Date 30.8.2023
Course Mongodb
Notes By Azan imtiaz

Mongodb is a papular open source, document oriented NoSQL (non relational) database management system. It was desioned to handle large amount of data and provide high scalability for modern applications. Mongodb stores data in a JSON-like document with dynamic schemas, which allows for easy changes and updates to the data structure without requiring a predefined schema like traditional relational databases.

Some key features + concepts associated with mongodb.

(1) Document

Data in mongodb is stored as documents which are collection of key value pairs. Documents are analogous to rows in relational databases but are stored in more flexible and hierarchical format.

(2) Collection

Documents are grouped into collections which are analogous to tables in relational databases. Each document with in a collection can have a different structure which provide flexibility in data modeling.

③ Schema-less
Unlike relational databases, Mongodb does not required a fixed schema. This means that different documents in the same collection can have different fields and structure

④ JSON-like format
Data in Mongodb is stored in BSON (Binary Json) format which is a binary encoded serialization of JSON like documents. This format allows for efficient storage and retrieval of data

⑤ Query language
Mongodb provides a powerful query language for retrieving and manipulating data Queries can be performed using the mongodb Query language(MQL). which is similar to SQL for relational databases

⑥ Indexes
Indexes can be created on specific fields in document to improve query performance; indexes allow mongodb to quickly locate and retrieve data based on the specified criteria

⑦ Replication
Mongodb supports data replication which involves maintaining multiple copies of data across multiple servers or clusters

This helps to handle large amount of data and high levels of traffic by distributing the load.

⑨ Aggregation Framework

Mongodb aggregation framework allows for complex data transformations and calculations to be performed directly with in the database.

⑩ Geospatial Data

Mongodb has built in support for geospatial data, enabling the storage and querying of location based information.

⑪ ACID Transaction

Starting from version 4.0, mongodb supports multi document ACID (Atomicity, consistency, Isolation, Durability) transactions allowing for more complex data operations while maintaining data integrity.

⑫ Community and Enterprise Editions

Mongodb is available in both community and enterprise editions. The community edition is open source and free to use, while the enterprise edition offers additional features and support option for businesses.

Does mongodb uses BSON or JSON?

It stored data in BSON format both internally and over the network but that does not mean, you can't think MongoDB as a JSON database. Anything you can represent in JSON can be natively stored in Mongodb and retrieved just as easy as JSON.
So BSON is just binary JSON mean binary representation of JSON a superset of JSON with some more data types (most importantly binary byte array).

SQL and Mongodb example

In SQL → Table, column

id	Username	email	mobile	symp	message
1	Azm	azm@43	0317	tired	I'm fine

In MongoDB → Collection

```
[
  {
    "id": ObjectId("190279043948"),
    "username": "Azm",
    "email": "azm@",
    "mobile": 0317
    "symp": ['cold', 'fever'],
    "message": "I'm Awesome",
    "report": false
  },
  {
    // data
  }
]
```

key/keys

document

Each column of your table would be the keys of your document Because data stored in key: Value pair in Mongodb Documents

Mongodb flexible esliye hai yky usr hmary SQL me ek table create kry kisi attributes ky ith to wo attributes har row py apply hoty. But in mongodb ek collection mn different documents mn zyada kam ditt keys ho skti hai Thats the beauty of mongodB.

Lacture 2)
Complete installation of mongodB
Setting Environment path of mongodb

Lacture 3)
Create Database, Collections and Documents in Mongodb

Open a new terminal window and connect to the mongodb server using the mongodb shell. You can used command mongo.

(1) Create database
use dbName
//It will do two things if database with this name does not exist then create it. And if it exist then make it active

There database already (implicitly) exists. admin, config, local. we can see them by using show dbs command.

Remember jab be koi new database create karty hain we directly show dbs ey show nai hoti until we add at least one collection to it

(2) Create a Collection

Collections are created when you insert documents into them You dont need to explicitly create collections for example if you want to create a collection named "user" you can insert a document into it.

```
db.user.insertOne ( {
    "name": "Azan imtiaz",
    "age": 301
    "email": "Azan@43gmail.com"
    } )
```

This will create the user collection if it does not exist and insert the provided document. If the this collection already exist this document is also added with the previous ones

⑤ Insert Documents

To insert documents into a collection we can use the 'insertOne' or 'insertMany' methods

```
db.collectionName.insertOne({
    "field1": "value1",
    "field2": "value2"
})
db.collectionName.insertMany([{
    "field1": "value1",
    "field2": "value2"
},
{
    // Another document
}
])                    // format ([{},{},..])
```

Remember that mongodb is flexible and schema less which means that different documents in the same collection can have different fields

One more thing agr hum Id add na be karan behind the scene khud sy eke objectId generate ho jati hai we can also add boolean data in document like

"active": true

↳ ya jo values hain agr shell use kar sy ho to "keys" sy na be dalo koi forcy nai parta khud sy generate ho jaty hain. But in stack

case compulsory

Check current active database name
db

See Collections inside current database
show collections
This will display a list of all the collections present in the currently selected database

fetch documents inside collection
We can use find method

optional ← db.collectionName.find(query, projection) → optional

collectionName Name of collection

query This is an optional parameter that specifies the criteria for selecting documents. If not provided it fetches all documents in the collection.

projection This is an optional parameter that specifies which fields to include or exclude in the returned documents. If not provided, all field will be included

Lacture 4)

MonGoDB CRUD Operation

Create or insert Documents into collection

One document insertion format

({ })

Many document insertion format

([{ }, { }, { } . . .])

lacture 5)

MongodB Tutorial in Hindi

Read or Queries the Documents into Collection

To fetch or retrieve documents

db.collection.find (query, projection)

(1) find all the result/documents of given collection

db.collectionName.find()

(2) Show the result in pretty formal

db.collectionName.find().pretty()

(3) Get only mongodb data as a output with only name field. Consider one document exist which has field name with value mongodb. our task is to point just that document inside collection which contain name field with mongodb value

query

db.collectionName.find({ name: "mongodb" });

it will print that document with all fields.

[redacted]

(4) Get only MongoDB data as a output with only name field

_Query _Projecting

db.collectionName.find({name:"mongodB"}, {name:1})

• 1 showing true mean just name dikhy ga aur name:0 likhy to its mean false mean bawi sb dikhy gy name ko chor ky

But escy sth name ky id be point hogi (9[redacted]).

(5) Do something Id be na dikhy just name field dikhy jo mongodb ky krvado.

db.collectionName.find({name:"mongodb"}, {_id:0, name:1});

(6) Set the filter to "active:true" and get only the first document with "active:true" value

Esky mtlb hai consider ek collection hai usky andar bohot sary documents hain. And sbky andar active:true ke ek field hai So kuch aesa karo ky just first wala document print

ho jismy ya field ho bario
mn ho le phir be display na hon

db.collectionName.find({active:true}).limit(1)

If we want to display first two document that matches this query.

db.collectionName.find({active:true}).limit(2)

⑦ Do the same as 6th question But with different method

db.collectionName.findOne({active:true})

⑧ Do the same as 6th question but this time get the 2nd field with active:true by skipping the first field(mean first document that has that field ignore that and print 2nd which matches that query).

db.collectionName.find({active:true}).limit(1).skip(1)

Two new functions we studied

limit()

The limit() function is used to specify the maximum number of documents that should be returned

by a query. It takes an integer argument representing the maximum number of document to be returned.

db.collection.find().limit(5)

This query will retrieve the first 5 documents from collection.

skip()

The skip() is used to skip() a certain number of documents from the beginning of the query result. It takes an integer argument representing the number of document to skip.

db.collection.find().skip(10)

This query will skip 10 document and return the remaining document.

Lecture 6)

Update documents in collection

Syntax

updateOne

UpdateOne() ⇒ db.collectionName.(<filter>, <update>)

to first only only one condition to fulfil query go update that

UpdateMany() ⇒ db.collectionName.update(<filter>, <update>)

Update Operators

MongoDB provides a set of update Operators that allow you to

perform specific modifications to fields within documents. Some commonly used operator include

$set	Set the value of a field
$inc	Increaments the value of a numeric field
$push	Adds an element to an array field
$pull	Removes elements from an array field
$pop	Remove the first or last element from an array field
$addToSet	Adds an element to an array field if it doesnot already exists

updateOne(filter, update, options)
Update a single document that matches the filter

updateMany(filter, update, options)
Update a multiple document that match the filter

Both methods take a filter object that specifies the documents to be updated, an update object that define the modifications to be applied and optional options such as upsert (insert if not found) and multi (update multi documents)

```
db.collectionName.updateOne(
    { name: "D1" },
    { $set: { value: 150 } }
)
```

```
db.collectionName.updateMany(
    { category: "electronics" },
    { $inc: { price: 10 } }
)
```

4 replaceOne(). Method

```
db.collectionName.replaceOne(
    { name: "Doc1" },
    { name: "NewDoc", value: 200 }
);
```

replaceOne() replaces the entire document, so any fields not present in the replacement document will be removed

The replaceOne() replaces a single document that matches the filter.

Lecture 7)

Delete Operation in mongodB

In mongodB we can perform deletion operation using various methods

1) deleteOne() and deleteMany() Method

deleteOne(filter, options) → optional
Delete single document that matches the filter

deleteMany(filter, options)
Delete multiple documents that matches the filter

Examples
db.collection.deleteOne({name: "Document 1"});
db.collection.deleteMany({category: "absolute"});
To delete All documents db.collection.deleteMany({})

(2) Remove() Method
It was commonly used for deleting documents, but now in recent version of MongodB its deprecated in favour of using deleteOne() and deleteMany()

db.collection.remove({name: "Document 1"});
It delete all documents that matches filter but now its not in 'use'

Lacture 8)

Install mongodb GUI Compass

Lecture 9)

MongoDB CRUD Operation in 1 minute using mongoDB Compass

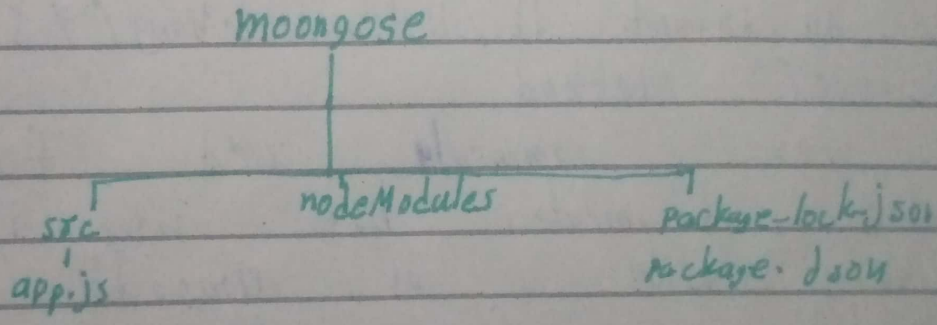
Its so easy Go and watch video

Lacture 10)

Introduction to mongoose | Connect Nodejs Express to mongodb using mongoose in Hindi

Mongoose is an object Data modeling library for mongoDB and Nodejs

It is used to connect mongoDB with nodejs / express.



For package.json npm install -y

To add moongoose as dependency

npm install mongoose

it will add modules into our the folder

Connecting to MongoDB

Mongodb Mongoose provides a way to connect your Node.js application to a mongodb database using the 'mongoose.connect()'

In app.js

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

mongoose.connect("mongodb://127.0.0.1:27017/azan", {
useNewUrlParser: true,
useUnifiedTopology: true, })          // return Promise
.then(() => {
    console.log("connect to mongodb"); })
.catch((err) => {
    console.log("Connect not successful", err);
});
```

We will create database azan if not exists.

The URL we used above was /azan was database and previous URI come from when we start server using mongosh in Cmd.

Other key features of mongoose are Schemas, models, CRUD Operations, Validation etc. We will study them in later video

node app.js <- file name will run in terminal

Lecture #11

Create and insert the Document using Express in Mongodb using Mongoose

Lecture 11)

Mongoose Schema and models Explained

Schema

A mongoose schema define the structure of the document, default values, validators etc

```
const playlistSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },

    ctype: String,
    videos: number,
    auther: String,
    active: Boolean,
    date: {
        type: Date,
        default: Date.now          // default value
    }
})
```

Model

A mongoose model is a wrapper on the mongoose Schema A mongoose model provides an interface to the database for creating, querying, updating, deleting records etc In a layman term we will create collection

in it

const Playlist = new mongoose.model("Playlist", playlistSchema);

Playlist → class

behind the scene plural ho jayga

// It simply mean create collection Playlists in which document will follow the structure playlistSchema

Lacture 12)

Lacture 12)

Create and insert Document using Express in Mongodb using Mongoose in

```
const reactPlaylist = new Playlist({
    name: "Node js",
    ctype: "Black End",
    video: 50,
    author: "Azan",
    active: true
})

reactPlaylist.save();   // return promise it will create a document in playlist collection
```

Advanced + best way

```
const createDocument = async () => {
  try {
    const reactPlaylist = new Playlist({
      name: "Node js",
      ctype: "Black End",
```

```
    videos: 50, author: "Azan", active: true })

    const result = await reactPlaylist.save();
    console.log(result);
  }
  catch(err){
        console.log(err);
  }
}
createDocument();
```

(Lecture 13)

How to insert multiple Documents using One line in Mongoose

Peachy jo humny para usmy ek hum easily add kar sktay hain if we want to add multiple documents using one line.

```
const createDocument = async ()=>{
  try{
    const mongoosePlaylist = new Playlist({
      name: "Mongoose Js",
      ctype: "Database",
      video: 4,
      author: "tape technical",
      active: true
    })

    const expressPlaylist = new Playlist({
      name: "Express ds",
```

ctype: "Back End", video: 20, author: "azad",
active: true
})

const result = await Playlist.insertMany([
mongoosePlaylist, expressPlaylist]);
console.log(result);
}
catch(err){
console.log(err);
}
}
createDocument();

Lacture 14)

How to read or querying the document using mangoose

const getDocument= async ()=>{
try{
const result = await Playlist.find().
select({name:1}); // just name field come
console.log(result);
}
catch(err){ console.log(err); }
}
getDocument();

Lecture 15

Mongodb Comparison Operators using mangoos and Express or Nodejs

$eq	Matches value that are equal to specified value
$gt	Matches value that are greater than specified value
$gte	Matches value >=
$in	Matches any of the values specified in array
$lt	Matches value < to specified value
$lte	<=
$ne	!=
$nin	Matches non of the values specified in an array

Example

```
const result = Playlist.find({videos: {$gt: 50}})

const result = Playlist.find({name: {$in: ["reactjs", "nextjs"]}});
```

Lecture 16)

Mongodb logical Query Operators

$and	Joins query clauses of with logical AND returns all document that matches the condition of both clauses
$not	Invert the effect of a query expression and returns the documents that do not match the query expression
$nor	Joins query clauses with a logical NOR returns all documents that fails to match both clauses
$or	Joins query clauses with a logical OR returns all document that matches the conditions of either clause

Syntax

$and

Perform operation on an array of expressions and selects the document that satisfy all other conditions

```
db.collectionName.find( {$and: [{condition1},
{condition2}]})
```

like

```
db.collectionName.find({$and:[{name:"Aron"},
{age:{$gt:25}}]});
```

$or
select documents that satisfy atleast one of the condition mentioned in array

db.collectionName.find({$or: [{cond1}, {cond2}]})

$not
Selects the documents that do not match the condition
db.collection.find({field: {$not: {$ex: value}}})

$nor
Donot satisfy any of the condition
db.collection.find({$nor: [{cond1}, {condition2}]})

$exists (use logical opr syntax)
Select documents where the specified field exits (or does not exit) in the document.
db.collection.find({field: {$exits: true}})
db.collection.find({field: {$exists: false}})

Lacture 17)
Mongodb Sorting and Count Query Methods

(1) countDocument()
db.collection.find().countDocument();

(2) Sort()
The Sort method allows you to specify one or more fields by which

to sort the documents and the sorting order (Asc or Desc) - Here the syntax
db.collectionName.find(query, Projection).sort({
sorting criteria});

(1) Sort by single field
db.collectionName.find({},{}).sort({fieldName:1}); //Asc
db.collection.....:-1}); //Desc

(2) Sort by multiple fields
db.collectionName.find().sort({field1:1, field2:-1});
asc

(3) Sort by nested field
db.collection.....({"outerfield.innerfield":-1});

Point to be Noted
When we used Mongoose we just used CollectionName.find() or collectionName.insertMany() etc - Collection Name is the name that come from (new mongoose.model(...))

Lecture 18)
Mongodb Update the Documents Using mongoose in Express

To update documents we can use
(1) UpdateOne or UpdateMany
(2) FindOneAndUpdate

We knows about updateOne or Many but let see one code

```
const filter = { name: "Ali" }; // Define a filter to match document
```

(1)

```
const update = { $set: { marks: 50 } };
```

(1)

```
Data.updateOne(filter, update)
  .then((result) => {
    console.log("Document updated", result);
  })
  .catch((error) => {
    console.error(error);
  });
```

We can use async/await for better readability.

(2nd)

```
Data.findOneAndUpdate(filter, update, { new: true, useFindAndModify: false })
  .then
  .
  .
  .
```

// new: true used to return updated document
// useFindAndModify we use here to warning ati hai

There are many other functions as well like findByIdAndUpdate()

Lacture 19)

MongoDB Delete documents using mongoose in Express App

If we want that just ek promise return ho jisey hmy pata chalay kitny document delete hovy we can use deleteOne() or Many()

If we want promise mn full document return ho jo hmny delete kia then used findByIdAndDelete()

Lacture 20)

Mongoose Build-in Validation

There are so many built in Validation we can see from official website

Mongoose provides severall built in validation options that you can use when defining your schema. These built in Validators help enforce data integrity by checking that values med specific criteria

Some commonly used built in validators in mongoose are

(1) Required Validator (required)

This validator ensure that a field is not empty or undefined

```
const userSchema = new mongoose.Schema
```

```
name: {
    type: String
    required: true
    },
});
```

(2) Min and Max Length of string Validator (minlength and maxlength)

These validors allow us to specify minimum and max string length for a field.

minlength: 5
maxlength: 20

(3) Min and Max Value Validator (min and max) value fall in specified range

min: 0
max: 1000

(4) Enumeration Validator (enum)

```
enum: ["users", admin", "guest"]
```

This Validator restrict a field value with a predefined set of values

(5) Match Validator (match)

checks if the field value matches a regular exp pattern

Point to remember

unique: true or unique: false is not considered as validator. In interview this question sometime asked

Lacture 21)

Create your Own Custom Validation using mongodb

One way

```
videos: {
        type: Number,
validate(value){
    If(value)<0){    throw new Error("not
    }
}
```

2nd Way

```
videos: {    type:Number,
  validate: {
        validator: function(value){
                return value.length <0
                },
        message: "video should not negative"
        }
}.
```

Lecture 22)

Using NPM validator Package
For validation using Mongodb

npmjs.com/package/validator website

npm i validator install

var validator = require('validator');

There are so many things we can do like
validator.isEmail("...");
It will return boolean

so Example in schema

```
email: {        type: String,
    validate(value){
        if(!validator.isEmail(value)){
            return throw Error("invalid Email");
            }
        }
    }
```

Lecture 23)

Postman

Download it from
postman.com/downloads/

Postman is an interactive and automatic tool for creating and verifying the Apis of your project
Postman is a Google Chrome app

for interacting with Http Apis.

It working or functionality is very easy we can revisit video if we forget.

We can test the Api by performing all the http method (Get, Post, Delete, Put) based on the operation or we can say mongodb operation we want to perform

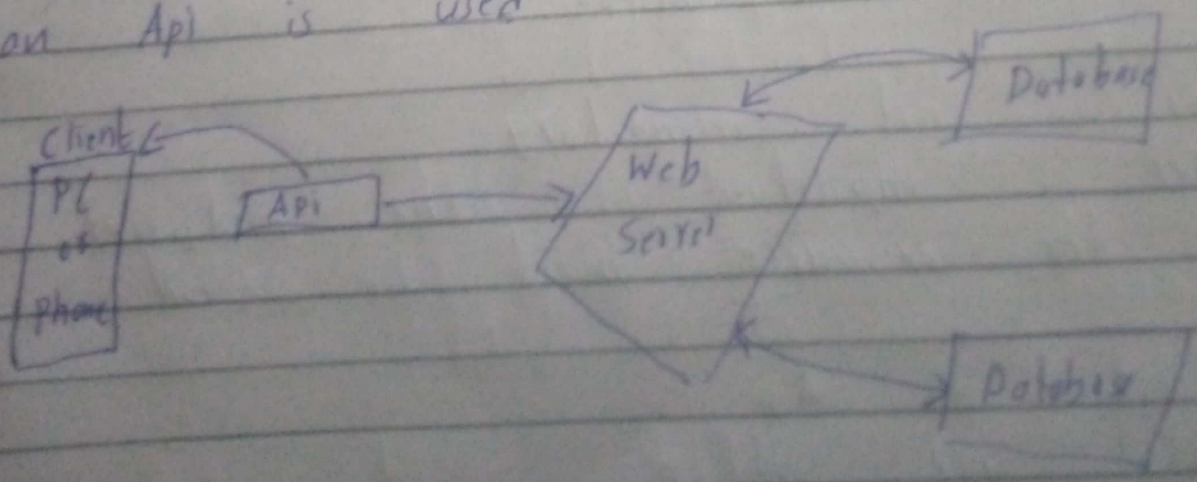
Lecture 24)

REST Api

Representational State transfer Application Programming interface

First let study what is Api.

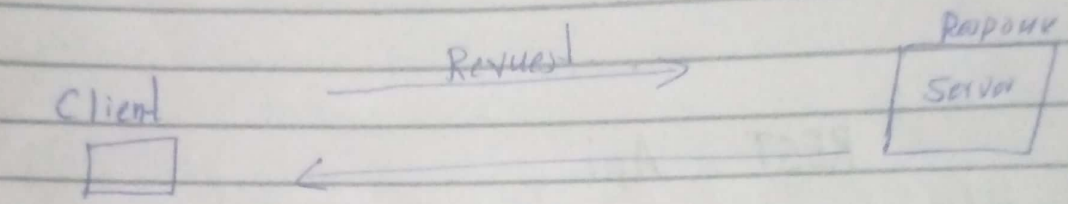
Api is a software that allows two applications to communicate with each other over the internet and through various devices. Every time you access an app like facebook or check weather on your smart phone an Api is used



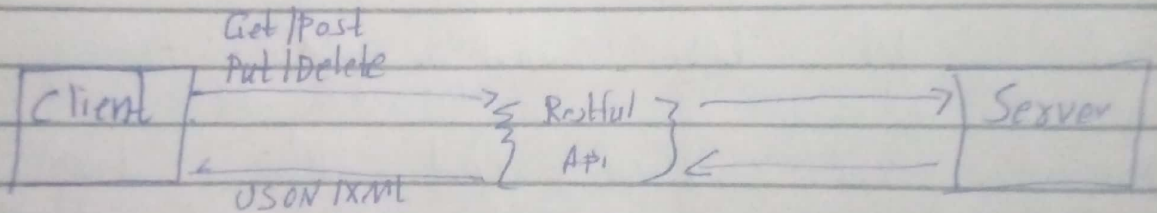
A restful Api is an architectural style for an Api that uses Http requests to access and use data Rest is not a programming language

Soap Api was used before that Now a days GraphQL is also used

Traditional way was



Now with restful Api



Now lets take an example of

http://azan.com/api/users

/api is not mandatory
what is important is the endpoint or the resource must be present

Url and Api
Urls are often used to access specific Api endpoints by making http requests that they are

not Apis themselves. they are simply the addresses used to reach Apis and other resources

example of URL
https://www.example.com/product?category=electronics&page=1

API
- endpoint naming
Api often have endpoint names in the url that imply data retrieval or interaction. look for keywords like "api", "v1", "data" or other terms that suggest programatic interface

URL of API frequently return structured data such as JSON or XML instead of displaying web content like html like https://api.example.com/v1/users just an example
(v1 endpoint(resource))

They are general guidelines and not all URLs or Apis follow these conventions

Create	POST		
Read	GET		
Update	PUT	PATCH	
Delete	Delete		

Ab hmy etna pta chal gya key now with REST Api hmary pass kuch http verb hain (POST, Delete ...) jinko hum CRUD operation ky liya use karany

The best benefit of it is to agr hum esy use nai karty hmy diff endpoint create karny pard like

/api/createUser
/api/deleteUser
...
etc

But jab hum REST API use karty hain tab endpoint hum same rakh skty like
/api/users
And http verb (POST, Delete, PUT, Get) sy hmy pata chaly ga konsa operation perform karna hai

If we want to read or update some specific user/document we will used id

http://azon.com/api/users/:id

Last 10 minutes must watch video from (15 to 25)

So summary of what we study is in restful api two norms exist

(1) Only http method change
(2) endpoint/resource is same

Very important lecture tha (i)

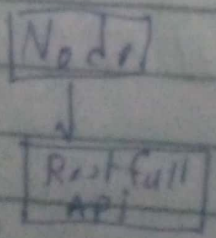
As per the REST architecture, the the server does not store any state about the client session on the server side statelessness means that every http request happens in complete isolation When clients makes an http request it include all information for the server to fulfil that request

Lecture 25)

Create Our own RESTFul Api. Handling POST requests in Rest Api

Total 5 videos honge jismy hum sb coud ops dalkangy. its first so esmy hum POST Ops or req dalkangy. In between some challenges ke videos bc ahyn gi

folder



So summary of what we study is in restful api two norms exist
(1) Only http method change
(2) endpoint/resource is same

Very important lecture tha ☺.

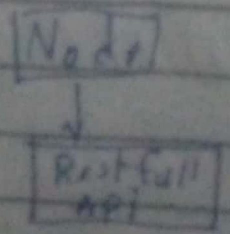
As per the REST architecture, the the server does not store any state about the client session on the server side statelessness means that every http request happens in complete isolation When clients makes an http request it include all information for the server to fulfil that request

Lecture 25)

Create Our own RESTful Api Handling POST requests in Rest Api

Total 5 videos hongi jismy humne sb crud ops dahkangy its first so esmy hum POST ops or req dahkangy In between some callenges ke videos be ahyn ge

Folder

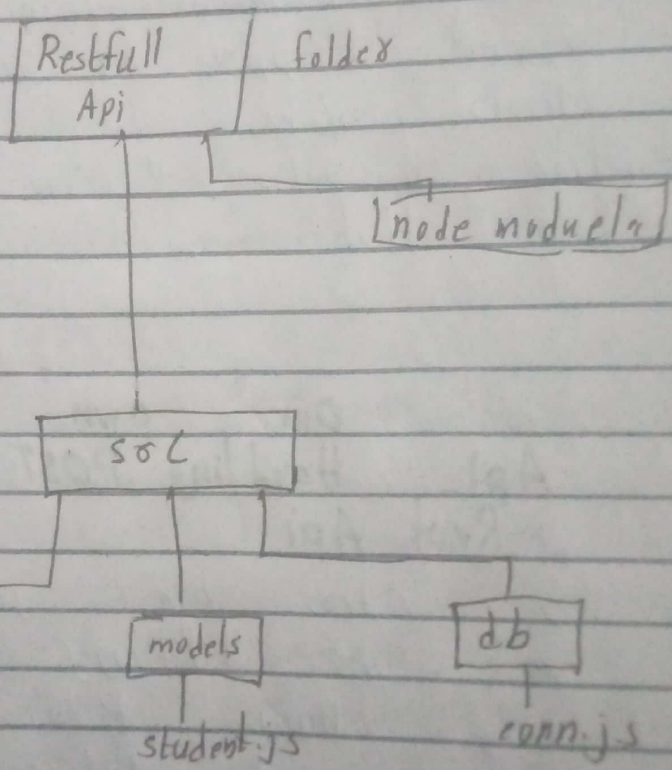


In restfull api folder add package.json by typing npm init -y in terminal

Now install express dependency into it
npm install express

Now install mongoose in it
npm i mongoose

Now install validator
npm i validator



Conn.js

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

mongoose.connect("mongodb://127.0.0.1:27017/studentdb", {
useNewUrlParser: true
useUnifiedTopology: true
});
```

```
.then(() => {
    console.log("Connection is successful");  })

.catch((err) => {
    console.log(`Error occured: ${err}`);
});
```

student.js

```
const mongoose = require('mongoose');
const validator = require("validator");

const schema = new mongoose.Schema({
  name: {
        type: String,
        required: true,
        minlength: 3  },
  email: {
        type: String,
        required: true,
        unique: [true, "email id already present"],
        validate(value){
          If(!validator.isEmail(value)){
            throw new Error("invalid email");
              }
          }
      },
Phone: {
      type: Number,
      min: 9,
      required: true,
      unique: true
  },
```

```
    address: {  type: String
                required: true
             }
});

// create collection
const Stud = new mongoose.model("Student", schema
);

module.exports = Stud;
```

app.js

```
const express = require("express");
require("./db/conn");

const Stud = require("./models/students");

const app = express();
const port = process.env.PORT || 3000;

app.use(express.json())      // middleware

// We don't need express.json() and express
.urlencoded() for GET Requests or Delete
Requests we only need it for
POST and put req
express.json() is a method inbuilt in express
to recognize the incoming Req object
as a json object This method
is called as a middleware in your
application using the code: app.use(
express.json());
```

```
app.post("/students", (req,res)=>{
    console.log(req.body);
    const user= new stud(req.body);

    user.save().then(
      ()=>{
        res.status(201).send("data added");
      })
    .catch((err)=>{
        res.status(400).send(err)
      })
  })

app.listen(port, ()=>{
    console.log(`server is started at
  ${port}`);
  })
```

Now we can ~~make~~ use Postman to check or test our above created Api.
Ok open postman create new collection then ~~make~~ add request give it any name

Now select POST and add Api localhost:3000/students
Now click on body choose raw and also instead of text choose JSON

Now make
{
"name": "azan",
"email": "azan@gmail.com",
"phone": 33129641,
"address": "emland"
}

Click on send Now (data added) print u will see Because we send it in res.send in our code

Extra Info
Pably jsy hum express mn hum routing krty hoty app.get("/", (req,res) => { res.send(" . . . "); }) para that esmy res.send wali cher browser mn es liya display hoti the because browser http req get send krta hai when we put localhost: Port and in return respon server snd krta tha esi liya we abi display hota hai

ture 26)
Now Build Your Own Rest Api using Async-Await

```
app.post("/students", async(req,res)=>{
    try{
        const user= new Stud(req.body
    );

    const data= await user.save();
    res.status(201).send("data added");}
catch(err){
    res.status(400).send(err)
    }
})
```

Lacture 27)

Build Restful Api, Handling Get request in Rest Api using express js and Mongodb

```
// Read data from registered student

app.get("/students", async(req, res)=>{
  try{
      const data= await Stud.find();
      res.status(201).send(data);
    }
catch(err){
    res.status(400)send(err)
    }
})
```

// get the individual student data using id

```
app.get("/students/:id", async(req, res)=>{
    try{
        const data= await Stud.findById({_id:req.
params.id});
        console.log(req.params);
        res.status(201).send(data)
        }

    catch(err){
        res.status(400).send(err)
        }
})
```

// NOW

Now we can use localhost:3000/students or localhost:3000/students/anyId in POSTMAN or browser to check we will get the required data or not. Browser is here mentioned it mean if we paste this Api in browser it will print the res.send because our server is working browser will send the get req to server server will get the data from database and send it as an response back to browser

We can also use Postman to test Api by checking get operation on it

Lecture 23)

Build Restful Api. Handling Put and Patch Requests in Rest Api

Put and Patch are two http method used for updating resource on a server in a Restful Api but they have different purposes and behaviour

PUT is used to replace the entire resource with new representation

PATCH is used to make partial updates to a resource, modifying only the specified fields or properties

```
//update the students by id

app.patch("/students/:id", async (req,res)=>{
  try{
    const data= Stud.updateOne({_id: req.params
.id }, {$set: req.body});
console.log(data)
If(!data){
      return res.status(404).send("Student not
      found"); }
res.status(200).send("updated");
  }
  catch(err){
    res.status(200).send(err);
      }
```

3)

Now test this Api using Postman
on patch. add some data
in body like { "email": "..." }
before that select raw and then replace
text option with json.

* We can also used
const data = await stud.findByIdAndUpdate(id
req.body, { new:true });
U also have to add
useFindAndModify: ~~true~~ false in mongoose.connect

Lecture 29)

Build Restful Api Handling Delete Reyuests in Rest Api

// here we will use id to delete ~~oth~~.
we can use any field

app.delete("/students/:id", asyn (req, res) => {
try{
const deletStudent = await Stud.
deleteOne({_id: req.params.id });

//we can also used
// const deleteStudent = await stud.findByIdAndDelete(
// {id: req.params.id });

if(!deleteStudent) {
return res.stated(404).send("student not found");
}
res.status(200).send(data)
}

```
catch(err) {
        res.status(400).send(err);
    }
});
```

Lecture 30)

Adding Express Router in Restful Api

This feature will make our restful Api more easy, and short and simple

```
// create a new router
const router = new express.Router()

// we need to define the router
router.get("/ozan", (req, res) => {
    res.send("hi i m ozan");
});
```

We can do any ~~fun~~ http function like Post, Put, Patch, get

```
// We need to register our router
app.use(router)
```

Ya simple hai basic use ky main file mn code thora cy rhy and most code dosri jgha cy import hoty ahy

If i forget plz revisit code or video

Hi its not part of playlist But its important lets study middleware from chatgpt

Middleware refer to functions or piece of code that sit between the incoming http request and the outgoing http response Middleware function have access to the req and res objects and they can perform various task such as modifying the req or res processing data and controlling the flow of the req handling pipeline

Example.

```
const express = require('express');
const app = express();
// middleware to log incoming requests
app.use((req, res, next) => {
console.log(`Request recieved: ${req.method} ${req.url}`);
next() // call next middleware in chain
});
// Route handler
app.get('/', (req, res) => { res.send('...'); });

// middleware for error handling
app.use((err, req, res, next) => {
console.log(err.stack);
res.status(500).send('something went wrong');
});
// start server
app.listen(3000, () => {
console.log('server is listening');
});
```

// in this example
(1) we create express.js application
(2) We define a middleware function that log information about the incoming req such as http method and URL
(3) We define a route handler for the root path
(4) We define another middleware function for error handling which will be invoked if an error occurs in any prev middleware or route handler.

Middleware are executed in the order they are defined
Express.js provide wide range of built in middleware and allow us to easily used third party middleware as well some commonly used built in middleware provided by express
- Express.json()
- express.urlencoded()
- express.static()
- express.cookieParser()
- express.session()
- express.compress()
- express.methodOverride()
- express.logger()

They are just a few example of the built in middleware provided by Express
Additionally there are vast ecosystem of third party middleware available through npm

Build Restful Api

Same previous concept repeated

Lecture 32)

Complete registration form using html Nodejs Express and mongodb

Es video mn humny pahly server side rendering ky liya files bnai And 2sra humny Register ka form bnaya mean user cy data laky database mn store karna.

Register.hbs or js jsy be apka logic ho

<form method="POST" action="/register">
<input type="text" placeholder="enter name" name="name">

<forminp type="password" ~~action~~ name="password">
<input type="submit">

</form>

Point to Note

- method and action attributes
- each input field must have name

in app.js → para code nai likh rha bas logic pracchy paran hain

```
app.post("/register", (req, res) {
try{
const data = new registerPdatabase({
    name: req.body.passname,
    password: req.body.password
})
const d = await data.save();
console.log("Data added");
res.send("You are registered");
}
catch(err){
    res.send(err);
}
}
```

→ name of input elem

Remember jab hum postman ky api ko access karty thy to usky ja data ata thy usky liye jab app.use(express.json()) use karty thy but jab form ky data ko usky liye apx wahy ky sth ye be add karna parta hai

app.use(express.urlencoded({ extended: false }))

(1) app.use(express.json())

This line of code sets up middleware to parse incoming requests with a JSON payload

It is commonly used when we expect

clients to send data in JSON format in the requested body, such as handling Api req or form submission with json data

When this middleware is used, express will automatically parse the JSON data from the request body and make it available in the req.body as a javascript object

(2) app.use(express.urlencoded({extended: "false"}))

This line of code sets up middleware to parse incoming req with url encoded data (typical from html forms)

This middleware is used when you expect client to send data using the "application/x-www-form-urlencoded" content-type which is a common way to submit form data in web application

When this middleware is used express will parse the URL encoded data from the req body and make it available in the req.body as javascript object

Lecture 33)

Signing form (login)

In login.hbs
// I will not write full code

(we are doing server side rendering we are not using react or .. for client side rendering)

```
<form method="POST" action="/login">
    <input type="email" placeholder="Enter email"
    name="email">
    <br>
    <input type="password" placeholder="enter password"
    name="email> <p style={color:red}> {{value}} </p>
    <button> login </button>
</form
```

In app.js // where backend is written

```
app.get("/login", (req, res) => {
        value: " " } )

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

const user= await RegisteredPeople.findOne({email:
  email} );

If(user) {
        If(user.password === password) {
```

```
res.status(200).render("index");
}
else{
res.status(404).render("login",{value:"Password is not matching"});
}

else{
    res.status(404).render("login",{value:"No user found with email"});
}
}
catch(err){
    res.status(500).render("login",{value:"An error occured"});
}
})
```

Diff b/w method="POST" method. "GET"

for sensitive data it is preferred to used POST Because in Get the data we are sending is visible in the url(query parameter) So for searching filters we can used that for for register/login we used POST So data is no where visible

Lacture 74)

Encryption vs Hashing || Secure Password using Bcryptjs in Nodejs and mongodb

Encryption

The primary purpose of encryption is to protect data confidentiality by converting plaintext(original data) into ciphertext(encoded data) using an encryption algo and a secret key The goal is to ensure that only authorized parties can decrypt and access original data

Hashing

Hashing is used to create a fixed size, irreversible representation(hash value) of data. Its mainly used for data integrity and verification. Hashes are used to verify that data has not been temperred with during storage or transmission

Points

Encryption is reversible Mean it is two way communication it can be decoded

Hashing is not reversible It is one way communication it cannot be decoded

So we will used one hashing algorithm - Bcryt

Because it is irreversible

install npm package for bcrypt
npm i bcryptjs

To use it

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

const userPassword = "...";

async function securePassword(userPassword){

    const passwordHash = await bcrypt.hash(password, 10);
    console.log("Hash is ${passwordHash}")
}

securePassword(userPassword);
```

Salt rounds

10 in above code in salt rounds parameter it defines the number rounds the library should go through to given you a secure hash. so dont set this value so high because it required high computational resource

Comparing Passwords

```
const hashedPassword = ""; // retrieve from database

const userEnteredPassword = 'mySecret';
```

```
async function verify(hashedPassword,
userEnteredPassword) {
    const result = await bcrypt.compare(
userEnteredPassword, hashedPassword);
  if (result){
      console.log('pass correct');
    }
  else{
      console.log('Pass is incorrect');
    }
  }
  catch(err) {
     console.log(err);
    }
  }

verify(hashedPassword, userEnteredPassword);
```

Note

bcrypt.compare(userEnteredPass, hashed);

(sequence matter)

Lecture 35)

Secure Registration System password with Bcrypt.js

Its very important and new thing if i forget i have to watch video bcz its not easy to explain. Yes concept is easy for hashing password enter by user in registeration

The code in app.post will remain same we will add one middleware in Scheama page where schema is defined which will make sure to hashed the password before calling .save() in the app.post which is on another file.
So in scheama page after defining scheama.

```
registerScheama.pre("save", async function(next)
{
  if(this.isModified("password")){

    this.password = await bcrypt.hash(this.password,
    10);
  }
  next()
})
```

its mean save call hony sy pre(pahly ye act hoga)

Remember

we used .pre() to define middleware like functions that run before specific operation such as saving or updating document in a database. These function are often refered as middleware in the context of database operations.

Lecture 36)

Complete Login Form with validation and Bcrypt JS a

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

const user= await RegisterPeople.findOne({email:email});
 If(user){
      const isMatch= await bcrypt.compare( password,
user.password );

 If(isMatch){
     res.status.render("index", {
          value: "You logged in successfully });
      }
else{
    res.status(404).render("login", {value: "Password is
 not matching" });
 } }
else{
    res.status(404).render ("login", { value: "No
user found with this email" });
 } }
     catch(err){
  res.status(500).render("login", {value: "An error
    occurred" });
 })
```

Lecture 37)

Node.js authentication & cookies JSON WebToken (JWT)

Cookies are small piece of data that a web server sends to the user's web browser and are stored on the user device.

They are commonly used for various purpose in web development, including user authentication, session management and tracking user behaviour - In the context of the Node.js and web application we can work with cookies to store and retrieve information b/w the server and the client.

Most important one use is jab hum ek website py sign in karty hain to wo usi baar although we sign in it after ky so liye kabhi nai bolti until we log out - hmary browser ko ksy pata chalta hai es bandy ny login pahly kia hova hai To yai hum cookies karti hain

Cookies cy ta hmy (mean browser) data milta hai But authentication bhot important hai Mean they are who they claim to be

And yai authentication ky bht hain method but JSON WebToken is very popular

Json webtoken is unique for every user.

General sequence of events when JWTs are used in Web application

User authentication

When we open a website, we are usually not authenticated initially, we might see a login page where we can enter our credentials or use some other authentication method (e.g OAuth with a social media account)

JWT creation Upon Authentication

After we successfuly authenticate ourself, the server generates a JWT as a part of authentication process. This JWT is often send back to the client as a response to the successful login

JWT usage

This JWT is then typically stored in a secure location on the client side, such as cookie or local storage It can be included in a subsequent requests to the browser server to prove your identity and acess protected resources

Server Side Validation

When a server recieve a request with a JWT, it validates the

token to ensure its legitimate and not expired. It also checks the claim with in the token to determine whether the user has the necessary permission to access the requested resource

Install package

npm i jsonwebtoken

let study how we can work with it

```
const jwt = require("jsonwebtoken");
const secretkey = "....."; // create more 32 characters
const createToken = async() => {

const token = await jwt.sign({ id: "....." },
secretkey );

console.log(token);

// To verify it is used when 2nd or later
user come to website

const userVer = await jwt.verify(token, secretkey);
console.log(userVer);
}
createToken()
```

jwt.sign()

optional optional

jwt.sign(payload, secretkey, options, callback)

Payload

This is the data you want to include in the JWT. Its typically an object containing claims (statements) about the user and additional data like { userId: ..., username: '...' }

Secret key

It is a string value that should be kept secret. They key is used to create signature which ensure authenticity of token

options(optional)

This is an optional object that allows you to specify various options for the JWT, such as its expiration time, not before time, algorithem and more example { expiresIn: '1h' }

callback(optional)

This is an optional callback function that is used or called once the JWT is signed. Mostly used to save token to database or sending it in a response

When user will able to login successfully server will generate jwt and send it to client

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

const user= await RegisterPeople.findOne({email:email})

If(user){
    const isMatch= await bcrypt.compare(password,
user.password);

IF(isMatch){ const token= jwt.sign({userId:user.id
                                    username:user.username
                                    secretkey});
    res.json({token});
}
else{
    res . . .
    }
} else{
        res . . . .
} catch(err){
    . . . .
})
```

Protecting Routes with JWT

We can protect routes by adding middleware to validate jwt before allowing access to those routes

the example

```
function authenticatetoken(req, res, next){
const tokene req.header('Authorization');
```

```
if(!token){
        return res.status(401).json({message: 'Authorization
token is miss' });
}
jwt.verify(token, secretkey, (err, user) => {
    if(err){
return res.status(403).json({message: 'invalid token'})
    );
}
req.user = user;
next();
});
}
//Example protected route
app.get('/profile', authenticateToken, (req, res)=>{
    res.json({message: 'welcome, ${req.user.username}!'});
});
```

req.user = user is used to store the decoded ~~token~~ user info from the JWT payload into the request object

(Lecture 38)
Registration form Signing Up user with JWT OAth token

(Lecture 39)
login form Signing in User with JWT OAth Token

Both these videos u can watch i can explain them with word some concepts enmy bith important hain jsy instance cy call kro to .method

to static

read itself collectionName ey karo my
etc. But hore videos ky
Just ye hai token bana ky
database my store filled enmy
ko res > send nai key
But enmy enjy ek chez sm) nai
i wo thapa rejister ky time token
generate karty database ko my store
v kar re hai wky usky koi
need nai hoti because token
generate login ky time karna
chaiya and usko database my
store ke jaga cookie my set
karky browser ko send karna
chaiya. And browser will automaticall
send that cookie to the ~~user~~
~~because~~ server always ~~ader~~ in the req.
until it expired or deleted

Lecture 40)

Manage Secrets & configs using .DOTENV in Nodejs and mongodb

• Dotenv is very important package. dsy
peachy hummy secret key dahki to
hmy usy hide karna perta hai
and usy or be info hoti hai.
Because wo both important hai.

It helps manage the environment
variable in our code. Environment

variable are used to store configuration information secrets or any data that should not be hard coded in your database

install

npm i dotenv

Simply ek .env file create karo root directory mn and usmy wo jo data rakhna hai wo kehlao like

any name
SECRET_KEY = "- - - - - ..".

Now come to other file where you need this key do

main file

require('dotenv').config() — must be at top

Now we can ~~use~~ access ~~(remove)~~ that data like process.env.SECRET_KEY

Once dotenv is configured in env file, it sets environment variables that can be accessed from any file with in your application without the need to re-require dotenv

Now Question is when we host this Project on github or anywhere use can access the .env file also thru whats the point

So ya github ky related hai do abi mujy learning hai But here
we will do
instal git init

Now we can creat one .gitignore file
Insid it hum
node_modules
.env
ya zour file ka name likh lyngy so onko use access nai kar skta user
.gitignore file ko koi be access nai kar skta

video mn es baary mn r be hai but wo abi mjy smj nai ah ra vky abi mujy github nai para

Important lecture

lecture 41)
Secure JWT Authentication
Store JWT Tokens in httpOnly Cookie

// cookies hum inspect kar ky application mein jaky dekh skty hain

// The res.cookie() function is used to set the cookie name to value
// The value paramater may be a string or object converted to JSON

syntax

optional

```
res.cookie(name, value, [options])
```

let say we want when user login generate a JWT token and stored that token in the client browse as a cookie

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

const user = await RegisterPeople.findOne({ gmail: email
});
  if (user) {
      const isMatch = await bcrypt.compare
(password, user.password);
  if (isMatch) {
  const token = jwt.sign({ id: user._id, gmail: user.gmail },
  process.env.SECRET_KEY);

  res.cookie("jwt", token);
  res.status(201).render("index");
  }
  else {
    res.status(404).render("login", { value: "password is
  not matching" });
  }
else {
```

```
}
    res.status(404).render("login", {value: "No user
found with this email"});
  }
}
catch(err){
    res.status(500).render("login", {value: "
An error occured"}); }
});
```

There are also optional options we can add for diff purpose like

```
res.cookie('jwt', token, {
    expires: '2023-12-31T23:59:59',
    domain: 'example.com',
    path: '/pth',
    secure: true,
    httpOnly: true,
    sameSite: 'strict',
    firstPartyOnly: true
})
```

(any name, value)

We can search at chatgpt for all these option what are there purpose

Lecture 42)

Challange

Same Prev concept

(Lecture 48)

How to get Cookie Value (JWT Token) using Cookie-Parser

JWT Token esliye like hai vky hummy cookie my yai set krna tha hum kuch x be set kar skty hain. name + value dy ky.

We need to install one package

```
npm i cookie-parser
```

It work as a middleware. So first we need to require and then use as middleware like

```
const cookieParser = require('cookie-Parser');

// To use as middleware
app.use(cookieParser())

// Let say we want to make one data file which user cannot access if he is not login

app.get("/data", (req, res) => {
                          // name we used during login
  IF(req.cookies.jwt){
    res.status(200).render("data");
  }
  else{
    res.status(500).send("you need to login if you want to access this page");
```

here is the security concern because we are not doing server validation means ~~same~~ token that is coming as our req is the same token or tempered one. so in next lecture how to verify token

Lecture 44)

Complete User Authentication & Authorization with JWT

```
//scenerio is same of prev video

app.get("/data", async (req, res) => {
    if(req.cookies.jwt) {
        try {
            const verifyUser = await jwt.verify(req.
cookies.jwt, process.env.secretkey);
            res.status(200).render("data");
        }
        catch(error) {
            res.status(402).send("<h1> To ~~access~~ t
Invalid or expired token </h1>");
        }

    else {
        res.status(400).send("<h1> To access this
page you need to login first </h1>")
    }
```

Lecture 45)

How to logout User And Delete Cookie

```
clearCookie('cookie name');

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

if(req.cookies.jwt) {
const isMatch = await jwt.verify(req.cookies.jwt,
process.env.SECRET_KEY);

if(isMatch){
  res.clearCookie("jwt");
res.status(200).render("logout");
}

else{
  res.status(400).send("<h1> invalid or expired
  token </h1>");
}
} else{
  res.status(400).send("<h1> To access This
page you need to login first </h1>");
}
});
```

Small Project

React → Express → ~~node~~.mongodb

One concept pahly clear kardy (javascript)

When while fetching we used response .json() it convert the coming data into. Sorry i want to say, if data is coming as JSON object it parse it into simple object.

When let JsonData = "[{ "name": "azure" }, { "name": "ali" }]";

```
JsonData[0].name        // error
let d = JSON.parse(JsonData)

d[index].name           // it will work
```

we can do

```
d.map((i) => {
    console.log(i.name);
  }
)
```

// We will do One form add data and submit to database
One display data - fetch all records from database and display on page.
And delete option delete all or delete one by Arid No.

// fetch karty hovy pahly database sy ata hatalangy. Then usy ek array of object banayn gy

using map and then usy as
an json send kardy gy.
hmy stringify used karny ki zaroorat
nai bc cause we will used ~~res.json~~
res.status().json() ya automatically usy
json bna dyga

Point to be noted
When we do just res.status.json("Hi");
ya hum koi object ya array
of objects mn sy to kuch pass
kar nai rhy so hmy khud
end py parse ng kr karon
zaroorat nai- Parse tab karty hain
jab JSON array of object or
just JSON object send horg ho.

To send data from React form to
Backend

```
function handleSubmit(e){
    e.preventDefault();
```
using Post 5000 in backend

```
fetch("http://localhost:5000/submit", {
  method: 'POST',
  headers: {
      'Content-Type': 'application/json'
  },
  body: JSON.stringify(val);
```
we state object

```
}).
.then((response) => {
    if(!response.ok){
```

```
        throw new Error('Arid No already exists');
      }
      return response.json();
    })
    .then((data)=>{
      alert(`form data submitted successfully! ${data.
          message}`);
    })
    .catch((error)=>{
      alert(`An error occured while submitting the
  form data. ${error}`);
    });
}
```

In app.js

```
app.post("/submit", async(req,res)=>{
  try{
    const user = await ReactForm.findOne({AridNo:
  req.body.aridNo});
    if(~~arid~~ user){
      throw new Error("Roll no already exist");
    }
    else{
    const add = new ReactForm({ name:req.body.name
    , AridNo:req.body.aridNo, Section:req.body.section, Marks:
    req.body.marks});

    const addData = await add.save();
  IF(!addData){
        res.status(500).json({message:"Data not
  found"});
  else{
    res.status(201).json({message:"form data
```

saved successfully" });
}
}
catch(error){
res.status(500).json({ message: error });
}
});

in Display.jsx

const Display = () => {
const [data, setData] = useState([]);

useEffect(() => {
fetch("http://localhost:5000/getData", { method:
"get" }).then((res) => { if(!res.ok){
throw new Error("Network response not ok");
} return res.json(); })
.then((responseData) => { setData(responseData);
})
.catch((error) => {
console.log("An erro occured");
});
}, []);
return(

<h1> data display </h1>

{data.length > 0 ? (

{data.map((item, index) => {
return(
<li key={index}>

```
<P> Name: {item.name} </P>  <P> AridNo:
{item.AridNo} </P>
<P> Section: {item.Section} </P>  <P> Marks:
{item.Marks} </P>
</li>
);  }))  </ul>  ):  (
<P> No data available </P>
)}
</div>
);
);
```

In app.js

```
app.get("/getData", async (req, res) => {
  try{
    const data = await ReactForm.find();
  if(!data){
    res.status(500).json("No data currently stored");
    }
  else{
    const dataArray = data.map((value) => {
    return { name: value.name, AridNo: value.
AridNo, Section: value.Section, Marks: value.Marks } });

  res.status(200).json(dataArray);
  }
} catch(err){
res.status(500).json({ message: "internal server error" }
);  }
});
```

Data Delete

```
const Delete = () => {
  const [val, setVal] = useState(null);
```

```
function handleChange(e){
    setVal(e.target.value); }

function handleSubmit(e){ e.preventDefault();
fetch("http://localhost:5000/deleteUser", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ aadNo: val }); important
})
  .then((res) => {  if(!res.ok){
    throw new Error("No user exists with this
AadNo");
  }
  return res.json();
})
  .then((data) => {
    alert(data);
  })
  .catch((err) => {
    alert(err);
  });
}

function deleteAll(){

fetch("http://localhost:5000/deleteAllUser", {
  method: "POST".  })
  .then((res) => {
    if(!res.ok){
  throw new Error('Network response was
not ok');
  }
  return res.json();
```

```
})
.then((resData) => {
  alert(resData.message);
})
.catch((error) => {
  alert(error);
});
}
return (
  <div>
    <h1>Delete Page</h1>
    <form onSubmit={handleSubmit}>
      <input type="text" value={val} name="aridNo"
      onChange={handleChange} />
      <button type="submit">Delete</button>
    </form>

    <button onClick={deleteAll}>Delete All
    </button>
    <p>{val}</p>
  </div>
)
}
```

Delete app.js

```
app.post("/deleteUser", async (req, res) => {
  try {
  const deletedUser = await ReactForm.findOneAndModify
  ({aridNo: req.body.aridNo});
  if(!deletedUser){
    throw new Error("No user found with
  this Arid No");
  }
  else{
  res.status(200).json("Record deleted");
```

```
  }
}
catch(err) {    res.status(400).json(err); }
});

app.post("/deleteAllUsers", async(req,res)=>{
  try{
const userCount=await RealForm.countDocuments({});
if(userCount===0){
return res.status(200).json({message:"No users
found"});
}
await RealForm.deleteMany({});
res.status(200).json({message:"All users removed
successfully"});
}
catch(err){
    res.status(500).json("internal server error");
  }
});
```

Mandatory

in app.js for react to talk with express/backend we used

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

app.use(cors());
```

Store files

To store file npm i multer

multer ky through he destination (jiss folder mn store honge files) filename (ky name jo rakhna hoga file ka) set karenge. Sath filter be lagayn gy ky [redacted] ki extension wali files ab skti hain. Jab one file ky work karna hoto ... single("fileInputName"); use karenge set at FrontEnd

Phir hum req.file ky single file ky related req.body ky baqi inputs ka data lyn ge

multerConfig folder
storageConfig.js file
const multer = require("multer");
// storage config

```
const storage = multer.diskStorage({
    destination: (req, file, callback) => {
        callback(null, "./uploads");
    },
    filename: (req, file, callback) => {
        const filename = `image-${Date.now()}.${file.originalname}`
        callback(null, filename);
    }
})
```

3)

```
// filter
const filefilter = (req, file, callback) => {
```

```
if( file.mimetype == image/png
"image/jpg" || file.mimetype === "image/jpeg"){
    callback(null, true);
  }
  else {
    callback(null, false);
    return callback(new Error("only .png, .jpg & .jpe
      formated Allowed"))
  }
}

const upload = multer({
    storage: storage,
    fileFilter: fileFilter
  })

exports.module = upload;

// If we follow router/controller
structure
Router Folder
    router.js
const express = require("express");
const upload = require("../multerConfig/storageConfig")

const router = new express.Router();

const controllers = require("../controllers/usersControllers

router.post("/user/register", upload.single("user_profile
, controllers.userpost);

module.export = router;
```

Controllers // folder
usersControllers.js

```
const user = require("../models/schema");
exports.userpost = asyn (req,res) => {
    console.log(req.file); console.log(req.file.filename);
    console.log(req.body);
}
```

In App.js

```
const express = require("express");
const app = express();
const cors = require("cors");
const router = require("./Routes/router");

app.use(cors());
app.use(express.json());
app.use(router);

app.listen(2010, () => { console.log("server started"
})
```

So in common Practise on server side in one folder we store images And store its name in database document.

So to get images from backend and print on front end

bad

on react side

src = {`http://localhost:6010/uploads/${receivedData.
```
```

profile}`}

One server side in App.js

app.use("/uploads", express.static("./uploads"));

Project learning

CQ- Difference b/w Params(Parameters) and query Params(Parameters)

Params is a general term refers to a parameter passed to a function or a method In context of web development it can refer to parameters passed in a various ways such as in the URL, in the request body or as part of header.

Query Parameters are included in the URL of an Http request. These parameters ~~after~~ too are usually appended to end of URL after ? mark ('?') and multiple parameters are separated by ampersands (&

Example

`http://localhost:6010/user/${id}?search=${search}&gender=${gender}`

here id is path parameter
Search and gender are query parameters
To access them at backend

```
const {id} = req.params;
const {search} = req.query.search;
```

For searching matching pattern (regex)

- Direct equality check

```
const query = { fname: search };
// This would only match exact matches
```

- Using regex for partial matching

```
const query = { fname: { $regex: "John", $options: "i" } }
//This would match any document where fname
//contain the specified substring. option i is
for case sensitive ignoring.
```

Fast-csv package

Used in project `npm i fast-csv`