Lecture 12

Using MongoDB with Express

Agenda

1. Why use MongoDB?
2. Installing MongoDB
3. Connecting it to Express

Why use MongoDB?

The NoSQL Database Layer

- MongoDB is an open source database source, that lets you store arbitrary data as JSON objects (internally represented as BSON data types).
- Lightweight, fast and reliable.
- MongoDB was written to work seamlessly with a JavaScript web stack.

Installing MongoDB

Assuming you already have an Express app working locally, you can either host your MongoDB database instance locally or on the cloud.

- If you're installing locally, you need to install MongoDB on your machine using homebrew.
- Install the NPM package for your application

```
$  npm install mongodb --save
```

- Alternatively, check out the amazing NPM package, Mongoose!

There are a few lines of code needed to connect your Express app to the database:

```
var MongoClient = require('mongodb').MongoClient;
var MONGO_URI = '';

MongoClient.connect(MONGO_URI, function(err, db) {
  if (err) return console.log(err)

  // connected database is 'db'

  app.listen(3000, function() {
    console.log('listening on 3000')
  });
})
```

The above code, which should reside in your `app.js` file, only starts the server once the connection is established.

The MongoDB Connection URI

The MongoDB URI is a string that represents the location of your MongoDB database instance. If you're hosting this on your local machine, the connection URI is:

```
'mongodb://localhost:27017/<name-of-your-db>'
```

If you're hosting your MongoDB instance on the cloud then the connection URI will likely look something like this:

```
'mongodb://<dbuser>:<dbpassword>' // auth
+ '@ds04569826.mongolab.com:47955' // mongolab host
+ '/<your-db-name>' // specific db
```

Example

Check out the code for the Penn Course Review homework assignment!

We connect using Mongoose, but the principles behind the connection are all the same.

Mongo Collections

Every database consists of collections. Collections are a way of grouping your documents/files within a MongoDB data source. You save individual documents within a Collection.

```
app.post('/quotes', function(req, res) {
  var quotes = db.collection('quotes');
  quotes.save(req.body, function(err, result) {
    if (err) return console.log(err);

    console.log('saved to database');
    res.redirect('/')
  });
});
```

Fetching Data

Use the `find` or `findById` functions to select documents within collections and extract them from the database.

```
app.get('/', function(req, res) {
  var quotes = db.collection('quotes');
  var all_quotes = quotes.find();
  var second_quote = quotes.findById(2);
});
```

Mongoose

Straight-forward, schema-based solution to model your app data.

Website

Basic Boilerplate

Similar syntax. Let's make a User Schema!

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

var userSchema = new Schema({
  name: String,
  username: { type: String, required: true },
  password: { type: String, required: true },
  created_at: Date,
  updated_at: Date
});

var User = mongoose.model('User', userSchema);
```

Models, Schemas, and Methods

A mongoose Model is a "constructor" that allows us to make documents (think objects) that can be saved and retrieved from the database

A mongoose Schema allows us to define attributes for the documents

A mongoose Method can be defined on a Schema (think methods in a class).

Custom Methods

Let's make our users cool!

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

var userSchema = // ref slide 12

userSchema.methods.coolify = function(callback) {
  this.name = this.name + ' is cool';
  // standard callback pattern!
  callback(null, this.name);
}

var User = mongoose.model('User', userSchema);

module.exports = User;
```

Saving a Cool User

```
var User = require('./user-module-location');

var devesh = new User({
  name: 'Devesh'
});

devesh.coolify(function(err, name) {
  if (err) throw err;
  console.log(name); // Devesh is cool
});

devesh.save(function(err) {
  if (err) throw err;
  console.log('User saved!');
});
```

Get a User and update them

```
var User = require('./user-module-location');


User.find({ name: 'Devesh' }, function(e, user) {
  if (e) throw e;
  user.name = 'Devesh Dayal';
  user.save(function(err) {
    if (err) throw err;
    console.log('User successfully updated!');
  });
});
```

Delete a User

```
User.find({ name: 'Devesh' },  function(e, user) {
  if (e) throw e;
  user.remove(function(e) {
    if (e) throw e;
    console.log('User successfully deleted!');
  });
});
```

Do Something Before A User is Saved

```
userSchema.pre('save', function(next) {
  var user = this;
  // check to see if the password was changed
  if (!user.isModified('password')) return next();

  // bcrypt generates secure password hashes
  bcrypt.hash(user.password, 'salt',
    function(err, hash) {
      user.password = hash;
      next();
  });
});
```

Check a User's Hashed Password

```
userSchema.methods.checkPassword =
function(possiblePass, cb) {
  bcrypt.compare(possiblePass, this.password,
  function(err, isRight) {
    if (err) return cb(err);
    cb(null, isRight);
  });
};
```

How to Use this to Log in someone

```
app.post('/login', function(req, res) {
  username = req.body.username;
  password = req.body.password;

  if (User.findOne({ username: username },
    function (err, user) {
      user.checkPassword(password,
        function(err, isRight) {
          if (isRight) {
            req.session.username = username;
            res.redirect('/protected');
          } else {

            res.send('wrong');
          }
        });
```

Example Login System

https://github.com/abhisuri97/mongo-login-example

Also check out passport.js for login schemes with facebook, google, twitter OAuth etc.