**Local Setup for using Mongoo**

* **Software:**
  + VS Code
  + NodeJS
  + MongoDB Compass

**CHEAT SHEET MongoDB**

**Local installation and Setup of MongoDB**

* URL to download: [https://www.mongodb.com/try/download/...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbmVPMHhRQ1BiN1lSZ2RWa2FwbUhnZDhMdVZsd3xBQ3Jtc0treWNCcFQ4enk1RUR3aEZrX1l1aWdpMnF2OUt4OVFrUC1BSGM1b0lsV1NoaUlReFUtYVBPemlSd0tWX0xYSkstRnFYQVdDUXRZVWRibWZCSk5vTnJHSWhlc1RQY2pucThUTHV3UTVocWVJbXkyb0Jnaw&q=https%3A%2F%2Fwww.mongodb.com%2Ftry%2Fdownload%2Fcommunity&v=yiz5bmLVVhs)
* URL docs: <https://www.mongodb.com/docs/manual/>
* Copy the **Data directory** path during installation of MongoDB
  + exp: C:\Program Files\MongoDB\Server\5.0\bin\
* To check mongodb: Open cmd and run below command
  + "C:\Program Files\MongoDB\Server\5.0\bin\mongod.exe" --version
  + "C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe" --version
    - "C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe"
    - > show dbs

**Additional :Setup path for MongoDB in cmd:**

* Search env in local system
* Enviroment variable
* Path
* Edit
* New
* Paste path **“C:\Program Files\MongoDB\Server\5.0\bin”**
* Ok ok ok
* Open cmd
  + Mongo --version
  + Mongod –version
  + Mongo
  + Show dbs

**CREATE DATABASE USING CMD**

**Now we will first see the fundamental of mongodb and after connect with nodejs:**

* Open cmd
* "C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe"
* show dbs
* use firstdb
* db.firstdata.insertOne({name: "nodejs", type: "backend", videos: 80, active:true})
  + output:
  + **{**

**"acknowledged" : true,**

**"insertedId" : ObjectId("62b94b54ccb2c0ad69902a92")**

**}**

* show dbs
* db **“ to show active db”**
* show collections
* db.firstdata.find() **“to find documents and records”**
* db.firstdata.find().pretty()
* ctrl+c ctrl+c OR quit()

**CRUD OPERATION using CMD:**

**Insert document:**

* show dbs
* db.firstdata.insertOne({name: "mongodb", type:"database", videos: 5, active:true})
* db.firstdata.insertMany([{},{},{}])
  + db.firstdata.insertMany([{**name: "html", type:"database", videos: 5, active:true },{ name: "css", type:"database", videos: 5, active:true },{ name: "bootstrap", type:"database", videos: 5, active:true** }])
  + db.firstdata.find().pretty()

**READ (Q&A)**

* db.collection.find(query,projection)
  + **Find all the results of the given collection?**
    - show dbs
    - use firstdb
    - db
    - show collections
    - db.firstdata.find()
  + **Show the results in a pretty format?**
    - db.firstdata.find().pretty()
  + **Get only MongoDB data as an output ?**
    - db.firstdata.find({name:"mongodb"})
    - db.firstdata.find({name:"mongodb"}).pretty()
  + **Get the MongoDB data as an output with the only name field?**
    - db.firstdata.find({name:"mongodb"},{name:1}).pretty()
    - db.firstdata.find({name:"mongodb"},{name:0}).pretty()
  + **Get the MongoDB data without\_ID field in it?**
    - db.firstdata.find({name:"mongodb"},{ \_id:0, name:1}).pretty()
  + **Set the filter to “active: true” and get only the first field with the “active: true” value?**
    - db.firstdata.find({active:true}).pretty().limit(1)
  + **Do the same as 6 questions but with a different method?**
    - db.firstdata.findOne({active:true})
  + **Do the same as 6 questions but this time, get the 2nd field with active: true by skipping the 1st field?**
    - db.firstdata.find({active:true}).pretty().limit(1).skip(1)

**UPDATE (Q&A)**

* UpdateOne() => db.COLLECTION\_NAME.updateOne(<filter>,<update>)
* UpdateMany() => db.COLLECTION\_NAME.update(<filter>,<update>)
  + **Update the bootstrap type value to “full-Stack”?**
    - showdbs
    - use firstdb
    - db
    - show collections
    - db.firstdata.find().pretty()
    - db.firstdata.updateOne({name: "bootstrap"},{$set: {type:"fullstack"}})
  + **Update all the fields with the type value equal to “frontend” and set the value of status to false?**
    - db.firstdata.updateMany({type: "frontend"}, {$set: {active:false}})

The $set operator replaces the value of a field with the specified value.

**DELETE (Q&A)**

* **deleteMany => db.COLLECTION\_NAME.deleteMany(DELLETION\_CRITTERIA**
  + **Delete the field with the type that match ”frontend”**
    - show dbs
    - use firstdb
    - db
    - find collections
    - db.firstdata.find().pretty()
    - db.firstdata.deleteMany({type: "frontend"})
    - db.firstdata.find().pretty()

We also have remove() method to perform the delte opetation but it’s deprecated as per documents

* + **If we want to delete all the documents at once then we can pass an empty object inside the deleteMany() method.**
    - db.firstdata.deleteMany({})

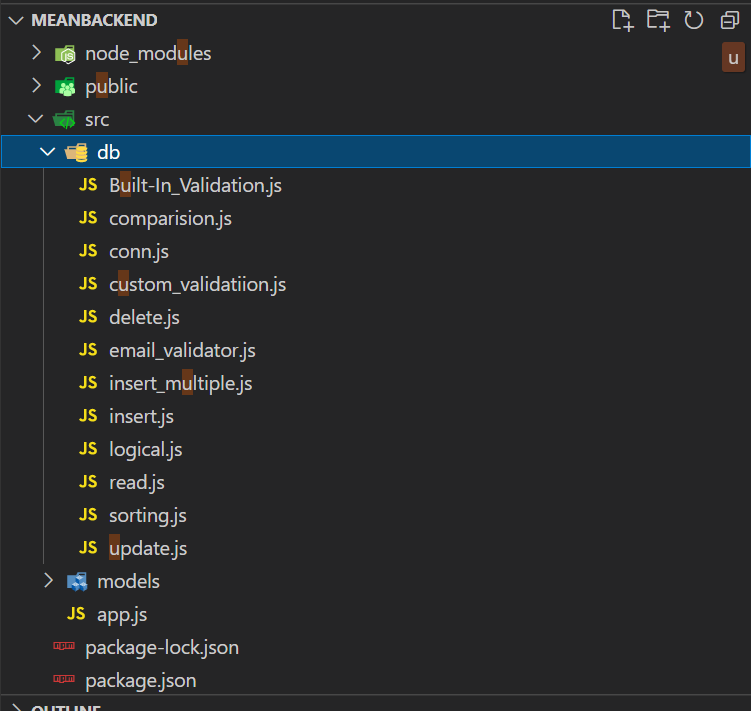
**MongoDB CRUD Operations using MongoDB Compass**

**Installation MongoDB GUI Compass**

* URL to download: <https://www.mongodb.com/products/compass>
* After installation click on **new connection** then **connect**

**Introduction to Mongoose ????**

**Folder Structure:**

****

**Install NPM :**

* npm init -y
* npm install mongoose
* npm i express
* npm I validator

**Connect NodeJS, Express to MongoDB using Mongoose:**

* **code:**

**app.js**

const express = require("express");

const path = require("path");

const app = express();

require("./db/conn");

require("./db/insert");

require("./db/insert\_multiple");

require("./db/read");

require("./db/comparision");

require("./db/logical");

require("./db/sorting");

require("./db/update");

require("./db/delete");

require("./db/Built-In\_Validation");

require("./db/custom\_validatiion");

require("./db/email\_validator");

const port  = process.env.PORT ||  3000;

app.listen(port, () => {

    console.log(`server is running ar port no ${port}`);

})

**conn.js**

const mongoose = require ("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb",{

    useNewUrlParser:true,

    useUnifiedTopology:true

    })

.then(() => {console.log("connection sussessful")})

.catch((err) => {console.log("no connection",err)});

**Create and Insert the Document using Express in MongoDB using Mongoose:**

**insert.js**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

// schema

// A Mongoose schema defines the struture of the document,

// default values, validatorrs, etc.,

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    ctype: String,

    videos: Number,

    author: String,

    active: Boolean,

    date: {

        type: Date,

        default: Date.now

    }

})

// A Mongoose model is a wrapper ont he Mongoose schema.

//  A mongoose schema defines the structure of the document.

// Create document of insert

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const result = await reactPlaylist.save();

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

createDocument();

**Insert Multiple Documents using One Line in Mongoose:**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

// schema

// A Mongoose schema defines the struture of the document,

// default values, validatorrs, etc.,

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    ctype: String,

    videos: Number,

    author: String,

    active: Boolean,

    date: {

        type: Date,

        default: Date.now

    }

})

// A Mongoose model is a wrapper ont he Mongoose schema.

//  A mongoose schema defines the structure of the document.

// Create document of insert

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const frontPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const result = await playlist.insertMany([reactPlaylist, frontPlaylist])

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

createDocument();

**Read or Querying the Documents using Mongoose in Express App:**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

// schema

// A Mongoose schema defines the struture of the document,

// default values, validatorrs, etc.,

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    ctype: String,

    videos: Number,

    author: String,

    active: Boolean,

    date: {

        type: Date,

        default: Date.now

    }

})

// A Mongoose model is a wrapper ont he Mongoose schema.

//  A mongoose schema defines the structure of the document.

// Create document of insert

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const frontPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const front2Playlist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const result = await playlist.insertMany([reactPlaylist, frontPlaylist, front2Playlist])

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

// createDocument();

const getDocument = async() =>{

// const result = await playlist.find();

// const result = await playlist.find({name:"node js"});

// const result = await playlist.find({name:"node js"}).select({name:1})

const result = await playlist.find({name:"node js"}).select({name:1}).limit(2);

console.log(result);

}

getDocument();

**MongoDB Comparison Query Operators using Mongoose and Node(Express JS) :**

**URL for many more Operators:** [**https://www.mongodb.com/docs/manual/reference/operator/query/**](https://www.mongodb.com/docs/manual/reference/operator/query/)

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

// schema

// A Mongoose schema defines the struture of the document,

// default values, validatorrs, etc.,

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    ctype: String,

    videos: Number,

    author: String,

    active: Boolean,

    date: {

        type: Date,

        default: Date.now

    }

})

// A Mongoose model is a wrapper ont he Mongoose schema.

//  A mongoose schema defines the structure of the document.

// Create document of insert

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const frontPlaylist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const front2Playlist = new playlist({

            name: "node js",

            ctype: "backend",

            videos: 50,

            author: "sahni",

            active: true

        })

        const result = await playlist.insertMany([reactPlaylist, frontPlaylist, front2Playlist])

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

// using comparision operator

const getDocument = async() =>{

const result = await playlist

// $eq is a equal to oprerator

.find({name : {$eq : "node js"}})

.select({name:1})

// .limit(2);

console.log(result);

}

getDocument();

**MongoDB Logical Query Operators using Mongoose and Node(Express JS) :**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    ctype: String,

    videos: Number,

    author: String,

    active: Boolean,

    date: {

        type: Date,

        default: Date.now

    }

})

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

const getDocument = async() =>{

const result = await playlist

// $and using logical operator

.find({$and : [{author: "savita"},{videos: 50}]})

.select({author:1})

// .limit(2);

console.log(result);

}

getDocument();

**MongoDB Sorting and Count Query Methods using Mongoose and Node(Express JS) :**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    type: String,

    videos: Number,

    active: Boolean

    })

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

const getDocument = async() =>{

const result = await playlist

// // using count

// .find({$and : [{author: "savita"},{videos: 50}]})

// .select({author:1})

// .countDocuments();

// using shoting method

.find({name: "node js"})

.select({author:1})

.sort({author : 1});

console.log(result);

}

getDocument();

**MongoDB Update the Documents using Mongoose in Express App:**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    type: String,

    videos: Number,

    active: Boolean

})

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

const updateDocument = async (\_id) => {

    try {

        const result = await playlist.findByIdAndUpdate({ \_id }, {

            $set: {

                name: "javascript"

            }

        }, {

            new: true,

            useFindAndModify: false

        }

        )

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

updateDocument("62bd5f6ff744e3c1688f2198");

# Delete the Documents using Mongoose in Express App

**:**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    type: String,

    videos: Number,

    active: Boolean

})

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

const deleteDocumet = async (\_id) => {

    try {

        const result = await playlist.deleteOne({\_id});

        console.log(result);

    }catch(err){

        console.log(err)

    }

}

deleteDocumet("62bdd2004a3ee2b64c1418ca");

# Delete the Documents using Mongoose in Express App: also=>

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    type: String,

    videos: Number,

    active: Boolean

})

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

const deleteDocumet = async (\_id) => {

    try {

        const result = await playlist.findByIdAndDelete({\_id});

        console.log(result);

    }catch(err){

        console.log(err)

    }

}

deleteDocumet("62bdd2004a3ee2b64c1418ca");

# Mongoose Built-In Validation using MongoDB in (NodeJS) Express App:

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true,

        unique: true,

        lowercase : true,

        trim: true,

        minlength: [2,"minimum 2 letters"],

        maxlength :30

    },

    ctype : {

        type : String,

        required : true,

        lowercase : true,

        enum:["frontend","backend","database"]

    },

    videos: Number,

    active: Boolean

})

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "my\_SQL",

            ctype: "database",

            videos: 50,

            author: "sahni",

            active: true

        })

        const result = await reactPlaylist.save();

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

createDocument();

# Own Custom Validation using MongoDB in (NodeJS) Express App:

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true,

        unique: true,

        lowercase : true,

        trim: true,

        minlength: [2,"minimum 2 letters"],

        maxlength :30

    },

    ctype : {

        type : String,

        required : true,

        lowercase : true,

        enum:["frontend","backend","database"]

    },

    videos :{

        type : Number,

        validate(value){

            if(value < 0){

                throw new Error("videos count should not be negative")

            }

        }

    },

    active: Boolean

})

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "no\_SQL",

            ctype: "database",

            videos: 10,

            author: "sahni",

            active: true

        })

        const result = await reactPlaylist.save();

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

createDocument();

# Using NPM Validator Package for Validation using MongoDB in (NodeJS) Express App :

URL to install validator NPM: <https://www.npmjs.com/package/validator>

const mongoose = require("mongoose");

const validator = require("validator");

mongoose.connect("mongodb://localhost:27017/firstdb", {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

    .then(() => { console.log("connection sussessful") })

    .catch((err) => { console.log("no connection", err) });

const playlistSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true,

        unique: true,

        lowercase : true,

        trim: true,

        minlength: [2,"minimum 2 letters"],

        maxlength :30

    },

    ctype : {

        type : String,

        required : true,

        lowercase : true,

        enum:["frontend","backend","database"]

    },

    videos :{

        type : Number,

        validate(value){

            if(value < 0){

                throw new Error("videos count should not be negative")

            }

        }

    },

    email : {

        type : String,

        unique : true,

        required: true,

        unique: true,

        validate(value){

            if(!validator.isEmail(value)){

                throw new Error ("This email is invaid");

            }

        }

    },

    active: Boolean

})

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

const createDocument = async () => {

    try {

        const reactPlaylist = new playlist({

            name: "Email\_\_",

            ctype: "database",

            videos: 10,

            author: "sahni",

            email: "aaa@gmail.com",

            active: true

        })

        const result = await reactPlaylist.save();

        console.log(result);

    } catch (err) {

        console.log(err);

    }

}

createDocument();

# REST API POSTMAN

Postman is an interactive and automatic tool for verifying the APIs of our Project.

Postman is a Google Chrome app for interacting with HTTP APIs.

It presents you with a friendly GUI for constructing requests and reading responses. It works on the backend and makes sure that each API is working as intended.