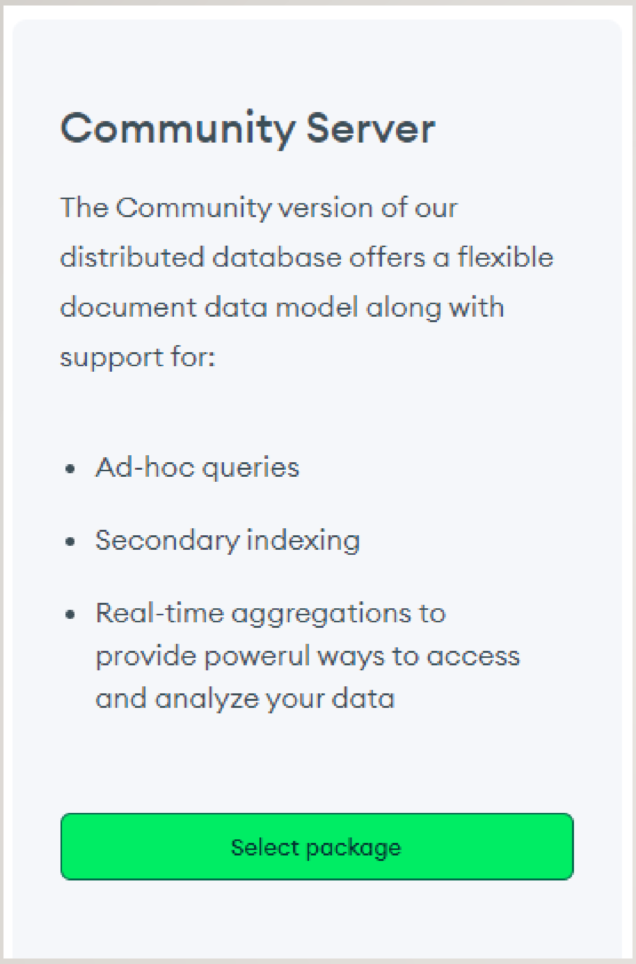
MongoDB

- MongoDB is like a big digital filing cabinet for storing information. Instead of storing data in tables like a traditional database, MongoDB stores data in collections of documents, which are like digital folders.
- Each document is like a file in a folder and can contain different types of information, such as text, numbers, lists, or even other documents. These documents are stored in a format called BSON, which is similar to JSON.
- One of the cool things about MongoDB is that it's flexible and can handle data that changes over time. You don't need to have a fixed structure for your data like you do in a traditional database. This makes it great for storing things like user profiles, product catalogs, or any other kind of dynamic data.
- Plus, MongoDB is designed to be really fast and scalable, meaning it can handle lots of data and lots of users without slowing down. So whether you're building a small app or a huge website, MongoDB can help you store and manage your data efficiently.

INSTALLATION

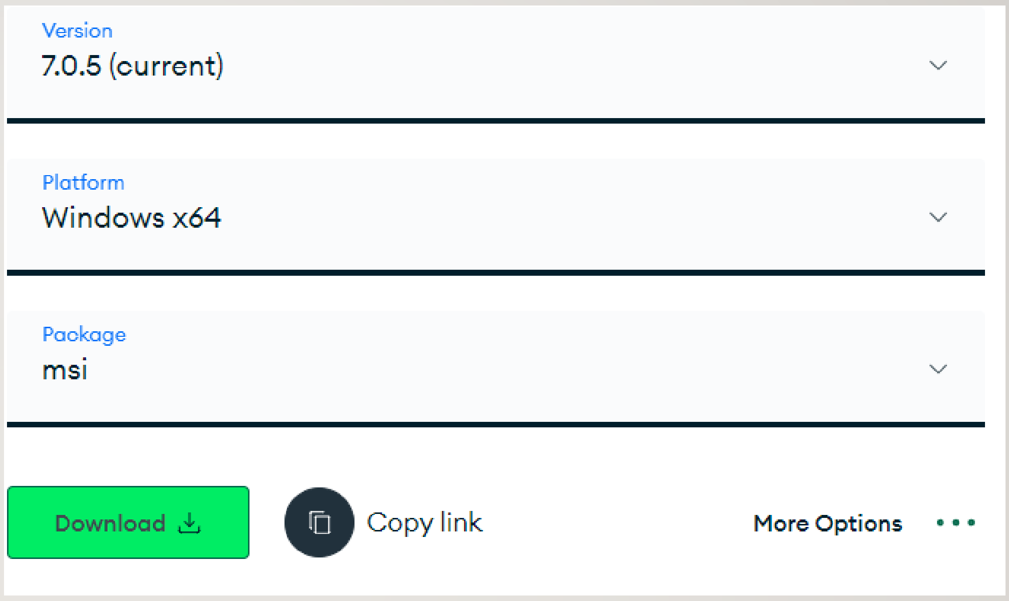
1. Go to search engine and type Mongodb community server
2. Click on first link
3. Now select package



Community Server

The Community version of our distributed database offers a flexible document data model along with support for:

- Ad-hoc queries
- Secondary indexing
- Real-time aggregations to provide powerul ways to access and analyze your data



Version: 7.0.5 (current)

Platform: Windows x64

Package: msi

SOME BASIC POINTS REALTED TO MONGODB

- In mongodb the folder we create called collection
- And the file that we create inside the folder called database

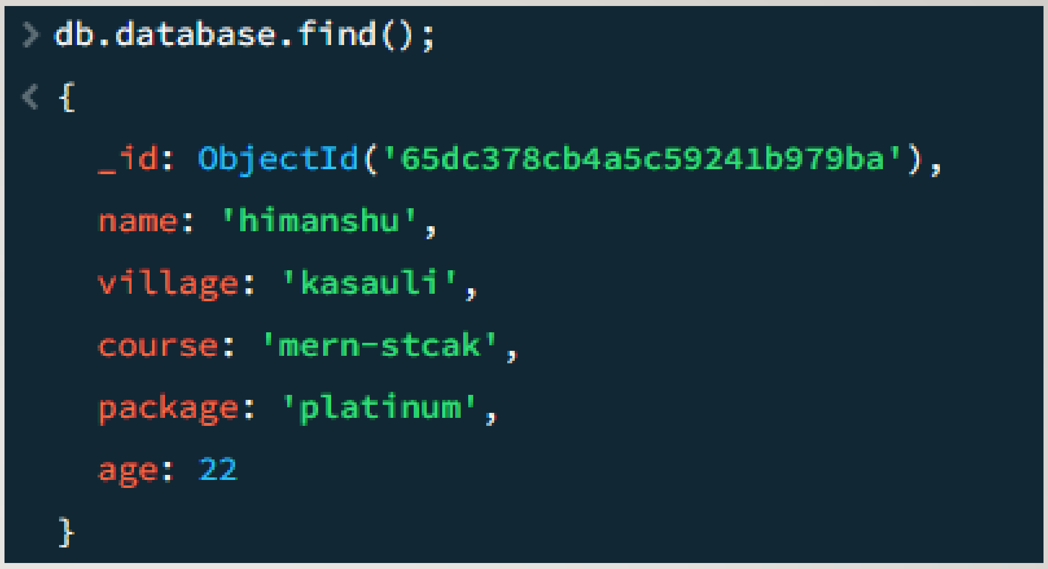
Diff. type of methods

FIND METHOD

The find() method is used to fetch a particular data from the table. In other words, it is used to select data in a table. It is also used to return all events to the selected data.

SYNTAX

db.(collection name).find();



```
> db.database.find();
< {
    _id: ObjectId('65dc378cb4a5c59241b979ba'),
    name: 'himanshu',
    village: 'kasauli',
    course: 'mern-stcak',
    package: 'platinum',
    age: 22
  }
```

INSERT METHOD

- To add a single data or insertion

SYNTAX

```
db.(collection name).insert ({Name: "Akshay", Marks: 500})
```

- To add a multiple data or insertion

SYNTAX

```
db.(collection name).insert([{Name: "Bablu", Marks: 550}, {Name:
"Chintu", Marks: 430}, {Name: "Devanshu", Marks: 499})
])
```

- Insert a document using _id field

SYNTAX

```
db.(collection name).insert({_id: 102,Name: "Anup", Marks: 400})
```

db.users.find({ age: { $gt: 20 } })   this is to find number greater than 20 you can change the value as your requirement

DELETEONE & DELETEMANY

SYNTAX of DELETING ONE OBJECT

db.(collection name).deleteone({name:"parth"})

This is where you target the object
Which you want to delete

SYNTAX of DELETING MANY OBJECT

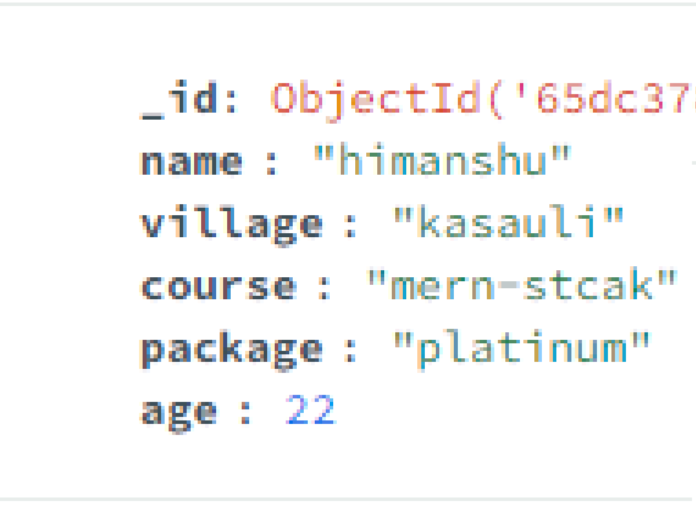
To delete many object together is quit tricky lets understand firstly target those id's which have common suppose there is a age section having the same age of 20 when you target the age all the id's having 20 age get targeted now the interesting part is the value you what to delete add that in the syntax suppose you have to delete the product but we have to remember that only those product gonna delete which is common in the target id not product but that whole id which is targeted other will remain same

SYNTAX

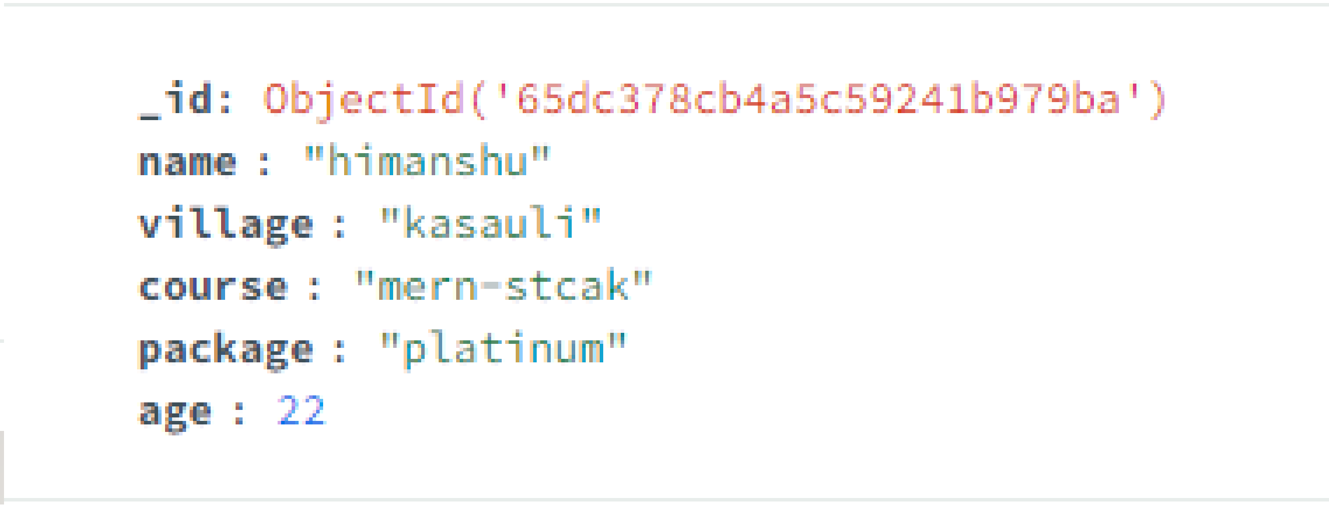
db.(collection name).deleteMany({age:20},{$set:{product:"phone"}})

This is targeted id

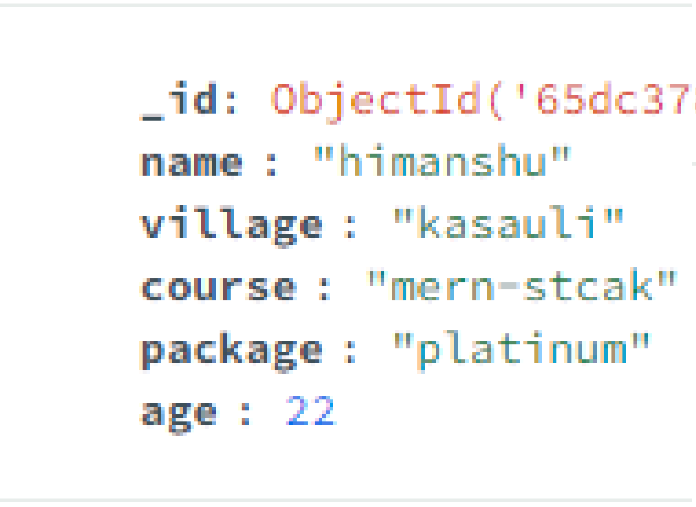
This is where the product deleted not only product but the whole id where the product preset and only that which having the age factor 20



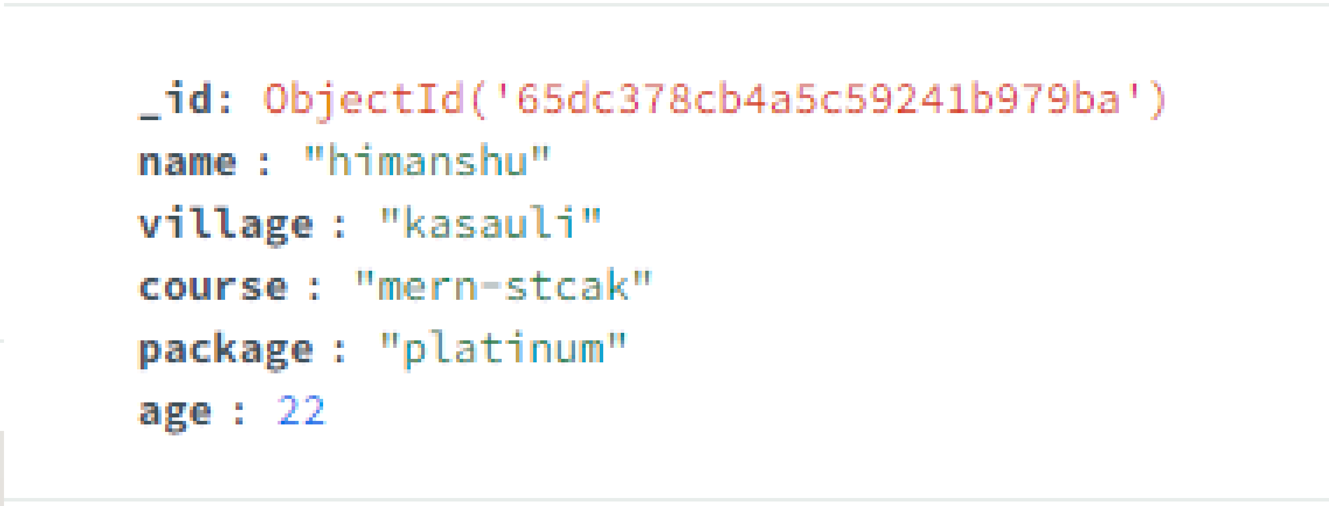
```
_id: ObjectId('65dc378cb4a5c59241b979ba')
name : "himanshu"
village : "kasauli"
course : "mern-stcak"
package : "platinum"
age : 22
```



```
_id: ObjectId('65dc378cb4a5c59241b979ba')
name : "himanshu"
village : "kasauli"
course : "mern-stcak"
package : "platinum"
age : 22
```



```
_id: ObjectId('65dc378cb4a5c59241b979ba')
name : "himanshu"
village : "kasauli"
course : "mern-stcak"
package : "platinum"
age : 22
```



```
_id: ObjectId('65dc378cb4a5c59241b979ba')
name : "himanshu"
village : "kasauli"
course : "mern-stcak"
package : "platinum"
age : 22
```

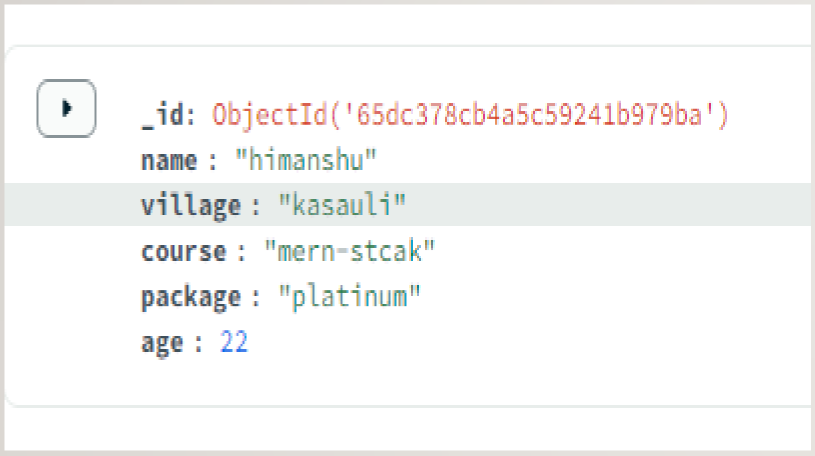
Suppose this is 4 different type of id's having same age of 22 so when we target the age it will automatically target those object having age 20 factor and suppose we have to delete the course which is mern stack where it find the target and deletion in the same id it will delete the whole id this is how it work **deleteMany**

UPDATEONE & UPDATEMANY

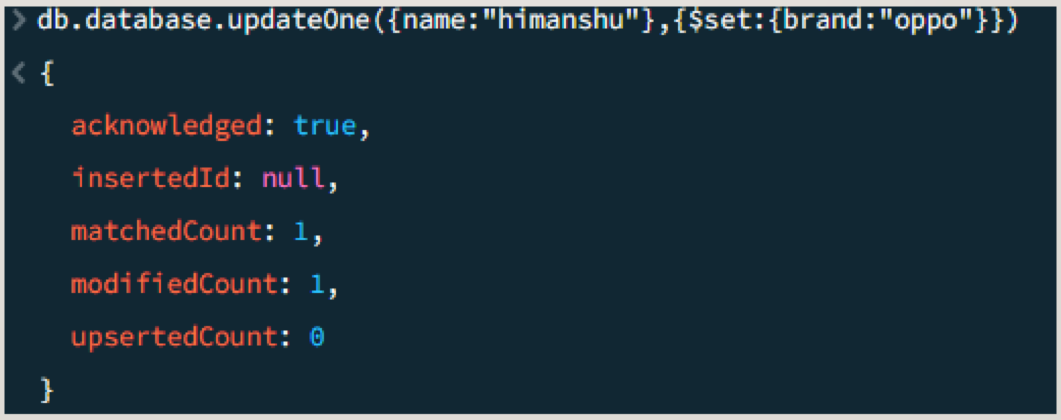
The **update()** method updates the values in the existing document in the collections of MongoDB. When you update your document the value of the _id field remains unchanged. By default, the db.collection.update() method updates a single document. Include the option multi: true to update all documents that match the given query. This method can be used for a single updating of documents as well as multi documents.

UPDATEONE

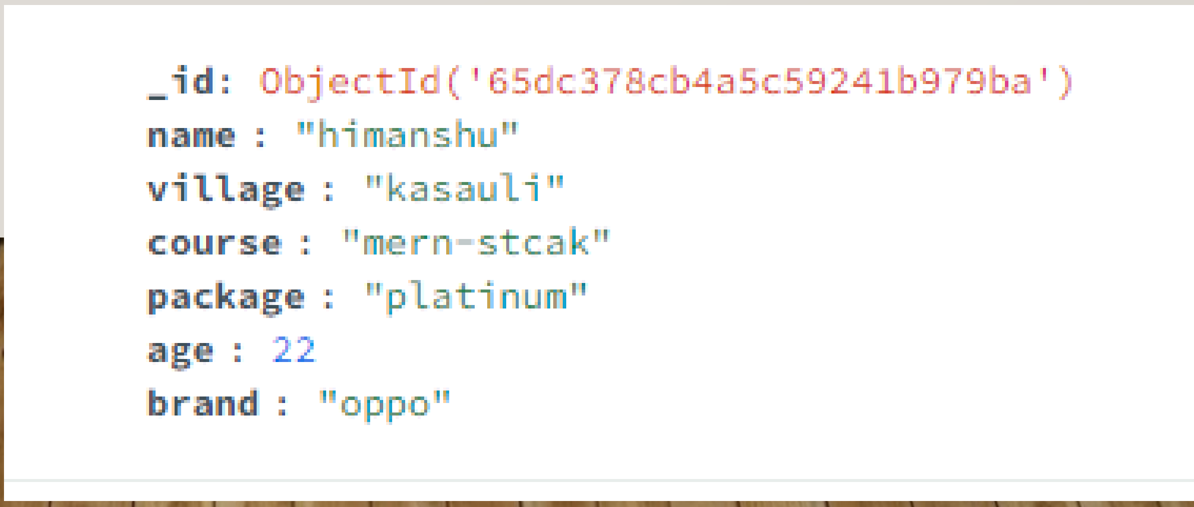
SYNTAX
db.(collection name).({brand:”nokia”})



```
_id: ObjectId('65dc378cb4a5c59241b979ba')
name : "himanshu"
village : "kasauli"
course : "mern-stcak"
package : "platinum"
age : 22
```



```
> db.database.updateOne({name:"himanshu"},{$set:{brand:"oppo"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```



```
_id: ObjectId('65dc378cb4a5c59241b979ba')
name : "himanshu"
village : "kasauli"
course : "mern-stcak"
package : "platinum"
age : 22
brand : "oppo"
```

UPDATEMANY

Updatemany work as update many id in single click we have to target which is common among all while it is age or something else and then we have to enter the new value which we want to update among all

SYNTAX

db.(collection name).updateMany({age:20},{$set:{product:"phone"}})

This is the targeted object

The value we have to update or enter it as new

The id's those are targeted get update or a new value will added

Connectivity of node with NODE Js.

Set the node environment of node js as usual and add the dependencies of mongodb by installing **npm I mongodb**

SYNTAX to connect node with mongodb

```
const { MongoClient } = require("mongodb");
const url = "mongodb://127.0.0.1:27017";
const database = "(database name)";
const client = new MongoClient(url);

  console.log(response);
}
getData();
```

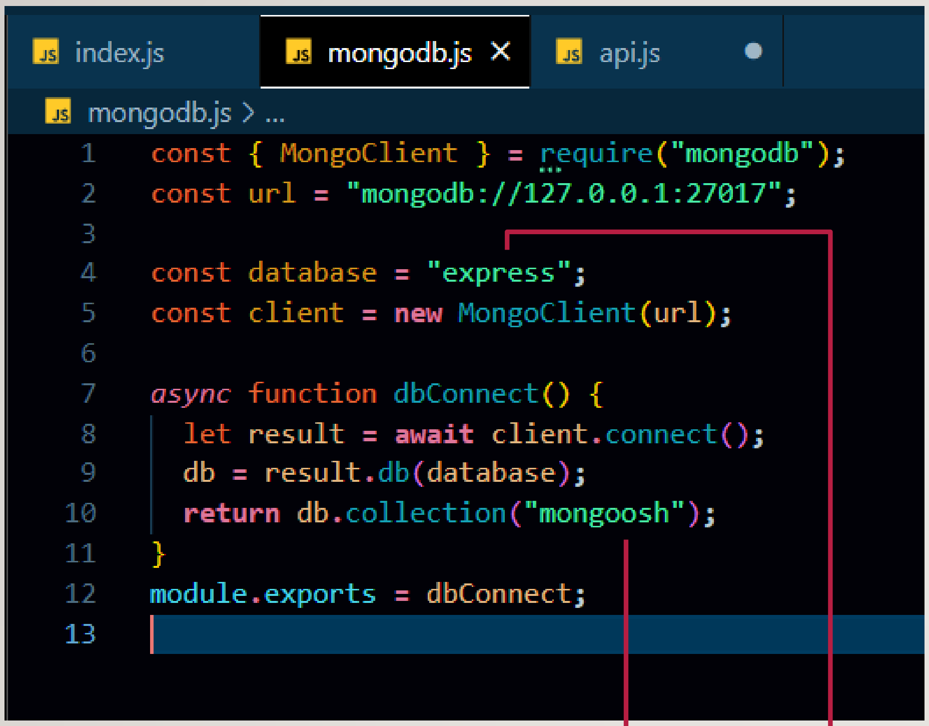
```
const { MongoClient } = require("mongodb");
const url = "mongodb://127.0.0.1:27017";
const database = "(database name)";
const client = new MongoClient(url);
async function getData() {
  let result = await client.connect();
  let db = result.db(database);
  let collection = db.collection("collection name");

  console.log(response);
}
getData();
```

Create, Read, Update, Delete file using mongoBD

By the use of get, put ,post method we actually create read update file in mongodb through post man
For this we have to apply some steps

To get all the data



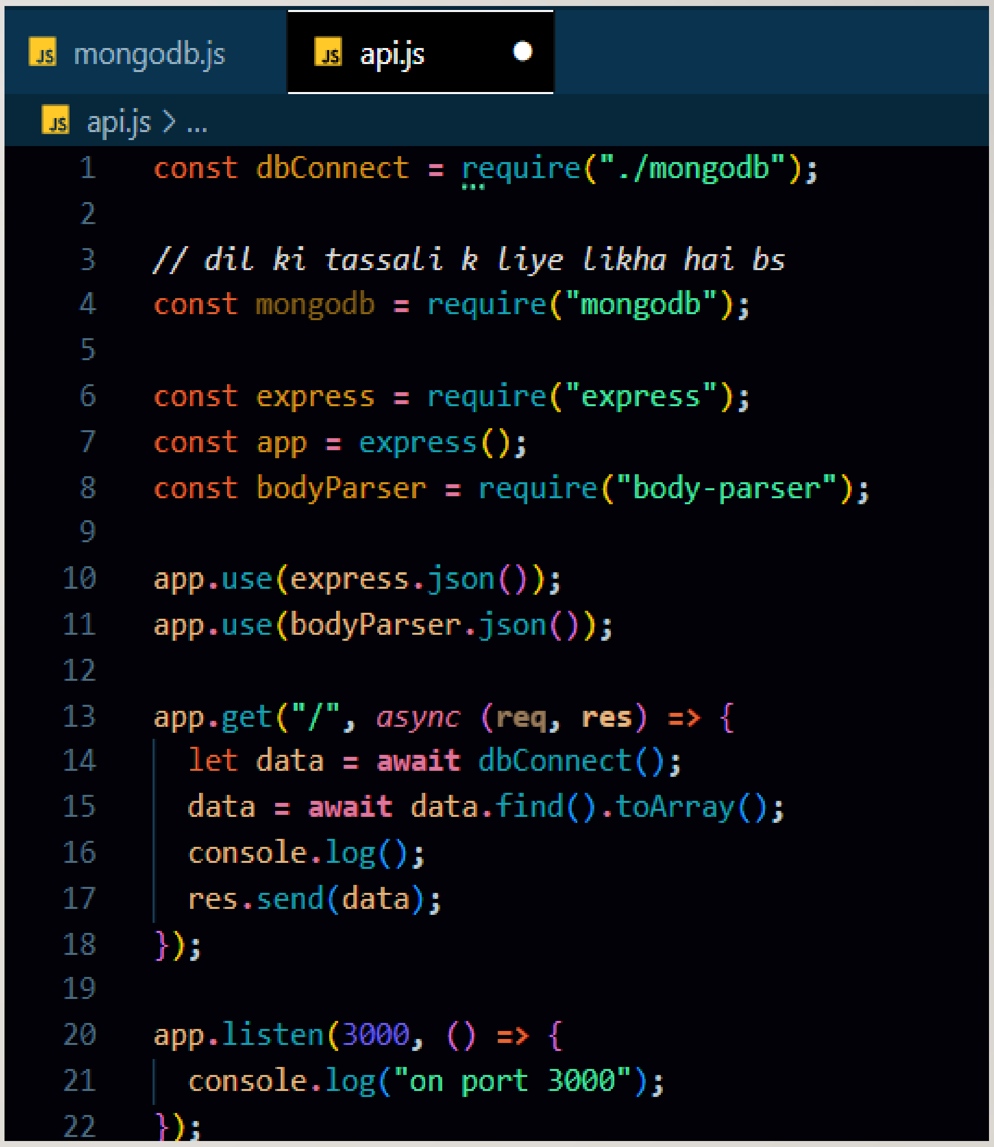
```
const { MongoClient } = require("mongodb");
const url = "mongodb://127.0.0.1:27017";

const database = "express";
const client = new MongoClient(url);

async function dbConnect() {
  let result = await client.connect();
  db = result.db(database);
  return db.collection("mongoosh");
}
module.exports = dbConnect;
```

Change it with your database name

Change it with your collection name



```
const dbConnect = require("./mongodb");

// dil ki tassali k liye likha hai bs
const mongodb = require("mongodb");

const express = require("express");
const app = express();
const bodyParser = require("body-parser");

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

app.get("/", async (req, res) => {
  let data = await dbConnect();
  data = await data.find().toArray();
  console.log();
  res.send(data);
});

app.listen(3000, () => {
  console.log("on port 3000");
});
```

After that open post man create a new http request add the localhost and route that you given in the program then send it after that click on body the click raw data change text into JSON then you will get your data

InsertOne by post method

```
app.post("/user", async (req,
res) => {
  let data = await dbConnect();
  let result = await
data.insertOne(req.body);
  res.send(result);
});
```

UpdateOne by put method

```
app.post("/update/:id", async (req, res) => {
  const data = await dbConnect();
  const { id } = req.params;
  const { name } = req.body;
  const objId = new mongodb.ObjectId(id);
  const result = await data.updateOne({ _id: objId }, {
$set: req.body });
  res.send("update data successfully");
});
```

To update the one object add new request in postman then (localhost:3000/(route)/(id name)) same goes for deleteMany

MONGOOSE

Mongoose is an Object Data Modeling (ODM) library for MongoDB. MongoDB is a NoSQL database and Mongoose is used to interact with MongoDB by providing a schema-based solution. The Mongoose acts as the abstraction layer over the MongoDB database. It is generally preferred over using normal MongoDB because it simplifies the process of sending complex queries.

Mongoose is used in Node application to interact with MongoDB without write complex queries. It acts as an Object Data Modeling (ODM) used to define schema model and provides easy communication between application and database. It provides many features like schema validation, middleware support, and easy query building It manages relationships between data, and is used to translate between objects in code and the representation of those objects in MongoDB.

Installation and Syntax

Npm I install

Syntax

```
const mongoose = require("mongoose");
mongoose.connect("mongodb://127.0.0.1:27017/(databasename)");
```

Terms to remember

```
const productmodel = mongoose.model("product", productschema, "product");
Write this instead of
const productmodel = mongoose.model("product", productschema);
This is because mongoose environment allows not to rewrite the folder name so
when we start with a collection name and note end it with the same name it
remane the collection by adding s and es so when you start a collection name
remember it to end it
```

Mongoose Schematype

•**Mongoose** is a MongoDB object modeling and handling for node.js environment. **Mongoose Schematype** is a configuration for the Mongoose model. Before creating a model, we always need to create a Schema. The SchemaType specifies what type of object is required at the given path. If the object doesn't match, it throws an error. The SchemaType is declared as follows:

`const schema = new Schema({ name: { type: String }, age: { type: Number, default: 10 }, });`

Data Types supported by SchemaType:

- **String**
- **Number**
- **Date**
- **Buffer**
- **Boolean**
- **Mixed**
- **ObjectId**
- **Array**
- **Decimal128**
- **Map**
- **Schema**

```
const productschema = new mongoose.Schema({
  name: String,
  price: Number,
  brand: String,
  category: String,
  country: String,
});
const main = async () => {
  const productmodel = mongoose.model("product",
productschema, "product");
  let data = new productmodel({
    name: "s24",
    price: 130000,
    brand: "samsung",
    category: "flagship",
    country: "Japan",
  });
  let result = await data.save();
  console.log(result);
};
```

Mongoose mongoose.model() Function

The **mongoose.model()** function of the mongoose module is used to create a collection of a particular database of MongoDB. The name of the collection created by the model function is always in plural format mean *GFG* to *gfss* and the created collection imposed a definite structure.

Syntax:

```
mongoose.model(<Collectionname>, <CollectionSchema>)
```