Mongoose

MongoDB + Node.JS

What's That

- An Object Relational Mapper for Node.JS
- Handles gory details so you don't have to
- Fat Models

Agenda

- [ ] Hello Mongoose
- [ ] Schema and Data Types
- [ ] Custom Validators
- [ ] Querying Data
- [ ] Poor Man’s Joins (Populate)
- [ ] Mongoose Plugins

Online Resources

- http://mongoosejs.com/
- https://github.com/LearnBoost/mongoose
- http://www.youtube.com/watch?v=4fQsDiioj3I
- irc: #mongoosejs on freenode

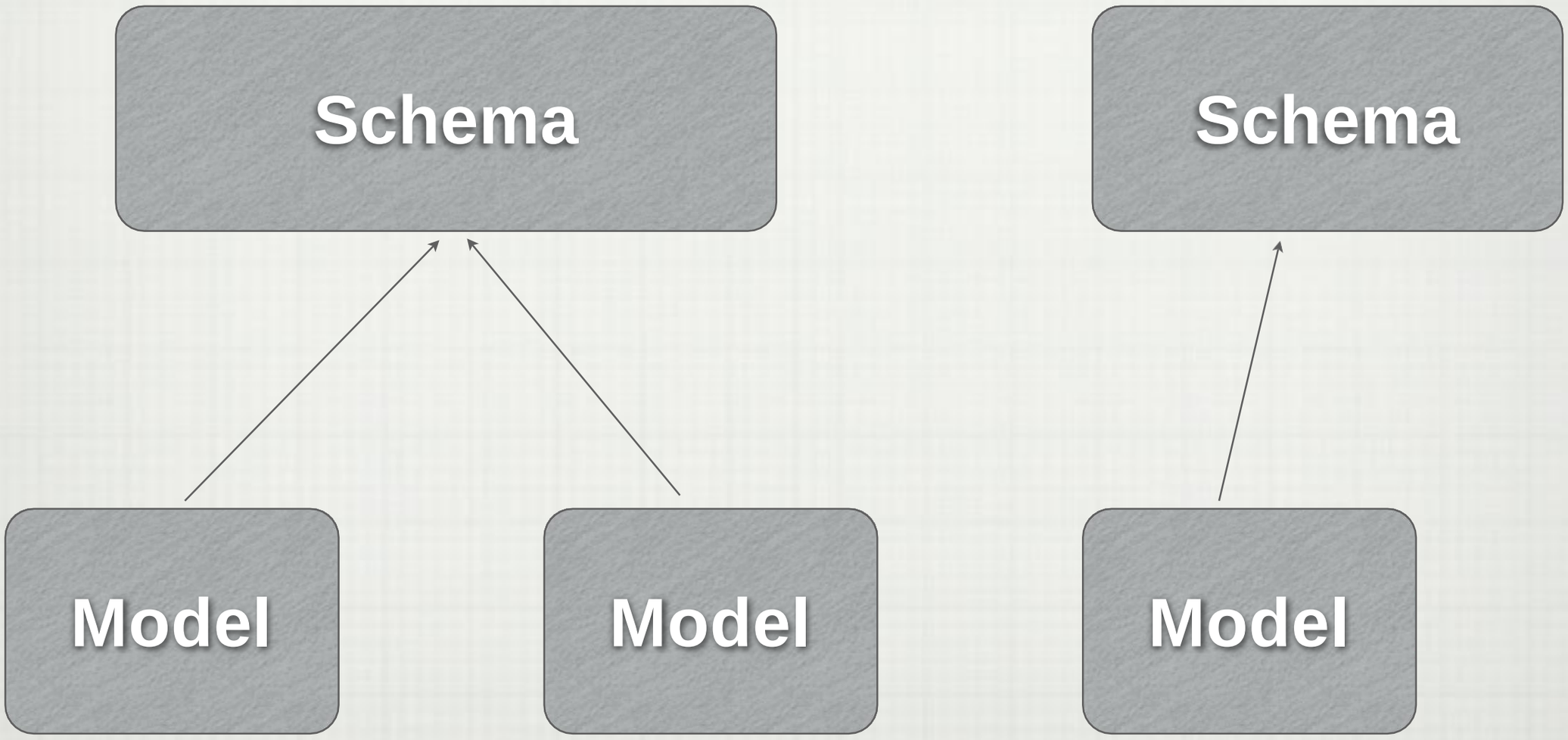
Hello Mongoose

```
var mongoose = require('mongoose');
mongoose.connect('localhost', 'test');

var schema = mongoose.Schema({ name: 'string' });
var Cat = mongoose.model('Cat', schema);

var kitty = new Cat({ name: 'Zildjian' });
kitty.save(function (err) {
  if (err) // ...
  console.log('meow');
});
```

Mongoose Objects



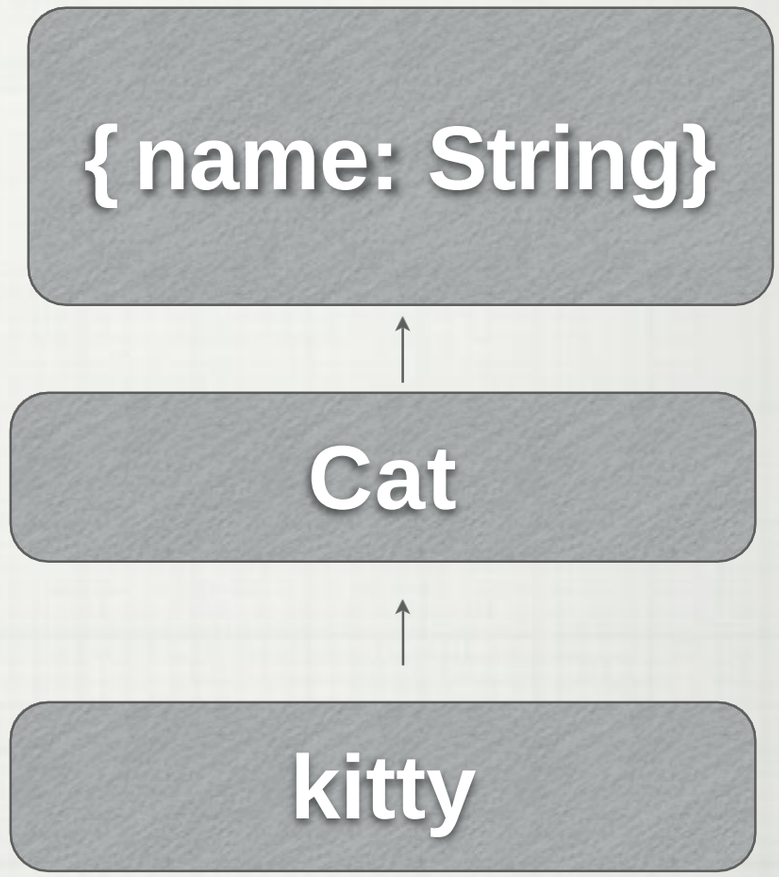
Mongoose Objects

```
var mongoose = require('mongoose');
mongoose.connect('localhost', 'test');

var schema = mongoose.Schema(
                { name: 'string' });

var Cat = mongoose.model(
                'Cat', schema);

var kitty = new Cat(
                { name: 'Zildjian' });
```



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

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 ctor

```
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];
});
```

Q & A

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);
});
```

Lab

- [ ] Create a Schema called “Album”
- [ ] Add fields: artist, year, tracks
- [ ] Create a model and a document
- [ ] Add validator for year
- [ ] Save it in the DB

Lab

- [ ] Create 5 albums from years 2008, 2009, 2010, 2011, 2012
- [ ] Query the 3 newest albums
- [ ] Print the artist name and the number of tracks
- [ ] Print the artist who has the most albums

Populating Collections

- Mongo has no joins
- Let’s Fake Them

Start With Relationships

- Execute the following to create an initial relationship https://gist.github.com/4657446
- Watch the data in the DB:
- Album.artist = ObjectId("5106b6e6fde8310000000001")

Use Query#populate

- query#populate sends another query for the related object

```
Album.findOne().exec(function(err, doc) {
  // prints undefined
  console.log( doc.artist.name );
});


Album.findOne().populate('artist').exec(function(err, doc) {
  // prints Pink Floyd
  console.log( doc.artist.name );
});
```

Use Query#populate

- Full method signature:
- Query#populate(path, [fields], [model], [cond], [options])
- cond is a query condition object (i.e. { age: { $gte: 21 }}
- options is a query options object (i.e. { limit: 5 }
- Helps when populating arrays

Mongoose Plugins

- A plugin connects to the Schema and extends it in a way

Mongoose Plugins

- A mongoose plugin is a simple function which takes schema and options
- Demo: lastModifiedPlugin
  https://gist.github.com/4657579

Mongoose Plugins

- find or create plugin:

  https://github.com/drudge/mongoose-findorcreate

Mongoose Plugins

- Hashed password field plugin: https://gist.github.com/4658951

Mongoose Plugins

- Mongoose troops is a collection of useful mongoose plugins: https://github.com/tblobaum/mongoose-troop