mongoose

mongoose

elegant mongodb object modeling for node.js

mongoose

Mongoose provides a straight-forward, schema-based solution to model your application data.

It includes

- built-in type casting
- validation
- query building
- business logic hooks
- and more, out of the box.

mongoose

```
//install mongoose first
$ npm install mongoose --save
```

mongoose

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/training');

//Mongoose Schema
var catSchema = mongoose.Schema({
    name: String,
});

//Mongoose Model
var Cat = mongoose.model('Cat', catSchema);

//Mongoose Object
var kitty = new Cat({ name: 'Zildjian' });

//Save kitty to database
kitty.save();
```

schema definitions

- A schema takes a description object which specifies its keys and their types
- Types are mostly normal JS

```
new Schema({
    title: String,
    body: String,
    date: Date,
    hidden: Boolean,
    meta: {
        votes: Number,
        favs: Number
    }
});
```

schema types

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array

```
new Schema({
    title: String,
    body: String,
    date: Date,
    hidden: Boolean,
    meta: {
        votes: Number,
        favs: Number
    }
});
```

Nested objects

- Creating nested objects is easy
- Just assign an object as the value

```
var PersonSchema = new Schema({
    name: {
            first: String,
            last: String
          }
});
```

Array fields

- Array fields are easy
- Just write the type as a single array element

```
var PersonSchema = new Schema({
    name: {
            first: String,
            last: String
          },
    hobbies: [String]
});
```

Schema Use Case

- Let's start writing a photo taking app
- Each photo is saved in the DB as a Data URL
- Along with the photo we'll save the username

```
var PhotoSchema = new Schema({
    username: String,
    photo: String,
    uploaded_at: Date
});

var Photo = mongoose.model('Photo',
                                    PhotoSchema
```

Creating New Objects

- Create a new object by instantiating the model
- Pass the values to the constructor

```
var mypic = new Photo({
        username: 'ynon',
        photo: 'foo',
    uploaded_at: new Date()
});
```

Creating New Objects

- After the object is ready, simply save it

```
mypic.save()
```

What Schema Can Do For You

- Add validations on the fields
- Stock validators: required, min, max
- Can also create custom validators
- Validation happens on save

```
var PhotoSchema = new Schema({
    username: { type: String,
                required: true },
    photo:    { type: String,
                required: true },
    uploaded_at: Date
});
```

What Schema Can Do For You

- Provide default values for fields
- Can use a function as default for delayed evaluation

```
var PhotoSchema = new Schema({
    username: { type: String,
                required: true },
    photo:    { type: String,
                required: true },
    uploaded_at:
              { type: Date,
                default: Date.now }
});
```

What Schema Can Do For You

- Add methods to your documents

```
var EvilZombieSchema = new Schema({
    name: String,
    brainz: { type: Number, default: 0 }
 });

EvilZombieSchema.methods.eat_brain = function() {
        this.brainz += 1;
};
```

Custom Validators

- It’s possible to use your own validation code

```
var toySchema = new Schema({
    color: String,
    name: String
});

toySchema.path('color').validate(function(value) {
    return ( this.color.length % 3 === 0 );
});
```

Schema Create Indices

- A schema can have some fields marked as “index”. The collection will be indexed by them automatically

```
var PhotoSchema = new Schema({
    username: { type: String, required: true, index: true },
    photo: { type: String, required: true },
    uploaded_at: { type: Date, default: Date.now }
});
```

Schemas Create Accessors

- A virtual field is not saved in the DB, but calculated from existing fields. “full-name” is an example.

```
personSchema.virtual('name.full').get(function () {
    return this.name.first + ' ' + this.name.last;
});

personSchema.virtual('name.full').set(function (name) {
    var split = name.split(' ');
    this.name.first = split[0];
    this.name.last = split[1];
});
```

Querying Data

- Use Model#find / Model#findOne to query data

```
// executes immediately, passing results to callback
MyModel.find({ name: 'john', age: { $gte: 18 }},
              function (err, docs) {
                // do something with data
                // or handle err
                });
```

Querying Data

- You can also chain queries by not passing a callback
- Pass the callback at the end using *exec*

```
var p = Photo.find({username: 'ynon'}).
            skip(10).
            limit(5).
            exec(function(err, docs) {
                    console.dir( docs );
        });
```

Other Query Methods

- find(cond, [fields], [options], [cb])
- findOne (cond, [fields], [options], [cb])
- findById (id, [fields], [options], [cb])
- findOneAndUpdate(cond, [update], [options], [cb])
- findOneAndRemove(cond, [options], [cb])

Counting Matches

- Use count to discover how many matching documents are in the DB

```
Adventure.count({ type: 'jungle' }, function (err, count) {
    if (err) ..
    console.log('there are %d jungle adventures', count);
});
```

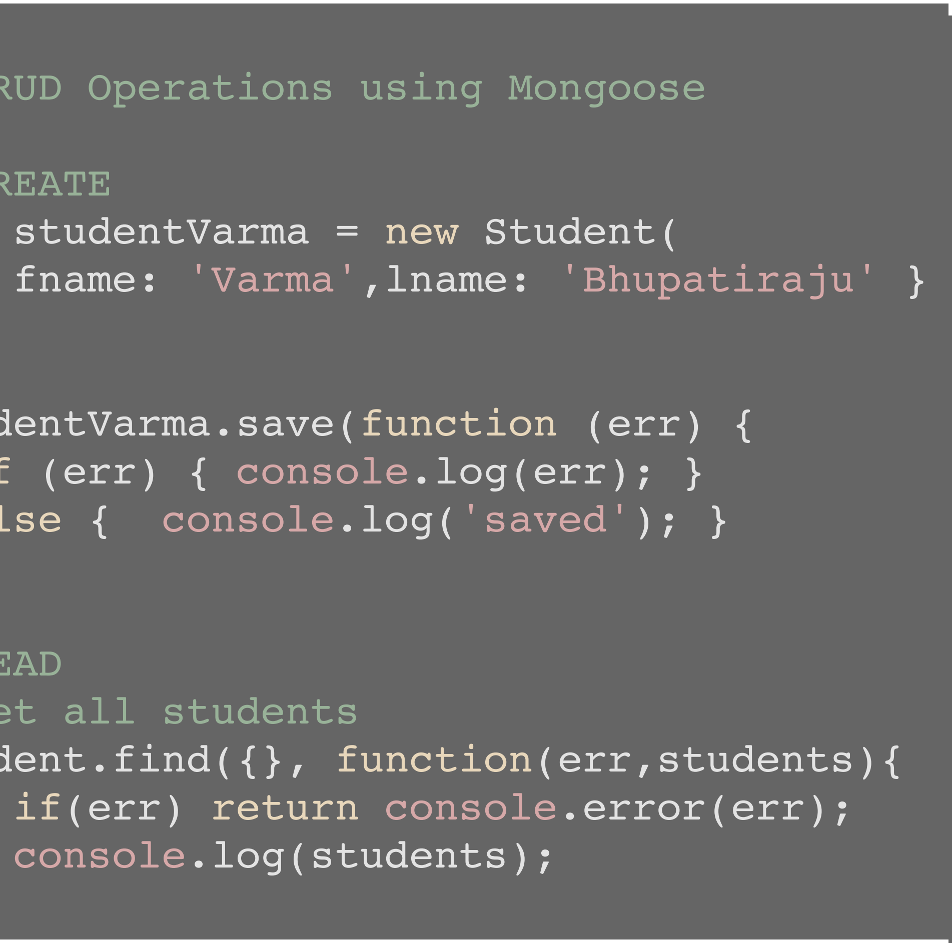
mongoose CRUD

```
ndex.js
 mongoose = require('mongoose');
goose.connect('mongodb://localhost:27017/training');

ith Mongoose, everything is derived from a Schema.
et's get a reference to it and define our student.
 studentSchema = mongoose.Schema({
 fname: String,
 lname: String,
 age: Number,
 degree: String,
 gender: String


he next step is compiling our schema into a Model.
 Student = mongoose.model('Student', studentSchema, "students");
```

mongoose CRUD



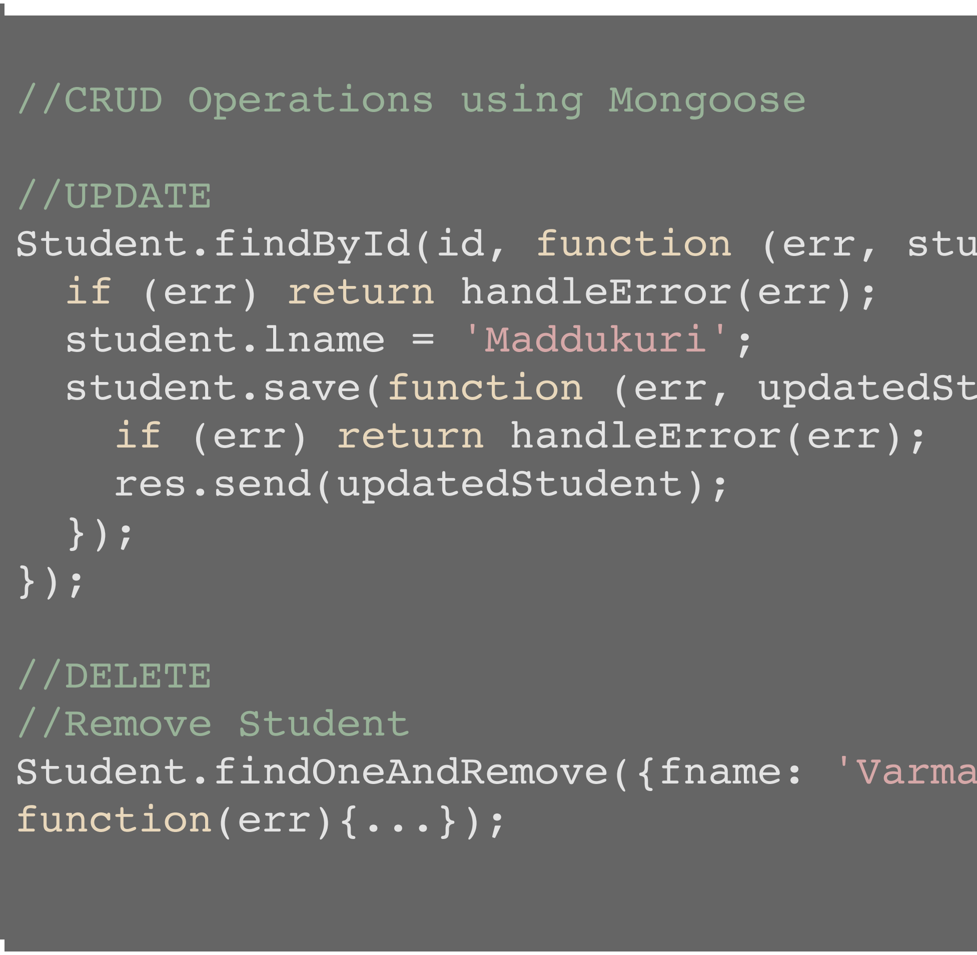
```
RUD Operations using Mongoose

REATE
 studentVarma = new Student(
 fname: 'Varma',lname: 'Bhupatiraju' }


dentVarma.save(function (err) {
f (err) { console.log(err); }
lse {  console.log('saved'); }



EAD
et all students
dent.find({}, function(err,students){
 if(err) return console.error(err);
 console.log(students);
```



```
//CRUD Operations using Mongoose

//UPDATE
Student.findById(id, function (err, stu
  if (err) return handleError(err);
  student.lname = 'Maddukuri';
  student.save(function (err, updatedSt
    if (err) return handleError(err);
    res.send(updatedStudent);
  });
});

//DELETE
//Remove Student
Student.findOneAndRemove({fname: 'Varma
function(err){...});
```