Open in app

Yogesh Agrawal

170 Followers · About Follow

Mongoose(mongoDB) functions for CRUD Application

Yogesh Agrawal Jan 23, 2018 · 4 min read

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need. MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time unlike your RDBMS where it stores the data in relational format(tabled-structure).

MongoDB preferred because it provide number of features:

- The document model maps to the objects in your application code, making data easy to work with.
- It is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use.
- Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data.

Let's see how can we use MongoDB using mongoose and what is mongoose ?

Mongoose is a MongoDB ***object modeling tool*** designed to work in an asynchronous environment.It 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.

Let's see how to use mongoose:

Prequisites:

1. Download MongoDB and install.[Download link].
2. Download and Install Node.js [Download Link].

Start mongoDB or we can use remote mongoDB [check this link].

Let's start:-

To use mongoose , we need to install mongoose package -

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

At First we have to create connection with MongoDB through mongoose.

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

secret: 'devdacticIsAwesome',
database: 'mongodb://localhost/Test_server'
mongoose.connect(database).then(
  ()=>{console.log("connected")},
  err =>{console.log("err",err);}
);
```

Now, we have to create `schema` for using mongoose as mongo is schema based storage. (I have created a example schema for user).

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

const userSchema = mongoose.Schema({
    name:String,
    age:String,
    state:String,
    country:String
});
module.exports = mongoose.model('User',userSchema);
```

We are ready to move , let's check each function in mongoose :

1. **find() :-** find() method in mongoose retrieve all record from Particular collection.We can pass `QUERY` also to get that specific record.

```
var User = require('./../schema/user');

User.find({})
 .then((data)=>{
    console.log(data);
  })
 .catch((err)=>{
   console.log(err);
 })

// you can pass query parameter to get particular record

User.find({name:"YOUR_NAME"})
 .then((doc)=>{
    console.log(doc);
 })
.catch((err)=>{
    console.log(err);
});
```

2. **findById :-** findById() is used to fetched record from schema based on your mongo Id.

```
User.findById(userId.id)
 .then((doc)=>{
    console.log(doc);
 }).catch((err)=>{
    console.log(err);
 });
```

Here we need to handle casting errors such as wrong `mongoId` Example: -

```
{
    "message": "Cast to ObjectId failed for value
\"5a5ef1d5d48c273c2c2ce75h\" at path \"_id\" for model \"User\"",
    "name": "CastError",
    "stringValue": "\"5a5ef1d5d48c273c2c2ce75h\"",
    "kind": "ObjectId",
    "value": "5a5ef1d5d48c273c2c2ce75h",
    "path": "_id"
}
```

So how to solve this ?

We have to check mongoId whether it is valid or not , if valid then only reuqest else throw error.

```
if (mongoose.Types.ObjectId.isValid(userId.id)){
User.findById(userId.id)
  .then((doc)=> {
     if (doc) {
       console.log(doc)
     } else {
       console.log("No data exist for this id");
     }
 })
.catch((err)=> {
    console.log(err);
 });
} else {
  console.log("Please provide correct Id");
}
```

3. **findOne():-** findOne() method is used to get 1 recored from schema based on condition.

```
if (mongoose.Types.ObjectId.isValid(userId.id)) {
   User.findOne({ _id: userId.id })
    .then((doc) => {
       if (doc) {
         console.log(doc);
       } else {
         console.log("no data exist for this id");
       }
    })
   .catch((err) => {
     console.log(err);
    });
} else {
  console.log("please provide correct id");
}
```

4. **save():-** save() is used to insert a record in collection. It is used with a **Modal object.** i.e. we have to create a modal object.

```
let Newuser = new User(user); // this is modal object.
Newuser.save()
  .then((data)=> {
    console.log(data);
   })
```

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

5. **insert() , insertMany() or create() :-** insert() and insertMany() method both array of records but difference is, in **insert()** it return response as only number of insert and confirmation about inserted record where as in **insertMany()** it return inserted record id and inserted record.

create() is also used to insert array of records but as it is modal based operation , it require object Modal like **save()** and *slow* compare to **insert() & insertMany().**

```
  let Newuser = new User(user);
  let Newuser1 = new User(user);
  let Newuser2 = new User(user);

  // it srore directly to collection

  User.collection.insert([Newuser,Newuser1,Newuser2])
    .then((data)=>{
       resolve(data);
    }).catch((err)=>{
       reject(err);
  })
```

insertMany():

```
  let Newuser = new User(user);
  let Newuser1 = new User(user);
  let Newuser2 = new User(user);

  User.collection.insertMany([Newuser,Newuser1,Newuser2])
    .then((data)=>{
      resolve(data);
    }).catch((err)=>{
      reject(err);
    })
```

create():

```
  let Newuser = new User(user);
  let Newuser1 = new User(user);
  let Newuser2 = new User(user);
```

```
// it is using schema model for operation `User`

User.create([Newuser,Newuser1,Newuser2])
   .then((data)=>{
     resolve(data);
   }).catch((err)=>{
      reject(err);
  })
```

6. **insertOne() :-** insertOne() is used to insert single record in collection. same like `save()` but it is faster as it directly insert into *collection*.

```
let Newuser = new User(user);

User.collection.insertOne(Newuser)
  .then((data)=>{
    resolve(data);
  }).catch((err)=>{
    reject(err);
 })
```

7. **findOneAndUpdate() :-** findoneAndUpdate() method is used for update a record in collection. it is good to use this method because other way to perform same operation is first `find record and then update`. It saves time.It contain a `option` flag like `new:true` by which it will *return updated record* in response.

```
if(mongoose.Types.ObjectId.isValid(id)) {
 User.findOneAndUpdate({_id: id},{$set:{name:user.name}},{new:true})
.then((docs)=>{
    if(docs) {
       resolve({success:true,data:docs});
    } else {
       reject({success:false,data:"no such user exist"});
    }
}).catch((err)=>{
   reject(err);
})
} else {
   reject({success:"false",data:"provide correct key"});
}
```

8. **findByIdAndUpdate() :-** findByIdAndUpdate() method is used to update record based on Id.

```
if(mongoose.Types.ObjectId.isValid(id)) {
User.findByIdAndUpdate(id,{$set:{name:user.name}},{new:true})
.then((docs)=>{
   if(docs) {
     resolve({success:true,data:docs});
   } else {
     reject({success:false,data:"no such user exist"});
   }
}).catch((err)=>{
    reject(err);
})
} else {
  reject({success:"false",data:"provide correct key"});
}
```

9. **update() :-** update() is used for updating a record based on condition. it can update multiple field by using flag `multi:true`

```
User.update({_id:id},{$set:{name:user.name,state:user.state}},
{multi:true,new:true})
  .then((docs)=>{
    if(docs) {
      resolve({success:true,data:docs});
    } else {
      reject({success:false,data:"no such user exist"});
    }
 }).catch((err)=>{
    reject(err);
})
```

10. **remove() :-** remove() is used to delete record based on condition. If we won't provide any condition it will remove all record.

```
if(mongoose.Types.ObjectId.isValid(id)) {
  User.remove({_id: id})
    .then((docs)=>{
      if(docs) {
        resolve({"success":true,data:docs});
      } else {
        reject({"success":false,data:"no such user exist"});
      }
  }).catch((err)=>{
     reject(err);
  })
}else {
  reject({"success":false,data:"please provide correct Id"});
}
```

11. **findOneAndRemove() :-** findOneAndRemove() is used for removing 1 record from collection based on Id.

```
if(mongoose.Types.ObjectId.isValid(id)) {
   User.findOneAndRemove({_id: id})
     .then((docs)=>{
        if(docs) {
           resolve({"success":true,data:docs});
        } else {
           reject({"success":false,data:"no such user exist"});
        }
   }).catch((err)=>{
       reject(err);
   })
 } else {
     reject({"success":false,data:"please provide correct Id"});
 }
```

So, thats all for this post for more updates and code checkout my github

yug95 (yogesh agrawal)
yug95 has 17 repositories available. Follow their code on GitHub.
github.com

Happy Coding , Thanks for your precious time. I would love to hear from you.

JavaScript