Mongoose

Objectives

- Mongoose
  - Properties for organizing and validating data
  - Models

Demo: Warm-Up (15 min)

```
$ node 13-Warm-Up/Solved/server.js
```

http://localhost:3000

Activity: Warm-Up (15 min)

See 13-Warm-up

Your goal is to complete the routes in the server file so the site can display and edit the book data. Don't worry about the front end, just use MongoJS to finish the routes

Review: Warm-Up

- morgan?
  - HTTP request logger
    - See console
  - https://www.npmjs.com/package/morgan
- How does each route work?
  - Basically, each one is taking the body of the front end's request, and using that information in the relevant MongoJS queries, as the comments in the file describe

There Must Be a Better Way!

- Mongo is very flexible, and makes it a cinch to save & retrieve data.
- But sometimes we need to be stricter. We don't want to save 'true' or an actual array as a book, or save a book without a title, or save a book with the same name and author. But the flexibility of MongoDB makes that possible.
- Sometimes, we want to combine two collections for information. What if we have our authors and their books stored separately. How would we display a list of all author data along with all of the books they wrote?
  - MySQL makes that possible with joins, but how do we do that with MongoDB

Mongoose

http://mongoosejs.com/

Mongoose == Sequelize

- Mongoose lets us define models for our Mongo data.
    - No surprise data types for particular fields.
    - We can set required fields, too.
- We can create custom functions to handle our data
- We can combine collections with "populate," a method that offers very similar functionality to a MySQL join

Activity: Mongoose (5 min)

With your neighbor, discuss when you might want to use Mongoose versus a vanilla MongoJS set-up or MySQL with Sequelize

Review: Mongoose

Use Mongoose when you want the simplicity of parsing BSON data in a website, along with the same kinds of benefits that MySQL schemas offer.

Demo: Models

- 14-SchemaExample
- server.js
  - Note that the file loads mongoose and uses the url path of the MongoDB database to connect with MongoDB
  - Note that the file loads in the exampleModel.js file with require, and that this model is where much of the beauty of Mongoose lies
  - Models get called like objects instantiated from classes.
- exampleModel.js
  - We first set up a schema class, then define a schema, and use the schema to create a model

```
var Schema = mongoose.Schema;
```

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection.

```
var ExampleSchema = new Schema({
    // loads o’ code
});
```

To use our schema definition, we need to convert our Schema into a model we can work with.

```
var Example = mongoose.model("Example", ExampleSchema);
```

RTFM http://mongoosejs.com/docs/guide.html

```
var content = new Example(req.body);
```

```
content.save(function(error, doc) {
```

RTFM http://mongoosejs.com/docs/models.html

Demo: Schema

```
$ 15-User-Schema/Solved/server.js
```

Activity: Schema (20 min)

See 15-User-Schema

Review: Schema

But what about when we want more functionality than what these schema properties offer?
What if we want to create new entries with the information that the user sends automatically?

Custom methods

Demo: Custom Methods with Schema

```
$ node 16-Custom-Methods/server.js
```

- userModel.js
    - Note the custom methods
- server.js
    - Note how the functions fall underneath the initial definition of the user object, which accepts req.body (the form info). The functions add data to the user variable, which then gets saved to MongoDB

Activity: Custom Methods with Schema (10 min)

See 17-Custom-Method-Exercise

Review: Custom Methods with Schema

- userModel.js
  -
- server.js
  -

Lunch O’Clock

Reverse Engineering Time

https://docs.google.com/document/d/1ZqAEk_YDP9PHu3fka5f7AG6lyU-dseDm224NWo_x6yo/edit?usp=sharing

!important

Demo: Populate

```
$ node 18-Intro-Populate/server.js
```

- http://localhost:3000/
- http://localhost:3000/books
- http://localhost:3000/library

What if we want to see the data for all of the books stored in our library?

Populate lets users relate one collection to another.

http://localhost:3000/populated

- Library.js
  - reference to Book in schema
- server.js
  - .populate()

RTFM http://mongoosejs.com/docs/populate.html

Activity: Populate (10 min)

See 19-Populate-Exercise

Review: Populate

See 19-Populate-Exercise

Demo: Scraping with Mongoose

```
$ node 20-Scraping-With-Mongoose/Solved/server.js
```

http://localhost:3000/scrape

Activity: Scraping with Mongoose (30 min)

See 20-Scraping-With-Mongoose

Review: Scraping with Mongoose