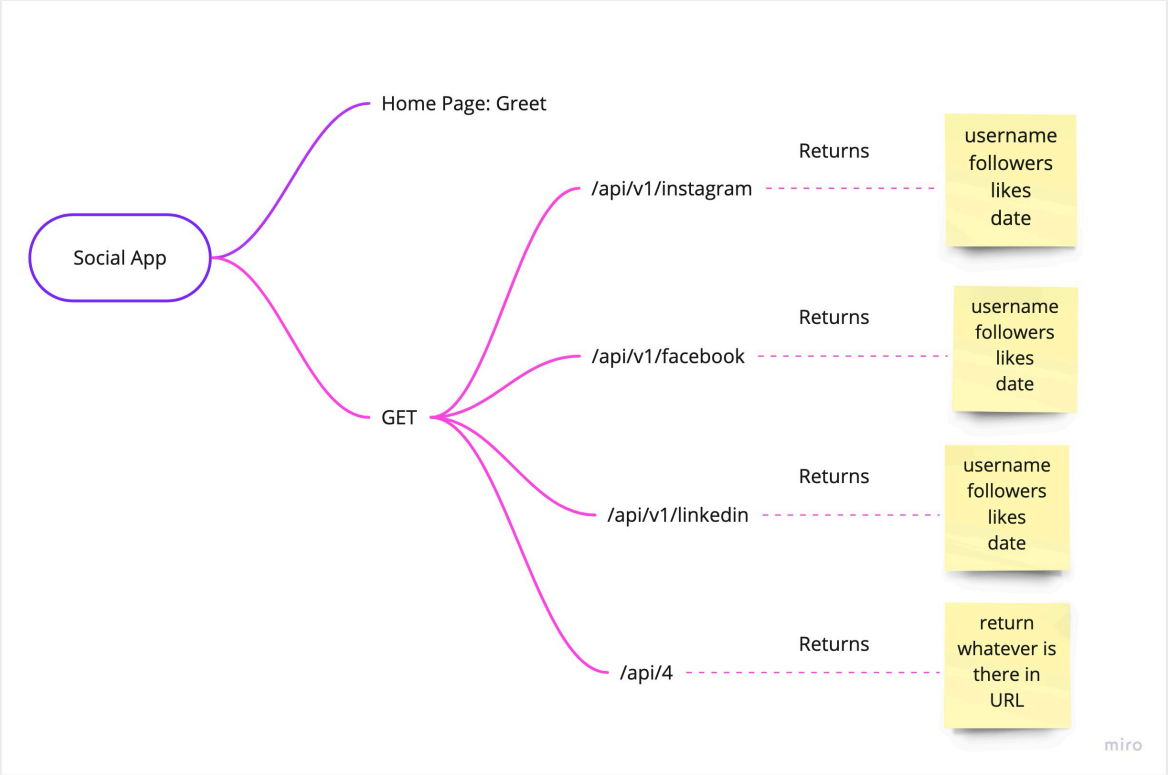
Once website loads it requests and maybe search for something after which it responds or status (also known as status code)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

200 and 201 - Ok
400 - something wrong from client side
404 - Not found
500 - something wrong from server side

Lecture 14 - All social routes

Miro Mind Map:



Edit - /api/v1/instagram

This v1 is more professional showing this is version 1 using express 4.x

```
app.get("/api/v1/instagram", (req, res) => {
  const instaSocial = {
    username: "Yash_patel4900",
    followers: 66,
    follows: 70,
    date: format.asString("dd/MM/yyyy - hh:mm:ss", new Date()),
  };

  res.status(200).json(instaSocial);
});
```

```
app.get("/api/v1/:token", (req, res) => {
//   console.log(req.params.token);

  res.status(200).json({ param: req.params.token });
});
```

Lecture 15 - Handle the Date situation

Search for whatever package you need at https://www.npmjs.com

Read - https://www.npmjs.com/package/date-format

```
const format = require("date-format");
```

```
  date: format.asString("dd/MM/yyyy - hh:mm:ss", new Date()),
```

Lecture 16 - Params and bugs

For api/v1/3

We need to write api/v1/:token || api/v1/:token/:id -to grab token as well as id using req.params.token or req.params.id

```
app.get("/api/v1/:token", (req, res) => {
//    console.log(req.params.token);

  res.status(200).json({ param: req.params.token });
});
```

Lecture 17 - Pushing app to Heroku

Git init
Add and commit to git
Optionally push to GITHUB

Create Heroku app
Push code to Heroku app
Debug
Push again

Install Heroku CLI, git

Git init
Create .gitignore - node_module/
Git add .
git commit -m “my 1st production release”

heroku login
heroku git:remote -a lcoyash

Lecture 18 - Debug social app in production

Firstly,

```
const PORT = process.env.PORT || 4000;
```

Heroku and any other uses their own port so it needs to be changed like that

Secondly,
In production dev dependencies are pruned so in scripts nodemon index.js needs to be changed with again node index.js

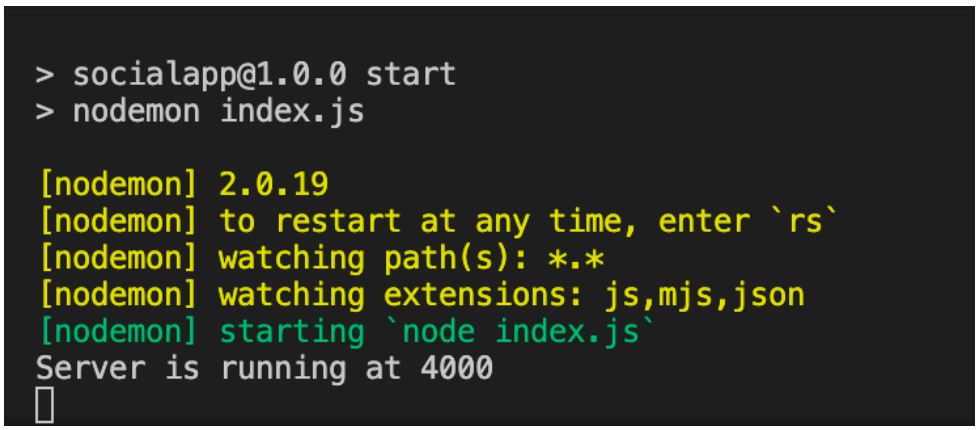
Section 3 - Swagger Docs

Lecture 19 - Swagger Documentation

npm i swagger-ui-express
npm i swagger-jsdoc - To write documentation along with code

npm install --save yamljs

Lecture 20 - Nodemon ext and Yaml



In this you can see watching extension: js, pjs, json

So we need to fix this problem of not rendering our yaml file

Create new file nodemon.json
{
"ext": ".json, .js, .yaml, .jsx"
}

Lecture 21 - Writing Documentation

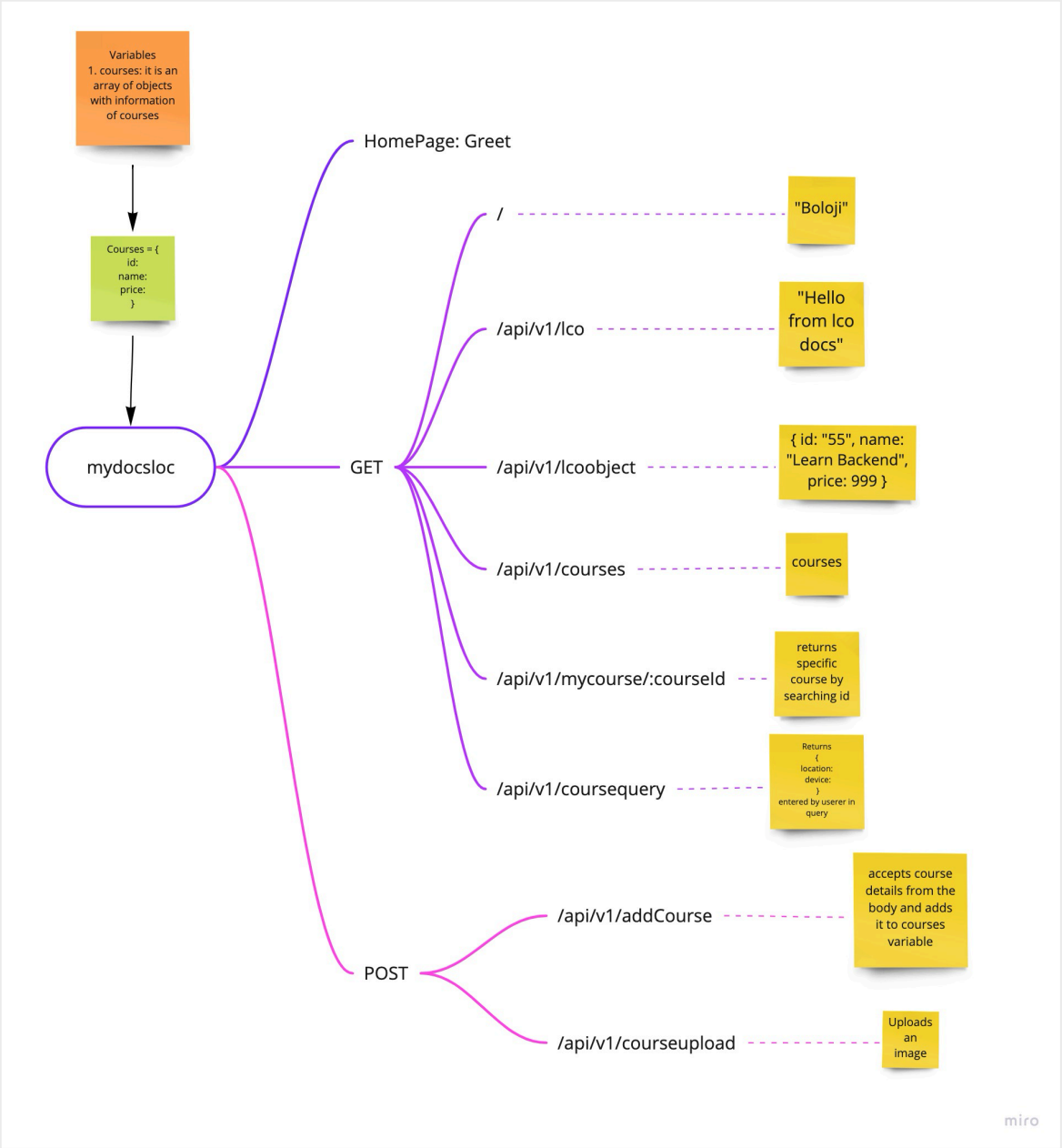
https://swagger.io/docs/specification/basic-structure/

Lecture 22 - Docs for HTTP Method Swaggers

Lecture 23 - New Document Centric Project

Watch this video to get an overview of complete setup of backend project.

Mind Map using Miro:



Lecture 28 - Handling Array in Swagger Docs

Inside array we have items
Inside object we have property (property means a key value pair)

Lecture 29 - Sending Data in URL - Swagger

In this we will learn a “Get” request but a different one in which we will send some information using URL which is through parameters or also called paths to database and based in that will get a result

This is an API to request course information based on courseId passed as a parameter in URL.

```
app.get("/api/v1/mycourse/:courseId", (req, res) => {
  const myCourse = courses.find((item) => item.id === req.params.courseId);
  res.send(myCourse);
});
```

Few things to learn from this

1. How to send parameters in URL
2. How to use .find() in JS or .map(), .filter(), .reduce()
3. Arrow functions in JS

We call URL as path

co > {} swagger.yaml

```yaml
/mycourse/{courseId}:
  get:
    tags:
      - String
    summary: Returns course based on request id
    parameters:
      - name: courseId
        in: path
        required: true
        default: 22
        schema:
          type: string
    responses:
      "200": # status code
        description: All good, success
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: string
                name:
                  type: string
                price:
                  type: number
      "400": # status code
        description: Bad Request

      "500": # status code
        description: Internal server error
```

This is how it is documented in swagger

Few things to note:

1. Tags is to make a group of APIs under same tag

2. Parameters here are used because we pass a value in URL
3. Under schema if type: object then we use ‘properties:’
    - While if it is type: array then we use ‘items:’

Lecture 30 - Managing Request Body in Swagger

Will learn

1. POST
2. Handle JSON data coming from body

Firstly, How we can receive data from webpage :-

1. Data could come as in parameters from URL or as query
    - URL - parameters
    - Query
2. Data could also come from body itself as a form or in the format of JSON
    - HTML Form
    - JSON

Most of the times data comes as JSON

To handle data come to you from express some middleware needs to be used here its app.use()

```
app.use(express.json())
```

Here it says express help me with json file 😂

Get data in body -> push that in array

WHY POST?

- We have used here post request because we are retrieving data from body
- If we wanted to retrieve query from URL we would have used “get” request.

```
app.post("/api/v1/addCourse", (req, res) => {
  console.log(req.body);
  courses.push(req.body);
  res.status(200).json(courses);
  // res.send(true)
});
```

Prints:

```
{ id: '55', name: 'Course added through body', price: 0 }
```

Swagger Documentation:

```yaml
/addCourse:
  post:
    tags:
      - String
    summary: adds a course to enxisting courses
    consumes:
      - application/json
    produces:
      - application/json
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              id:
                type: string
              name:
                type: string
              price:
                type: number
    responses:
      "200": # status code
        description: All good, success
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: string
                  name:
                    type: string
                  price:
                    type: number
      "400": # status code
        description: Bad Request

      "500": # status code
```

Things to learn from this:

1. Consumes: means what it takes as input
2. Produces: Means what it produces as an output

3. In earlier version of swagger a json body request was written in parameters:
   in:
   But now it is written through requestBody:

Lecture 31 - Handle URL Query in Swagger

Learn

1. Query

Query always goes through URL therefore “Get” request

```
app.get("/api/v1/coursequery", (req, res) => {
  let location = req.query.location;
  let device = req.query.device;

  res.send({ location, device });
});
```

Swagger for Query:

```yaml
/coursequery:
  get:
    tags:
      - String
    summary: trying to learn query
    parameters:
      - name: location
        in: query
        schema:
          type: string
          enum: [delhi, london, ahmedabad]
        required: true
      - name: device
        in: query
        required: true
        schema:
          type: string
          enum: [web, mobile]

    responses:
      "200": # status code
        description: All good, success
        content:
          application/json:
            schema:
              type: object
              properties:
                location:
                  type: string
                device:
                  type: string
      "400": # status code
        description: Bad Request
```

How query looks in URL: http://localhost:4000/api/v1/coursequery?location=delhi&device=web

Lecture 32 - Handling Images in Swagger

Express doesn't know how to handle binary data that is images, pdf,...
For this install express-fileupload

Command: npm i express-fileupload
Documentation link to look for example projects: https://www.npmjs.com/package/express-fileupload

Remembered in frontend :
<input type = "file" name="sampleFile" />

The 'name' here is used in backend to fetch this file
Ex: sampleFile = req.files.sampleFile;

```
const fileUpload = require('express-fileupload');
```

These are the middleware so whenever you need to handle JSON or body request or file these middleware helps:

```
app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocument));
app.use(express.json());
app.use(fileUpload());
```

Post API:

```
app.post("/api/v1/courseupload", (req, res) => {
  const sampleFile = req.files.sampleFile;
  let path = __dirname + "/images/" + Date.now() + ".jpg";

  sampleFile.mv(path, (err) => {
    res.send(true);
  });
});
```

Here sampleFile is use in frontend at name="sampleFile"

__dirname - gives complete path of project folder
.mv() helps to move the file to the desired location

These files could be easily saved to firebase or any other platform.

Swagger Documentation for image upload

```yaml
/courseupload:
  post:
    tags:
      - Files
    summary: uploading course image
    requestBody:
      content:
        multipart/form-data:
          schema:
            type: object
            properties:
              sampleFile:
                type: string
                format: binary
    responses:
      "200": # status code
        description: All good, success

      "400": # status code
        description: Bad Request

      "500": # status code
        description: Internal server error
```

Section 4

Lecture 34 - What We Have Done Till Section 3 - Backend

What we will be exploring next :-

1. How to organise file effectively
2. Right now anyone can access APIs and put information, so now we will learn how to secure them
3. Till now whatever we add in server got deleted after a refresh so we need to inject a database.

Lecture 35 - Hiding Secrets in Backend

Some sensitive information must be saved from the attacks and therefore your bank and should only have access to this sensitive information for this

Package Used - dotenv
Link - https://www.npmjs.com/package/dotenv
Installation - npm i dotenv

Lecture 36 - Picking Up a Database For Backend

Going with most popular and widely used database which is MongoDB

Do I Really need ORM or ODM?

ORM - Object Relational Mapper
ODM - Object Data Mapper

These helps to do operations with database

Answering the question : Can you run without wearing running shoes? Definitely, but jogging with a running shoes make it much easier. Similarly, ORM and ODM helps to talk to database.

For this we are going to use Mongoose ODM.
Link - https://mongoosejs.com
Description - Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

Lecture 37 - Why we need Mongoose - ODM

When installing a database like MongoDB you could directly talk to it using its drivers else you could inject mangoes which helps you with some pre-existing requirements like validation and checks.

Another Question,
Should you take data directly from the web and through it in your database?

Absolutely not, data always need to pre-processed first before adding it to a database.

Things to read in Mongoose Documentation

1. Schemas
2. Validation
3. Queries
    1. Model.deleteMany()
    2. Model.deleteOne()
    3. Model.find()
    4. Model.findById()
    5. Model.findByIdAndDelete()
    6. Model.findByIdAndRemove()
    7. Model.findByIdAndUpdate()
    8. Model.findOne()
    9. Model.findOneAndDelete()
    10. Model.findOneAndRemove()
    11. Model.findOneAndReplace()
    12. Model.findOneAndUpdate()
    13. Model.replaceOne()
    14. Model.updateMany()
    15. Model.updateOne()

Lecture 39 - Creating Model for Auth System

Building Authentication System

1. Register
    - User need to provide range of information
2. Login
    - User can now provide short info like username and password to access application
3. Register Routes
    - Some routes could be open for everyone while others could be restricted like only admin could access them

Use Miro for mind mapping

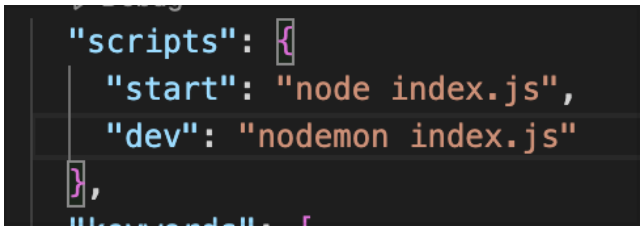
Models generally have first letter capital

Token (From model User:)

- It is auto generated and assigned to a user.
- If you want to use some protected route you have to use this token

Lecture 40 - Creating Basic Structure for Auth System

Setup the project folder



We need to:

1. Connect with Database
2. Put env file
3. Create Models

Create these folders in root directory:

```
lcoauthsystem % mkdir model config middleware
```

Note: You can put env file in config folder but sir has not put that right now in this project.

Now Create these files:

```
lcoauthsystem % touch model/user.js config/database.js middleware/auth.js
```

- database.js is responsible for connecting with database

Some Changes in writing code this time:

app.js M | index.js M

lcoauthsystem > app.js > ...

You, 3 minutes ago | 1 author (You)

```
1  const express = require("express");
2
3  const app = express();
4
5  app.get("/", (req, res) => {
6    res.send("<h1>Hello from Auth System - LCO</h1>");
7  });
8
9  // Instead of writing all listen statements in app.js we will write them in index.js
10 module.exports = app;
11
```

Lecture 41 - Creating User Schema and Dotenv

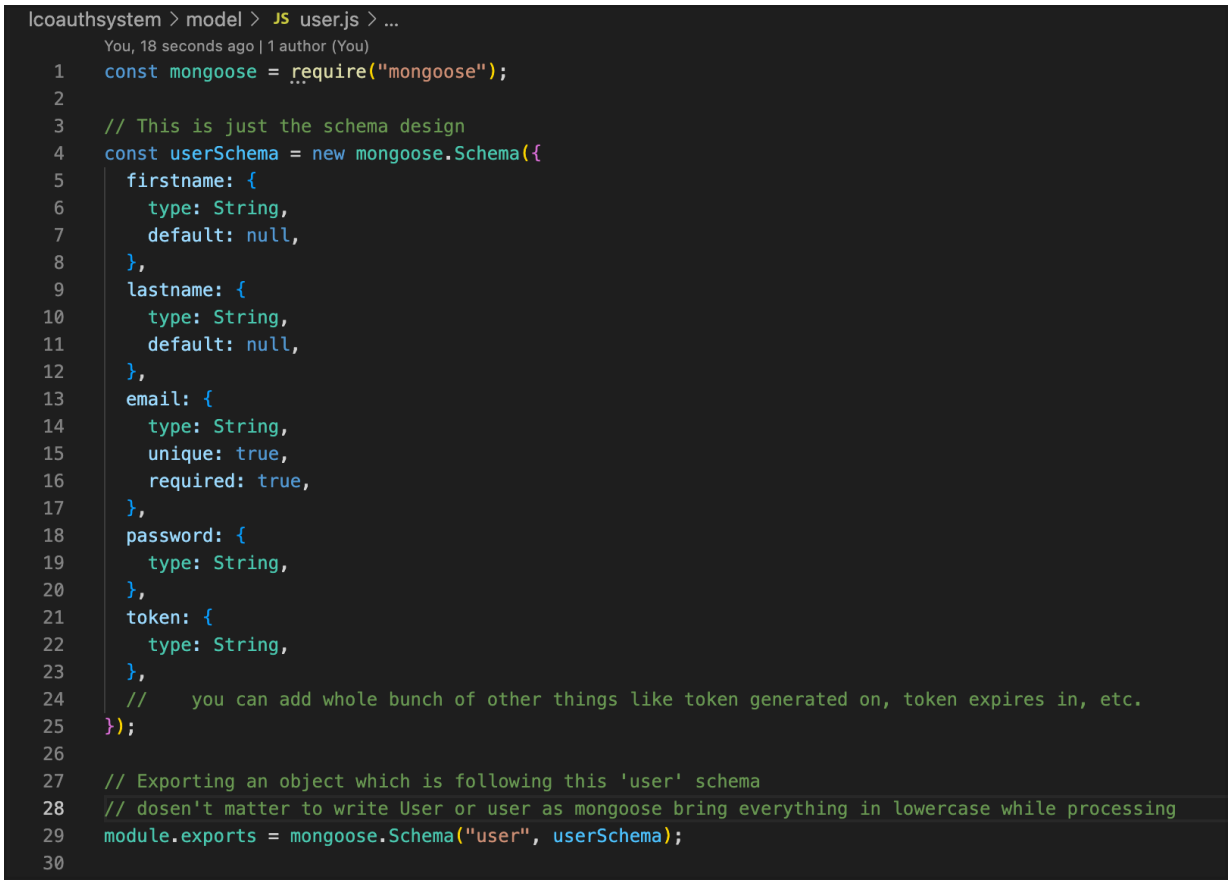
We are going to write env file, some secrets and will write basic model

Setting up PORT Variable:

- Define PORT in .env file and import it in app.js with the help of:
  - require("dotenv").config();

- Later on use it in index.js with the help of:
  - const { PORT } = process.env;
- This statement is same as process.env.PORT

Setting up Model:



The permitted SchemaTypes are:

1. String
2. Number
3. Date
4. Buffer
5. Boolean
6. Mixed
7. ObjectId
8. Array
9. Decimal128
10. Map

Lecture 42 - Registering a User in 4th System

Learn - What are the issues we are going to face while registering a user.

Strategy while registering a user :-

1. Get all information
2. Check mandatory fields
3. Already registered

4. Take care of password
5. Generate Token or send success message

Note: const firstname = req.body.firstname is same as const { firstname } = req.body

```
app.post("/register", (req, res) => {
    const {firstname, lastname, email, password} = req.body

    // Error condition to check whether the requried information is provided or not
    if(!(firstname && lastname && email && password)){
        res.status(400).send("All fields are required.")
    }

    const existingUser = User.findOne({ email: email})

    if(existingUser){
        res.status(401).send("User already exists")      You, 1 second ago • Uncommi
    }
})
```

```
// User or user doesn't matter here also but industry standard in app.js is User
const User = require("./model/user");
```

Lecture 43 - Database Connection in Auth

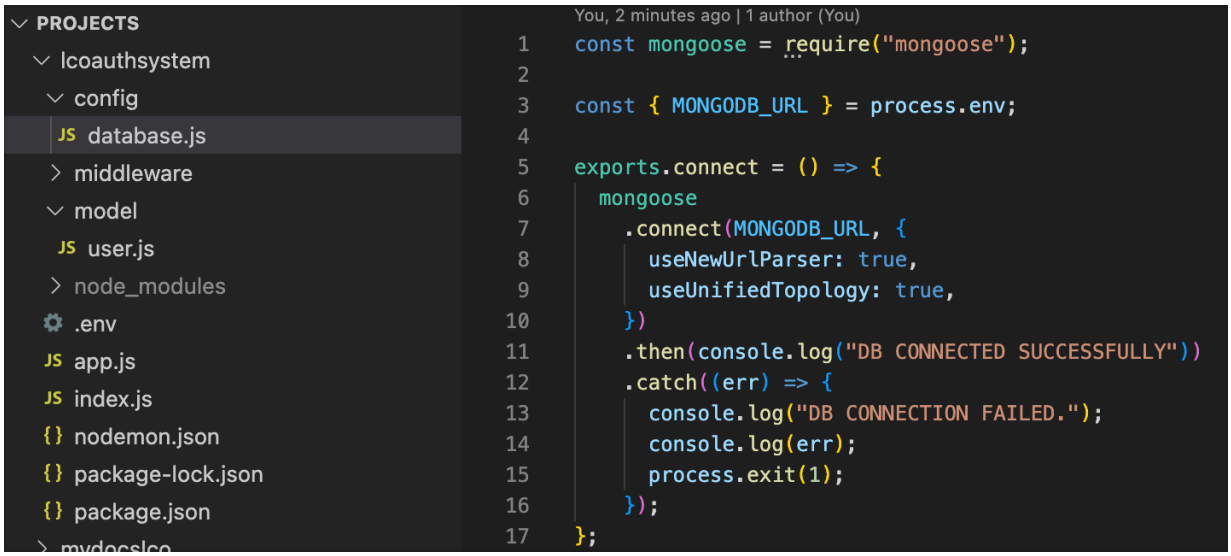
Database is always on different continent

First you need to start mongoDB in terminal for that:

- brew services start mongodb-community
- mongosh

Open MongoDB Compass and connect

I have also created database on MongoDB Atlas
Project name : lcoauthsystem
Username: yash
Password: yash
URL: mongodb+srv://yash:yash@atlascluster.wnh9dnl.mongodb.net/test



Lecture 44 - What is a Middleware

Concept of Middleware?

Simply it means somethings that happens in between.

For example:

1. Office ———> Bakery(to buy pizza and soft drink) ——>Home
2. Office ——> Vendor|
   | If vendor gives cheque > Bank
   |
   | If gives cash > Deposit Office
   |
   | If gives nothing > Don't go anywhere and beat the shit out of him.

In second example you can see the Vendor highly influences what next to do. Thus middleware are important.

Technically,

Some checks/functionality injected in between
It can be in between OR just before OR just after

Also called as Life Cycle Event OR Life Cycle Hooks

Learn:

Next()

Req —> Next() —> Next() —> Dashboard

Lecture 45 - Handling Password Situation

Read this information

Middleware

Middleware (also called pre and post *hooks*) are functions which are passed control during execution of asynchronous functions. Middleware is specified on the schema level and is useful for writing plugins.

- Types of Middleware
- Pre
- Errors in Pre Hooks
- Post
- Asynchronous Post Hooks
- Define Middleware Before Compiling Models
- Save/Validate Hooks
- Naming Conflicts
- Notes on findAndUpdate() and Query Middleware
- Error Handling Middleware
- Aggregation Hooks
- Synchronous Hooks

Types of Middleware

Mongoose has 4 types of middleware: document middleware, model middleware, aggregate middleware, and query middleware. Document middleware is supported for the following document functions. In document middleware functions, `this` refers to the document.

- validate
- save
- remove
- updateOne
- deleteOne
- init (note: init hooks are synchronous)

Right now we will be studying PRE and POST

```
const schema = new Schema(..);
schema.pre('save', function(next) {
  // do stuff
  next();
});
```

- Whatever object (here schema) you are declaring you can use .pre to it
- Here 'save' is an event which means just before saving perform a function called next.

- Remember you must write function(next) and do not make it an arrow function otherwise it will not work.

To Encrypt password we are going to use bcryptjs package
Package name: **bcryptjs**
Documentation: https://www.npmjs.com/package/bcryptjs
Installation: npm i bcryptjs

Import package in variable:

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

Encrypt your password using this statement:

```
const myEncPassword = await bcryptjs.hash(password, 10);
```

Creating a user with all the details we have grabbed:

```
const user = User.create({
  firstname,
  lastname,
  email: email.toLowerCase(),
  password: myEncPassword,
});
```

Lecture 46 - What is JWT and Creating Token

A very simple example explaining token:

- Someone with key can access the car whether or not he has the proper documents or not
- So here keys are your tokens

From the example we can clearly understand tokens needs to be handled very carefully as they give access to routes.

Tokens should get expired it is a great functionality, and whenever user logs in again he should be assigned a new token

Token is made up of three parts:

1. Header

2. Payload
3. Signature

JWT Web Tokens
Documentation - https://jwt.io/introduction

When should you use JSON Web Tokens?

1. Authorization
2. Information Exchange

Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them!

Package name: **jsonwebtoken**
Documentation: https://www.npmjs.com/package/jsonwebtoken
Installation: npm i jsonwebtoken

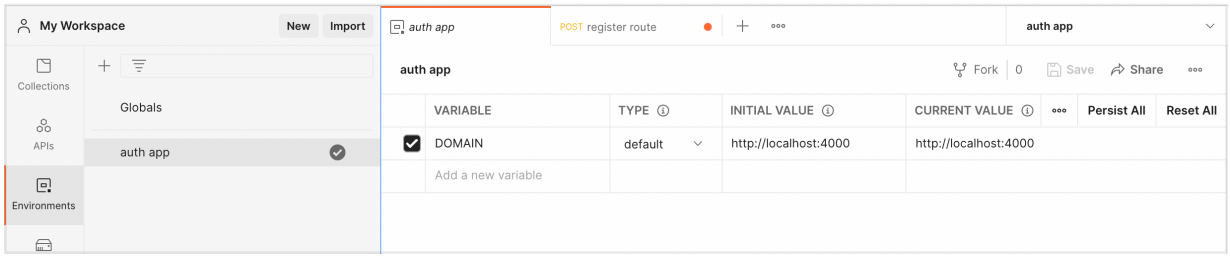
```
// Token Creation - 3 Parts
// Header, Payload, Signature
const token = jwt.sign(
  { user_id: user._id, email },
  process.env.SECRET_KEY,
  // Some Hashing algorithm can ne also added with expiresIn
  {
    expiresIn: "2h",
  }
);

// Assigning the token to the user object
user.token = token;
// Various strategies on Update or not      You, 1 minute ago
```

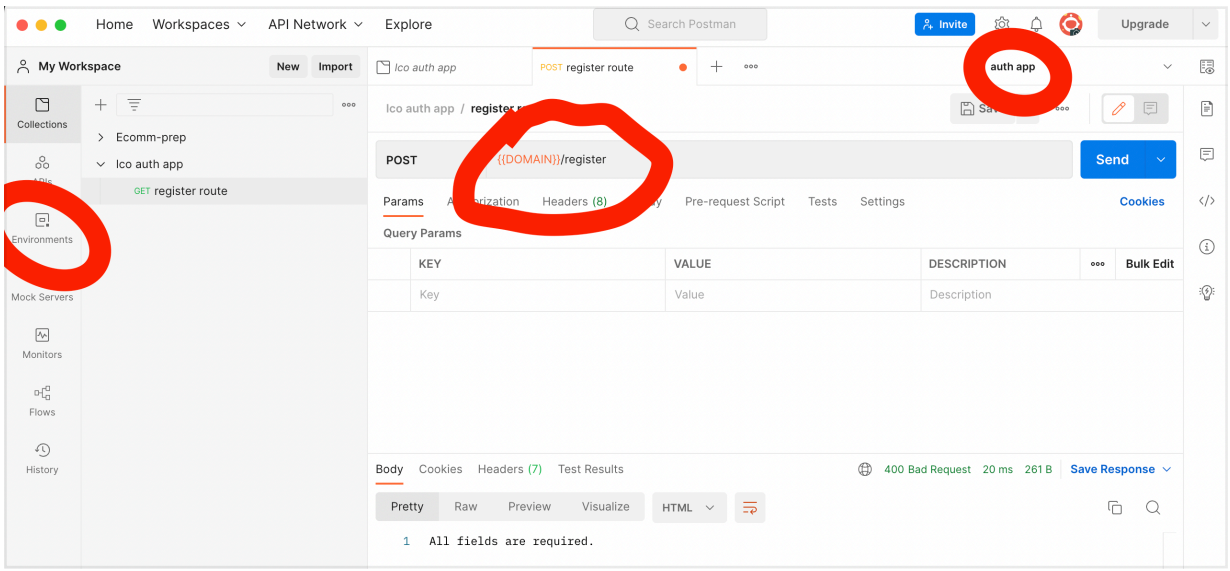
NOTE - here user._id is something that MongoDB database created for the new object created so it is also called objectId

Lecture 47 - Register Route in Auth App

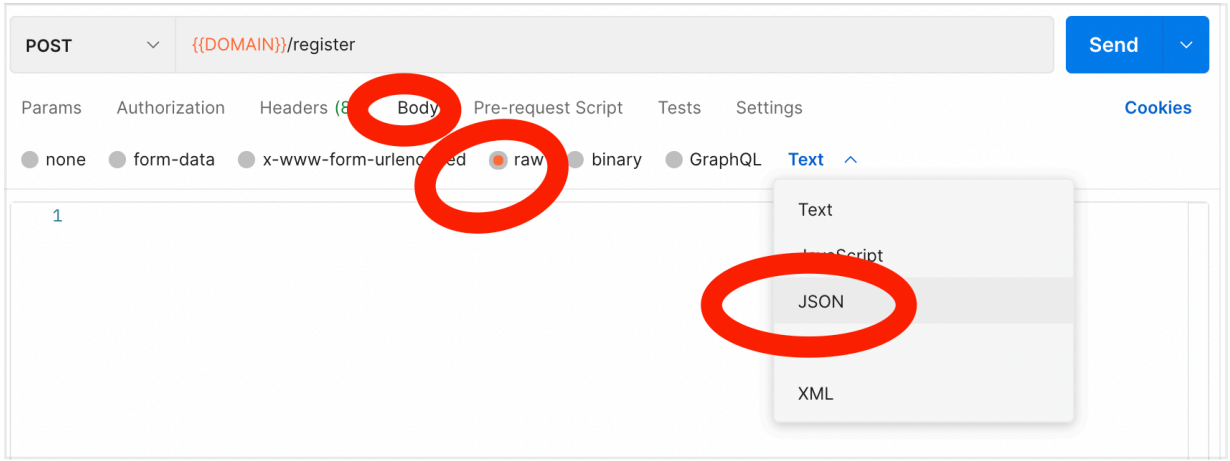
Watch video to learn Postman Professional usage

1. Create Environment
2. Create Collection
3. Use the created environment

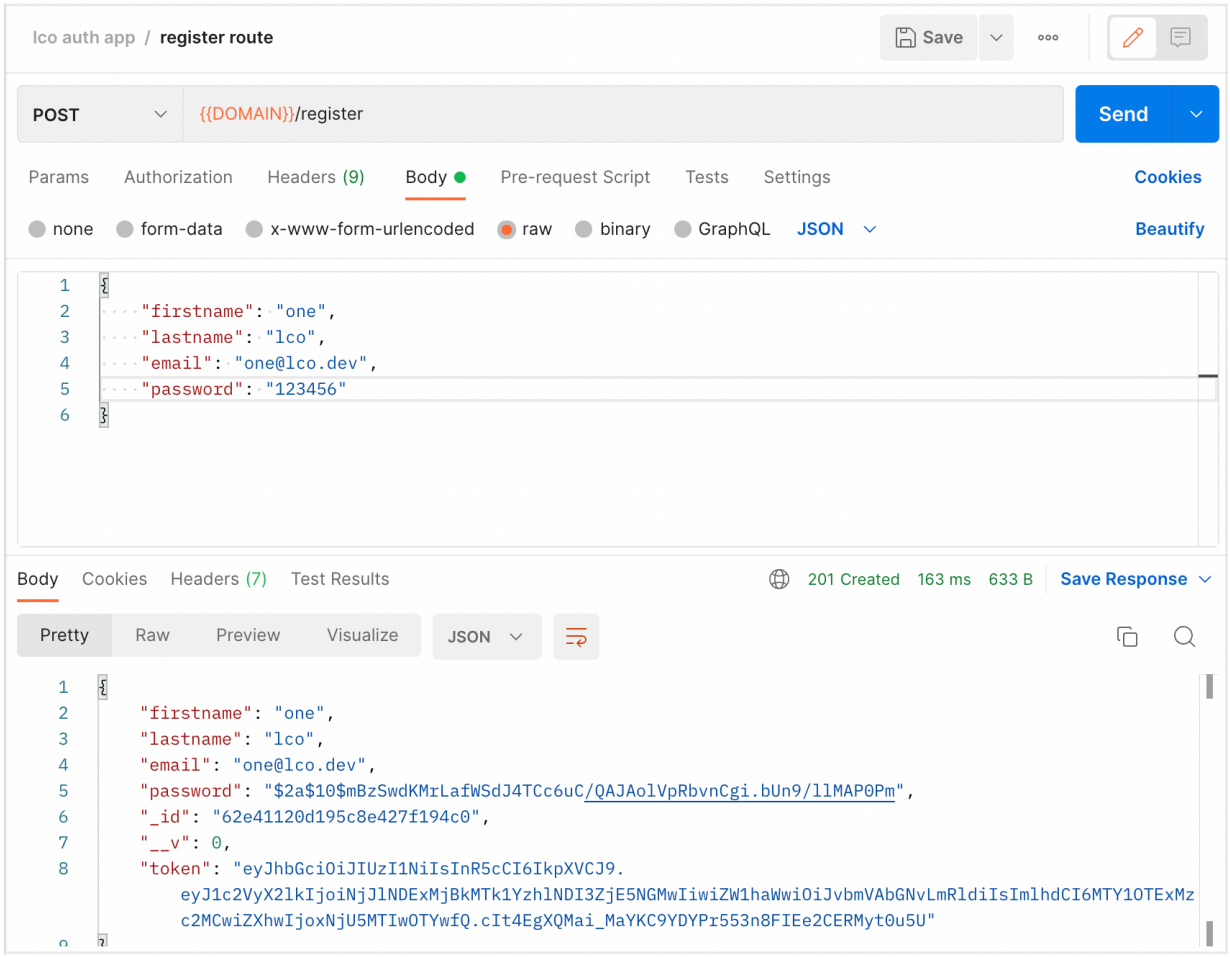




MOST IMPORTANT



GREAT SUCCESS



Problem here is why to send even the encrypted password to someone.

MongoDB Atlas Database : https://cloud.mongodb.com/v2/62e3b9cccf269049e7a04828#metrics/replicaSet/62e3ba032a8ab87d2315bd86/explorer/test/users/find

Lecture 48 - Login Flow For Auth App

Login User

1. Get all Information
2. Check mandatory fields
3. Get user from database
4. Compare and verify password
5. Give token or other information to user

```
app.post("/login", async (req, res) => {
  try {
    // Grabbing enterned email, password
    const { email, password } = req.body;

    // Check if any mandatory field is missing
    if (!(email && password)) {
      res.status(400).send("A required field is missing");
    }

    // Search for the user in database
    const user = await User.findOne({ email });

    // Check is the user registered one?
    if (user === null) {
      res
        .status(400)
        .send(
          "Email entered is not a registered one. Please first register it."
        );
    }
    // For debugging
    // console.log(user);

    // If everything matching generate and asign a token for 2h
    if (email && (await bcryptjs.compare(password, user.password))) {
      const token = await jwt.sign(
        { user_id: user._id, email },
        process.env.SECRET_KEY,
        {
          expiresIn: "2h",
        }
      );

      user.token = token;
      user.password = undefined;
      res.status(200).json(user);
    }

    // If password doesn't match      You, now • Uncommitted changes
    res.send(400).send("Either email or password is incorrect");
  } catch (error) {
    console.log(error);
  }
});
```

Lecture 49 - Web VS Mobile

Why we have created this register and login route?
Of course, to protect other routes which can only be accessed by loved in user.

Strategy for protecting Route

1. Use middleware
2. Check foretoken presence
3. Verify the token
4. Extract info from payload
5. NEXT()

For Web

- Just pass the token to frontend engineer and he will deal with it
- Ways of handling token :-
  1. Cookie - Frontend engineer should know that while he request to access a route he should pass this token in cookies to the backend
     - Use httpOnly so that only backend can access this cookie.
  2. In headers
  3. Body

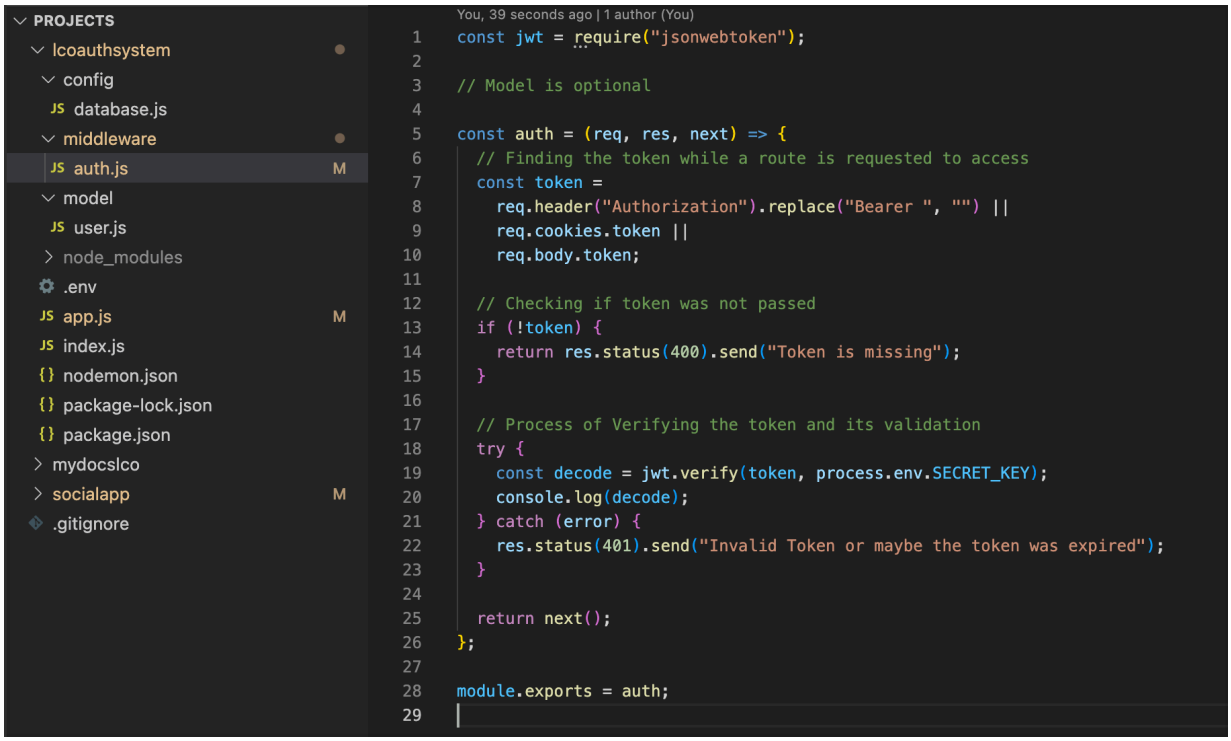
Define a route which can only be accessed by Loged in User

```
app.get("/dashboard", auth, (req, res) => {
  res.send("Welcome to secret information.");
});
```

Here, you can see we have used an ‘auth’ middleware which is defined based on Strategies for Protected Route

Lecture 50 - Writing Custom Middleware

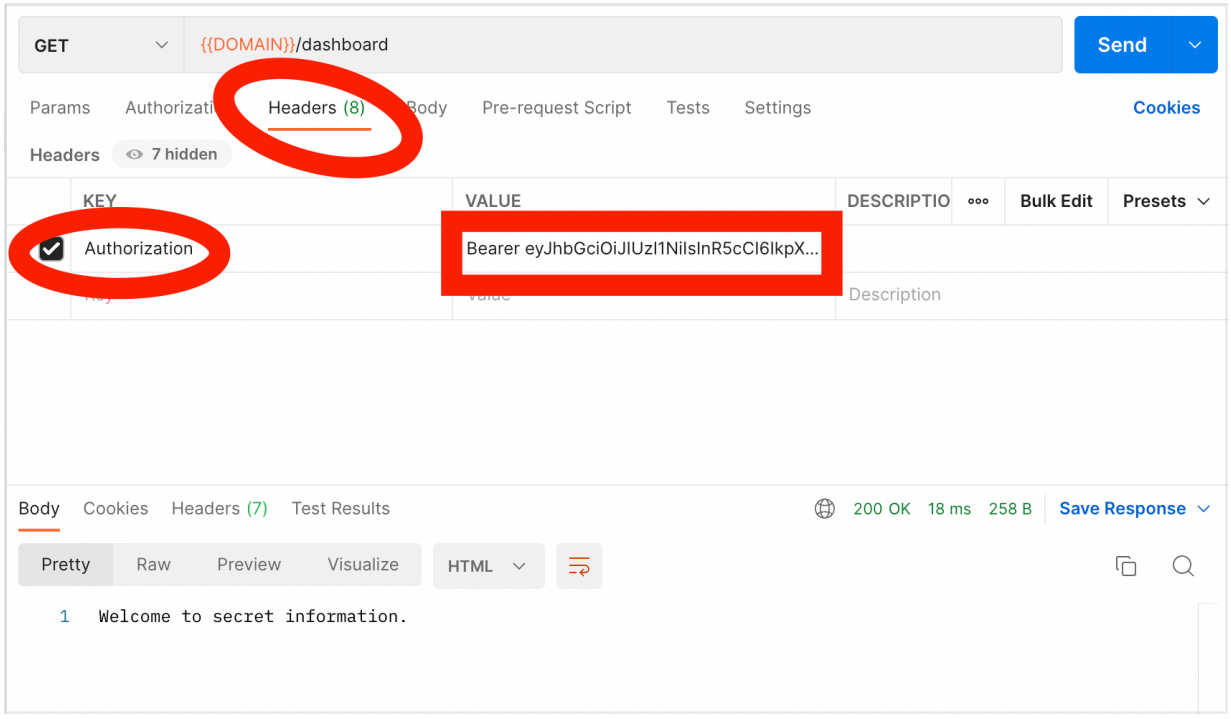
Below is the code for middleware :-



Things to learn :-

- Always use next in middleware
- In line 8 : In headers, token is generally passed with a variable called ‘Authorization’ and its value need to be only defined as ‘Bearer <token>’
- Remember - Always call next() function to return before ending the execution.

Postman



What line 20 prints is :

```
DB CONNECTED SUCCESSFULLY
Server is running at 4000
{
  user_id: '62e41120d195c8e427f194c0',
  email: 'one@lco.dev',
  iat: 1659425121,
  exp: 1659432321
}
```

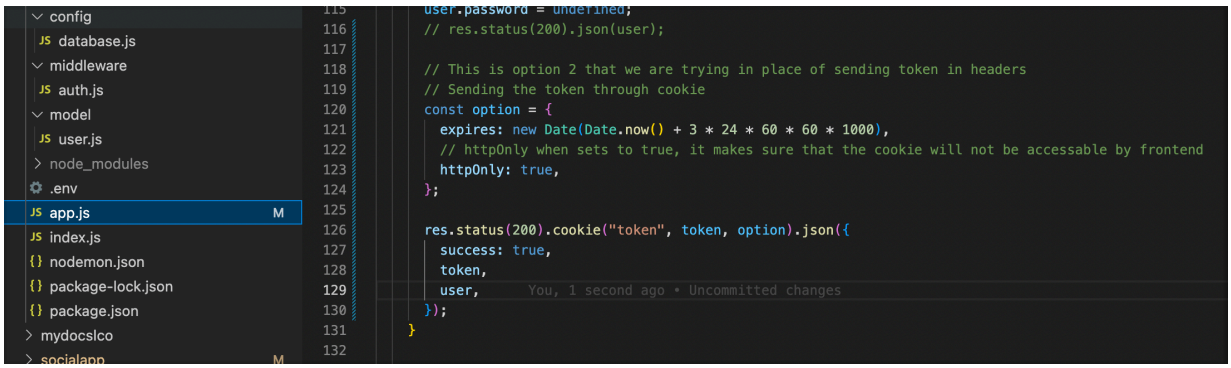
Now there are many ways to handle the decode variable, but the most common one is :

```
17  // Process of Verifying the token and its validation
18  try {
19    const decode = jwt.verify(token, process.env.SECRET_KEY);
20
21    // To check what this decode variable has
22    // console.log(decode);
23
24    // Many ways to use decode variable but the most common one is
25    req.user = decode
26
27    // OR bring in info from DB to grab some more infromation about the user
28
29  } catch (error) {
30    res.status(401).send("Invalid Token or maybe the token was expired");
31  }
```

Lecture 51 - Setting up Secure Cookie

NOTE: Cookies are only accessible in web.
For mobile, use headers.

Code which needs to be add to send a cookie is in file app.js



In line 126,

- 1st argument ‘token’ is as per the name which we are trying to grab in auth.js file with line req.cookies.token
- 2nd argument, token is the name of variable holding the token
- 3rd argument, option is set of properties of that cookie

NOTE: This is not enough to pass and grab a cookie
Just like we used app.use(express.json()) to access json files, similarly, we need to install cookie-parser

Package name: **cookie-parser**
Documentation : https://www.npmjs.com/package/cookie-parser
Installation: npm i cookie-parser

In app.js add:

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

```
app.use(cookieParser())
```

Now, you can see we are able to grab cookie still facing one error of replace:

Note: this was an issue causing because of the order in which token was getting searched

So, this are the modifications to be done in auth.js

```
const token =
  req.cookies.token ||
  req.body.token ||
  req.header("Authorization").replace("Bearer ", "");
```

Now we want to expire this token as soon as the user visits /dashboard, for this:

```
142   app.get("/dashboard", auth, (req, res) => {      You, 41 minutes ago
143     res.clearCookie("token").send("Welcome to secret information.");
144   });
```

.clearCookie('token') helps to delete the token.

Section 5 - File, Image and Form Handling

Lecture 52 - Why People Face Issues in Image Upload

Handling Forms and Images
There are two types of forms :-

1. GET Form
2. POST Form
3. Upload file/files

There are several problems faced while uploading files

There are three modes

`FileReader.readyState` Read only

A number indicating the state of the `FileReader`. This is one of the following:

Name	Value	Description
`EMPTY`	`0`	No data has been loaded yet.
`LOADING`	`1`	Data is currently being loaded.
`DONE`	`2`	The entire read request has been completed.

To understand all the problems lets go back to mydocslco Project

After adding the line 77 you can see

```
{
  name: 'Screenshot 2022-08-02 at 2.53.01 PM.png',
  data: <Buffer 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 07 5c 00 00 02 d6 08 06 00 00 00 e2 60 e1 4c 00 00 0c 6d 69 4
3 43 50 49 43 43 20 50 72 6f 66 69 ... 463543 more bytes>,
  size: 463593,
  encoding: '7bit',
  tempFilePath: '',
  truncated: false,
  mimetype: 'image/png',
  md5: '142a1bae64cccbee10f94898864f7a10',
  mv: [Function: mv]
}
```

Here the mail variable causing all the problems is **tempFilePath**. We need to understand this.

Lecture 53 - Cloudinary and EJS

Install EJS for embedding HTML in nodeJS
Create Cloudinary account

Make new Project - formsandimages

```
npm i express express-fileupload ejs cloudinary nodemon
```

Lecture 54 - How GET works and Postman Issues

IMPORTANT: All of your form comes in body with URL Encoding and not in query and therefor there is a need to use a middleware:

```
// When Data comes in URL Encoded format on not JSON, we must use this middleware
app.use(express.urlencoded());
```

BUT if you want to handle a form which comes as :-

```
{
    "name" : {
        "firstname" : "Yash",
        "lastname" : "Patel"
    }
}
```

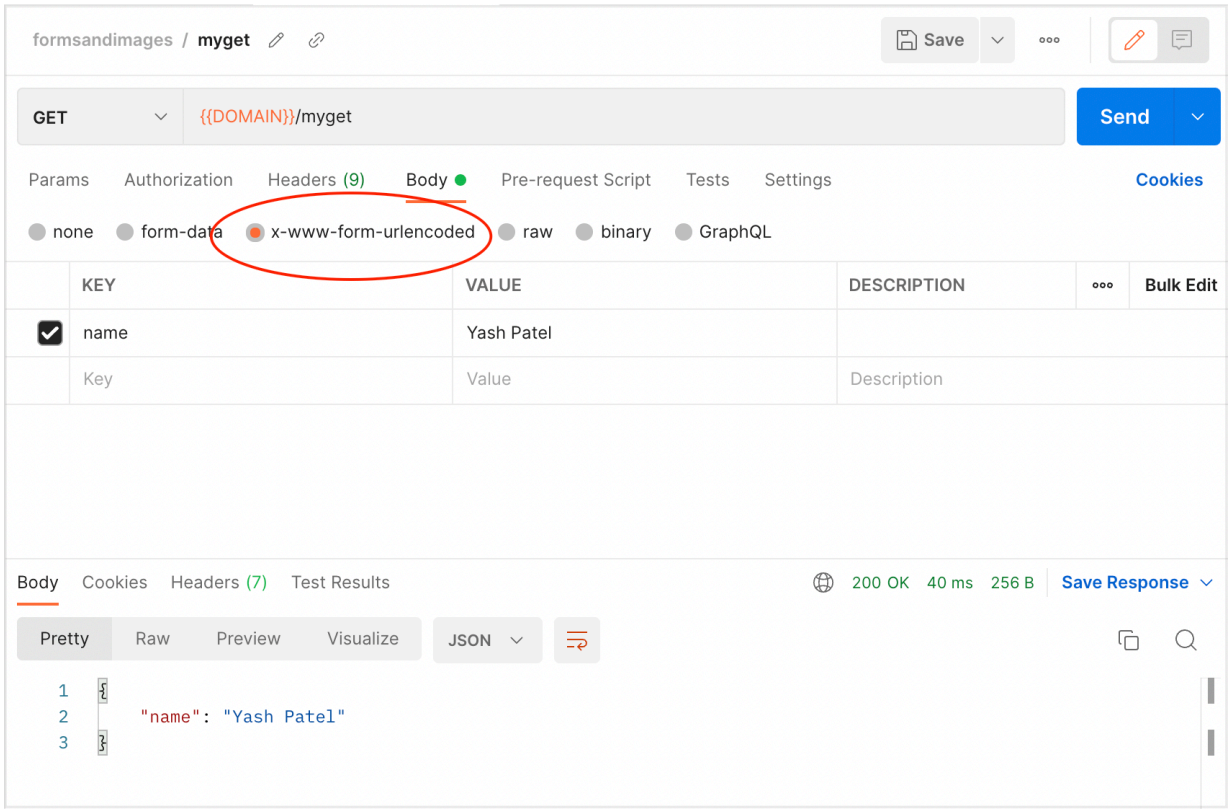
Then you need to use

```
// When Data comes in URL Encoded format on not JSON, we must use this middleware
app.use(express.urlencoded( { extended: true } ));
```

API

```
app.get("/myget", (req, res) => {
  console.log(req.body);
  res.send(req.body);
});
```

Postman



LIMITATION OF POSTMAN: here we can access forms data with URL encoding only but in that we do not get a feature of uploading images.
In form-data we can pass files but that could not read by our express app.

Lecture 55 - Using Templet Engines

We need to define this view middleware to use ejs

```
app.set("view engine", "ejs");
```

After this, we must create a folder with 'views' directory name only strictly, in which two files getform.ejs and postform.ejs

Lecture 56 - Biggest Confusion in Front-end Forms

IMPORTANT
In case of EJS or any other templating engine, data from form is coming purely from URL.
In any other case, like any framework say ReactJS, angular, anything, data behaves exactly like in postman.

THESE ARE ALL CASE OF GET FORM

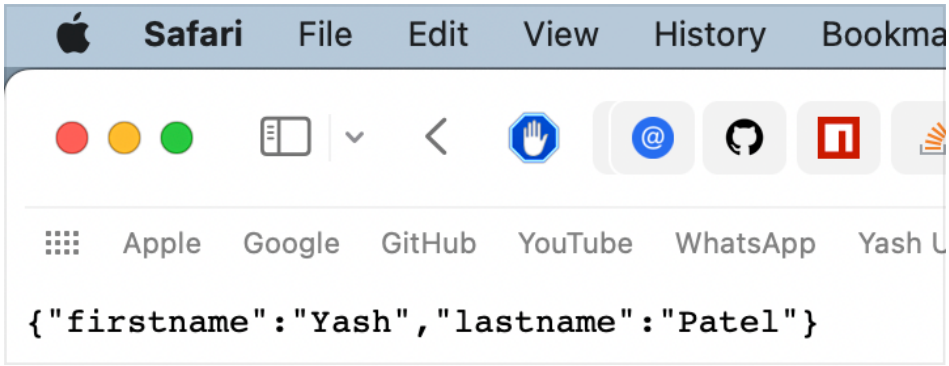
In case of templating engine, after filling the form data will travel through URL in Query form like this - http://localhost:4000/myget?firstname=Yash&lastname=Patel

So in API /myget you need to change res.send(req.body) to res.send(req.query)

MAKE DECISION WITH FRONTEND TEAM
What they are going to use - Template Engine OR Framework

In template engine - not a great idea

```
10
11  app.get("/myget", (req, res) => {
12    console.log(req.body);
13    res.send(req.query);
14  });
```



And postman will also not work

formsandimages > JS index.js > ...

You, 9 minutes ago | 1 author (You)

```
1   const express = require("express");
2
3   const app = express();
4
5   app.set("view engine", "ejs");
6   app.use(express.json());
7
8   // When Data comes in URL Encoded format on not JSON, we must use this middleware
9   app.use(express.urlencoded({ extended: true }));
10
11  app.get("/myget", (req, res) => {
12    console.log(req.body);
13    res.send(req.query);
14  });      You, 9 minutes ago • Uncommitted changes
15
16  app.get("/mygetform", (req, res) => {
17    res.render("getform");
18  });
19  app.get("/mypostform", (req, res) => {
20    res.render("postform");
21  });
22
23  app.listen(4000, () => {
24    console.log("listening on Port 4000...");
25  });
```

Lecture 57 - Handling Images in Forms

NOTE - Never every handle images using GET form, always use POST form to deal with images

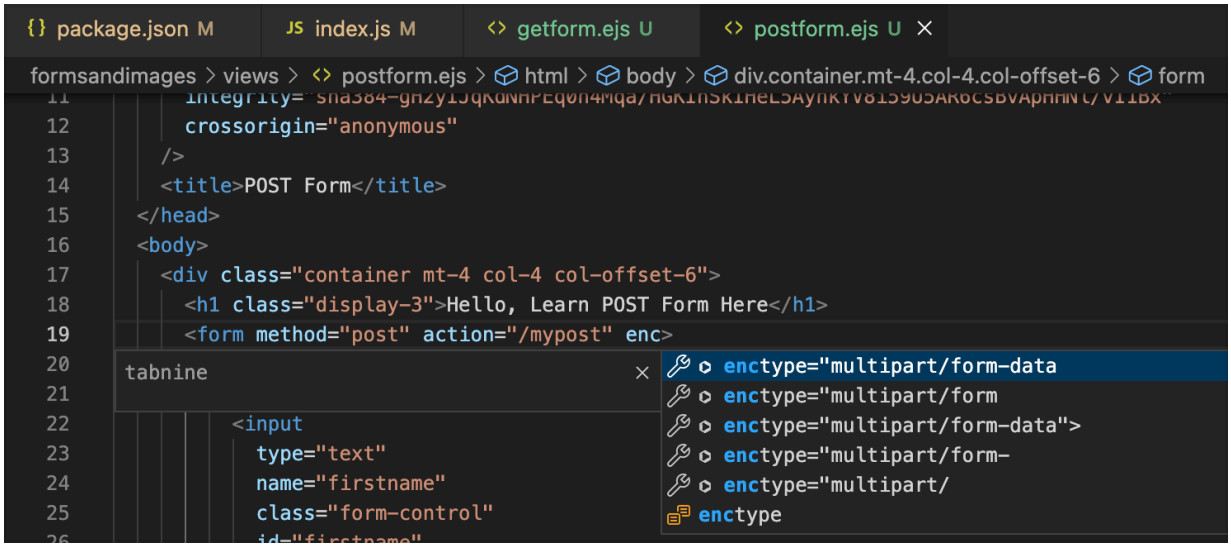
IMPORTANT - How middleware needs to be modified in order to handle files

```
2   const fileUpload = require("express-fileupload");
```

Creates temporary zone where our file which is in a queue of being uploaded will stay for a while

```
16  app.use(
17    fileUpload({
18      useTempFiles: true,
19      tempFileDir: "/tmp/",
20    })
21  );
```

Frontend thing



Final API and its output

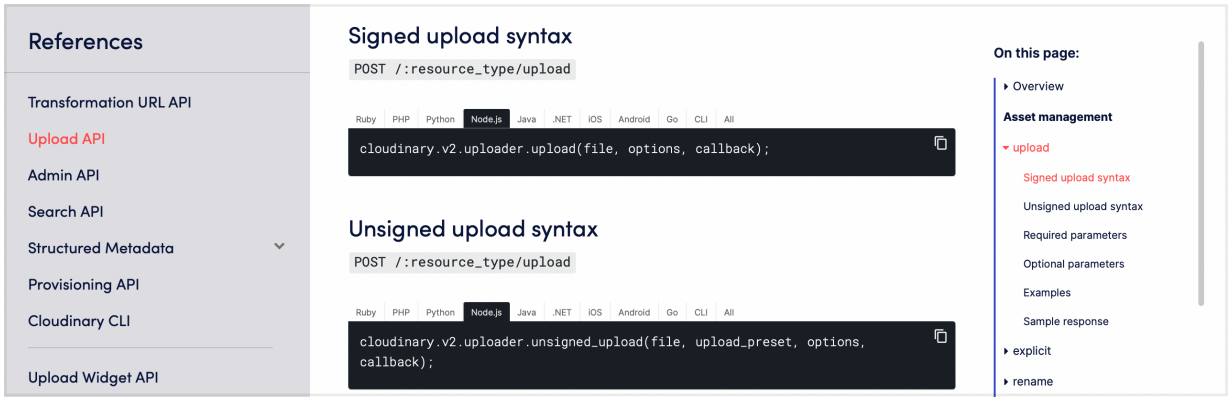
```
23  app.post("/mypost", (req, res) => {
24    console.log(req.body);
25    console.log(req.files);      You, 1 second ago • Uncom
26    res.send(req.body);
27  });
28
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS JUPY

```
listening on Port 4000...
{ firstname: 'Yash', lastname: 'Patel' }
[Object: null prototype] {
  sampleFile: {
    name: 'Screenshot 2022-08-02 at 2.53.01 PM.png',
    data: <Buffer 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52
 43 43 50 49 43 43 20 50 72 6f 66 69 ... 463543 more bytes>,
    size: 463593,
    encoding: '7bit',
    tempFilePath: '',
    truncated: false,
    mimetype: 'image/png',
    md5: '142a1bae64cccbee10f94898864f7a10',
    mv: [Function: mv]
  }
```

Lecture 59 - Upload Image to Cloudinary or Other Providers

Search for cloudinary API



Give unique name to each file while uploading anywhere using NANO ID, YOYO ID. But cloudinary returns a public_id which helps in overcoming this situation

You can see we are able to catch the file but its not getting uploaded because we need to access the temporary zone in which file has been moved from frontend and thus updating file to file.tempFilePatth will work

```
42      result = await cloudinary.uploader.upload(file, { folder: "users" });
43
```

need to change

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS JUPYTER COMMENTS node - formsandima

```
    size: 463593,
    encoding: '7bit',
    tempFilePath: '/tmp/tmp-1-1659517675140',
    truncated: false,
    mimetype: 'image/png',
    md5: '142a1bae64cccbee10f94898864f7a10',
    mv: [Function: mv]
  }
}
node:internal/errors:477
    ErrorCaptureStackTrace(err);
    ^

TypeError [ERR_INVALID_ARG_TYPE]: The "path" argument must be of type string. Received an instance of Object
    at new NodeError (node:internal/errors:387:5)
    at validateString (node:internal/validators:115:11)
```

Final API

```
33  app.post("/mypost", async (req, res) => {
34    console.log(req.body);
35    console.log(req.files);
36
37    // Grabbing file comming from form
38    let file = req.files.sampleFile;
39
40    // Use exact keyword 'result' according to docs as cloudinary will return some data after upload
41    // according to docs - cloudinary.v2.uploader.upload(file, options, callback);
42    result = await cloudinary.uploader.upload(file.tempFilePath, {
43      folder: "users",
44    });
45
46    console.log(result);      You, 1 second ago • Uncommitted changes
47
48    // Setting up details of what we have got from POST API
49    details = {
50      firstname: req.body.firstname,
51      lastname: req.body.lastname,
52      result,
53    };
54    res.send(details);
55  });
```

All the Details of Upload

```
[nodemon] starting `node index.js`
listening on Port 4000...
{ firstname: 'Yash', lastname: 'Patel' }
[Object: null prototype] {
  sampleFile: {
    name: 'Screenshot 2022-08-02 at 2.53.01 PM.png',
    data: <Buffer >,
    size: 463593,
    encoding: '7bit',
    tempFilePath: '/tmp/tmp-1-1659518313681',
    truncated: false,
    mimetype: 'image/png',
    md5: '142a1bae64cccbee10f94898864f7a10',
    mv: [Function: mv]
  }
}
{
  asset_id: 'f2befd3b94ca1603c9fa588808b2ec3a',
  public_id: 'users/ldn8pfhzrjxnalsssz7t',
  version: 1659518315,
  version_id: '3173e57b429d0e5d2e04156d286f99f1',
  signature: '4c0fb466c95e4bca8afd8737dae8136234b31874',
  width: 1884,
  height: 726,
  format: 'png',
  resource_type: 'image',
  created_at: '2022-08-03T09:18:35Z',
  tags: [],
  bytes: 463593,
  type: 'upload',
  etag: '142a1bae64cccbee10f94898864f7a10',
  placeholder: false,
  url: 'http://res.cloudinary.com/yashpatel4900/image/upload/v1659518315/users/ldn8pfhzrjxnalsssz7t.png',
  secure_url: 'https://res.cloudinary.com/yashpatel4900/image/upload/v1659518315/users/ldn8pfhzrjxnalsssz7t.png',
  folder: 'users',
  original_filename: 'tmp-1-1659518313681',
  api_key: '284557364345656'
}
```

SUCCESS!!
But we need to learn about how to handle multiple files

Lecture 60 - Handling Multiple Files and Uploading Them

A modification in Frontend to handle multiple files

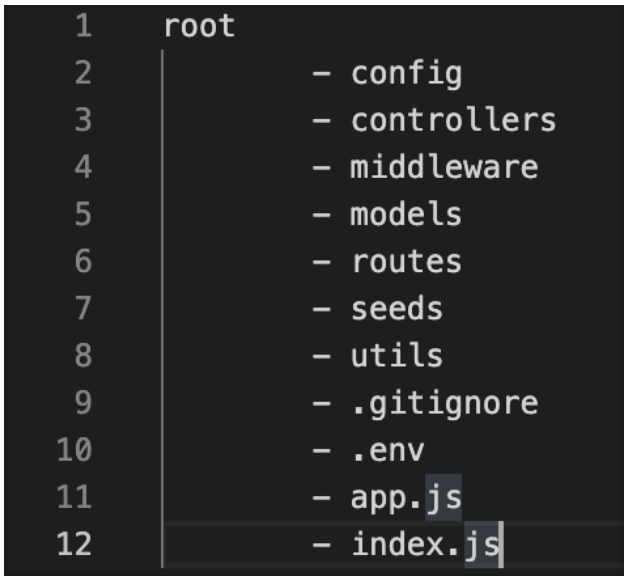
```
42          <div class="mb-3">
43            <label for="sampleFile" class="form-label">File</label>
44            <input
45              type="file"
46              name="sampleFile"      You, 2 hours ago • getforms and pos
47              multiple
48              class="form-control"
49              id="sampleFile"
50            />
```

POST API handling multiple files and uploading them

```
app.post("/mypost", async (req, res) => {
  console.log(req.body);
  // console.log(req.files);

  // Handling Multiple files

  let result;
  let imageArray = [];

  if (req.files) {
    for (let index = 0; index < req.files.sampleFile.length; index++) {
      result = await cloudinary.uploader.upload(
        req.files.sampleFile[index].tempFilePath,
        {
          folder: "users",
        }
      );
      You, 21 seconds ago • Handling and Uploading Multiple Files - POST Form…
      // Pushing details of each file being uploaded in array
      imageArray.push({
        public_id: result.public_id,
        secure_url: result.secure_url,
      });
    }
  }

  // Handling a single file
  // // // // // // // // // // // // // // // // // // // // //
  // Grabbing file comming from form
  // let file = req.files.sampleFile;

  // Use exact keyword 'result' according to docs as cloudinary will return some data after upload
  // according to docs - cloudinary.v2.uploader.upload(file, options, callback);
  // result = await cloudinary.uploader.upload(file.tempFilePath, {
  //   folder: "users",
  // });

  console.log(result);

  // Setting up details of what we have got from POST API
  details = {
    firstname: req.body.firstname,
    lastname: req.body.lastname,
    result,
    imageArray,
  };

  console.log(details);
  res.send(details);
});
```

Section 6 - Theory and Razorpay

File Structure



Install and Use Production Grade Loggers
Example: solarwinds, splunk, winston, morgan
Crash Course: https://www.youtube.com/watch?v=m2q1Cevl_qw

Here we are going to use Morgan
Package name: **morgan**
Documentation: https://www.npmjs.com/package/morgan
Install: npm i morgan

Promises-trycatch-BigPromise

```
// Suppose you want to add this user to database
myuser = {
    name : "Yash",
}

// Using Promises
const saveUser = (req, res) => {
    User.create(myuser)
        .then( user => {} )
        .catch()
}

// Using try catch - Remember its not enough and somethimes you still need to use Promises
const saveUser = async (req, res) => {
    try {
        const user = await User.create(myuser);
    } catch(){

    }
}

// Helper
const BigPromise = func => (req,res,next) => {
    Promise.resolve( func(req,res,next) ).catch(next)
}
```

Mail Service to be used - NodeMailer
Documentation - https://nodemailer.com/about/
Installation - npm install nodemailer

All that needs to be taken care of in case of nodemailer:

```
"use strict";
const nodemailer = require("nodemailer");

// async..await is not allowed in global scope, must use a wrapper
async function main() {
  // Generate test SMTP service account from ethereal.email
  // Only needed if you don't have a real mail account for testing
  let testAccount = await nodemailer.createTestAccount();

  // create reusable transporter object using the default SMTP transport
  let transporter = nodemailer.createTransport({
    host: "smtp.ethereal.email",
    port: 587,
    secure: false, // true for 465, false for other ports
    auth: {
      user: testAccount.user, // generated ethereal user
      pass: testAccount.pass, // generated ethereal password
    },
  });

  // send mail with defined transport object
  let info = await transporter.sendMail({
    from: '"Fred Foo 👻" <foo@example.com>', // sender address
    to: "bar@example.com, baz@example.com", // list of receivers
    subject: "Hello ✔", // Subject line
    text: "Hello world?", // plain text body
    html: "<b>Hello world?</b>", // html body
  });

  console.log("Message sent: %s", info.messageId);
  // Message sent: <b658f8ca-6296-ccf4-8306-87d57a0b4321@example.com>
```

```
  // Preview only available when sending through an Ethereal account
  console.log("Preview URL: %s", nodemailer.getTestMessageUrl(info));
  // Preview URL: https://ethereal.email/message/WaQKMgKddxQDoou...
}

main().catch(console.error);
```

Razorpar Frontend Integration Documentation: https://razorpay.com/docs/

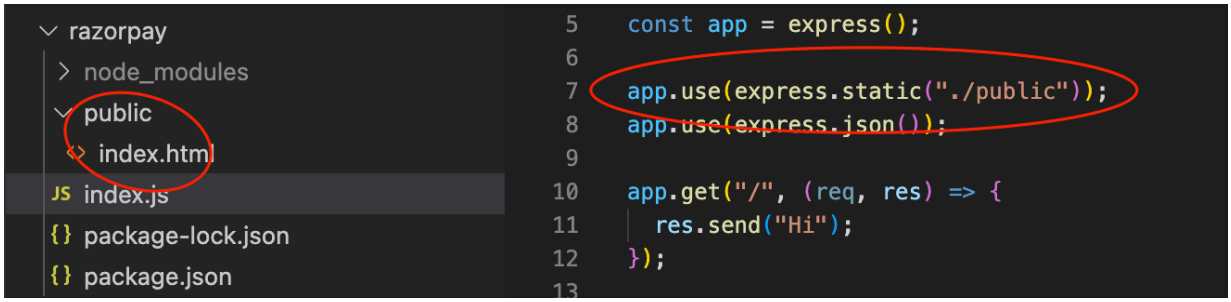
payments/payment-gateway/web-integration/standard/build-integration/#12-integrate-with-checkout-on-client-side

Razorpay API

```
app.post("/order", async (req, res) => {      You, 1 hour
  const amount = req.body.amount;

  var instance = new Razorpay({
    key_id: "rzp_test_zhN8KyrEfaawFX",
    key_secret: "hgTStQlvJPveKgjPFh1Jz27l",
  });

  var options = {
    amount: amount * 100,
    currency: "INR",
    receipt: "receipt#1",
  };

  const myOrder = await instance.orders.create(options);

  res.status(200).json({
    success: true,
    amount,
    order: myOrder,
  });
});
```

Frontend Part

How to render HTML File through express

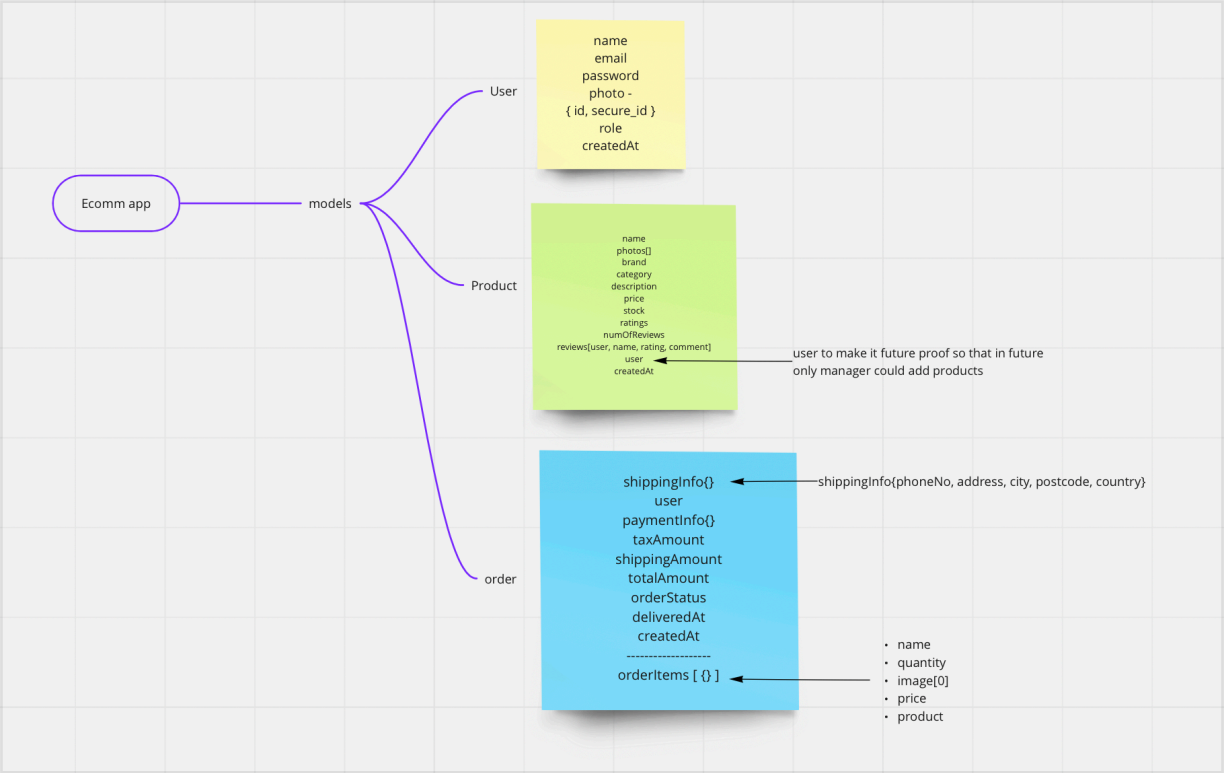


Section 7 - Big Ecommerce App Starts

Lecture 68 - Project Requirement

Problem Statement
I want to sell T-shirts (3 main category initially) online, keep track of all customers and orders. I also want to add new products and keep check on stock.

Lecture 71 - Model Discussion



Lecture 72 - How Forget Password Feature Works

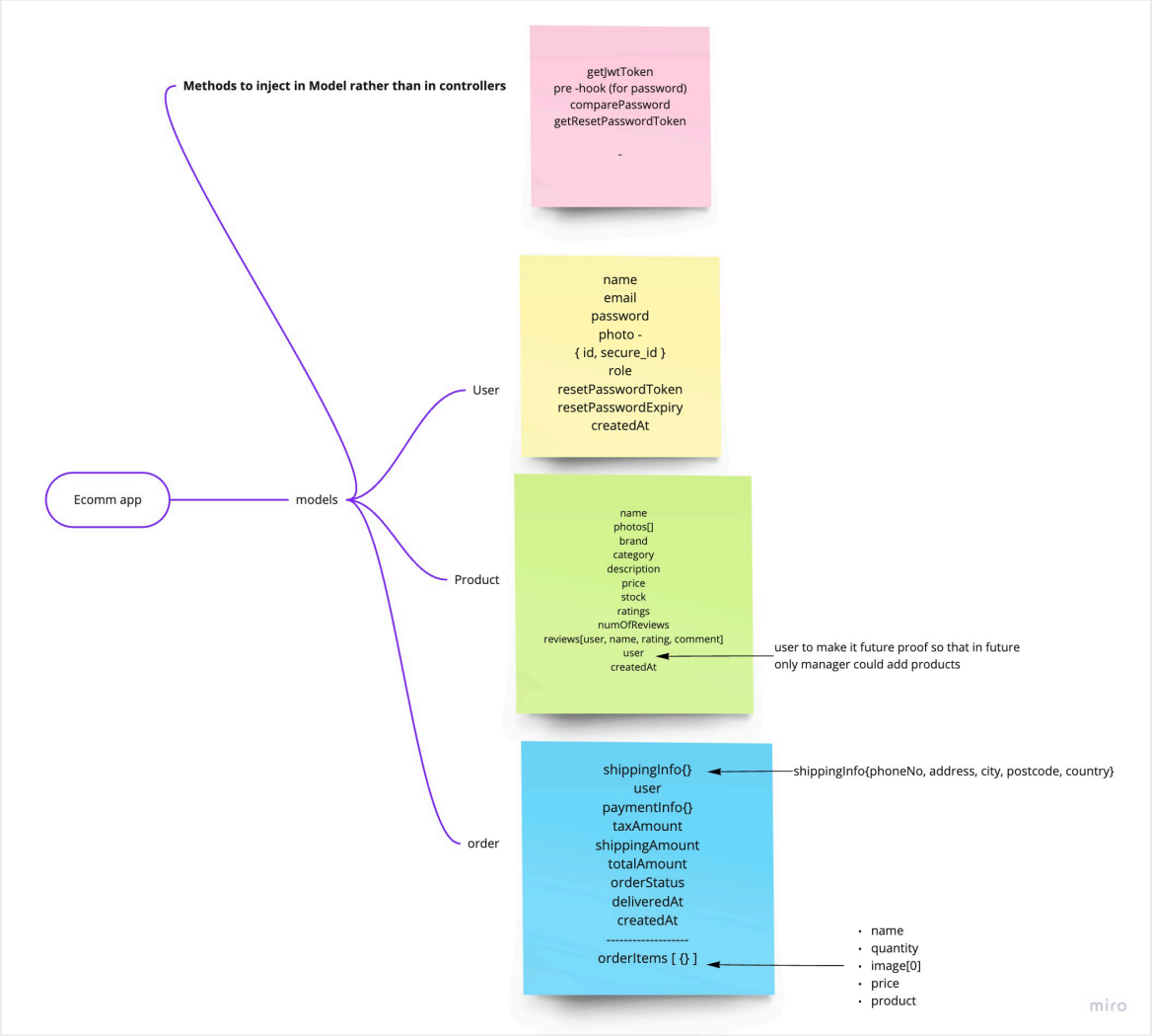
Similar logic for verifying the email and user

There will be a database and client

1. When clients sends a request of Forgot Password
2. A function called tokenGeneration() on server will generate a token and pass it to database as well as client's email with an expiration time
3. When client sends request through this token
    1. First check that token has not expired
    2. Allow client to enter new password

4. Delete old password and token
5. Add new password

Lecture 73 - Functions in User Model and Hooks



Section 8 - Basic config and Imports

Lecture 74 - Getting Files and Folders Ready

List of all Packages we will be needing for this project
npm i bcryptjs cloudinary cookie-parser dotenv express express-fileupload mongoose jsonwebtoken nodemailer stripe razorpay validator morgan swagger-ui-express yamljs ejs

npm i -D nodemon

Create Organised directory
mkdir config controllers middlewares models routes utils

Create files
touch index.js app.js

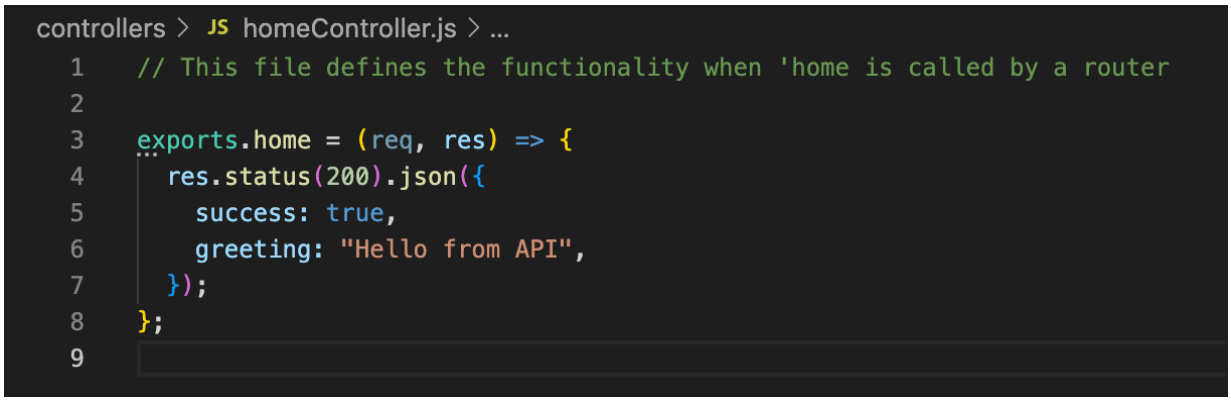
Create new file nodemon.json , swagger.yaml
{
"ext": ".json, .js, .yaml, .jsx ."env
}

Lecture 75 Preparing Basic Express App

Set up index and app.js files as before

Lecture 76 Routes and Controllers in Dummy

Set up a basic controller like:



Now, home.js in routers will be responsible for all the routes coming to home page which will be '/'

routes > JS home.js > ...

```
1   const express = require("express");
2   const router = express.router();
3
4
5
6   module.exports = router;
```

These are the first things to do while defining router

Now,

routes > JS home.js > ...

```
1   const express = require("express");
2   const router = express.router();
3
4   // This {home} comes as the variable name defined in homeController.js
5   const { home } = require("../controllers/homeController");
6
7   // This statement defines where to go when '/' hits
8   router.route("/").get(home);
9
10  module.exports = router;
11
```

So basically when anyone send ‘/‘ request and its a get request then the routers will be triggered searching for a functionality which needs to be executed. This functionality is defined inside a controller and that’s how everything works

Defining how and who will call these routes - Of course app.js

```
6   // // // // // This was traditional way of doing things
7   // app.get("/", (req, res) => {
8   //   res.status(200).send("Helloji");
9   // });
10  // // // // // // // Now we need to just import all the routes
11  // // // // // // // and routes will guide us to what functionality should get triggered
12
13  // Import all routes
14  const home = require("./routes/home");
15
16  // Router Middleware - REQUIRED
17  app.use("/api/v1", home);
```

Complete Mind Map

app.js

```
// Import all routes
const home = require("./routes/home");

// Router Middleware - REQUIRED
app.use("/api/v1", home);
```

Import the route

Asks what happens after /api/v1 to be handled by 'home' route

./routes/home.js

If it's '/' and a GET request it gets handled by

If it's '/dummyHome' and a GET request it gets handled by

```
// This {home} comes as the variable name defined in homeController.js
const { home, dummyHome } = require("../controllers/homeController");

// This statement defines where to go when '/' hits
router.route("/").get(home);
router.route("/dummyHome").get(dummyHome);
```

./controllers/homeController.js

controllers > JS homeController.js > ...

```
// This file defines the functionality when 'home is called by a router

exports.home = (req, res) => {
  res.status(200).json({
    success: true,
    greeting: "Hello from API",
  });
};

exports.dummyHome = (req, res) => {
res.status(200).json({
    success: true,
    greeting: "This is a dummy route",
});
};
```

Lecture 77 Injecting Docs and Middleware

app.js > ...

You, 27 seconds ago | 1 author (You)

```
require("dotenv").config();
const express = require("express");
const morgan = require("morgan");
const cookieParser = require("cookie-parser");
const fileUpload = require("express-fileupload");

const app = express();

// For swagger Documentation
const swaggerUi = require("swagger-ui-express");
const YAML = require("yamljs");
const swaggerDocument = YAML.load("./swagger.yaml");
app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocument));

// Regular Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Cookies and File Middleware
app.use(cookieParser());
app.use(fileUpload());

// Morgan Middleware
app.use(morgan("tiny"));
```

Lecture 78 - Custom Error Handlers

Create file in utils folder with name customError.js

```
utils > JS customError.js > ...
1   class CustomError extends Error {
2
3   }
4
5   module.exports = CustomError;
```

Lecture 79 - Big Promise

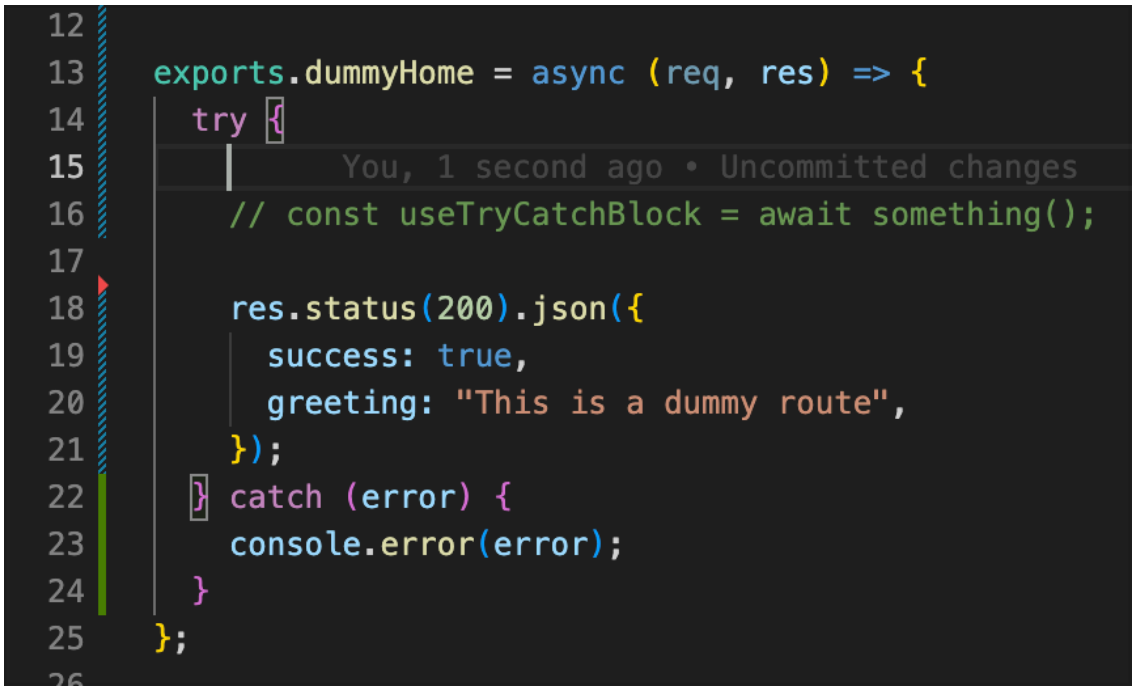
Create file named bigPromise.js in middlewares folder to define it

```
middlewares > JS bigPromise.js > ...
1   // try catch and async - await || use promise || use bigPromise
2
3   module.exports = (func) => (req, next, res) => {
4     Promise.resolve(func(req, next, res)).catch(next);
5   };
6
```

How to use this?

```
controllers > JS homeController.js > ...
      You, now | 1 author (You)
1   const BigPromise = require("../middlewares/bigPromise");
2
3   // This file defines the functionality when 'home is called by a router
4
5   exports.home = BigPromise( async (req, res) => {
6     // const useBigPromise = await something();
7     res.status(200).json({
8       success: true,
9       greeting: "Hello from API",
10    });
11  });
```

What if we don't want to use BigPromise - either use try-catch OR promise

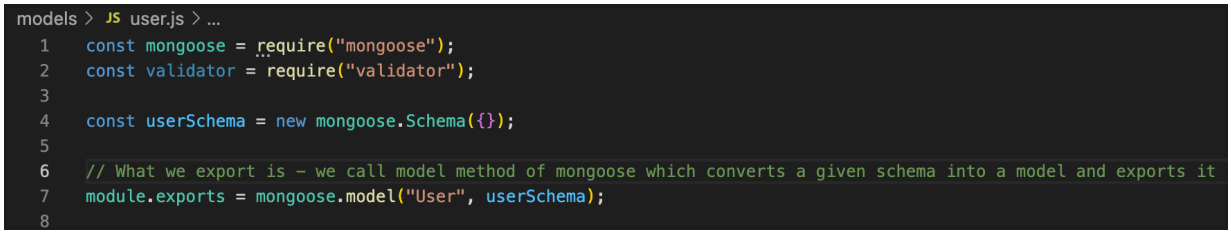


Section 9 - User model and Signup

Dedicated Models, Pre - hooks, controllers, routes, validator and its use cases, all these just for USERS

Lecture 80 - Creating a User Model and Validator

Create user.js file in models



Defining Schema according to Mind Map

```
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, "Please provide a name."],
    maxLength: [40, "Name should be under 40 Characters."],
  },

  email: {
    type: String,
    required: [true, "Please provide an email address"],
    validate: [validator.isEmail, "Please enter email in correct format."],
    unique: true,
  },

  password: {
    type: String,
    required: [true, "Please provide an email address"],
    validate: [validator.isStrongPassword, "Please enter a strong password."],
    // select: false, means whenever we call a user's data password will not be sent to us.
    // We need to explicitly mention it to get password
    select: false,
  },

  photo: {
    id: {
      type: String,
      required: [true, "Please upload your photo."],
    },
    secure_id: {
      type: String,
      required: [true],
    },
  },
```

```
  role: {
    type: String,
    default: "user",
  },

  forgotPasswordToken: String,
  forgotPasswordExpiry: Date,

  createdAt: {
    type: Date,
    // Difference between Date.now and Date.now() is that
    // Date.now - gives time at which user was created
    // Date.now() - given time when schema was declared
    default: Date.now,
  },
});
```

isStrongPassword(str [, options])	Check if a password is strong or not. Allows for custom requirements or scoring rules. If `returnScore` is true, then the function returns an integer score for the password rather than a boolean. Default options: `{ minLength: 8, minLowercase: 1, minUppercase: 1, minNumbers: 1, minSymbols: 1, returnScore: false, pointsPerUnique: 1, pointsPerRepeat: 0.5, pointsForContainingLower: 10, pointsForContainingUpper: 10, pointsForContainingNumber: 10, pointsForContainingSymbol: 10 }`

Lecture 81 - Password Encryption and Mongoose Prototype

Before what we were doing is that bring our bcryptjs library in app.js so that while creating a users on giving required field it encrypts the password

Now what we want to do is to declare encrypt method in model itself
Problems:

1. Every time we edit some field in schema of a user, the password will get encrypted again and again which we do not want

For this there is a method called this.isModified("password") which helps to only encrypt password when user asks to modify it

Therefore, in ./models/user.js

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

Just before exporting the model, add a pre hook that needs to be executed before saving a user in database

```
55
56  userSchema.pre("save", async function (next) {
57    if (!this.isModified("password")) {
58      return next();
59    }
60    this.password = await bcrypt.hash(this.password, 10);
61  });
62
63  // What we export is - we call model method of mongoose which converts a given schema into a model a
64  module.exports = mongoose.model("User", userSchema);
65
```

Lecture 82 - Validating the Password

Validate (check whether the password sent by user while logging in) the password with passed on user password

Just like PRE we are going to use “methods” to define any method on userSchema we want

```
63
64  // Validate (check whether the password send by user while logging in) the password with passed on user password
65  // Just like PRE we have Methods to define any we want
66  userSchema.methods.isValidatedPassword = async function (userSendPassword) {
67    return await bcrypt.compare(userSendPassword, this.password);
68  };
```

Lecture 83 - Creating JWT Tokens

Firstly, define the secret variables in .env file

```
JWT_SECRET=thisismyjwttopecret
JWT_EXPIRY=3d
```

Add a method named as getJwtToken in ./models/user.js

```
4   const jwt = require("jsonwebtoken");
```

```
71  // Create and return JWT Token
72  userSchema.methods.getJwtToken = async function () {
73    return await jwt.sign({ id: this._id }, process.env.JWT_SECRET, {
74      expiresIn: process.env.JWT_EXPIRY,
75    });
76  };
```

Lecture 84 - Forgot Password and Crypto Hashing

For forgot Password functionality to work we need to generate a random string to be saved in model and verify it by the user to change the password so for that we are using crypto package that comes by default in js library

Package name: crypto

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

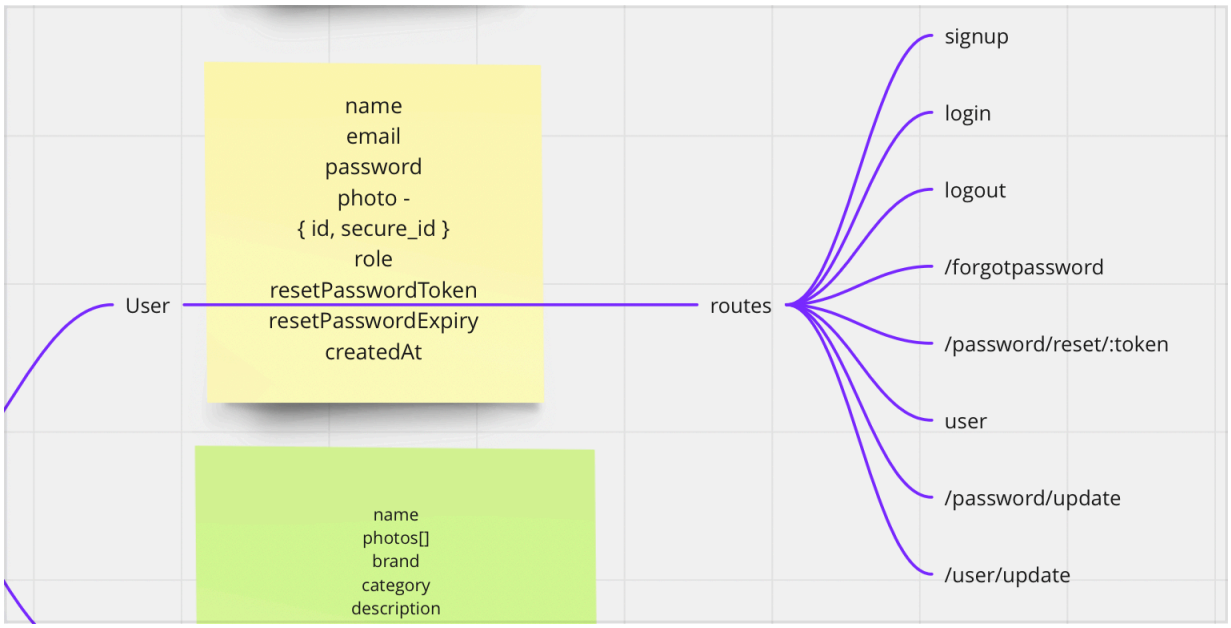
Again going to define a method based on this library to generate token

```
79  userSchema.methods.getForgotPasswordToken = function () {
80    // generate a long and random string
81    const forgotToken = crypto.randomByte(20).toString("hex");
82
83    // this.forgotPasswordToken = forgotToken;    VALID BUT FOR PROTECT WE USE HASHING
84    this.forgotPasswordToken = crypto
85      .createHash("sha256")
86      .update(forgotToken)
87      .digest("hex");
88
89    // NOTE: we are storing this cryptographic hash in database but are sending forgotToken to user
90    // so it is important to get the forgetToken from user while verification and again convert it using
91    // exact same hashing algorithm to compare and verify
92
93    this.forgotPasswordExpiry =
94      Date.now() + process.env.FORGOT_PASSWORD_EXPIRY_TIME;
95
96    return forgotToken;      You, 1 second ago • Uncommitted changes
97  };
```

NOTE: we are storing this cryptographic hash in database but are sending forgotToken to user so it is important to get the forgetToken from user while verification and again convert it using exact same hashing algorithm to compare and verify

Lecture 85 - User Routes and Postman

Each model will be following up some routes. That is when you visit these routes, this model will be controlling these routes OR are affected by this route



We are going to declare all these routes

Firstly, create ./controllers/userController.js
After that, first thing to do is to import the model.

In app.js

```
33  // Import all routes
34  const home = require("./routes/home");
35  const user = require("./routes/user");
36
37  // Router Middleware - REQUIRED
38  app.use("/api/v1", home);
39  app.use("/api/v1", user);      You, 3 hou
40
```

In ./routes/user.js
NOTE - It's a POST request. Just for testing purposes we have written get here

```
routes > JS user.js > ...
1  const express = require("express");
2  const router = express.Router();
3
4  const { signup } = require("../controllers/userController");
5
6  router.route("/signup").get(signup);
7
8  module.exports = router;
9
```

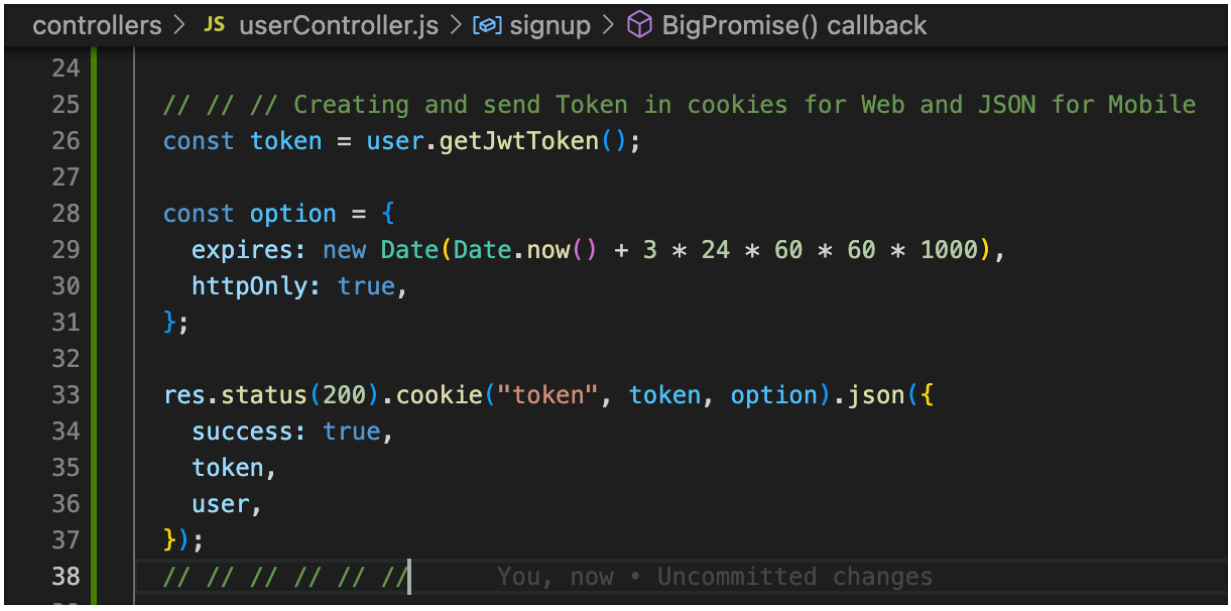
In ./controllers/userController.js

```
controllers > JS userController.js > signup > BigPromise() callback
      You, 3 hours ago | 1 author (You)
1  //  Import Model
2  const User = require("../models/user");
3
4  const BigPromise = require("../middlewares/bigPromise");
5
6  exports.signup = BigPromise(async (req, res, next) => {
7    res.send("Signup Request");      You, 3 hours ago • Uncom
8  });
9
```

Setup Postman to check all things are working or not

Lecture 86 - Signup a User and Cookies

How to create token and assign it to cookie for web OR to JSON for mobile?



As this token generation and passing to cookie will be repeated again and again for different things like login, and other.

It's a great idea to put this code in utils and define a function based on this so that

just by passing required parameters like ‘user’ and ‘res’ this lines of code could be reused again and again.

CONVERT THIS INTO A METHOD IN UTILS FOR REUSABILITY

utils > JS cookieToken.js > ...

```
const cookieToken = (user, res) => {
  const token = user.getJwtToken();

  const option = {
    expires: new Date(Date.now() + process.env.COOKIE_TIME),
    httpOnly: true,
  };

  res.status(200).cookie("token", token, option).json({
    success: true,
    token,
    user,
  });
};

module.exports = cookieToken;

```

```
41    cookieToken(user, res);
```

Lecture 87 - Database Connection

First declare DB_URL link in .env

```
10  DB_URL=mongodb+srv://yash:<password>@atlascluster.wnh9dnl.mongodb.net/?retryWrites=true&w=majority
```

Then create database.js file in config folder

config > JS database.js > ...

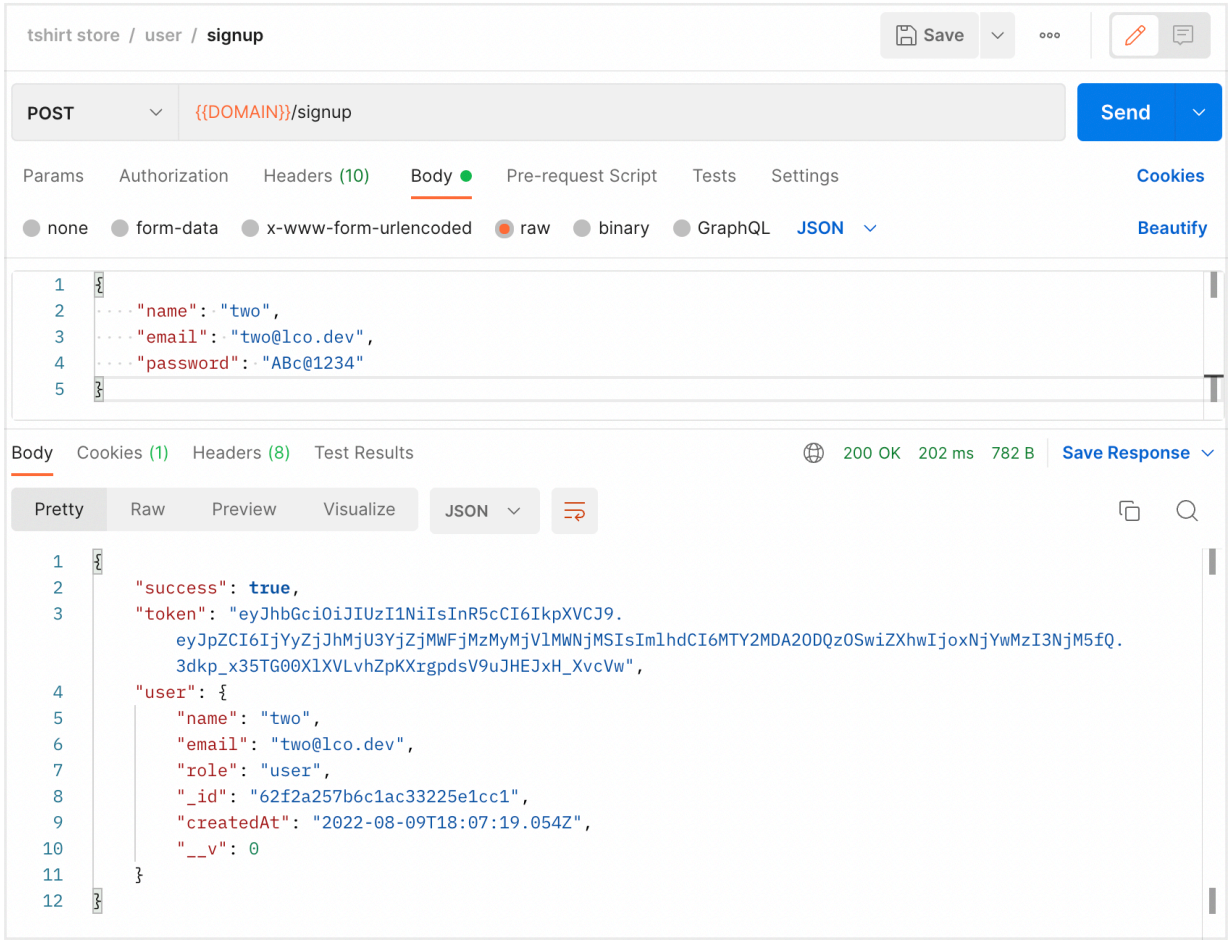
```
1   const mongoose = require("mongoose");
2
3   const connectWithDB = () => {
4     mongoose
5       .connect(process.env.DB_URL, {
6         useNewUrlParser: true,
7         useUnifiedTopology: true,
8       })
9       .then(console.log("Database Connected Successfully"))
10      .catch((error) => {
11        console.log("COULD NOT CONNECT TO DATABASE");
12        console.log(error);
13        process.exit(1);
14      });
15  };
16
17  module.exports = connectWithDB;
18
```

Lecture 88 - Testing Signup Route with Postman

NOTE: use ‘await’ before creating User

Make sure to use this code before sending res from ./utils/cookieToken.js

```
9   // For not sending encrypted password as JSON output
10  user.password = undefined;
```



Lecture 89 - Handling Image Upload

Add cloudinary variables in .env file

```
CLOUDINARY_NAME=
CLOUDINARY_API_KEY=
CLOUDINARY_API_SECRET=
```

In index.js, connect with cloudinary

```
// Connection with Cloudinary
const cloudinary = require("cloudinary");
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});
```

MOST IMPORTANT : In app.js,

```
app.use(
  fileUpload({
    useTempFiles: true,
    tempFileDir: "/tmp",
  })
);
```

In userController.js

```
// For Uploading Files, CLOUDINARY
const fileUpload = require("express-fileUpload");
const cloudinary = require("cloudinary");
```

```
// // // // // Grabing and Uploading Photo to Cloudinary
let result;
if (req.files) {
  let file = req.files.photo;
  result = await cloudinary.v2.UploadStream.upload(file, {
    folder: "users",
    width: 150,
    crop: "scale",
  });
}
```

```
const user = await User.create({
  name,
  email,
  password,
  photo: {
    id: result.public_id,
    secure_id: result.secure_id,
  },
});
```

NOTICE: We are declaring ‘result’ outside if statement and are using it afterwards to fetch public_id and secure_id. This creates a situation in which even if model doesn’t make it compulsory to add photo but our code makes it compulsory.

Lets see solution in next video

Lecture 90 - Testing Photo Upload and User Signup

For a bit of the frontend for submitting images, create views folder for sigupformtest.ejs

In app.js

```
31  // EJS
32  app.set("view engine", "ejs");
```

```
49  // For testing only
50  app.get("/signuptest", (req, res) => {
51    res.render("signuptest");
52  });
```

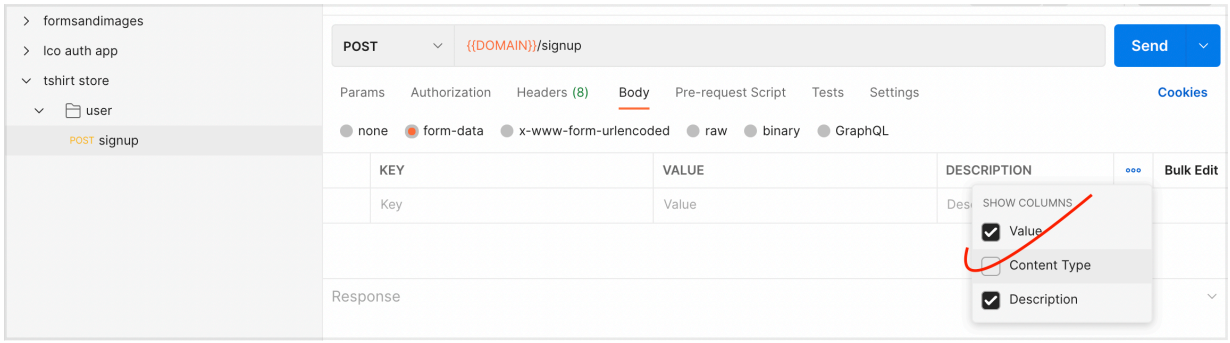
A Classic error when you haven't used the file.tempFilePath for wloudinary to upload images from

```
GET /signuptest 200 1984 - 6.368 ms
/Users/yashpatel/Work/Courses/Pro Backend/tshirtstore/controllers/userController.js:20
    result = await cloudinary.v2.UploadStream.upload(file, {
                                              ^

TypeError: Cannot read properties of undefined (reading 'upload')
    at /Users/yashpatel/Work/Courses/Pro Backend/tshirtstore/controllers/userController.js:20:47
    at /Users/yashpatel/Work/Courses/Pro Backend/tshirtstore/middlewares/bigPromise.js:4:19
```

To correct this: in userController.js

```
18    if (req.files) {
19      let file = req.files.photo;
20      result = await cloudinary.v2.uploader.upload(file.tempFilePath, {
21        folder: "users",
22        width: 150,
23        crop: "scale",
24      });
25    }      You, 3 days ago • Integrating Cloudinary with some errors
```

Lecture 91 - Yes, We Know About Postman Files



Section 10 - User Controller and Routes

Lecture 92 -Login Route and Controller

```
controllers > JS userController.js > login > BigPromise() callback
70  exports.login = BigPromise(async (req, res, next) => {
71    // Grab the email and password
72    const { email, password } = req.body;
73
74    // If any field is empty throw error
75    if (!email || !password) {
76      return next(new Error("Both, email and password are required fields."));
77    }
78
79    // Search in DB
80    const user = await User.findOne({ email }).select("+password");
81
82    // If no Record match throw error
83    if (!user) {
84      return next(new Error("Email and Password does not match or exist."));
85    }
86
87    // Check if password is entered correctly
88    const isPasswordCorrect = user.isValidatedPassword(password);
89
90    // If not Throw error
91    if (!isPasswordCorrect) {
92      return next(new Error("Email and Password does not match or exist."));
93    }
94
95    // If everything goes right create and send token as cookies.      You, 2 m
96    CookieToken(user, res);
97  });
```

PROBLEM: Currently, if anyone log's in then the token I=could be used by anyone to login even if they enter password wrong which should not happen.

SOLUTION: Used console.log to check value of isPasswordCorrect. The problem of logging in everybody was not because of cookie but it was because of not using await in front of checking isValidatePassword

```
87  // Check if password is entered correctly
88  const isPasswordCorrect = await user.isValidatedPassword(password);
```

Lecture 93 - Logout Controller and Route

IMPORTANT - We are working on JWT Tokens which are stateless which means they doesn't really care who you are. As long as anybody is having this token can login

Depends on who is managing token he should delete it from their storage. Like if frontend person is managing then he should delete it from there.

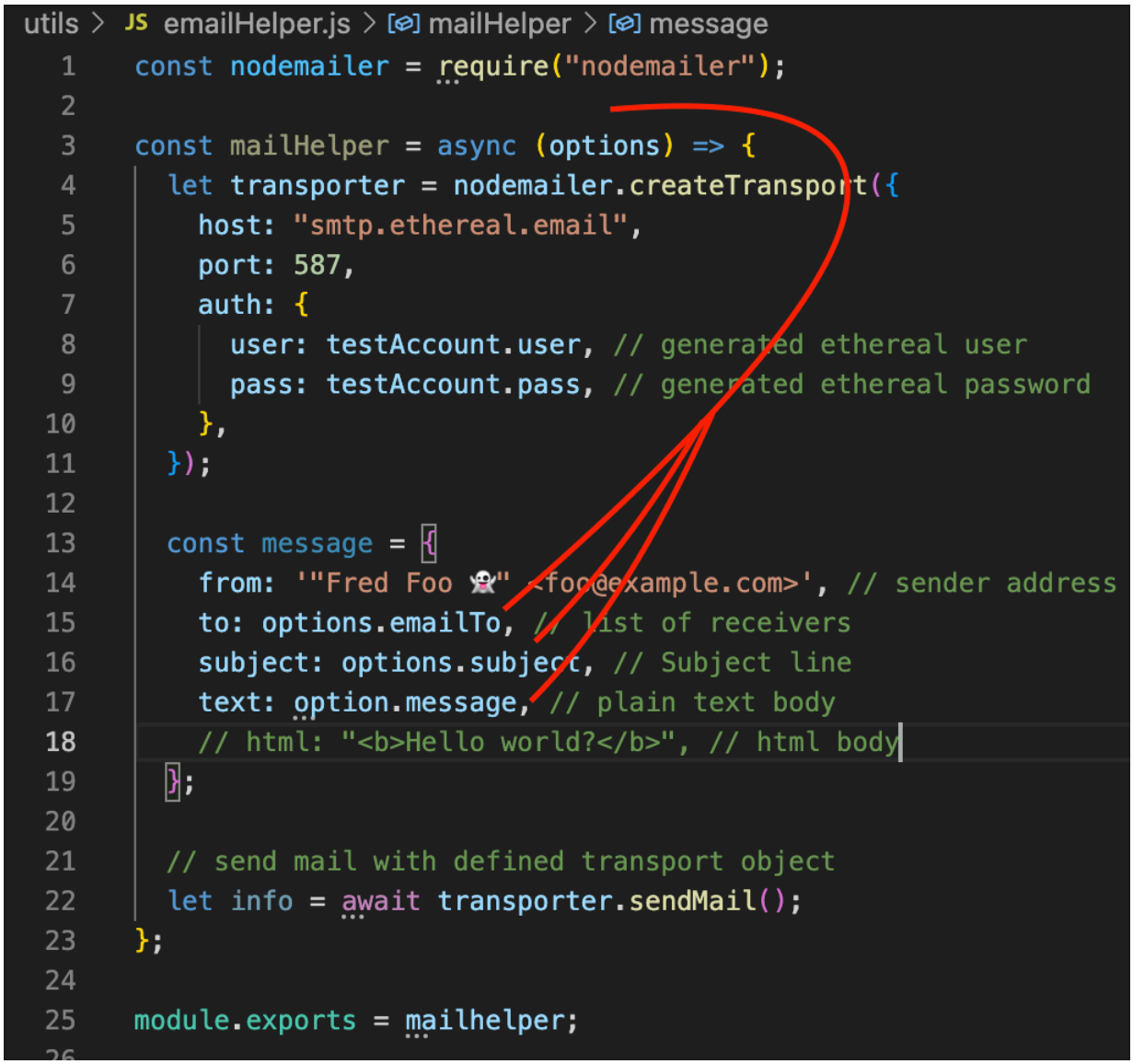
In usercontroller.js

```
100
101   // Set the token value to null and expire it immediately
102   res.cookie("token", null, {
103     expires: new Date(Date.now()),
104     httpOnly: true,
105   });
106
107   res.status(200).json({
108     success: true,
109     message: "Logout successful.",
110   });
111 });
112
```

Lecture 94 - Send Email From Node

Create a file named emailHelper.js in utils folder

Require nodemailer



For testing out emails use MailTrap

From https://mailtrap.io/inboxes/1853861/messages# to .env

```
SMTP_HOST=
SMTP_PORT=
SMTP_USER=
SMTP_PASS=
```

utils > JS emailHelper.js > ...

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

const mailHelper = async (options) => {
  let transporter = nodemailer.createTransport({
    host: process.env.SMTP_HOST,
    port: process.env.SMTP_PORT,
    auth: {
      user: process.env.SMTP_USER, // generated ethereal user
      pass: process.env.SMTP_PASS, // generated ethereal password
    },
  });

  const message = {
    from: 'yash@lco.dev', // sender address
    to: options.emailTo, // list of receivers
    subject: options.subject, // Subject line
    text: option.message, // plain text body
    // html: "<b>Hello world?</b>", // html body
  };

  // send mail with defined transport object
  await transporter.sendMail(message);
};

module.exports = mailhelper;
```

Lecture 95 - Forgot Password Controller

In userController.js

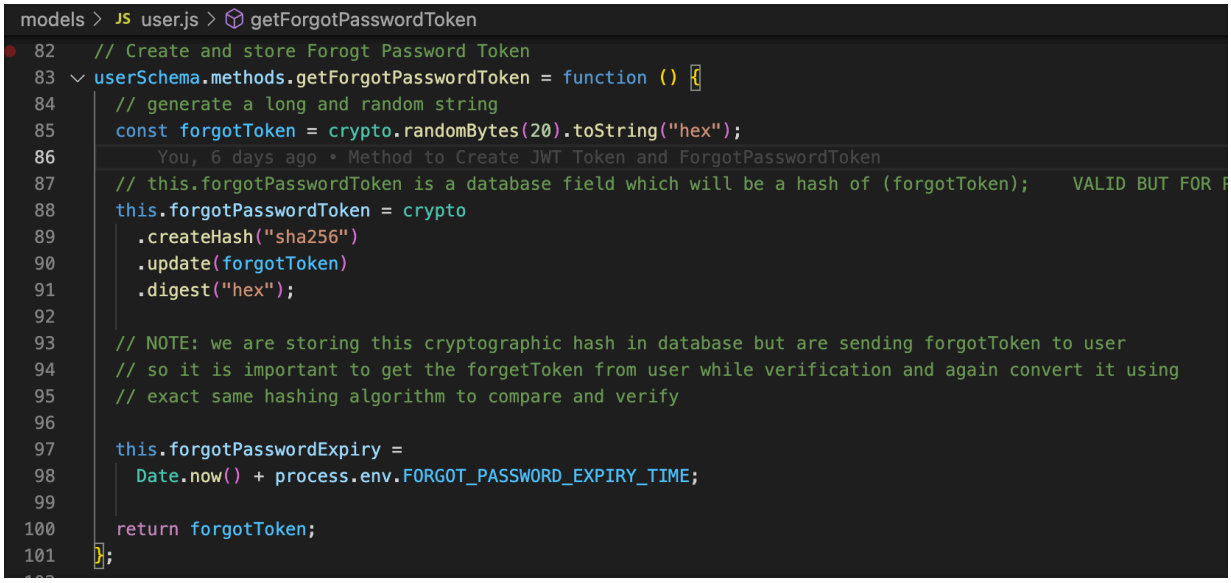
controllers > JS userController.js > forgotPassword > BigPromise() callback

```
exports.forgotPassword = BigPromise(async (req, res, next) => {
  // Grab email from to body
  const { email } = req.body;

  // Find it in database
  const user = await User.findOne({ email });

  // If email is not found in database
  if (!user) {
    return next(new Error("Email not found as registered.", 400));
  }
```

Generate forgotToken using method defined in ./models/user.js

```
const forgotToken = user.getForgotPasswordToken();
```

This was defined in models just look at it



After generating forgotToken we get 3 values as per our defined method in ./models/user.js

1. forgotToken
2. forgotPasswordToken
3. forgotPasswordExpiry

These all needs to be SAVED in database so await is used

IMPORTANT: validateBeforeSave is used because name, email, password are set to be required before save so just to ignore that we need to use this flag

A Url is generated according to password generation and a message is constructed based on that to send user to reset password

Using Try Catch is IMPORTANT as sending email may go wrong

IMPORTANT: If email is not sent, we need to set all generated field to undefined again

```
147     try {
148       await mailHelper({
149         emailTo: user.email,
150         subject: "LCO TShirt Store Password Reset",
151         message,
152       });
153
154       res.status(200).json({
155         success: true,
156         message: "Email sent Successfully.",
157       });
158     } catch (error) {
159       user.forgotPasswordToken = undefined;
160       user.forgotPasswordExpiry = undefined;
161       await user.save({ validateBeforeSave: false });
162
163       return next(
164         new Error(`Couldn't send email due to this error: ${error}, 500`)
165       );
166     }
167   });
168
```

Lecture 96 - Reset Password Controller and Routes

Till now user gets an email with a link to reset the password but actually no routes are set to reset this password

So, lets now do this

controllers > JS userController.js > passwordReset > BigPromise() callback

```
exports.passwordReset = BigPromise(async (req, res, next) => {
  const token = req.params.token;

  // After grabing the token from the Emailed URL we are encrypting it again same as it was stored in database so
  // that we can find the user based on it and verify to let him change the password
  const encryToken = crypto.createHash("sha256").update(token).digest("hex");

  // MongoDB Query is used here so that if the time has not expired for the token then only a user will be returned
  const user = await User.findOne({
    encryToken,
    forgotPasswordExpiry: { $gt: Date.now() },
  });

  // If Token has expired
  if (!user) {
    return next(
      new Error("Either User does not exist or time to reset token has expired")
    );
  }

  // If the password does not match
  if (req.body.password != req.body.confirmPassword) {
    return next(
      new Error("Re-enter the passwords as they do not match with each other.")
    );
  }

  // Updating password in database
  user.password = req.body.password;

  // IMPORTANT: After updating password the tokens saved in database should be removed
  user.forgotPasswordToken = undefined;
  user.forgotPasswordExpiry = undefined;

  // Updating the User data
  await user.save();
```

Once this passwordReset Controller has been defined, we need to make sure that in routes the link is properly set so that token could be fetched.

routes > JS user.js > ...

```
router.route("/password/reset/:token").post(forgotPassword);
```

IMPORTANT: we want forgotPasswordExpire to be current date in milliseconds so it cannot be written as type Date.

models > JS user.js > userSchema

```
  // NOTE: This field stores current Date as milliseconds so it needs to be defined as Number
  forgotPasswordExpiry: {
    type: Number,
    default: new Date().getTime(),
  },
```