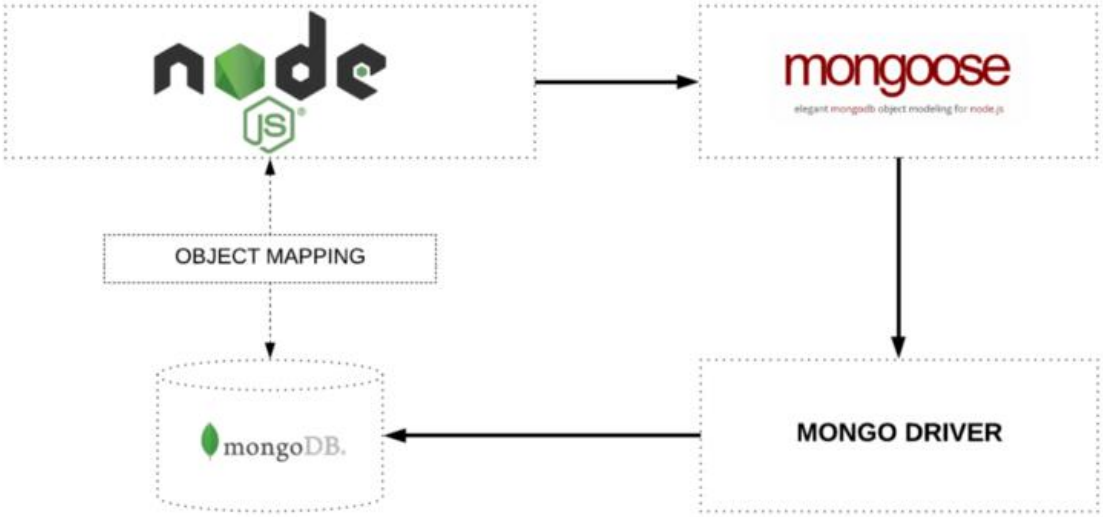
Connection with NodeJS

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.



Object Mapping between Node and MongoDB managed via Mongoose

MongoDB is a schema-less NoSQL document database. It means you can store JSON documents in it, and the structure of these documents can vary as it is not enforced like SQL databases. This is one of the advantages of using NoSQL as it speeds up application development and reduces the complexity of deployments.

Mongoose is built on top of the MongoDB driver to provide us with a means to model data easily.

Advantages of mongoose:

1. Validation of the MongoDB database collections can be done very effectively with mongoose.
2. Implementation of predefined structures on Collections is done quickly.
3. Mongoose module provides the abstraction layer for reading and defining a query.

Disadvantages of mongoose:

The main disadvantage of mongoose is that the abstraction comes at the cost of performance as compared to the MongoDB driver. MongoDB driver is around 2x faster than the Mongoose.

Why Mongoose?

By default, MongoDB has a flexible data model. This makes MongoDB databases very easy to alter and update in the future. Mongoose forces a semi-rigid schema from the beginning. With Mongoose, developers must define a Schema and Model.

First be sure you have MongoDB and Node.js installed.

Install Mongoose from the command line using npm:

```
npm install mongoose
```

The first thing we need to do is include mongoose in our project and open a connection to the test database on our locally running instance of MongoDB.

```
const mg = require('mongoose');
mg.connect("mongodb://127.0.0.1:27017/test").then(()=>{console.log("success")}).catch((err)
=>{console.error(err)});
```

What is a schema?

A schema defines the structure of your collection documents. A Mongoose schema maps directly to a MongoDB collection.

```
const myschema=new mg.Schema({
   name:{type:String,required:true}, //name must be given else error
   surname:String,
   age:Number,
   active:Boolean,
   date:{type:Date,default:new Date()}
})
```

With schemas, we define each field and its data type. Permitted types are:
String,Number,Date,Buffer,Boolean,Mixed,ObjectId,Array,Decimal128,Map

What is a model?

Models take your schema and apply it to each document in its collection.

Models are responsible for all document interactions like creating, reading, updating, and deleting (CRUD).

An important note: the first argument passed to the model should be the singular form of your collection name. Mongoose automatically changes this to the plural form, transforms it to lowercase, and uses that for the database collection name.

```
const person=new mg.model("person",mySchema)
```

it will automatically converted to “people” by mongoose

```
mg.pluralize(null) // to add collection as we have mentioned
```

Await and Async

If you are using async functions, you can use the await operator on a Promise to pause further execution until the Promise reaches either the Fulfilled or Rejected state and returns. Since the await operator waits for the resolution of the Promise, you can use it in place of Promise chaining to sequentially execute your logic.

Inserting data

Now that we have our first model and schema set up, we can start inserting data into our database.

Back in the file, let's insert a new data.

```
const persondata=new person({
 name:"DDD",
surname:"PQR2",
age:2,
active:true
})
persondata.save()
//to add multiple data at once
const persondata2=new person({
name:"ABC2",
surname:"PQR2",
age:2,
active:true
})
persondata2.save()
```

we create a new object and then use the save() method to insert it into our MongoDB database.

Example: Add name, Surname,Age,Active and date field in persons collection inside Test database.

```
const mg=require("mongoose")
mg.connect("mongodb://127.0.0.1:27017/test").then(()=>{console.log("success")}).catch((err)
=>{console.error(err)});
//mg.pluralize(null)
const mySchema=new mg.Schema({
        name:{type:String, required:true},
        Surname:String,
        age:Number,
        date:{type:Date, default:new Date()
//This adds date field with all dates set in ISO format.ISO format is yyyy-mm-dd included with time 5:35:35. The time has zero time zone, so we have to add 5:30, because India has GMT+5:30 time zone.
        }
    }
);
const person=new mg.model("person",mySchema)
```

```
// Type:1 To insert one by one data
const personData=new person({ name:"ABC",  Surname:"Patel",   age:30})
personData.save() // also returns promise

const personData1=new person({ name:"XYZ",  Surname:"Patel",   age:30})
personData1.save() // also returns promise

// Type:2 Allow to create table though data has error
const createdoc=async()=>
{
try
   {
      Const persondata=new person({name:" ABC ",surname:" Patel ",age:30})
      const result=await persondata.save();
      console.log(result)
   }
catch(err){console.log("There is error")}
}
createdoc()
// Type:3 To insert Multiple data at once
const createDoc=async()=>
   {
      try{
      const personData=new person({ name:"ABC",  Surname:"Patel",   age:30})
      const personData1=new person ({ name:"XYZ",  Surname:"Patel",   age:30})
      const result= await person.insertMany   ([personData,personData1])
      console.log(result)
      }
         catch(err)
         {
            console.log("problem");
         }
   }
createDoc();
```

****Alternatively one can use push method also.**

Mongoose Example using push method: Insert/Find/Update /Delete

Create Collection “employees” with following data

```
[{_id: 1,name: "Eric",age: 30,position: "Full Stack Developer",salary: 60000},

{_id: 2,name: "Erica",age: 35,position: "Intern",salary: 8000},

{_id: 3,name: "Erical",age: 40,position: "UX/UI Designer",salary: 56000},

{_id: 4,name: "treric7",age: 37,position: "Team Leader",salary: 85000},

{_id: 5,name: "Eliza",age: 25,position: "Software Developer",salary: 45000},

{_id: 6,name: "Trian",age: 29,position: "Data Scientist",salary: 75000},

{_id: 7,name: "Elizan",age: 25,position: "Full Stack Developer",salary: 49000}]
```

1) Find All Documents:
2) Find Documents by Position “Full Stack Developer”:
3) Retrieve name of employees whose age is greater than or equal to 25 and less than or equal to 40.
4) Retrieve name of the employee with the highest salary.
5) Retrieve employees with a salary greater than 50000.
6) Retrieve employees' names and positions, excluding the "_id" field.
7) Count the number of employees who have salary greater than 50000
8) Retrieve employees who are either " **Software Developer**" or "**Full Stack Developer**" and are below 30 years.
9) Increase the salary of an employee who has salary less than 50000 by 10%.
10) Delete all employees who are older than 50.
11) Give a 5% salary raise to all "**Data Scientist**"
12) Find documents where name like “%an”
13) Find documents where name like “Eri--” (Case Insensitive)
14) Find documents where name like “%ric%”
15) Find documents where name contains only 4 or 5 letters.
16) Find documents where name must end with digit

The ***push*()** ***method*** adds new items to the end of an array. The ***push*()** ***method*** changes the length of the array. The ***push*()** ***method*** returns the new length. It will display all results in array.

```
const mg=require("mongoose")
const v=require("validator")
mg.connect("mongodb://127.0.0.1:27017/lju").then(()=>{console.log("success")}).catch((err)=
>{console.error(err)});
//mg.pluralize(null)

   const mySchema=new mg.Schema( {
```

```
        _id:Number,    name:String,
        age:Number,    position:String,
        salary:Number
    });
const emp=new mg.model("employ",mySchema)
const createDoc=async()=>
  {
    try{
      const personData1=[{_id: 1,name: "Eric",age: 30,position: "Full Stack Developer",salary:
60000},
      {_id: 2,name: "Erica",age: 35,position: "Intern",salary: 8000},
      {_id: 3,name: "Erical",age: 40,position: "UX/UI Designer",salary: 56000},
      {_id: 4,name: "treric7",age: 37,position: "Team Leader",salary: 85000},
      {_id: 5,name: "Eliza",age: 25,position: "Software Developer",salary: 45000},
      {_id: 6,name: "Trian",age: 29,position: "Data Scientist",salary: 75000},
      {_id: 7,name: "Elizan",age: 25,position: "Full Stack Developer",salary: 49000}]
      const result=[]
      result.push(await emp.insertMany(personData1))
      result.push(await emp.find())
      result.push(await emp.find({position:"Full Stack Developer"}))
      result.push(await emp.find({ age: { $gte: 30, $lte: 40 } },{name:1,_id:0}))
      result.push(await emp.find().sort({ salary: -1 }).limit(1))
      result.push(await emp.find({ salary: { $gt: 50000 } }))
      result.push(await emp.find({salary:{$gt:50000}}).count())
      result.push(await emp.find({ $and: [{ $or: [{ position: "Software Developer" }, { position:
"Full Stack Developer" }] }, { age: { $lt: 30 } }] }))
      result.push(await emp.updateOne({salary:{$lt:50000}},{ $mul: { salary: 1.1 } }))
      result.push(await emp.deleteMany({ age: { $gt: 50 } }))
      result.push(await emp.updateMany({ position: "Data Scientist" }, { $mul: { salary: 1.05 }
}))
      result.push(await emp.find({name:{$regex:/an$/}}))
      result.push(await emp.find({name:{$regex:/^eri[A-z]{2}$/i}}))

      console.log('Query Results:', result);
          }
    catch(err)    {
      console.log(err);
    } }
createDoc()
```

FindByIdAndUpdate()

```
const mg=require("mongoose")
mg.connect("mongodb://127.0.0.1:27017/test").then(()=>{console.log("success")}).catch((err)
=>{console.error(err)});
//mg.pluralize(null)
const mySchema=new mg.Schema(
   {
      name:{
         type:String,
         required:true
      },
      Surname:String,
      age:Number,
      active:Boolean
   }
);
const person=new mg.model("person",mySchema)
const updateDoc=async(i) =>{
  const result=await person.findByIdAndUpdate({_id:i},
//      or const result=await person.updateOne({_id:i},

 {
         $set:{age:27}
  },
    {new:true}) // only in case of using findByIdAndUpdate()
console.log(result)
 }
updateDoc("64e8527c4e2f858cfc0457bb");
```

TO findByIdAndDelete()

```
const deleteDoc=async(_id) =>
{

      const result=await person.findByIdAndDelete({_id})
        console.log(result)
}
deleteDoc("64e8527c4e2f858cfc0457bb");
```

Validator

1. **Basic validation:** While creating document,if there are attributes and we want to specify some constraints,then it's possible by adding respective members in schema creation. If we insert duplicate elements in name field,then it may throw error,if specify name as **unique:true.** After insertion,if we have specified **lowercase:true** then it may show all newly inserted names in lowercase.
If we have set a list of allowed values in enum, then it requires exact match.
2. **Custom valication:** We can provide custom validation using **validate(value)** inbuild function.
3**. Validator package**: Using Validator package by using npm i validator

```
const mg=require("mongoose")
const v=require("validator")
mg.connect("mongodb://127.0.0.1:27017/lju").then(()=>{console.log("success")}).catch((err)=
>{console.error(err)});
//mg.pluralize(null)
const mySchema=new mg.Schema(
   {
      name:{  type:String,  required:true,  lowercase:true,  trim:true,
              minlength:[3,"Min length must be 3"], maxlength:[7,"max length must be 7"],
              enum:["abc","xyz","def"] },
// enum: {
   //  values: ['Coffee', 'Tea'],
  //   message: '{VALUE} is not supported'
 //  }

     age:{   type:Number,
         validate(v1){
            if(v1<=0)
            {throw new Error("Number must be positive")  }  }
            },

    email:{ type:String,
              validate(val)
             {
                if(!v.isEmail(val))
            {
              throw new Error("Enter valid email_id")
            }}}
  });
const person=new mg.model("person",mySchema)
const personData=new person( {  name:"def",   Surname:"hi",
            age:0,
            email:"abc@gmail.com"    }  )
const personData1=new person( {
            name:"xyz",   Surname:"hello",
```

```
        age:32,
        email:"xyz@gmail.com"
    } )
personData.save();
personData1.save();
```

Connectivity of MongoDB with ExpressJS

Task: Create a form having username and password and insert data entered by user in collection named "data1" in mongoDB.

In task.js file

```
var expr=require("express")
var app=expr()
const mg=require("mongoose")

mg.connect("mongodb://127.0.0.1:27017/login")
.then(()=>{console.log("Successful")})
.catch((err)=>{console.error(err)})

mg.pluralize(null)
const myschema=new mg.Schema({
       uname:{type:String, required:true},
       password: {type:String, required:true} })

const person =new mg.model("data1", myschema)

app.use(expr.static(__dirname,{index:"form.html"}))
//Or app.get("/",(req,res)=>{ res.sendFile(__dirname+"/form.html") })

app.get("/process_get",(req,res)=>{
      const personData=new person({
      uname:req.query.uname,
      password:req.query.pwd
})
personData.save()
res.send("Record inserted")
})
app.listen(3000)
```

form.html

```
<html>
   <form action="/process_get" method="get">
      Username: <input type="text" name="uname"/>
   </br> Password: <input type="password" name="pwd"/>
   </br> <input type="submit"/>
   </form>
</html>
```

Connectivity of MongoDB with React

Set Up a MongoDB Database

- MongoDB is the most popular NoSQL database. It is an open-source database that stores data in JSON-like documents (tables) inside collections (databases).
- Open the MongoDB Compass app. Then, click the **New Connection** button to create a connection with the MongoDB server running locally.
- If you do not have access to the MongoDB Compass GUI tool, you can use the MongoDB shell tool to create a database and the collection.
- Provide the connection URI and the name of the connection, then hit **Save & Connect**.
- Lastly, click on Create Database button, fill in the database name, and provide a collection name for a demo collection.

Create a React Client

```
npx create-react-app my-app
cd my-app
npm start
```

Next, install Axios. This package will enable you to send HTTP requests to your backend Express.js server to store data in your MongoDB database.

```
npm install axios
```

Create a Demo Form to Collect User Data

Signup.js

```
import React, { useState } from 'react';
import axios from 'axios';

function Signup() {
  const [username, setUsername] = useState('');

  const handleSignup = async (e) => {
    e.preventDefault();

    try {
      await axios.post('http://localhost:5000/signup', {
        username
      });
     alert('User signed up successfully.');
```

```
    setUsername('');
   }
 catch (error) {
    console.error('Error signing up:', error);
   alert('An error occurred.');
  }
 };

 return (
  <div>
   <h1>Sign Up</h1>
   <form onSubmit={handleSignup} method =’post’>
   Name: <input type="text" placeholder="Username" value={username}
      onChange={(e) => setUsername(e.target.value)} />
     <button type="submit">Sign Up</button>
   </form>
  </div>
 )}

export default Signup;
```

App.js

```
import Signup from "./Signup"
function App(){
return(
<>
 <Signup/>
</>
 )}
export default App
```

To run use → npm start

- Declare one state a name to hold the user data collected from the input field using the useState hook.
- The **onChange** method of each input field runs a callback that uses the state methods to capture and store data the user submits via the form.
- To submit the data to the backend server, the onSubmit handler function uses the **Axios.post** method to submit the data passed from the states as an object to the backend API endpoint.

Create an Express js Backend

An Express backend acts as middleware between your React client and the MongoDB database. From the server, you can define your data schemas and establish the connection between the client and the database.

Create an Express web server and install these packages:

```
npm install mongoose
npm install cors
npm install body-parser
```

- Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node. It provides a simplified schema-based method, to model your application data and store it in a MongoDB database.

- The CORS (Cross-Origin Resource Sharing) package provides a mechanism for the backend server and a frontend client to communicate and pass data via the API endpoints.

- **app.use(bodyParser.json())** basically tells the system that you want json to be used. Here we are using .json

- **bodyParser.urlencoded({extended: ...})** basically tells the system whether you want to use a simple algorithm for shallow parsing (i.e. false) or complex algorithm for deep parsing that can deal with nested objects (i.e. true).

server.js

```
const express = require('express');
const mg = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();

app.use(cors());
app.use(bodyParser.json());

mg.connect('mongodb://127.0.0.1:27017/User') .then(()=>{console.log("Connection
Success")})

const UserSchema = new mg.Schema({  username:String  });

const User = new mg.model('User', UserSchema);
```

```
app.post('/signup', async (req, res) => {
  try {
    const { username} = req.body;
//console.log("Username is" + req.body.username)

    const newUser = new User({ username});
    await newUser.save();
    res.send();
// or res.json({message:'Username inserted Successfully'})
  } catch (error) {
    res.send(error);
//or res.json({message:'Error in inserted Record'})
  }
});

app.listen(5000);
```

hit: node server.js

Miscellaneous Task (*For Reference Only)

Create a form with login and signup facilities. If user is already registered than he can login if registered username and password matches with entered details. If user has not registered than he can signup with username and password and details should be entered in database. After login it should display that login successful.

Express Server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const bcrypt = require('bcryptjs');
const cors = require('cors');

const app = express();

app.use(cors());
app.use(bodyParser.json());

mongoose.connect('mongodb://127.0.0.1:27017/KPP');

const UserSchema = new mongoose.Schema({
  username: String,
  password: String,
});

const User = mongoose.model('User', UserSchema);

app.post('/api/signup', async (req, res) => {
  try {
    const { username, email, password } = req.body;
    console.log("Usename is"+req.body.username);
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();
    res.status(201).json({ message: 'User signed up successfully.' });
  } catch (error) {
    res.status(500).json({ error: 'An error occurred.' });
  }
});

app.post('/api/login', async (req, res) => {
  try {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
```

```
  if (!user) {
    return res.status(401).json({ error: 'User not found.' });
  }

  const passwordMatch = await bcrypt.compare(password, user.password);

  if (!passwordMatch) {
    return res.status(401).json({ error: 'Invalid password.' });
  }

  res.json({ message: 'Login successful.' });
 } catch (error) {
  res.status(500).json({ error: 'Aa error aave che.' });
 }
});
app.listen(5000)
```

React:

Signup2.js

```
import React, { useState } from 'react';
import axios from 'axios';

function Signup2() {
 const [username, setUsername] = useState('');
 const [password, setpassword] = useState('');
 const handleSignup = async (e) => {
  e.preventDefault();

  try {
    await axios.post('http://localhost:5000/api/signup', {
      username,
     password
    });
    alert('User signed up successfully.');
    setUsername('');
   setpassword('');
  }
  catch (error) {
     alert('An error occurred.');
  }
 };

 return (
```

```
   <div >
<h1>Sign Up</h1>
     <form onSubmit={handleSignup} method='post'>
     <label >Name</label>
<input
         type="text"
         value={username}
         required
         onChange={(e) => setUsername(e.target.value)}
       />
<label>Password</label>
<input
         type="password"
         value={password}
         required
         onChange={(e) => setpassword(e.target.value)}
       />

<button type="submit">Sign Up</button>
     </form>
   </div>
  );
}

export default Signup2;
```

Login2.js

```
import React, { useState } from 'react';
import axios from 'axios';


function Login2() {
  const [username, setUsername] = useState('');
  const [password, setpassword] = useState('');
  const handleLogin = async (e) => {
    e.preventDefault();

    try {
      await axios.post('http://localhost:5000/api/login', {
      username,
       password
      });
      alert('User Log in successfully.');
      setUsername('');
     setpassword('');
    }
    catch (error) {
       alert('An error occurred.');
    }
```

```
  };

  return (
    <div >
<h1>Login</h1>
      <form onSubmit={handleLogin} method='post'>
      <label >Name</label>
<input
          type="text"
          value={username}
          required
          onChange={(e) => setUsername(e.target.value)}
        />
<label>Password</label>
<input
          type="password"
          value={password}
          required
          onChange={(e) => setpassword(e.target.value)}
        />

<button type="submit">Login</button>
      </form>
    </div>
  );
}

export default Login2;
```

*Import Signup2.js and Login2.js in App.js