MongooseJS

Mongoose (https://mongoosejs.com/) is a library that you can use to interact with your MongoDB server (just like the native Node.js MongoDB driver). It differs in that it bases everything on a Schema.

Install MongooseJS in your Node.js app

In your terminal, run **`npm i mongoose`**.

In your **code**, to connect to MongoDB:

```
const mongoose = require('mongoose');
await mongoose.connect('<replace_with_mongodb_url>');
```

Create a Schema

This allows you to define the fields for a particular collection and the data types for them so there's less room for error and ensures consistent, valid data will be entered into the database.

The Schema is like the blueprint for what a document for a particular collection would entail.

In your code:

```
const PetSchema = new Schema({
  name: String,
  type: String,
  breed: String,
  age: Number
});
```

Valid Schema data types are:

- **`String`**
- **`Number`**
- **`Date`**
- **`Boolean`**
- `ObjectId`
- `Buffer` (converts to MongoDB binary)
- `Mixed` (will accept anything as the data)
- `Array` (can be a SchemaType or an array of subdocuments)
- `Decimal128` (128-bit floating decimal—when using as a type in a Schema, use mongoose.Decimal128 as the data type)
- `Map` (use to define a subdocument with arbitrary keys)
- `UUID` (universally unique identifier—stores as UUID, BSON type 4)

You can then use this Schema to make a model which we would use to add, update, delete or read from the database.

Create a model from a Schema

In your code:

`const Pet = mongoose.model("Pet", PetSchema);`

This will create a model named Pet (first parameter in the model() method) which uses the PetSchema. By default, the above line will assume a collection with the name "pets" in MongoDB. Mongoose takes the model name, makes it all lowercase and takes the plural form of the word as the assumed collection name.

If you have an existing collection and you want to associate *that* collection with your model, use the following code:

```
const Pet = mongoose.model("Pet", PetSchema, "<name-of-your-MongoDB-collection>");
```

Alternatively, you can associate a collection when you create a Schema.

```
const PetSchema = new mongoose.Schema(
  {
    name: String,
    type: String,
    breed: String,
    age: Number
  },
  {
    collection: "<name-of-your-MongoDB-collection>"
  }
);
const Pet = mongoose.model("Pet", PetSchema);
```

Insert into a collection

Once you've got your model, you can then insert by creating an instance of a model.

```
const pet1 = new Pet({
  name: "Max",
  type: "cat",
  breed: "Maine Coon",
  age: 2
}); //where "Pet" is the name of your Pet model
await pet1.save(); //saves the new pet into the collection on MongoDB
```

If you want to insert many documents into a collection, use insertMany() and use an array of JSON objects (documents) as the parameter.

```
const petList = [
  {
    …<pet 1>…
  },
  {
    …<pet 2>…
  }
];
await Pet.insertMany(petList);
```

Read from a collection

You can use the following methods with your model:

- `async find(<query>)`
  Returns an array of matching documents.
- `async findById(<id>)`

Find all

```
//assuming there is a model named Pet
let results = await Pet.find({}); //results is an array
```

Find by id

```
//assuming the model is Pet
let result = await Pet.findById()
```

Update a document

You can use the following queries for updating one document:

- `async updateOne(<filter>, <updated-fields-as-object>, <options>)`
- `async replaceOne(<filter>, <replacing-document>, <options>)`
  This is similar to updateOne() but is meant to be used if the entire document is updated (though updateOne() would still work anyway).

For example, in your code:

```
//assuming you're connected to MongoDB
const result = await Pet.updateOne({ name: "Max" }, { age: 8 });
```

Delete a document

You can use:

- `async deleteOne(<filter>)`

For example:

```
//assuming you're connected to MongoDB via Mongoose
await Pet.deleteOne({ breed: "fish" }); //delete all fish from "pets" collection
```

A quick example

In this example, we'll keep the code simple just to demo some functions. Some skeleton code is already on Blackboard in the *Week 5 lesson*. The complete code is below and will be added and demoed in class.

/modules/pets/db.js

This file contains the Mongoose code to connect and set up the Schema and model.

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

const dbUrl =
`mongodb+srv://${process.env.DBUSER}:${process.env.DBPWD}@${process.env.DBHOST}
/testdb?retryWrites=true&w=majority&appName=Cluster0`;

//set up Schema and model
const PetSchema = new mongoose.Schema({
  name: String,
  type: String,
  breed: String,
  age: Number
});
const Pet = mongoose.model("Pet", PetSchema);

//MONGODB FUNCTIONS
async function connect() {
  await mongoose.connect(dbUrl); //connect to mongodb
}

//Get all pets from the pets collection
async function getPets() {
  await connect();
  return await Pet.find({}); //return array for find all
}
//Function to initialize pets with some content.
async function initializePets() {
  await connect();
  const pets = [
    {
      name: "Max",
      type: "cat",
      breed: "Maine Coon",
      age: 7
    },
    {
      name: "Mittens",
      type: "cat",
      breed: "Ragdoll",
      age: 3
    },
    {
      name: "Spot",
      type: "dog",
      breed: "Dalmatian",
      age: 6
    }
  ];
  await Pet.insertMany(pets);
}
//Insert one document into pets collection
async function addPet(petName, petType, petBreed, petAge) {
  await connect();
  let pet = new Pet({
    name: petName,
    type: petType,
    breed: petBreed,
    age: petAge
  });
  await pet.save(); //save to db
}
async function updateName(oldName, newName) {
  await connect();
  await Pet.updateOne({ name: oldName }, { name: newName });
}

module.exports = {
  initializePets,
  getPets,
  addPet,
  updateName
}
```

/index.js:

```
const express = require("express");
const path = require("path"); //needed when setting up static/file paths
const dotenv = require("dotenv");

//load the environment variables from .env
```

```
dotenv.config();

const db = require("./modules/pets/db"); //load db.js and initialize with some
new documents

//set up the Express app
const app = express();
const port = process.env.PORT || "8888";

//set up application template engine
app.set("views", path.join(__dirname, "views")); //the first "views" is the
setting name
//the second value above is the path: __dirname/views
app.set("view engine", "pug");

//set up folder for static files
app.use(express.static(path.join(__dirname, "public")));

//USE PAGE ROUTES FROM ROUTER(S)
app.get("/", async (request, response) => {
  let petList = await db.getPets();
  if (!petList.length) {
    await db.initializePets();
    petList = await db.getPets();
  }
  response.render("index", { pets: petList });
});
app.get("/add", async (request, response) => {
  await db.addPet("Fred", "fish", "Koi", 1);
  response.redirect("/");
});
app.get("/update", async (request, response) => {
  await db.updateName("Max", "Max II");
  response.redirect("/");
})

//set up server listening
app.listen(port, () => {
  console.log(`Listening on http://localhost:${port}`);
});
```

index.pug: (assuming there's a layout.pug with the rest of the HTML structure)

```
extends common/layout

block layout-content
  .main-content
    h1.page-title Testing Mongoose
    a(class="button", href="/add") Test add pet
    each pet in pets
      div
        h2 #{pet.name}
        div Breed: #{pet.breed} (#{pet.type})
        div Age: #{pet.age}
```

More

You can write your filters for Mongoose just as you would have done for MongoDB Shell or the native Node.js driver. Take a look at these resources to explore more:

- https://www.mongodb.com/docs/manual/reference/operator/query/
  Contains links for different operations which could be done on a query (e.g. where greater than).
- https://www.mongodb.com/resources/products/capabilities/aggregation
  This is for MongoDB in general, but Mongoose has an aggregate() method as well and the syntax for the pipeline is essentially the same (see the first snippet in the example code for Mongoose).
- https://www.workfall.com/learning/blog/how-to-perform-a-session-based-user-authentication-in-express-js/
  This is a tutorial showing you how to use sessions for a login in Express. You may have seen sessions in your PHP class so the concept is probably more familiar to you. In the tutorial, it's using Jade template files (Jade is now Pug so just use Pug). Also, the HTML for the form is a little improper (e.g. using p instead of label) so just make sure to use proper HTML, but the bones of the code is fine.
- https://medium.com/@anandam00/build-a-secure-authentication-system-with-nodejs-and-mongodb-58accdeb5144
  This is another login example but using JSON web tokens.