**Practical - 4**

By using Mongoose, developers can define the structure of their data, identify each field's data type, and apply validation rules to ensure data integrity and consistency.

Mongoose is a library for Node.js and MongoDB that enables Object Data Modeling (ODM). It functions as an abstraction layer above MongoDB.

Middleware functions are available to assist in handling pre-and post-processing of data, while virtual properties enable developers to define computed fields. Query builders are also available to help construct complex queries with ease.

One of the major advantages of using a NoSQL database like MongoDB is the flexibility it offers in data modeling. Unlike traditional relational databases, MongoDB allows you to add or remove fields, nest data multiple layers deep, and create a truly flexible data model that can adapt to your current needs and evolving requirements.

However, having too much flexibility can also be a drawback. If there is no agreement on what the data model should look like, and every document in a collection contains vastly different fields, it can become a significant challenge to manage the data. In such a scenario, the lack of structure can make it difficult to query and analyse the data efficiently, leading to potential performance and maintenance issues.

npm install mongoose

**Task 1: Connect with mongoDB using MONGOOSE**

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/my\_database', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

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

**Task 2: Create a schema**.

const krischema = new mongoose.Schema({

name: {

type: String,

required: true

},

email: {

type: String,

required: true,

unique: true

},

age: {

type: Number,

default: 0

},

deregistered: {

type: Date,

default: Date.now

}

});

**Creating a model from this schema.**

const obj = mongoose.model('kri', krischema);

**Task 3:** **CRUD OPERATIONS**

Create:

const newobj = new obj({

name: 'Jo',

email: 'jon@gmail.com',

age: 25

});

// Save the new user document to the database

newobj.save()

.then(kri => console.log('User created:', kri))

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

Find :

Without query:

obj.find({})

.then((doc)=>{

console.log(doc);

})

.catch((err)=>{

console.log(err);

});

With query:

obj.find({name:"Jo"})

.then((doc)=>{

console.log(doc);

})

.catch((err)=>{

console.log(err);

});

Insert:

insertOne:

const Newuser = new obj({name: 'krish',

email: 'ky@gmail.com',

age: 20});

// it store directly to collection

obj.collection.insertOne(Newuser)

.then((data)=>{

console.log(data);

}).catch((err)=>{

console.log(err);}

);

insertMany:

const Newuser = new obj({name: 'krishna',

email: 'ku@gmail.com',

age: 20});

const Newuser1 = new obj({name: 'kriti',

email: 'kr@gmail.com',

age: 21});

// it store directly to collection

obj.collection.insertMany([Newuser,Newuser1])

.then((data)=>{

console.log (data);

}).catch((err)=>{

console.log(err);}

);

Update:

UpdateOne:

obj.updateOne({name:"josh"},{$set:{name: "jade"}})

.then((docs)=>{

if(docs) {

console.log(docs);

} else {

console.log("no such user exist");

}

}).catch((err)=>{

console.log(err);

})

UpdateMany:

obj.updateMany({age: 20},{$set:{ age: 15}})

.then((docs)=>{

if(docs) {

console.log(docs);

} else {

console.log("no such user exist");

}

}).catch((err)=>{

console.log(err);

})

Delete:

deleteOne:

obj.deleteOne({name: "jade"})

.then((docs)=>{

if(docs) {

console.log(docs);

} else {

console.log("no such user exist");

}

}).catch((err)=>{

console.log(err);

})

deleteMany:

obj.deleteMany({age: 15})

.then((docs)=>{

if(docs) {

console.log(docs);

} else {

console.log("no such user exist");

}

}).catch((err)=>{

console.log(err);

})

obj.findOneAndDelete({name:"kriti"})

.then((docs)=>{

if(docs) {

console.log(docs);

} else {

console.log("no such user exist");

}

}).catch((err)=>{

console.log(err);

})

findOneAndUpdate() – to find and update