Lab 4

Start Assignment

Due Monday by 11:59pm **Points** 100 **Submitting** a file upload

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 (https://mongodb.github.io/node-mongodb-native/)** package to hook into MongoDB

You **may** use the **lecture 4 code (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_04/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
"Got an error!"
"Numerator needs to be a number"
```

Folder Structure

You will use the folder structure in the stub. 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. There is an extra file in the stub called helpers.js. You can add all your helper/validation functions in that file to use in your other modules.

YOU MUST use the directory and file structure in the code stub or points will be deducted. You can download the starter template here: **lab4_stub.zip** (https://sit.instructure.com/courses/61513/files/10338004/download?download_frd=1) **PLEASE NOTE: THE STUB DOES NOT INCLUDE THE PACKAGE.JSON FILE. YOU WILL NEED TO CREATE IT! DO NOT ADD ANY OTHER FILE OR FOLDER APART FROM PACKAGE.JSON FILE.**

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,
    genres: [strings],
    rating: string,
    studio: string,
    director: string,
    castMembers: [strings],
    dateReleased: string representation of a date
    runtime: string

}
```

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 Hackers would be stored in the DB:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    title: "Hackers",
    plot: "Hackers are blamed for making a virus that will capsize five oil tankers.",
    genres: ["Crime", "Drama", "Romance"],
    rating: "PG-13",
    studio: "United Artists",
    director: "Iain Softley",
    castMembers: ["Jonny Miller", "Angelina Jolie", "Matthew Lillard", "Fisher Stevens"],
    dateReleased: "09/15/1995",
    runtime: "1h 45min"
}
```

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 createMovie(title, plot, genres, rating, studio, director, castMembers, dateReleased, runtime);

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

If `title, plot, genres, rating, studio, director, castMembers, dateReleased, runtime` are not provided at all, the method should throw. (**All fields need to have valid values**);

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

`title` must be at least two characters and can contain letters a-z, A-Z or numbers. i.e. the movie "42" about Jackie Robinson. No special characters or punctuation.

`studio` must be at least 5 characters long and only letters a-z or A-Z. No numbers or special characters.

`director` must have the following format "first name space last name" i.e "Patrick Hill". first name and last name must be at least 3 characters each and only letters a-z or A-Z. No numbers or special characters or punctuation.

If `rating` is not one of the following values, this method will throw. Valid values (case sensitive): `G, PG, PG-13, R, NC-17`

If `genres` is not an array that has at least one string element contained in it, this method will throw.

if any of the elements in `genres` is not a valid string (empty strings or strings with just spaces are invalid), the method should throw.

Each element in `genres` must be at least five characters long and only letters a-z or A-Z. No numbers or special characters or punctuation.

If `castMembers` is not an array that has at least one string element contained in it, this method will throw.

if any of the elements in `castMembers` is not a valid string (empty strings or strings with just spaces are invalid), the method should throw.

Each element in `castMembers` must have the following format "first name space last name" i.e "Patrick Hill". first name and last name must be at least 3 characters each and only letters a-z or A-Z. No numbers or special characters.

If `dateReleased` is not a valid date string (09/31/2019 is not valid as there are not 31 days in September. 02/30/2020 is not valid as there are not 30 days in February this field MUST be a valid date), or if `dateReleased` is less than 01/01/1900 or greater than the current year + 2 (2024 in this case) the the method should throw. You do not have to take leap years into account and the format of the date must be in mm/dd/yyyy format.
Note: so only years 1900-2024 are valid values. Do not hardcode the year to be 2024, use the current year and then add 2 years to it. This will ensure that the application will function past 2024).

`runtime` MUST be in the following format "#h #min". both #'s must be a positive whole number, but for minutes, that may be zero but the max value for min should be 59 since 60min would be 1 hour:

For example: valid: "2h 30min", "2h 0min", "1h 59min" not valid: "-5h 20min", "3.5h 10min", "0h 30min" (most movies are longer than an 30 min, and usually longer than 1 hour), "2h 60min" (this should just be 3 hours). This field will be case sensitive so you MUST match the format shown exactly.

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 hackers = await movies.createMovie("Hackers", "Hackers are blamed for making a virus that will caps
ize five oil tankers.", ["Crime", "Drama", "Romance"], "PG-13", "United Artists", "Iain Softley", ["Jonny Mil
ler", "Angelina Jolie", "Matthew Lillard", "Fisher Stevens"], "09/15/1995", "1h 45min");
    console.log(hackers);
}
```

```
main();
```

Would return and log:

```
{
    _id: "507f1f77bcf86cd799439011",
    title: "Hackers",
    plot: "Hackers are blamed for making a virus that will capsize five oil tankers.",
    genres: ["Crime", "Drama", "Romance"],
    rating: "PG-13",
    studio: "United Artists",
    director: "Iain Softley",
    castMembers: ["Jonny Miller", "Angelina Jolie", "Matthew Lillard", "Fisher Stevens"],
    dateReleased: "09/15/1995",
    runtime: "1h 45min"
}
```

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, your function needs to return it as shown.

async getAllMovies();

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.getAllMovies();
    console.log(allMovies);
}

main();
```

Would return and log all the movies in the database.

```
[{
    _id: "507f1f77bcf86cd799439011",
    title: "Hackers",
    plot: "Hackers are blamed for making a virus that will capsize five oil tankers.",
    genres: ["Crime", "Drama", "Romance"],
    rating: "PG-13",
    studio: "United Artists",
    director: "Iain Softley",
    castMembers: ["Jonny Miller", "Angelina Jolie", "Matthew Lillard", "Fisher Stevens"],
    dateReleased: "09/15/1995",
    runtime: "1h 45min"
},
{
    _id: "507f1f77bcf86cd799439012",
    title: "42",
    plot: "In 1947, Jackie Robinson becomes the first African-American to play in Major League Baseball in th
e modern era when he was signed by the Brooklyn Dodgers and faces considerable racism in the process.",
    genres: ["Biography", "Drama", "Sport"],
    rating: "PG-13",
    studio: "Warner Brothers",
    director: "Brian Helgeland",
    castMembers: ["Chadwick Boseman", "Harrison Ford", "Nicole Beharie", "Christopher Meloni"],
    dateReleased: "04/09/2013",
    runtime: "2h 8min"
},
{
    _id: "507f1f77bcf86cd799439013",
    title: "The Breakfast Club",
    plot: "Five high school students meet in Saturday detention and discover how they have a lot more in comm
on than they thought.",
    genres: ["Comedy", "Drama"],
    rating: "R",
    studio: "Universal Pictures",
    director: "John Hughes",
    castMembers: ["Judd Nelson", "Molly Ringwald", "Ally Sheedy", "Anthony Hall", "Emilio Estevez"],
    dateReleased: "02/07/1985",
    runtime: "1h 37min"
}
]
```

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, your function needs to return it as shown.

async getMovieById(id);

When given an id, this function will return a 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 theBreakfastClub = await movies.getMovieById("507f1f77bcf86cd799439013");
 console.log(theBreakfastClub);
}

main();
```

Would return and log The Breakfast Club:

```
{
    _id: "507f1f77bcf86cd799439013",
    title: "The Breakfast Club",
    plot: "Five high school students meet in Saturday detention and discover how they have a lot more in comm
on than they thought.",
    genres: ["Comedy", "Drama"],
    rating: "R",
    studio: "Universal Pictures",
    director: "John Hughes",
    castMembers: ["Judd Nelson", "Molly Ringwald", "Ally Sheedy", "Anthony Hall", "Emilio Estevez"],
    dateReleased: "02/07/1985",
    runtime: "1h 37min"
}
```

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, your function needs to return it as shown.

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 i
t is 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, you will throw an error
  if (!ObjectId.isValid(id)) throw "ID is not a valid Object ID";

  console.log('Valid Object ID, now I can pass ObjectId(id) as the ID 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 removeMovie(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 removeFortyTwo = await movies.removeMovie("507f1f77bcf86cd799439012");
 console.log(removeFortyTwo);
}
main();
```

Would return and then log: "42 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 `getById()`.

async renameMovie(id, newName)

This function will update the name of the 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 `newName` is not provided, the method should throw.

If `newName` is not a `string`, or an empty string, the method should throw. The same constraints apply for `newName` as they do for `title` in the createMovie function.

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

if the `newName` is the same as the current value stored in the database, 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 renamedFortyTwo = await movies.renameMovie("507f1f77bcf86cd799439012", "Forty Two");
 console.log(renamedFortyTwo);
}
main();
```

Would return and log the updated movie:

```
{
    _id: "507f1f77bcf86cd799439012",
    title: "Forty Two",
    plot: "In 1947, Jackie Robinson becomes the first African-American to play in Major League Baseball in th
e modern era when he was signed by the Brooklyn Dodgers and faces considerable racism in the process.",
    genres: ["Biography", "Drama", "Sport"],
    rating: "PG-13",
    studio: "Warner Brothers",
    director: "Brian Helgeland",
    castMembers: ["Chadwick Boseman", "Harrison Ford", "Nicole Beharie", "Christopher Meloni"],
    dateReleased: "04/09/2013",
    runtime: "2h 8min"
}
```

Notice the output does not have ObjectId() around the ID field and no quotes around the key names, your function needs to return it as shown.

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 `getById()`.

app.js

For your app.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 the 3rd movie of your choice.
6. Log the newly created 3rd movie. (Just that movie, not all movies)
7. Rename the first movie
8. Log the first movie with the updated name.
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 `newName` 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 or package-lock.json
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 `app.js` as your starting script!
8. You **must** submit a zip file named in the following format: `LastName_FirstName_CS546_SECTION.zip (ie. Hill_Patrick_CS546_WS.zip)`