Documentation-

Google this-

node mongo db native

open first git hub link-

<https://github.com/mongodb/node-mongodb-native>

scroll down you will find 4 links for documentation. Open first link. Here you will see for each version there are 2links, - refrence and api. Refrence will help you in like crud operation,how to pass username and passoword while connecting. code to perform these operations. Second link will help you in api, like you want to see methods on cursur object.

in latest api docs, no examples are given if you want to see examples, see older version

61)Installing MongoDb and Robomogo

Go to mongo db website. There click on download button. Now go to community server option and from dropdown select **windows server 2008 R2 64-bit and later, with SSl support.** Then click on download msi file. after setup is downloaded open it.

During installtion there will be option of custom and complete installation. Click on custom option for a second, althrough we will following with complete installation. After clicking on custom option it will show where (location) it will be installed, it is important. It will be installed at-

C:\program files\mongodb\server\3.6\

This is important because we have to navigate to this directory in order to start mongo db server. Now go back and select complete option. Then click on next to complete installition.

Now it is installed we have to go to cmd and boot up the server. Git bash will not work for starting mongo db. open this path in cmd-

**C:\Program Files\MongoDB\Server\3.6\bin**

This directory uses whole bunch of executables thatw eare going to use to do things like start up server and connect to it. But first we have to create afolder where all of our data will be stored. We create folder named mongo-data in D drive. Now this is path that e have to specify when we run mongod command.

Now run this command-

**mongod.exe --dbpath D:\mongo-data**

your path may vary. If you see message –

**waiting for connections on port 27017**

then your mogo db server has started successfully. Now server is up lets run some commands. Open cmd and navigate to same path. Then run this-

**mongo.exe**

this is going to connect to our local mongo db database and it’s going to put us in some sort of command propt view of database. Here we can run commands to manipulate the data. Lets create new todo, run this-

**db.Todos.insert({text: 'first todo'})**

after running it, we should see this-

**WriteResult({ "nInserted" : 1 })**

It means one record is inserted.

if yu get any errors go through this link-

<https://stackoverflow.com/questions/41615574/mongodb-server-has-startup-warnings-access-control-is-not-enabled-for-the-dat>

lets run another command-

**db.Todos.find()**

here we will see our data. We get-

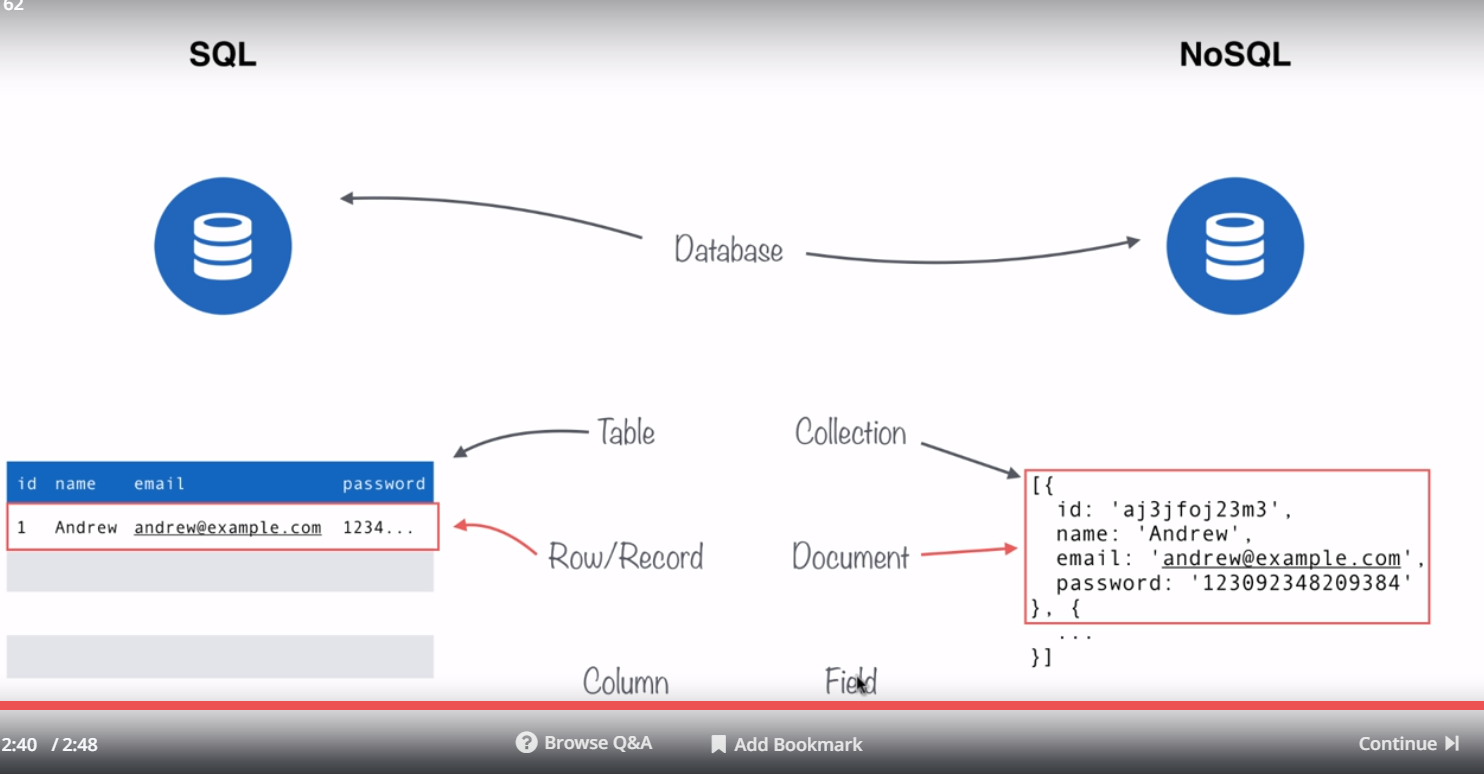
**{ "\_id" : ObjectId("5b2f81409461c6dba875acc5"), "text" : "first todo" }**

We also get id property, this is mongo db’s unique identifier. This si the property they use to give your document, (in this case Todo) a unique identifier.

We will also install robo mongo, it is gui for mongo db. install it and then open it.

You will be greated by connection screen. This screen lets you configure all the connections for robo mingo, tou might have local connection for your local database, you might have connection to real url where your production data is stored.we will get into all this alter. For now we will click on crete. Here you can give address, port and password etc. do create connection .

62)Building a NoSQL Vocabulary



On important thing is in sql database, al rows will have same colmns but in case of mongodb, different documents in a collections can have same or different properties.

63)Connecting to mongo and writing data

In next couple of videos we will code in playground folder.

To connect to mongo db inside of nodejs, we will be using npm module created by mongo db team. It’s called node-mongodb-native. Google it- node mongo db native.

Open first github link, scroll down , you will see some few important links like docs, api –docs etc. after tht you can find tons of exemples. We will go through lot of this stuff. But I want to tell you where you can find resources because mongo db library has tons of features.

There are entire courses dedicated just to mongo db and they don’t even begin to cover everything that’s built into this library. We are going to be focussing on important and common subset of mongo db that we need for node js applications. To start click on documentation link. Now in documentation click on your version and you will see documentation and api refrence. Lets write some code.

Create a new folder ,node-todo-api. Then run npm init to create package.json file. then run this-

**npm install mongodb –save**

lets use it. Code-

const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017';

const dbName = 'TodoApp';

MongoClient.connect(url, function (err, client) {

if (err) {

console.log('Error in connection');

} else {

console.log("Connected successfully to server");

const db = client.db(dbName);

client.close();

}

});

So lets insert some data. Code-

const MongoClient = require("mongodb").MongoClient;

const url =

"mongodb://sumeet:sood@localhost:27017/?authMechanism=SCRAM-SHA-1&authSource=TodoApp";

const dbName = "TodoApp";

MongoClient.connect(

url,

function(err, client) {

if (err) {

console.log("Error in connection");

} else {

// assert.equal(null, err);

console.log("Connected successfully to server");

const db = client.db(dbName);

db.collection("Todos").insertOne(

{ text: "Something to do", completed: false },

(err, result) => {

if (err) {

console.log("Unable to insert", err);

} else {

console.log(result.ops);

}

}

);

client.close();

}

}

);

Output-

**Connected successfully to server**

**[ { text: 'Something to do',**

**completed: false,**

**\_id: 5b2ff1a0791d7108e0997a10 } ]**

To access a collection we do not to create it first.

MongoClient takes 2 arguments first one is url where your database lives, in pridcution yhis can be amazon web services example or heroko url for our case it is going to be localhost url, second one is going to be callback. This cllback will be fired when connection has either succeeded or failed and then we can go ahead and handle things appropriately. If connection fails we will stop the program, if connection is made we can go ahead and manipulate the data.

Callback takes 2 objects, one is err and other is client. if connection is made we will use this client object to issue commands. After making connection, we call close method on client, this closes the connection with mongo db server.

In mongi db there is no need to create databse before connecting to it. But mongo will not create that databse until we start adding data to it. So lets add data. Code-

Like databse, for collections also we do not need to make collection before we start accessing it. We can start inserting data into it. insertOne is method avalaible on collection.it lets us insert new document into collection.it takes 2 arguments first one is object that you want to store, second one is callback function that will get fired when either things fail or go well. It has 2 arguments first is err which may or may not exist, second one is result which will be provided if data is inserted successfully.

In case of success we print result.ops, ops is going to store all docs that were inserted, it prints the array of documents that are inserted. In this case we are inserting only one.

In output we can see that \_id property is automatically generated by mongo. We will discuss it in detail later. For now just know that is unique identifier, it’s id given to just this document.

Go to this link to see docs-

64)The ObjectId

We saw whenever we insert a object into db, mongo db adds \_id property to this object. this property is unique for each object being added. It is not auto incrementing as we can see for sql databases. First one has id 1 and it keeps on increasing there after. Mongo does not use this apporch. Mongo was designed to scale out really easily . scaling out means that you add on more database servers to handle that extra load.

Imagine you have web app that gets about 200 users a day and your current servers are ready for that traffic. Then suddenly 10,000 people flood your site, with mongo db , it is really to kickup new servers to handle that extra load. When we use randomly generated id like this one , we do not need to constantly communicate with the other database servers to check what highest incrementing is. We are going to generate unique object id and use that for document’ unique identifier.

Now the object id is made up of few different things. Its is 12 byte vlue. First 4 bytes are timestamp. We will talk about it in second. that means that built into the data right here, we have a time stamp that refers to the moment in time, when id was created. That means in our documents we do not need to create that field, its already encoded in the ID.

The next 3 bytes, those are machine indentifier. That means if 2 computers generate object id’s their machine id’s is going to be different. This ensures that object id is different.

Next up we have 2 bytes, the process id , just another way to create unique identifier. Last up we have 3 byte counter. This is what my sql do. This is only 3 bytes of id.

Well objectId is default value for \_id, if nothing is provided. You can inded do what ever you like with this property. We can give value to \_id while inserting document.

db.collection('Users').insertOne({

\_id: 123,

name: 'Sumeet',

age: 25,

location: 'Pune'

}, (err, result) => {

if (err) {

return console.log('Unable to insert user', err);

}

console.log(result.ops);

});

This is perfectly valid. ObjectId is default way by which mongo db creates id’s but you can do anything you like for id creation.

Lets see what we can do with \_id with code. Earlier w esaid that timestamp is encoded in objectId. We can pull that out. Code-

const url =

"mongodb://sumeet:sood@localhost:27017/?authMechanism=SCRAM-SHA-1&authSource=TodoApp";

const dbName = "TodoApp";

MongoClient.connect(

url,

function(err, client) {

if (err) {

console.log("Error in connection");

} else {

console.log("Connected successfully to server");

const db = client.db(dbName);

db.collection('Users').insertOne({

name: 'Nitesh',

age: 26,

location: 'Kolkata'

}, (err, result) => {

if (err) {

return console.log('Unable to insert user', err);

}

console.log(result.ops[0].\_id);

console.log(JSON.stringify(result.ops[0].\_id.getTimestamp(),undefined,2));

});

client.close();

}

}

);

Here time is in gmt. getTimestamp() is a function that returns a date object.

Now we do not have to rely on mongo db to create our Objectid’s. inside of mongo db library, they actually give us function we can run to make an object id whenever we. Let do it. Up above, at very top of file we are going to change our import statement, to load something new of mongo db. we are going to do this using an ES6 feature known as object destructing. Object destructuting lets you pull out property from an object creating variables. Code-

let user = {name: 'Sumeet', age: 26};

let {name} = user;

here we have created name variable with value ‘sumeet’. ES destructing is fantastic way of creating variables from object properties. Now lets use destructing I our import statement. Code-

// const MongoClient = require("mongodb").MongoClient;

const { MongoClient, ObjectID } = require("mongodb");

here we pull one additional property. So here we create 2 variables. this ObjectID constructor function lets us make new object ID’s on the fly. We can do anything we like with them even if we are not using mongo db as our database. We can use it to uniquely identify things. Code-

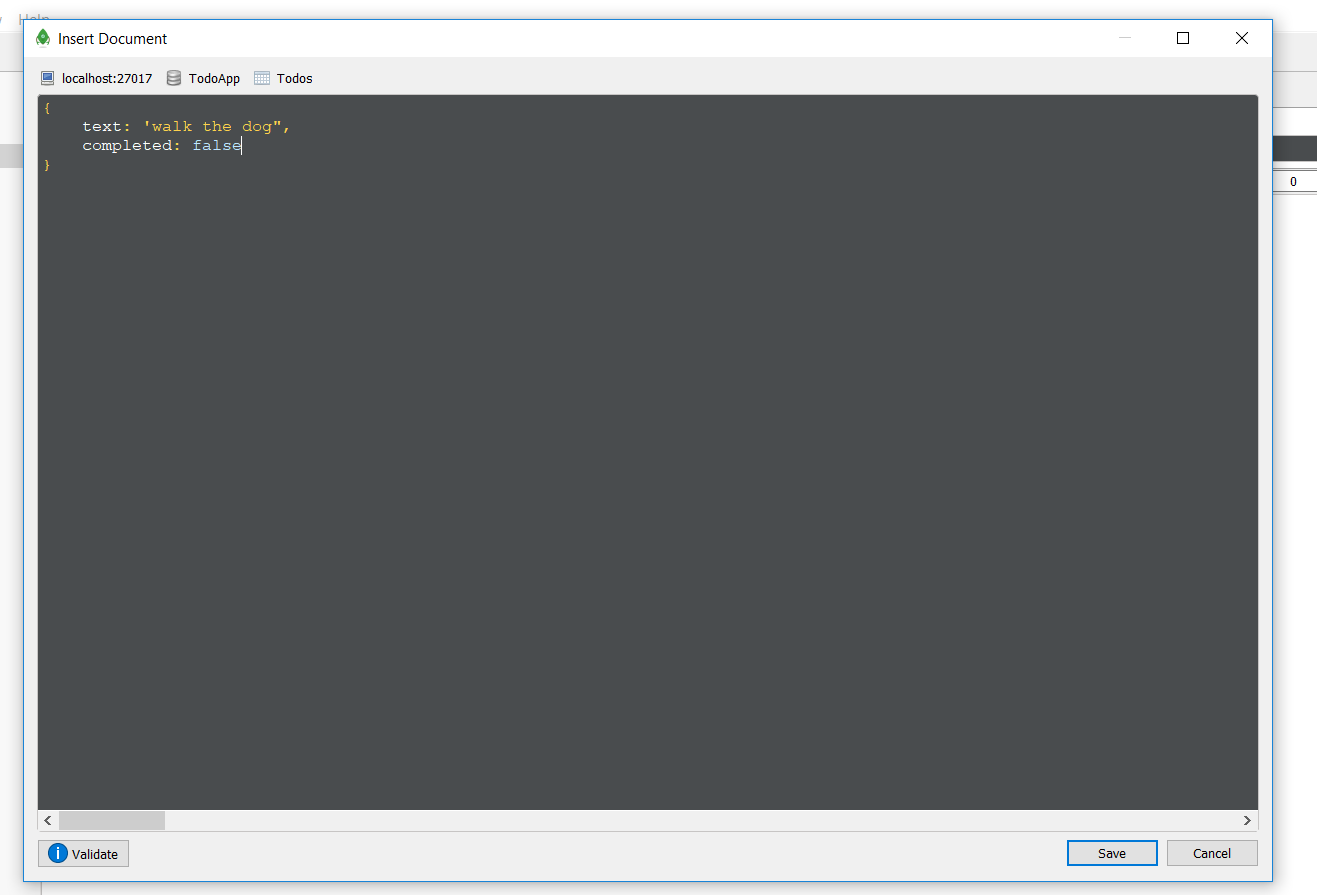
const { MongoClient, ObjectID } = require("mongodb");

let obj = new ObjectID();

here we created new instance of objectId. Each time we call constructor we get new ObjectId. We can mannualy assign them to \_id property of object, that we are going to save in database. We can let mongo db to create id’s for us. So we wnt be using this code. Comment it out.

65)Fetching Data

Create new file mongodb-find in playground folder. robomongo can delete, insert, update and read all of your documents. This makes it fantastic tool for debugging. Open collection(double click) on it. Then right click and select insert document. Insert json object.



We do not provide \_id, mongo db will do it. We call find method on collection to get records, by default it does not take any parameters. Which means get all documents

db.collection('Todos').find()

calling find is only first step. find returns the mongo db cursor, this cursor is not the actual document themselves. There could be couple of thousands and that would be really inefficient. It’s actually a pointer to those documents and cursors has ton of methods, we can use those methods to get our documents. We will method ToArray. Now instead of having a cursor we have an array of documents. This means we have an array of objects, they have ID property, text properties etc. this toArray gets us exactly what we wanted, we wanted documents. toArray returns the promise. That means we can call then method on it.

Code-

const { MongoClient, ObjectID } = require("mongodb");

const url =

"mongodb://sumeet:sood@localhost:27017/?authMechanism=SCRAM-SHA-1&authSource=TodoApp";

const dbName = "TodoApp";

MongoClient.connect(

url,

function (err, client) {

if (err) {

console.log("Error in connection");

} else {

console.log("Connected successfully to server");

const db = client.db(dbName);

db.collection('Todos').find().toArray().then(docs => {

console.log(docs);

}, err => {

console.log('Unable to ftech todos', err);

});

// client.close();

}

}

);

Output-

**Connected successfully to server**

**[ { \_id: 5b2ff1a0791d7108e0997a10,**

**text: 'Something to do 2',**

**completed: false },**

**{ \_id: 5b33da6cab14bdec01cd820c,**

**text: 'walk the dog',**

**completed: false } ]**

We have commented out close connection code, because it will interfere with our code above. As we are using promise.

Now this was very basic query regardless of whether or not it has certain values. Lets say we want to see records for which completed is true. Code-

db.collection('Todos').find({completed: false}).toArray().then(docs => {

console.log(docs);

}, err => {

console.log('Unable to ftech todos', err);

});

Lets say we want to find object by id. Lets way we have copied object’id from console. But this will not work-

db.collection('Todos').find({ \_id: '5b33da6cab14bdec01cd820c'}).toArray().then(docs => {

console.log(docs);

}, err => {

console.log('Unable to ftech todos', err);

});

This is because what we had inside \_id property is not string, it’s an object ID which means that we need to use this object id constructor, which we have used previously to create object id manually. This will work-

const { MongoClient, ObjectID } = require("mongodb");

db.collection('Todos').find({ \_id: new ObjectID('5b33da6cab14bdec01cd820c')}).toArray().then(docs => {

console.log(docs);

}, err => {

console.log('Unable to ftech todos', err);

});

There are methods other than toArray() that are avalaible on our cursors. Click on apiDocs page on git hub page. On the left side we have cursor. Click on it. Here you will find all methos. We will look at count. Here in official docs you can see that we need to provide a callback to count, but we we do not do that , then it returns a promise. Code-

db.collection('Todos').find().count().then(count => {

console.log(`Todos Count ${count}`);

}, err => {

console.log('Unable to ftech todos', err);

});

Output- **2**

toArray and count are subset of awesome functions avalaible to you.there are entire courses dedicated to mongo db and mongo db for node. Now you know where to find them.

66)Setting up the Repo

Here we will add version to our project. In this project we will create new repo locally, we will make a new git hub repository and we will push our code to that git hub repository.

First we need to initialize new git repository.

**git init**

this is going to initialize a new repository and I can always run and get status like this-

**git status**

to look at file that are untracked. Here we have node\_modules , we do not want to track this directory. This contains all of our npm libraries. To ignore this files, we will make .gitignore file in root of our project. This lets you specify files and folders that you want to leave out of version control.

See lecture for details

67)Deleting Documents

First we will create a new file named- mongodb-delete.js. then we have created some duplicate records in Todos collection. In this newly created file we will explore methods of deleting data. Here we will keep db.close method commented out as it is going to interfere with statements that we are going to write. Now there are 3 methods we will be using to remove data. These are-

1. deleteMany – it lets you delete many documents
2. deleteOne – you can delete one document
3. findOneAndDelete – it lets you remove individual items and it also returns those values.

Code-

const { MongoClient, ObjectID } = require("mongodb");

const url =

"mongodb://sumeet:sood@localhost:27017/?authMechanism=SCRAM-SHA-1&authSource=TodoApp";

const dbName = "TodoApp";

MongoClient.connect(

url,

function(err, client) {

if (err) {

console.log("Error in connection");

} else {

console.log("Connected successfully to server");

const db = client.db(dbName);

db.collection('Todos').deleteMany({text: 'Have Sex'}).then(result => {

console.log(result);

});

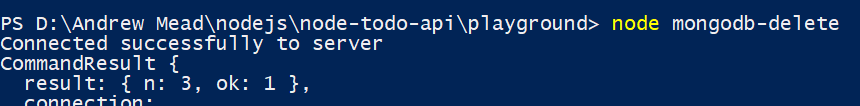
// client.close();

}

}

);

Output-



We added a then call to do something when it either succeed or fails. On result object we have property called result, it has 2 properties ok(value 1 means operation was successful) and n(it’s value indicates the number of records deleted).

now lets see deleteOne, it works exactly same as deleteMany, only difference is it only deletes the first item that matches criteria and then it stops.

Now lets see **findOneAndDelete** , most of the time we have id f document that I want to delete. In this I want to display the document that I have deleted. Another case can be we can show user that are you sure you want to delethe tis record, if user says no, then we can insert that document again in collection. Lets use it-

const { MongoClient, ObjectID } = require("mongodb");

const url =

"mongodb://sumeet:sood@localhost:27017/?authMechanism=SCRAM-SHA-1&authSource=TodoApp";

const dbName = "TodoApp";

MongoClient.connect(

url,

function(err, client) {

if (err) {

console.log("Error in connection");

} else {

console.log("Connected successfully to server");

const db = client.db(dbName);

// db.collection('Todos').deleteMany({text: 'Have Sex'}).then(result => {

// console.log(result);

// });

db.collection('Todos').findOneAndDelete({text: 'walk the dog'}).then(result => {

console.log(result);

});

// client.close();

}

}

);

Output-

**node mongodb-delete**

**Connected successfully to server**

**{ lastErrorObject: { n: 1 },**

**value:**

**{ \_id: 5b33da6cab14bdec01cd820c,**

**text: 'walk the dog',**

**completed: true },**

**ok: 1 }**

Instead of returning the result object, findOneAndDelete returns the deleted object.

Value object is object that is deleted. We will talk about lastErrorObject in a second. for now lastErrorObject has only 1 property called, n which tells the number of documents deleted.

68)Updating Data

First create a called mongodb-updte.js. now go to api docs. Click on collection, there you will see method findOneAndUpdate. It find a document, updates it and then returns that document. Lets use it.

Code-

const { MongoClient, ObjectID } = require("mongodb");

const url =

"mongodb://sumeet:sood@localhost:27017/?authMechanism=SCRAM-SHA-1&authSource=TodoApp";

const dbName = "TodoApp";

MongoClient.connect(

url,

function (err, client) {

if (err) {

console.log("Error in connection");

} else {

console.log("Connected successfully to server");

const db = client.db(dbName);

db.collection('Todos').findOneAndUpdate(

{ \_id: new ObjectID('5b33da6cab14bdec01cd820c')},

{$set: {completed: false}},

{returnOriginal: false}

).then(result => console.log(result));

}

}

);

First argument is user for filtering. Next argument is actual updates that we want to apply. This is not straight forward. What we have to do here is learn about the mongo-db update operators. Google it-

Mongo db update drivers

Open official mongodb link. Now this documentation is specific to mongo db which means it’s going to work with all of drivers. In our case it is nodejs driver. Scroll down, you will see all operators. The most important and one we are going to get started with is **$set** operator. It let us set the value of field in document. There are other operators like **$inc** which let u increment the value of field. Lets use $set. So type $set and set it equal to object and in this object,these are things that we are actually going to be setting. Like here we want to set completed equal to true. If we do this without $set operator-

db.collection('Todos').findOneAndUpdate(

{ \_id: new ObjectID('5b33da6cab14bdec01cd820c')},

{completed: true}

);

Document will not be updated. We have to use update operators. Then we provide third argument. In api docs we can see that there are lot of things we can configure . we are ging to set returnOrignal to false, so that we get updated document instead of original document.

The goal of this lecture is to let you search docs. We cannot cover everything here.

Now we want to change the name of user as well as increment its age by using $inc operator. I want you to dig into documentation. Code –

db.collection('Users').findOneAndUpdate(

{ \_id: new ObjectID('5b312584bd89642f6ca677a2')},

{$set: {name: 'sumit'}, $inc: {age: 5}},

{returnOriginal: false}

).then(result => console.log(result));

Please go ahead and explore all update operators.

69)The Mongoose ORM

In last several videos you learned how to use mongo db native driver and that let you perform crud operations. From here on , we are also going to use something called mongoose, mongoose which is an ORM(object relational mapping) is a fantastic NPM library. It makes it really easier to structure your data. For example in this app we are going to have Todo’s and users. Todos needs certain things, it needs certain properties of certain type, maybe we want to add some custom validation for them, for ex for user, we need to validate their password. All of this is lot easier with mongoose. Is it possible without mongoose? Yes it is. Native driver can do everything that mongoose can do, but you have to write a lot of boiler plate code.

70)Setting up mongoose

Now we will move out from playground folder. also clear the database, because the data inside of here is going to be a little different. just drop the database. The mongo db will automatically start create database once you start writing data to it. Go to official docs on mongoose.js.

Here you can find examples, guides, a full list of plugins , a ton of great resources. Clikc on read the docs. Here you will find everything . it is fantastic tool. If you want to use some feature that we do not cover here, visit the docs and you will find that. First install library-

**npm install mongoose@latest –save**

now create a folder named server. everything about server will be in this folder. firstv we create server.js. it is root of our app. To run this app , you have to run this file. code-

const mongoose = require('mongoose');

mongoose.Promise = global.Promise;

mongoose.connect("mongodb://localhost/TodoApp");

mongoose supports callbacks by default but callbacks really are’nt how I like to program. I like promises as they are lot simpler to chain, manage and scale. before connection we tell mongoose that we want to built in promise library, we do not want to use some third party library.

mongoose.Promise = global.Promise;

when making connection with native driver we passed a callback. In that callback we wrote statements that query database. In case of mongoose, next lines will be executed onl when connection is made. Mongoose is complex which is good , it means our code can be simpler. We don’t have to micromanage the order in which things happen. Mongoose will tke care ot it.