Lab 4

New Attempt

Due Oct 7 by 11:59pm **Points** 100 **Submitting** a file upload
Available after Sep 30 at 12am

CS-546 Lab 4

MongoDB

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

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() {
```

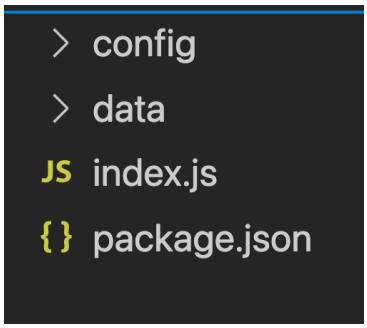
```
    const six = await divideAsync(12, 2);
    console.log(six);

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

}

main();
```

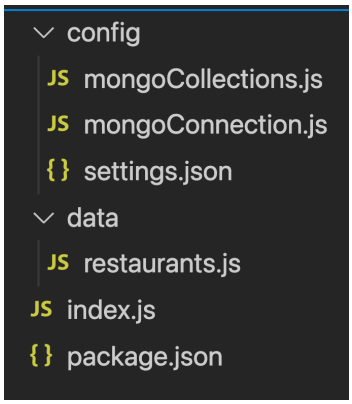
Would log:

```
6
"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 `restaurants`

restaurants

The schema for a restaurant is as followed:

```
{
    _id: ObjectId,
    name: string,
    location: string,
    phoneNumber: string (xxx-xxx-xxxx format),
    website: string (must contain full url http://www.patrickseats.com),
```

```
    priceRange: string (from $ to $$$$),
    cuisines: [strings],
    overallRating: number (from 0 to 5),
    serviceOptions: {dineIn: Boolean, takeOut: Boolean, delivery: Boolean}
}
```

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

An example of how The Saffron Lounge would be stored in the DB:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    name: "The Saffron Lounge",
    location: "New York City, New York",
    phoneNumber: "123-456-7890",
    website: "http://www.saffronlounge.com",
    priceRange: "$$$$",
    cuisines: ["Cuban", "Italian" ],
    overallRating: 3
    serviceOptions: {dineIn: true, takeOut: true, delivery: false}
}
```

restaurants.js

In restaurant.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, location, phoneNumber, website, priceRange, cuisines, overallRating, serviceOptions);

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

If `name, location, phoneNumber, website, priceRange, cuisines, overallRating, serviceOptions` are not provided at all, the method should throw. (**All fields need to have valid values**);

If `name, location, phoneNumber, website, priceRange` are not `strings` or are empty strings, the method should throw.

If `phoneNumber` does not follow this format: xxx-xxx-xxxx, the method will throw.

If `website` does not contain `http://www.` and end in a `.com`, and have at least 5 characters in-between the `http://www.` and `.com` this method will throw.

If `priceRange` is not between '`$`' to '`$$$$`', this method will throw.

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

If `serviceOptions.dineIn, serviceOptions.takeOut, serviceOptions.delivery` is not a boolean, 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 restaurants = require("./restaurants");

async function main() {
    const safrronLounge = await restaurants.create("The Saffron Lounge", "New York City, New York",
 "123-456-7890", "http://www.saffronlounge.com", "$$$$", ["Cuban", "Italian"], 3, {dineIn: true, tak
eOut: true, delivery: false});
    console.log(safrronLounge);
}

main();
```

Would return and log:

```
{
    _id: "507f1f77bcf86cd799439011",
    name: "The Saffron Lounge",
    location: "New York City, New York",
    phoneNumber: "123-456-7890",
    website: "http://www.saffronlounge.com",
    priceRange: "$$$$",
    cuisines: ["Cuban", "Italian" ],
    overallRating: 3
    serviceOptions: {dineIn: true, takeOut: true, delivery: false}
}
```

This restaurant will be stored in the **restaurants** collection.

If the restaurant 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 restaurants in the collection. **If there are no restaurants in your DB, this function will return an empty array**

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

async function main() {
    const allResturants = await restaurants.getAll();
    console.log(allResturants);
}

main();
```

Would return and log all the restaurants in the database.

```
[{
    _id: "507f1f77bcf86cd799439011",
    name: "The Saffron Lounge",
    location: "New York City, New York",
    phoneNumber: "123-456-7890",
    website: "http://www.saffronlounge.com",
    priceRange: "$$$$",
    cuisines: ["Cuban", "Italian" ],
    overallRating: 3
    serviceOptions: {dineIn: true, takeOut: true, delivery: false}
},
{
    _id: "306f1f77bcf86cd799431423",
    name: "Black Bear",
    location: "Hoboken, New Jersey",
    phoneNumber: "456-789-0123",
    website: "http://www.blackbear.com",
    priceRange: "$$",
    cuisines: ["Cuban", "American" ],
    overallRating: 4
    serviceOptions: {dineIn: true, takeOut: true, delivery: true}
},
{
    _id: "204adw77bcf86cd799439011",
    name: "Pizza Lounge",
    location: "New York City, New York",
    phoneNumber: "999-999-9999",
    website: "http://www.pizzalounge.com",
    priceRange: "$",
    cuisines: ["Italian" ],
    overallRating: 5
    serviceOptions: {dineIn: false, takeOut: true, delivery: true}
}
]
```

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 a restaurant 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 restaurant exists with that `id`, the method should throw.

For example, you would use this method as:

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

async function main() {
    const pizzaLounge = await restaurants.get("204adw77bcf86cd799439011");
 console.log(pizzaLounge);
}
```

```
main();
```

Would return and log Pizza Lounge:

```
{
    _id: "204adw77bcf86cd799439011",
    name: "Pizza Lounge",
    location: "New York City, New York",
    phoneNumber: "999-999-9999",
    website: "http://www.pizzalounge.com",
    priceRange: "$",
    cuisines: ["Italian" ],
    overallRating: 5
    serviceOptions: {dineIn: false, takeOut: true, delivery: true}
}
```

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 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, it will throw an error (you do not have to throw the error, it does it automaticall
y and the catch where you call the function will catch the error just as it catches your other error
s).
```

```
  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 restaurant 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 restaurant cannot be removed (does not exist), the method should throw.

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

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

async function main() {
    const removeBlackBear = await resturants.remove("306f1f77bcf86cd799431423");
 console.log(removeBlackBear);
}
main();
```

Would return and then log: "Black Bear 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, newWebsite)

This function will update the website of the restaurant 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 `newWebsite` is not provided, the method should throw.

If `newWebsite` is not a `string`, is an empty string, or does not contain `http://www.` and end in a `.com`, and have at least 5 characters in-between the `http://www.` and `.com`, the method should throw.

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

if the `newWebsite` is the same as the current value stored in the database, the method should throw.

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

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

async function main() {
    const renamedSaffronLounge = await resturants.rename("507f1f77bcf86cd799439011", "http://www.the
saffronlounge.com");
 console.log(renamedSaffronLounge);
}
main();
```

Would log the updated restaurant:

```
{
    _id: "507f1f77bcf86cd799439011",
    name: "The Saffron Lounge",
    location: "New York City, New York",
    phoneNumber: "123-456-7890",
    website: "http://www.thesaffronlounge.com",
    priceRange: "$$$$",
    cuisines: ["Cuban", "Italian" ],
    overallRating: 3
    serviceOptions: {dineIn: true, takeOut: true, delivery: false}
}
```

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 restaurant of your choice.
2. Log the newly created restaurant. (Just that restaurant, not all restaurants)
3. Create another restaurant of your choice.
4. Query all restaurants, and log them all
5. Create the 3rd restaurant of your choice.

6. Log the newly created 3rd restaurant. (Just that restaurant, not all restaurants)
7. Rename the first restaurant website
8. Log the first restaurant with the updated website.
9. Remove the second restaurant you created.
10. Query all restaurants, and log them all
11. Try to create a restaurant with bad input parameters to make sure it throws errors.
12. Try to remove a restaurant that does not exist to make sure it throws errors.
13. Try to rename a restaurant that does not exist to make sure it throws errors.
14. Try to rename a restaurant passing in invalid data for the parameter to make sure it throws errors.
15. Try getting a restaurant 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 (ie. Hill_Patrick_CS546_WS.zip)`