Mongo/Mongoose

Mongo/Mongoose Outline

I'd start by continuing on from the previous day, but this time building in fetch on the front end, and building a very small in-memory structure to use as a database (Justin object with some bits and pieces). From here we could play around (exercises again) with sending information to and from the front end, and building up our rudimentary db.

The limitations of this would become obvious (hopefully), and so we would want a way to persist data over time, and this would necessitate a db. A proper one. From this point we can talk about dis generally, but then move on to talking about SQL and relational dis, but pivot quickly into MongoDBs (and NoSQL more generally). The main thing would be to get across the basic ideas (documents, DB server, CLI) quickly, but I feel like we might get it all into the cloud quickly too, and then work from there.

This would lead to another exercise, and this exercise can point out another problem, and that being that we have a fairly clumsy way of getting data in and out, and also that we might want to do some validations. Mongoose is the next step.

Here I would talk about ORMs and ODMs, and move to taking about models and validations. We would set up some very basic models, and then some validations for these, and then run the forms into the server, and into the db via the models.

There are many extensions that could be built from there.

Pathway

- Mongo installation
- Mongo locally
- Mongo into the cloud
- Basic mongo commands (but keep very brief)
- Mongoose, and Mongoose models
- Exercise around these models and searching

DBs

In the last lesson we started setting up

Mongoose Models

We have set up our MongoDB locally, and we are also set up in the cloud. We have practiced getting documents in and out of collections, and now we are going to begin using a tool to make that easier, and also to help us with some validations. That tool is MongooseJS, more usually called Mongoose.

First lets talk more generally about what it is that we are doing here.

DBs

We have talked about DBs and why they are useful (for persistence of data, and because they are set up to store and retrieve that data quickly). If we were using the another type of DB, we also might like a tool

to help us be able to still write in the JS that we are used to, and that can make these queries in and out of the DB easier for us. Were we to be using an SQL based DB, then we would like a tool to be able to write JS, and end up with that making the SQL calls to the DB.

Here, as we know, we have a document DB, and so instead of tables, we have objects that resemble the objects that we have been learning about in our JS content. But, they are not exactly the same, and obviously we have been using the great, but somewhat clunky, MongoDB commands to get things in and out of the DB.

Here we are going to use Mongoose, which is a program that will help us to write in a way that is totally in JS, which makes our life much easier. But that's not all.

The two things to discuss about what also makes Mongoose interesting is that it helps us to create our models, and it will help with validations.

Let's briefly discuss both of these concepts.

Why have models

Our models are the shape of the documents that are being loaded into the database. Let's say that our DB is going to contain albums, so that a user can upload a list of their favourite albums. An album is has some features that we would like to capture. Most obviously an album has a title. We can imagine that this title is a string. So let's start creating this album.

We can do this in JS, and start like this:

```
album1 = {
    title: “Back In Black”,
    artist: “ACDC”
}
```

This is all well and good as a JS object, but we are going to import Mongoose to help us to formalise this as the shape of our data that will be going into the database.

This is the code we need to start making a model for our albums:

```
const mongoose = require("mongoose")

const Schema = mongoose.Schema

let album = new Schema(
    {
        title: {
            type: String
        },
        artist: {
            type: String
        },
    },
```

```
    {
      collection: "Albums"
    }
)

module.exports = mongoose.model("albums", album)
```

here explain the code above

- We are requiring in the package
- We are getting the Schema part of the Mongoose package
- We are defining an album broadly
- We are exporting it out
- Explain about connecting to the particular collection, and naming more broadly

Validations

Validations are the fine tuning of the models. This is the way that we define the specifics of how we need the data to look. For example, we can ssay whether a particular field is required (an album is not really complete until it at least has a title and artist), or even something more specific like the format of an email.

```
let album = new Schema(
    {
        title: {
            type: String,
            required: true
        },
        artist: {
            type: String,
            required: true
        },
    },
    {
      collection: "Albums"
    }
)
```