Lab 6

Re-submit Assignment

Due Oct 24 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip

CS-546 Lab 6

A Book API

For this lab, you will create a simple server that provides an API for someone to Create, Read, Update, and Delete books and also book reviews.

We will be practicing:

- 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
- Developing a simple (10 route) API server

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)** and the **lecture 5 code (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_05/code)** and **lecture 6 code (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_06/code)** as a guide.

You can read up on **express (http://expressjs.com/)** on its home page. Specifically, you may find the **API Guide section on requests (http://expressjs.com/en/4x/api.html#req)** useful.

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

Folder Structure

You will use the following folder structure for the data module. **You may need other files to handle the connection to the database as well.**

```
./
../data/
../data/books.js
../data/index.js
../data/reviews.js
../routes/
../routes/books.js
../routes/index.js
../routes/reviews.js
../app.js
../package.json
```

I also recommend having your database settings centralized in files, such as:

```
./
../config/
../config/mongoConnection.js
../config/mongoCollections.js
```

Database Structure

You will use a database with the following structure:

- The database will be called **FirstName_LastName_lab6**
- The collection you use to store books will be called `books`
- The collection you use to store reviews will be called `reviews`

books

The schema for books is now as followed:

```
{
  _id: ObjectId generated by MongoDB,
  title: "book title",
  author: {authorFirstName: "first name", authorLastName: "last name"},
  genre: ["genre1", "genre2"], //array of genres, there must be at least one genre
  datePublished: Date field,
  summary: "Book summary...",
  reviews: [] //array of all the review ids for this book
}
```

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

For example:

```
{
  _id: ObjectId("bd8fa389-3a7a-4478-8845-e36a02de1b7b"),
  title: "The Shining",
  author: {authorFirstName: "Stephen", authorLastName: "King"},
  genre: ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
  datePublished: "1/28/1977",
  summary: "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the off
-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fami
ly and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever more
 remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
 the Overlook is Danny Torrance, a uniquely gifted five-year-old..",
  reviews: ["fd8fa389-3a7a-4478-8845-e36a02de1b2a7"] //array of all the review ids for this book
}
```

The Book Review Object

```
{
  _id: ObjectId generated by MongoDB,
  title: "title of review",
  reviewer: "name of reviewer",
  bookBeingReviewed: ID of book that is being reviewed,
  rating: 4,
  dateOfReview: "1/1/1930",
  review: "review will go here"
}
```

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

For example a review for The Shining:

```
{
  _id: ObjectId("fd8fa389-3a7a-4478-8845-e36a02de1b2a7"),
  title: "This book scared me to death!!",
  reviewer: "scaredycat",
  bookBeingReviewed: "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  rating: 5,
  dateOfReview: "10/7/2020",
  review: "This book was creepy!!! It had me at the edge of my seat.  One of Stephan King's best work!"
}
```

data/books.js

In books.js, you will create and export 5 methods. Create, Read (getting all and also getting by id), Update, and Delete. **You must do FULL error handling and input checking for ALL functions.**

data/reviews.js

In reviews.js, you will create and export 4 methods. Create, Read (getting all and also getting by id), and Delete. Y**ou must do FULL error handling and input checking for ALL functions.**

routes/books.js

You must do FULL error handling and input checking for ALL routes!

`GET /books`

Responds with an array of all books in the format of `{"_id": "book_id", "title": "book title"}` Note: Notice you are **ONLY** returning the book ID as a **string**, and Book Title

```
[{
  "_id": "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  "title": "The Shining"
},{
  "_id": "7d8fa389-3a7a-4478-8845-e8kskjh82jjk",
  "title": "Christine"
},.....]
```

`POST /books`

Creates a book with the supplied data in the request body, and returns the new book (ALL FIELDS MUST BE PRESENT AND CORRECT TYPE)

You should expect the following JSON to be submitted in the request.body:

```
{
  "title": "The Shining",
  "author": {authorFirstName: "Stephen", authorLastName: "King"},
  "genre": ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
  "datePublished": "1/28/1977",
  "summary": "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the o
ff-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fa
mily and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever mor
e remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
the Overlook is Danny Torrance, a uniquely gifted five-year-old.."
}
```

If the JSON provided does not match that schema, you will issue a 400 status code and end the request.

If the JSON is valid and the book can be created successful, you will return the newly created book with a 200 status code. **When a book is created, you will initialize the reviews array to be an empty array (since there can't be reviews of a book until the book is in the system)**

```
{
  _id: ObjectId("bd8fa389-3a7a-4478-8845-e36a02de1b7b"),
  title: "The Shining",
```

```
  author: {authorFirstName: "Stephen", authorLastName: "King"},
  genre: ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
  datePublished: "1/28/1977",
  summary: "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the off
-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fami
ly and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever more
 remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
 the Overlook is Danny Torrance, a uniquely gifted five-year-old..",
  reviews: []
}
```

`GET /books/{id}`

Example: `GET /books/bd8fa389-3a7a-4478-8845-e36a02de1b7b`

Responds with the full content of the specified book. So you will return all details of the book. Your function should return the book_id as a string, not an object ID

If no book with that `_id` is found, you will issue a 404 status code and end the request.

You will return the book with a 200 status code along with the book data if found.

```
{
  "_id": "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  "title": "The Shining",
  "author": {authorFirstName: "Stephen", authorLastName: "King"},
  "genre": ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
  "datePublished": "1/28/1977",
  "summary": "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the o
ff-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fa
mily and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever mor
e remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
the Overlook is Danny Torrance, a uniquely gifted five-year-old..",
  "reviews": ["fd8fa389-3a7a-4478-8845-e36a02de1b2a7"]
}
```

`PUT /books/{id}`

Example: `PUT /books/bd8fa389-3a7a-4478-8845-e36a02de1b7b`

This request will update an book with information provided from the PUT body. Updates the specified book **by replacing** the book with the new book content, and returns the updated book. (All fields need to be supplied in the request.body, even if you are not updating all fields)

You should expect the following JSON to be submitted:

```
{
  "title": "The Shining UPDATED",
  "author": {authorFirstName: "Patrick", authorLastName: "Hill"},
  "genre": ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
  "datePublished": "1/28/1977",
```

```
  "summary": "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the o
ff-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fa
mily and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever mor
e remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
the Overlook is Danny Torrance, a uniquely gifted five-year-old.."
}
```

reviews should not be able to be modified in this route. You must copy the old array of review ids from the existing book first and then insert them into the updated document so they are retained and not overwritten.

If the JSON provided in the PUT body is not as stated above, fail the request with a 400 error and end the request.

If no books exist with an `_id` of `{id}`, return a 404 and end the request.

If the update was successful, then respond with that updated book with a 200 status code

```
{
  "title": "The Shining UPDATED",
"author": {authorFirstName: "Patrick", authorLastName: "Hill"},
"genre": ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
"datePublished": "1/28/1977",
"summary": "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the off
-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fami
ly and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever more
 remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
 the Overlook is Danny Torrance, a uniquely gifted five-year-old..",
"reviews": ["fd8fa389-3a7a-4478-8845-e36a02de1b2a7"] }
```

`PATCH /books/{id}`

Example: `PATCH /books/bd8fa389-3a7a-4478-8845-e36a02de1b7b`

This request will update an book with information provided from the PATCH body. Updates the specified book with **only** the supplied changes, and returns the updated book. **One or more fields can be supplied but at least one field!**

For the `genres` array, you will append the genres in the request.body (if they are supplied in the request.body for this route) to the array already stored in the collection in the database, if a genre already exists in the genres array in the DB, you will just ignore it (there should be no duplicates in the genres array in the DB)

You should expect the following JSON to be submitted (this is just an example to show that you only need to supply the fields you are updating in the request.body, in the case below, you are only updating title and author, but this can be any combination of fields):

```
{
  "title": "The Shining UPDATED",
  "author": {authorFirstName: "Patrick", authorLastName: "Hill"}
}
```

reviews should not be able to be modified in this route. You must copy the old array of review ids from the existing book first and then insert them into the updated document so they are retained and not overwritten.

If the JSON provided in the PATCH body does not have at least one valid field, fail the request with a 400 error and end the request.

If no books exist with an _id of {id}, return a 404 and end the request.

If the update was successful, then respond with that updated book with a 200 status code.

```
{

  "title": "The Shining UPDATED",
"author": {authorFirstName: "Patrick", authorLastName: "Hill"},
"genre": ["Novel", "Horror fiction", "Gothic fiction", "Psychological horror", "Occult Fiction"],
"datePublished": "1/28/1977",
"summary": "Jack Torrance’s new job at the Overlook Hotel is the perfect chance for a fresh start. As the off
-season caretaker at the atmospheric old hotel, he’ll have plenty of time to spend reconnecting with his fami
ly and working on his writing. But as the harsh winter weather sets in, the idyllic location feels ever more
 remote . . . and more sinister. And the only one to notice the strange and terrible forces gathering around
 the Overlook is Danny Torrance, a uniquely gifted five-year-old..",
"reviews": ["fd8fa389-3a7a-4478-8845-e36a02de1b2a7"]

}
```

`DELETE /books/{id}`

Example: `DELETE /books/bd8fa389-3a7a-4478-8845-e36a02de1b7b`

If no book exists with an `_id` of `{id}`, return a 404 and end the request.

Deletes the book, sends a status code 200 and returns:

```
{"bookId": "bd8fa389-3a7a-4478-8845-e36a02de1b7b", "deleted": true}.
```

Note: When a book is deleted, you must delete all reviews for it that are stored in the reviews collection for that book ID

routes/reviews.js

You must do FULL error handling and input checking for ALL routes!

`GET /reviews/{bookId}`

Example: `GET /reviews/bd8fa389-3a7a-4478-8845-e36a02de1b7b`

Getting this route will return an array of all reviews in the system for the specified book id.

If no reviews for the book `_id` are found, you will issue a 404 status code and end the request.

You will return the array of reviews with a 200 status code along with the review data if found.

```
[{
  "_id": "fd8fa389-3a7a-4478-8845-e36a02de1b2a7"),
  "title": "This book scared me to death!!",
  "reviewer": "scaredycat",
  "bookBeingReviewed": "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  "rating": 5,
  "dateOfReview": "10/7/2020",
  "review": "This book was creepy!!! It had me at the edge of my seat.  One of Stephan King's best work!"
}, .........
]
```

`POST /reviews/{bookId}`

Example: `POST /reviews/bd8fa389-3a7a-4478-8845-e36a02de1b7b`

You should expect the following JSON to be submitted in the request.body:

```
{
  "title": "This book scared me to death!!",
  "reviewer": "scaredycat",
  "bookBeingReviewed": "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  "rating": 5,
  "dateOfReview": "10/7/2020",
  "review": "This book was creepy!!! It had me at the edge of my seat.  One of Stephan King's best work!"
}
```

If the JSON provided does not match that schema, you will issue a 400 status code and end the request.

if the bookId is not valid (it cannot be found), you will issue a 400 status code and end the request.

When a review is created, you must insert the review ID into the the book document in the reviews array for the provided bookId. Hence, why we need to make sure the book is a valid book in our books collection as noted above.

If the JSON is valid and the review can be created successful, you will return the newly created review with a 200 status code.

```
{
  "_id": "fd8fa389-3a7a-4478-8845-e36a02de1b2a7"),
  "title": "This book scared me to death!!",
```

```
  "reviewer": "scaredycat",
  "bookBeingReviewed": "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  "rating": 5,
  "dateOfReview": "10/7/2020",
  "review": "This book was creepy!!! It had me at the edge of my seat.  One of Stephan King's best work!"
}
```

`GET /reviews/{bookId}/{reviewId)`

Example: `GET /reviews/bd8fa389-3a7a-4478-8845-e36a02de1b7b/fd8fa389-3a7a-4478-8845-e36a02de1b2a7`

If no review with that `_id` is found, you will issue a 404 status code and end the request.

You will return the review with a 200 status code.

```
{
  "_id": "fd8fa389-3a7a-4478-8845-e36a02de1b2a7",
  "title": "This book scared me to death!!",
  "reviewer": "scaredycat",
  "bookBeingReviewed": "bd8fa389-3a7a-4478-8845-e36a02de1b7b",
  "rating": 5,
  "dateOfReview": "10/7/2020",
  "review": "This book was creepy!!! It had me at the edge of my seat.  One of Stephan King's best work!"
}
```

`Delete /reviews/{bookId}/{reviewId)`

Example: `DELETE /reviews/bd8fa389-3a7a-4478-8845-e36a02de1b7b/fd8fa389-3a7a-4478-8845-e36a02de1b2a7`

Deletes the specified review for the specified book, sends a 200 status code and returns:

```
{"reviewId": "fd8fa389-3a7a-4478-8845-e36a02de1b2a7", "deleted": true}.
```

NOTE: You MUST also remove the review ID from the array of reviews in the books collection if the review is deleted

app.js

Your app.js file will start the express server on **port 3000**, and will print a message to the terminal once the server is started.

Tip for testing:

You should create a seed file that populates your DB with initial data for both books and reviews. This will GREATLY improve your debugging as you should have enough sample data to do proper testing and it would be rather time consuming to enter a book and reviews for that book one by one through the API. A seed file is not required and is optional but is highly recommended. You should have a DB with

at least 10 books and multiple reviews for each book for proper testing (again, this is not required, but it is to ensure you can test thoroughly.)

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 throw, 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`