MONGO DB

INNOWAVE TEAM
(100+ SKILLED RESEARCHERS POOL)

"BRIDGING THE GAP BETWEEN KNOWLEDGE AND APPLICATION. YOUR MULTI-DISCIPLINARY RESEARCH PARTNER."

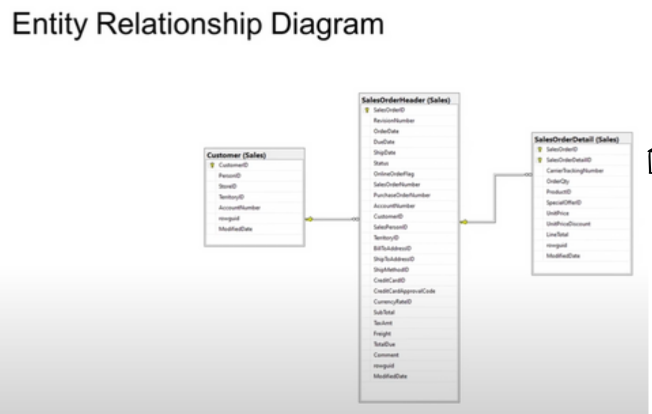
Contact US :

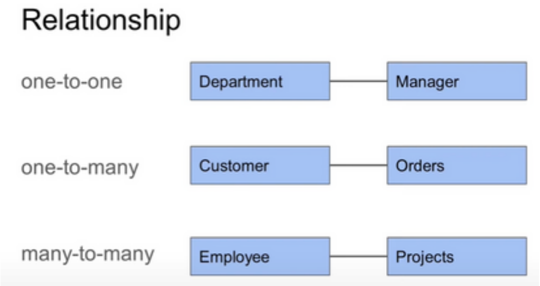
+94 70 225 2557 (WhatsApp)

HELPING ALL TYPE OF UNDERGRADES FOR COMPLETING THEIR PROJECTS AND ASSIGNMENTS

Feature	Relational Databases	Non-Relational Databases
Data Structure	Tables with rows and columns	Various: key-value, document, column-family, graph
Relationships	Defined using foreign keys	Implicit or handled separately
Schema	Fixed before data entry	Flexible or schema-less
Scalability	Vertical scaling (adding resources to a single server)	Horizontal scaling (distributing data across multiple servers)
Querying	SQL	Varies by model
Consistency	Strong ACID guarantees	Varies by implementation
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Cassandra, Redis, Neo4j
Best for	Structured data, predictable relationships, data integrity	Large datasets, unstructured data, flexibility, scalability

WHAN BEFORE USING RELATIONAL DATABASES


Entity Relationship Diagram


Relationship

one-to-one: Department — Manager

one-to-many: Customer — Orders

many-to-many: Employee — Projects

Normalization

First Normal Form (1NF)

Second Normal Form (2NF)

Third Normal Form (3NF)

Create DB

SQL USE FOR CRUD RELATIONAL DATABASES

CRUD	SQL	HTTP
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DELETE	DELETE

MONGODB

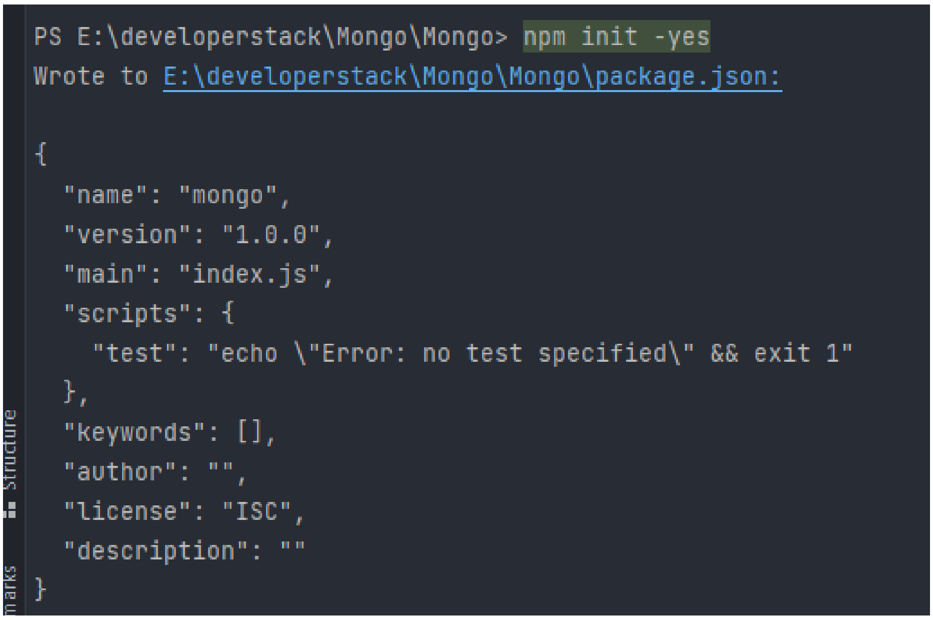
MongoDB is a document database and can be installed locally or hosted in the cloud.

The MongoDB Query API can be used two ways:

- CRUD Operations
- Aggregation Pipelines

Inserting Documents

two ways intert data
1.mongosh
2mongodb driver



```
PS E:\developerstack\Mongo\Mongo> npm init -yes
Wrote to E:\developerstack\Mongo\Mongo\package.json:

{
  "name": "mongo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

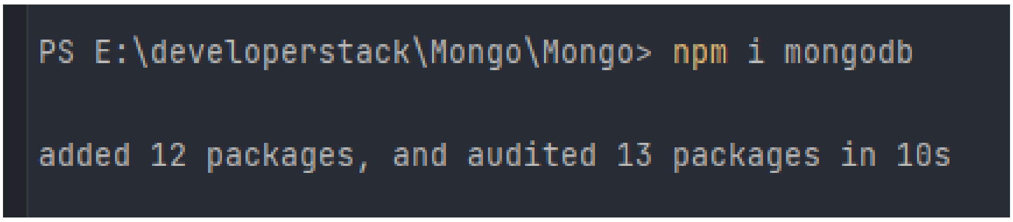
why we use npm init -yes before working

Project Initialization:

Creates a package.json file, essential for managing Node.js projects.
Stores project metadata (name, version, author, etc.).
Tracks project dependencies, ensuring consistent environments across machines

.
Dependency Management:

Prepares the project for installing MongoDB drivers or tools.
npm install command relies on package.json to manage dependencies.

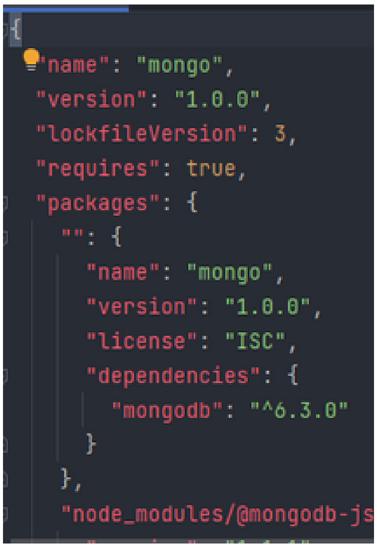


```
PS E:\developerstack\Mongo\Mongo> npm i mongodb

added 12 packages, and audited 13 packages in 10s
```

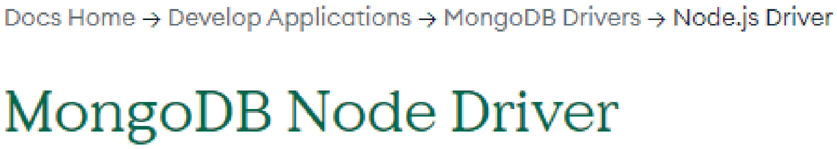
The command npm i mongodb will install the official Node.js driver for interacting with MongoDB into your current project directory. Here's what happens:

Download: It downloads the mongodb package and its dependencies from the npm registry.
Installation: The downloaded files are placed in the node_modules directory within your project.
Dependency Recording: An entry for the mongodb package is added to your package.json file, which keeps track of all your project's dependencies.
It's important to note that:

You need to be in the directory of your Node.js project when running the command.
Ensure you have Node.js and npm installed on your system.
Running this command without previous project initialization (using npm init -y) will still work, but it's good practice to have a package.json file for managing dependencies.



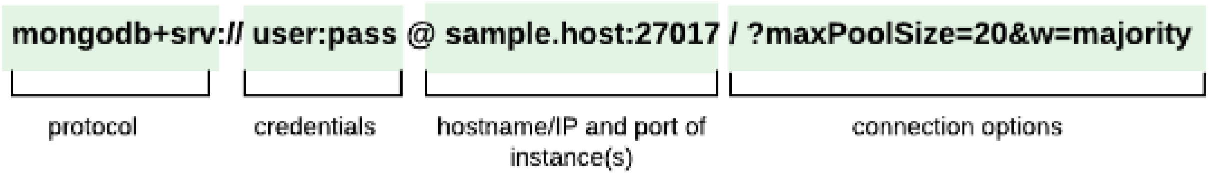
```
{
  "name": "mongo",
  "version": "1.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "mongo",
      "version": "1.0.0",
      "license": "ISC",
      "dependencies": {
        "mongodb": "^6.3.0"
      }
    },
    "node_modules/@mongodb-js
```



Docs Home → Develop Applications → MongoDB Drivers → Node.js Driver

MongoDB Node Driver

Connect to MongoDB


mongodb+srv:// user:pass @ sample.host:27017 / ?maxPoolSize=20&w=majority
protocol
credentials
hostname/IP and port of instance(s)
connection options

1. Choose Your Method:

- Mongo Shell (mongosh): A command-line interface for direct interaction.
- Node.js Driver: For integrating MongoDB into Node.js applications.
- MongoDB Compass: A GUI for visual interaction and data management.
- Other Drivers and Tools: MongoDB supports drivers for various programming languages and platforms.

2. Obtain Connection Information:

Local MongoDB Instance:

Default host: localhost

Default port: 27017

Remote MongoDB Instance:

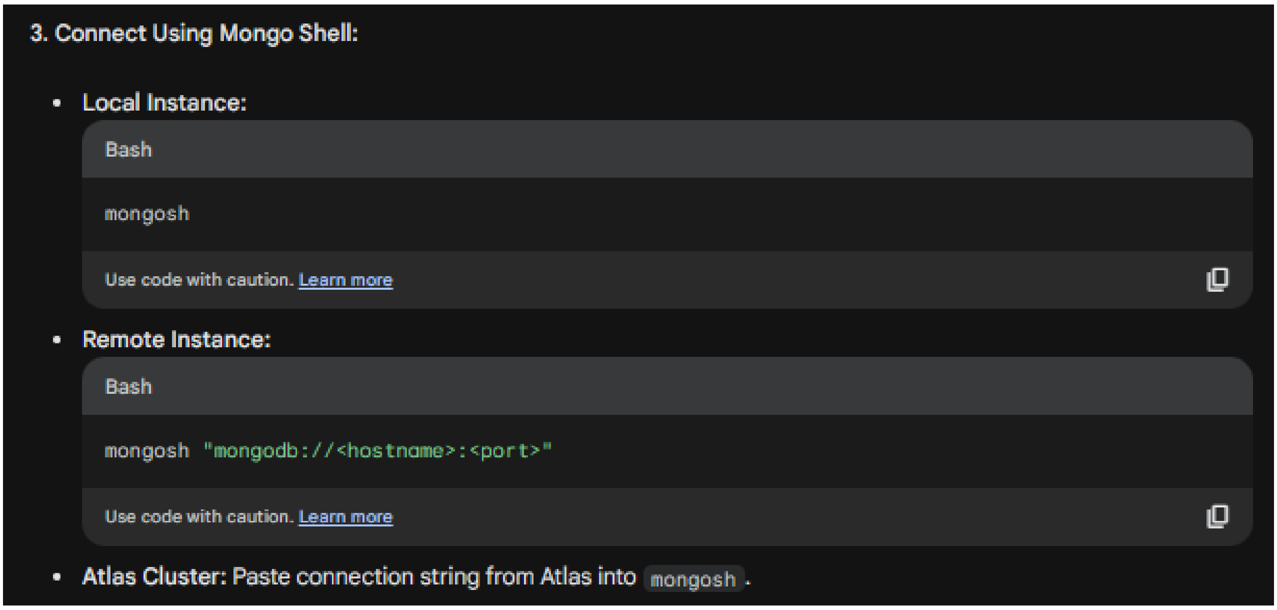
Obtain host name or IP address.

Verify port number (usually 27017).

May require authentication credentials.

MongoDB Atlas Cluster:

Get connection string from Atlas dashboard.



3. Connect Using Mongo Shell:

- Local Instance:

Bash

```bash
mongosh
```

Use code with caution. Learn more

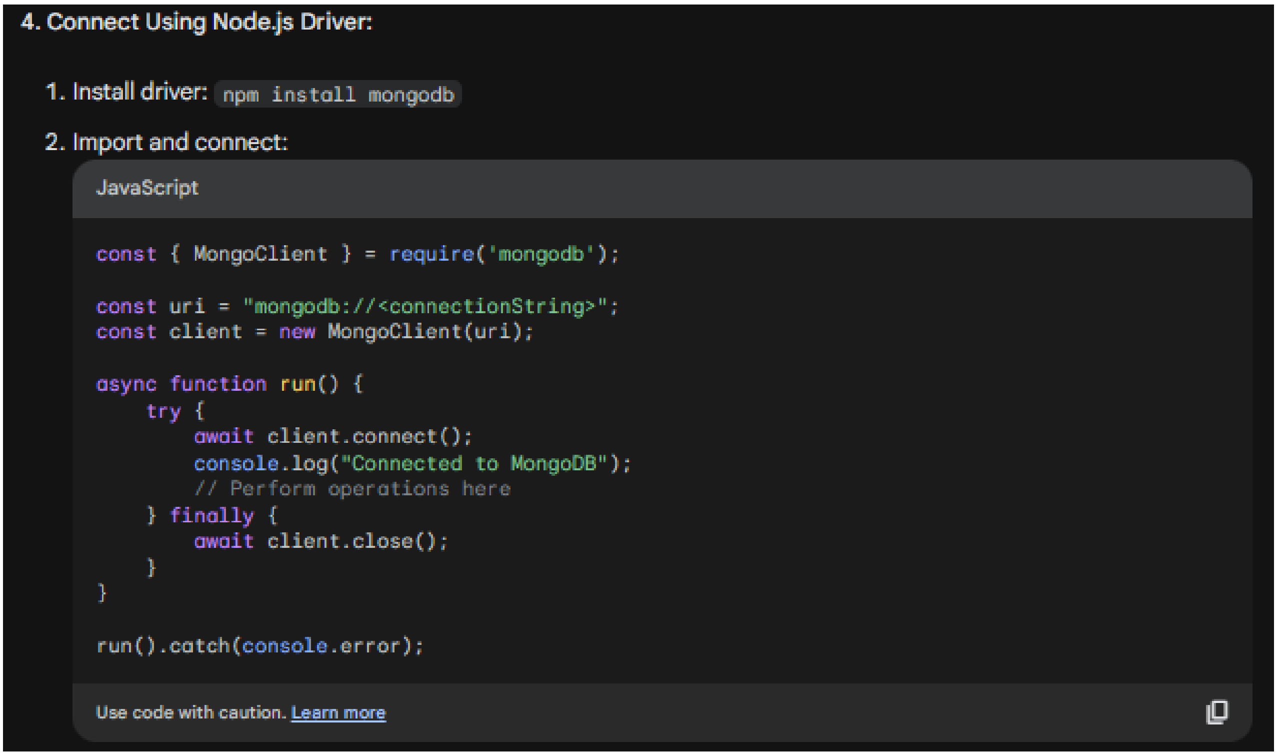
- Remote Instance:

Bash

```bash
mongosh "mongodb://<hostname>:<port>"
```

Use code with caution. Learn more

- Atlas Cluster: Paste connection string from Atlas into `mongosh`.



4. Connect Using Node.js Driver:

1. Install driver: `npm install mongodb`
2. Import and connect:

JavaScript

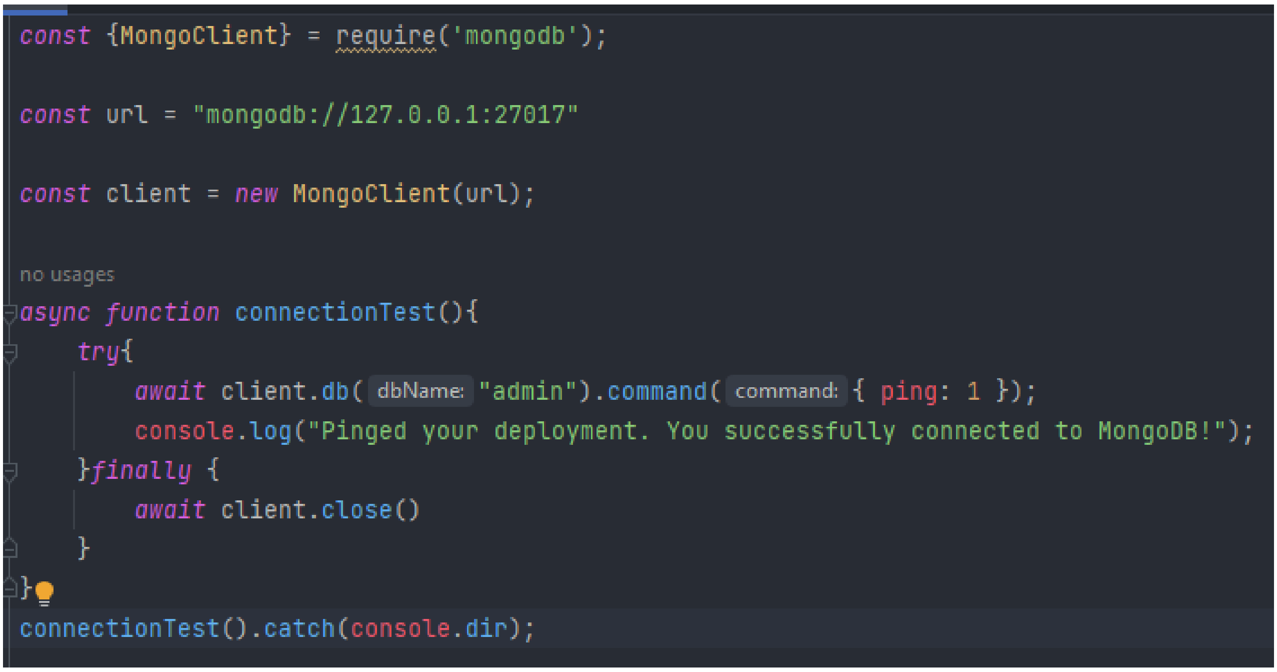
```javascript
const { MongoClient } = require('mongodb');

const uri = "mongodb://<connectionString>";
const client = new MongoClient(uri);

async function run() {
    try {
        await client.connect();
        console.log("Connected to MongoDB");
        // Perform operations here
    } finally {
        await client.close();
    }
}

run().catch(console.error);
```

Use code with caution. Learn more

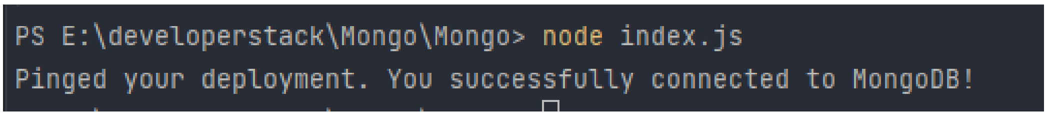
5. Connect Using MongoDB Compass:

1. Download and install MongoDB Compass.
2. Launch Compass and create a new connection.
3. Paste connection string or provide host, port, authentication details.
4. Click "Connect".

connect database



```
const {MongoClient} = require('mongodb');

const url = "mongodb://127.0.0.1:27017"

const client = new MongoClient(url);

no usages
async function connectionTest(){
    try{
        await client.db( dbName: "admin").command( command: { ping: 1 });
        console.log("Pinged your deployment. You successfully connected to MongoDB!");
    }finally {
        await client.close()
    }
}
connectionTest().catch(console.dir);
```



```
PS E:\developerstack\Mongo\Mongo> node index.js
Pinged your deployment. You successfully connected to MongoDB!
```

Insert Data

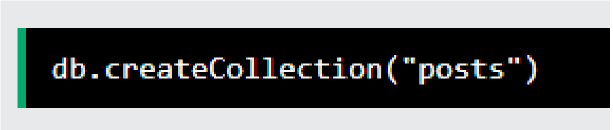
Create database

const database = client.db('myDatabase');

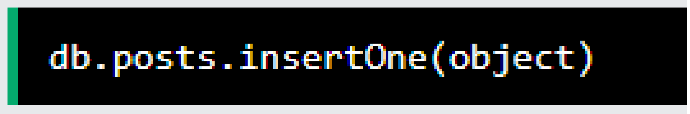
Create collection

const collection = database.collection('products')

There are 2 ways to create a collection.



```
db.createCollection("posts")
```



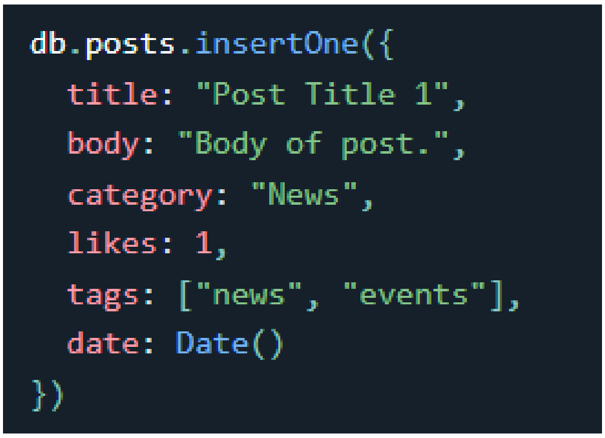
```
db.posts.insertOne(object)
```

insert a value

```
const doc = {
name:'abc',
price:100.00,
brand : "a"
}
const result = await collection.insertOne(doc);
console.log(result)
```

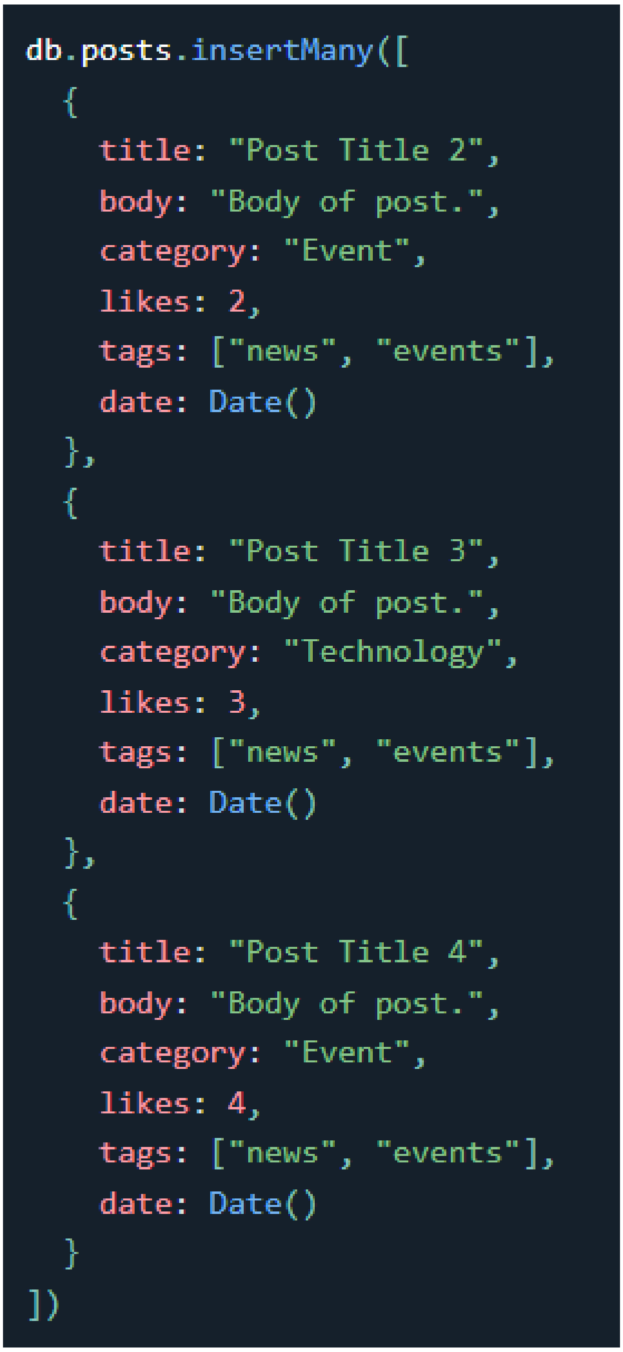
There are 2 methods to insert documents into a MongoDB database.

```
insertOne()
```

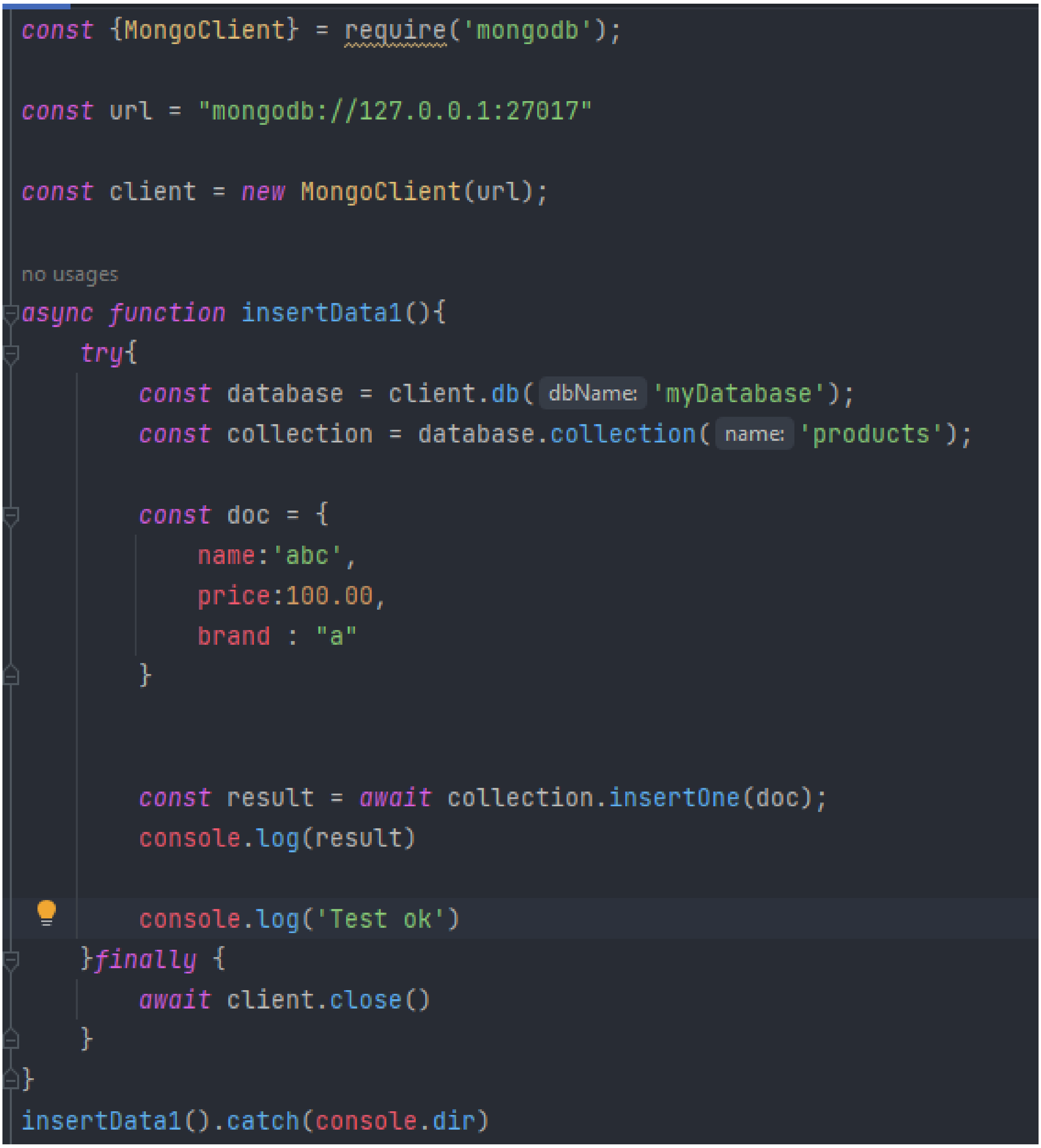


```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

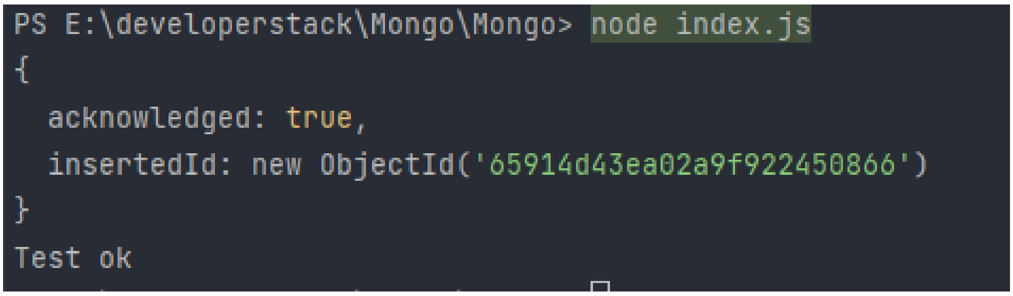
```
insertMany()
```



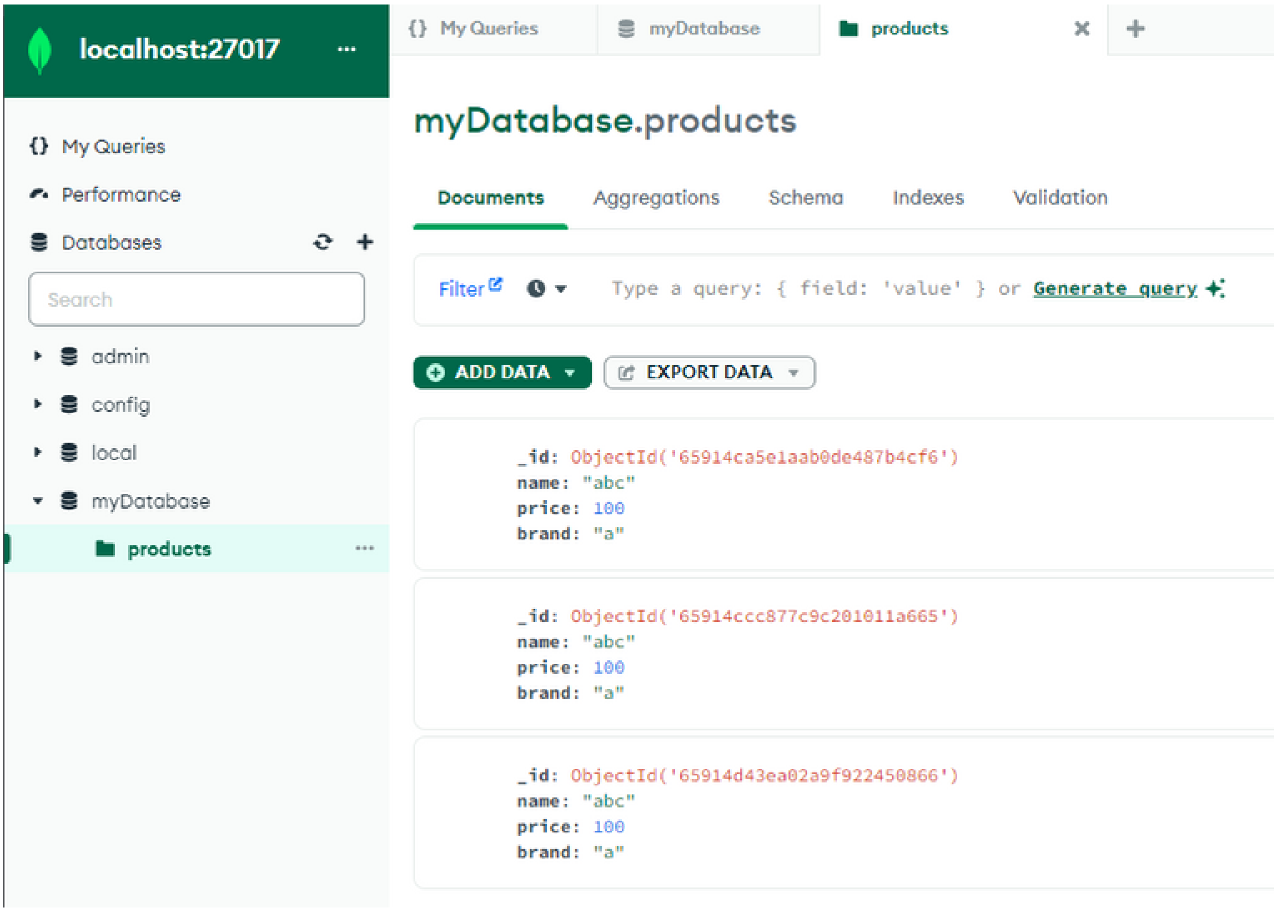
```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 4",
    body: "Body of post.",
    category: "Event",
    likes: 4,
    tags: ["news", "events"],
    date: Date()
  }
])
```



```
const {MongoClient} = require('mongodb');

const url = "mongodb://127.0.0.1:27017"

const client = new MongoClient(url);

no usages
async function insertData1(){
    try{
        const database = client.db( dbName: 'myDatabase');
        const collection = database.collection( name: 'products');

        const doc = {
            name:'abc',
            price:100.00,
            brand : "a"
        }


        const result = await collection.insertOne(doc);
        console.log(result)

        console.log('Test ok')
    }finally {
        await client.close()
    }
}
insertData1().catch(console.dir)
```



```
PS E:\developerstack\Mongo\Mongo> node index.js
{
  acknowledged: true,
  insertedId: new ObjectId('65914d43ea02a9f922450866')
}
Test ok
```

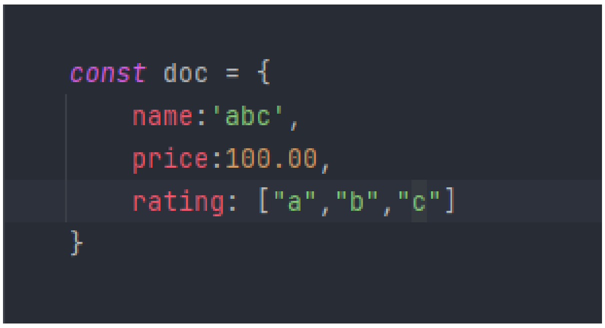


myDatabase.products

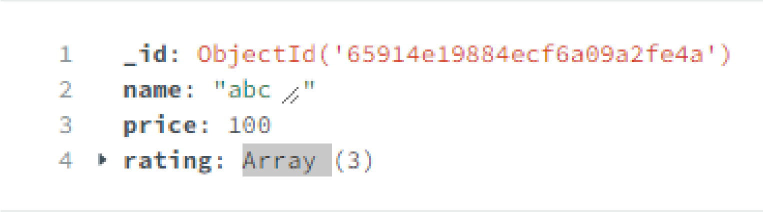
```
_id: ObjectId('65914ca5e1aab0de487b4cf6')
name: "abc"
price: 100
brand: "a"
```

```
_id: ObjectId('65914ccc877c9c201011a665')
name: "abc"
price: 100
brand: "a"
```

```
_id: ObjectId('65914d43ea02a9f922450866')
name: "abc"
price: 100
brand: "a"
```

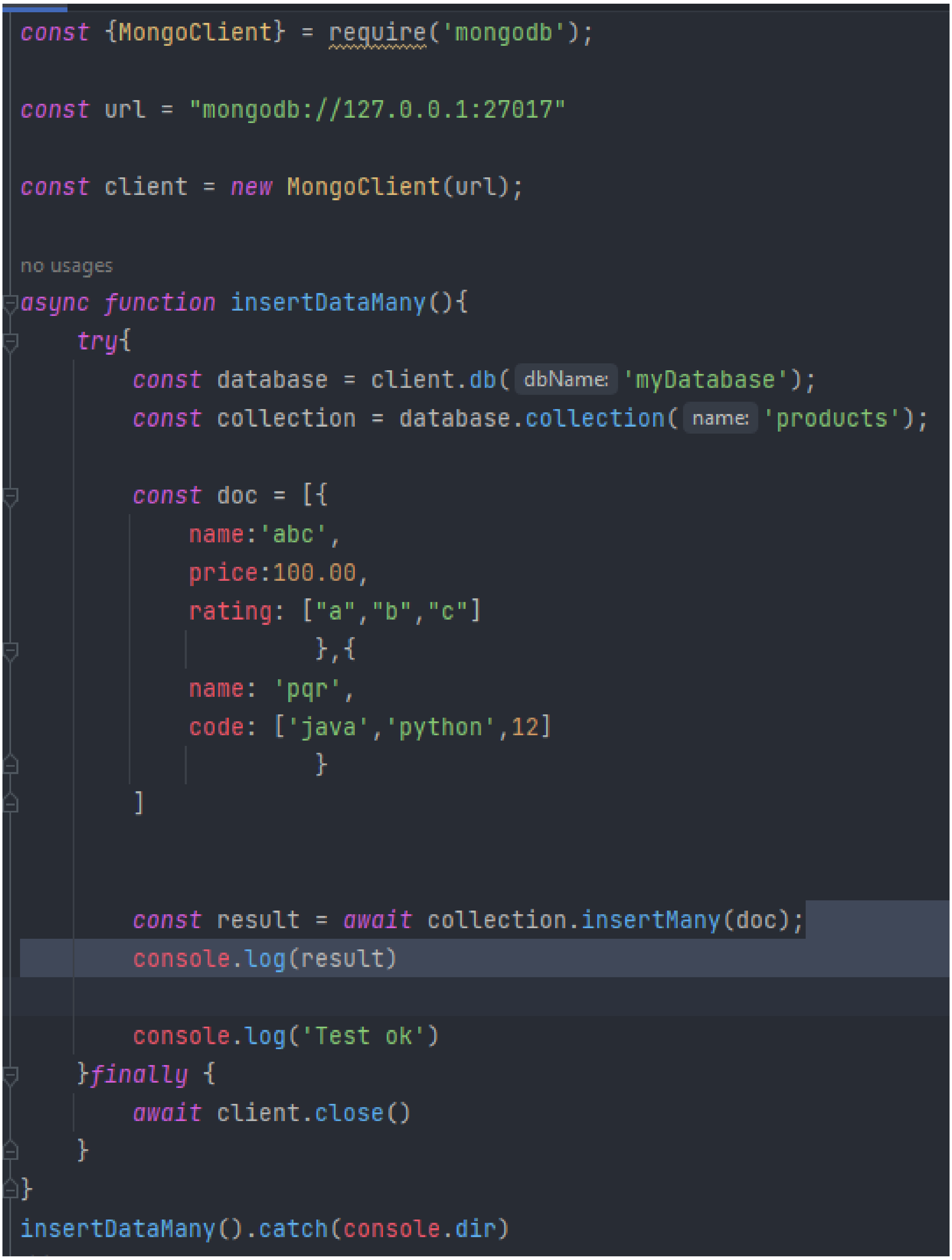


```
const doc = {
    name:'abc',
    price:100.00,
    rating: ["a","b","c"]
}
```

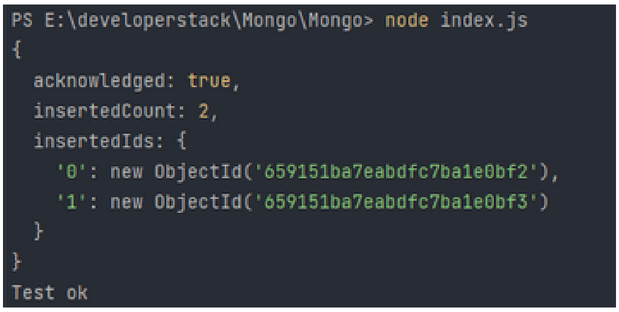


```
  _id: ObjectId('65914e19884ecf6a09a2fe4a')
  name: "abc"
  price: 100
▸ rating: Array (3)
```

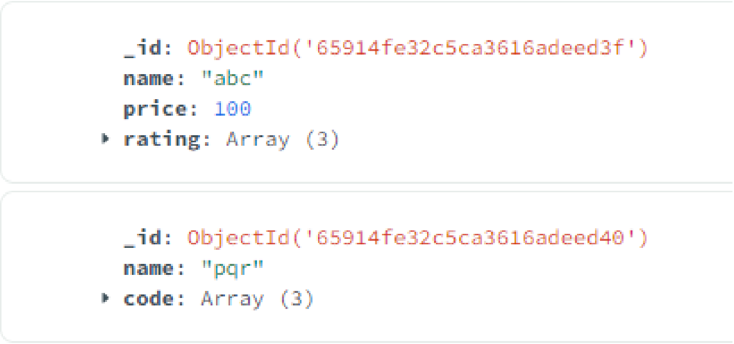
insert more value



```
const {MongoClient} = require('mongodb');

const url = "mongodb://127.0.0.1:27017"

const client = new MongoClient(url);

no usages
async function insertDataMany(){
    try{
        const database = client.db( dbName: 'myDatabase');
        const collection = database.collection( name: 'products');

        const doc = [{
            name:'abc',
            price:100.00,
            rating: ["a","b","c"]
                    },{
            name: 'pqr',
            code: ['java','python',12]
                    }
        ]


        const result = await collection.insertMany(doc);
        console.log(result)

        console.log('Test ok')
    }finally {
        await client.close()
    }
}
insertDataMany().catch(console.dir)
```



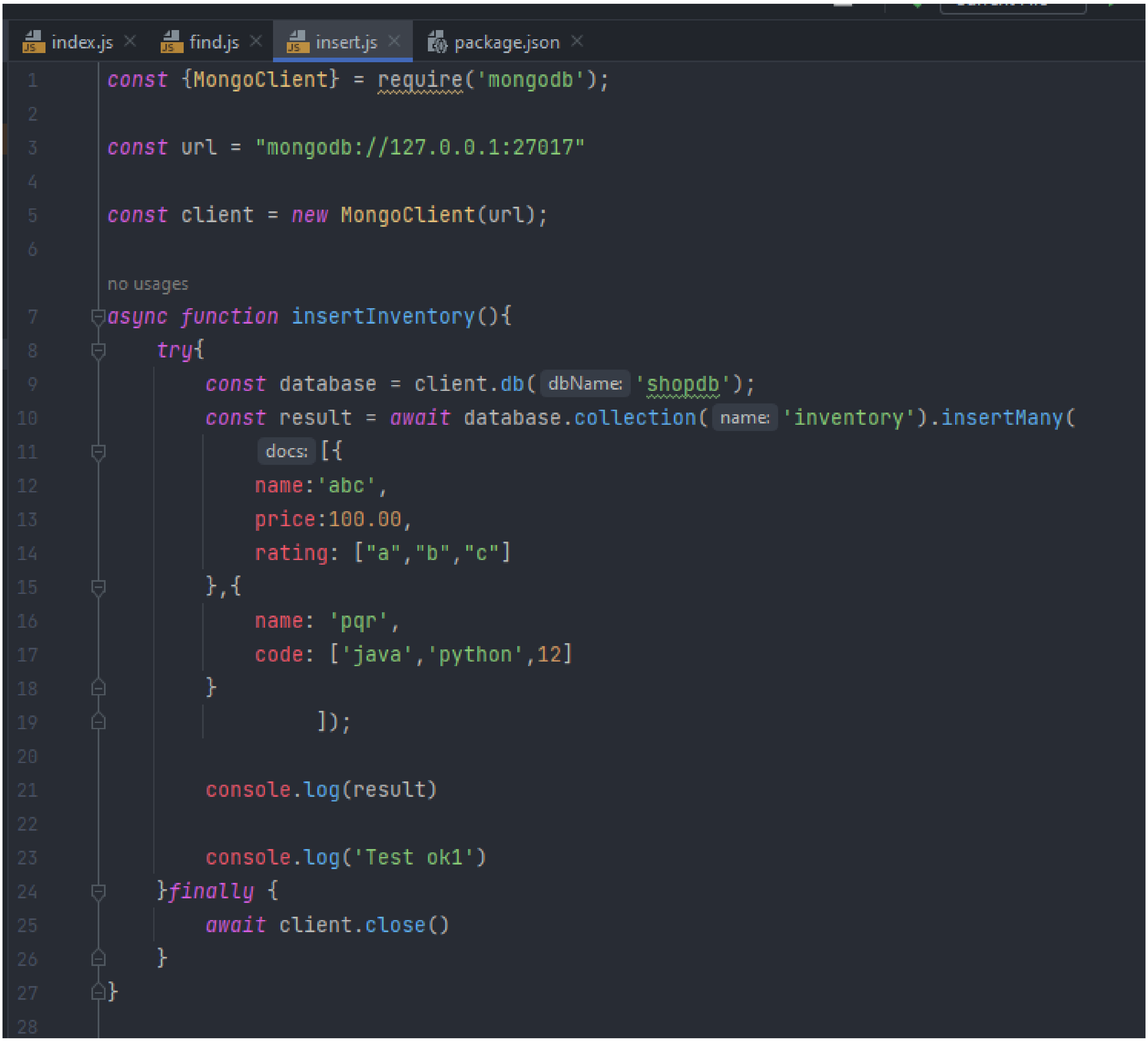
```
PS E:\developerstack\Mongo\Mongo> node index.js
{
  acknowledged: true,
  insertedCount: 2,
  insertedIds: {
    '0': new ObjectId('659151ba7eabdfc7ba1e0bf2'),
    '1': new ObjectId('659151ba7eabdfc7ba1e0bf3')
  }
}
Test ok
```



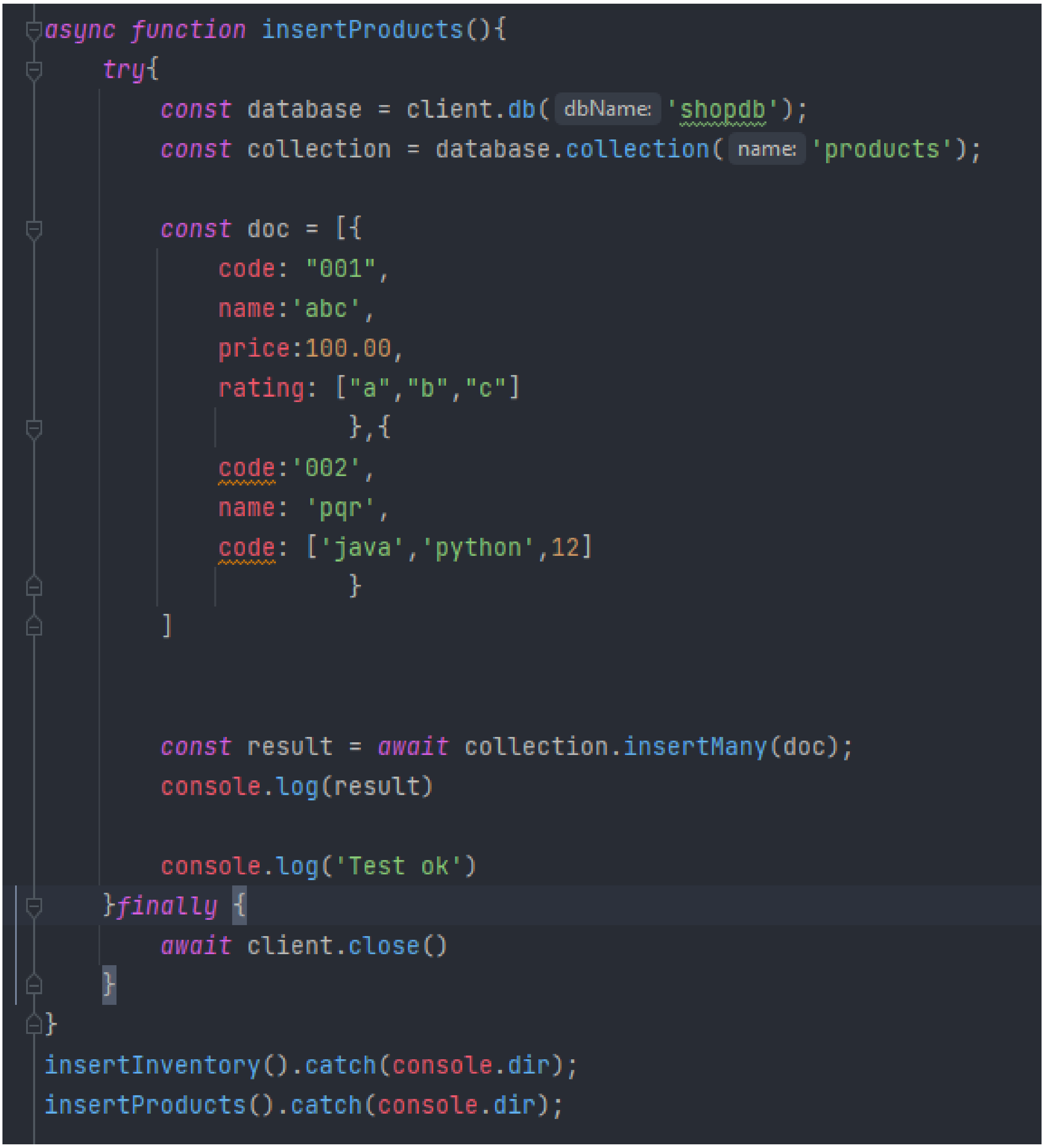
```
_id: ObjectId('65914fe32c5ca3616adeed3f')
name: "abc"
price: 100
▸ rating: Array (3)
```

```
_id: ObjectId('65914fe32c5ca3616adeed40')
name: "pqr"
▸ code: Array (3)
```

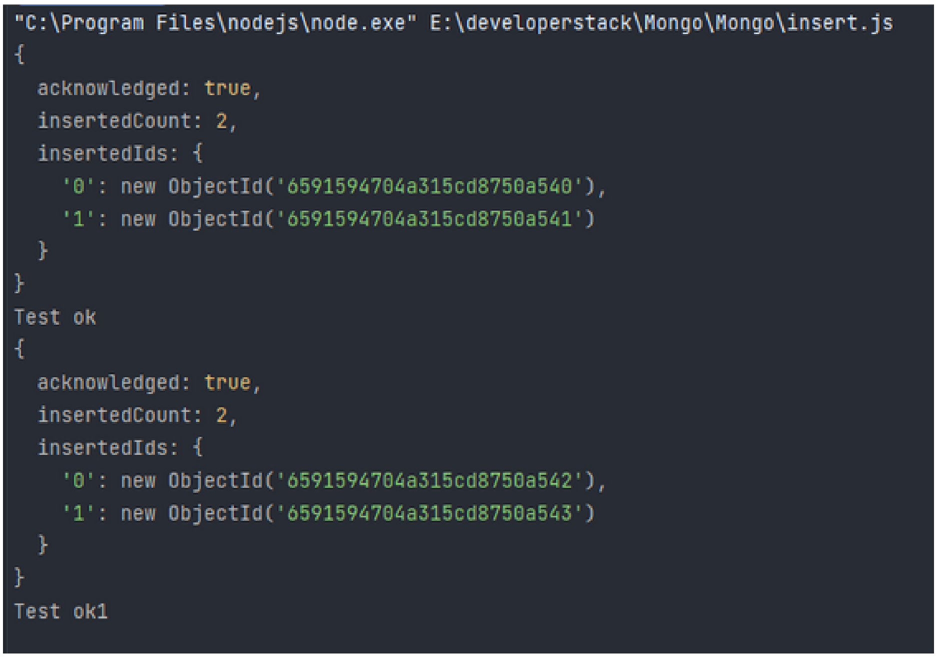
Find Documents



```
const {MongoClient} = require('mongodb');

const url = "mongodb://127.0.0.1:27017"

const client = new MongoClient(url);

no usages
async function insertInventory(){
    try{
        const database = client.db( dbName: 'shopdb');
        const result = await database.collection( name: 'inventory').insertMany(
             docs: [{
            name:'abc',
            price:100.00,
            rating: ["a","b","c"]
        },{
            name: 'pqr',
            code: ['java','python',12]
        }
                 ]);

        console.log(result)

        console.log('Test ok1')
    }finally {
        await client.close()
    }
}

```



```
async function insertProducts(){
    try{
        const database = client.db( dbName: 'shopdb');
        const collection = database.collection( name: 'products');

        const doc = [{
            code: "001",
            name:'abc',
            price:100.00,
            rating: ["a","b","c"]
                     },{
            code:'002',
            name: 'pqr',
            code: ['java','python',12]
                     }
        ]


        const result = await collection.insertMany(doc);
        console.log(result)

        console.log('Test ok')
    }finally {
        await client.close()
    }
}
insertInventory().catch(console.dir);
insertProducts().catch(console.dir);
```



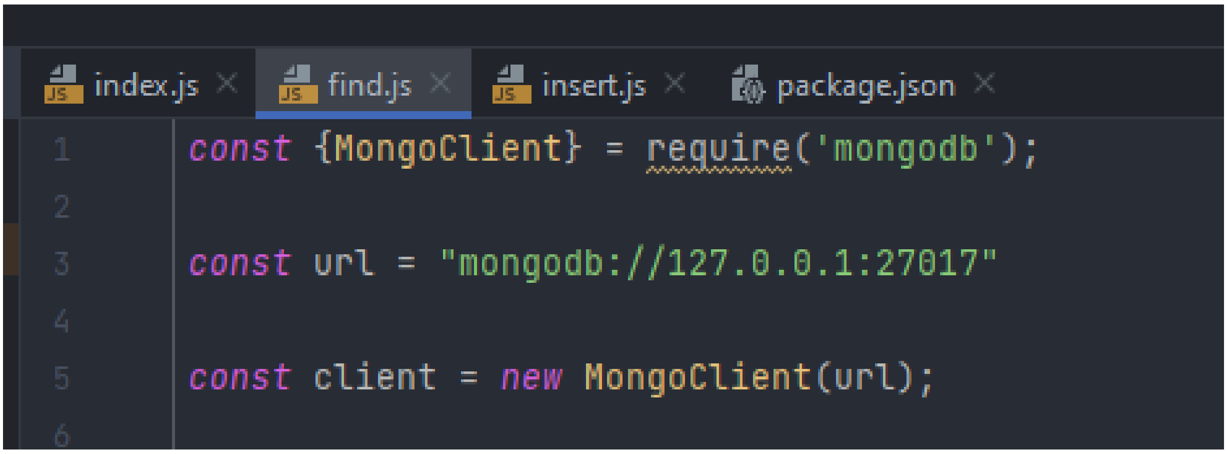
```
"C:\Program Files\nodejs\node.exe" E:\developerstack\Mongo\Mongo\insert.js
{
  acknowledged: true,
  insertedCount: 2,
  insertedIds: {
    '0': new ObjectId('6591594704a315cd8750a540'),
    '1': new ObjectId('6591594704a315cd8750a541')
  }
}
Test ok
{
  acknowledged: true,
  insertedCount: 2,
  insertedIds: {
    '0': new ObjectId('6591594704a315cd8750a542'),
    '1': new ObjectId('6591594704a315cd8750a543')
  }
}
Test ok1
```

Docs Home → Develop Applications → MongoDB Drivers → Node.js Driver

Retrieve Data

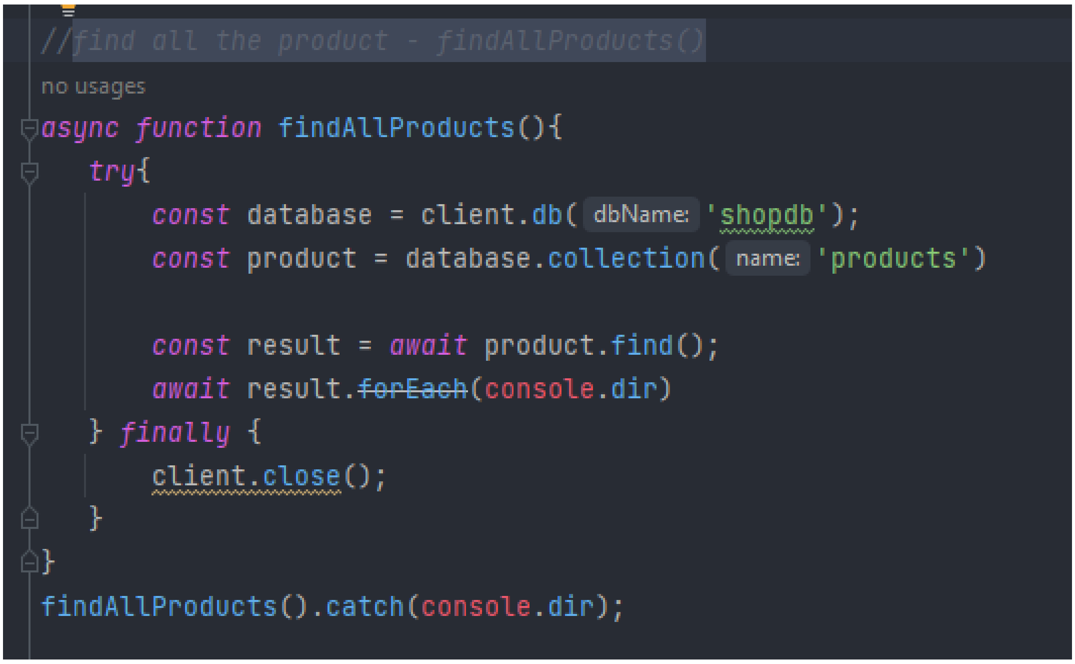
Find Data

There are 2 methods to find and select data from a MongoDB collection, find() and findOne().

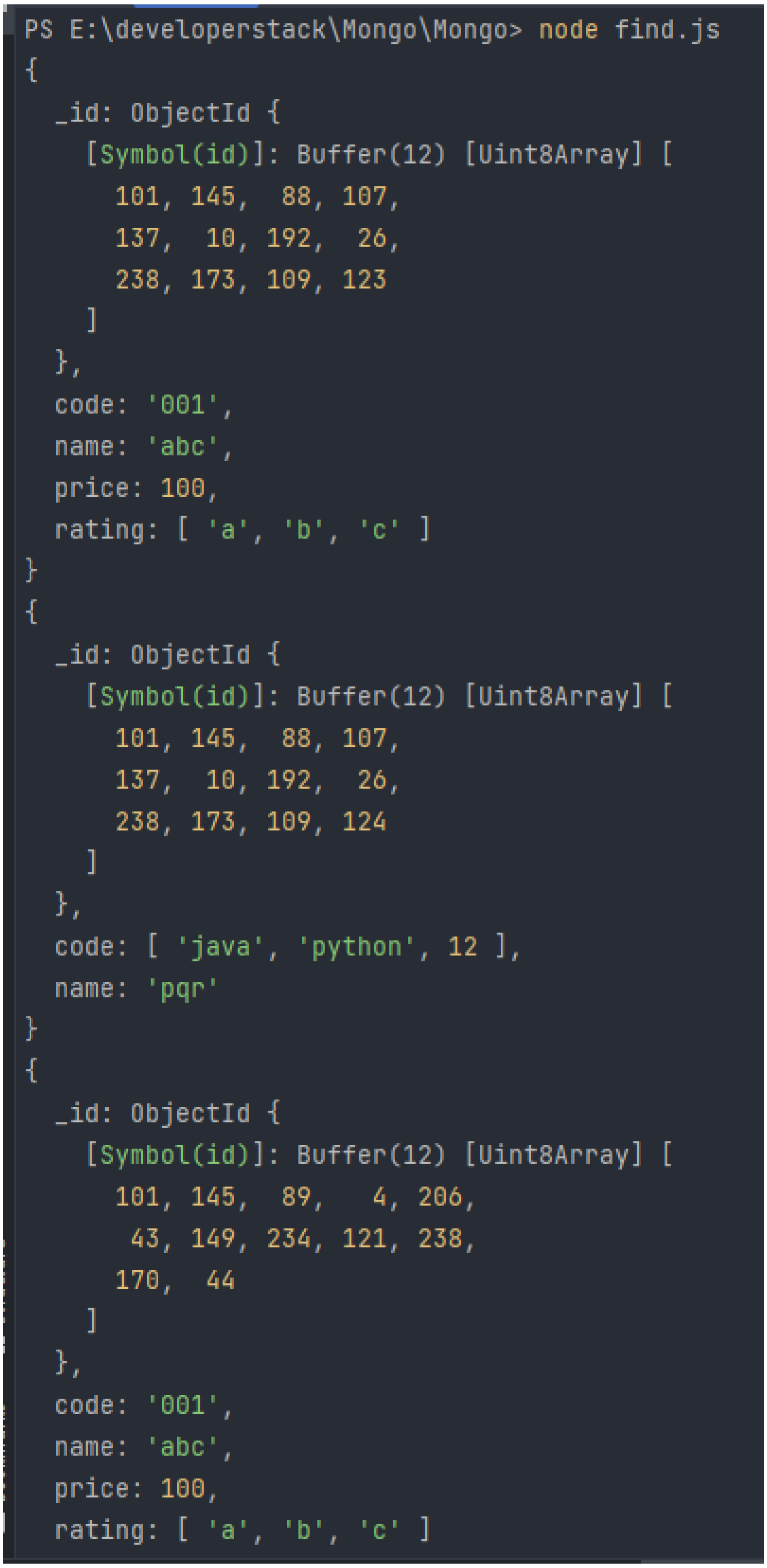


```
const {MongoClient} = require('mongodb');

const url = "mongodb://127.0.0.1:27017"

const client = new MongoClient(url);

```

find all the product - findAllProducts()

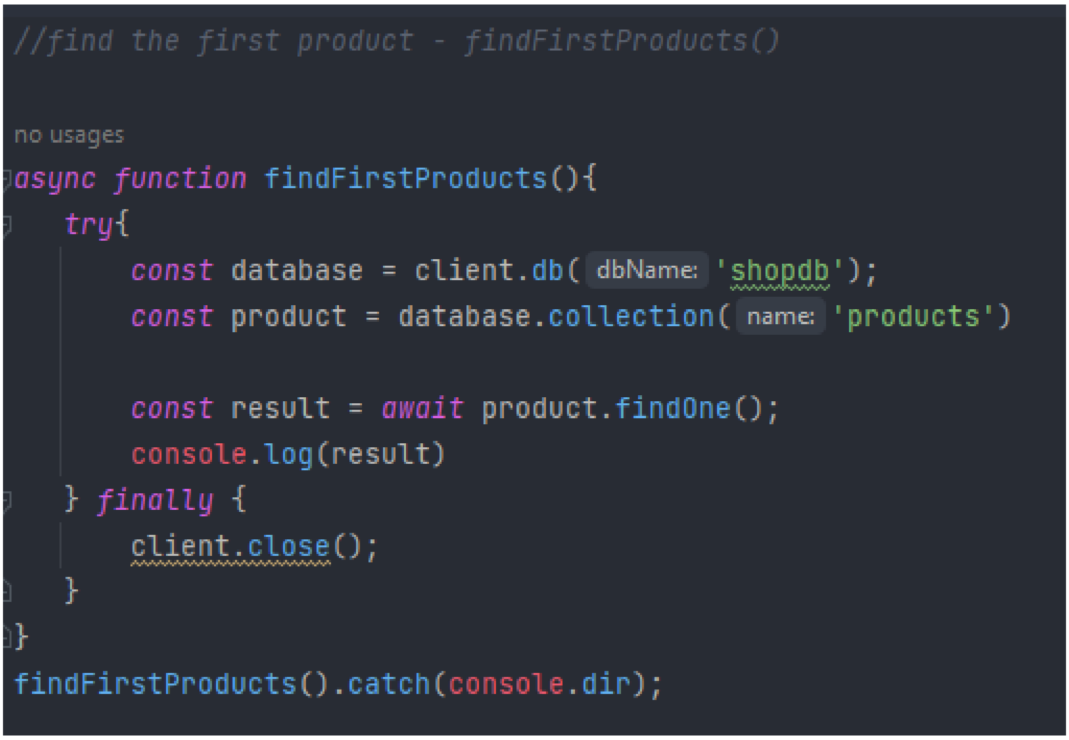


```
//find all the product - findAllProducts()
no usages
async function findAllProducts(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const result = await product.find();
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
findAllProducts().catch(console.dir);
```

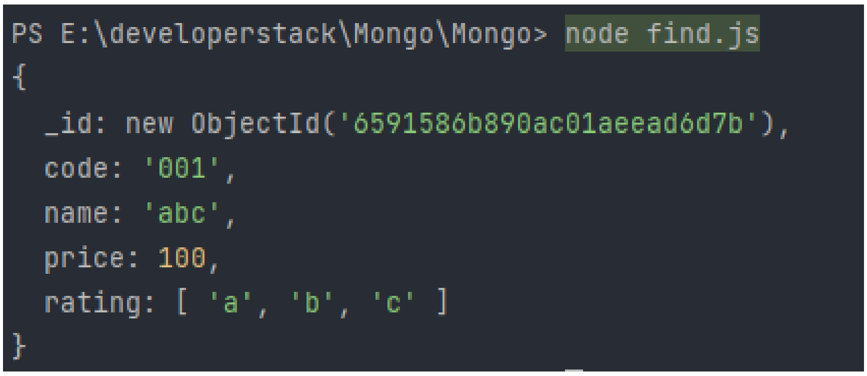


```
PS E:\developerstack\Mongo\Mongo> node find.js
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  88, 107,
      137,  10, 192,  26,
      238, 173, 109, 123
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  88, 107,
      137,  10, 192,  26,
      238, 173, 109, 124
    ]
  },
  code: [ 'java', 'python', 12 ],
  name: 'pqr'
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  89,   4, 206,
       43, 149, 234, 121, 238,
      170,  44
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
```

find the first product – findFirstProducts()



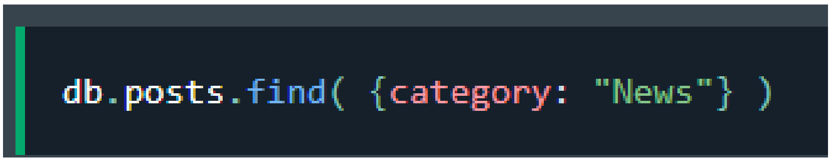
```
//find the first product - findFirstProducts()

no usages
async function findFirstProducts(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const result = await product.findOne();
        console.log(result)
    } finally {
        client.close();
    }
}
findFirstProducts().catch(console.dir);
```



```
PS E:\developerstack\Mongo\Mongo> node find.js
{
  _id: new ObjectId('6591586b890ac01aeead6d7b'),
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
```

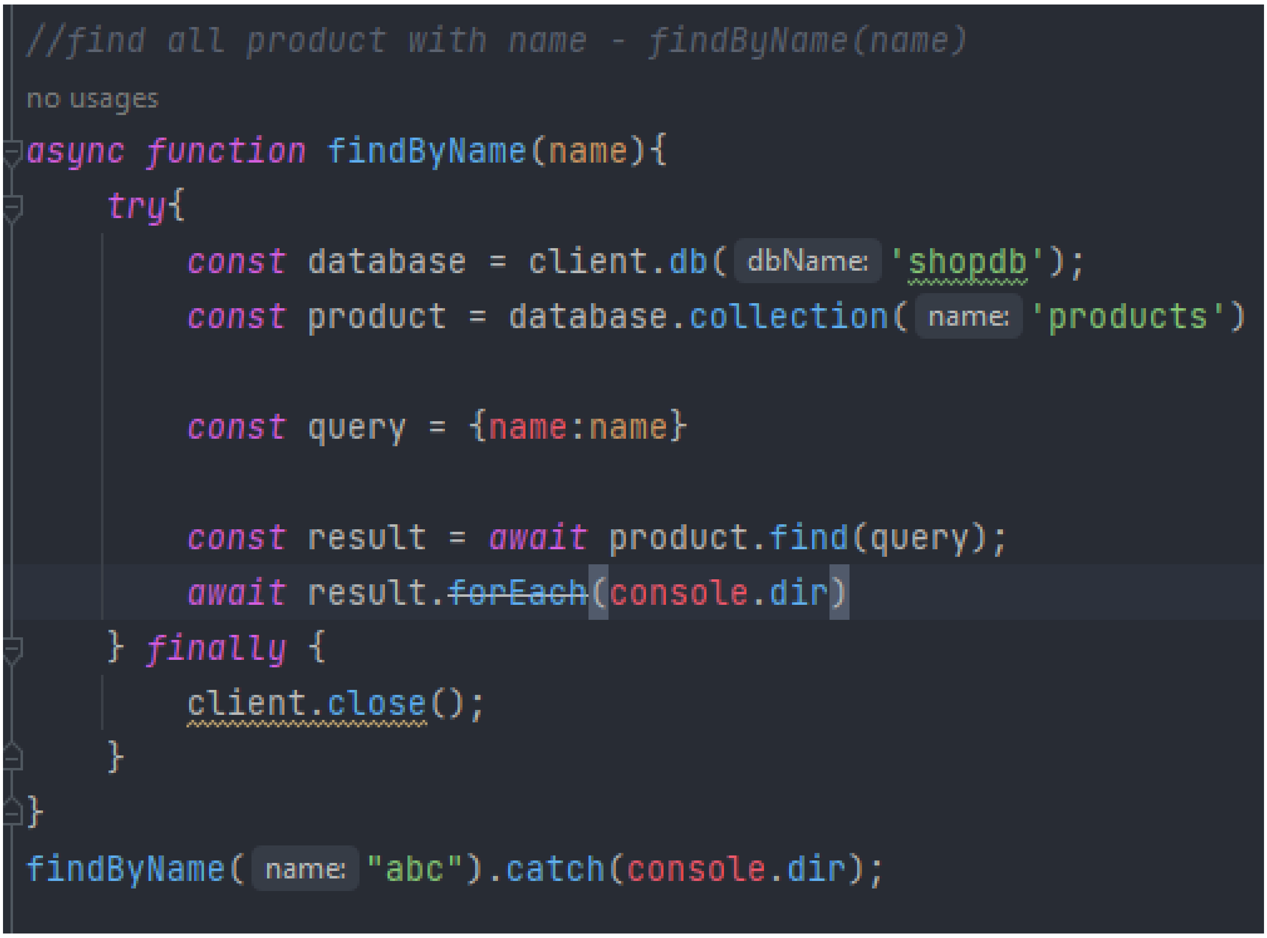
Querying Data

To query, or filter, data we can include a query in our find() or findOne() methods

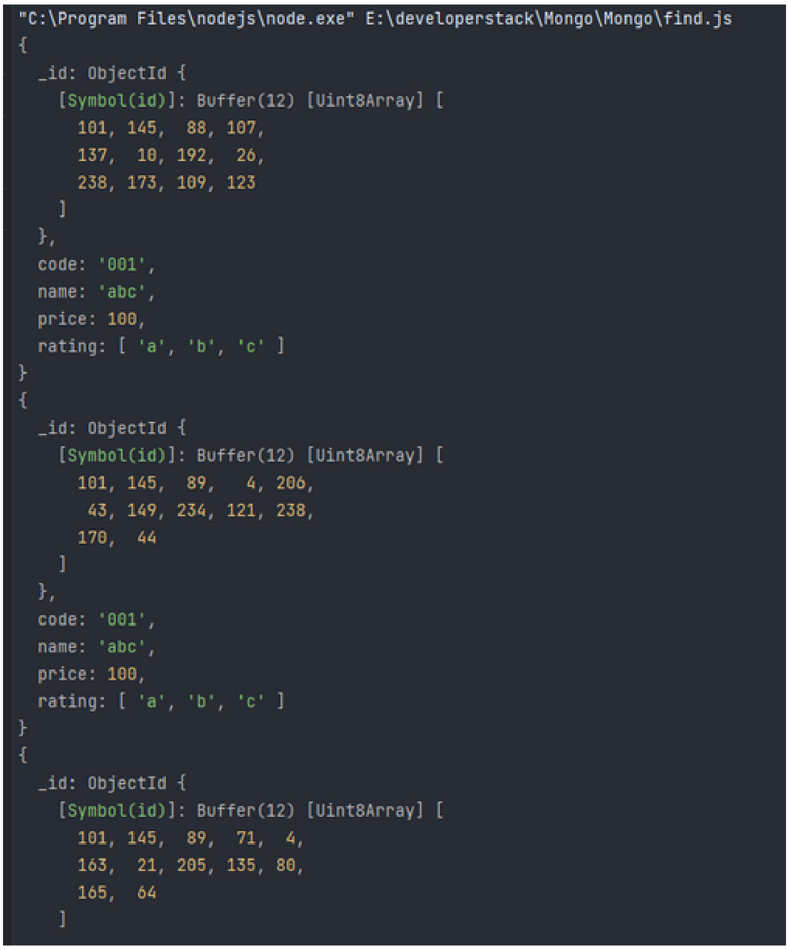


```
db.posts.find( {category: "News"} )
```

find all product with name - findByName(name)

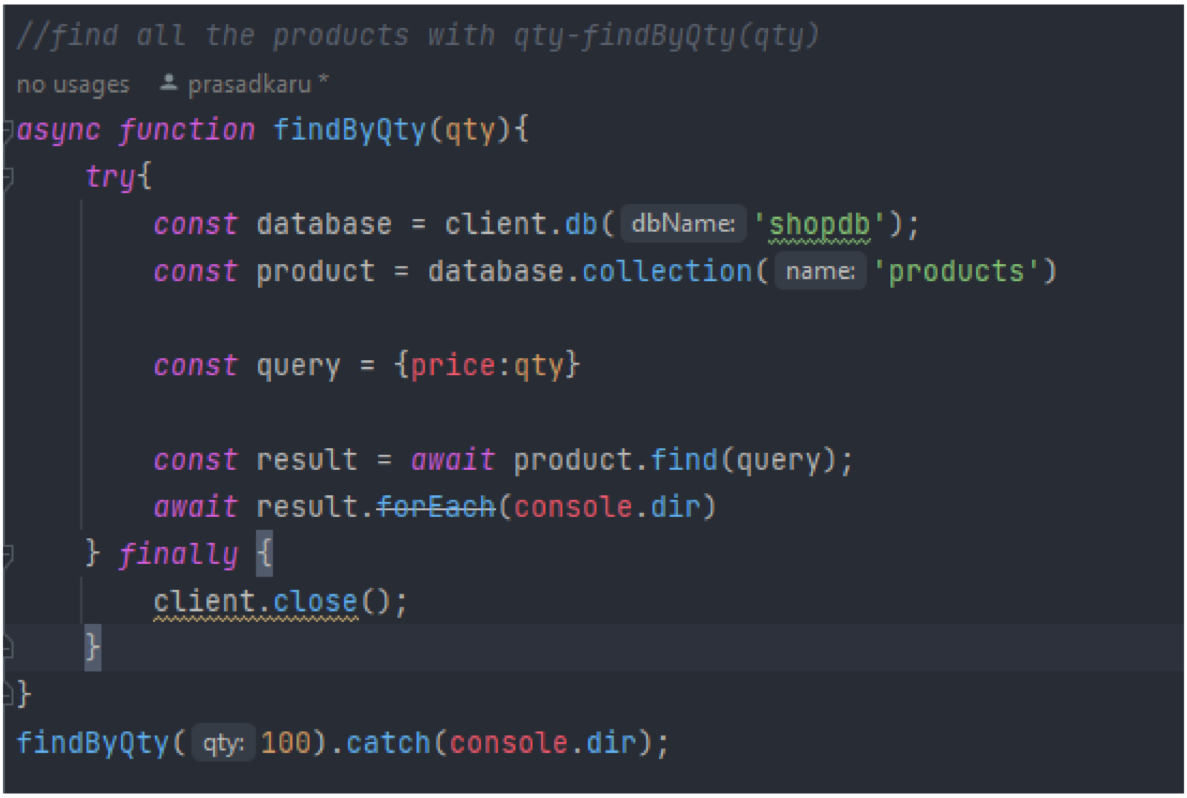


```
//find all product with name - findByName(name)
no usages
async function findByName(name){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:name}

        const result = await product.find(query);
        await result.~~forEach~~(console.dir)
    } finally {
        client.close();
    }
}
findByName( name: "abc").catch(console.dir);
```

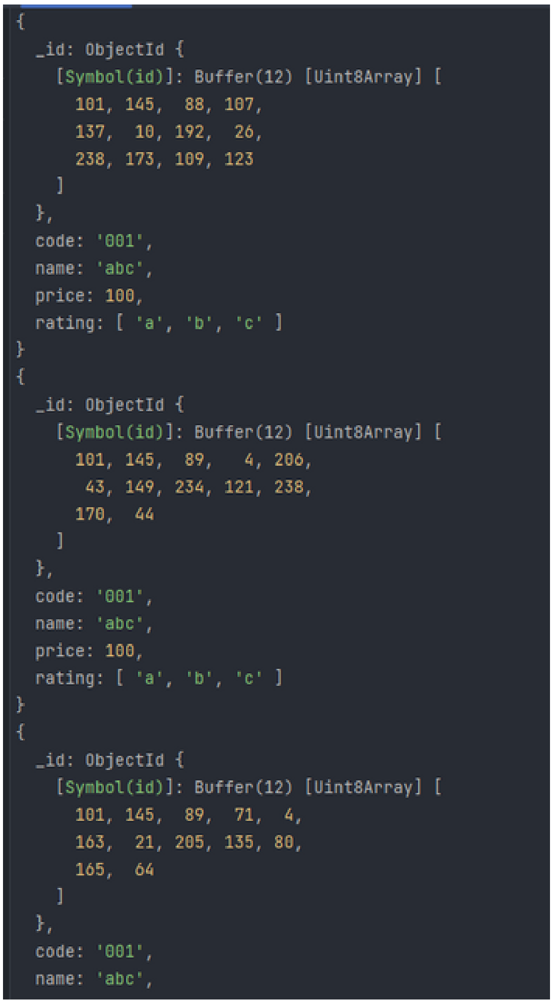


```
"C:\Program Files\nodejs\node.exe" E:\developerstack\Mongo\Mongo\find.js
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  88, 107,
      137,  10, 192,  26,
      238, 173, 109, 123
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  89,   4, 206,
       43, 149, 234, 121, 238,
      170,  44
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  89,  71,   4,
      163,  21, 205, 135,  80,
      165,  64
    ]
```

find all the products with qty-findByQty(qty)

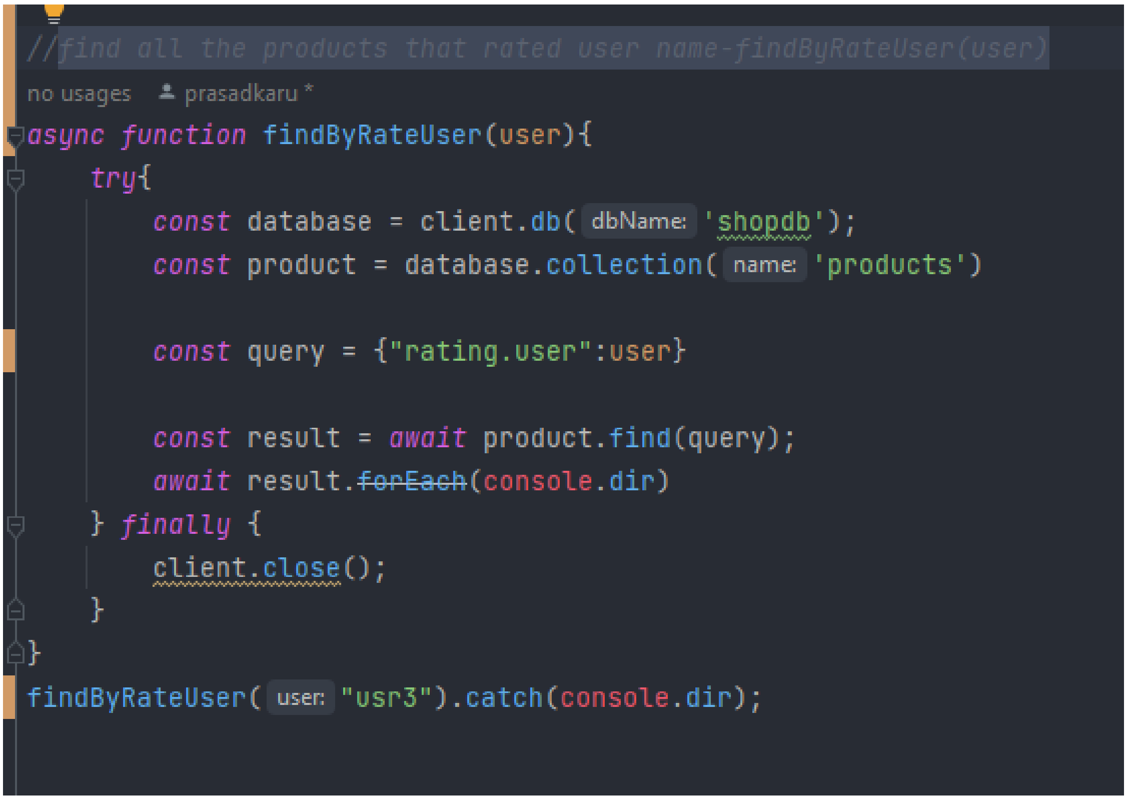


```
//find all the products with qty-findByQty(qty)
no usages   prasadkaru *
async function findByQty(qty){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {price:qty}

        const result = await product.find(query);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
findByQty( qty: 100).catch(console.dir);
```

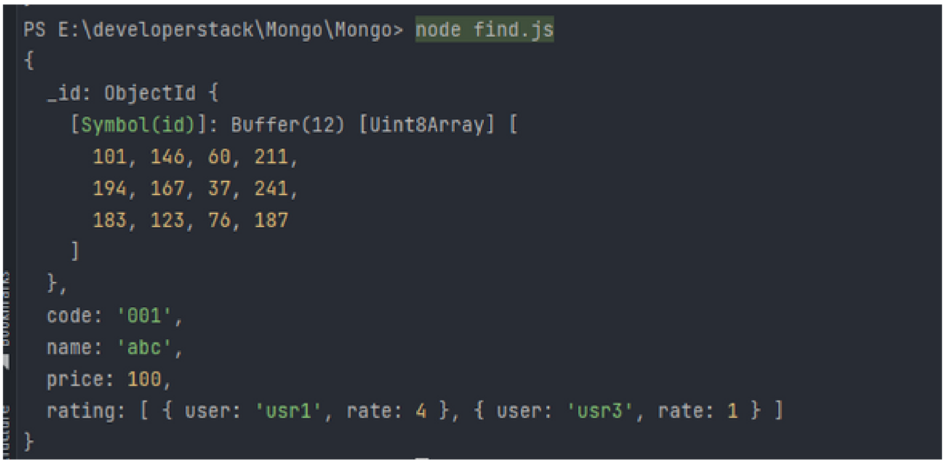


```
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  88, 107,
      137,  10, 192,  26,
      238, 173, 109, 123
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  89,   4, 206,
       43, 149, 234, 121, 238,
      170,  44
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  89,  71,   4,
      163,  21, 205, 135, 80,
      165,  64
    ]
  },
  code: '001',
  name: 'abc',
```

find all the products that rated user name-findByRateUser(user)

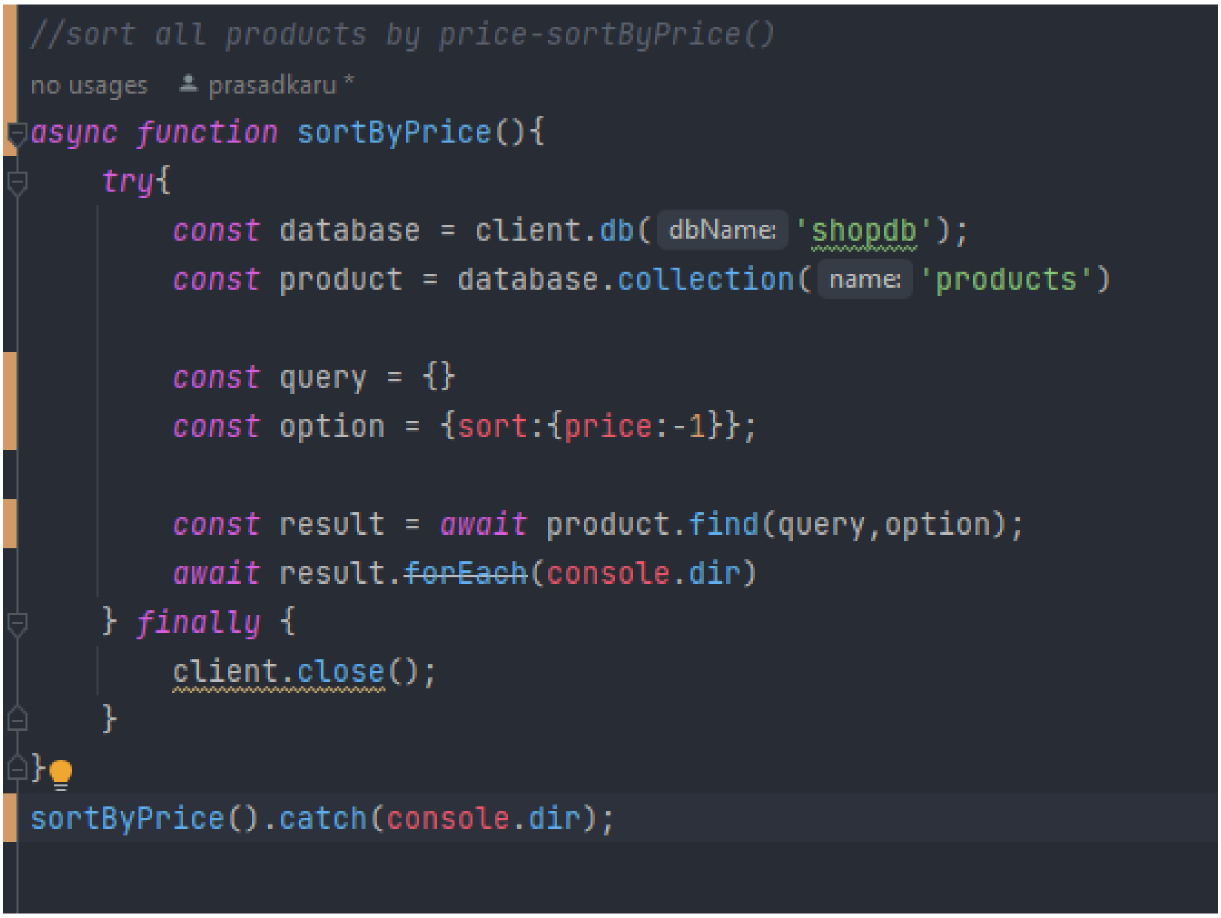


```
//find all the products that rated user name-findByRateUser(user)
no usages   prasadkaru *
async function findByRateUser(user){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {"rating.user":user}

        const result = await product.find(query);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
findByRateUser( user: "usr3").catch(console.dir);
```



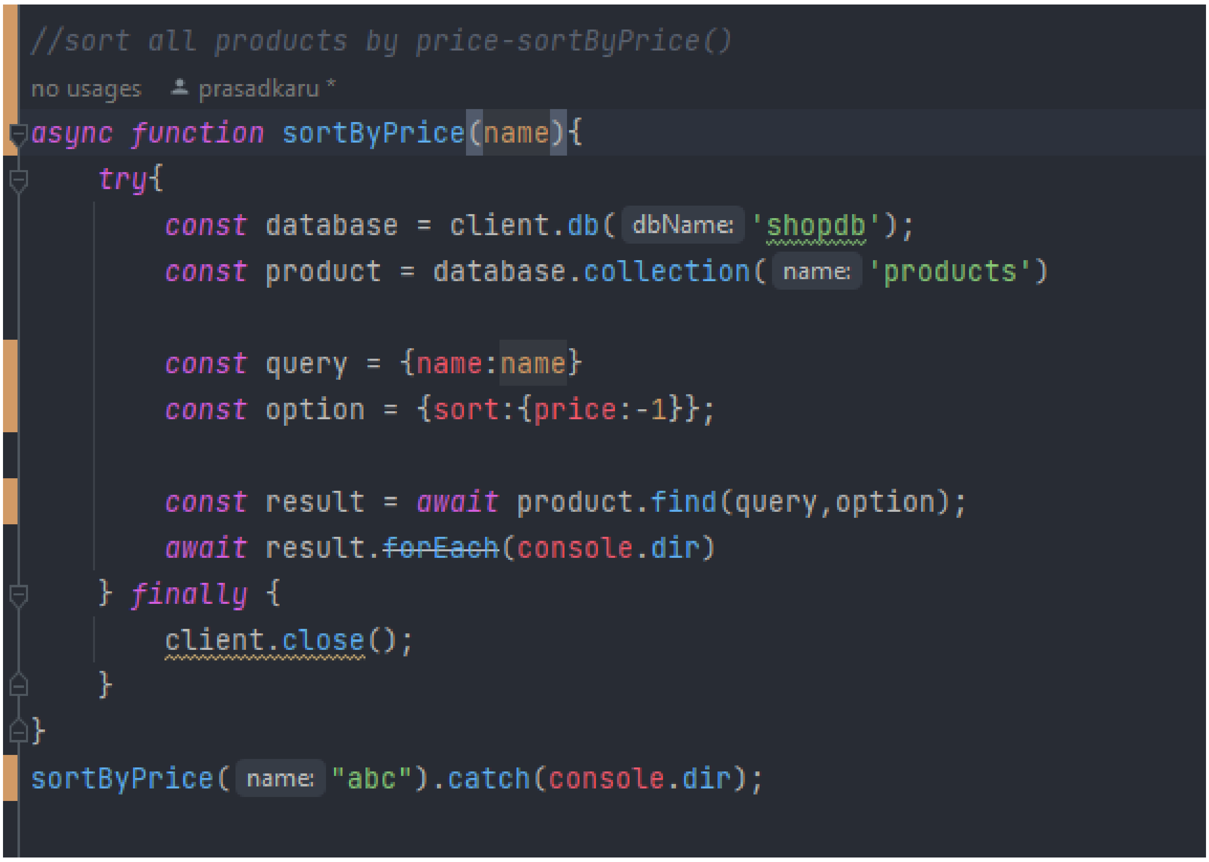
```
PS E:\developerstack\Mongo\Mongo> node find.js
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 146, 60, 211,
      194, 167, 37, 241,
      183, 123, 76, 187
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ { user: 'usr1', rate: 4 }, { user: 'usr3', rate: 1 } ]
}
```

sort all products by price-sortByPrice()

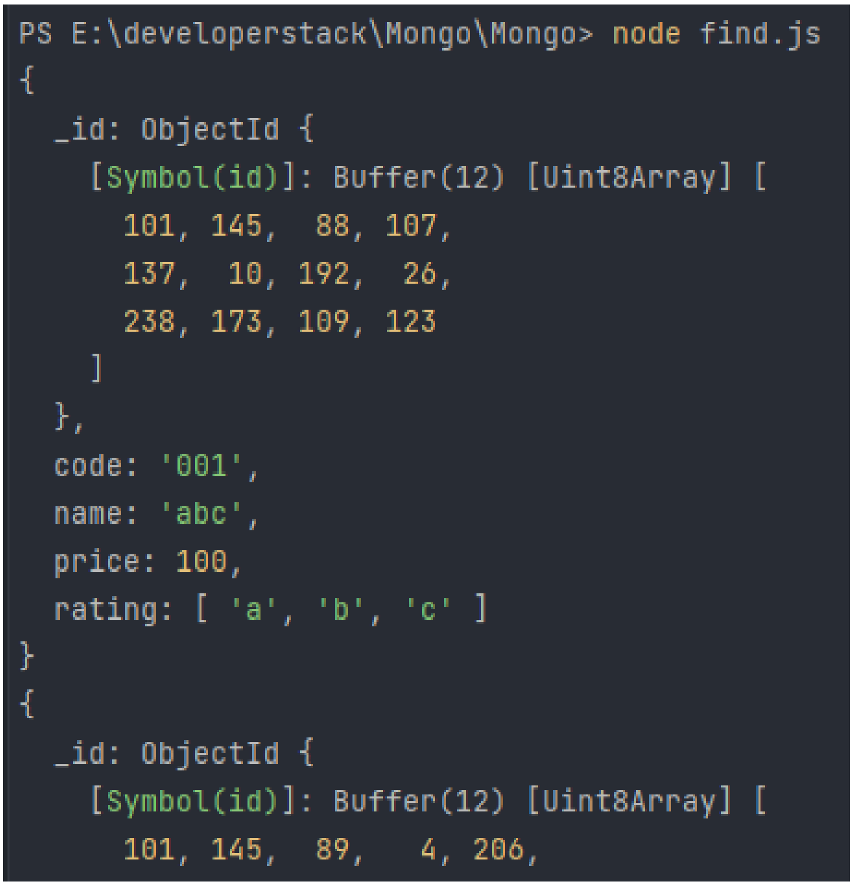


```
//sort all products by price-sortByPrice()
no usages  prasadkaru *
async function sortByPrice(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {}
        const option = {sort:{price:-1}};

        const result = await product.find(query,option);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
sortByPrice().catch(console.dir);
```

filter name and after sort



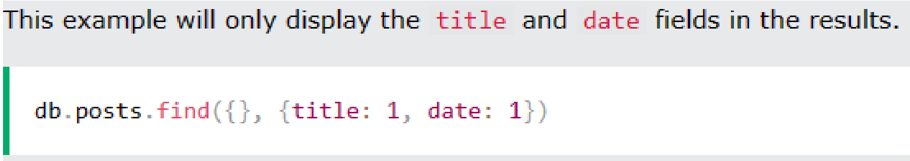
```
//sort all products by price-sortByPrice()
no usages  prasadkaru *
async function sortByPrice(name){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:name}
        const option = {sort:{price:-1}};

        const result = await product.find(query,option);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
sortByPrice( name: "abc").catch(console.dir);
```



```
PS E:\developerstack\Mongo\Mongo> node find.js
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  88, 107,
      137,  10, 192,  26,
      238, 173, 109, 123
    ]
  },
  code: '001',
  name: 'abc',
  price: 100,
  rating: [ 'a', 'b', 'c' ]
}
{
  _id: ObjectId {
    [Symbol(id)]: Buffer(12) [Uint8Array] [
      101, 145,  89,   4, 206,
```

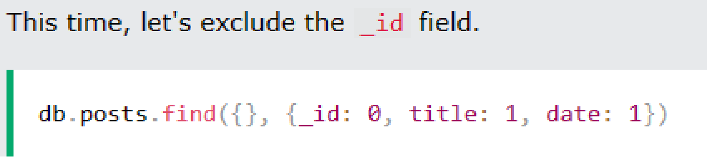
filter fields in all products-filterFields()

Both find methods accept a second parameter called projection.



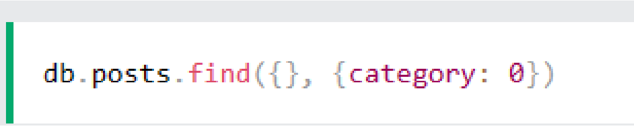
This example will only display the `title` and `date` fields in the results.

```
db.posts.find({}, {title: 1, date: 1})
```

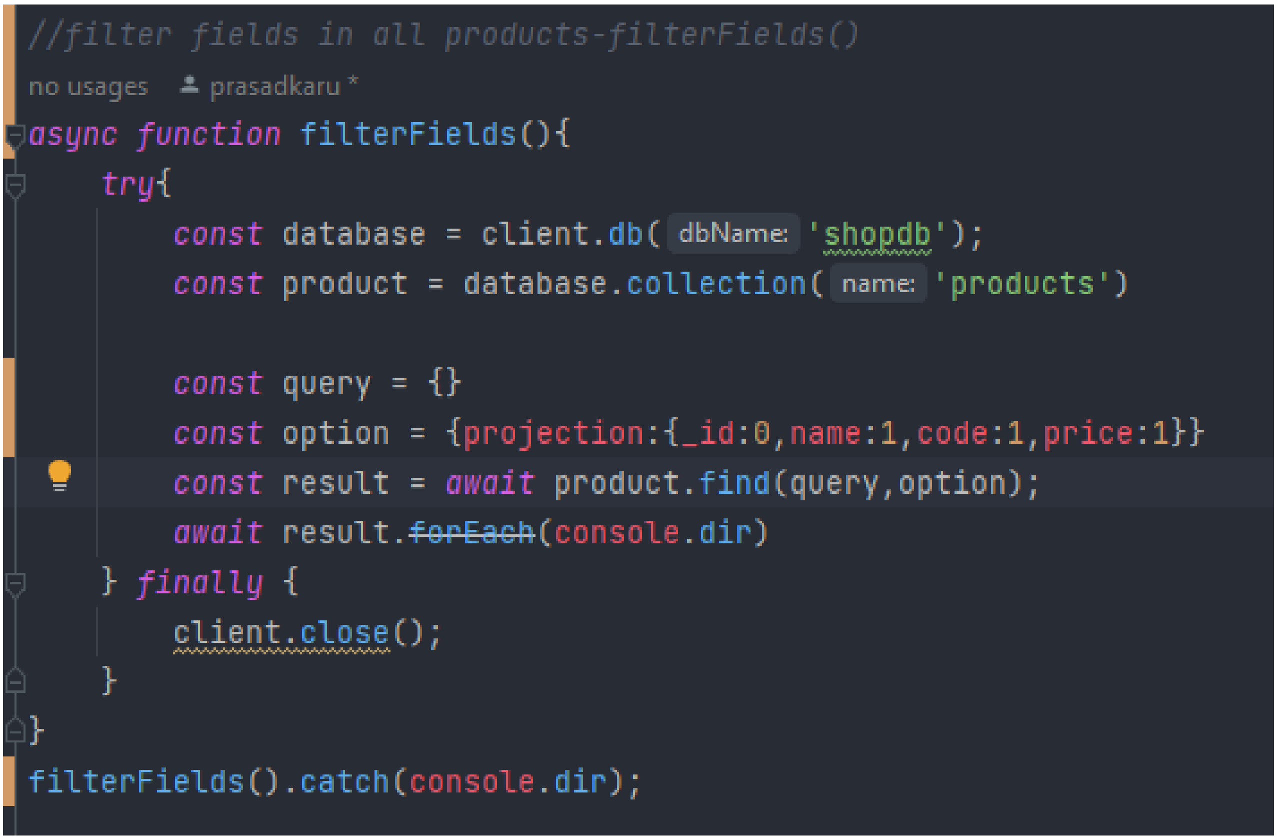


This time, let's exclude the `_id` field.

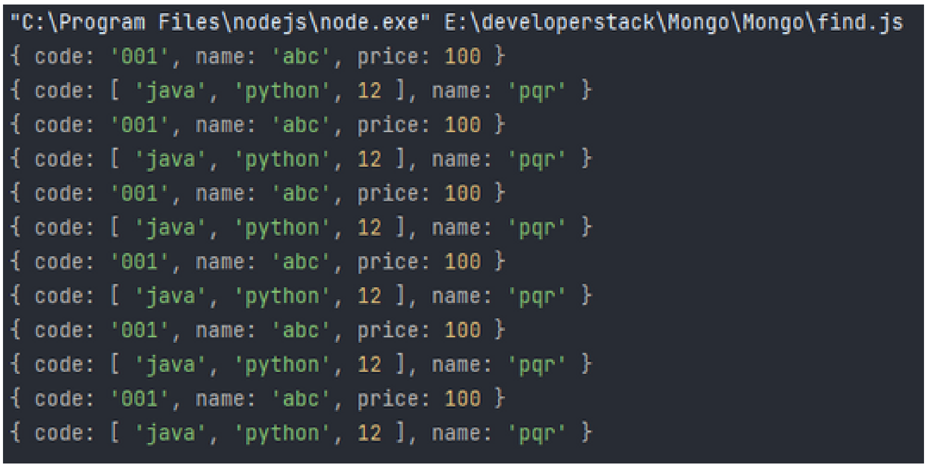
```
db.posts.find({}, {_id: 0, title: 1, date: 1})
```



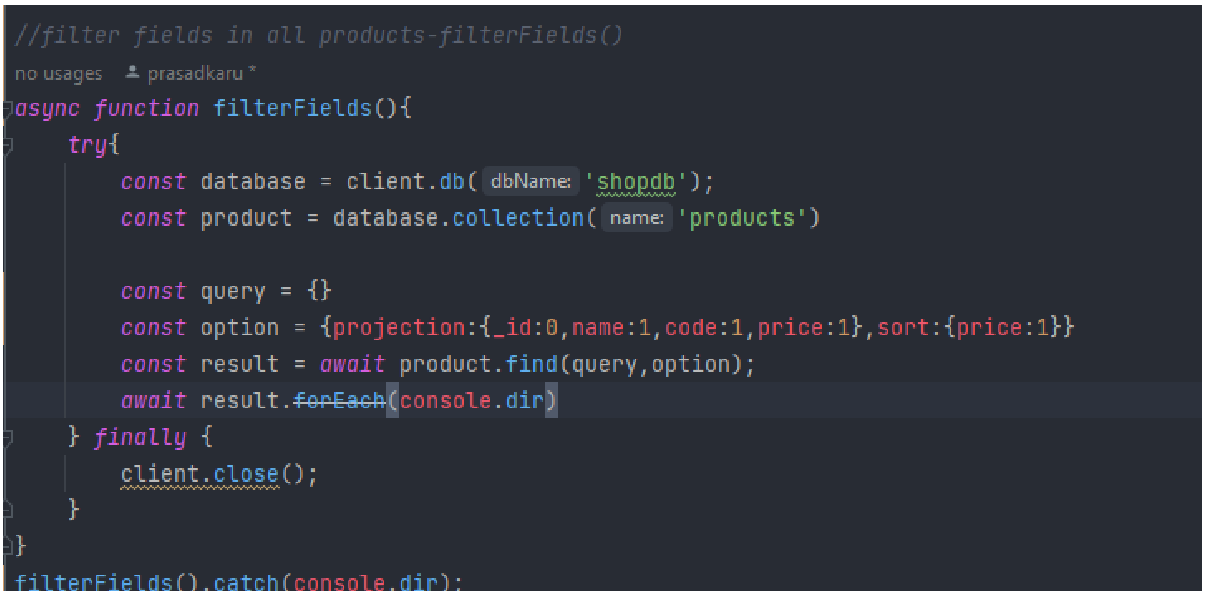
```
db.posts.find({}, {category: 0})
```



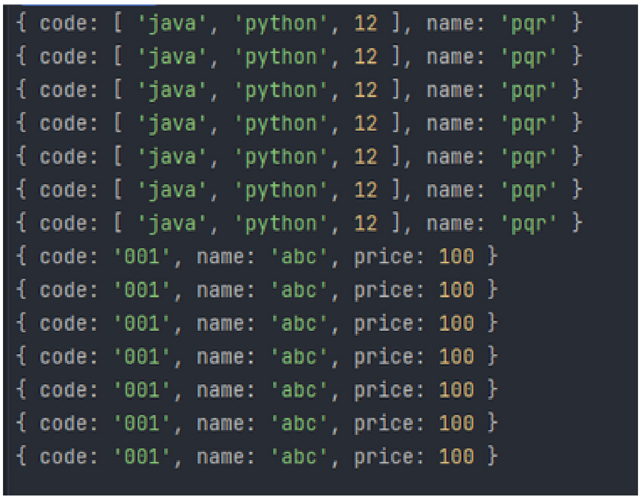
```
//filter fields in all products-filterFields()
no usages  prasadkaru *
async function filterFields(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {}
        const option = {projection:{_id:0,name:1,code:1,price:1}}
        const result = await product.find(query,option);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
filterFields().catch(console.dir);
```



```
"C:\Program Files\nodejs\node.exe" E:\developerstack\Mongo\Mongo\find.js
{ code: '001', name: 'abc', price: 100 }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: '001', name: 'abc', price: 100 }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: '001', name: 'abc', price: 100 }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: '001', name: 'abc', price: 100 }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: '001', name: 'abc', price: 100 }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: '001', name: 'abc', price: 100 }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
```

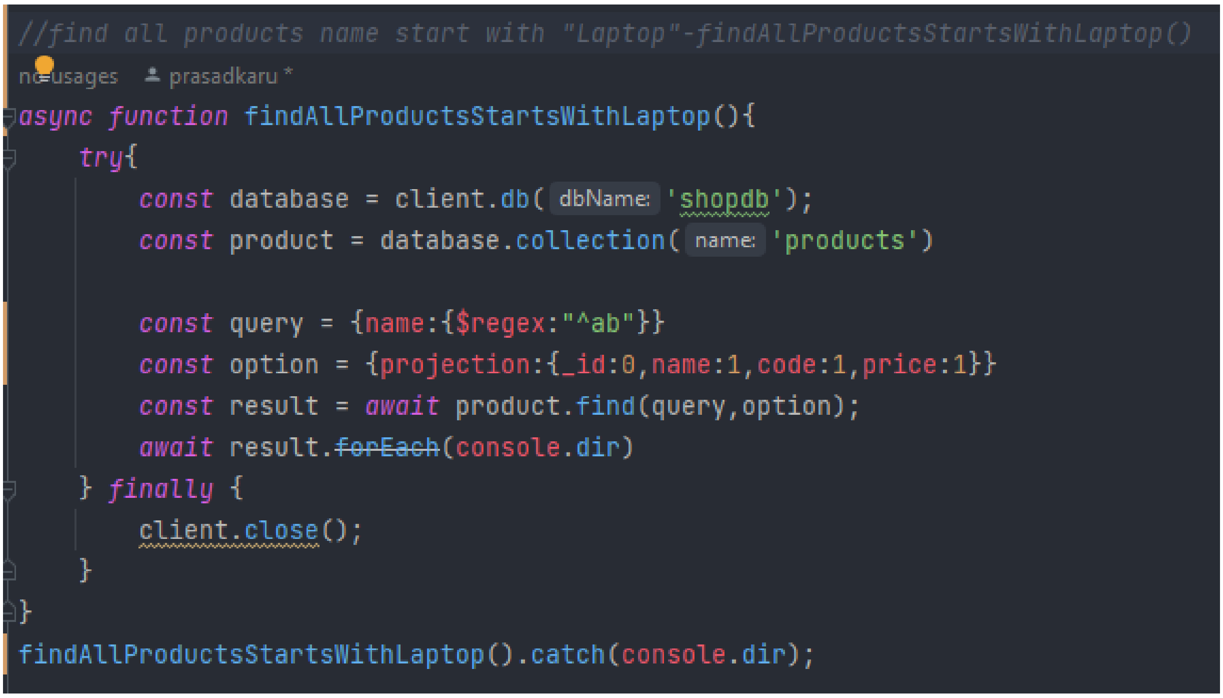


```
//filter fields in all products-filterFields()
no usages  prasadkaru *
async function filterFields(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {}
        const option = {projection:{_id:0,name:1,code:1,price:1},sort:{price:1}}
        const result = await product.find(query,option);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
filterFields().catch(console.dir);
```

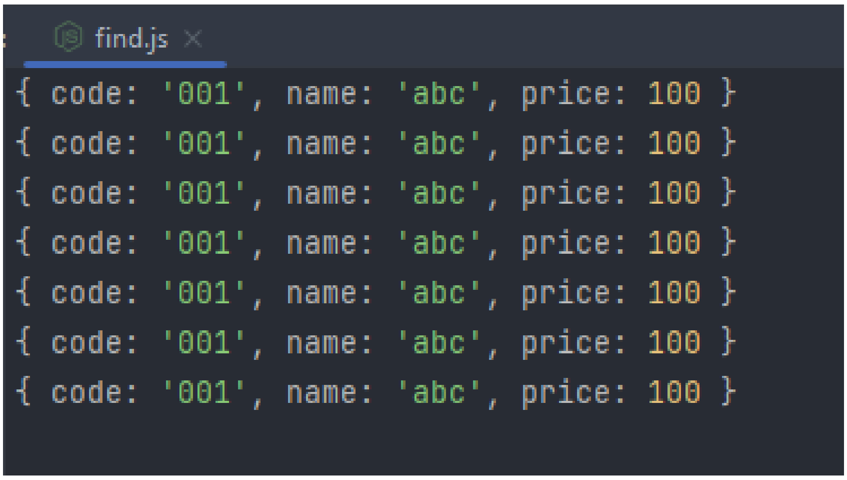


```
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
```

find all products name start with "Laptop"-findAllProductsStartsWithLaptop()



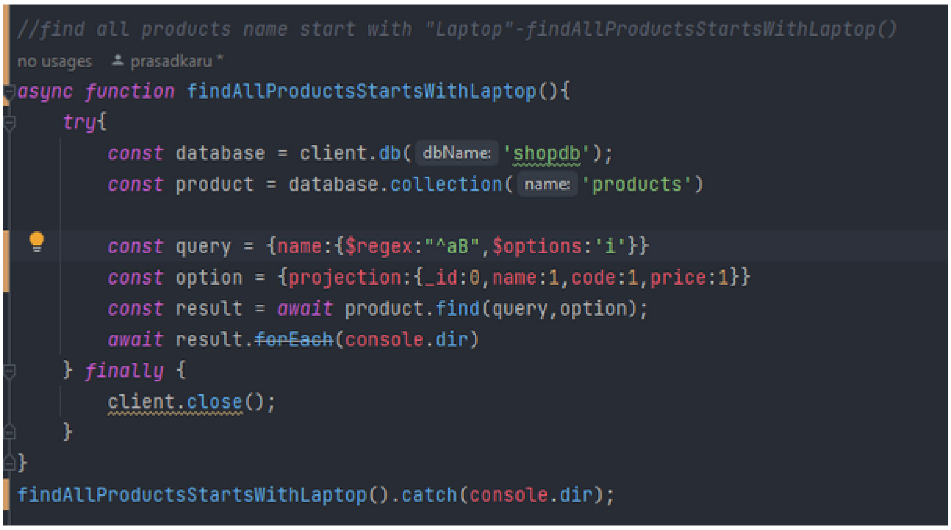
```
//find all products name start with "Laptop"-findAllProductsStartsWithLaptop()
no usages  prasadkaru *
async function findAllProductsStartsWithLaptop(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:{$regex:"^ab"}}
        const option = {projection:{_id:0,name:1,code:1,price:1}}
        const result = await product.find(query,option);
        await result.~~forEach~~(console.dir)
    } finally {
        client.close();
    }
}
findAllProductsStartsWithLaptop().catch(console.dir);
```



find.js

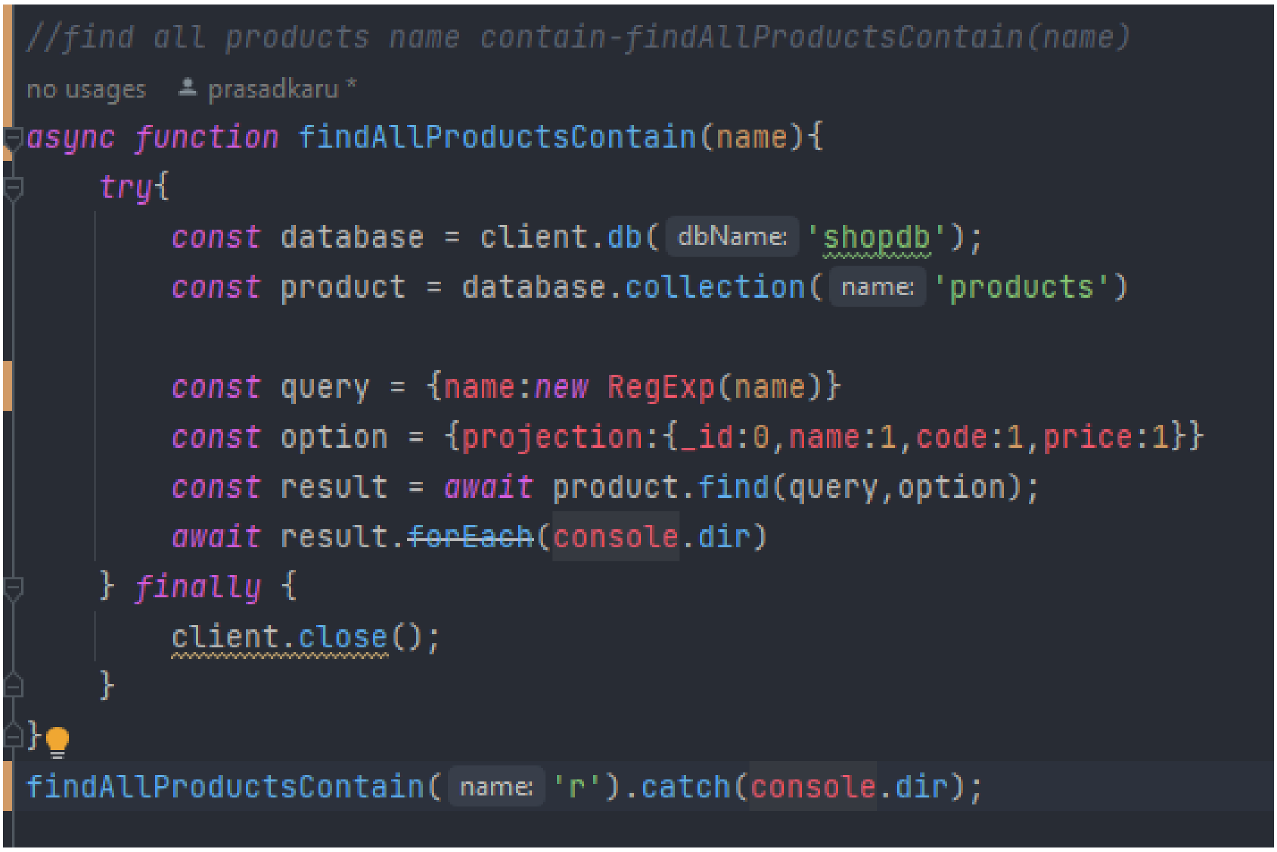
```
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
```

Without case sensative

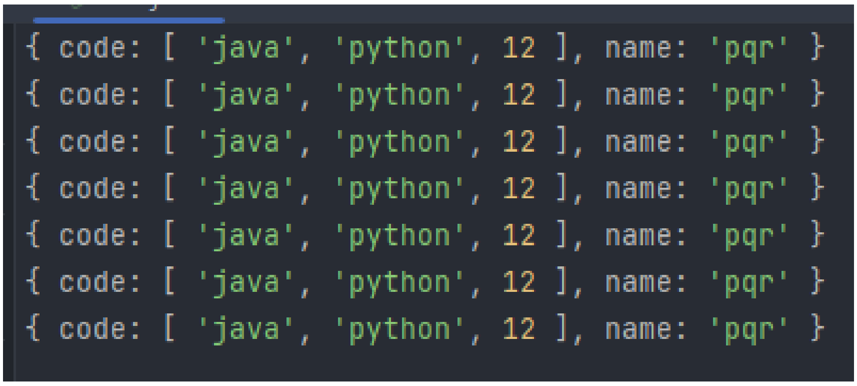


```
//find all products name start with "Laptop"-findAllProductsStartsWithLaptop()
no usages  prasadkaru *
async function findAllProductsStartsWithLaptop(){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:{$regex:"^aB",$options:'i'}}
        const option = {projection:{_id:0,name:1,code:1,price:1}}
        const result = await product.find(query,option);
        await result.~~forEach~~(console.dir)
    } finally {
        client.close();
    }
}
findAllProductsStartsWithLaptop().catch(console.dir);
```

//find all products name contain-findAllProductsContain(name)

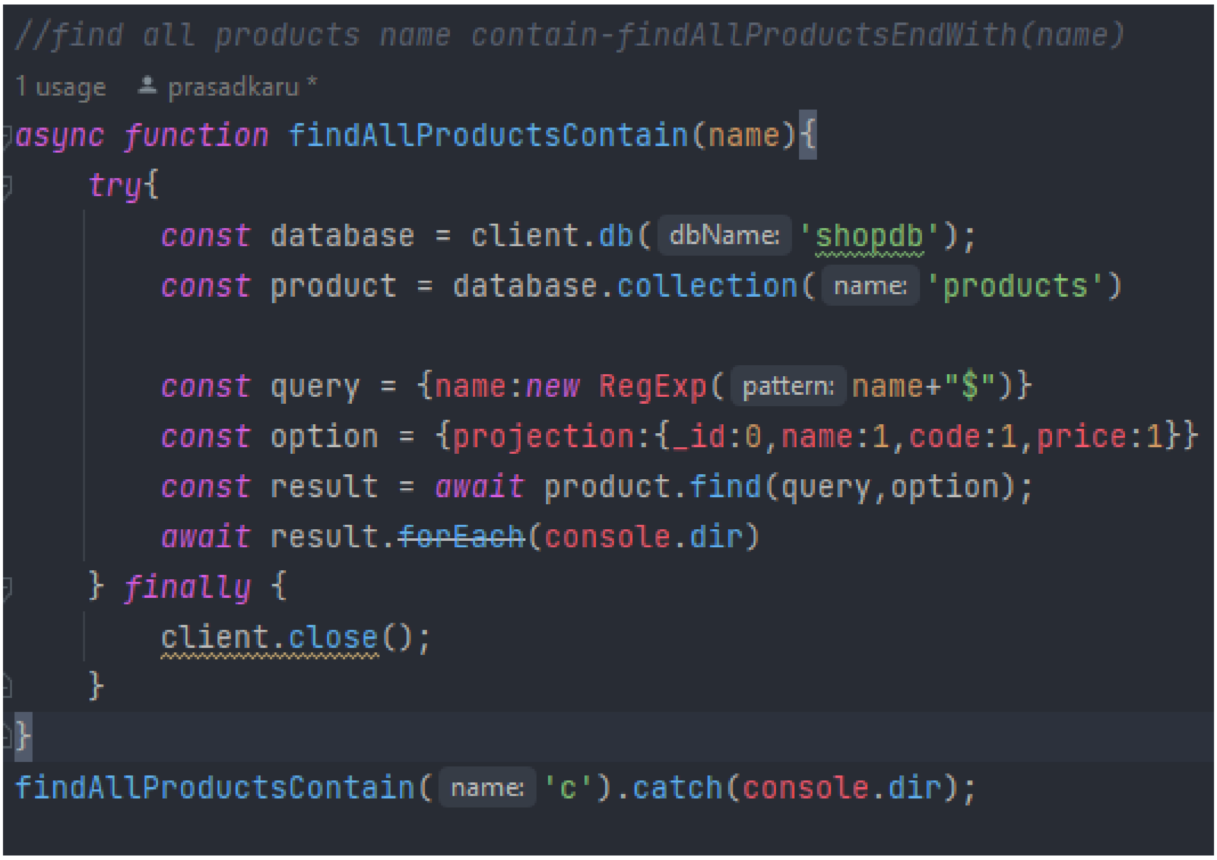


```
//find all products name contain-findAllProductsContain(name)
no usages  prasadkaru *
async function findAllProductsContain(name){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:new RegExp(name)}
        const option = {projection:{_id:0,name:1,code:1,price:1}}
        const result = await product.find(query,option);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
findAllProductsContain( name: 'r').catch(console.dir);
```

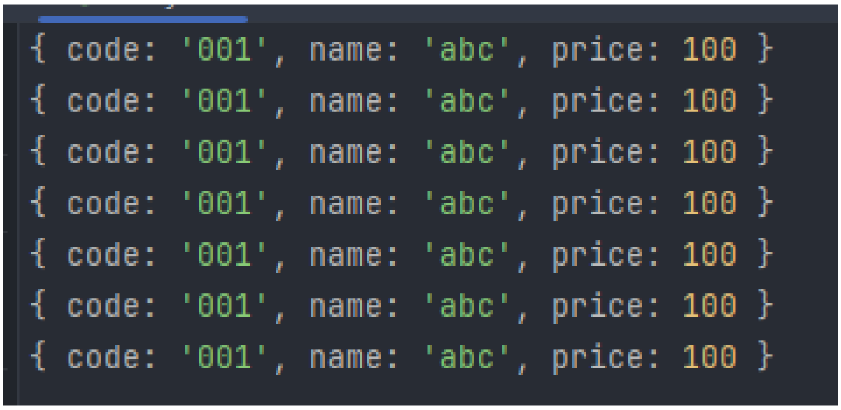


```
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
```

//find all products name contain end-findAllProductsEndWith(name)

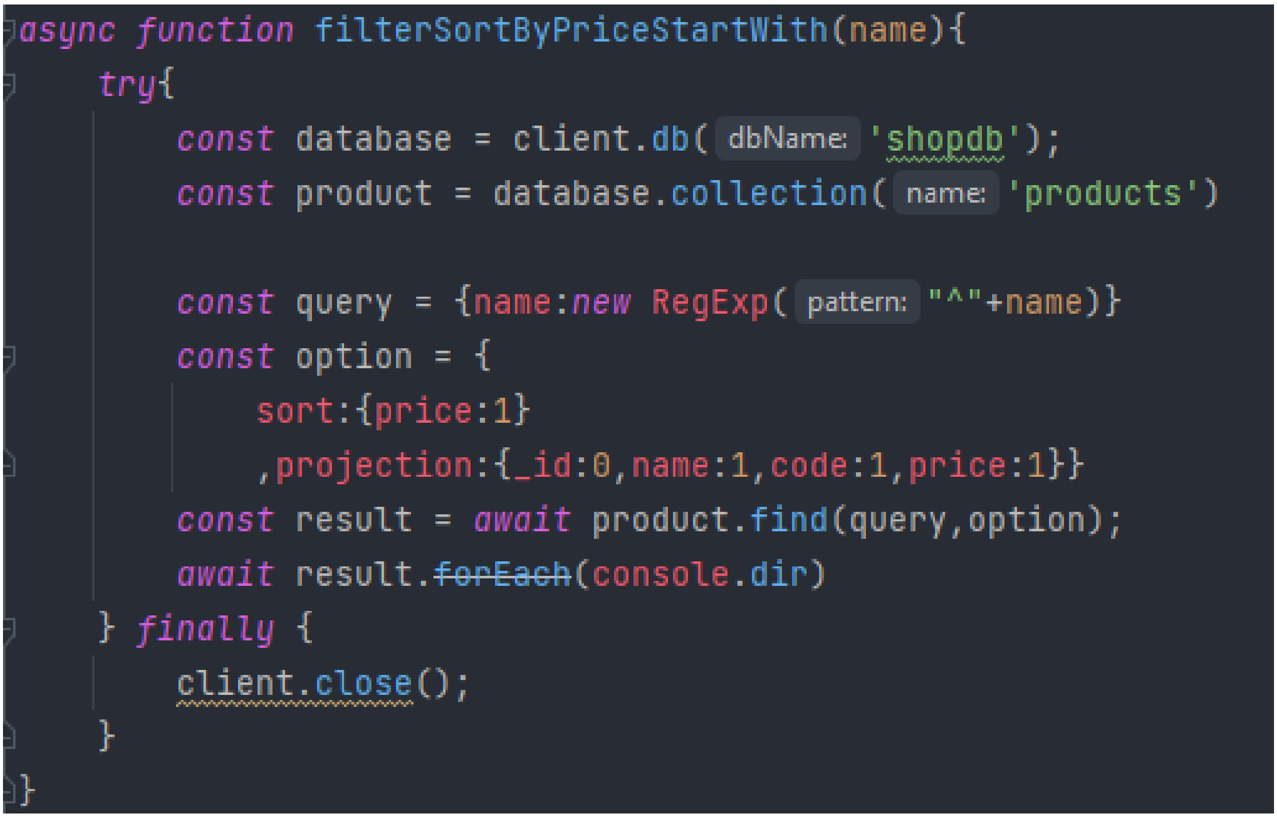


```
//find all products name contain-findAllProductsEndWith(name)
1 usage   prasadkaru *
async function findAllProductsContain(name){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:new RegExp( pattern: name+"$")}
        const option = {projection:{_id:0,name:1,code:1,price:1}}
        const result = await product.find(query,option);
        await result.~~forEach~~(console.dir)
    } finally {
        client.close();
    }
}
findAllProductsContain( name: 'c').catch(console.dir);
```

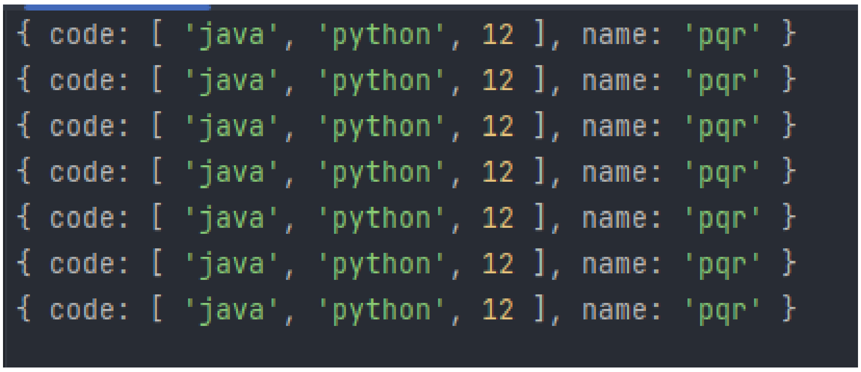


```
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
{ code: '001', name: 'abc', price: 100 }
```

//sort and filter all products by price,stats with –filterSortByPriceStartWith(name)



```
async function filterSortByPriceStartWith(name){
    try{
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = {name:new RegExp( pattern: "^"+name)}
        const option = {
            sort:{price:1}
            ,projection:{_id:0,name:1,code:1,price:1}}
        const result = await product.find(query,option);
        await result.~~forEach~~(console.dir)
    } finally {
        client.close();
    }
}
```

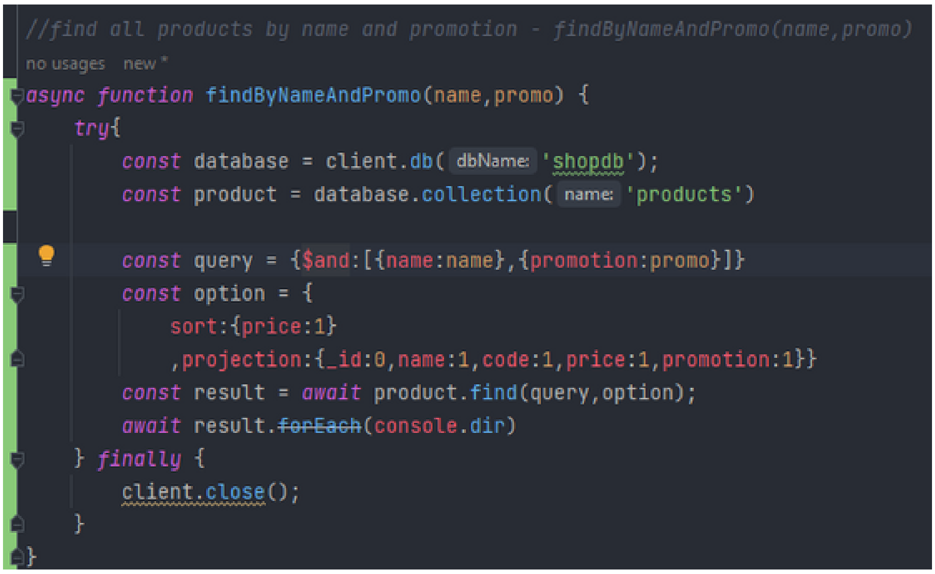


```
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
{ code: [ 'java', 'python', 12 ], name: 'pqr' }
```

Logical

The following operators can logically compare multiple queries.

- `$and` : Returns documents where both queries match
- `$or` : Returns documents where either query matches
- `$nor` : Returns documents where both queries fail to match
- `$not` : Returns documents where the query does not match

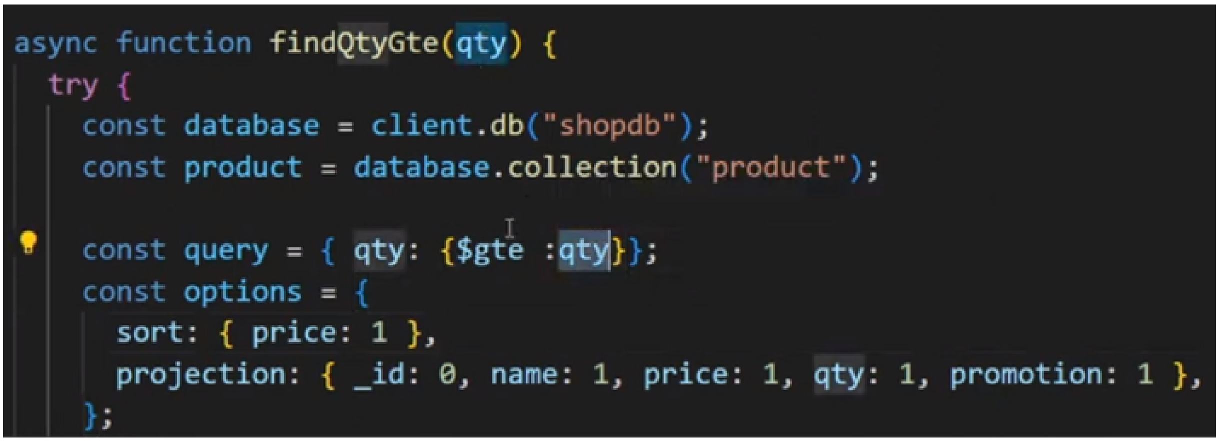


```
//find all products by name and promotion - findByNameAndPromo(name,promo)
async function findByNameAndPromo(name,promo) {
    try{
        const database = client.db('shopdb');
        const product = database.collection('products')

        const query = {$and:[{name:name},{promotion:promo}]}
        const option = {
            sort:{price:1}
            ,projection:{_id:0,name:1,code:1,price:1,promotion:1}}
        const result = await product.find(query,option);
        await result.forEach(console.dir)
    } finally {
        client.close();
    }
}
```

Comparison

The following operators can be used in queries to compare values:

- `$eq` : Values are equal
- `$ne` : Values are not equal
- `$gt` : Value is greater than another value
- `$gte` : Value is greater than or equal to another value
- `$lt` : Value is less than another value
- `$lte` : Value is less than or equal to another value
- `$in` : Value is matched within an array

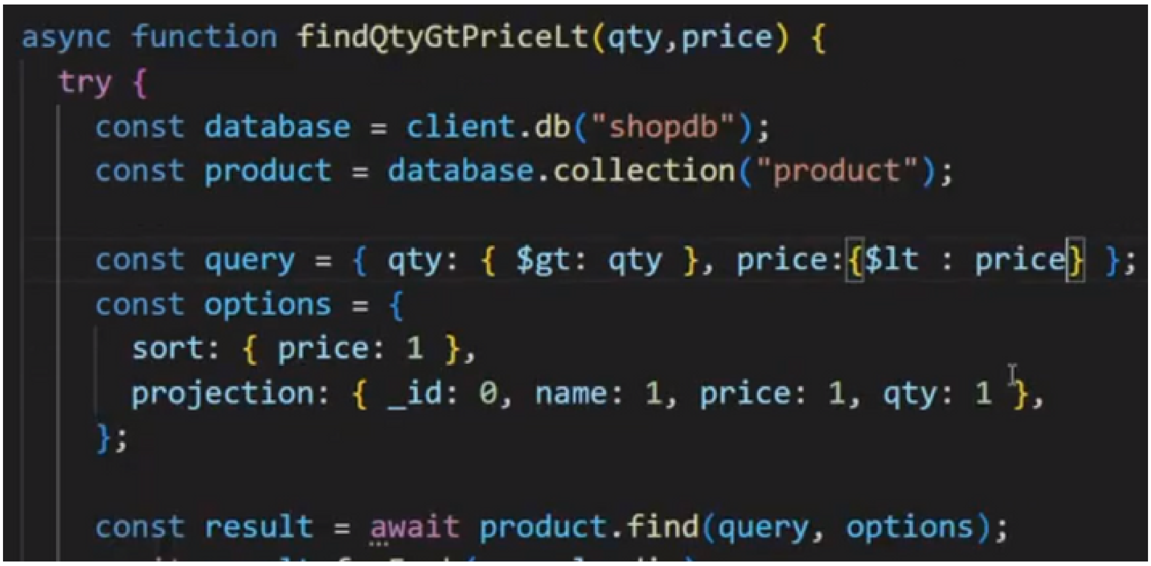


```
async function findQtyGte(qty) {
  try {
    const database = client.db("shopdb");
    const product = database.collection("product");

    const query = { qty: {$gte :qty}};
    const options = {
      sort: { price: 1 },
      projection: { _id: 0, name: 1, price: 1, qty: 1, promotion: 1 },
    };
```

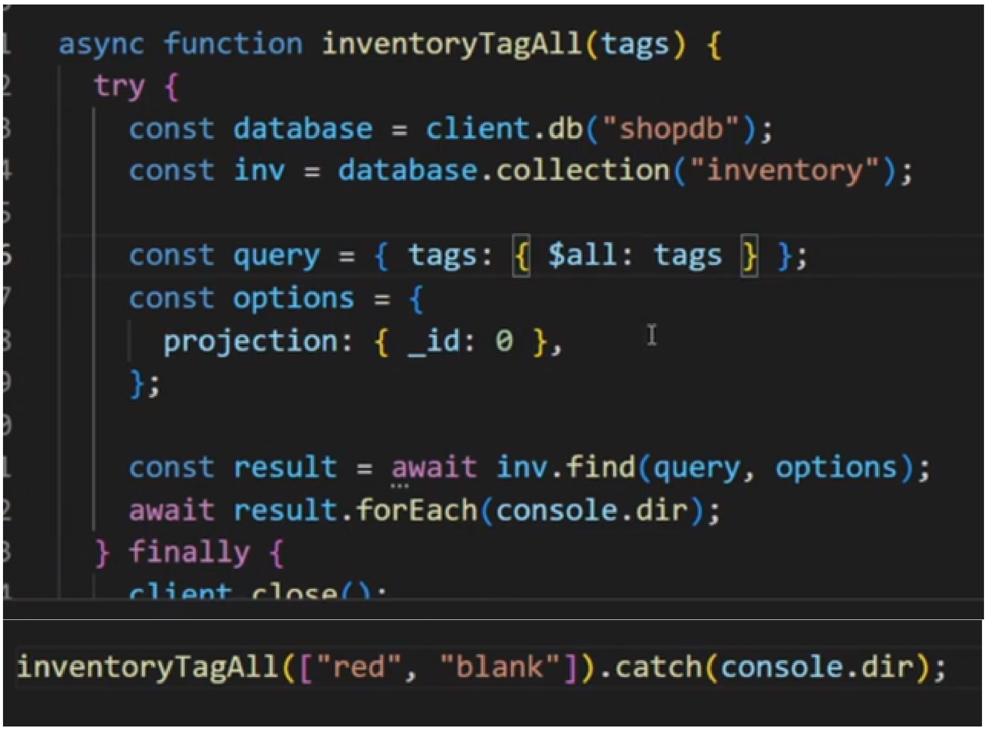
Comparison

The following operators can be used in queries to compare values:

- `$eq` : Values are equal
- `$ne` : Values are not equal
- `$gt` : Value is greater than another value
- `$gte` : Value is greater than or equal to another value
- `$lt` : Value is less than another value
- `$lte` : Value is less than or equal to another value
- `$in` : Value is matched within an array

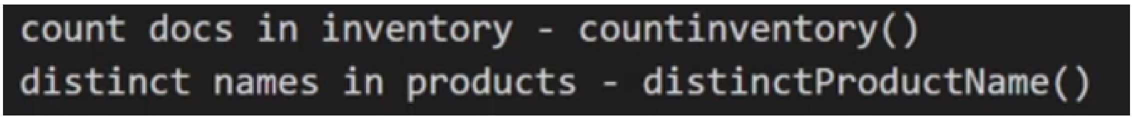


```
async function findQtyGtPriceLt(qty,price) {
  try {
    const database = client.db("shopdb");
    const product = database.collection("product");

    const query = { qty: { $gt: qty }, price:{$lt : price} };
    const options = {
      sort: { price: 1 },
      projection: { _id: 0, name: 1, price: 1, qty: 1 },
    };

    const result = await product.find(query, options);
```

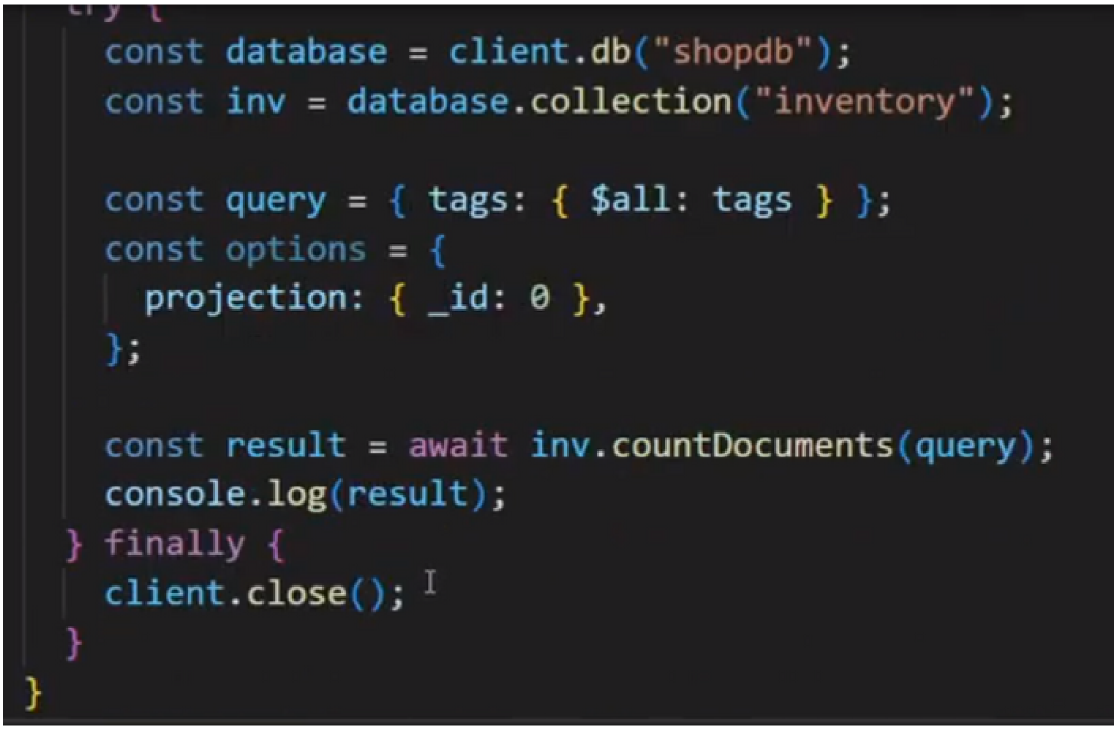


```
async function inventoryTagAll(tags) {
  try {
    const database = client.db("shopdb");
    const inv = database.collection("inventory");

    const query = { tags: { $all: tags } };
    const options = {
      projection: { _id: 0 },
    };

    const result = await inv.find(query, options);
    await result.forEach(console.dir);
  } finally {
    client.close();
```

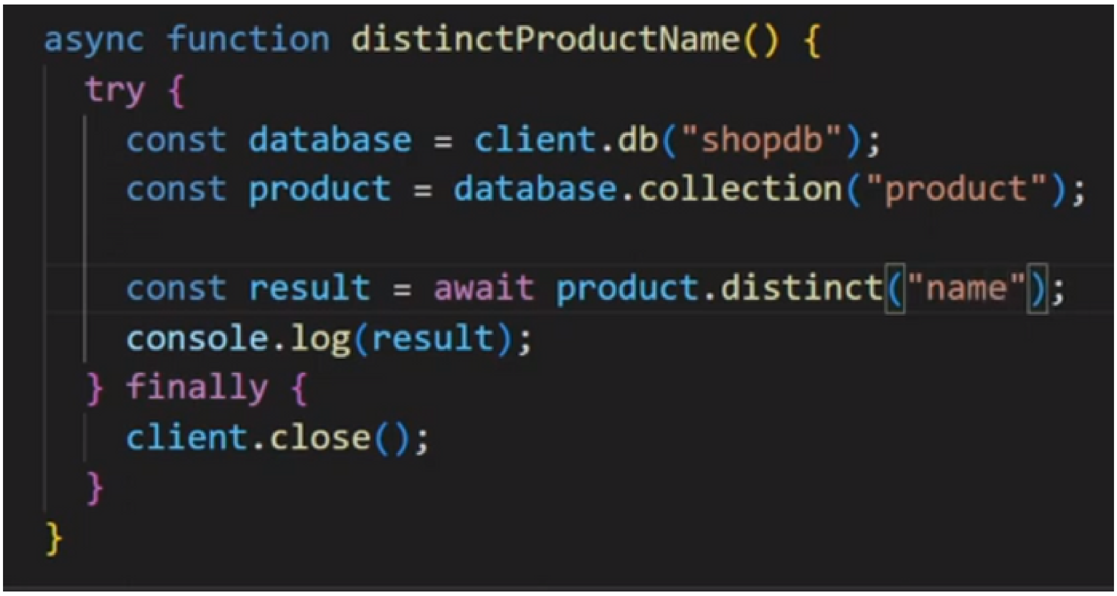
```
inventoryTagAll(["red", "blank"]).catch(console.dir);
```



```
count docs in inventory - countinventory()
distinct names in products - distinctProductName()
```



```
  try {
    const database = client.db("shopdb");
    const inv = database.collection("inventory");

    const query = { tags: { $all: tags } };
    const options = {
      projection: { _id: 0 },
    };

    const result = await inv.countDocuments(query);
    console.log(result);
  } finally {
    client.close();
  }
}
```

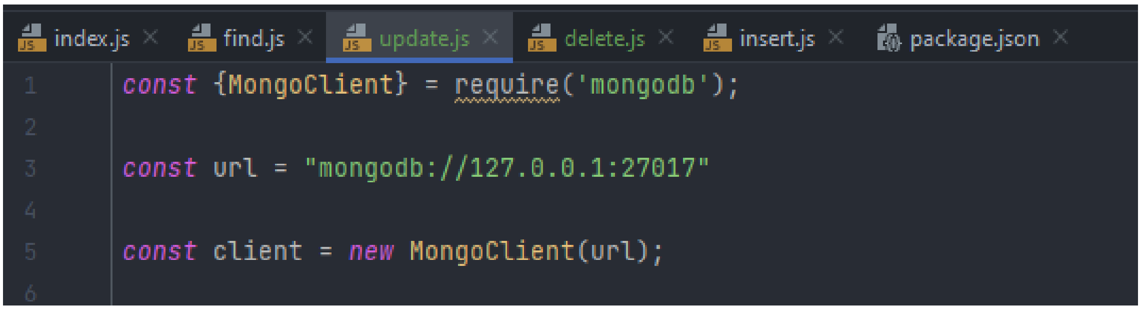


```
async function distinctProductName() {
  try {
    const database = client.db("shopdb");
    const product = database.collection("product");

    const result = await product.distinct("name");
    console.log(result);
  } finally {
    client.close();
  }
}
```

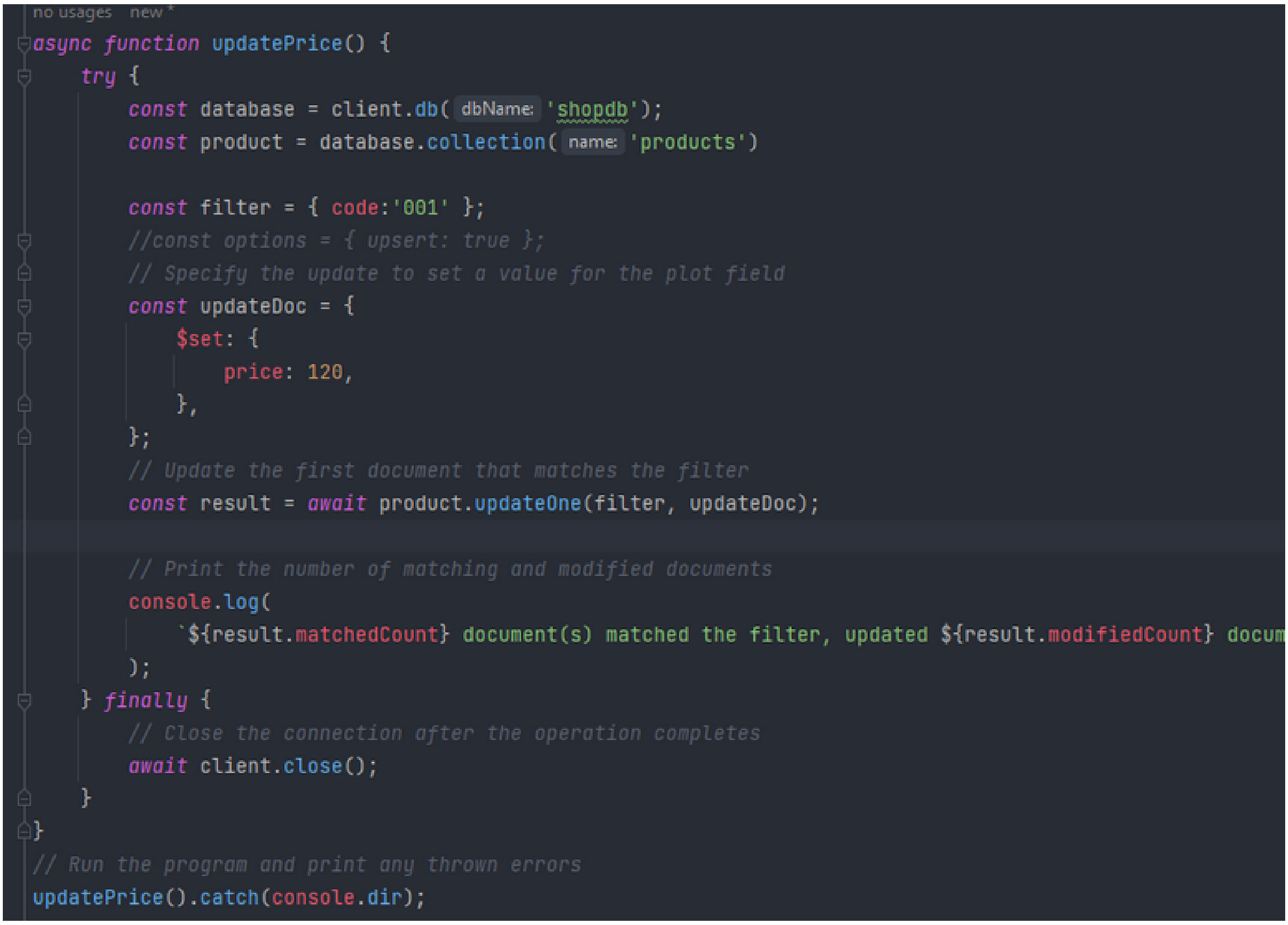
Update and Delete Documents

To update an existing document we can use the updateOne() or updateMany() methods.

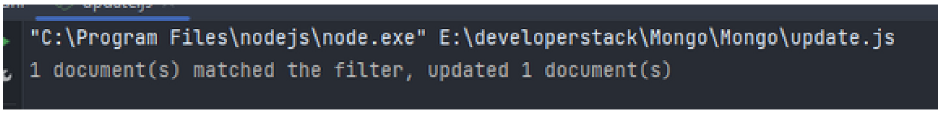
`updateOne()` `updateMany()`



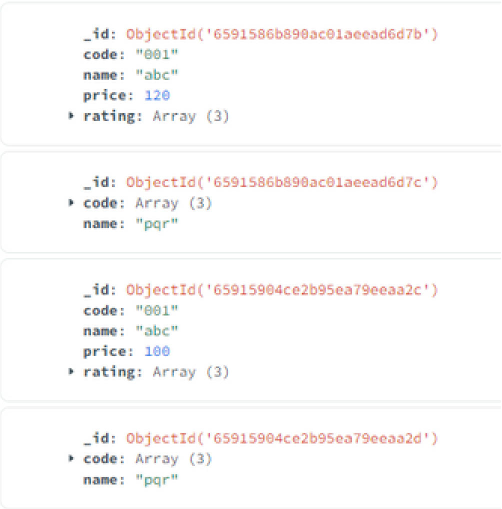
```
const {MongoClient} = require('mongodb');

const url = "mongodb://127.0.0.1:27017"

const client = new MongoClient(url);

```



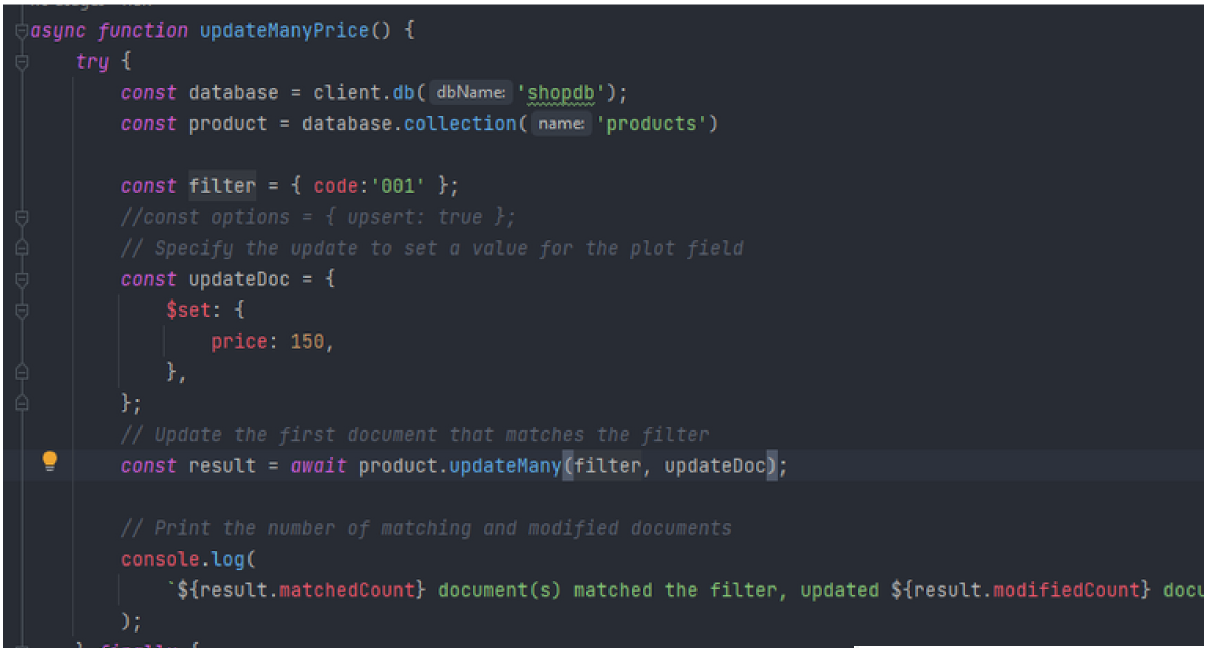
```
async function updatePrice() {
    try {
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const filter = { code:'001' };
        //const options = { upsert: true };
        // Specify the update to set a value for the plot field
        const updateDoc = {
            $set: {
                price: 120,
            },
        };
        // Update the first document that matches the filter
        const result = await product.updateOne(filter, updateDoc);

        // Print the number of matching and modified documents
        console.log(
            `${result.matchedCount} document(s) matched the filter, updated ${result.modifiedCount} docum
        );
    } finally {
        // Close the connection after the operation completes
        await client.close();
    }
}
// Run the program and print any thrown errors
updatePrice().catch(console.dir);
```



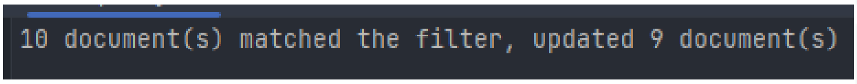
```
"C:\Program Files\nodejs\node.exe" E:\developerstack\Mongo\Mongo\update.js
1 document(s) matched the filter, updated 1 document(s)
```



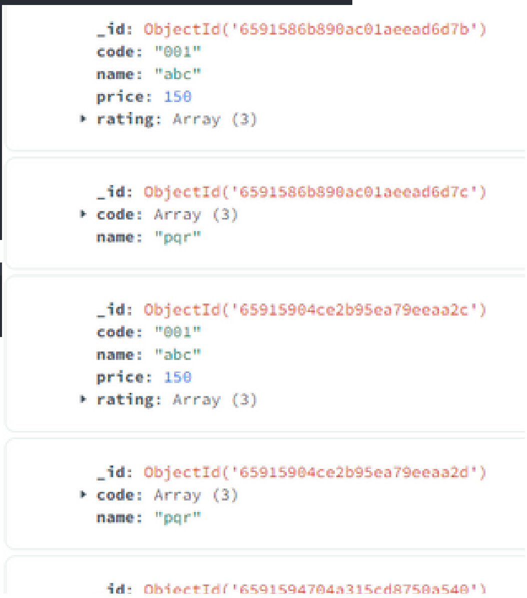
```
_id: ObjectId('6591586b890ac01aeead6d7b')
code: "001"
name: "abc"
price: 120
▸ rating: Array (3)

_id: ObjectId('6591586b890ac01aeead6d7c')
▸ code: Array (3)
name: "pqr"

_id: ObjectId('65915904ce2b95ea79eeaa2c')
code: "001"
name: "abc"
price: 100
▸ rating: Array (3)

_id: ObjectId('65915904ce2b95ea79eeaa2d')
▸ code: Array (3)
name: "pqr"
```



```
async function updateManyPrice() {
    try {
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const filter = { code:'001' };
        //const options = { upsert: true };
        // Specify the update to set a value for the plot field
        const updateDoc = {
            $set: {
                price: 150,
            },
        };
        // Update the first document that matches the filter
        const result = await product.updateMany(filter, updateDoc);

        // Print the number of matching and modified documents
        console.log(
            `${result.matchedCount} document(s) matched the filter, updated ${result.modifiedCount} docu
        );
    } finally {
        // Close the connection after the operation completes
        await client.close();
    }
}
// Run the program and print any thrown errors
updateManyPrice().catch(console.dir);
```

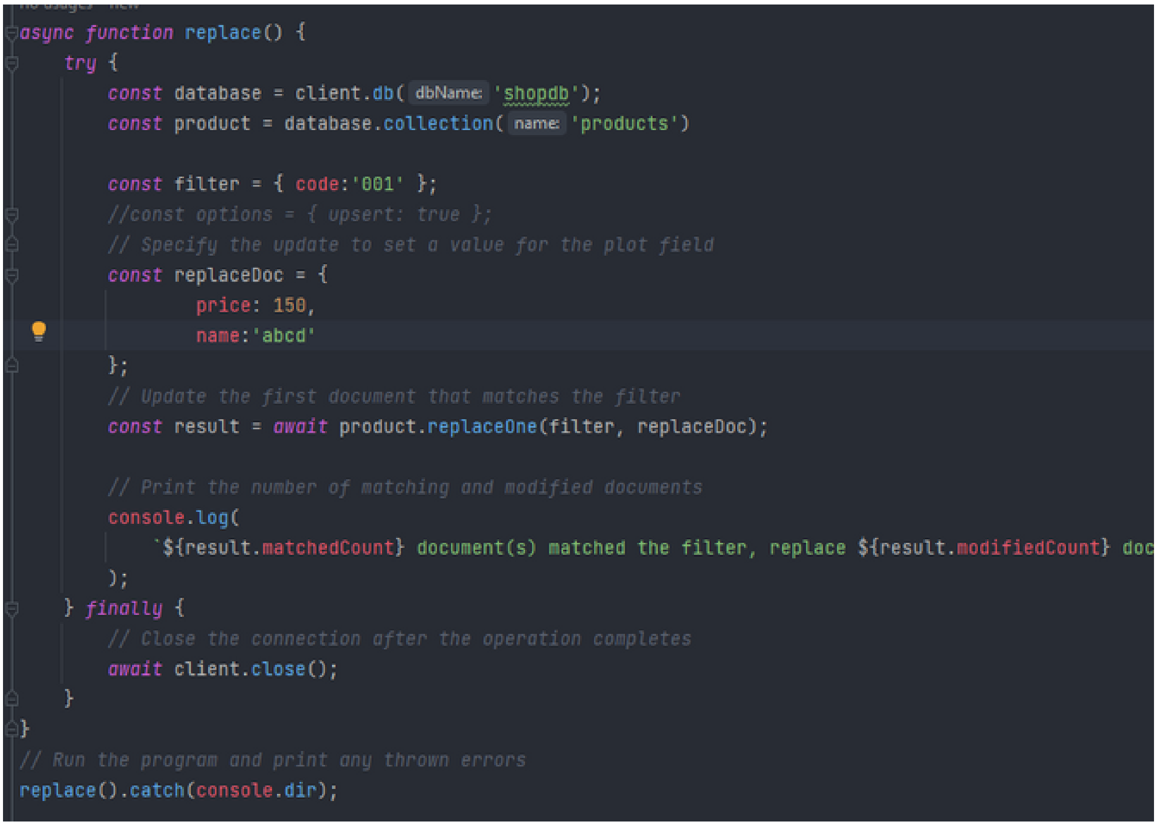


```
10 document(s) matched the filter, updated 9 document(s)
```

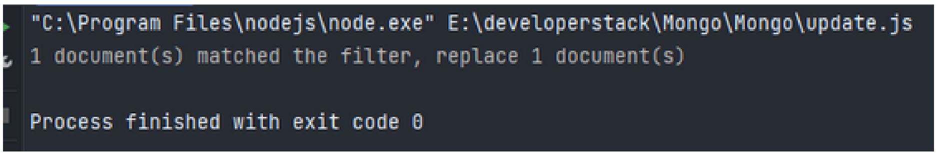


```
_id: ObjectId('6591586b890ac01aeead6d7b')
code: "001"
name: "abc"
price: 150
▸ rating: Array (3)

_id: ObjectId('6591586b890ac01aeead6d7c')
▸ code: Array (3)
name: "pqr"

_id: ObjectId('65915904ce2b95ea79eeaa2c')
code: "001"
name: "abc"
price: 150
▸ rating: Array (3)

_id: ObjectId('65915904ce2b95ea79eeaa2d')
▸ code: Array (3)
name: "pqr"

_id: ObjectId('6591594704a315cd8750a540')
```

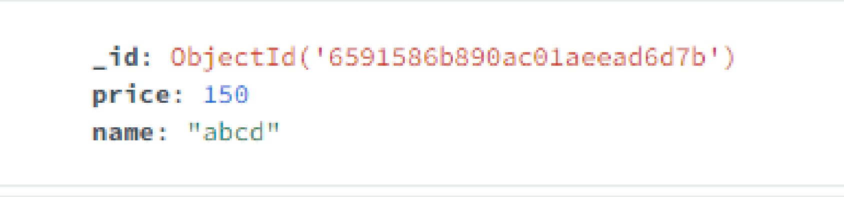
replace



```
no usages  new
async function replace() {
    try {
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const filter = { code:'001' };
        //const options = { upsert: true };
        // Specify the update to set a value for the plot field
        const replaceDoc = {
                price: 150,
                name:'abcd'
        };
        // Update the first document that matches the filter
        const result = await product.replaceOne(filter, replaceDoc);

        // Print the number of matching and modified documents
        console.log(
            `${result.matchedCount} document(s) matched the filter, replace ${result.modifiedCount} doc
        );
    } finally {
        // Close the connection after the operation completes
        await client.close();
    }
}
// Run the program and print any thrown errors
replace().catch(console.dir);
```



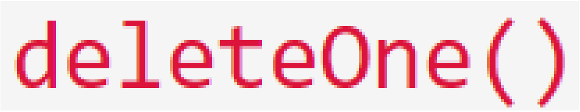
```
"C:\Program Files\nodejs\node.exe" E:\developerstack\Mongo\Mongo\update.js
1 document(s) matched the filter, replace 1 document(s)

Process finished with exit code 0
```



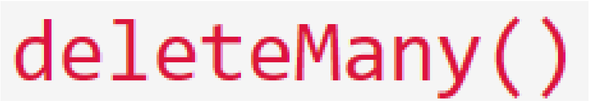
```
_id: ObjectId('6591586b890ac01aeead6d7b')
price: 150
name: "abcd"
```

Delete Documents

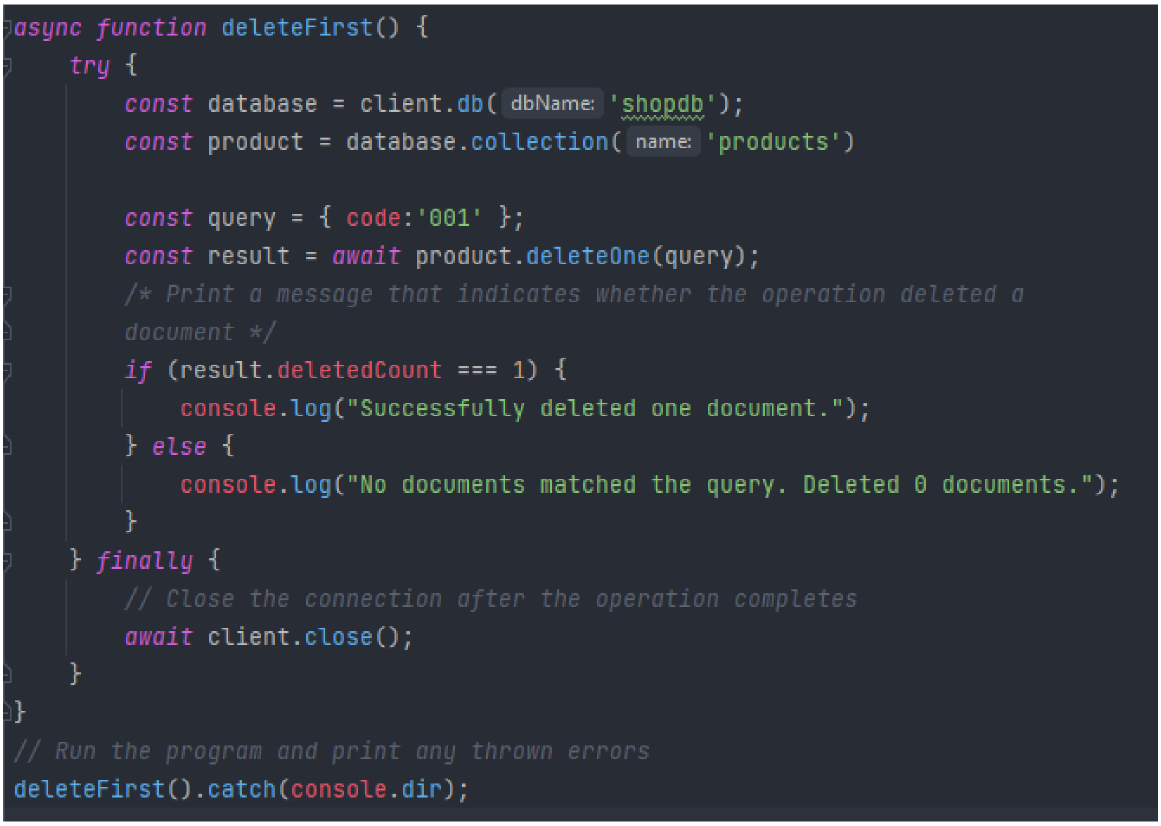
We can delete documents by using the methods deleteOne() or deleteMany().



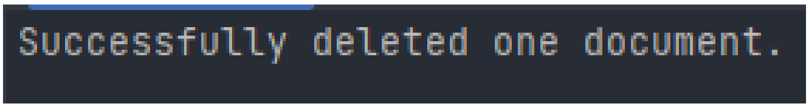
deleteOne()



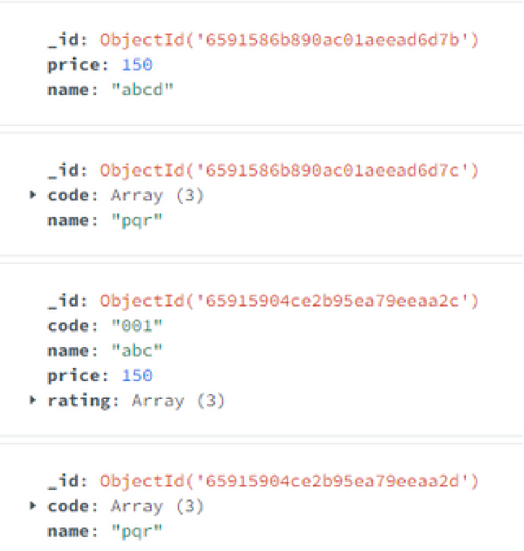
deleteMany()



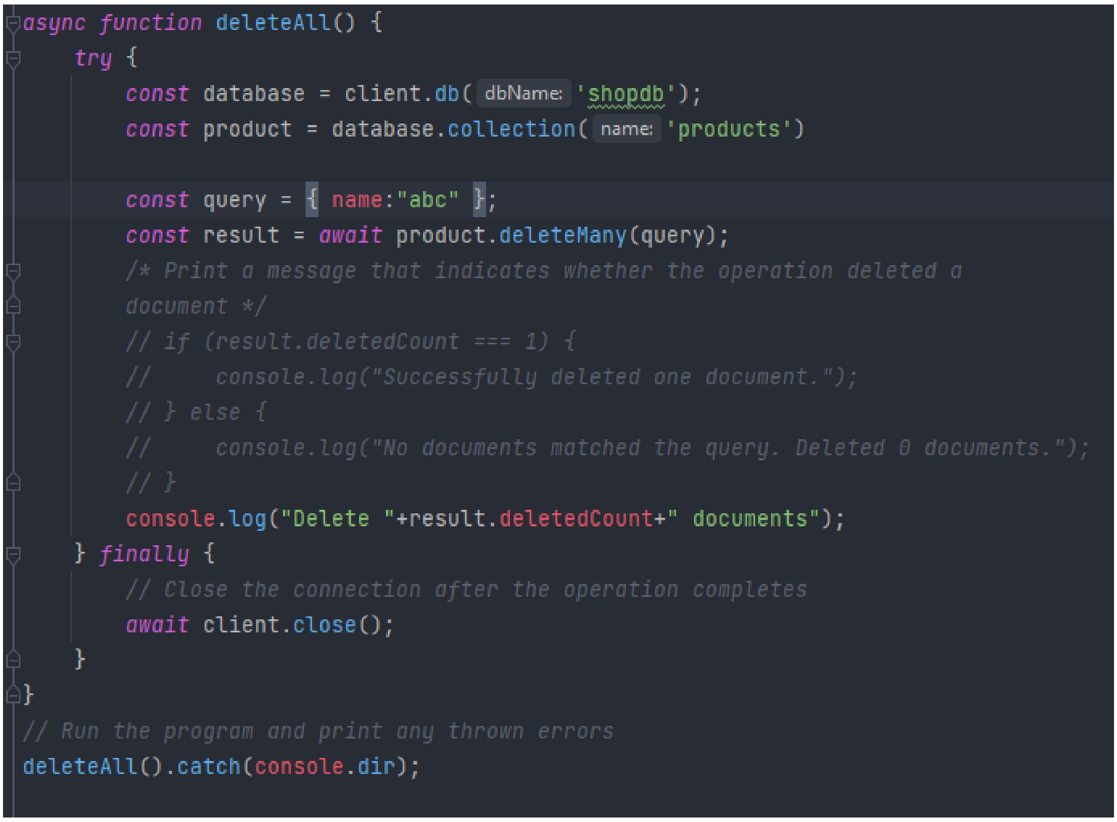
```
async function deleteFirst() {
    try {
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = { code:'001' };
        const result = await product.deleteOne(query);
        /* Print a message that indicates whether the operation deleted a
        document */
        if (result.deletedCount === 1) {
            console.log("Successfully deleted one document.");
        } else {
            console.log("No documents matched the query. Deleted 0 documents.");
        }
    } finally {
        // Close the connection after the operation completes
        await client.close();
    }
}
// Run the program and print any thrown errors
deleteFirst().catch(console.dir);
```



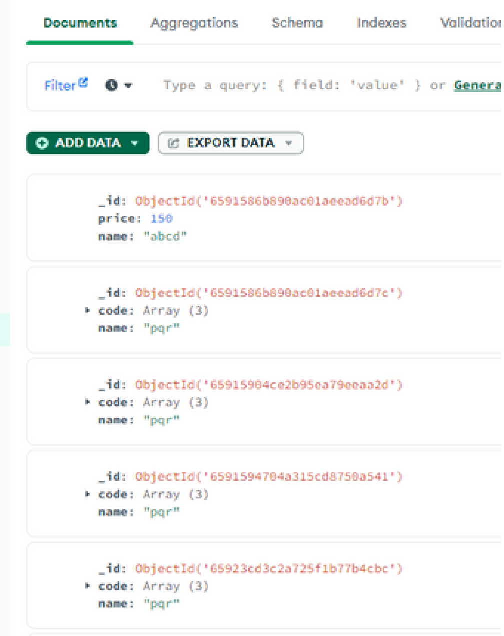
```
Successfully deleted one document.
```



```
  _id: ObjectId('6591586b890ac01aeead6d7b')
  price: 150
  name: "abcd"

  _id: ObjectId('6591586b890ac01aeead6d7c')
▸ code: Array (3)
  name: "pqr"

  _id: ObjectId('65915904ce2b95ea79eeaa2c')
  code: "001"
  name: "abc"
  price: 150
▸ rating: Array (3)

  _id: ObjectId('65915904ce2b95ea79eeaa2d')
▸ code: Array (3)
  name: "pqr"
```

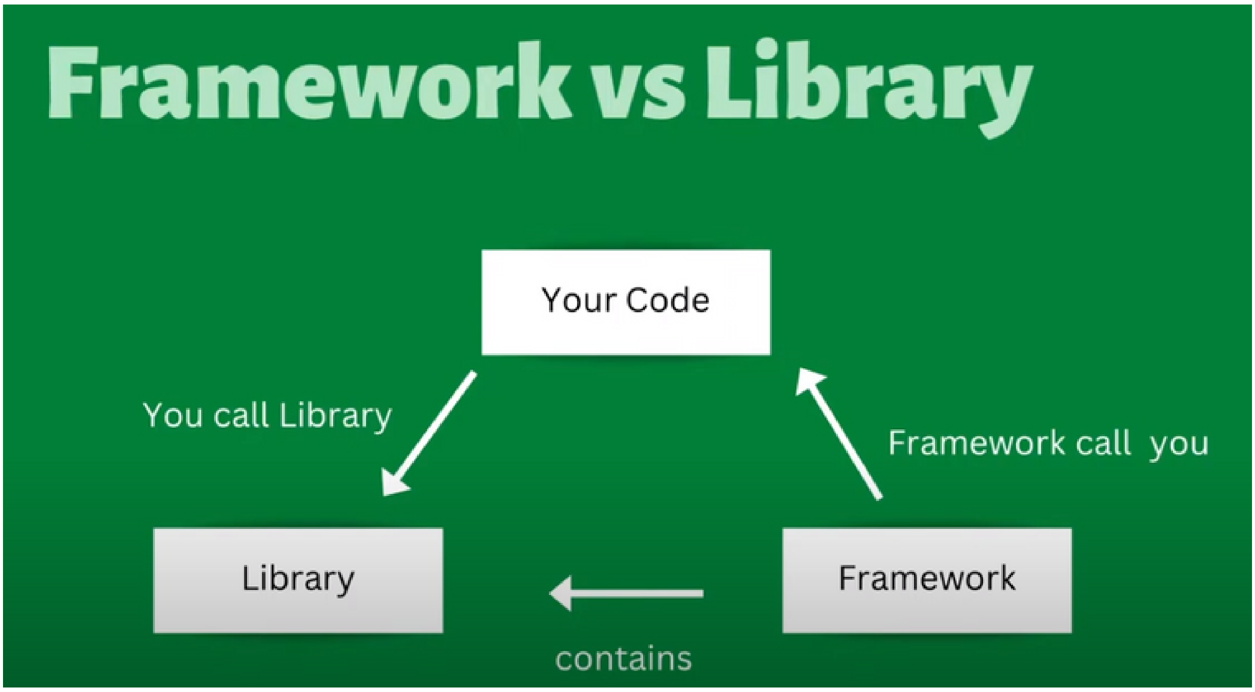


```
async function deleteAll() {
    try {
        const database = client.db( dbName: 'shopdb');
        const product = database.collection( name: 'products')

        const query = { name:"abc" };
        const result = await product.deleteMany(query);
        /* Print a message that indicates whether the operation deleted a
        document */
        // if (result.deletedCount === 1) {
        //     console.log("Successfully deleted one document.");
        // } else {
        //     console.log("No documents matched the query. Deleted 0 documents.");
        // }
        console.log("Delete "+result.deletedCount+" documents");
    } finally {
        // Close the connection after the operation completes
        await client.close();
    }
}
// Run the program and print any thrown errors
deleteAll().catch(console.dir);
```

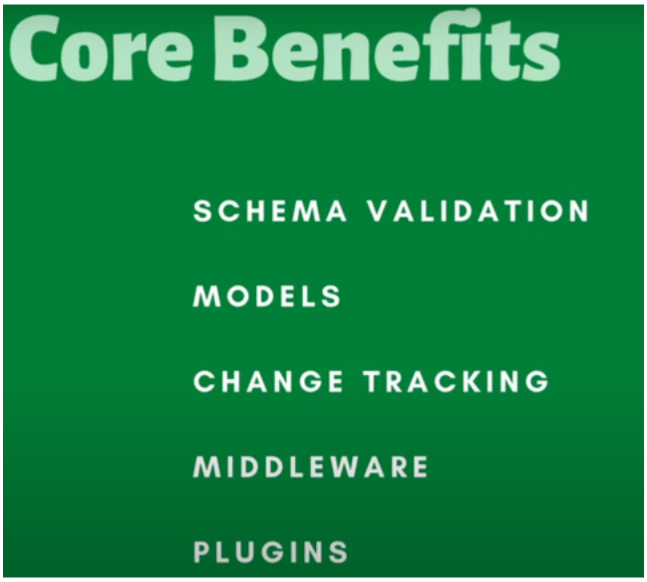


```
_id: ObjectId('6591586b890ac01aeead6d7b')
price: 150
name: "abcd"

_id: ObjectId('6591586b890ac01aeead6d7c')
code: Array (3)
name: "pqr"

_id: ObjectId('65915904ce2b95ea79eeaa2d')
code: Array (3)
name: "pqr"

_id: ObjectId('6591594704a315cd8750a541')
code: Array (3)
name: "pqr"

_id: ObjectId('65923cd3c2a725f1b77b4cbc')
code: Array (3)
name: "pqr"
```

Mongoose is an object-document mapping (ODM) framework for Node.js and MongoDB

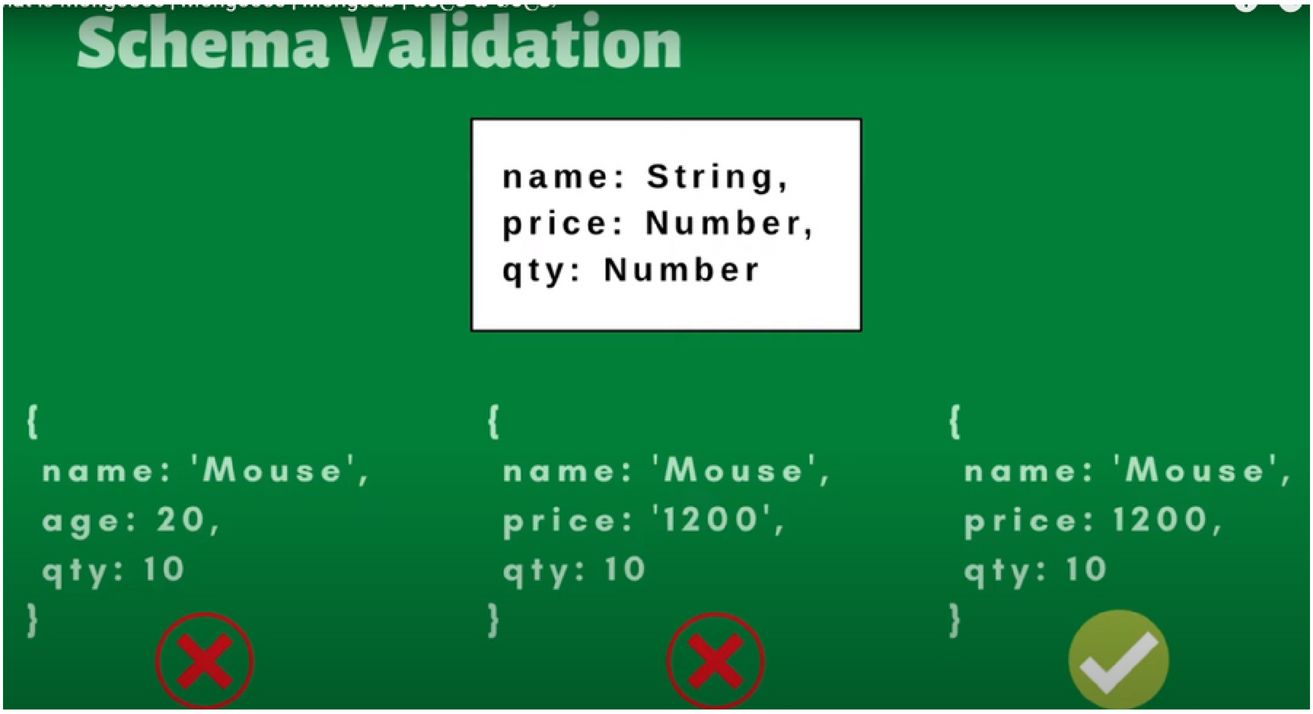


Framework vs Library



Core Benefits

- SCHEMA VALIDATION
- MODELS
- CHANGE TRACKING
- MIDDLEWARE
- PLUGINS



Schema Validation

```
name: String,
price: Number,
qty: Number
```

```
{
 name: 'Mouse',
 age: 20,
 qty: 10
}
```

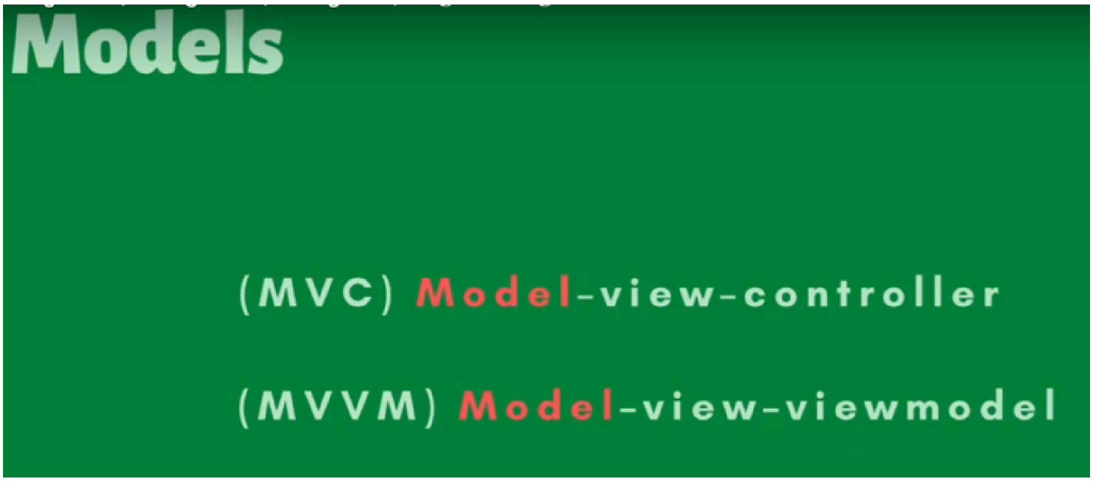
✗

```
{
 name: 'Mouse',
 price: '1200',
 qty: 10
}
```

✗

```
{
 name: 'Mouse',
 price: 1200,
 qty: 10
}
```

✓



Models

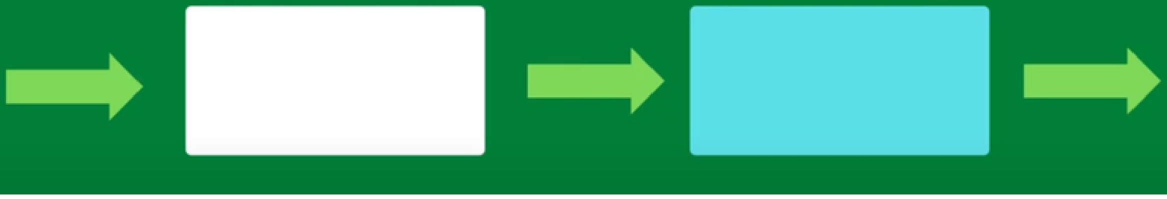
(MVC) Model-view-controller

(MVVM) Model-view-viewmodel

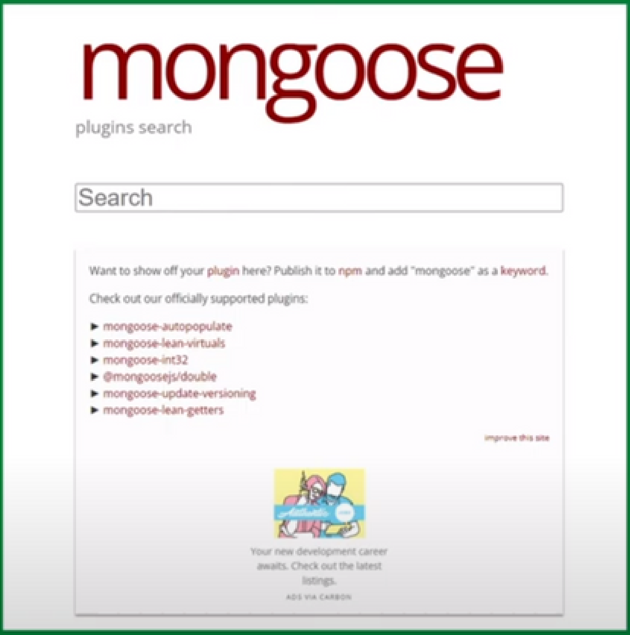
Change tracking

```
doc.save();
```

Middleware



Plugins



mongoose

plugins search

Search

Want to show off your plugin here? Publish it to npm and add "mongoose" as a keyword.

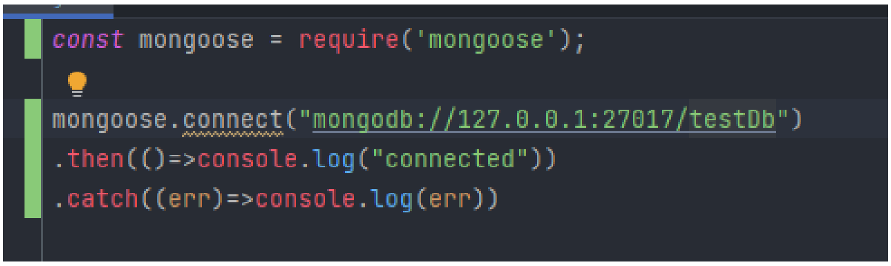
Check out our officially supported plugins:

- mongoose-autopopulate
- mongoose-lean-virtuals
- mongoose-int32
- @mongoosejs/double
- mongoose-update-versioning
- mongoose-lean-getters

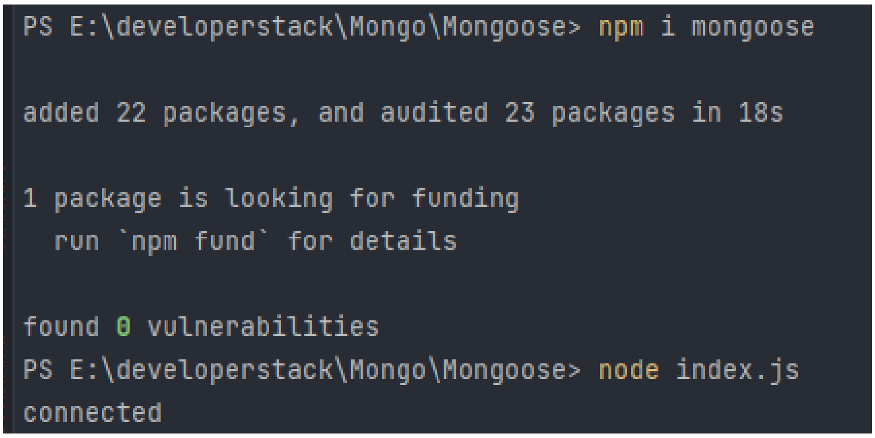
improve this site

https://plugins.mongoosejs.io/

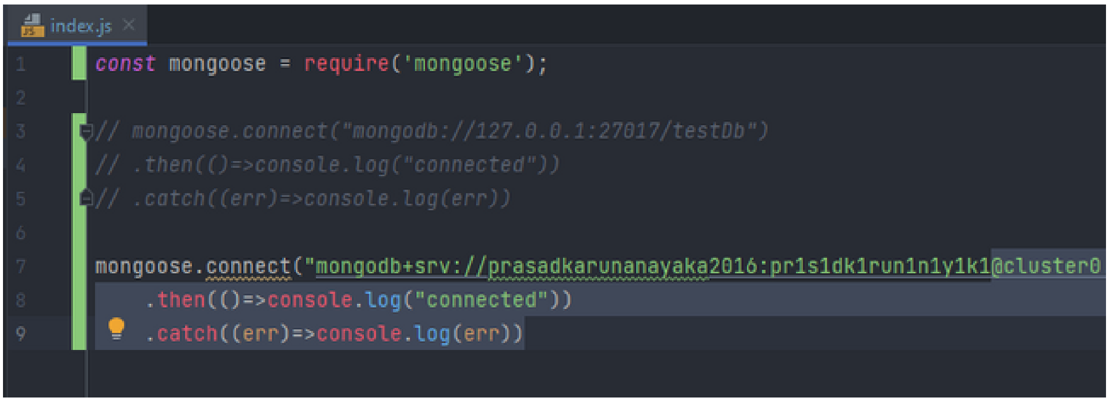
Connect atlas cloud



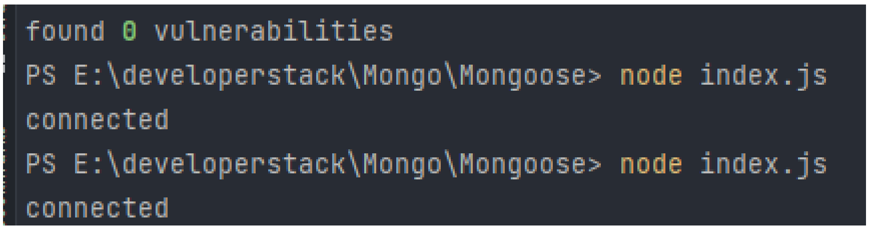
```javascript
const mongoose = require('mongoose');

mongoose.connect("mongodb://127.0.0.1:27017/testDb")
.then(()=>console.log("connected"))
.catch((err)=>console.log(err))
```



```
PS E:\developerstack\Mongo\Mongoose> npm i mongoose

added 22 packages, and audited 23 packages in 18s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS E:\developerstack\Mongo\Mongoose> node index.js
connected
```

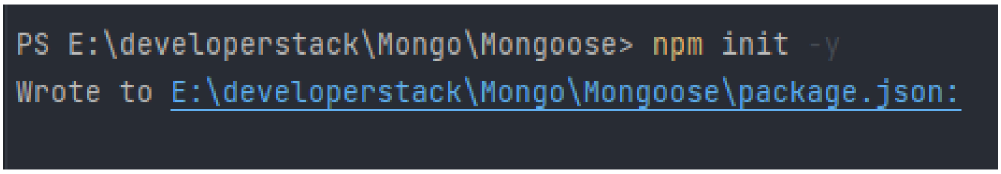


```javascript
const mongoose = require('mongoose');

// mongoose.connect("mongodb://127.0.0.1:27017/testDb")
// .then(()=>console.log("connected"))
// .catch((err)=>console.log(err))

mongoose.connect("mongodb+srv://prasadkarunanayaka2016:pr1s1dk1run1n1y1k1@cluster0
    .then(()=>console.log("connected"))
    .catch((err)=>console.log(err))
```

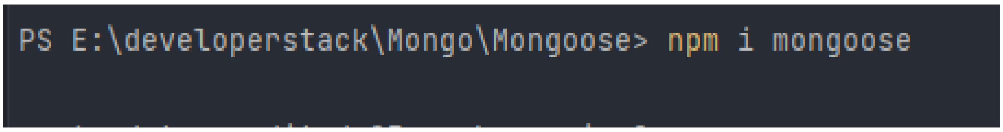


```
found 0 vulnerabilities
PS E:\developerstack\Mongo\Mongoose> node index.js
connected
PS E:\developerstack\Mongo\Mongoose> node index.js
connected
```

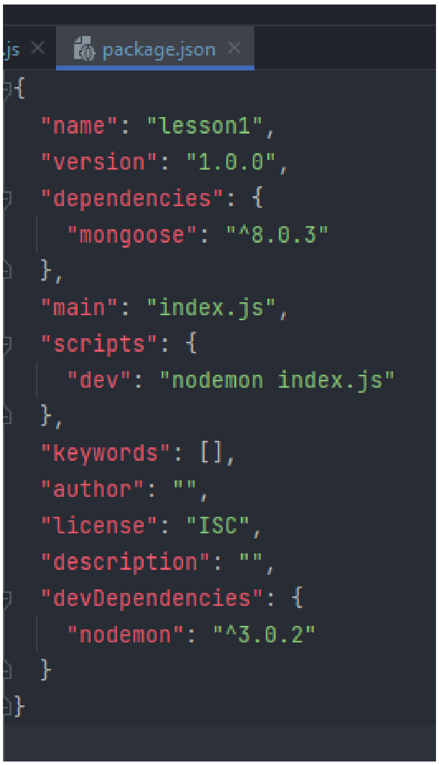
Mongoose Schema and model



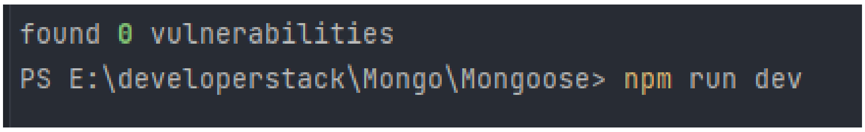
```
PS E:\developerstack\Mongo\Mongoose> npm init -y
Wrote to E:\developerstack\Mongo\Mongoose\package.json:
```



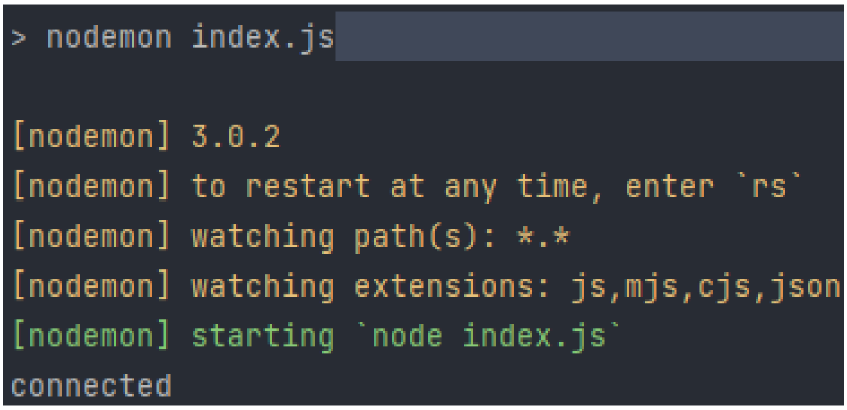
```
PS E:\developerstack\Mongo\Mongoose> npm i mongoose
```



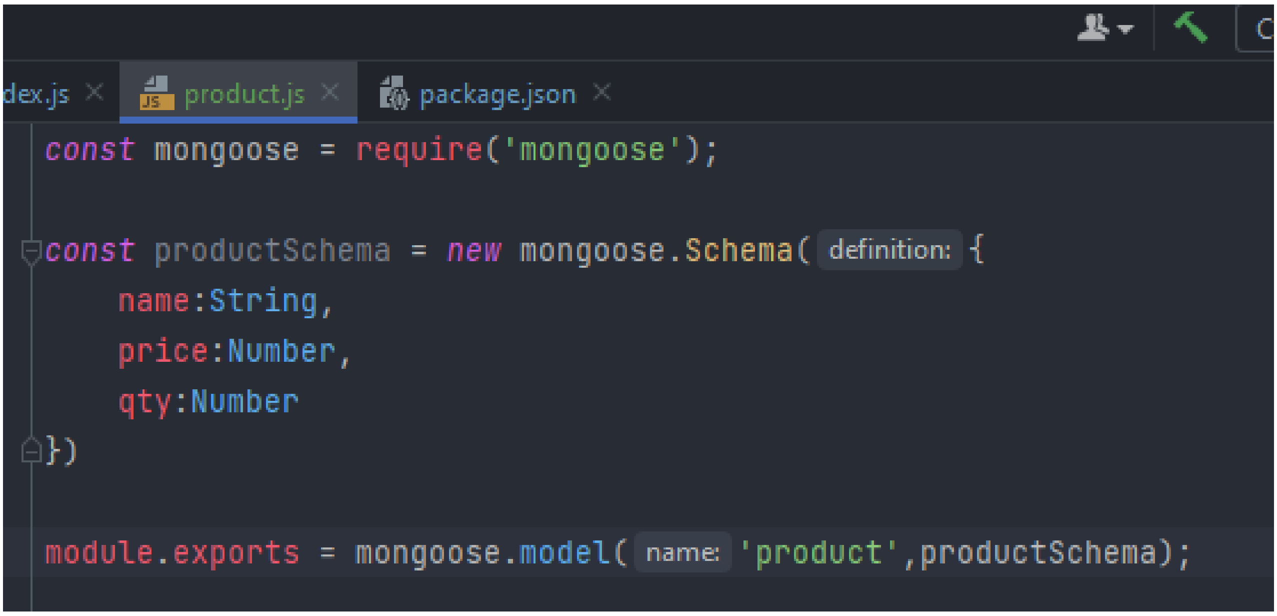
```
{
  "name": "lesson1",
  "version": "1.0.0",
  "dependencies": {
    "mongoose": "^8.0.3"
  },
  "main": "index.js",
  "scripts": {
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "nodemon": "^3.0.2"
  }
}
```



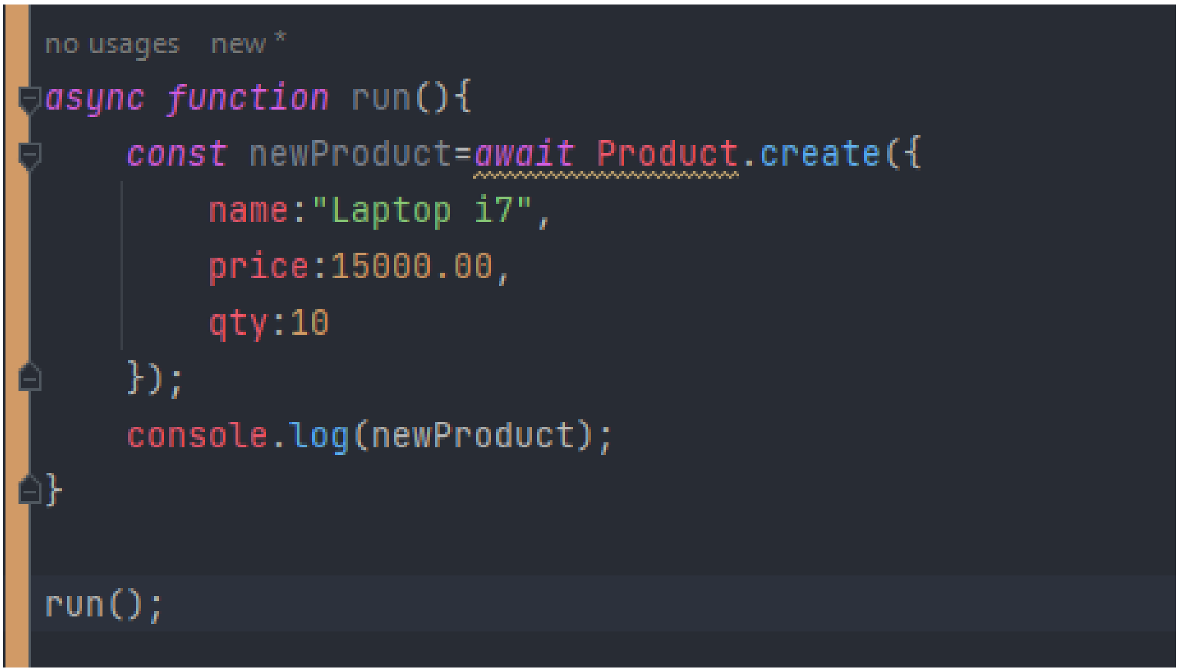
```
found 0 vulnerabilities
PS E:\developerstack\Mongo\Mongoose> npm run dev
```



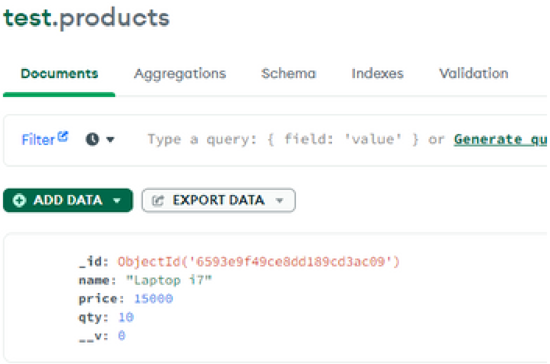
```
> nodemon index.js

[nodemon] 3.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
connected
```



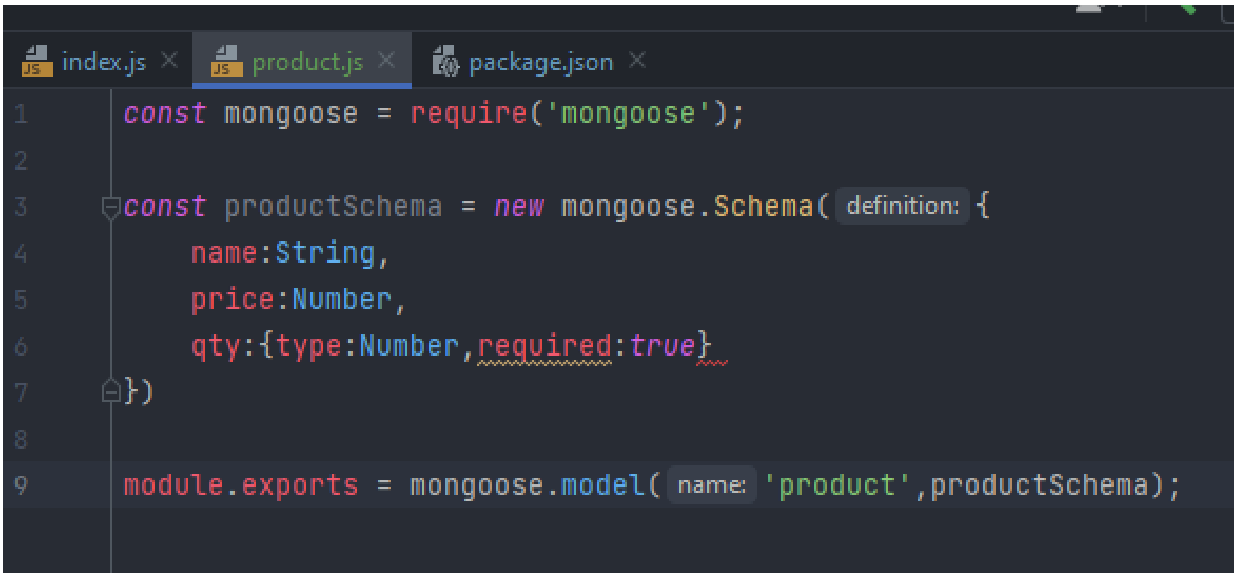
```
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema( definition: {
    name:String,
    price:Number,
    qty:Number
})

module.exports = mongoose.model( name: 'product',productSchema);
```



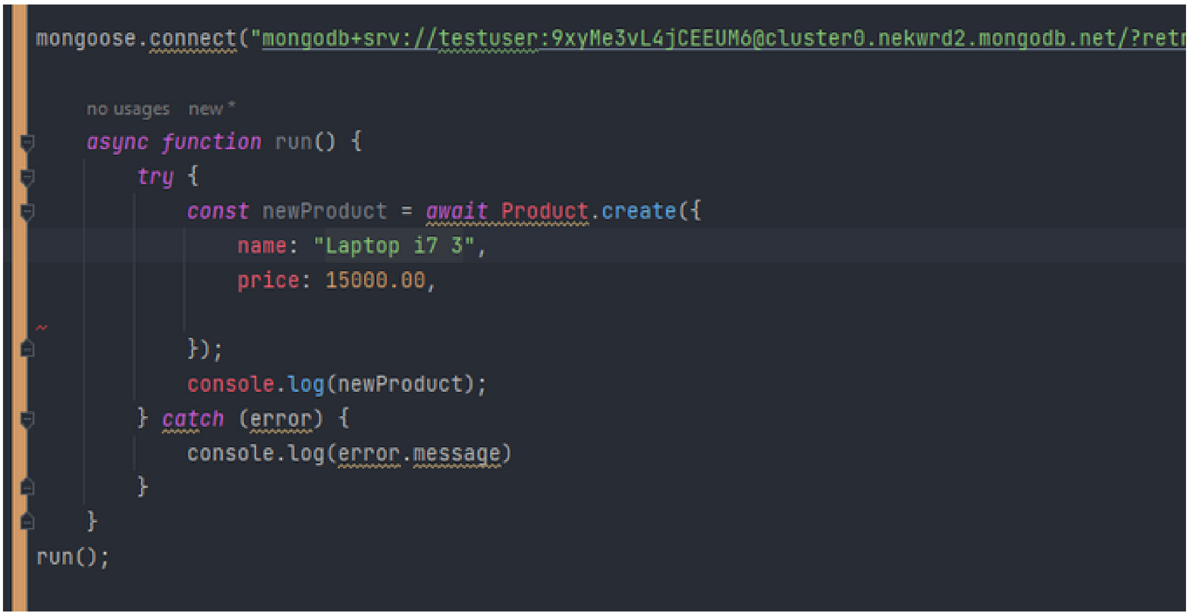
```
no usages   new *
async function run(){
    const newProduct=await Product.create({
        name:"Laptop i7",
        price:15000.00,
        qty:10
    });
    console.log(newProduct);
}

run();
```



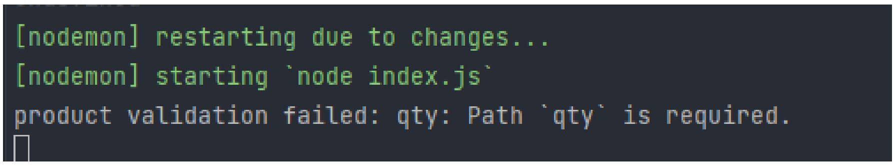
```
_id: ObjectId('6593e9f49ce8dd189cd3ac09')
name: "Laptop i7"
price: 15000
qty: 10
__v: 0
```



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

const productSchema = new mongoose.Schema( definition: {
    name:String,
    price:Number,
    qty:{type:Number,required:true}
})

module.exports = mongoose.model( name: 'product',productSchema);
```

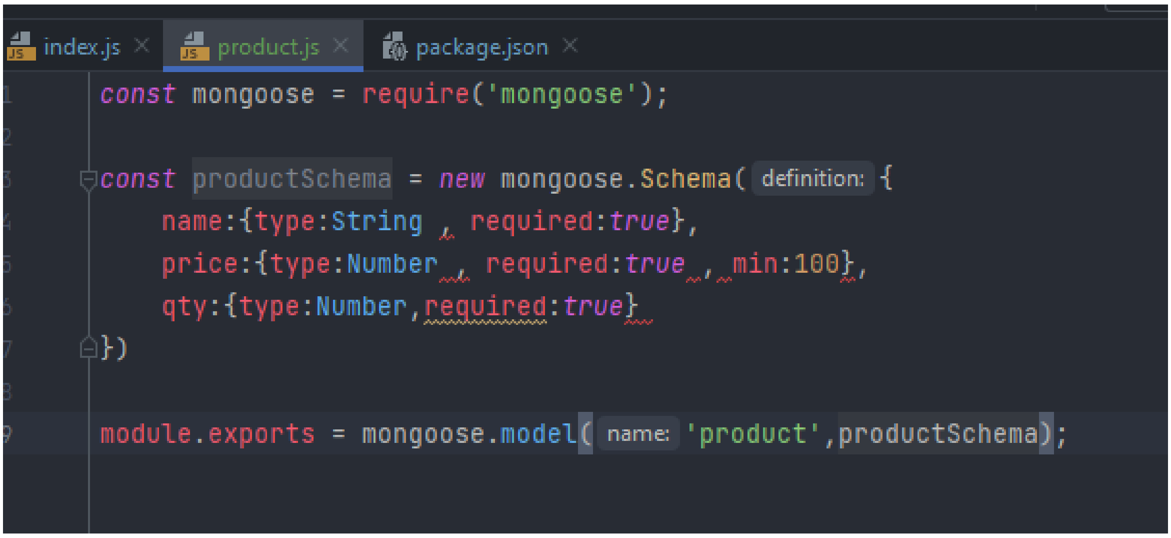


```
mongoose.connect("mongodb+srv://testuser:9xyMe3vL4jCEEUM6@cluster0.nekwrd2.mongodb.net/?retr

    no usages   new *
    async function run() {
        try {
            const newProduct = await Product.create({
                name: "Laptop i7 3",
                price: 15000.00,

            });
            console.log(newProduct);
        } catch (error) {
            console.log(error.message)
        }
    }
run();
```

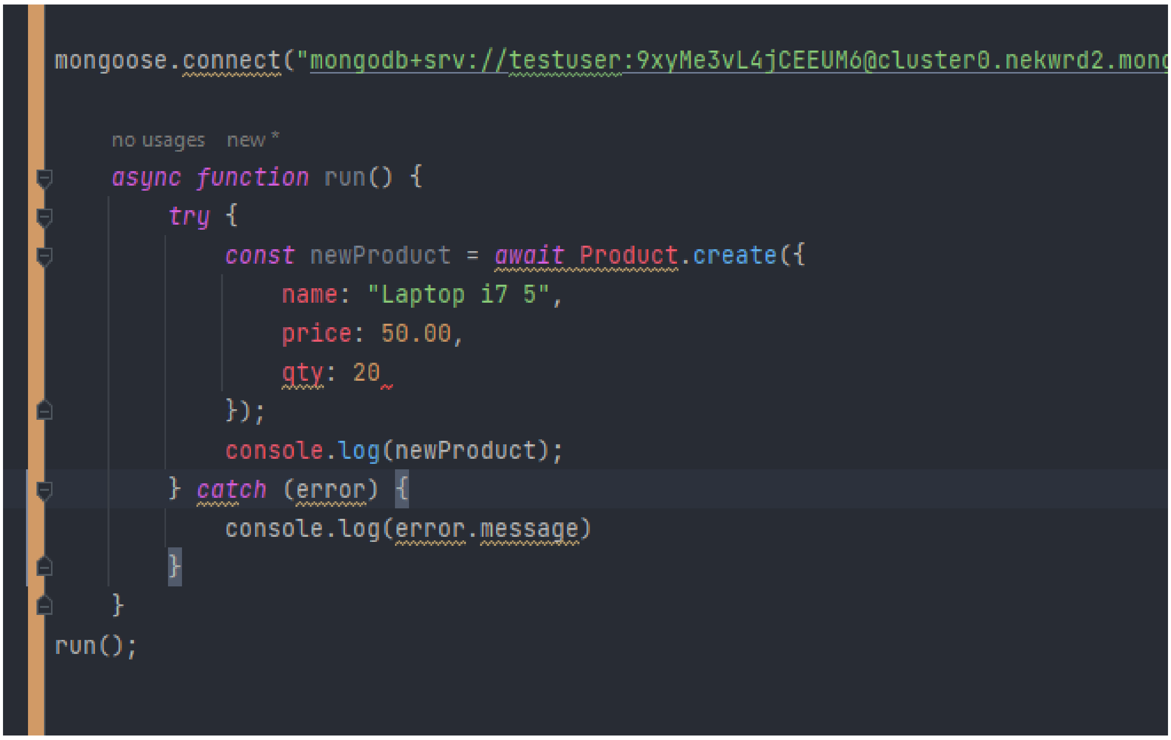


```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
product validation failed: qty: Path `qty` is required.
```

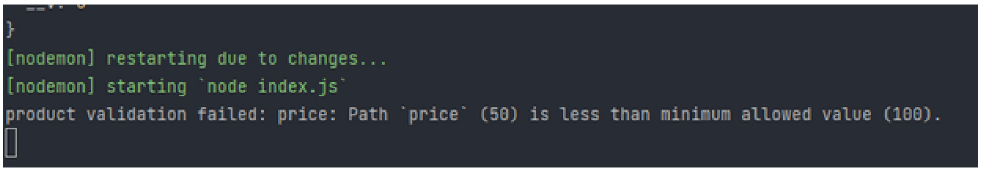
Document



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

const productSchema = new mongoose.Schema( definition: {
    name:{type:String , required:true},
    price:{type:Number , required:true , min:100},
    qty:{type:Number,required:true}
})

module.exports = mongoose.model( name: 'product',productSchema);
```



```
mongoose.connect("mongodb+srv://testuser:9xyMe3vL4jCEEUM6@cluster0.nekwrd2.mong

    no usages   new *
    async function run() {
        try {
            const newProduct = await Product.create({
                name: "Laptop i7 5",
                price: 50.00,
                qty: 20
            });
            console.log(newProduct);
        } catch (error) {
            console.log(error.message)
        }
    }
run();
```



```
}
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
product validation failed: price: Path `price` (50) is less than minimum allowed value (100).
```

String

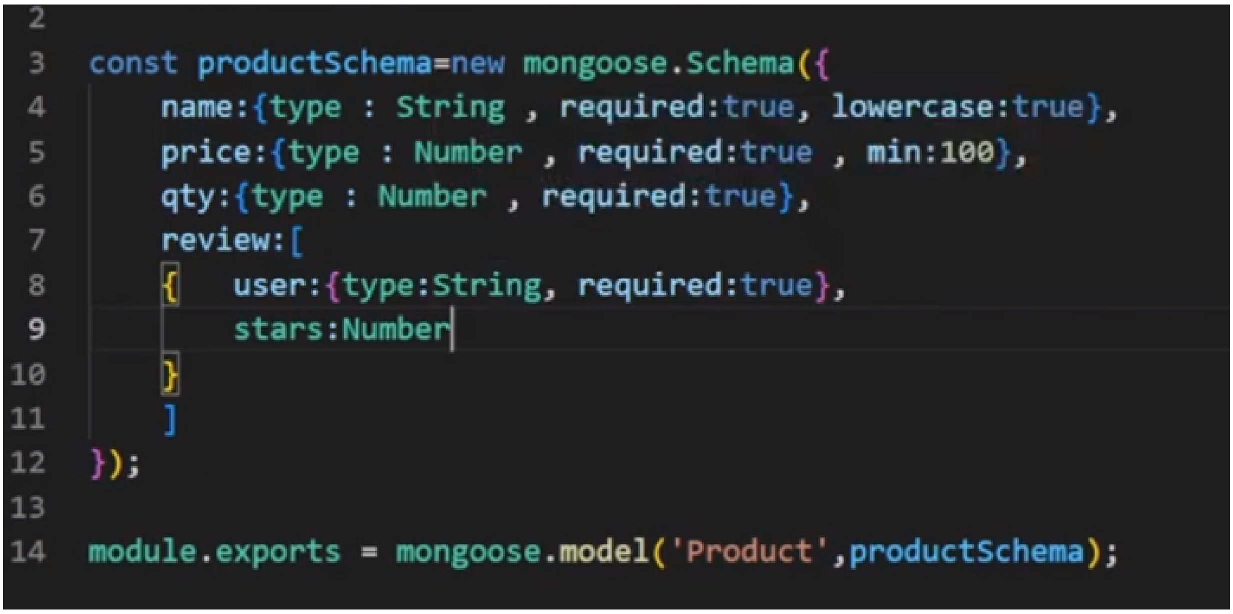
- `lowercase` : boolean, wh
- `uppercase` : boolean, wh
- `trim` : boolean, whether
- `match` : RegExp, creates a
- `enum` : Array, creates a va
- `minLength` : Number, cre
- `maxLength` : Number, cre
- `populate` : Object, sets d

Number

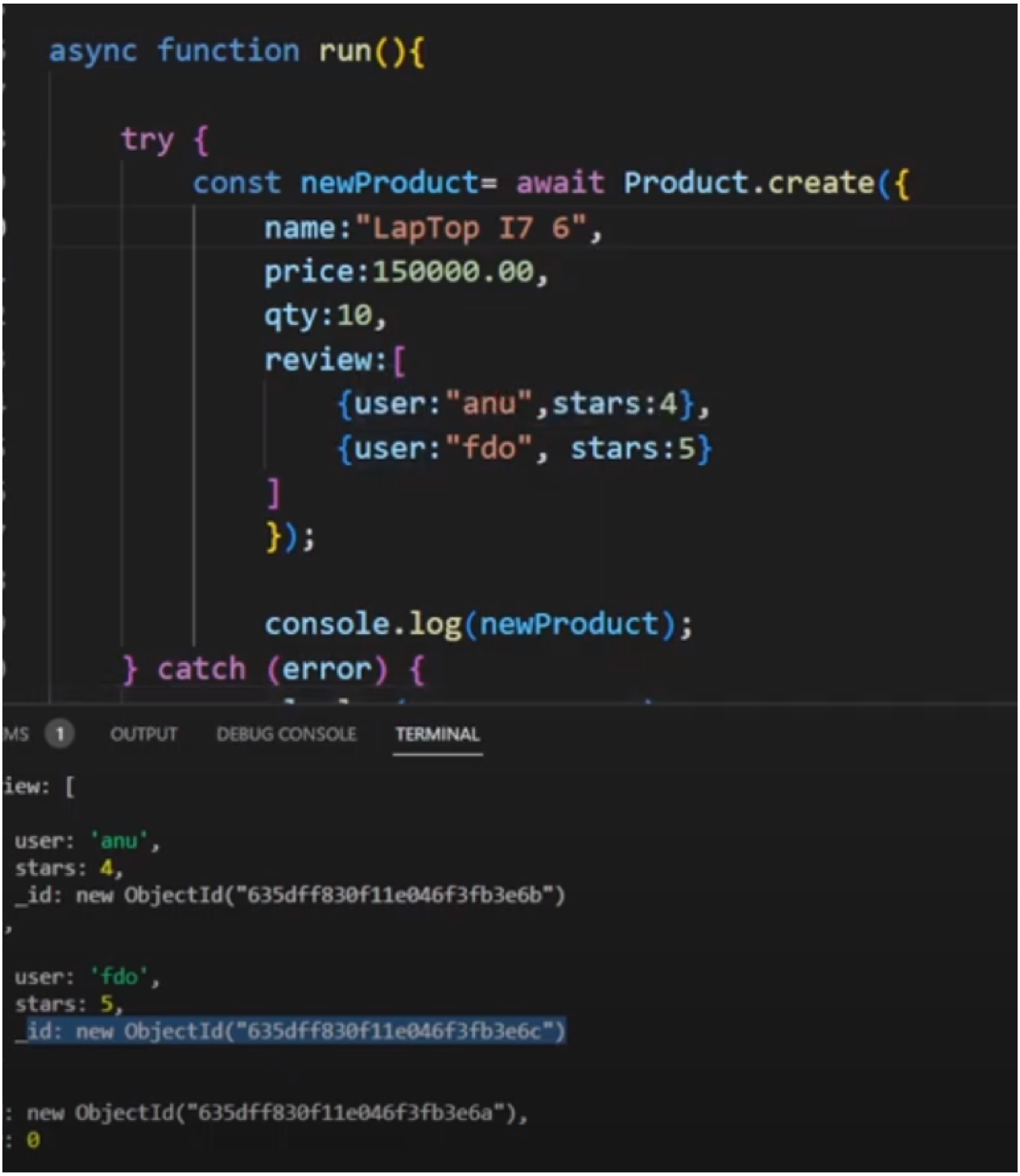
- `min` : Number, creates a
- `max` : Number, creates a
- `enum` : Array, creates a va
  array.
- `populate` : Object, sets d

Date

- `min` : Date, creates a vali
- `max` : Date, creates a vali
- `expires` : Number or Stri



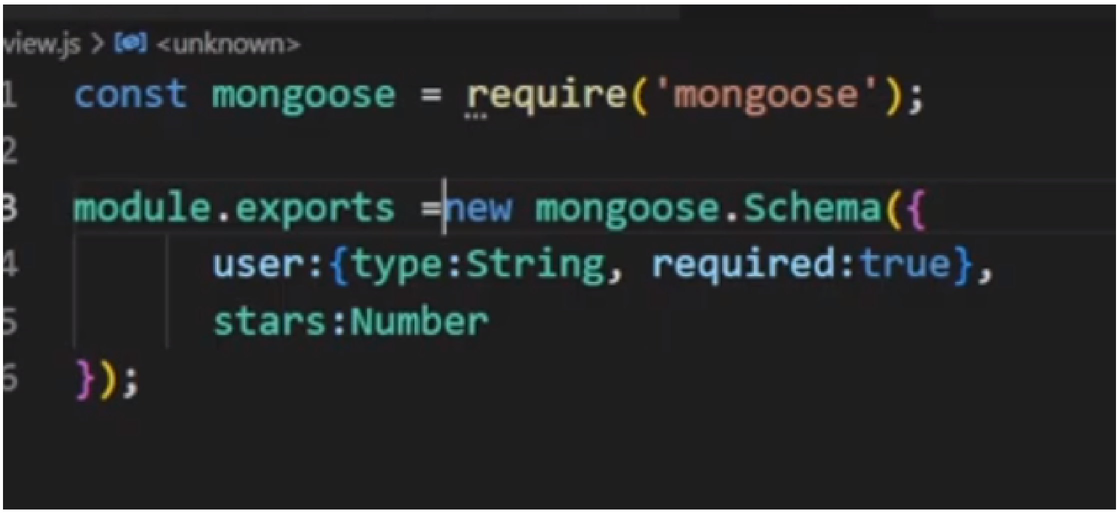
```
const productSchema=new mongoose.Schema({
    name:{type : String , required:true, lowercase:true},
    price:{type : Number , required:true , min:100},
    qty:{type : Number , required:true},
    review:[
    {   user:{type:String, required:true},
        stars:Number
    }
    ]
});

module.exports = mongoose.model('Product',productSchema);
```



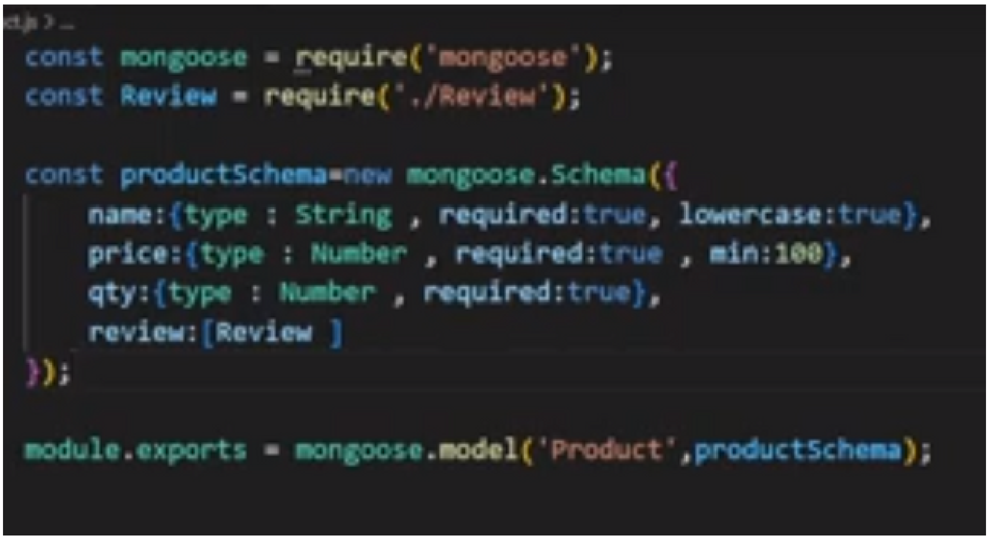
```
async function run(){

    try {
        const newProduct= await Product.create({
            name:"LapTop I7 6",
            price:150000.00,
            qty:10,
            review:[
                {user:"anu",stars:4},
                {user:"fdo", stars:5}
            ]
            });

            console.log(newProduct);
    } catch (error) {
```

```
iew: [

 user: 'anu',
 stars: 4,
 _id: new ObjectId("635dff830f11e046f3fb3e6b")
,

 user: 'fdo',
 stars: 5,
 _id: new ObjectId("635dff830f11e046f3fb3e6c")


: new ObjectId("635dff830f11e046f3fb3e6a"),
: 0
```

schema connect



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

module.exports =new mongoose.Schema({
        user:{type:String, required:true},
        stars:Number
});
```



```
const mongoose = require('mongoose');
const Review = require('./Review');

const productSchema=new mongoose.Schema({
    name:{type : String , required:true, lowercase:true},
    price:{type : Number , required:true , min:100},
    qty:{type : Number , required:true},
    review:[Review ]
});

module.exports = mongoose.model('Product',productSchema);
```