TP-09

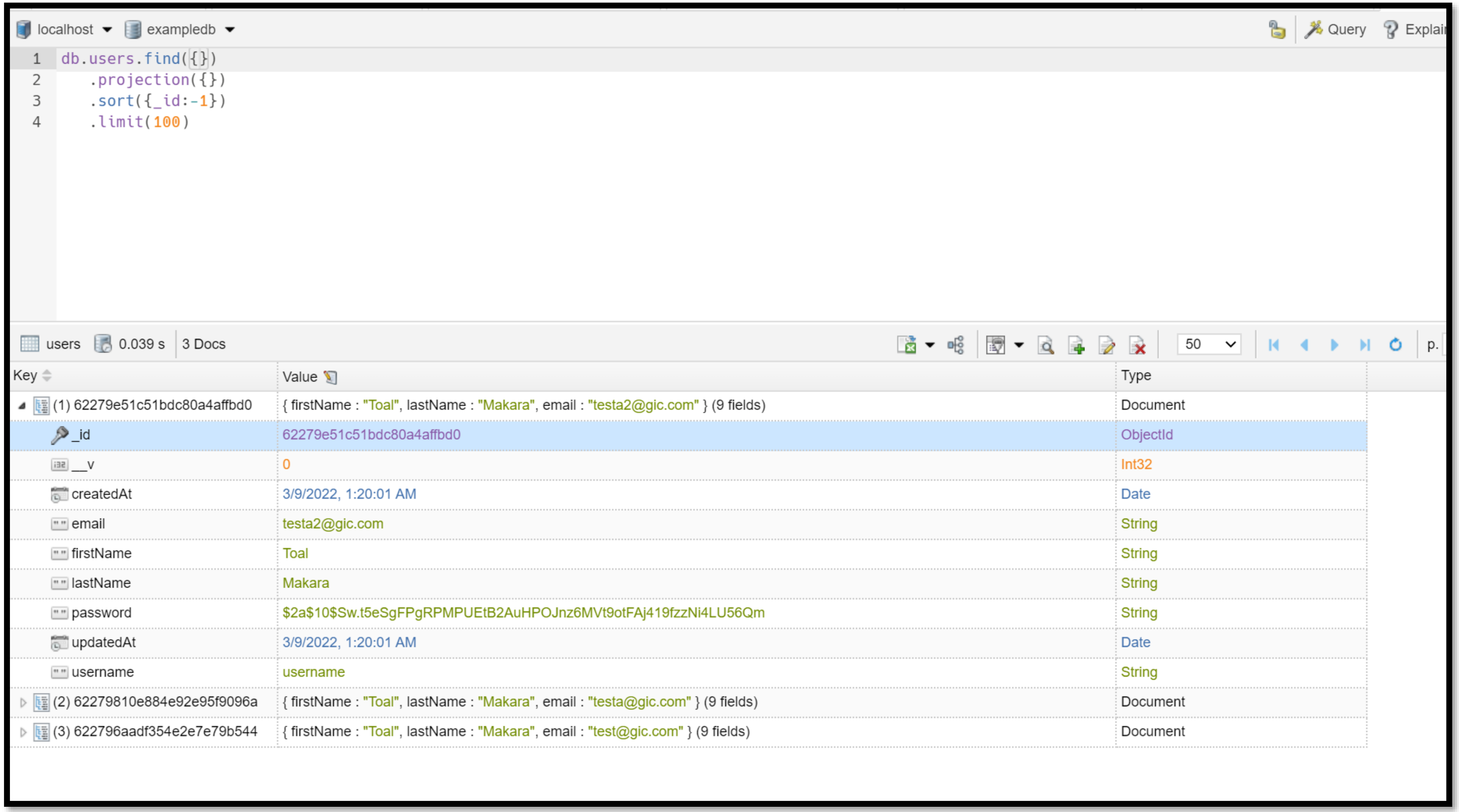
VueJS, NodeJS

Authentication (conti.)

TP09 Exercise

TP09.1: Mongoose

For the previous authentication APIs, replace your user.json storage by a NoSql database, mongoose



localhost ▾ exampledb ▾ Query Explai

```
db.users.find({})
   .projection({})
   .sort({_id:-1})
   .limit(100)
```

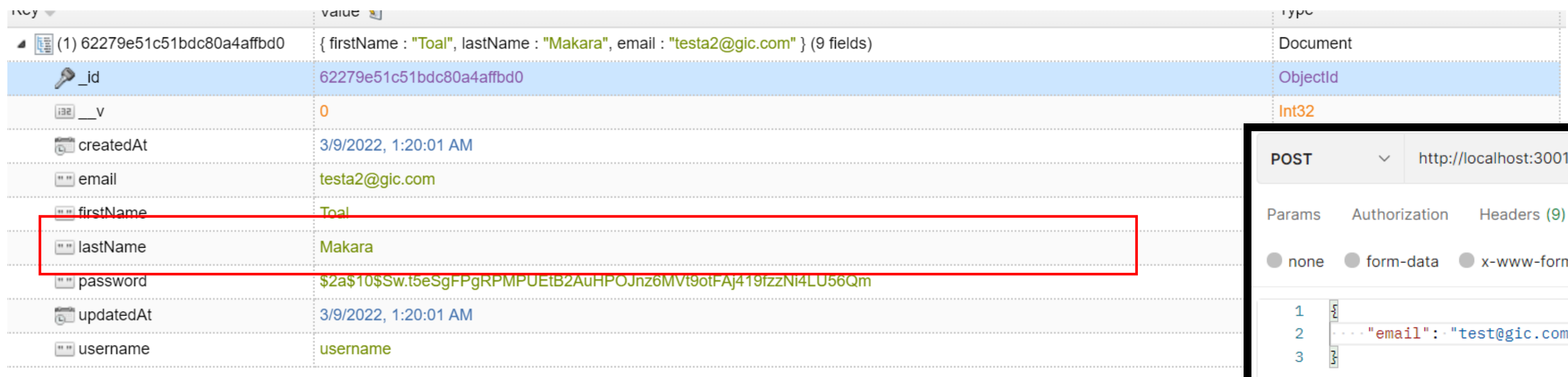
users 0.039 s 3 Docs 50

Key	Value	Type
(1) 62279e51c51bdc80a4affbd0	{ firstName : "Toal", lastName : "Makara", email : "testa2@gic.com" } (9 fields)	Document
_id	62279e51c51bdc80a4affbd0	ObjectId
__v	0	Int32
createdAt	3/9/2022, 1:20:01 AM	Date
email	testa2@gic.com	String
firstName	Toal	String
lastName	Makara	String
password	$2a$10$Sw.t5eSgFPgRPMPUEtB2AuHPOJnz6MVt9otFAj419fzzNi4LU56Qm	String
updatedAt	3/9/2022, 1:20:01 AM	Date
username	username	String
(2) 62279810e884e92e95f9096a	{ firstName : "Toal", lastName : "Makara", email : "testa@gic.com" } (9 fields)	Document
(3) 622796aadf354e2e7e79b544	{ firstName : "Toal", lastName : "Makara", email : "test@gic.com" } (9 fields)	Document

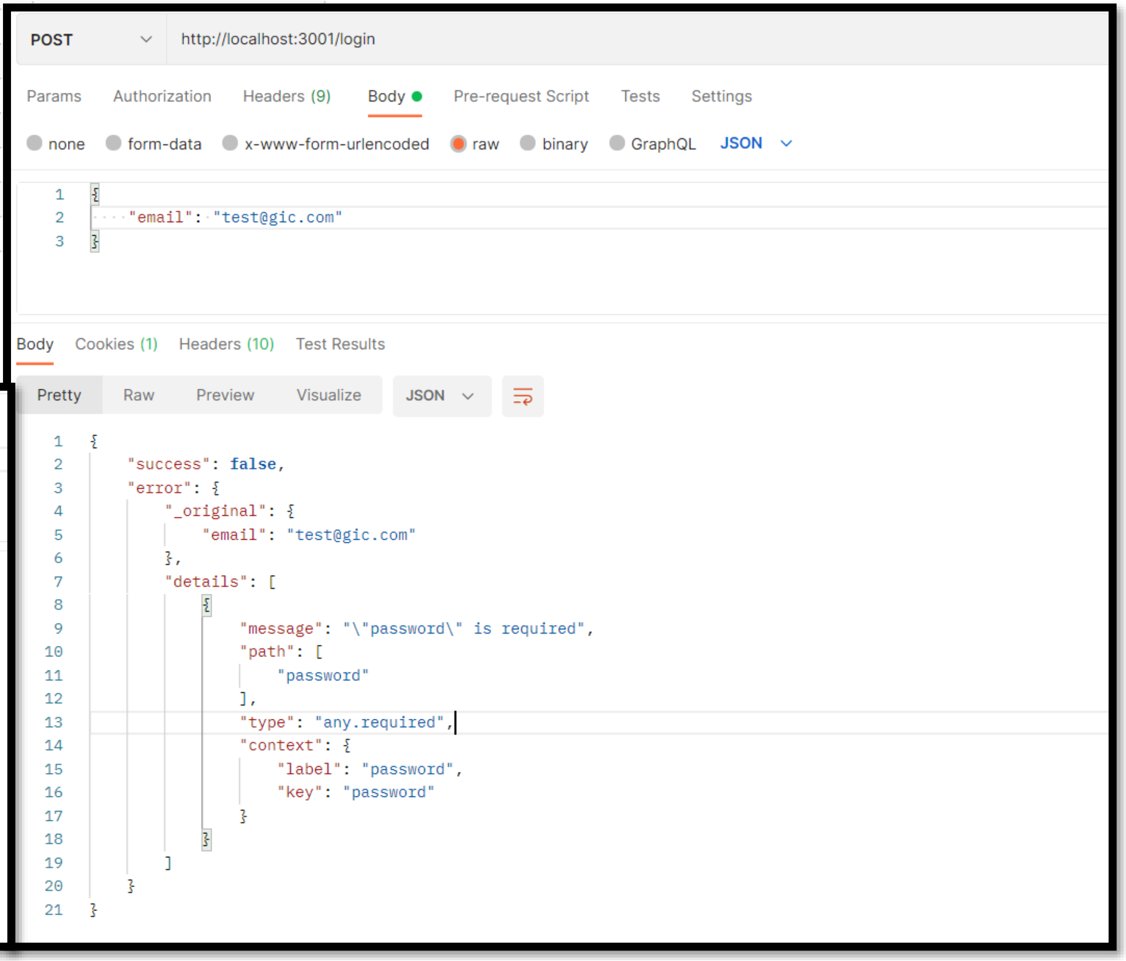
TP09 Exercise

TP09.2: Authentication with database

Implement the authentication with the password encrypted and request validation middleware.



Key	Value	Type
(1) 62279e51c51bdc80a4affbd0	{ firstName : "Toal", lastName : "Makara", email : "testa2@gic.com" } (9 fields)	Document
_id	62279e51c51bdc80a4affbd0	ObjectId
__v	0	Int32
createdAt	3/9/2022, 1:20:01 AM	
email	testa2@gic.com	
firstName	Toal	
lastName	Makara	
password	$2a$10$Sw.t5eSgFPgRPMPUEtB2AuHPOJnz6MVt9otFAj419fzzNi4LU56Qm	
updatedAt	3/9/2022, 1:20:01 AM	
username	username	



POST http://localhost:3001/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings

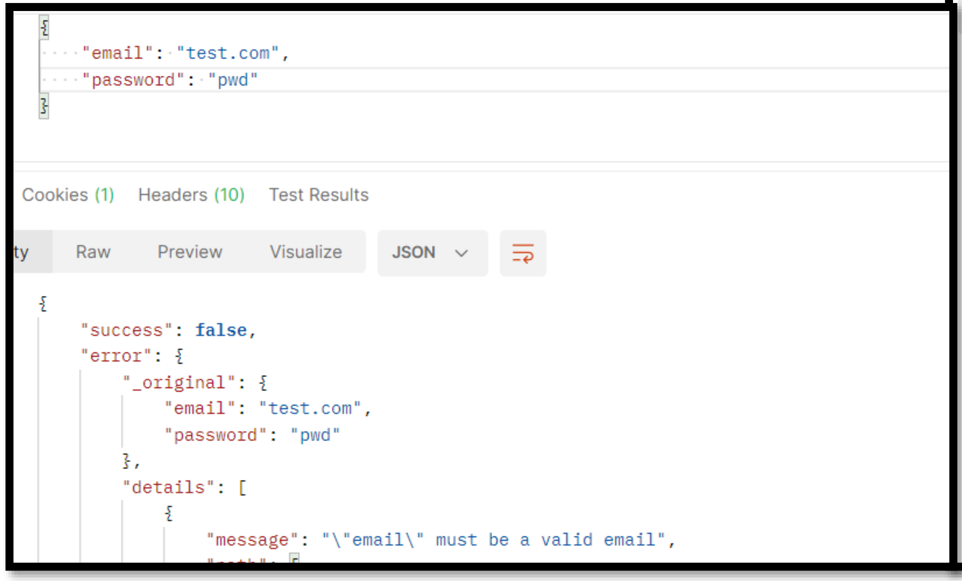
none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "email": "test@gic.com"
}
```

Body Cookies (1) Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "success": false,
    "error": {
        "_original": {
            "email": "test@gic.com"
        },
        "details": [
            {
                "message": "\"password\" is required",
                "path": [
                    "password"
                ],
                "type": "any.required",
                "context": {
                    "label": "password",
                    "key": "password"
                }
            }
        ]
    }
}
```



```
{
    "email": "test.com",
    "password": "pwd"
}
```

Cookies (1) Headers (10) Test Results

ty Raw Preview Visualize JSON

```
{
    "success": false,
    "error": {
        "_original": {
            "email": "test.com",
            "password": "pwd"
        },
        "details": [
            {
                "message": "\"email\" must be a valid email",
```

TP09 Exercise

EX3: Token, Cookie

Create a token of an authenticated user and store it as cookie for ensuring the authorized user. The token validation method must be a middleware function. Ex. Using express-session

If you're already signed in:

- /login (You cannot sign in again)
- /register (You cannot register a user)

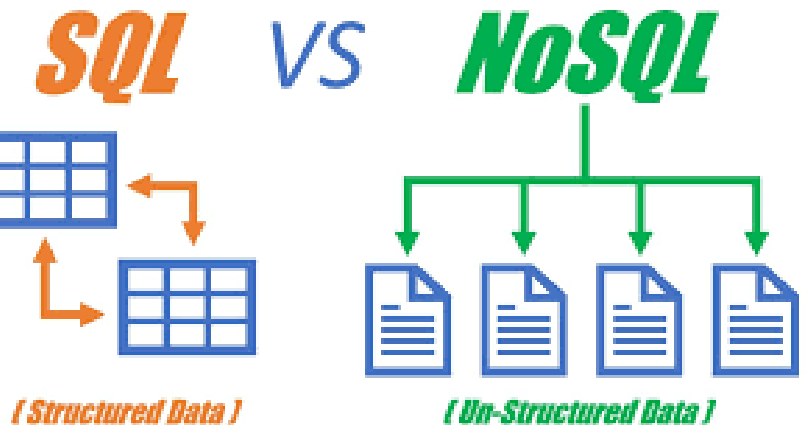
If you're not signed in:

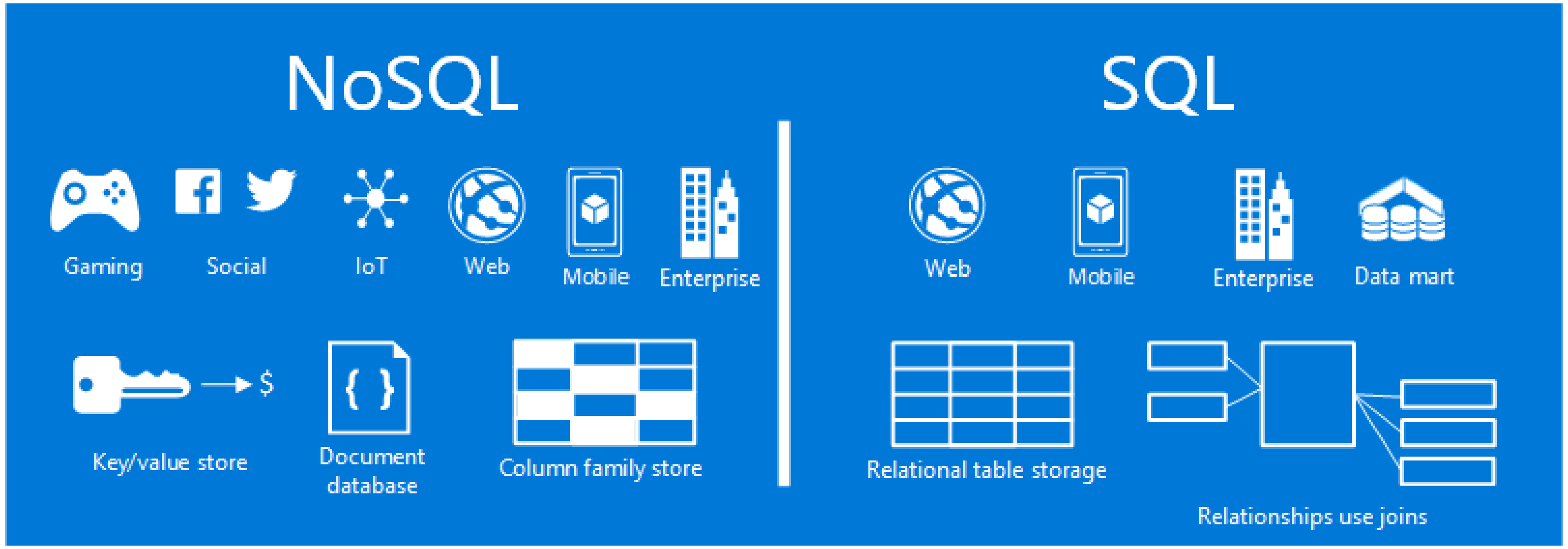
- /user/:id (You cannot get a user information)
- /logout (Attempt to sign out failed)

Getting to understand

“NoSQL: MongoDB”

Why MongoDB?? 😕


SQL
VS
NoSQL
(Structured Data)
(Un-Structured Data)


NoSQL
Gaming
Social
IoT
Web
Mobile
Enterprise
Key/value store
$
Document database
Column family store
SQL
Web
Mobile
Enterprise
Data mart
Relational table storage
Relationships use joins

Why MongoDB?? 😕

	SQL	NoSQL
Database Type	Relational Databases	Non-relational Databases / Distributed Databases
Structure	Table-based	• Key-value pairs • Document-based • Graph databases • Wide-column stores
Scalability	Designed for scaling up vertically by upgrading one expensive custom-built hardware	Designed for scaling out horizontally by using shards to distribute load across multiple commodity (inexpensive) hardware
Strength	• Great for highly structured data and don’t anticipate changes to the database structure • Working with complex queries and reports	• Pairs well with fast paced, agile development teams • Data consistency and integrity is not top priority • Expecting high transaction load

Mongoose (MongoDB)

Mongoose is an elegant **mongoDB** object modeling for **node.js**

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test');

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

Before getting started:

- Install mongoDB in your local machine

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

Mongoose (MongoDb)

First be sure you have MongoDB and Node.js installed.

- Installing a mongoose package

```
$ npm install mongoose --save
```

- Database connection

```
// getting-started.js
const mongoose = require('mongoose');

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

async function main() {
  await mongoose.connect('mongodb://localhost:27017/test');
}
```

```
const options = {
  autoIndex: false, // Don't build indexes
  maxPoolSize: 10, // Maintain up to 10 socket connections
  serverSelectionTimeoutMS: 5000, // Keep trying to send operations for 5 seconds
  socketTimeoutMS: 45000, // Close sockets after 45 seconds of inactivity
  family: 4 // Use IPv4, skip trying IPv6
};
mongoose.connect(uri, options);
```

Options

The `connect` method also accepts an `options` object which will be passed on to the underlying MongoDB driver.

```
mongoose.connect(uri, options);
```

Mongoose (MongoDb)

- Defining your schema

```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const blogSchema = new Schema({
  title:  String, // String is shorthand for {type: String}
  author: String,
  body:   String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs:  Number
  }
});
```

- When you create a new document

```
const Animal = mongoose.model('Animal', animalSchema);
const dog = new Animal({ type: 'dog' });
```

- The permitted SchemaTypes are:
  - String
  - Number
  - Date
  - Buffer
  - Boolean
  - Mixed
  - ObjectId
  - Array
  - Decimal128
  - Map

Mongoose (MongoDb)

- Queries

```
const Person = mongoose.model('Person', yourSchema);

// find each person with a last name matching 'Ghost', selecting the `name` and `occupation` fields
Person.findOne({ 'name.last': 'Ghost' }, 'name occupation', function (err, person) {
  if (err) return handleError(err);
  // Prints "Space Ghost is a talk show host".
  console.log('%s %s is a %s.', person.name.first, person.name.last,
    person.occupation);
});
```

```
// With a JSON doc
Person.
  find({
    occupation: /host/,
    'name.last': 'Ghost',
    age: { $gt: 17, $lt: 66 },
    likes: { $in: ['vaporizing', 'talking'] }
  }).
  limit(10).
  sort({ occupation: -1 }).
  select({ name: 1, occupation: 1 }).
  exec(callback);

// Using query builder
Person.
  find({ occupation: /host/ }).
  where('name.last').equals('Ghost').
  where('age').gt(17).lt(66).
  where('likes').in(['vaporizing', 'talking']).
  limit(10).
  sort('-occupation').
  select('name occupation').
  exec(callback);
```

- `Model.deleteMany()`
- `Model.deleteOne()`
- `Model.find()`
- `Model.findById()`
- `Model.findByIdAndDelete()`
- `Model.findByIdAndRemove()`
- `Model.findByIdAndUpdate()`
- `Model.findOne()`
- `Model.findOneAndDelete()`
- `Model.findOneAndRemove()`
- `Model.findOneAndReplace()`
- `Model.findOneAndUpdate()`
- `Model.replaceOne()`
- `Model.updateMany()`
- `Model.updateOne()`

I want more about Mongoose

☞ https://mongoosejs.com/

Good luck 👌