CS-546 Lab 4

Animal Farm

For this lab, we are going to make a few parts of the database for a social media website for animals. You will create a the first of these data modules, the module to handle animal listings.

* Seperating concerns into different modules:
* Database connection in one module
* Collections defined in another
* Data manipulation in another
* 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 (Links to an external site.)](https://mongodb.github.io/node-mongodb-native/) package to hook into MongoDB

You **may** use the [lecture 4 code (Links to an external site.)](https://github.com/Stevens-CS546/CS-546-WS-Summer-1/tree/master/Lecture%20Code/lecture_04) 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. **You will need other files to handle the connection to the database as well.**

./

../data

../data/animals.js

../index.js

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 animals

animals

The schema for animals is as followed:

{

\_id: "",

name: "",

animalType: ""

}

The \*\*\_id\*\* field will be automatically generated by MongoDB, so you do not need to provide it.

An example of this, for Mortimer the Giraffe:

{

\_id: "507f1f77bcf86cd799439011",

name: "Mortimer",

animalType: "Giraffe"

}

And for Howl the Worm:

{

\_id: "5324fbb60f664dc38e540408cfb0d64a",

name: "Howl",

animalType: "Worm"

}

And for Blub Blub the Otter:

{

\_id: "5324fbb60f664dc38e540408cfb0d64a",

name: "Blub Blub",

animalType: "Otter"

}

And finally, Jerry the Giraffe.

{

\_id: "5b0b614d46a8445083b965b88005e4cc",

name: "Jerry",

animalType: "Giraffe"

}

animals.js

In animals.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(name, animalType);

This async function will return to the newly created animal object, with **all** of the properties listed above. For example:

const animals = require("./animals");

async function main() {

const mortimer = await animals.create("Mortimer", "Giraffe");

console.log(mortimer);

}

main();

Would output:

{

\_id: "507f1f77bcf86cd799439011",

name: "Mortimer",

animalType: "Giraffe"

}

This animal will be stored in the **animals** collection.

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

async getAll();

This function will return an array of all animals in the collection.

const animals = require("./animals");

async function main() {

const allMyAnimals = await animals.getAll();

console.log(allMyAnimals);

}

main();

Would log all the animals in the database.

async get(id);

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

If no id is provided, the method should throw.

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

For example, you would use this method as:

const animals = require("./animals");

async function main() {

const blubBlub = await animals.get("5324fbb60f664dc38e540408cfb0d64a");

console.log(blubBlub);

const noMatch = await animals.get("BADID");

console.log(noMatch)

}

main();

Would log Blub Blub the Otter, and then would error out.

async remove(id)

This function will remove the animal from the database.

If no id is provided, the method should throw.

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

If the removal succeeds, return the animal as it was before it was deleted.

const animals = require("./animals");

async function main() {

const removeBlubBlub = await animals.remove("5324fbb60f664dc38e540408cfb0d64a");

console.log(blubBlub);

}

main();

Would log the data of Blub Blub before Blub Blub goes Bye-Bye.

async rename(id, newName)

This function will update the name of an animal currently in the database.

If no id is provided, the method should throw.

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

If the update succeeds, return the animal as it is after it is updated.

const animals = require("./animals");

async function main() {

const bubba = await animals. rename("5324fbb60f664dc38e540408cfb0d64a", "Bubba");

console.log(bubba);

}

main();

Would log our newly named Otter, Bubba:

{

\_id: "5324fbb60f664dc38e540408cfb0d64a",

name: "Bubba",

animalType: "Otter"

}

index.js

For your index.js file, you will:

1. Create an animal named Sasha with the type of Dog
2. Log the newly created animal
3. Create an animal named Lucy, with the type of Dog
4. Query all animals, and log them all
5. Create an animal named Duke, with a type of Walrus
6. Log the newly created Duke
7. Rename Sasha to Sashita
8. Log the newly named Sashita
9. Remove Lucy
10. Query all animals, and log them all

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