Lab 6

Start Assignment

Due Tuesday by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip
Available Oct 21 at 12am - Nov 7 at 11:59pm

CS-546 Lab 6

A Event API

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

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)** and the **lecture 5 code (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_05)** and **lecture 6 code (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_06)** 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 MUST use the directory and file structure in the code stub or points will be deducted. You can download the starter template here: **Lab6_stub.zip (https://sit.instructure.com/courses/68229/files/12273423?wrap=1)**

(https://sit.instructure.com/courses/68229/files/12273423/download?download_frd=1)
(https://sit.instructure.com/courses/70006/files/12260140?wrap=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" property.

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 events will be called `events` you will store a sub-document of `attendees`
- **You must only use ONE collection. The attendees are stored in the events collection as an array of objects. If you use two collections, you will lose 50% of your grade!**

events

The schema for a event is as followed:

```
{
    _id: ObjectId,
    eventName: string,
    description: string,
    eventLocation: {streetAddress: string, city: string, state: string, zip: string}
    contactEmail: string,
    maxCapacity: number,
    priceOfAdmission: number,
    eventDate: string (string representation of a date),
    startTime: string (string representation of a time in 12 hour AM/PM format), NOTE: For data c
onsistency's sake we will follow the H:MM AM/PM format.  Example: 2:00 PM no leading zero! AM/PM
must be in caps and there should be one space between the time and AM/PM
    endTime: string (string representation of a time in 12 hour AM/PM format), NOTE: For data con
sistency's sake we will follow the H:MM AM/PM format.  Example: 2:00 PM no leading zero! AM/PM mu
st be in caps and there should be one space between the time and AM/PM
    publicEvent: boolean,
    attendees:[], (an array of attendee objects, you will initialize this field to be an empty ar
ray when an event is created),
    totalNumberOfAttendees: number (this field will hold a count to the total number of attendees
in the attendees array for an event, the initial value of this field will be 0 when an event is c
reated)
}
```

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

An example of how Patrick's Big End of Summer BBQ would be stored in the DB when it's first created:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    eventName: "Patrick's Big End of Summer BBQ",
    description: "Come join us for our yearly end of summer bbq!",
    eventLocation: {streetAddress: "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zip:
"07030"}
    contactEmail: "phill@stevens.edu",
    maxCapacity: 30,
```

```
    priceOfAdmission: 0,
    eventDate: "08/25/2024",
    startTime: "2:00 PM",
    endTime: "8:00 PM",
    publicEvent: false,
    attendees: [],
    totalNumberOfAttendees: 0
}
```

The Event Attendee Sub-document (stored within the Event document)

```
{
  _id: ObjectId,
  firstName: string,
  lastName: string,
  emailAddress: string,
}
```

An example of the attendee sub-document:

```
{
  _id: ObjectId("603d992b919a503b9afb856e"),
  firstName: "Jennifer",
  lastName: "Cheng",
  emailAddress: "JCheng123@gmail.com",
}
```

data/events.js

In events, 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 must also trim all inputs! You can use the functions you used for lab 4 however, you will need to create an update function and also modify your create and getAll functions. rename from lab 4 will not be used.**

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

```
create(eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventD
ate, startTime, endTime, publicEvent)
getAll()
get(id)
remove(id)
update(eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventD
ate, startTime, endTime, publicEvent)  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 the new update function, those requirements are stated below.

NOTE: `attendees` and `totalNumberOfAttendees` are not passed into either create or update functions.

NOTE: When an attendee is added, you will need to recalculate the total number of attendees for that event and then update the `totalNumberOfAttendees` field in the main document to reflect the new number of attendees.

For create: When an event is created, in your DB function, you will initialize the attendees array to be an empty array. You will also initialize `totalNumberOfAttendees` to be 0 when an event is created.

For getAll: You will have to modify this function so it just returns the _id and name of the event as shown below in the GET /events route example. (Hint: use a projection!)

For update: You will not modify the `totalNumberOfAttendees` or attendees in the update function, you must keep them intact as they were before the update.

async create(eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventDate, startTime, endTime, publicEvent);

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

NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM So you may have to change your create to accommodate this. Do not forget to initialize attendees to an empty array and set `totalNumberOfAttendees` to 0 when an event is created!

Note: This function returns data in the same format as lab 4.

async getAll()

This async function will return an array all of the event documents in your events collection. For each event, you will ONLY return the event ID and the event name. So you will need to modify your lab 4 getALL for this.

For example,

```
import * as events import "./events.js";
const allEvents = await events.getAll();
```

Would return all the events in the database.

```
[
  {
    _id: ObjectId("507f1f77bcf86cd799439011"),
    eventName: "Patrick's Big End of Summer BBQ"
 },
{
    _id: ObjectId("507f1f77bcf86cd799439012"),
    eventName: "Aiden's Birthday Bash",
```

```
},
{
    _id: ObjectId("507f1f77bcf86cd799439013"),
    eventName: "Juniper Sky reunion concert!",
}
]
```

async get(id)

This async function will take in an event ID and return the corresponding event from your events collection.

Note: This function returns data in the same format as lab 4.

async remove(id)

This async function will take in an event ID and delete the corresponding event from your events collection.

Note: This function returns data in the same format as lab 4.

async update(eventId, eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventDate, startTime, endTime, publicEvent)

This async function will return the newly updated event object, with **all** of the properties listed in the above schema.

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

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

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

If the event doesn't exist with that `eventId`, the method should throw.

If `eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventDate, startTime, endTime, publicEvent` are not provided at all, the method should throw. (**All fields need to have valid values**);

If `eventName, description, contactEmail, eventDate, startTime, endTime` are not `strings` or are empty strings, the method should throw. (strings with only spaces are not valid)

if `eventName` is less than 5 characters, the method should throw.

if `description` is less than 25 characters, the method should throw.

if `contactEmail` is not in a valid email address format, the method should throw (please research email address rules).

if `eventDate` is not a valid date in "MM/DD/YYYY" format. If it's not in the expected format or is not a valid date, the method should throw (09/31/2024 is not valid since there are not 31 days in Sept).

The `eventDate` must be greater than the current date (so only future events can be created).

The `startTime` must be a valid time in 12-hour AM/PM format "11:30PM": If it's not in the expected format or not a valid time, the method should throw. `NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM`

The `startTime` cannot be later than the `endTime`, if it is, the method should throw.

The `endTime` must be a valid time in 12-hour AM/PM format "11:30PM": If it's not in the expected format or not a valid time, the method should throw. `NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM`

The `endTime` cannot be earlier than the `startTime`, if it is, the method should throw.

The `endTime` should be at least 30 minutes later than the `startTime`, if it's not, the method should throw.

If `publicEvent` is not the expected type (boolean), the method should throw.

if `maxCapacity, priceOfAdmission` are not the expected type (numbers), the method should throw.

`maxCapacity` should be a positive whole number > 0, if it's not, the method should throw.

`priceOfAdmission` should be a positive whole number, positive 2 decimal place float or 0, if it's not, the method should throw. (a price of 0 would mean it's a free event, price can be a whole number `25` or a two decimal place float `25.50`

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

If `eventLocation.streetAddress`, `eventLocation.city`, `eventLocation.state`, `eventLocation.zip` are not supplied, the method should throw.

If `eventLocation.streetAddress`, `eventLocation.city`, `eventLocation.state`, `eventLocation.zip` are not all valid strings, the method should throw.

If `eventLocation.streetAddress`, is less than 3 characters, the method should throw.

If `eventLocation.city`, is less than 3 characters, the method should throw.

`eventLocation.state`, Must be a valid two character state abbreviation "NY", "NJ" etc..

If `eventLocation.zip`, is not a string that contains 5 numbers, 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!

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

For example:

```
consider the the following document:
{
     _id: ObjectId("507f1f77bcf86cd799439011"),
    eventName: "Patrick's Big End of Summer BBQ",
    description: "Come join us for our yearly end of summer bbq!",
    eventLocation: {streetAddress: "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zip:
"07030"}
    contactEmail: "phill@stevens.edu",
    maxCapacity: 30,
    priceOfAdmission: 0,
    eventDate: "08/25/2023",
    startTime: "2:00 PM",
    endTime: "8:00 PM",
    publicEvent: false,
    attendees: [{_id: ObjectId("603d992b919a503b9afb856e") firstName: "Jennifer", lastName: "Chen
g", emailAddress: "jCheng1234@gmail.com"}, {_id: ObjectId("603d992b919a503b9afb856f") firstName:
"Aiden", lastName: "Hill", emailAddress: "AHill1234@gmail.com"} ],
totalNumberOfAttendees: 2
}
```

If we update it:

```
import * as events from "./events.js";
const patrickBBQ = await events.update("Patrick's Birthday Bash!","Come join us for my yearly bir
thday bash!",{streetAddress: "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zip: "0703
0"}, "phill@stevens.edu",30,0,"08/25/2023","2:00 PM","9:00 PM",true);
```

Would return:

```
{
     _id: ObjectId("507f1f77bcf86cd799439011"),
    eventName: "Patrick's Birthday Bash!",
    description: "Come join us for my yearly birthday bash!",
    eventLocation: {streetAddress: "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zip:
"07030"}
    contactEmail: "phill@stevens.edu",
    maxCapacity: 30,
    priceOfAdmission: 0,
    eventDate: "08/25/2023",
    startTime: "2:00 PM",
    endTime: "9:00 PM",
    publicEvent: true,
    attendees: [{_id: ObjectId("603d992b919a503b9afb856e") firstName: "Jennifer", lastName: "Chen
g", emailAddress: "JCheng123@gmail.com"}, {_id: ObjectId("603d992b919a503b9afb856f") firstName:
"Aiden", lastName: "Hill", emailAddress: "AHill1234@gmail.com"} ],
totalNumberOfAttendees: 2
}
```

data/attendees.js

In attendees, 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. <u>You must also trim all inputs!</u>**

Function Names:

`createAttendee(eventId, firstName, lastName, emailAddress);`

`getAllAttendees(eventId)`

`getAttendee(attendeeId)`

`removeAttendee(attendeeId)`

async createAttendee(eventId, firstName, lastName, emailAddress);

This async function will return the event data with the newly created attendee

If `eventId, firstName, lastName, emailAddress` are not provided, the method should throw.(**All fields need to have valid values**);

If `eventId, firstName, lastName, emailAddress` are not `strings` or are empty strings, the method should throw.

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

If the event does not exist with that `eventId`, the method should throw.

If `emailAddress` is not in a valid email address format, the method should throw.

If an attendee with given `emailAddress` already exists in the `attendees` array for that `eventId`, the method should throw.

You MUST make sure that you do not add an attendee if the Max capacity has been reached already. Meaning, if the maxCapacity is 30 and you already have 30 attendees, this function should throw an error saying the event is already full and should not allow a 31st attendees.

```
import * as events from "./events.js";
const event = await events.createAttendee("507f1f77bcf86cd799439011", "Jennifer", "Cheng", "JCheng123@gmail.com")

returns:
{
     _id: ObjectId("507f1f77bcf86cd799439011"),
    eventName: "Patrick's Birthday Bash!",
    description: "Come join us for my yearly birthday bash!",
    eventLocation: {streetAddress: "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zip: "07030"}
    contactEmail: "phill@stevens.edu",
    maxCapacity: 30,
    priceOfAdmission: 0,
    eventDate: "08/25/2023",
    startTime: "2:00 PM",
    endTime: "9:00 PM",
    publicEvent: true,
    attendees: [{_id: ObjectId("603d992b919a503b9afb856e") firstName: "Jennifer", lastName: "Cheng", emailAddress: "JCheng123@gmail.com"}],
totalNumberOfAttendees: 1
}
```

NOTE: When an attendee is added, you will need to recalculate the total number of attendees for that event and then update the `totalNumberOfAttendees` field in the main document to reflect

the new number of attendees.

async getAllAttendees(eventId);

This function will return an array of objects of the attendees given the `eventId` return **ONLY** the attendees for the event, not any of the other event data. **If there are no attendees for the event, this function will return an empty array**

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

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

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

If the event doesn't exist with that `eventId`, the method should throw.

Note: the `{....}` is just to show there are more attendees objects in the array. Obviously you should not be returning "`{....}`" literally

```
import * as events from "./events.js";
const allEvents = await events.getAllAttendees("507f1f77bcf86cd799439011")

Would return:

[{
  _id: ObjectId("603d992b919a503b9afb856e"),
  firstName: "Jennifer",
  lastName: "Cheng",
  emailAddress: "JCheng123@gmail.com",
}, {.....}, {....}
]
```

async getAttendee(attendeeId);

When given a `attendeeId`, this function will return a single attendee from a event with the matching _id in mongo. Return ONLY the attendee object and not all of the event data. **Please make sure you return ONE single object, not an object inside an array!**

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

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

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

If the attendee doesn't exists with that `attendeeId`, the method should throw.

```
import * as events from "./events.js";
const attendee= await events.getAttendee("603d992b919a503b9afb856e")
would return:

{
  _id: ObjectId("603d992b919a503b9afb856e"),
  firstName: "Jennifer",
  lastName: "Cheng",
```

```
  emailAddress: "JCheng123@gmail.com",
}
```

async removeAttendee(attendeeId):

This function will remove the attendee from the event in the database and then return the event object that the attendee belonged to show that the attendee sub-document was removed from the event document.

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

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

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

If the event doesn't exists with that `attendeeId`, the method should throw.

```
import * as events from "./events.js";
const deleteAttendee = await events.removeAttendee("603d992b919a503b9afb856e")
```

Returns the full event object showing the attendee has been removed:

```
{
    _id: ObjectId("507f1f77bcf86cd799439011"),
    eventName: "Patrick's Big End of Summer BBQ",
    description: "Come join us for our yearly end of summer bbq!",
    eventLocation: {streetAddress: "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zip:
"07030"}
    contactEmail: "phill@stevens.edu",
    maxCapacity: 30,
    priceOfAdmission: 0,
    eventDate: "08/25/2024",
    startTime: "2:00 PM",
    endTime: "8:00 PM",
    publicEvent: false,
    attendees: [],
    totalNumberOfAttendees: 0
}
```

NOTE: When an attendee is deleted, You will need to recalculate the total number of attendees for that event and then update the `totalNumberOfAttendees` field in the main document to reflect the new number of attendees.

routes/events.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 /events`

Responds with an array of all events in the format of `{"_id": "event_id", "eventName": "event_name"}`
Note: Notice you are **ONLY** returning the event ID, and event name

```
[ {
    "_id": "507f1f77bcf86cd799439011",
    "eventName": "Patrick's Big End of Summer BBQ"
 },
{
    "_id": "507f1f77bcf86cd799439012",
    "eventName": "Aiden's Birthday Bash",

},
{
    "_id": "507f1f77bcf86cd799439013",
    "eventName": "Juniper Sky reunion concert!",
} ]
```

`POST /events`

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

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

```
{
    "eventName": "Patrick's Big End of Summer BBQ",
    "description": "Come join us for our yearly end of summer bbq!",
    "eventLocation": {"streetAddress": "1 Castle Point Terrace", "city": "Hoboken", "state": "N
J", "zip": "07030"}
    "contactEmail": "phill@stevens.edu",
    "maxCapacity": 30,
    "priceOfAdmission": 0,
    "eventDate": "08/25/2024",
    "startTime": "2:00 PM",
    "endTime": "8:00 PM",
    "publicEvent": false,
}
```

If `eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventDate, startTime, endTime, publicEvent` 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 `eventName, description, contactEmail, eventDate, startTime, endTime` are not `strings` or are empty strings, the route should issue a 400 status code and end the request.. (strings with only spaces are not valid)

if `eventName` is less than 5 characters, the route should issue a 400 status code and end the request.

if `description` is less than 25 characters, the route should issue a 400 status code and end the request.

if `contactEmail` is not in a valid email address format, the route should issue a 400 status code and end the request.

if `eventDate` is not a valid date in "MM/DD/YYYY" format. If it's not in the expected format or is not a valid date, the route should issue a 400 status code and end the request. (09/31/2024 is not valid since there are not 31 days in Sept).

The `eventDate` must be greater than the current date (so only future events can be created), if it's not, the route should issue a 400 status code and end the request.

The `startTime` must be a valid time in 12-hour AM/PM format "11:30PM": If it's not in the expected format or not a valid time, the route should issue a 400 status code and end the request. `NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM`

The `startTime` cannot be later than the `endTime`, if it is, the route should issue a 400 status code and end the request.

The `endTime` must be a valid time in 12-hour AM/PM format "11:30PM": If it's not in the expected format or not a valid time, the route should issue a 400 status code and end the request. `NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM`

The `endTime` cannot be earlier than the `startTime`, if it is, the route should issue a 400 status code and end the request.

The `endTime` should be at least 30 minutes later than the `startTime`, if it's not, the route should issue a 400 status code and end the request.

If `publicEvent` is not the expected type (boolean), the route should issue a 400 status code and end the request.

if `maxCapacity, priceOfAdmission` are not the expected type (numbers), the route should issue a 400 status code and end the request.

`maxCapacity` should be a positive whole number > 0, if it's not, the route should issue a 400 status code and end the request.

`priceOfAdmission` should be a positive whole number, positive 2 decimal place float or 0, if it's not, tthe route should issue a 400 status code and end the request. (a price of 0 would mean it's a free event, price can be a whole number `25` or a two decimal place float `25.50`

If `eventLocation` is not an object, the route should issue a 400 status code and end the request.

If `eventLocation.streetAddress`, `eventLocation.city`, `eventLocation.state`, `eventLocation.zip` are not supplied, the route should issue a 400 status code and end the request.

If `eventLocation.streetAddress`, `eventLocation.city`, `eventLocation.state`, `eventLocation.zip` are not all valid strings, tthe route should issue a 400 status code and end the request.

If `eventLocation.streetAddress`, is less than 3 characters, the route should issue a 400 status code and end the request.

If `eventLocation.city`, is less than 3 characters, the route should issue a 400 status code and end the request.

`eventLocation.state`, Must be a valid two character state abbreviation "NY", "NJ" etc.., If it's not, the route should issue a 400 status code and end the request.

If `eventLocation.zip`, is not a string that contains 5 numbers, the route should issue a 400 status code and end the request.. (only 5 digit zip but represented as a string, because leading 0's are valid in zip codes, yet JS drops leading 0's)

Note: FOR ALL STRING INPUTS: Strings with empty spaces are NOT valid strings. So no cases of " " are valid. **You MUST trim all inputs that are strings**

NOTE: as a reminder, `totalNumberOfAttendees` and `attendees` are not passed into this route or the PUT route, When an event is created, in your DB function, you will initialize the attendees array to be an empty array. You will also initialize `totalNumberOfAttendees` to be 0 when an event is created.

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 event can be created successfully, you will return the newly created event (as shown below) with a 200 status code.

```
{
    "_id": "507f1f77bcf86cd799439011",
    "eventName": "Patrick's Big End of Summer BBQ",
    "description": "Come join us for our yearly end of summer bbq!",
    "eventLocation": {"streetAddress": "1 Castle Point Terrace", city: "Hoboken", state: "NJ", zi
p: "07030"}
    "contactEmail": "phill@stevens.edu",
    "maxCapacity": 30,
    "priceOfAdmission": 0,
    "eventDate": "08/25/2024",
    "startTime": "2:00 PM",
    "endTime": "8:00 PM",
    "publicEvent": false,
    "attendees": [],
    "totalNumberOfAttendees": 0
}
```

`GET /events/:eventId`

Example: `GET /events/507f1f77bcf86cd799439011`

Responds with the full content of the specified event. So you will return all details of the event.

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

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

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

```
{
    "_id": "507f1f77bcf86cd799439011",
    "eventName": "Patrick's Big End of Summer BBQ",
    "description": "Come join us for our yearly end of summer bbq!",
    "eventLocation": {"streetAddress": "1 Castle Point Terrace", "city": "Hoboken", "state": "N
J", "zip": "07030"}
    "contactEmail": "phill@stevens.edu",
    "maxCapacity": 30,
    "priceOfAdmission": 0,
    "eventDate": "08/25/2024",
```

```
    "startTime": "2:00 PM",
    "endTime": "8:00 PM",
    "publicEvent": false,
    "attendees": [{"_id": "603d992b919a503b9afb856e" "firstName": "Jennifer", "lastName": "Chen
g", "emailAddress": "jCheng1234@gmail.com"}, {"_id": "603d992b919a503b9afb856f" "firstName": "Aid
en", "lastName": "Hill", "emailAddress": "AHill1234@gmail.com"} ],
"totalNumberOfAttendees": 2
}
```

`PUT /events/:eventId`

Example: `PUT /events/507f1f77bcf86cd799439011`

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

```
{
 "eventName": "Patrick's Birthday Bash!",
"description": "Come join us for my yearly birthday bash!",
"eventLocation": {"streetAddress": "1 Castle Point Terrace", "city": "Hoboken", "state": "NJ", "z
ip": "07030"}
"contactEmail": "phill@stevens.edu",
"maxCapacity": 30,
"priceOfAdmission": 0,
"eventDate": "08/25/2024",
"startTime": "2:00 PM",
"endTime": "9:00 PM",
"publicEvent": true
}
```

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

If no event exists with the `:eventId` provided, return a 404 and end the request.

If `eventName, description, eventLocation, contactEmail, maxCapacity, priceOfAdmission, eventDate, startTime, endTime, publicEvent` 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 `eventName, description, contactEmail, eventDate, startTime, endTime` are not `strings` or are empty strings, the route should issue a 400 status code and end the request.. (strings with only spaces are not valid)

if `eventName` is less than 5 characters, the route should issue a 400 status code and end the request.

if `description` is less than 25 characters, the route should issue a 400 status code and end the request.

if `contactEmail` is not in a valid email address format, the route should issue a 400 status code and end the request.

if `eventDate` is not a valid date in "MM/DD/YYYY" format. If it's not in the expected format or is not a valid date, the route should issue a 400 status code and end the request. (09/31/2024 is not valid

since there are not 31 days in Sept).

The `eventDate` must be greater than the current date (so only future events can be created), if it's not, the route should issue a 400 status code and end the request.

The `startTime` must be a valid time in 12-hour AM/PM format "11:30PM": If it's not in the expected format or not a valid time, the route should issue a 400 status code and end the request. `NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM`

The `startTime` cannot be later than the `endTime`, if it is, the route should issue a 400 status code and end the request.

The `endTime` must be a valid time in 12-hour AM/PM format "11:30PM": If it's not in the expected format or not a valid time, the route should issue a 400 status code and end the request. `NOTE: For data consistency's sake we will follow the H:MM AM/PM format. Example: 2:00 PM no leading zero! AM/PM must be in caps and there should be one space between the time and AM/PM`

The `endTime` cannot be earlier than the `startTime`, if it is, the route should issue a 400 status code and end the request.

The `endTime` should be at least 30 minutes later than the `startTime`, if it's not, the route should issue a 400 status code and end the request.

If `publicEvent` is not the expected type (boolean), the route should issue a 400 status code and end the request.

if `maxCapacity, priceOfAdmission` are not the expected type (numbers), the route should issue a 400 status code and end the request.

`maxCapacity` should be a positive whole number > 0, if it's not, the route should issue a 400 status code and end the request.

`priceOfAdmission` should be a positive whole number, positive 2 decimal place float or 0, if it's not, tthe route should issue a 400 status code and end the request. (a price of 0 would mean it's a free event, price can be a whole number `25` or a two decimal place float `25.50`

If `eventLocation` is not an object, the route should issue a 400 status code and end the request.

If `eventLocation.streetAddress`, `eventLocation.city`, `eventLocation.state`, `eventLocation.zip` are not supplied, the route should issue a 400 status code and end the request.

If `eventLocation.streetAddress`, `eventLocation.city`, `eventLocation.state`, `eventLocation.zip` are not all valid strings, tthe route should issue a 400 status code and end the request.

If `eventLocation.streetAddress`, is less than 3 characters, the route should issue a 400 status code and end the request.

If `eventLocation.city`, is less than 3 characters, the route should issue a 400 status code and end the request.

`eventLocation.state`, Must be a valid two character state abbreviation "NY", "NJ" etc.., If it's not, the route should issue a 400 status code and end the request.

If `eventLocation.zip`, is not a string that contains 5 numbers, the route should issue a 400 status code and end the request.. (only 5 digit zip but represented as a string, because leading 0's are valid in zip codes, yet JS drops leading 0's)

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.

NOTE: attendees should not be able to be modified in this route. You must copy the array of attendee objects from the existing product first and then insert them into the updated document so they are retained and not overwritten. You should also not modify or overwrite the totalNumberOfAttendees field, so keep that field intact as it was before the update. I know this technically isn't a full PUT, but with so many fields in the resource, I felt this was easier than a patch where it could be one or many fields.

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 product (as shown below) with a 200 status code

```
{
  "_id": "507f1f77bcf86cd799439011"),
  "eventName": "Patrick's Birthday Bash!",
"description": "Come join us for my yearly birthday bash!",
"eventLocation": {"streetAddress": "1 Castle Point Terrace", "city": "Hoboken", "state": "NJ", "z
ip": "07030"}
"contactEmail": "phill@stevens.edu",
"maxCapacity": 30,
"priceOfAdmission": 0,
"eventDate": "08/25/2024",
"startTime": "2:00 PM",
"endTime": "9:00 PM",
"publicEvent": true,
  "attendees": [{"_id": "603d992b919a503b9afb856e" "firstName": "Jennifer", "lastName": "Cheng",
"emailAddress": "jCheng1234@gmail.com"}, {"_id": "603d992b919a503b9afb856f" "firstName": "Aiden",
"lastName": "Hill", "emailAddress": "AHill1234@gmail.com"} ],
"totalNumberOfAttendees": 2
}
```

`DELETE /events/:eventId`

Example: `DELETE /events/507f1f77bcf86cd799439011`

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

If no prodeventsuct exists with the provided `eventId` url parameter, return a 404 and end the request.

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

```
{"eventName": "Patrick's Birthday Bash!", "deleted": true}
```

routes/attendees.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 /attendees/:eventId`

Example: `GET /attendees/507f1f77bcf86cd799439011`

Getting this route will return an array of all attendees in the system for the specified event id. You are only returning the array of attendees from the event, not any of the other event data!

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

If no attendees for the `eventId` are found, you will just return an empty array and 200 status code.

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

You will return the array of attendees (as shown below) with a 200 status code if found. Note: the `{....}` is just to show there are more attendees objects in the array. Obviously you should not be returning "`{....}`" literally

```
[{
  "_id": "603d992b919a503b9afb856e",
  "firstName": "Jennifer",
  "lastName": "Cheng",
  "emailAddress": "JCheng123@gmail.com",
}, {.....}, {....}
]
```

`POST /attendees/:eventId`

Example: `POST /attendees/507f1f77bcf86cd799439011`

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

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

```
{
"firstName": "Jennifer",
"lastName": "Cheng",
"emailAddress": "JCheng123@gmail.com",
}
```

If `firstName, lastName, emailAddress` 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 `firstName, lastName, emailAddress` are not `strings` or are empty strings, the route will respond with a status code of 400 and end the request.

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

If an event doesn't exist with that `eventId`, the route will respond with a status code of 404 and end the request.

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 attendee can be created successful, you will return all the event data showing the attendees (as shown below) with a 200 status code.

```
{
    "_id": "507f1f77bcf86cd799439011",
    "eventName": "Patrick's Big End of Summer BBQ",
    "description": "Come join us for our yearly end of summer bbq!",
    "eventLocation": {"streetAddress": "1 Castle Point Terrace", "city": "Hoboken", "state": "N
J", "zip": "07030"}
    "contactEmail": "phill@stevens.edu",
    "maxCapacity": 30,
    "priceOfAdmission": 0,
    "eventDate": "08/25/2024",
    "startTime": "2:00 PM",
    "endTime": "8:00 PM",
    "publicEvent": false,
    "attendees": [{"_id": "603d992b919a503b9afb856e", "firstName": "Jennifer", "lastName": "Chen
g", "emailAddress": "JCheng123@gmail.com"}],
"totalNumberOfAttendees": 1
}
```

`GET /attendees/attendee/:attendeeId`

Example: `GET /attendees/attendee/603d992b919a503b9afb856e`

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

If no attendee for the `attendeeId` is found, you will issue a 404 status code and end the request.

You will return the attendee (as shown below) with a 200 status code. Make sure you return ONE single object, not an array with an object in it

```
{
  "_id": "603d992b919a503b9afb856e",
  "firstName": "Jennifer",
  "lastName": "Cheng",
  "emailAddress": "JCheng123@gmail.com",
}
```

`Delete /attendees/attendee/:attendeeId`

Example: `DELETE /attendees/attendee/603d992b919a503b9afb856e`

Deletes the specified attendee for the specified `attendeeId`, and then sends a 200 status code and returns the full event data that the attendee belonged to to show the attendee has been removed :

```
{
 "_id": "507f1f77bcf86cd799439011",
 "eventName": "Patrick's Big End of Summer BBQ",
 "description": "Come join us for our yearly end of summer bbq!",
 "eventLocation": {"streetAddress": "1 Castle Point Terrace", "city": "Hoboken", "state": "NJ",
"zip": "07030"}
 "contactEmail": "phill@stevens.edu",
 "maxCapacity": 30,
 "priceOfAdmission": 0,
 "eventDate": "08/25/2024",
 "startTime": "2:00 PM",
 "endTime": "8:00 PM",
 "publicEvent": false,
 "attendees": [{"_id": 603d992b919a503b9afb856f "firstName": "Aiden", "lastName": "Hill", "emailA
ddress": "AHill1234@gmail.com"}],
"totalNumberOfAttendees": 1
}
```


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

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

NOTE: If an attendee is deleted, you need to recalculate and update the `totalNumberOfAttendees` field in the main product 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 events and attendees. 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 product and reviews for that product 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 events and multiple attendees for each event 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`