Lab 4

- Due Feb 27 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip
- Available until Mar 5 at 11:59pm

This assignment was locked Mar 5 at 11:59pm.

CS-546 Lab 4

MongoDB

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

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/)** 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 "Numerator needs to be a number";
    if (typeof denominator !== "number") throw "Denominator needs to be a number";

    if (denominator === 0) throw "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.js 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/71954/files/13066967?wrap=1)** **(https://sit.instructure.com/courses/71954/files/13066967/download?download_frd=1)** **PLEASE NOTE: THE STUB DOES NOT INCLUDE THE PACKAGE.JSON FILE. YOU WILL NEED TO CREATE IT! DO NOT FORGET TO ADD THE START COMMAND AND "type": "module". 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 `products`

Products

The schema for a Product is as followed:

```
{
    _id: ObjectId,
    productName: string,
    productDescription: string,
    modelNumber: string,
    price: number/float (9.99, 199.99, 59.00, 100, 5 etc),
    manufacturer: string,
    manufacturerWebsite: string (must contain full url http://www.phillips.com),
    keywords: [array, of, strings],
    categories: [array, of, strings],
```

```
    dateReleased: string date in mm/dd/yyyy format,
    discontinued: boolean

}
```

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

An example of how the 83 inch LG C3 OLED TV would be stored in the DB:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    productName: "83 inch LG C3 OLED TV",
    productDescription: "The advanced LG OLED evo C-Series is better than ever. The LG OLED evo C3 is pow
ered by the next-gen a9 AI Processor Gen6—exclusively made for LG OLED—for ultra-realistic picture and so
und. And the Brightness Booster improves brightness so you get luminous picture and high contrast, even i
n well-lit rooms.* AI-assisted deep learning analyzes what you're watching to choose the best picture and
sound setting for your content. The LG OLED evo C3 not only performs great, but looks great as well. With
an almost invisible bezel, it will blend into the background for a seamless look. When you're finished wa
tching, display paintings, photos and other content to blend the LG OLED evo C3 into your space even mor
e. But that's not all. Experience less searching and more streaming, thanks to the next generation of AI
technology from LG webOS 23. Every LG OLED comes loaded with Dolby Vision for extraordinary color, contra
st and brightness, plus Dolby Atmos** for wrap-around sound. And LG's FILMMAKER MODE allows you to see fi
lms just as the director intended. Packed with gaming features, the LG OLED evo C-Series comes with every
thing you need to win like a 0.1ms response time, native 120Hz refresh rate and four HDMI 2.1 inputs. *Ba
sed on LG internal testing: 55/65/77/83 LG OLED evo C3 models are brighter than non-OLED evo B3 models an
d excludes the 42 and 48 LG OLED evo C3. **Dolby, Dolby Atmos and the double-D symbol are registered trad
emarks of Dolby Laboratories.",
    modelNumber: "OLED83C3PUA",
    price: 4757.29,
    manufacturer:"LG",
    manufacturerWebsite: "http://www.lgelectronics.com",
    keywords: ["TV", "Smart TV", "OLED", "LG", "Big Screen", "83 Inch"],
    categories: ["Electronics", "Television & Video", "Televisions",  "OLED TVs"],
    dateReleased: "02/27/2023",
    discontinued: false
}
```

products.js

In products.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.

General note that applies to ALL functions that take in strings as input. You should trim all string inputs before storing them in the DB or querying against the DB. That includes the _id field. String inputs should almost ALWAYS be trimmed, there are some cases where you may not want to trim string inputs but almost always, we trim string inputs.

async create(productName, productDescription, modelNumber, price, manufacturer, manufacturerWebsite, keywords, categories, dateReleased, discontinued);

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

If `productName, productDescription, modelNumber, price, manufacturer, manufacturerWebsite, keywords, categories, dateReleased, discontinued` are not provided at all, the method should throw. (**All fields need to be supplied**);

If `productName, productDescription, modelNumber, manufacturer, manufacturerWebsite, dateReleased` are not `strings` or are empty strings, the method should throw.

If `price` is not a number greater than 0, whole numbers and decimals allowed (**only allow 2 decimal points for the cents, no more than two!**), the method should throw.

If `manufacturerWebsite` does not start with `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 `keywords, categories` are not arrays and if they do not have **at least** one element in each of them that is a valid `string`, or are empty strings the method should throw. (each element should be a valid string but the arrays should contain at LEAST one element that's a valid string.

If `dateReleased` is not a valid date in mm/dd/yyyy format then the method should throw. You will not have to take into account leap years but it must be a valid date. For example: 9/31/2022 would not be valid as there are not 31 days in September. 2/30/1995 would not be valid as there are never 30 days in Feb.

If `discontinued` is not a boolean, then the method should throw.

Note: FOR ALL INPUTS: Strings with empty spaces are NOT valid strings. So no cases of " " are valid. You should also trim all string inputs!

For example:

```
import * as products from "./products.js";

async function main() {
    const lgTV = await products.create("83 inch LG C3 OLED TV", "The advanced LG OLED evo C-Series is bet
ter than ever. The LG OLED evo C3 is powered by the next-gen a9 AI Processor Gen6—exclusively made for LG
OLED—for ultra-realistic picture and sound. And the Brightness Booster improves brightness so you get lum
inous picture and high contrast, even in well-lit rooms.* AI-assisted deep learning analyzes what you're
watching to choose the best picture and sound setting for your content. The LG OLED evo C3 not only perfo
rms great, but looks great as well. With an almost invisible bezel, it will blend into the background for
a seamless look. When you're finished watching, display paintings, photos and other content to blend the
LG OLED evo C3 into your space even more. But that's not all. Experience less searching and more streamin
g, thanks to the next generation of AI technology from LG webOS 23. Every LG OLED comes loaded with Dolby
Vision for extraordinary color, contrast and brightness, plus Dolby Atmos** for wrap-around sound. And L
G's FILMMAKER MODE allows you to see films just as the director intended. Packed with gaming features, th
e LG OLED evo C-Series comes with everything you need to win like a 0.1ms response time, native 120Hz ref
resh rate and four HDMI 2.1 inputs. *Based on LG internal testing: 55/65/77/83 LG OLED evo C3 models are
brighter than non-OLED evo B3 models and excludes the 42 and 48 LG OLED evo C3. **Dolby, Dolby Atmos and
the double-D symbol are registered trademarks of Dolby Laboratories.",
 "OLED83C3PUA", 4757.29, "LG" , "http://www.lgelectronics.com",  ["TV", "Smart TV", "OLED", "LG", "Big Sc
reen", "83 Inch"],["Electronics", "Television & Video", "Televisions",  "OLED TVs"],
 "02/27/2023", false );

   console.log(lgTV);
}

main();
```

Would return and log:

```
{
    _id: "507f1f77bcf86cd799439011",
    productName: "83 inch LG C3 OLED TV",
    productDescription: "The advanced LG OLED evo C-Series is better than ever. The LG OLED evo C3 is pow
ered by the next-gen a9 AI Processor Gen6—exclusively made for LG OLED—for ultra-realistic picture and so
und. And the Brightness Booster improves brightness so you get luminous picture and high contrast, even i
n well-lit rooms.* AI-assisted deep learning analyzes what you're watching to choose the best picture and
sound setting for your content. The LG OLED evo C3 not only performs great, but looks great as well. With
an almost invisible bezel, it will blend into the background for a seamless look. When you're finished wa
tching, display paintings, photos and other content to blend the LG OLED evo C3 into your space even mor
e. But that's not all. Experience less searching and more streaming, thanks to the next generation of AI
technology from LG webOS 23. Every LG OLED comes loaded with Dolby Vision for extraordinary color, contra
st and brightness, plus Dolby Atmos** for wrap-around sound. And LG's FILMMAKER MODE allows you to see fi
lms just as the director intended. Packed with gaming features, the LG OLED evo C-Series comes with every
thing you need to win like a 0.1ms response time, native 120Hz refresh rate and four HDMI 2.1 inputs. *Ba
sed on LG internal testing: 55/65/77/83 LG OLED evo C3 models are brighter than non-OLED evo B3 models an
d excludes the 42 and 48 LG OLED evo C3. **Dolby, Dolby Atmos and the double-D symbol are registered trad
emarks of Dolby Laboratories.",
    modelNumber: "OLED83C3PUA",
    price: 4757.29,
    manufacturer:"LG",
    manufacturerWebsite: "http://www.lgelectronics.com",
    keywords: ["TV", "Smart TV", "OLED", "LG", "Big Screen", "83 Inch"],
    categories: ["Electronics", "Television & Video", "Televisions",  "OLED TVs"],
    dateReleased: "02/27/2023",
    discontinued: false
}
```

This product will be stored in the **products** collection.

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

Notice the output/return value from your data function 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 products in the collection. **If there are no products in your DB, this function will return an empty array**

```
import * as products import "./products.js";

async function main() {
    const allProducts = await products.getAll();
    console.log(products);
}

main();
```

Would return and log all the products in the database.

```
[{
    _id: "507f1f77bcf86cd799439011",
    productName: "83 inch LG C3 OLED TV",
    productDescription: "The advanced LG OLED evo C-Series is better than ever. The LG OLED evo C3 is pow
ered by the next-gen a9 AI Processor Gen6—exclusively made for LG OLED—for ultra-realistic picture and so
und. And the Brightness Booster improves brightness so you get luminous picture and high contrast, even i
n well-lit rooms.* AI-assisted deep learning analyzes what you're watching to choose the best picture and
sound setting for your content. The LG OLED evo C3 not only performs great, but looks great as well. With
an almost invisible bezel, it will blend into the background for a seamless look. When you're finished wa
tching, display paintings, photos and other content to blend the LG OLED evo C3 into your space even mor
e. But that's not all. Experience less searching and more streaming, thanks to the next generation of AI
technology from LG webOS 23. Every LG OLED comes loaded with Dolby Vision for extraordinary color, contra
st and brightness, plus Dolby Atmos** for wrap-around sound. And LG's FILMMAKER MODE allows you to see fi
```

```
lms just as the director intended. Packed with gaming features, the LG OLED evo C-Series comes with every
thing you need to win like a 0.1ms response time, native 120Hz refresh rate and four HDMI 2.1 inputs. *Ba
sed on LG internal testing: 55/65/77/83 LG OLED evo C3 models are brighter than non-OLED evo B3 models an
d excludes the 42 and 48 LG OLED evo C3. **Dolby, Dolby Atmos and the double-D symbol are registered trad
emarks of Dolby Laboratories.",
    modelNumber: "OLED83C3PUA",
    price: 4757.29,
    manufacturer:"LG",
    manufacturerWebsite: "http://www.lgelectronics.com",
    keywords: ["TV", "Smart TV", "OLED", "LG", "Big Screen", "83 Inch"],
    categories: ["Electronics", "Television & Video", "Televisions",  "OLED TVs"],
    dateReleased: "02/27/2023",
    discontinued: false
},
{
    _id: "507f1f77bcf86cd799439012",
    productName: "Apple iPhone 14 Pro 1TB - Space Grey",
    productDescription: "The all new iPhone 14 pro has many upgraded features.............",
    modelNumber: "MQ2L3LL/A",
    price: 1499,
    manufacturer:"Apple",
    manufacturerWebsite: "http://www.apple.com",
    keywords: ["Cell Phone", "Phone", "iPhone", "Apple", "Smartphone", "iPhone 14", "pro"],
    categories: ["Electronics", "Cell Phones and Accessories", "Cell Phones"],
    dateReleased: "09/16/2022",
    discontinued: false
},
{
    _id: "507f1f77bcf86cd799439013",
    productName: "Apple iPhone 13 mini 512GB - Product Red",
    productDescription: "The iPhone 13 Mini is the perfect sized smartphone.............",
    modelNumber: "MQ3L2ML/A",
    price: 999,
    manufacturer:"Apple",
    manufacturerWebsite: "http://www.apple.com",
    keywords: ["Cell Phone", "Phone", "iPhone", "Apple", "Smartphone", "iPhone 13", "mini", "smaller smar
tphones"],
    categories: ["Electronics", "Cell Phones and Accessories", "Cell Phones"],
    dateReleased: "09/24/2021",
    discontinued: 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.

async get(id);

When given an id, this function will return a product 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 product exists with that `id`, the method should throw.

For example, you would use this method as:

```
import * as products from "./products.js";

async function main() {
    const iPhonePro = await products.get("507f1f77bcf86cd799439012");
 console.log(iPhonePro);
}
```

```
main();
```

Would return and log the iPhone 14 pro:

```
{
   _id: "507f1f77bcf86cd799439012",
    productName: "Apple iPhone 14 Pro 1TB - Space Grey",
    productDescription: "The all new iPhone 14 pro has many upgraded features.............",
    modelNumber: "MQ2L3LL/A",
    price: 1499,
    manufacturer:"Apple",
    manufacturerWebsite: "http://www.apple.com",
    keywords: ["Cell Phone", "Phone", "iPhone", "Apple", "Smartphone", "iPhone 14", "pro"],
    categories: ["Electronics", "Cell Phones and Accessories", "Cell Phones"],
    dateReleased: "09/16/2022",
    discontinued: 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 and then pass that converted value to your query.

```
//We need to import ObjectId from mongo
import { ObjectId }  from '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 (whi
ch it is in this case)
I also pass in an invalid object ID when I call my function to show the error */


let newObjId = new 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";

 //check to make sure it's not a string with just spaces
  if (id.trim.length === 0 ) throw "Id cannot be a blank string or a string with just spaces";

 //trim the string to remove any leading/trailing whitespace if any
 id= id.trim()
```

```
//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 'new 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 remove(id)

This function will remove the product 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 product cannot be removed (does not exist), the method should throw.

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

```
import * as products from "./products.js";

async function main() {
    const removeIphonePro = await products.remove("507f1f77bcf86cd799439012");
 console.log(removeIphonePro);
}
main();
```

Would return and then log: `"Apple iPhone 14 Pro 1TB - Space Grey has been successfully deleted!"`

NOTE: YOU MUST RETURN THIS EXACT SENTENCE. Points will be deducted if you do not match this format 100% including the "!" and including the product name that was deleted. Your output must match exactly! Word for word, character for character. It must match 100% or points will be deducted.

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, newProductName)

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

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

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

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

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

```
import * as products from "./products.js";

async function main() {
    const renamedIPhonePro = await products.rename("507f1f77bcf86cd799439012", "Apple iPhone 14 Pro 1TB -
Deep Purple");
 console.log(renamedIPhonePro);
}
main();
```

Would log the updated product:

```
{
   _id: "507f1f77bcf86cd799439012",
    productName: "Apple iPhone 14 Pro 1TB - Deep Purple",
    productDescription: "The all new iPhone 14 pro has many upgraded features.............",
    modelNumber: "MQ2L3LL/A",
    price: 1499,
    manufacturer:"Apple",
    manufacturerWebsite: "http://www.apple.com",
    keywords: ["Cell Phone", "Phone", "iPhone", "Apple", "Smartphone", "iPhone 14", "pro"],
    categories: ["Electronics", "Cell Phones and Accessories", "Cell Phones"],
    dateReleased: "09/16/2022",
    discontinued: 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()`.

app.js

For your app.js file, you will:

1. Create a product of your choice.
2. Log the newly created product. (Just that product, not all products)
3. Create another product of your choice.
4. Query all products, and log them all

5. Create the 3rd product of your choice.
6. Log the newly created 3rd product. (Just that product, not all product)
7. Rename the first product
8. Log the first product with the updated name.
9. Remove the second product you created.
10. Query all products, and log them all
11. Try to create a product with bad input parameters to make sure it throws errors.
12. Try to remove a product that does not exist to make sure it throws errors.
13. Try to rename a product that does not exist to make sure it throws errors.
14. Try to rename a product passing in invalid data for the `newProductName` parameter to make sure it throws errors.
15. Try getting a product 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)`

Lab 4 Rubric

Criteria	Ratings		Pts
products.js - create Test cases used for grading will be different from assignment examples.	**40 to >0.0 pts** **2 Points/Error Handling Test Case & 5 Points/Valid Input Test Case** 15 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, returns correct data, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation, none of the test cases pass, or the function does not return anything.	40 pts
products.js - getAll Test cases used for grading will be different from assignment examples.	**10 to >0.0 pts** **10 Points/Valid Input Test Case** 1 Valid Input Test Cases. The function is implemented correctly, returns correct data, and all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation, none of the test cases pass, or the function does not return anything.	10 pts
products.js - get Test cases used for grading will be different from assignment examples.	**15 to >0.0 pts** **1 Points/Error Handling Test Case & 10 Points/Valid Input Test Case** 5 Error Handling Test Cases & 1 Valid Input Test Cases. The function is implemented correctly, returns correct data, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation, none of the test cases pass, or the function does not return anything.	15 pts
products.js - remove Test cases used for grading will be different from assignment examples.	**15 to >0.0 pts** **1 Points/Error Handling Test Case & 10 Points/Valid Input Test Case** 5 Error Handling Test Cases & 1 Valid Input Test Cases. The function is implemented correctly, returns correct data, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation, none of the test cases pass, or the function does not return anything.	15 pts
products.js - rename Test cases used for grading will be different from assignment examples.	**20 to >0.0 pts** **2 Points/Error Handling Test Case & 5 Points/Valid Input Test Case** 5 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, returns correct data, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation, none of the test cases pass, or the function does not return anything.	20 pts
			Total Points: 100