# Day 1

**Basics:**

* Like window, there is global object in node called ‘global’.
* \_\_dirname: it give the absolute path of current file without a file name
* \_\_filename: it give the absolute path of current file with the file name at the end.

**Module and require**

* To export something to be used in another file we need to export which furthur get attached to the variable in which that was imported to
  + module.exports = { name, ages };
* we can import the module using
  + const varname=require(path)
* Or for importing ant module we can make use of destructuring functionality of the javascript.

**The File system**

* node provide a inbuilt module for file system called “fs”.
* Const fs=require(“fs”) #importing fs module
* For reading file:
  + Use readFile(path,callback(err,data))

//readFile method is async function

fs.readFile("blogs/blog1.txt", (err, data) => {

  if (err) {

    console.log(err);

  }

  //toString method is used to extract actual data from the buffer

  console.log(data.toString());

});

* For writing a file we make use of writeFile method:
  + writeFile(path,dataToWrite,callback(err))

//if we are writing into the file which does not exist then it will crate a new file and write the data into automatically.

fs.writeFile("blogs/blog1.txt", "hello everyone", (err) => {

  if (err) console.log(err);

  console.log("written succesfully");

});

* **delete a file**

//to delete the file we can use unlink method

if (fs.existsSync("./blogs/deleteme.txt")) {

    fs.unlink("./blogs/deleteme.txt", (err) => {

      if (err) console.log(err);

      else console.log("file deleted");

    });

  }

* **create and delete directory**

//Before creating any folder we check whether that concern folder already exists or not

if (!fs.existsSync("./assets")) {

  //we can create any folder using mkdir method

  //if that directory already exists then it will throw some error

  fs.mkdir("./assets", (err) => {

    if (err) console.log(err);

    else console.log("folder created");

  });

} else {

  //To delete any directory, we can use rmdir method

  fs.rmdir("./assets", (err) => {

    if (err) console.log(err);

    else console.log("Folder deleted");

  });

}

* **Streams**
  + **Using streams we can pass the data like a small buffers.**

//creating a readstream

const readstream = fs.createReadStream("./blogs/blog2.txt", {

  encoding: "utf8",

});

//creating a write stream

//this will create new file if that file does not exist.

const writestream = fs.createWriteStream("./blogs/blogs3.txt");

//reading data using read stream

//on is event listener which get triggered everytime new cgunks of data will arrived

readstream.on("data", (chunk) => {

  console.log("------New Chunks---------");

  console.log(chunk);

  //write file using write stream

  writestream.write(chunk);

});

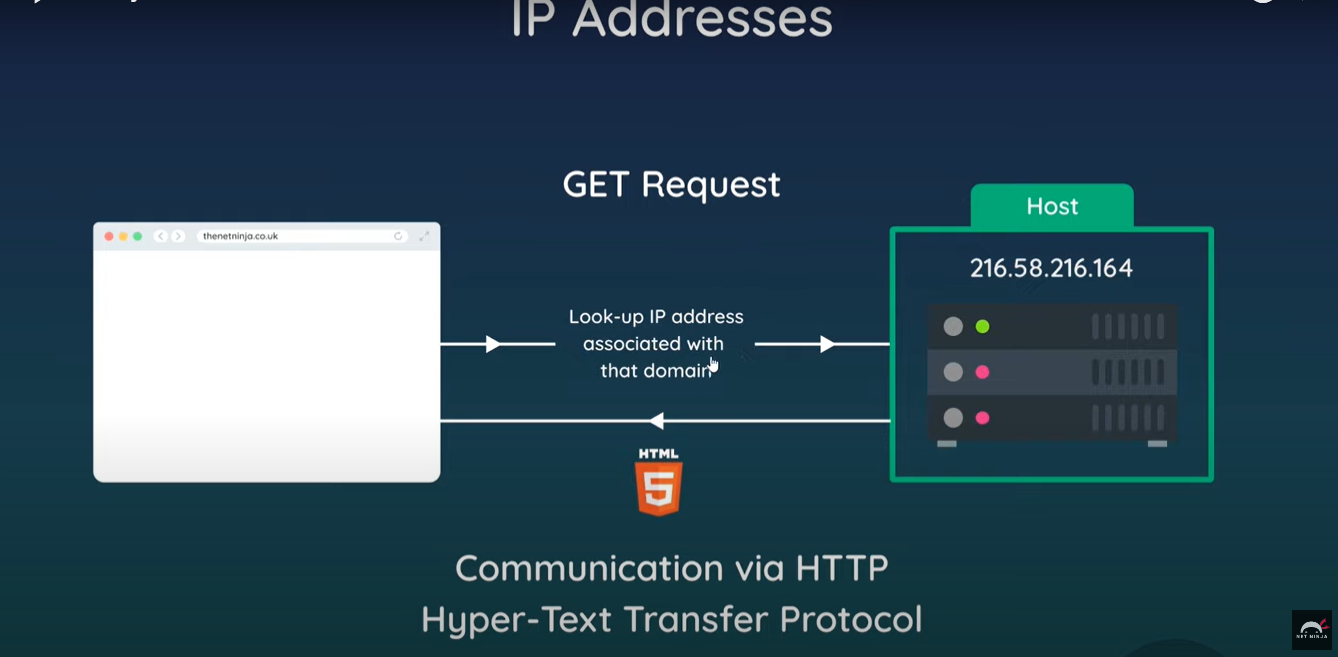
//Piping

//we can copy data from readstream to write stream using pipe method

readstream.pipe(writestream);

**HTTP module**

* How website works?



* **Creating and starting a server**

const http = require("http");

//Creating Server

//createServer method is used create server and takes two arguments reques and response

//request: gives the information about the URL from where the request has been made

//respose: what reponse to give back to that URL

const server = http.createServer((req, res) => {

  console.log("request made");

});

//To start the server we have to start listening to the request,

//we can do that by using listen method come with the server object

//listen method takes in 3 arguments portnumber,hostname,callbackfunction

//callback function get fired when we first stred listening to the request

server.listen(3000, "localhost", () => {

  console.log("litening to the localhost on port 3000");

});

* **Sending response back to front end**

const server = http.createServer((req, res) => {

  //sending back the res

  //1. set the header of the reposnse which tell the browser which type of data is being sent

  res.setHeader("Content-Type", "text/plain");

  //2.write the reposne object with our content

  res.write("Hello from server");

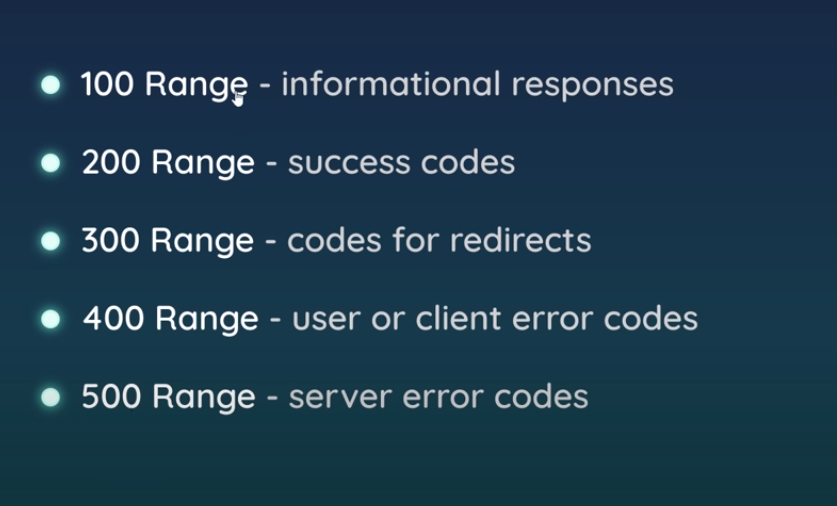
  //3. ending the message to send the data

  res.end();

});

# **Day 2**

* **Status codes**



* + We can set the status code to the response object by using
    - res.statusCode=statuscode
* Routing and redirecting using Nodejs

//For routing

  let path = "./views/";

  switch (req.url) {

    case "/":

      path += "index.html";

      //setting status codes to let brwoser know about the status of the request

      res.statusCode = 200;

      break;

    case "/about":

      path += "about.html";

      res.statusCode = 200;

      break;

    //this case is used to redirect the user to the new location

    case "/about-me":

      res.statusCode = 301; //status code is to let browser know that content has shifted to another location

      res.setHeader("Location", "/about"); //redirecting the user

      res.end();

      break;

    default:

      path += "404.html";

      res.statusCode = 400;

  }

  fs.readFile(path, (err, data) => {

    if (err) {

      console.log(err);

      res.end();

    }

    res.write(data);

    res.end();

  });

**Expressjs**

* Express is nodejs framework that helps us to manage routes, response and different types of requests like post and get request.
* Starting code

const express = require("express");

//initialize the express app

//using this variable we can use different methods of express

const app = express();

//listening to the server

const server = app.listen(3000, "localhost");

//listening to the get request

app.get("/", (req, res) => {

  //send method is used to send the response to the frontend

  //this method automatically set the socntent type and also

  //we dont have to set the status code most of the time

  res.send("hello");

});

* Sending file and doing redirect using express

//listening to the get request

app.get("/", (req, res) => {

  //send method is used to send the response to the frontend

  //this method automatically set the socntent type and also

  //we dont have to set the status code most of the time

  //res.send("hello");

  res.sendFile("./views/index.html", { root: \_\_dirname });

});

app.get("/about", (req, res) => {

  res.sendFile("./views/about.html", { root: \_\_dirname });

});

//redirecting

app.get("/about-us", (req, res) => {

  res.redirect("/about");

});

//404 page

//use method is used to create a middleware

//it will get fired for every routes if control reaches at this point

//this should be the very last in the file

app.use((req, res) => {

  res.status(404);

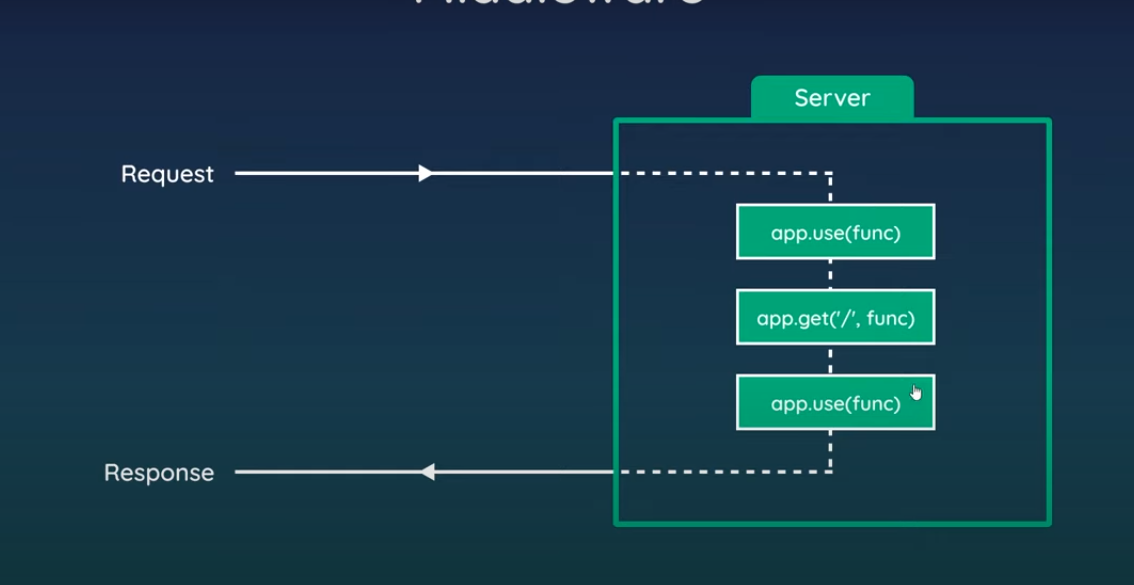
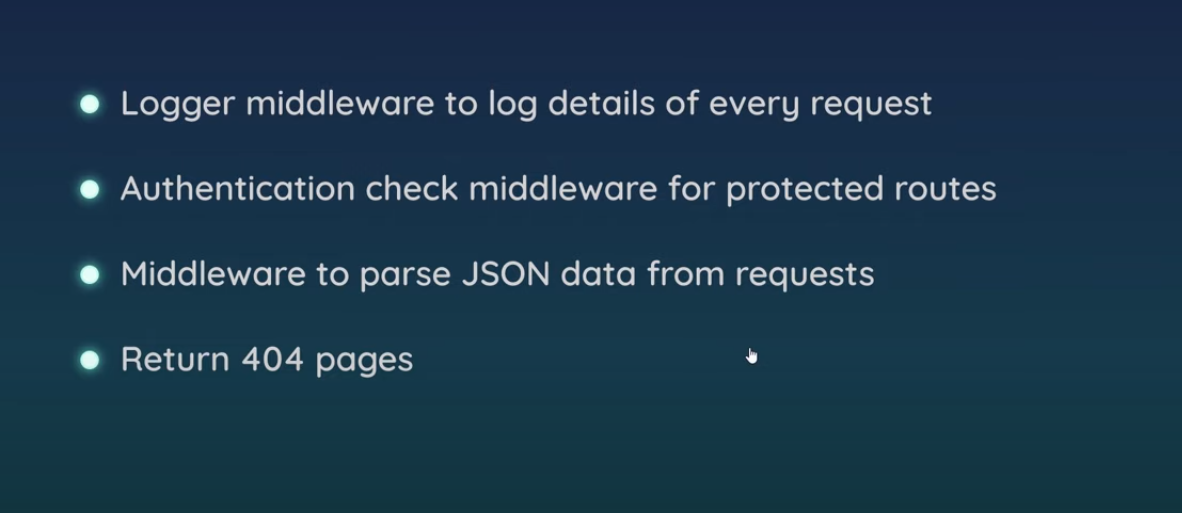
  res.sendFile("./views/404.html", { root: \_\_dirname });

});

# Day 3

* View engine
  + View engine let us insert our dynamic data into the html file.
* We cannot go I deep into it because we can use reactjs for that matter

**Middleware**

* Middleware is a code which runs (on a server) between getting a request and sending a request.
* 
* Here, fun is middleware.
* Middleware with use method runs every time but with get or post method runs only once when that particular route get called.
* Middleware can be used for different purpose such as
* 
* When we use any middleware , then to let browser know what to do next we make use of the next() methods of the express.
* Next() is used when we are not sending any response to the frontend when we use the middleware.

//defining the middleware

app.use((req, res, next) => {

  console.log("Requeste made");

  console.log(req.hostname);

  next();

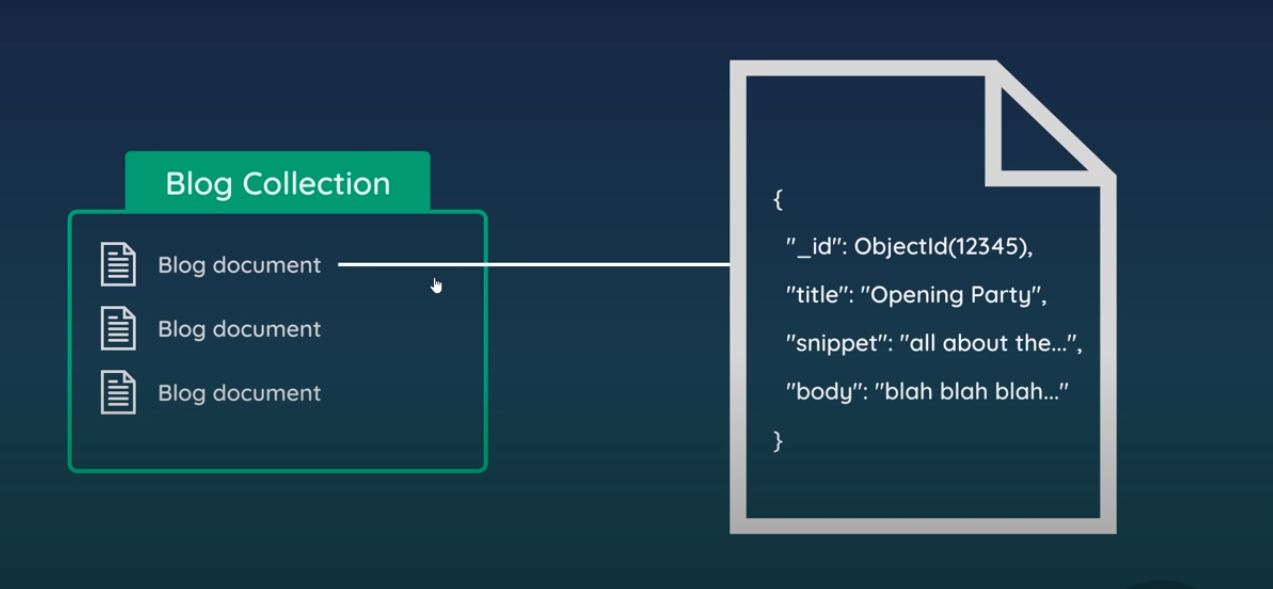
});

* We can use third party middleware for session, validation check, cookies and many more things.
* While we are using server, express does not allow static files like css to render in the front end.
* To let that reference in the front end, we have use put everything in a public folder and refence that folder as a public using express.static(‘public’).

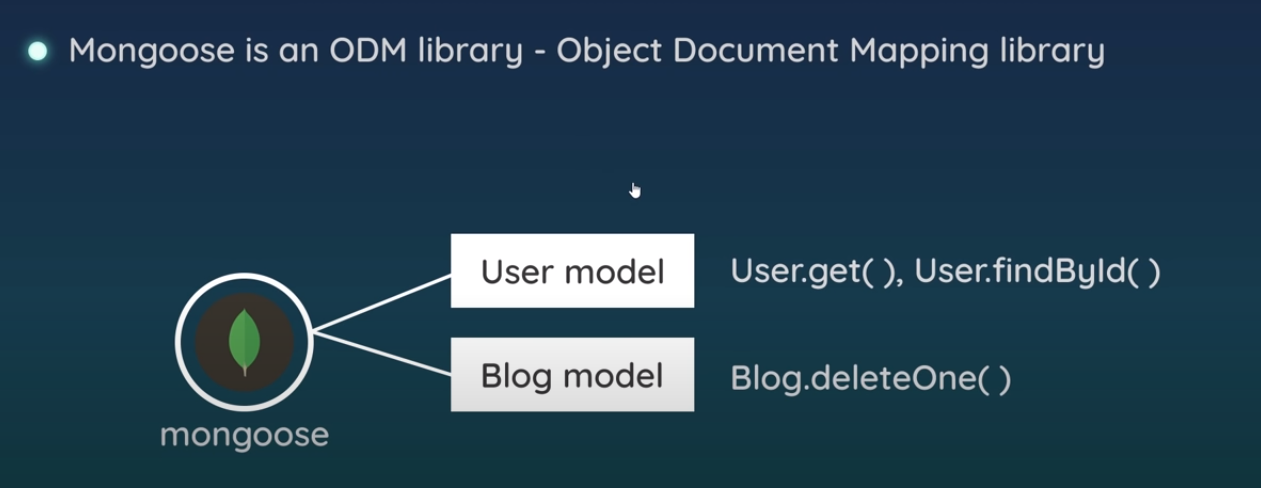
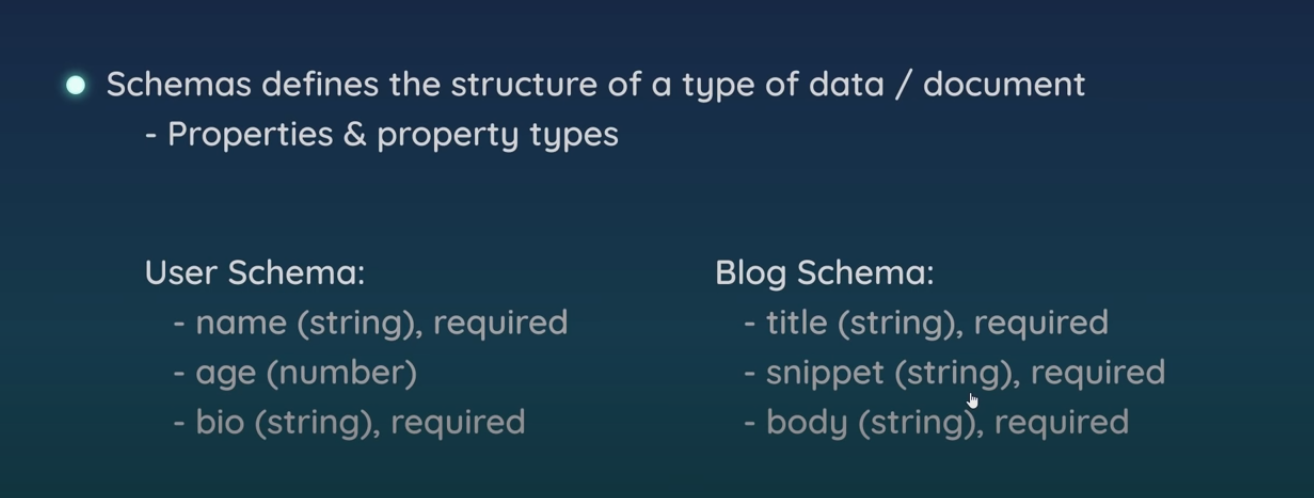
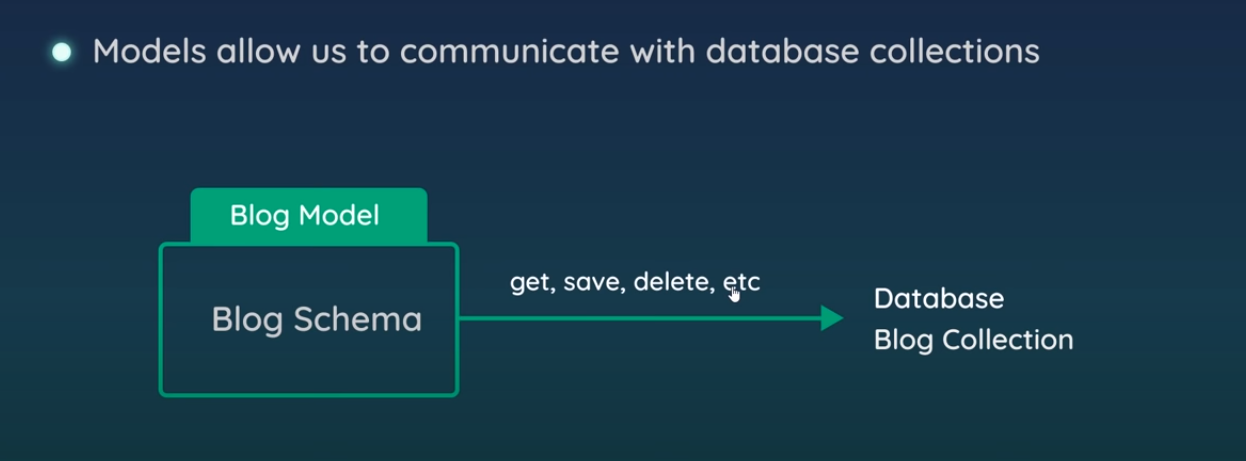
//Making static file accessible to the frontend

app.use(express.static("public"));

**MongoDB**

* MongoDB is a open source noSQL database.
* There is a collection and documents in the mongoDB. This is how they correlate to each other.
* 
* We can create cluster , then database and collection in to that.(refer latest video to that).

**Connecting MongoDB using Mongoose**

* Mongoose is a ODM(object document model) library which helps us create model and that model provide different function which can be used for the fetch, update, delete and many other operations.
* 
* To work with mongoose , we first create a schema which define the structure of document.
* 
* Then, we create a model which allow us to communicate with the database.
* 
* **Steps to connect to the mongoDB**
  + **1. Copy the connecting string from the user access. Make sure to change the user and specify the password int the URI and also mention the database name.**

**e.g.**

mongodb+srv://**username:password**@cluster0.u2554.mongodb.net/**dbname**?retryWrites=true&w=majority

const mogoose = require("mongoose");

//URI for mongoDB URI

const dbURI =

  "mongodb+srv://sanket:fedfesf@cluster0.u2554.mongodb.net/dbname?retryWrites=true&w=majority";

* + **2. Install and require mongoose in the app file**
  + **3. Use the variable(mongoose) use connect method which takes in two arguments. Connect method is async function so it returns a promise.**

//connecting to the mongodb

mogoose

//connect function is async function.

  .connect(dbURI, { useNewUrlParser: true, useUnifiedTopology: true })

  .then((result) => {

    console.log("Connected Sucessfully");

  })

  .catch((err) => {

    console.log(err);

  });

* + **4. Run the file and if “connected succesfull message is displayed then we connected with databse successfully.**

**Creating Schema**

* To create a schema we use Schema construct
* E.g.

const mongoose = require("mongoose");

//creating a construct which can be use to build a schema.

const schema = mongoose.Schema;

//using schema construct to build a schema

//it takes two argument first argument define the schema and second argument is option

//we can set timestamp as true to monitor the created and last updat of the record

const blogSchema = new schema({

  title: {

    type: String,

    required: true,

  },

  snippet: {

    type: String,

    required: true,

  },

  body: {

    type: String,

    required: true,

  },

},{timestamps:true});

**Creating a model**

* To create a model we can make use of model method of the mongoose.

//model method takes in two parameter i.e. collection name in singular form and schema which the collection based on

//it is important to pass the first argument in the singular form because mongoose will pluralize it and then look in

//the database( in this case mongoose will look for blogs collection in database)

const Blog=mongoose.model('Blog',blogSchema);

# Day 4

**Getting and saving data**

* Saving document to the database

//using the model construct which was exported , we can create new instance of the blog and

  //using those instance we can perform different operation

  const blog = new Blog({

    title: "third blog",

    snippet: "Snipper look like this",

    body: "Body of the blog",

  });

  //save method is used to save the data to the database

  //save is async function with return a promise and promise contain a result which we can tape on using the then()

  blog

    .save()

    .then((result) => {

      res.send(result);

    })

    .catch((err) => {

      console.log(err);

    });

* Getting data from database

app.get("/all-blogs", (req, res) => {

  //getting all the document usinf find method

  Blog.find()

    .then((result) => {

      res.send(result);

    })

    .catch((err) => {

      console.log(err);

    });

});

app.get("/single-blog", (req, res) => {

  //getting single document using findbyid method

  Blog.findById("649fa1826d092d9661da6679")

    .then((result) => {

      res.send(result);

    })

    .catch((err) => {

      comnsole.err;

    });

});

* We can also use sort method to sort the data based on the key we want.

**Get, Post and Delete Requests**

* 

* It is completely ok to send different types of request on the same routes.
* Get request is used to get some resource from the database.

**Making a post request**

* We can make post request by either using from itself or using some third party library such as axios.
* Lets see how we can make a post request using form itself
  + 1. Configuring the form.

 <!-- action="route where we want to send the request on" -->

 <!-- method type of request we are sending -->

 <!-- make sure to specify the name attribute of each input field this name will be used to get value of the each field -->

      <form action="/blogs" method="post">

        <label for="title">Blog title:</label>

        <input type="text" id="title" name="title" required />

        <label for="snippet">Blog snippet:</label>

        <input type="text" id="snippet" name="snippet" required />

        <label for="body">Blog body:</label>

        <textarea id="body" name="body" required></textarea>

        <button>Submit</button>

      </form>

* + 2. Making a post route
    - 2.1. make user to use middleware which is used to convert the data from form into object and attach it to the body property of the request.

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

* + - 2.2 save the posted data to the database.

app.post("/blogs", (req, res) => {

  //req.body is object which satisfy the blog model so we directly passed it else we can also do like

  //const blog = new Blog({

  //   title: req.body.title,

  //   snippet: req.body.snippet,

  //   body: req.body.body,

  // });

  const blog = new Blog(req.body);

  blog

    .save()

    .then((result) => {

     console.log("Seccesfully added")

     res.redirect("/");

    })

    .catch((err) => {

      console.log(err);

    });

});

* Route parameters : variable parts of the URL that could change.
  + E.g: localhost:3000/blogs/:id
    - Here, id is the route parameter.

app.get("/blogs/:id", (req, res) => {

  const id = req.params.id;

  Blog.findById(id)

    .then((result) => {

      res.render("details", { blog: result, title: "Blog Details" });

    })

    .catch((err) => {

      console.log(err);

    });

});

**Making delete request**

* Step1. Make AJAX request from frontend

 const trashcan = document.querySelector("a.delete");

      //adding eventlistener

      trashcan.addEventListener("click", (e) => {

        //endpoint is route where we want to send the delete request

        const endpoint = `/blogs/${trashcan.dataset.doc}`;

        //here we are making a AJAX request

        //with AJAX request we always look for some response not any redirect

        fetch(endpoint, {

          method: "DELETE",

        })

          //converting json into js object

          .then((response) => response.json())

          //redirecting

          .then((data) => (window.location.href = data.redirect))

          .catch((err) => console.log(err));

      });

* Step2. Handle the delete request

app.delete("/blogs/:id", (req, res) => {

  const id = req.params.id;

  Blog.findByIdAndDelete(id)

    .then(() => {

      //sending the jason data to the frontend

      res.json({ redirect: "/blogs" });

    })

    .catch((err) => {

      console.log(err);

    });

});

# Day 5

**Express Router and MVC**

* Express provide a router method which help us to organise the code in much efficient way.
* We can user the in following steps
  + Step1. Define a router, which will be used to bind all the routes

//using Router method which will attach all the routes to this object

const router = express.Router();

* + Step2. User router instead of app

router.get("/blogs/create", (req, res) => {

  res.render("create", { title: "Create a new blog" });

});

* + Step3. Import the router file

//importing the route file to use it here

const blogRoutes = require("./routes/blogRoutes");

* + Step4. Create a middleware to use that routes

//creating a middleware to use the route

app.use(blogRoutes);