MongoDB and
Building REST API using MongoDB

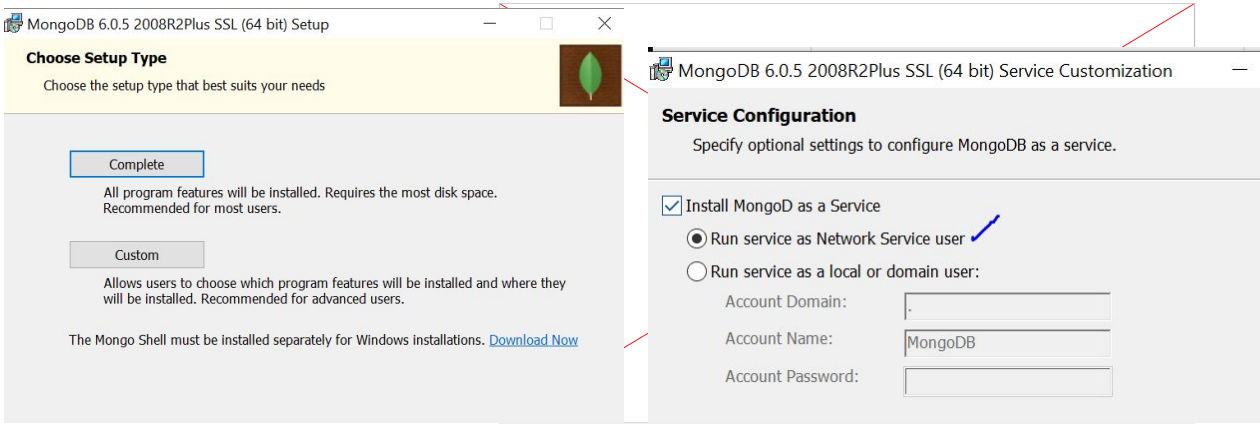
Module 4

Installation: MongoDB Compass

- Download

  https://www.mongodb.com/try/download/community

- Run Setup



- Set env PATH

  C:\Program Files\MongoDB\Server\5.0\bin

- MongoDB is a Database application
- NoSQL used for high volume storage
- It is a Collection and Document based
  - NoSQL database, doesn’t follow a schema
  - Documents contain key (field)-value pair
  - Collection is a set of Documents and functions

SQL(MySQL) VS NoSQL(MongoDB)

1. RDBMS ia a relational DB and works on Relational Database.
2. It stores data in the form of entity as table.
3. To perform CURD operation, SQL is used to query DB.
4. Tables, rows, columns

1. Non Relational, Document Oriented DBMS and works on Document Databases.
2. Stores data in the form of Documents
3. Uses BSON to query DB
4. Collection, Document, Field

MongoDB Features

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages.
3. The rows (or documents as called in MongoDB) doesn’t need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

WireTiger

- WiredTiger storage engine in MongoDB plays a vital role by providing
  - efficient storage
  - document-level concurrency control
  - ACID compliance
  - Compression
  - Encryption
  - Scalability.

JSON vs BSON

	JSON	BSON
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary
Readability	Human and Machine	Machine Only

What are alternatives to MongoDB?

- Cassandra,
- CouchDB,
- Redis,
- Riak

are a few good alternatives

First MongoDB Script

```
> var hello='good morning';
> printjson(hello);
"good morning"
>
```

Create database, collections, documents

Start cmd prompt

Run **mongosh** as command

Run **show dbs** to check default databases

1. Create new database-
2. **use db1** ....this command creates DB if it is not already exists, and let you start working if it does exists.
3. **show dbs** ...wont show up created DB if it is empty.
4. To add document
5. To add collection.

- Default database
  - Displays all database

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

- Find current DB

```
d15a> db
d15a
```

- Create Collection (new or switches)
  - Empty dbs are not listed by show
  - Db names are case sensitive

```
test> use d15a
switched to db d15a
```

- Show collections in the current DB

```
d15a> show collections
student
```

CRUD operations

- Insert 1 document

```
d15a> db.student.insertOne({fname:"urvi",lname:"pandit"})
```

  - Collection is not Schema dependent
  - Documents with different number and type of fields can be inserted

REFERS TO CURRENT db

NAME OF THE DOCUMENT

FIELD

VALUE

```
> db.regdata.insertOne({fname:"grinal", lname:"tus", no:26})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("61ae48b4f71fc45a59be2f00")
}
> db.regdata.insertOne({fname:"glad", lname:"mene", no:22, Pid:123})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("61ae48fff71fc45a59be2f01")
}
>
```

ONE ROW WITH THREE COLUMNS ARE ADDED

PRIMARY KEY

- Insert Many documents

db.collection.insertMany()

```
> db.regdata.insertMany([ {fname:"cayden", lname:"tusc", no:20}, {fname:"dielle", lname:"tusca", no:41}, {fname:"merwi
, lname:"tuscan", no:32} ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("61af90c111004965b3e120cd"),
                ObjectId("61af90c111004965b3e120ce"),
                ObjectId("61af90c111004965b3e120cf")
        ]
}
>
```

- View document inserted

db.collection.find()

CURRENT ACTIVE DATABASE

```
> db
registration
> show collections
regdata
> db.regdata.find()
{ "_id" : ObjectId("61ae48b4f71fc45a59be2f00"), "fname" : "grinal", "lname" : "tus", "no" : 26 }
{ "_id" : ObjectId("61ae48fff71fc45a59be2f01"), "fname" : "glad", "lname" : "mene", "no" : 22, "Pid" : 123 }
>
```

AVAILABLE COLLECTIONS IN CURRENT DATABASE.

AVAILABLE DOCUMENTS IN COLLECTION

- Search

```
d15a> db.student.find({fname:"meetali"})
[
  {
    _id: ObjectId("64102d1043c86ff3c6eaede5"),
    fname: 'meetali',
    lname: 'patil',
    rollno: 12
  }
]
```

- Find collection with specific field only

```
d15a> db.student.find({fname:"meetali"},{fname:1})
[ { _id: ObjectId("64102d1043c86ff3c6eaede5"), fname: 'meetali' } ]
```

- Without _id

```
d15a> db.student.find({fname:"meetali"},{_id:0,fname:1})
[ { fname: 'meetali' } ]
```

- What happens here?

```
d15a> db.student.find({ fname: "meetali" }, { _id: false, lname: 0 })
[ { fname: 'meetali', rollno: 12 } ]
```

- Update

UpdateOne()=>db.collection.updateOne(<filter>, <update>)

UpdateMany()=>db.collection.update(<filter>, <update>)

```
d15a> db.student.updateOne({fname:"vanita"},{$set:{lname:"dariyani"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
d15a> db.student.findOne({fname:"vanita"})
{
  _id: ObjectId("64102d1043c86ff3c6eaede4"),
  fname: 'vanita',
  lname: 'dariyani'
}
d15a> db.student.updateOne({fname:"meetali"},{$set:{lname:"C Patil"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
d15a> db.student.findOne({fname:"meetali"})
{
  _id: ObjectId("64102d1043c86ff3c6eaede5"),
  fname: 'meetali',
  lname: 'C Patil',
  rollno: 12
}
```

- UpdateMany

```
d15a> db.student.updateMany({} ,{$set:{classname:"D15A"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

```
  {
    _id: ObjectId("64102d1043c86ff3c6eaede3"),
    fname: 'urvi',
    lname: 'pandit',
    classname: 'D15A'
  },
  {
    _id: ObjectId("64102d1043c86ff3c6eaede4"),
    fname: 'vanita',
    lname: 'dariyani',
    classname: 'D15A'
  },
  {
    _id: ObjectId("64102d1043c86ff3c6eaede5"),
    fname: 'meetali',
    lname: 'C Patil',
    rollno: 12,
    classname: 'D15A'
  }
]
```

- Delete

```
d15a> db.student.deleteMany({fname:"urvi"})
{ acknowledged: true, deletedCount: 2 }
d15a> db.student.find()
[
  {
    _id: ObjectId("64102d1043c86ff3c6eaede4"),
    fname: 'vanita',
    lname: 'dariyani',
    classname: 'D15A'
  },
  {
    _id: ObjectId("64102d1043c86ff3c6eaede5"),
    fname: 'meetali',
    lname: 'C Patil',
    rollno: 12,
    classname: 'D15A'
  }
]
```

- Delete all documents

```
d15a> db.student.deleteMany({})
{ acknowledged: true, deletedCount: 2 }
d15a> db.student.find()

d15a>
```

Key Components of MongoDB Architecture

_id

- This is a field required in every MongoDB document.
- The _id field **represents a unique** value in the MongoDB document.
- The _id field is like the **document's primary key**.
- If you create a new document without an _id field, MongoDB will automatically create the field.
- No two documents will have same _id

Collection

- This is a grouping of MongoDB documents.
- A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL.
- A collection exists within a single database.

Database

- This is a container for collections like in RDMS wherein it is a container for tables.
- Each database gets its own set of files on the file system.
- A MongoDB server can store multiple databases.

Document

- A record in a MongoDB collection is basically called a document.
- The document, in turn, will consist of field name and values.

Field

- A name-value pair in a document.
- A document has zero or more fields.
- Fields are analogous to columns in relational databases.

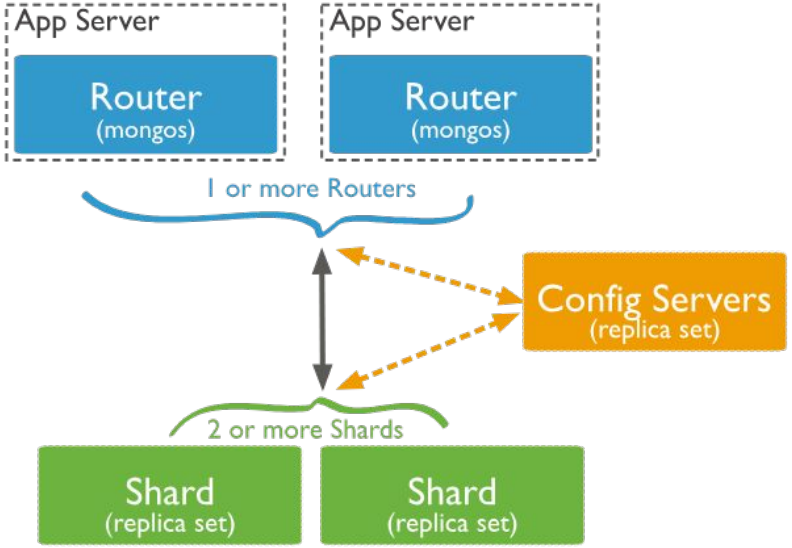
Why use MongoDB

- Document-oriented
  - Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very **flexible and adaptable** to real business world situation and requirements.
- Ad hoc queries
  - MongoDB supports searching by field, range queries, and regular expression searches. **Queries** can be made to return specific fields within documents.

- Replication
  - MongoDB can provide **high availability** with replica sets.
  - A replica set consists of two or more mongoDB instances.
  - Each replica set member may act in the role of the primary or secondary replica at any time.
  - The primary replica is the main server which interacts with the client and performs all the read/write operations.
  - The Secondary replicas maintain a copy of the data of the primary using built-in replication.
  - When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

- Indexing
  - Indexes can be created to improve the performance of searches within MongoDB.
  - Any field in a MongoDB document can be indexed.
- Load balancing
  - MongoDB uses the concept of **sharding** to scale horizontally by splitting data across multiple MongoDB instances.
  - MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

- Sharding
  - Supports Sharding which is the process of dividing large datasets across multiple collections to ensure that queries can be executed efficiently.
  - Splitting up large Collections into Shards allows MongoDB to execute queries without putting much load on the Server.
  - MongoDB Sharding can be implemented by creating a Cluster of MongoDB Instances.



Data Types in MongoDB

- String − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- Integer − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- Boolean − This type is used to store a boolean (true/ false) value.
- Double − This type is used to store floating point values.
- Min/ Max keys − This type is used to compare a value against the lowest and highest BSON elements.
- Arrays − This type is used to store arrays or list or multiple values into one key.

- Timestamp − ctimestamp. This can be handy for recording when a document has been modified or added.
- Object − This datatype is used for embedded documents.
- Null − This type is used to store a Null value.
- Symbol − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- Date − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- Object ID − This datatype is used to store the document’s ID.
- Binary data − This datatype is used to store binary data.
- Code − This datatype is used to store JavaScript code into the document.
- Regular expression − This datatype is used to store regular expression.

Operators

Comparison Operators:

- `$eq`: Matches values that are equal.
- `$ne`: Matches values that are not equal.
- `$gt`, `$lt`, `$gte`, `$lte`: Greater than, less than, greater than or equal to, less than or equal to.

Logical Operators:

- `$and`, `$or`, `$not`: Logical AND, OR, NOT.

Element Operators:

- `$exists`: Matches documents that have the specified field.
- `$type`: Matches documents based on the BSON type.

Array Operators:

- `$in`: Matches any of the values specified in an array.
- `$all`: Matches arrays that contain all elements specified in an array.

Text Search:

- `$text`: Performs a text search.

Regular Expression:

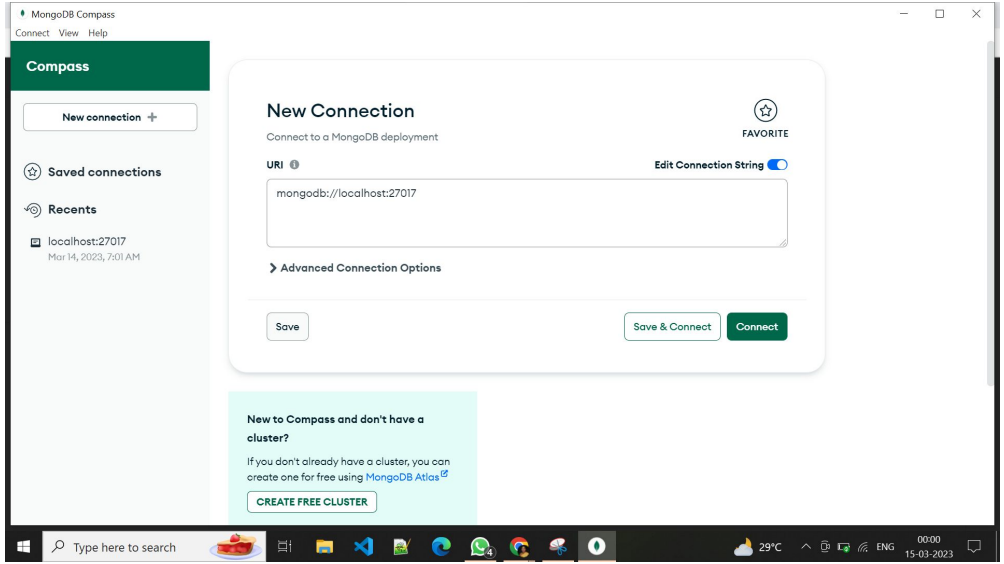
- `$regex`: Matches documents based on regular expression patterns.

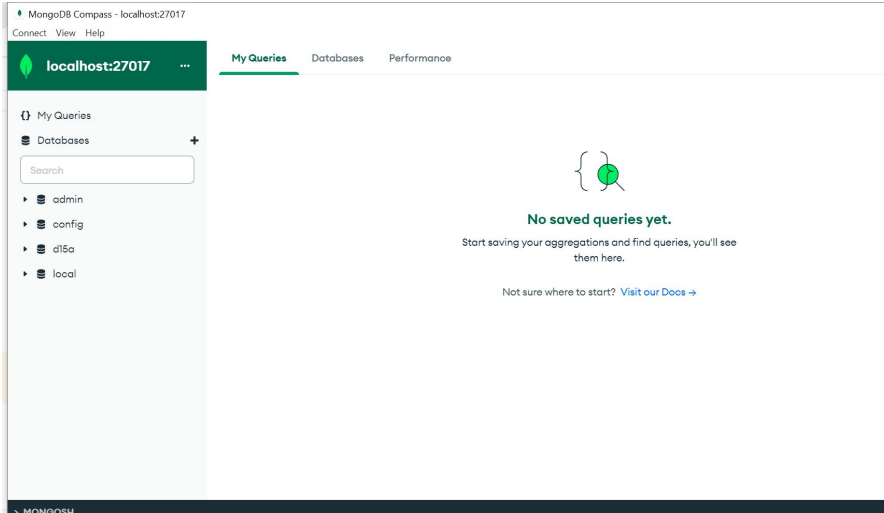
Arithmetic Operators:

- `$add`, `$subtract`, `$multiply`, `$divide`: Performs arithmetic operations.

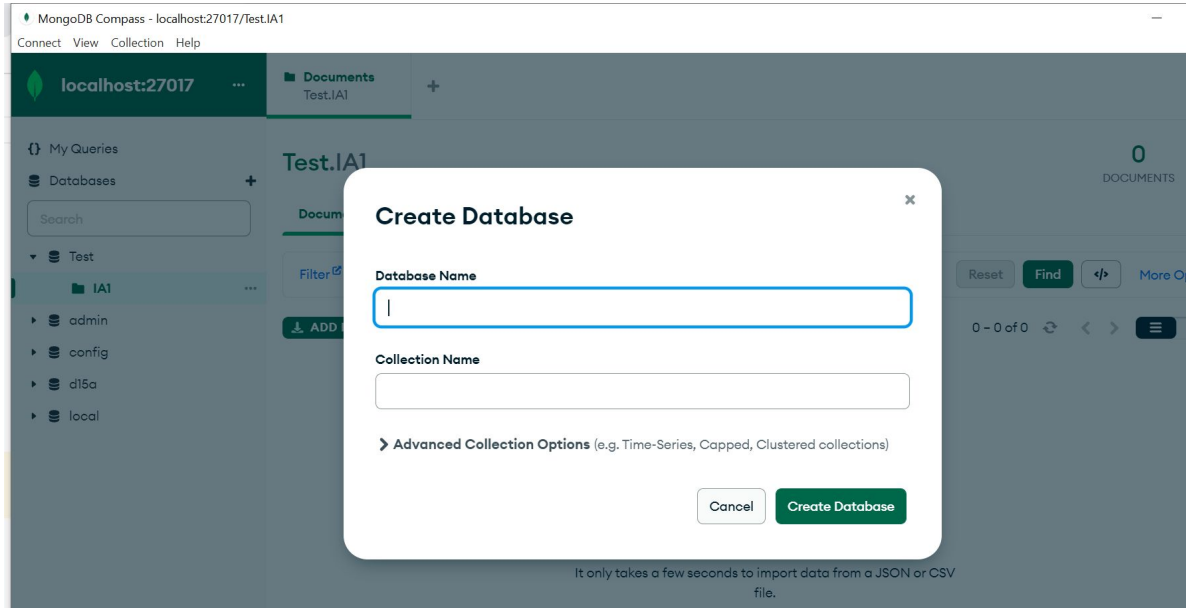
MongoDB Compass (GUI)

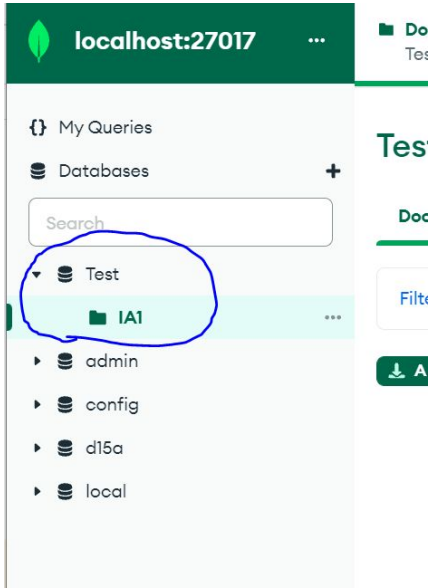
Connect



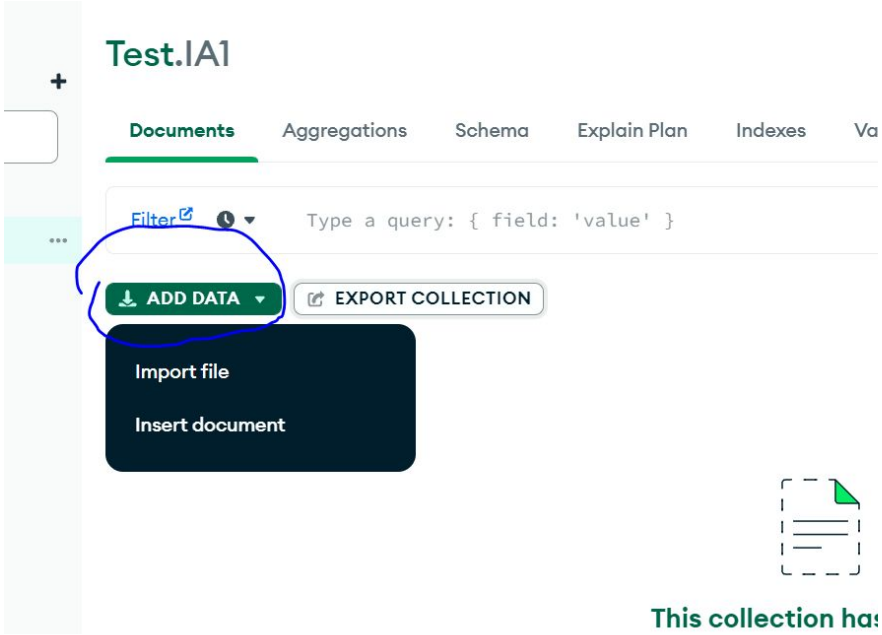


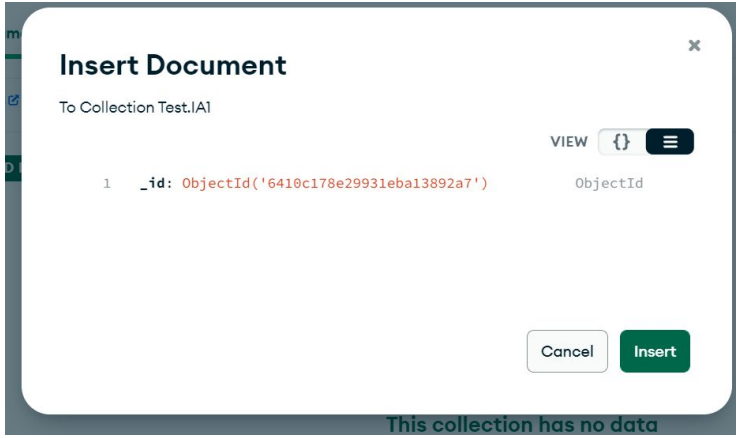
- Create Database

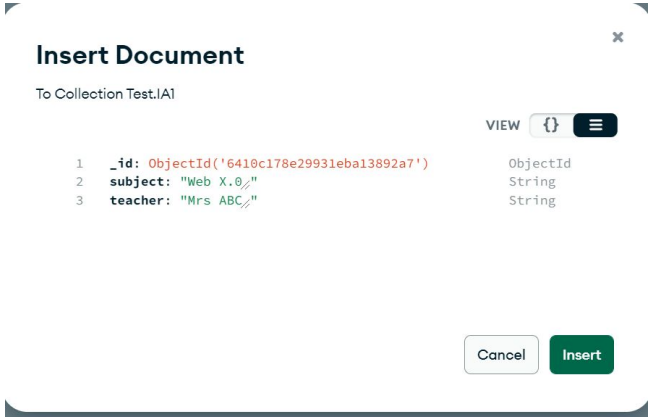




- Insert documents

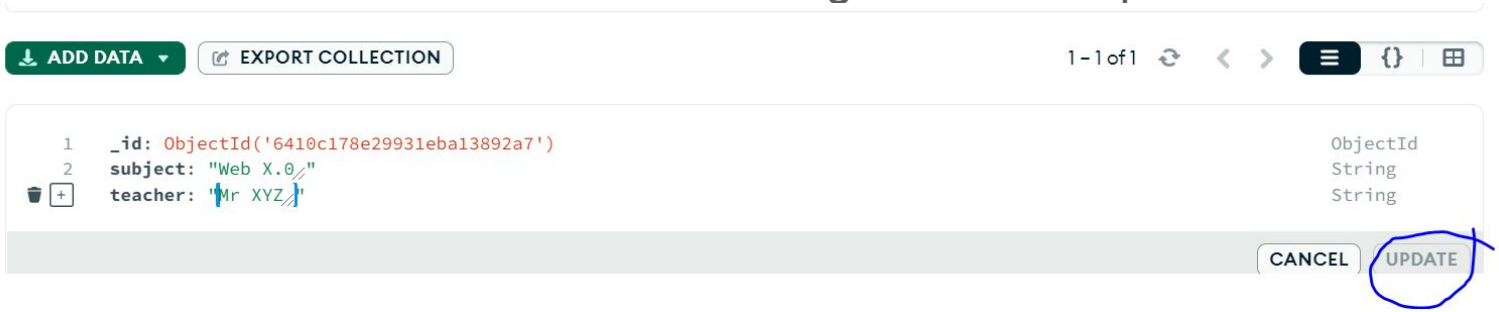




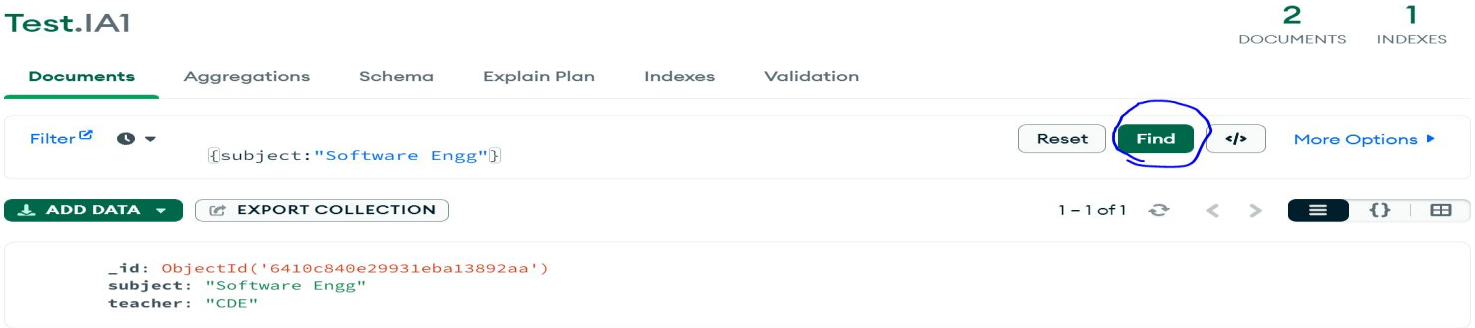


- Update

  Double click on the document to change and click update



- Search



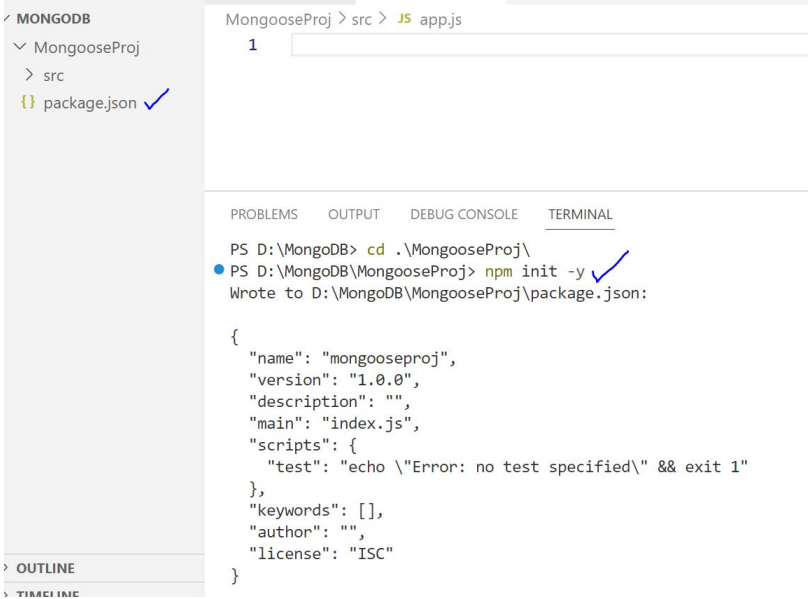
Moongose

- Mongoose: Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment.
- Mongoose helps in establishing connection between MongoDB and Node JS.
- Mongoose provides a straight-forward, schema-based solution to model your application data.
- It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

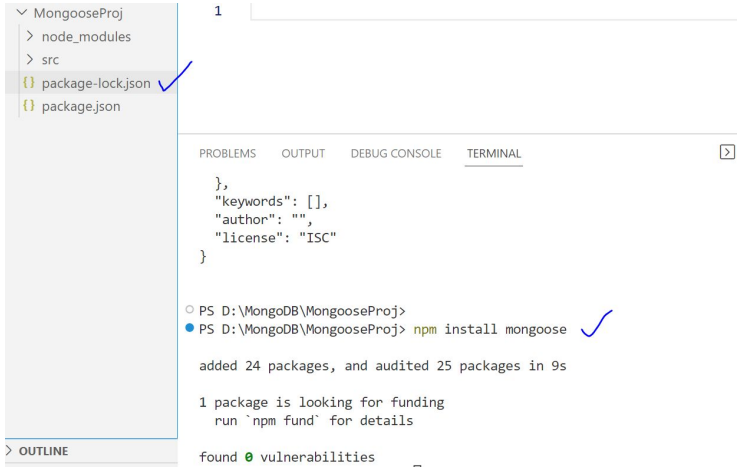
Checking for dependencies

Setting up Mongoose

1. npm init -y



1. npm install mongoose



- To create and connect to DB, define schema and create collections using mongoose models

Code to write inside app.js file

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

mongoose.connect("mongodb://localhost:27017/chanell1", {useNewUrlParser:true,
useUnifiedTopology:true }  )

.then( () => console.log("connection successful..."))

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

To connect mongoose with local DB

Local DB name, DB gets created if does not exists

To write o/p on console

Connect to server

```
PS D:\MongoDB\MongooseProj> node src/app.js
connection successful...
```

- Schema and Models in mongoose
  - Schema defines the structure of the document, default values and validators etc..
  - Eg: schema defines data type of key value pair.

Add to app.js

```
const listSchema=new mongoose.Schema({
    fname:String,
    lname:String,
    no:Number,
    date:{
        type:Date,
        default:Date.now
    },
    pid:{
        type:"Number"
        Required:true
    },
    submitStatus:Boolean
})
```

1. Create an instance of mongoose
2. Set data type for fname field
3. lname:String,
4. no:Number,
5. Create object inside date field
6. Make pid field compulsory

- Model
  - Mongoose model provides an interface to the database for querying, Creating, updating, deleting records etc..

```
//create a class for mongoose model

// class name to be written keeping first letter capital

const List1 = new mongoose.model("List1", listSchema);
```

Class name

Schema name

- Create multiple collections using mongoose model

```
List1.insertMany([d1,d2,d3]);
```

Name of a class

Class Method

Names of document

- Read documents using mongoose

```
const getDoc = async () =>{
    const r = await List1.find();
    console.log(r);
}
getDoc();  //calling a function give any name
```

Fat arrow

method

Name of collection

Name of a function

- # Read a specific documents using mongoose

```
const getDoc = async () =>{
    const r = await
List1.find({fname:"grinal2"});
    console.log(r);
}
getDoc();
```

Inside {} write key value

```
const getDoc = async () =>{
    const r = await List1.find({fname:"dielle"})
    .select({pid:1});
    console.log(r);
}
getDoc();//calling a function give any name
```

```
const getDoc = async () =>{
    const r = await List1.find({fname:"dielle"})
    .select({pid:0});
    console.log(r);
}
getDoc();//calling a function give any name
```

- Select _ exclude field

```
const getDoc = async () =>{
    const r = await List1.find({fname:"dielle"})
    .select({pid:0});
    console.log(r);
}
getDoc();//calling a function give any name
```

limit()

```
const getDoc = async () =>{
    const r = await List1.find({fname:"grinal2"})
    .select({pid:0})
    .limit(2);
    console.log(r);
}
getDoc();//calling a function give any name
```

- Display all documents with only pid that has ‘no’ greater than 452

```
const getDoc = async () =>{
    const r = await List1.find({no : {$gt : 453}})
    .select({pid:1})
    //.limit(2);
    console.log(r);
}
getDoc();//calling a function give any name
```

- Express with Mongo to update documents

```
const updateDoc = async (_id) =>{
    try{
        const result = await List1.updateOne({_id}, {
            $set : {
                fname : "GRINAL"
            }       });
        console.log(result);
    } catch (err){
        console.log(err);
    } }
updateDoc("61b1d5f304dc55af0e91647d");
```

```
PS C:\Users\91992\dockerexp\mongooseProj> node src/app.js
connection successful...
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

```
const updateDoc = async (_id) =>{
    try{
        const result = await List1.findByIdAndUpdate({_id}, {
            $set : {                fname : "GRINAL"
            }        });
        console.log(result);
    } catch (err){        console.log(err);
    }
 } updateDoc("61b1d5f304dc55af0e91647d");//calling a function with id as an arg
```

```
PS C:\Users\91992\dockerexp\mongooseProj> node src/app.js
connection successful...
{
  _id: new ObjectId("61b1d5f304dc55af0e91647d"),
  fname: 'GRINAL',
  lname: 'tuscano',
  no: 45,
  pid: 1234,
  submitStatus: true,
  date: 2021-12-09T10:09:55.729Z,
  __v: 0
}
```

- Delete one document

```
const deleteDoc = async (_id) =>{
    try{
        const result = await List1.deleteOne({_id});
        console.log(result);
    } catch(err){
        console.log(err);
    }
}deleteDoc("61b1d4dfeb39b70e33d99615");
```

REST API

```
PS D:\MongoDB\REST API> npm init

PS D:\MongoDB\REST API> npm i express
added 57 packages, and audited 58 packages in 3s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\MongoDB\REST API> npm i mongoose

added 24 packages, and audited 82 packages in 7s

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\MongoDB\REST API>
```

```
const express = require('express')
const app = express()
const mongoose = require('mongoose')

app.listen(3000, () => console.log('Server Started'))
```

```
PS D:\MongoDB\REST API> node server.js
Server Started
```

Administering User Accounts

Configuring Access Control

What is AJAX? Explain Working of AJAX in detail.

Understanding MongoDB,, Administering User Accounts, Configuring Access Control,

https://www.guru99.com/mongodb-create-user.html

https://www.javatpoint.com/mongodb-user-management-methods

REST API: Examining the rules of REST APIs, Evaluating API patterns, Handling typical CRUD functions (create, read, update, delete), Using Express and Mongoose to interact with MongoDB, Testing API endpoints

https://www.youtube.com/watch?v=fgTGADljAeg

https://www.youtube.com/watch?v=eYVGoXPq2RA

https://www.youtube.com/watch?v=WDrU305J1yw