# Lab 6

**Due** Mar 5, 2022 by 11:59pm **Points** 100 **Submitting** a file upload
**File Types** zip

# CS-546 Lab 6

# A Band API

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

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 (9 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 and routes module. **You may need other files to handle the connection to the database as well.**

```
./
../data/
../data/bands.js
../data/index.js
```

```
../data/albums.js
../routes/
../routes/bands.js
../routes/index.js
../routes/albums.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 bands will be called `bands` you will store a sub-document of `albums`

## bands

The schema for a band is as followed:

```
{
    _id: ObjectId,
    name: string,
    genre: [strings],
    website: string (must contain full url http://www.patrickseats.com),
    recordLabel: string,
    bandMembers: [strings],
    yearFormed: number,
    albums:[], (an array of album objects, you will initialize this field to be an empty array when a ban
d is created),
    overallRating: number (from 0 to 5 this will be a computed average from all the album ratings posted
for a band,
    the initial value of this field will be 0 when a band is created)
}
```

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

An example of how Pink Floyd would be stored in the DB:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    name: "Pink Floyd",
    genre: ["Progressive Rock", "Psychedelic rock", "Classic Rock"],
    website: "http://www.pinkfloyd.com",
    recordLabel: "EMI",
    bandMembers: ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    yearFormed: 1965.
    albums: [],
    overallRating: 0
}
```

# The Band Album Sub-document (stored within the bands document)

```
{
  _id: ObjectId,
  title: string,
  releaseDate: string (string value of a date in MM/DD/YYYY format),
  tracks: array of strings,
  rating: number 1-5 (floats will be accepted as long as they are in the range 1.5 or 4.8 for example. We will only use one decimal place)
}
```

For example an album:

```
{
  _id: ObjectId("603d992b919a503b9afb856e"),
  title: "Wish You Were Here",
  releaseDate: "09/12/1975",
  tracks: ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy Harper)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"],
  rating: 5
}
```

# data/bands.js

In bands, you will create and export 5 methods. Create, Read (one for getting all and also one getting by id), Update, and Delete. **You must do FULL error handling and input checking for ALL functions as you have in previous labs, checking if the input is supplied, correct type, range etc. and throwing errors when you encounter bad input. You can use the functions you used for lab 4 however, you will need to create an update function. rename from lab 4 will not be used.**

**As a reminder, you will keep the same function names you used in lab 4:**

`create(name, genre, website, recordLabel, bandMembers, yearFormed)`

`getAll()`

`get(id)`

`remove(id)`

`update (id, name, genre, website, recordLabel, bandMembers, yearFormed)` Note: this is the new function you will create

**For the functions you did in lab 4, all the same input requirements apply in this lab, for new update function, those requirements are stated below.**

**NOTE: `overallRating` is not passed into either create or update as it will be a computed average field of all album ratings. For create: When a band is created, in your DB function, you will initialize the albums array to be an empty array. You will also initialize overallRating to be 0 when a band is created. For update: you will not modify the overallRating field in the update function, you must keep that field intact as it was before the update.**

## async update(id, name, genre, website, recordLabel, bandMembers, yearFormed)

This function will update **all** the data of the band currently in the database.

If `id, name, genre, website, recordLabel, bandMembers, yearFormed` are not provided at all, the method should throw. (**All fields need to have valid values**);

If `id, name, website, recordLabel` are not `strings` or are empty strings, the method should throw.

If `id` is not a valid `ObjectId`, the method should 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 `genre, bandMembers` 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. (all elements must be strings as well)

If `yearFormed` is not a `number`, or if `yearFormed` is less than 1900 or greater than the current year (2022) the the method should throw. (so only years 1900-2022 are valid values)

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

# data/albums.js

In albums, you will create and export 4 methods. Create, Read (one for getting all and also one getting by id), and Delete. Y**ou must do FULL error handling and input checking for ALL functions as you have in previous labs, checking if input is supplied, correct type, range etc. and throwing errors when you encounter bad input.**

**Function Names:**

`create(bandId, title, releaseDate, tracks, rating)`
`getAll(bandId)`
`get(albumId)`
`remove(albumId)`

## async create(bandId, title, releaseDate, tracks, rating);

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

If `bandId, title, releaseDate, tracks, rating` are not provided at all, the method should throw. (**All fields need to have valid values**);

If `bandId, title, releaseDate` are not `strings` or are empty strings, the method should throw.

If the `bandId` provided is not a valid `ObjectId`, the method should throw

If the band doesn't exist with that `bandId`, the method should throw

If `tracks` is not an array and if it does not have **at least** 3 elements in the array that is are valid `strings`, or are empty strings the method should throw. (all elements must be strings as well, but there must be AT LEAST 3)

If `releaseDate` is not a valid date string, the method will throw.

If `releaseDate` is < 1900 or is > the current year + one year, the method should throw.

If `rating` is not a number from 1 to 5, the method will throw. (floats will be accepted as long as they are in the range 1.5 or 4.8 for example. We will only use one decimal place)

Note: FOR ALL INPUTS: Strings with empty spaces are NOT valid strings. So no cases of " " are valid.

## async getAll(bandId);

This function will return an array of objects of the albums given the `bandId`. **If there are no albums for the band, this function will return an empty array**

If the `bandId` is not provided, the method should throw.

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

If the `bandId` provided is not a valid `ObjectId`, the method should throw

If the band doesn't exist with that `bandId`, the method should throw.

## async get(albumId);

When given a `albumId`, this function will return an album from the band.

If the `albumId` is not provided, the method should throw.

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

If the `albumId` provided is not a valid `ObjectId`, the method should throw

If the album doesn't exists with that `albumId`, the method should throw.

## async remove(albumId):

This function will remove the album from the band in the database and then return the band object that the album belonged to showing that the album sub-document was removed from the band document.

If the `albumId` is not provided, the method should throw.

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

If the `albumId` provided is not a valid `ObjectId`, the method should throw

If the album doesn't exists with that `albumId`, the method should throw.

**General note on all data functions: Whenever you return data that has an _id included in the returned data, return it as a string as shown in all the examples below. Do NOT return it as an ObjectId**

# routes/bands.js

**You must do FULL error handling and input checking for ALL routes! checking if input is supplied, correct type, range etc. and responding with proper status codes when you encounter bad input. All the input types, values and ranges that apply to the DB functions apply to the routes as well so you will do all the same checks in the routes that you do in the DB functions before sending the data to the DB function**

`GET /bands`

Responds with an array of all bands in the format of `{"_id": "band_id", "name": "band_name"}` Note: Notice you are **ONLY** returning the band ID as a **string**, and band name

```
[{ "_id": "603d965568567f396ca44a72","name": "Pink Floyd"},{ "_id": "704f456673467g306fc44c34","name": "L
inkin Park"},.....]
```

`POST /bands`

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

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

```
{
    "name": "Pink Floyd",
    "genre": ["Progressive Rock", "Psychedelic rock", "Classic Rock"],
    "website": "http://www.pinkfloyd.com",
    "recordLabel": "EMI",
    "bandMembers": ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    "yearFormed": 1965.
}
```

If `name, genre, website, recordLabel, bandMembers, yearFormed` are not provided at all, the route should issue a 400 status code and end the request. (**All fields need to have valid values**);

If `name, website, recordLabel` are not `strings` or are empty strings, the route should issue a 400 status code and end the request.

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` the route should issue a 400 status code and end the request.

If `genre, bandMembers` 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 route should issue a 400 status code and end the request.

If `yearFormed` is not a `number`, or if `yearFormed` is less than 1900 or greater than the current year (2022) the route should issue a 400 status code and end the request. (so only years 1900-2022 are valid values)

Note: FOR ALL INPUTS: Strings with empty spaces are NOT valid strings. So no cases of " " are valid.

**Note: Notice that we are not passing the overallRating or the album array when we create a band. The album array will be initialized to an empty array and overallRating will be initialized with 0 in your DB function. We do not pass these fields in the request.body**

If the JSON provided does not match the above schema or fails the conditions listed above, you will issue a 400 status code and end the request.

If the JSON is valid and the band can be created successfully, you will return the newly created band (as shown below) with a 200 status code.

```
{
    "_id": "507f1f77bcf86cd799439011",
    "name": "Pink Floyd",
    "genre": ["Progressive Rock", "Psychedelic rock", "Classic Rock"],
    "website": "http://www.pinkfloyd.com",
    "recordLabel": "EMI",
    "bandMembers": ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    "yearFormed": 1965,
    "albums": [],
    "overallRating": 0
}
```

`GET /bands/{id}`

Example: `GET /bands/507f1f77bcf86cd799439011`

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

if `_id` is not a valid `ObjectId`, you will issue a 400 status code and end the request

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

You will return the band (as shown below) with a 200 status code along with the band data if found.

```
{
    "_id": "507f1f77bcf86cd799439011",
    "name": "Pink Floyd",
    "genre": ["Progressive Rock", "Psychedelic rock", "Classic Rock"],
    "website": "http://www.pinkfloyd.com",
    "recordLabel": "EMI",
    "bandMembers": ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    "yearFormed": 1965,
    "albums": [{"_id": "607f1f77bcf86cd799438122", "title": "Wish You Were Here", "releaseDate": "09/12/1
975", "tracks": ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy
Harper)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"], "rating": 5}, ....],
"overallRating": 4.5
}
```

`PUT /bands/{id}`

Example: `PUT /bands/507f1f77bcf86cd799439011`

This request will update a band with information provided from the PUT body. Updates the specified band **by replacing** the band with the new band content, and returns the updated band. **(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:

```
{
    "name": "Pink Floyddd",
    "genre": ["Classic Rock"],
    "website": "http://www.pinkfloydRocks.com",
    "recordLabel": "EMI",
    "bandMembers": ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    "yearFormed": 1965.
}
```

if the `id` url parameter is not a valid `ObjectId`, you will issue a 400 status code and end the request

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

If `name, genre, website, recordLabel, bandMembers, yearFormed` are not provided at all, the route should issue a 400 status code and end the request. (**All fields need to have valid values**);

If `name, website, recordLabel` are not `strings` or are empty strings, the route should issue a 400 status code and end the request.

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` the route should issue a 400 status code and end the request.

If `genre, bandMembers` 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 route should issue a 400 status code and end the request.

If `yearFormed` is not a `number`, or if `yearFormed` is less than 1900 or greater than the current year (2022) the route should issue a 400 status code and end the request.  (so only years 1900-2022 are valid values)

Note:  FOR ALL INPUTS: Strings with empty spaces are NOT valid strings.  So no cases of "  " are valid.

If the JSON provided does not match the above schema or fails the conditions listed above, you will issue a 400 status code and end the request.

**albums should not be able to be modified in this route.  You must copy the old array of album objects from the existing band first and then insert them into the updated document so they are retained and not overwritten. You should also not modify or overwrite the overallRating field, so copy that as well.**

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

If the update was successful, then respond with that updated restaurant  (as shown below) with a 200 status code

```
{
    "_id": "507f1f77bcf86cd799439011",
    "name": "Pink Floyddd",
    "genre": ["Classic Rock"],
    "website": "http://www.pinkfloydRocks.com",
```

```
    "recordLabel": "EMI",
    "bandMembers": ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    "yearFormed": 1965,
    "albums": [{"_id": "607f1f77bcf86cd799438122", "title": "Wish You Were Here", "releaseDate": "09/12/1
975", "tracks": ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy
Harper)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"], "rating": 5}, ....],
"overallRating": 4.5
}
```

`DELETE /bands/{id}`

Example: `DELETE /bands/` `507f1f77bcf86cd799439011`

if the `id` url parameter is not a valid `ObjectId`, you will issue a 400 status code and end the request

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

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

```
{"bandId": "507f1f77bcf86cd799439011", "deleted": true}.
```

# routes/albums.js

**You must do FULL error handling and input checking for ALL routes! checking if input is supplied, correct type, range etc. and responding with proper status codes when you encounter bad input. All the input types, values and ranges that apply to the DB functions apply to the routes as well so you will do all the same checks in the routes that you do in the DB functions before sending the data to the DB function**

`GET /albums/{bandId}`

Example: `GET /albums/` `306f1f77bcf86cd799431423`

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

if the `bandId` is not a valid `ObjectId`, you will issue a 400 status code and end the request

If no albums for the `bandId` are found, you will issue a 404 status code and end the request.

if the `bandId` is not found in the system, you will issue a 404 status code and end the request

You will return the array of albums (as shown below) with a 200 status code along with the album data if found.

```
[{
  "_id": "603d992b919a503b9afb856e",
  "title": "Wish You Were Here",
  "releaseDate": "09/12/1975",
  "tracks": ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy Harp
er)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"],
  "rating": 5
}, .........
]
```

`POST /albums/{bandId}`

Example: `POST /albums/306f1f77bcf86cd799431423`

Creates an album sub-document with the supplied data in the request body, and returns all the band data showing the albums **(ALL FIELDS MUST BE PRESENT AND CORRECT TYPE).**

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

```
{
  "title": "Wish You Were Here",
  "releaseDate": "09/12/1975",
  "tracks": ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy Harp
er)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"],
  "rating": 5
}
```

If `title, releaseDate, tracks` are not provided at all, the route will respond with a status code of 400 and end the request. (**All fields need to have valid values**);

If `bandId, title, releaseDate` are not `strings` or are empty strings, the route will respond with a status code of 400 and end the request.

If the `bandId` provided is not a valid `ObjectId`, the route will respond with a status code of 400 and end the request.

If the band doesn't exists with that `bandId`, the route will respond with a status code of 404 and end the request.

If `tracks` is not an array and if it does not have **at least** 3 elements in the array that is are valid `strings`, or are empty strings the route will respond with a status code of 400 and end the request. (all elements must be strings as well, but there must be AT LEAST 3)

If `releaseDate` is not a valid date string, the route will respond with a status code of 400 and end the request.

if `releaseDate` is < 1900 or is > the current year + one year, the route will respond with a status code of 400 and end the request.

If `rating` is not a number from 1 to 5, the route will respond with a status code of 400 and end the request. (floats will be accepted as long as they are in the range 1.5 or 4.8 for example. We will only use one decimal place)

If the JSON provided does not match that schema above or it fails any of the above conditions, you will issue a 400 status code and end the request.

If the JSON is valid and the album can be created successful, you will return all the band data showing the albums (as shown below) with a 200 status code.

```
{
    "_id": "507f1f77bcf86cd799439011",
    "name": "Pink Floyd",
    "genre": ["Progressive Rock", "Psychedelic rock", "Classic Rock"],
```

```
    "website": "http://www.pinkfloyd.com",
    "recordLabel": "EMI",
    "bandMembers": ["Roger Waters", "David Gilmour", "Nick Mason", "Richard Wright", "Sid Barrett" ],
    "yearFormed": 1965,
    "albums": [{"_id": "607f1f77bcf86cd799438122", "title": "Wish You Were Here", "releaseDate": "09/12/1
975", "tracks": ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy
Harper)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"], "rating": 5}, ....],
"overallRating": 4.5
}
```


`GET /albums/album/{albumId)`

Example: `GET /albums/album/603d992b919a503b9afb856e`

if the `albumId` is not a valid `ObjectId`, you will issue a 400 status code and end the request

If no album for the `albumId` are found, you will issue a 404 status code and end the request.

You will return the review (as shown below) with a 200 status code.


```
{
  "_id": "603d992b919a503b9afb856e",
  "title": "Wish You Were Here",
  "releaseDate": "09/12/1975",
  "tracks": ["Shine On You Crazy Diamond, Pts. 1-5", "Welcome to the Machine","Have a Cigar (Ft. Roy Harp
er)", "Wish You Were Here","Shine On You Crazy Diamond, Pts. 6-9"],
  "rating": 5
}
```


`Delete /albums/{albumId)`

Example: `DELETE /albums/603d992b919a503b9afb856e`

Deletes the specified album for the specified `albumId`, and then sends a 200 status code and returns:

`{"albumId": "603d992b919a503b9afb856e", "deleted": true}`.

if the `id` url parameter is not a valid `ObjectId`, you will issue a 400 status code and end the request.

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

NOTE: If an album is deleted, you need to recalculate the average for the `overallRating` field in the main band document

# 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 bands and albums. 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 bands and albums for that band 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 bands and multiple albums for each band 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`