CS-546 Lab 4

MongoDB

For this lab, we are going to make a few parts of a movie database. You will create the first of these data modules, the module to handle a listing of movies.

You will:

- Separate concerns into different modules.
- Store the database connection in one module.
- Define and store the database collections in another module.
- Define your Data manipulation functions in another module.
- Continue practicing the usage of `async` / `await` for asynchronous code
- Continuing our exercises of linking these modules together as needed

Packages you will use:

You will use the mongodb package to hook into MongoDB

You **may** use the lecture 4 code as a guide.

You must save all dependencies you use to your package.json file

How to handle bad data

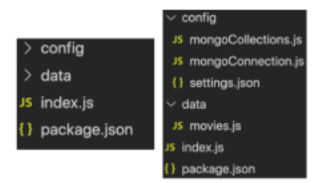
```
async function divideAsync(numerator, denominator) {
    if (typeof numerator !== "number") throw new Error("Numerator needs to be a number");
    if (typeof denominator !== "number") throw new Error("Denominator needs to be a number");

    if (denominator === 0) throw new Error("Cannot divide by 0!");

    return numerator / denominator;
}

async function main() {
    const six = await divideAsync(12, 2);
    console.log(six);

    try {
        const getAnError = await divideAsync("foo", 2);
    } catch(e) {
        console.log("Got an error!");
        console.log(e);
    }

}

main();
```

Would log:

```
6
"Numerator needs to be a number"
```

Folder Structure

You will use the following folder structure for the data module and other project files. You can use mongoConnection.js, mongoCollections.js and settings.json from the lecture code, however, you will need to modify settings.json and mongoCollections.js to meet this assignment's requirements.



Database Structure

You will use a database with the following structure:

- The database will be called **FirstName_LastName_lab4**
- The collection you use will be called `movies`

movies

The schema for a movie is as followed:

```
{
    _id: ObjectId,
    title: string,
    plot: string,
    rating: string,
    runtime: string,
    genre: string,
    cast: [strings],
    info: {director: string, yearReleased: number}
}
```

The `_id` field will be automatically generated by MongoDB when a movie is inserted, so you do not need to provide it when a movie is created.

An example of how Bill and Ted Face the Music would be stored in the DB:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    title: "Bill and Ted Face the Music",
    plot: "Once told they'd save the universe during a time-traveling adventure, 2 would-be rockers from San Dim
as, California find themselves as middle-aged dads still trying to crank out a hit song and fulfill their destin
y.",
    rating: "PG-13",
    runtime: "1hr 31min",
    genre: "Comedy",
    cast: ["Keanu Reeves","Alex Winter"],
    info: {director: "Dean Parisot", yearReleased: 2020}
}
```

movies.js

In movies.js, you will create and export five methods:

Remember, you must export methods named precisely as specified. The async keyword denotes that this is an async method.

async create(title, plot, rating, runtime, genre, cast, info);

This async function will return to the newly created movie object, with **all** of the properties listed above.

If `title, plot, rating, runtime, genre, cast, info` are not provided at all, the method should throw. (**All fields need to have valid values**);

If `title, plot, rating, runtime, genre` are not `strings` or are empty strings, the method should throw.

If `cast` is not an array and if it does not have at least one element in it that is a valid `string`, or are empty strings the method should throw.

If `info` is not an `object`, the method should throw.

If `info.director` is not a valid `string` or if `info.director` is not provided or is an empty string, the method should throw.

If `info.yearReleased` is not a 4 digit `number` or if `info.yearReleased` is not provided, the method should throw.

If `info.yearReleased` is less than 1930 or greater than the current year + 5 years, the method should throw.

Note: FOR ALL INPUTS: Strings with empty spaces are NOT valid strings. So no cases of "   " are valid.

For example:

```
const movies = require("./movies");

async function main() {
    const billAndTed = await movies.create("Bill and Ted Face the Music","Once told they'd save the universe dur
ing a time-traveling adventure, 2 would-be rockers from San Dimas, California find themselves as middle-aged dad
s still trying to crank out a hit song and fulfill their destiny.","PG-13", "1hr 31min","Comedy",["Keanu Reeve
s","Alex Winter"],{director: "Dean Parisot", yearReleased: 2020});
    console.log(billAndTed);
}

main();
```

Would return and log:

```
{
    _id: "507f1f77bcf86cd799439011",
    title: "Bill and Ted Face the Music",
    plot: "Once told they'd save the universe during a time-traveling adventure, 2 would-be rockers from San Dim
as, California find themselves as middle-aged dads still trying to crank out a hit song and fulfill their destin
y.",
    rating: "PG-13",
    runtime: "1hr 31min",
    genre: "Comedy",
    cast: ["Keanu Reeves","Alex Winter"],
    info: {director: "Dean Parisot", yearReleased: 2020}
}
```

This movie will be stored in the **movies** collection.

If the movie cannot be created, the method should throw.

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, you need to display it as such.

async getAll();

This function will return an array of all movies in the collection. **If there are no movies in your DB, this function will return an empty array**

```
const movies = require("./movies");

async function main() {
    const allMovies = await movies.getAll();
    console.log(allMovies);
}

main();
```

Would return and log all the movies in the database.

```
[{
    _id: "507f1f77bcf86cd799439011",
    title: "Bill and Ted Face the Music",
    plot: "Once told they'd save the universe during a time-traveling adventure, 2 would-be rockers from San Dim
as, California find themselves as middle-aged dads still trying to crank out a hit song and fulfill their destin
y.",
    rating: "PG-13",
    runtime: "1hr 31min",
    genre: "Comedy",
    cast: ["Keanu Reeves","Alex Winter"],
    info: {director: "Dean Parisot", yearReleased: 2020}
},
  {
    _id: "407f1f77bcf86cd7738930011",
    title: "The Dark Knight",
    plot: "When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, Batman must accept
one of the greatest psychological and physical tests of his ability to fight injustice.",
    rating: "R",
    runtime: "2hr 32min",
    genre: "Action",
    cast: ["Christian Bale","Heath Ledger"],
    info: {director: "Christopher Nolan", yearReleased: 2008}
 },
  {
    _id: "507f1f77bcf86cd77380011",
    title: "Hidden Figures",
    plot: "The story of a team of female African-American mathematicians who served a vital role in NASA during
the early years of the U.S. space program.",
    rating: "PG",
    runtime: "2hr 7min",
    genre: "Drama",
    cast: ["Taraji P. Henson","Octavia Spencer", "Janelle Monáe"],
    info: {director: "Theodore Melfi", yearReleased: 2016}
 }
]
```

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, you need to display it as such.

async get(id);

When given an id, this function will return an movie from the database.

If no `id` is provided, the method should throw.

If the `id` provided is not a `string`, or is an empty string, the method should throw.

If the `id` provided is not a valid `ObjectId`, the method should throw

If the no movie exists with that `id`, the method should throw.

For example, you would use this method as:

```
const movies = require("./movies");

async function main() {
    const billAndTed = await movies.get("507f1f77bcf86cd799439011");
    console.log(billAndTed);
}

main();
```

Would return and log Bill and Ted:

```
{
    _id: "507f1f77bcf86cd799439011",
    title: "Bill and Ted Face the Music",
    plot: "Once told they'd save the universe during a time-traveling adventure, 2 would-be rockers from San Dim
as, California find themselves as middle-aged dads still trying to crank out a hit song and fulfill their destin
y.",
   rating: "PG-13",
   runtime: "1hr 31min",
   genre: "Comedy",
   cast: ["Keanu Reeves","Alex Winter"],
   info: {director: "Dean Parisot", yearReleased: 2020}
```

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, you need to display it as such.

Important note: The ID field that MongoDB generates is an `ObjectId`. This function takes in a `string` representation of an ID as the `id` parameter. You will need to convert it to an `ObjectId` in your function before you query the DB for the provided ID and then pass that converted value to your query.

```
//We need to require ObjectId from mongo
let { ObjectId } = require('mongodb');


/*For demo purposes, I will create a new object ID and convert it to a string,
Then will pass that valid object ID string value into my function to check if it's a valid Object ID (which it i
s in this case)
I also pass in an invalid object ID when I call my function to show the error */


let newObjId = ObjectId(); //creates a new object ID


let x = newObjId.toString(); // converts the Object ID to string


console.log(typeof x); //just logging x to see it's now type string


//The below function takes in a string value and then attempts to convert it to an ObjectId


function myDBfunction(id) {


  //check to make sure we have input at all
  if (!id) throw 'Id parameter must be supplied';


  //check to make sure it's a string
  if (typeof id !== 'string') throw "Id must be a string";


  //Now we check if it's a valid ObjectId so we attempt to convert a value to a valid object ID,
  //if it fails, it will throw an error (you do not have to throw the error, it does it automatically and the ca
tch where you call the function will catch the error just as it catches your other errors).

  let parsedId = ObjectId(id);
  //this console.log will not get executed if Object(id) fails, as it will throw an error
  console.log('Parsed it correctly, now I can pass parsedId into my query.');
}

//passing a valid string that can convert to an Object ID
try {
  myDBfunction(x);
} catch (e) {
  console.log(e.message);
}

//passing an invalid string that can't be converted into an object ID:
try {
  myDBfunction('test');
} catch (e) {
  console.log(e.message);
}
```

async remove(id)

This function will remove the movie from the database.

If no `id` is provided, the method should throw.

If the `id` provided is not a `string`, or is an empty string the method should throw.

If the `id` provided is not a valid `ObjectId`, the method should throw

If the movie cannot be removed (does not exist), the method should throw.

If the removal succeeds, return the name of the movie and the text " has been successfully deleted"

```
const movies = require("./movies");

async function main() {
    const removeBillAndTed = await movies.remove("507f1f77bcf86cd799439011");
    console.log(removeBillAndTed);
}

main();
```

Would return and then log: "Bill and Ted Face the Music has been successfully deleted".

Important note: The ID field that MongoDB generates is an `ObjectId`. This function takes in a `string` representation of an ID as the `id` parameter. You will need to convert it to an `ObjectId` in your function before you query the DB for the provided ID. See example above in `getId()`.

async rename(id, newTitle)

This function will update the title of a movie currently in the database.

If no `id` is provided, the method should throw.

If the `id` provided is not a `string`, or is an empty string the method should throw.

If the `id` provided is not a valid `ObjectId`, the method should throw.

If `newTitle` is not provided, the method should throw.

If `newTitle` is not a `string`, or is an empty string, the method should throw.

If the movie cannot be updated (does not exist), the method should throw.

If the update succeeds, return the entire movie object as it is after it is updated.

```
const movies = require("./movies");

async function main() {
    const renamedBillAndTed = await movies.rename("507f1f77bcf86cd799439011", "Patrick and Ted Face the Music");
    console.log(renamedBillAndTed);
}

main();
```

Would log the updated movie:

```
{
    _id: "507f1f77bcf86cd799439011",
    title: "Patrick and Ted Face the Music",
    plot: "Once told they'd save the universe during a time-traveling adventure, 2 would-be rockers from San Dim
as, California find themselves as middle-aged dads still trying to crank out a hit song and fulfill their destin
y.",
    rating: "PG-13",
    runtime: "1hr 31min",
    genre: "Comedy",
    cast: ["Keanu Reeves","Alex Winter"],
    info: {director: "Dean Parisot", yearReleased: 2020}
```

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, you need to display it as such.

Important note: The ID field that MongoDB generates is an `ObjectId`. This function takes in a `string` representation of an ID as the `id` parameter. You will need to convert it to an `ObjectId` in your function before you query the DB for the provided ID. See example above in `getId()`.

index.js

For your index.js file, you will:

1. Create a movie of your choice.
2. Log the newly created movie. (Just that movie, not all movies)
3. Create another movie of your choice.
4. Query all movies, and log them all
5. Create a 3rd movie of your choice.
6. Log the newly created 3rd movie. (Just that movie, not all movies)
7. Rename the first movie's title
8. Log the first movie with the updated title.
9. Remove the second movie you created.
10. Query all movies, and log them all
11. Try to create a movie with bad input parameters to make sure it throws errors.
12. Try to remove a movie that does not exist to make sure it throws errors.
13. Try to rename a movie that does not exist to make sure it throws errors.
14. Try to rename a movie passing in invalid data for the parameter to make sure it throws errors.
15. Try getting a movie by ID that does not exist to make sure it throws errors.

General Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder
3. You must do basic error checking in each function
4. Check for arguments existing and of proper type.
5. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
6. If a function should return a promise, you should mark the method as an `async` function and return the value. Any promises you use inside of that, you should *await* to get their result values. If the promise should reject, then you should throw inside of that promise in order to return a rejected promise automatically. Thrown exceptions will bubble up from any awaited call that throws as well, unless they are caught in the async method.
7. You **must remember** to update your package.json file to set `index.js` as your starting script!
8. You **must** submit a zip file named in the following format: `LastName_FirstName_CS546_SECTION.zip`

◂ Previous

Next ▸