Web Framework 1 – Week 9

Professor: Harman Mann

Summer 2024

Agenda

Adding Mongoose into our Node Application

Using callback, Promise, Async/await

- The Node.js driver uses the asynchronous Javascript API to communicate with your MongoDB cluster.
- Asynchronous Javascript allows you to execute operations without waiting for the processing thread to become free. This helps prevent your application from becoming unresponsive when executing long-running operations.
- https://www.mongodb.com/docs/drivers/node/current/fundamentals/promises/

Mongoose.js

- In terms of Node. js, mongodb is the native driver for interacting with a mongodb instance and mongoose is an Object modeling tool for MongoDB.
- Mongoose is built on top of the MongoDB driver to provide programmers with a way to model their data.
- Why we need mongoose?

Mongoose.js

- Using Mongoose, a user can define the schema for the documents in a particular collection. It provides a lot of convenience in the creation and management of data in MongoDB.
- Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node. Js
    - Mongoose plays as a role of abstraction over your database model.

Sending JSON data to MongoDB BSON

- As mentioned in a previous lecture, MongoDB uses BSON (Binary)
- For us to send information over, we must create a file called a Model, which contains a template for how information should be stored and sent over to MongoDB

Lets install a new Module: Mongoose

- MongoDB and mongoose aren’t native to our application, so its something we need to download and add onto our projects
- We will need to go and add it using:
  - **npm install mongoose**
- Then we must declare it inside our server file

Connecting to MongoDB

```
//Import the mongoose module
const mongoose = require('mongoose’);
//Set up default mongoose connection
const mongoDB = 'mongodb://127.0.0.1/my_database’;
mongoose.connect(mongoDB, {useNewUrlParser: true, useUnifiedTopology: true});
//Get the default connection
var db = mongoose.connection;
//Bind connection to error event (to get notification of connection errors)
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
```

Creating multiple connections

- To create additional connections, you can use mongoose.createConnection()
- This takes the same form of database URI (with host, database, port, options etc.) as connect() and returns a Connection object)

Defining and creating models

- Models are defined using the Schema interface.
- The Schema allows you to define the fields stored in each document along with their validation requirements and default values.
- Each model maps to a collection of documents in the MongoDB database.
  - The documents will contain the fields/schema types defined in the model Schema

Example

```
const mongoose = require('mongoose’);
mongoose.connect('mongodb://127.0.0.1:2701 7/test’);
const Cat = mongoose.model('Cat', { name: String });
const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

Mongoose Schema vs. Model

- A Mongoose model is a wrapper on the Mongoose schema.
- A Mongoose schema defines the structure of the document, default values, validators, etc., whereas a Mongoose model provides an interface to the database for creating, querying, updating, deleting records, etc.

Defining and creating models

- Schemas are then "compiled" into models using the mongoose.model() method.
- Once you have a model you can use it to find, create, update, and delete objects of the given type

Defining schemas

- A schema can have an arbitrary number of fields
  - each one represents a field in the documents stored in MongoDB.

```
//Require Mongoose var mongoose = require('mongoose’);
//Define a schema var Schema = mongoose.Schema;
var SomeModelSchema = new Schema({ a_string: String,
a_date: Date });
```

Creating a model

- Models are created from schemas using the mongoose.model() method:
- // Compile model from schema
- var SomeModel = mongoose.model(‘myModel', SomeModelSchema);
  - The first argument is the singular name of the collection that will be created for your model (Mongoose will create the database collection for the above model SomeModel above),
  - and the second argument is the schema you want to use in creating the model.

Execute a Mongoose query

- There are two ways to execute a Mongoose query.
- You can either pass in a callback function or use the then() method to execute the query as a promise.

Creating and modifying documents

- To create a record, you can define an instance of the model and then call save()

```
// Create an instance of model SomeModel

var awesome_instance = new SomeModel({ name: 'awesome' });

// Save the new model instance, passing a callback
awesome_instance.save(function (err) { if (err) return handleError(err);

// saved!

});
```

Creating and modifying documents

- Creation of records (along with updates, deletes, and queries) are asynchronous operations
  - you supply a callback that is called when the operation completes

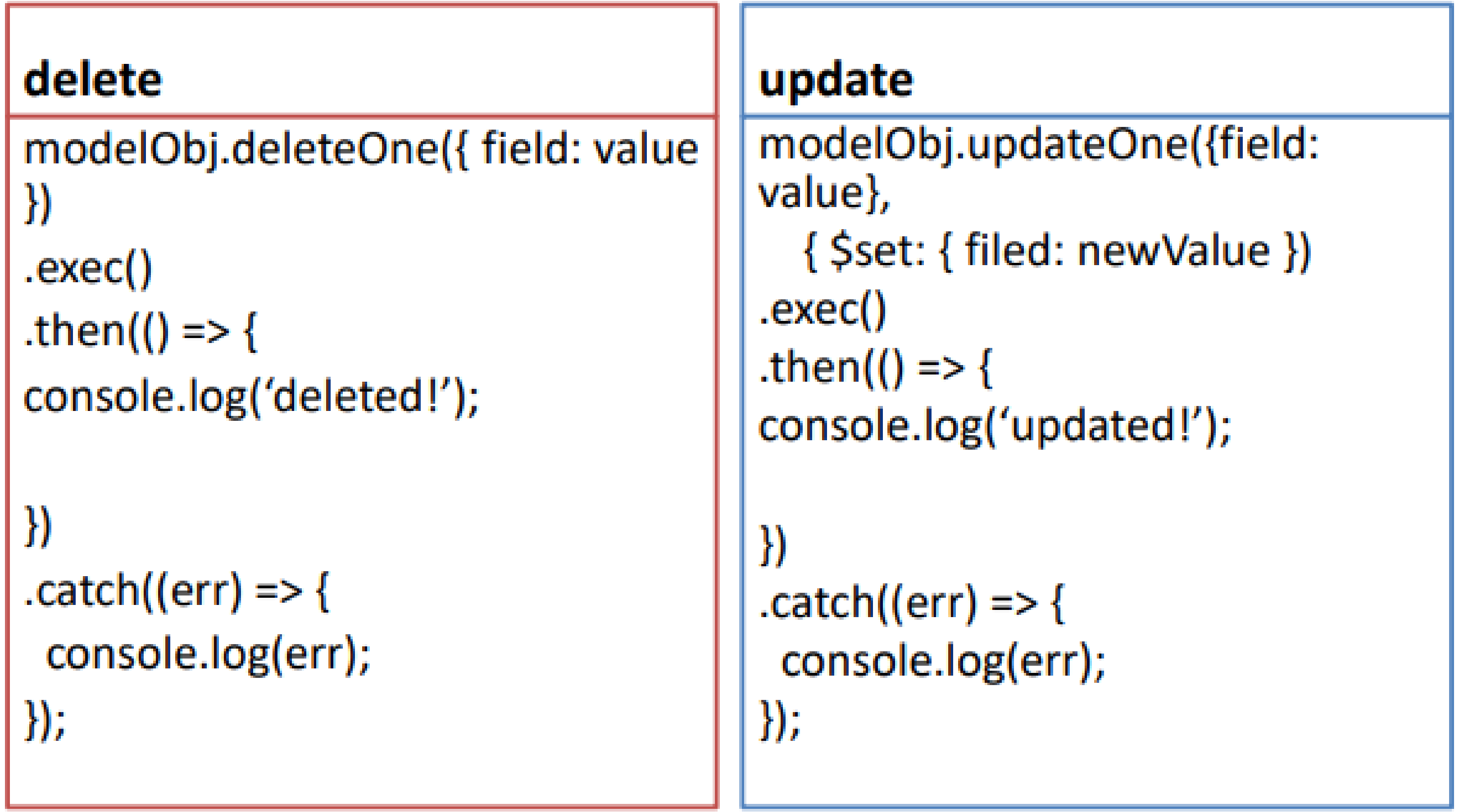
How to find records

- Every model method that accepts query conditions can be executed by means of a callback or the exec method.
  - callback: User.findOne({ name: ‘xyz' }, function (err, user) { });
  - exec() User.findOne({ name: ‘xyz' }).exec(function (err, user) { });
- Mongoose queries are not Promises. Queries do return a thenable, but if you need a real Promise you should use the exec method.
  - More info: https://mongoosejs.com/docs/promises.html

How to insert/update/delete record?

- For insert,
  - Call save() to store modified values back to the database.
  - newDocument.save((err) => { if(err) { console.log(“error"); } else { console.log(“inserted"); } });

Delete/Update in mongoose



delete

```
modelObj.deleteOne({ field: value
})
.exec()
.then(() => {
console.log('deleted!');

})
.catch((err) => {
  console.log(err);
});
```

update

```
modelObj.updateOne({field:
value},
   { $set: { filed: newValue })
.exec()
.then(() => {
console.log('updated!');

})
.catch((err) => {
  console.log(err);
});
```

Activity cont

- Implement deleteOne and updateOne feature to the previous activity to
  - Delete the existing record
  - Update the existing record

Lets work on the Practical

- Document Available on BB