MongoDB driver and Postman

MongoDB driver

- For this module, we will use the latest native MongoDB driver for Node.js (https://mongodb.github.io/node-mongodb-native/)
- Why not a library like `Mongoose`?
  - Library like `Mongoose` enforces a fixed schema one the database;
  - This is common for relational databases, such as MySQL;
  - This is useful to ensure data is consistent;
  - But this is against the fundamental principle of NoSQL or JSON database being 'schema-less';
  - Introduce extra complexity that we don't have time for.

Project Dependencies

- These are all the modules needed (in the `package.json` file):

```
"dependencies": {
  "express": "^4.17.1",
  "mongodb": "^3.6.3",
}
```

- You can install them with `npm install` if they are not already installed globally.
- For example, to install the MongoDB driver use `npm install mongodb --save`
  - Note the package is `mongodb` not `mongo`
- You can add the `-g` switch to install it 'globally' so any other project can use it as well
  - `npm install -g mongodb --save`

List globally installed pacakges

- To see a list of globally installed node.js modules:

```
npm list -g --depth 0
```

- The `-g` flag shows globally installed modules. Without it, `npm` lists module installed in the current directory.
- With the `--dpeth 0` flag, `npm` shows only the top-level module. Otherwise, `npm` shows all the dependencies as well, which can be a lot of modules.

Creating the REST API

- Main file: `server.js` (all the code goes in here).
- Import dependencies modules: `const express = require('express')`
- Create an Express.js instance: `const app = express()`
- To extract parameters from the requests, we used to need the `body-parser` module
  - `app.use(bodyParser.json())`
- If you use Express 4.16+ you can now replace it with `app.use(express.json())`

Connect to MongoDB

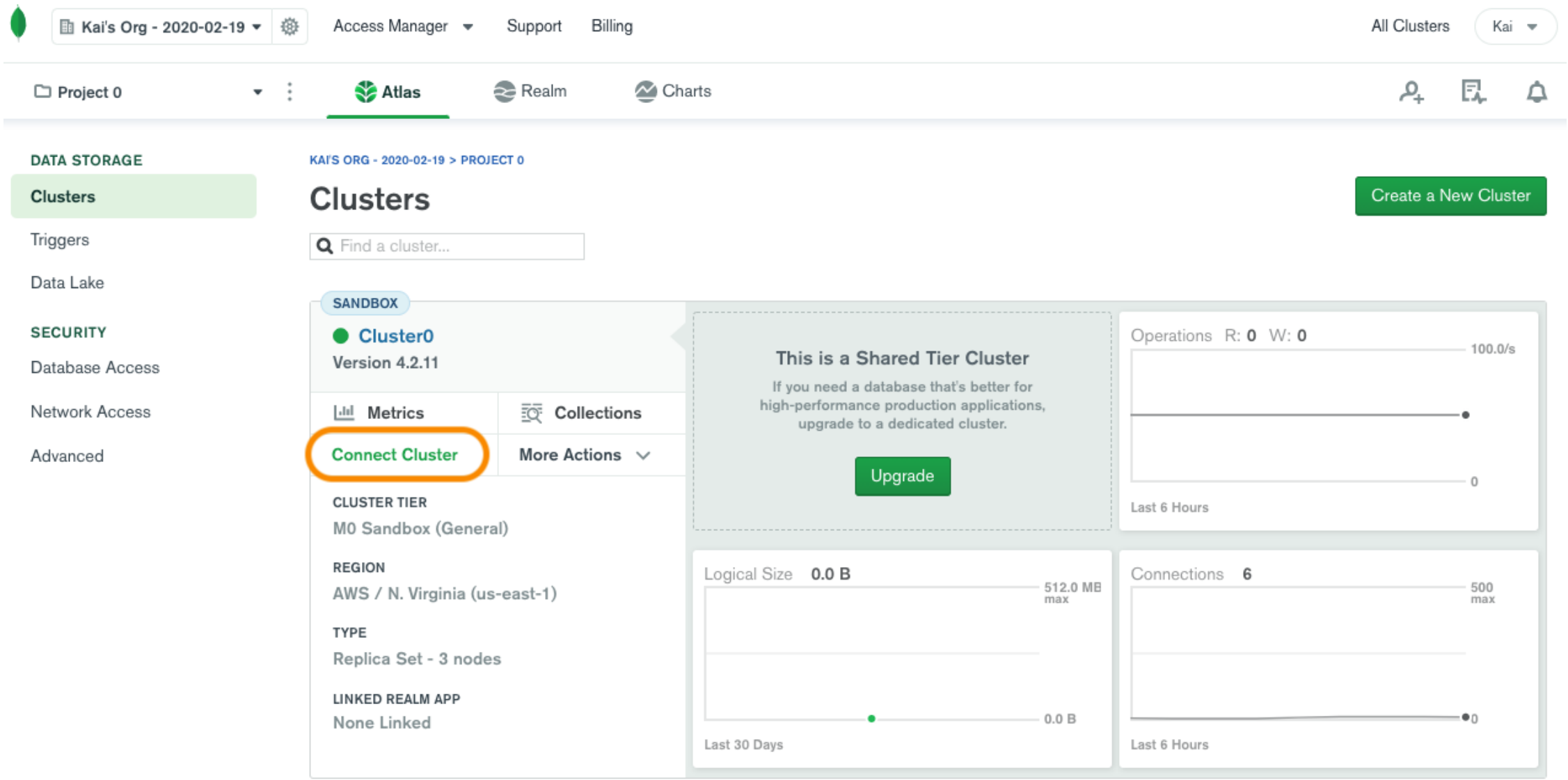
- To connect to a database in MongoDB, you need a `MongoClient` object, which is part of the driver:
- `const MongoClient = require('mongodb').MongoClient;`
- Then you can connect to a database (the `webstore` database in this example):

```
let db;
MongoClient.connect('mongodb://localhost:27017/', (err, client) => {
    db = client.db('webstore')
})
```

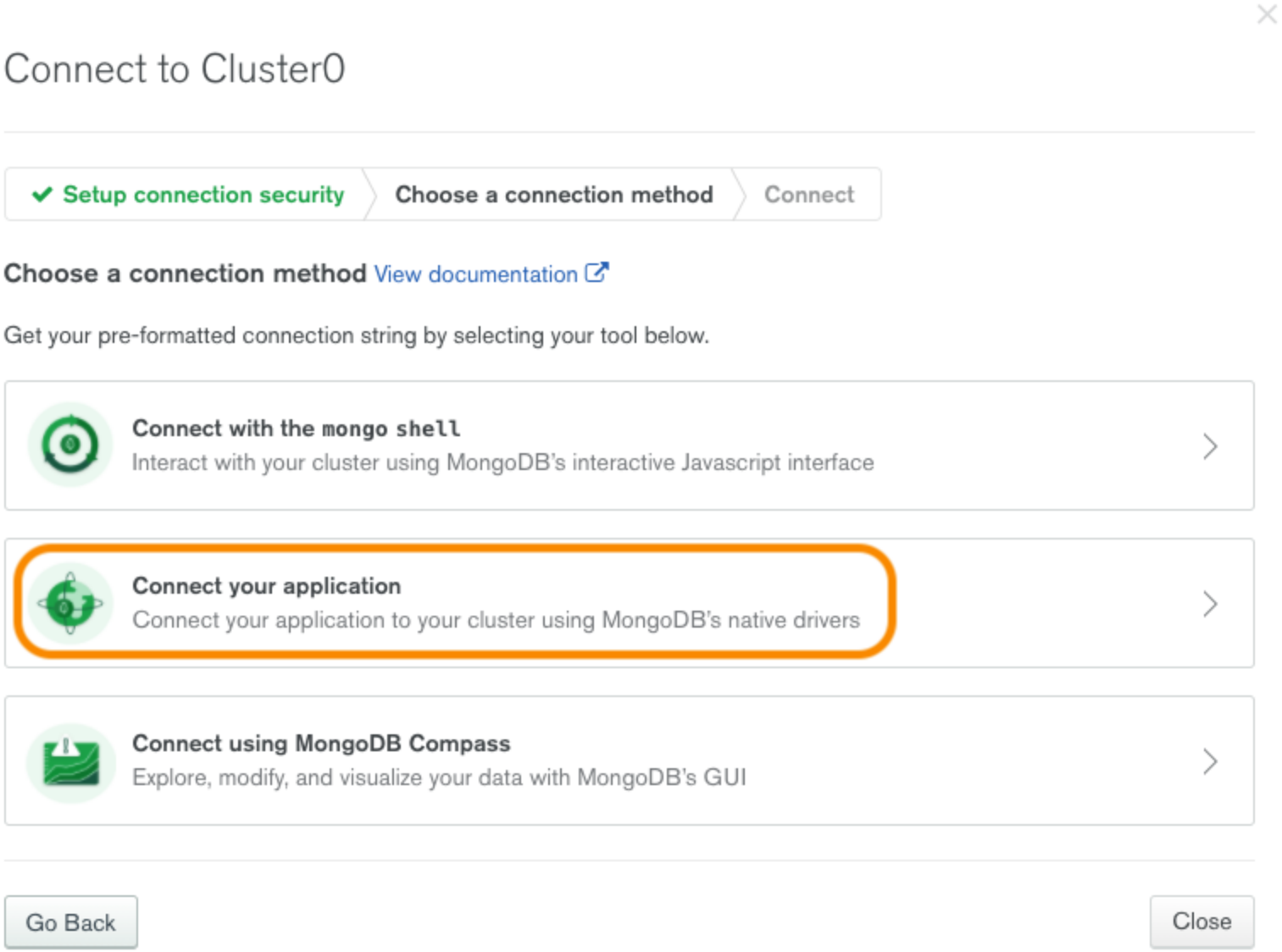
- If you need to connect to a (remote) database with username/password:

```
mongodb://username:password@hostURL:port
```

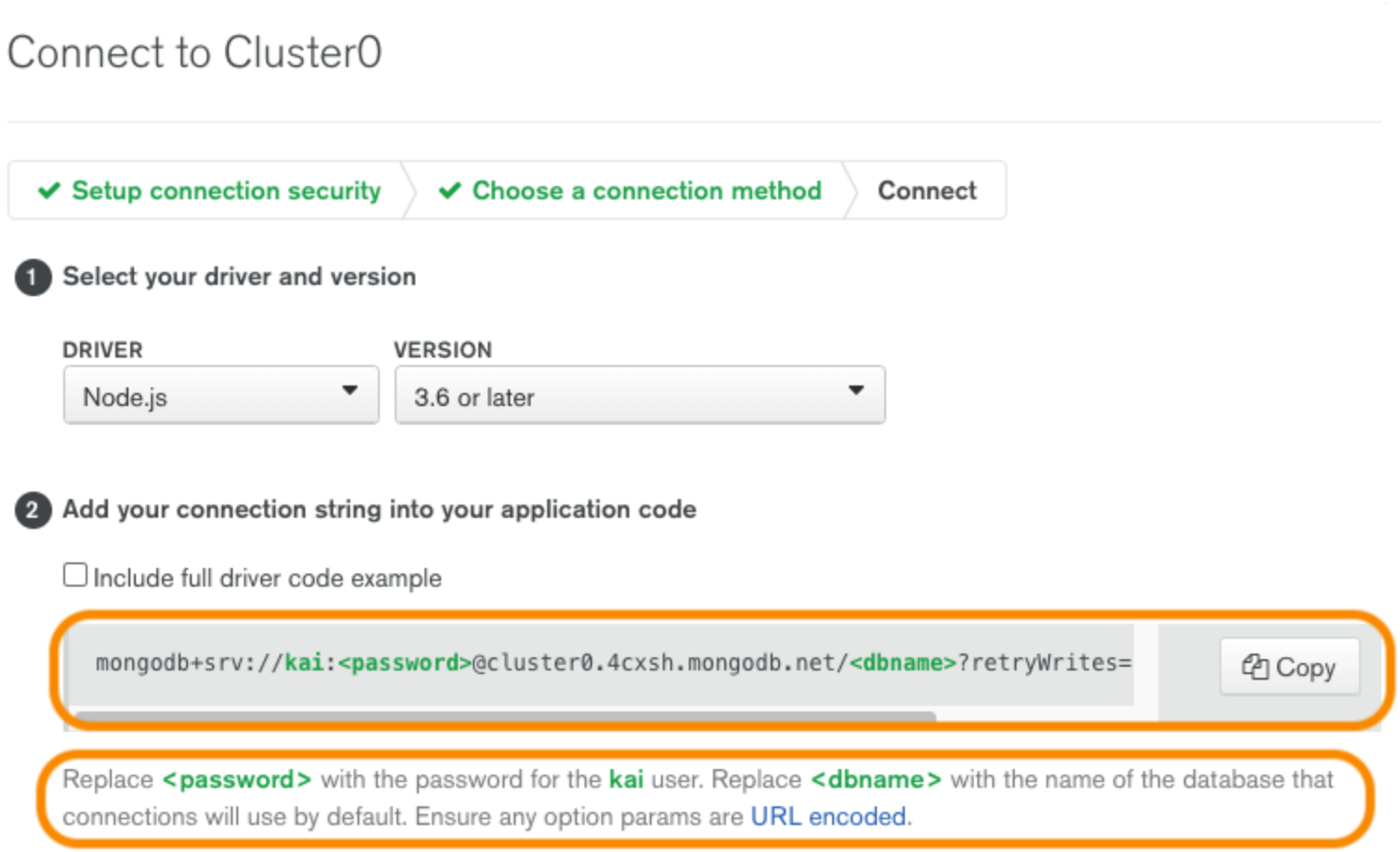
Connect to MongoDB Atlas



- Choose 'Connect your application'



- Copy the connection string
- Replace the `<password>` and `<dbname>` with the actual password and mongo database name



Get the MongoDB collection name

- The `app.param()` middleware allows to do something every time there is this value in the URL pattern of the request handler.
- In our case, we select a particular collection when a request pattern contains a string `collectionName` prefixed with a colon
    - Something like `/collection/:collectionName`

```
app.param('collectionName', (req, res, next, collectionName) => {
  req.collection = db.collection(collectionName)
  return next()
})
```

Root path response

- A common mistake when coding REST API is that it appears the API is not working, because nothing happens after typing `localhost:3000` in the browser.
- Instead, the URL should be an end point such as
  - `localhost:3000/collection/messages`
- To avoid such confusion, we include a root route message that asks users to specify a collection name in their URLs:

```
app.get('/', (req, res, next) => {
  res.send('Select a collection, e.g., /collection/messages')
})
```

Retrieve collection with the `Get`

- Here we create a Express route (end point) to retrieve all the objects in a collection.
- Here we use `find()` with the `req.collection`, which stores the name of the MongoDB Collection (two slides ago):

```
app.get('/collection/:collectionName', (req, res, next) => {
  req.collection.find({}).toArray((e, results) => {
      if (e) return next(e)
      res.send(results)
    }
  )
})
```

- The `find()` function returns a *cursor* that points to result array, not the actual results, so we need the `toArray()` function to get the actual results.

More complex `find()` query

- The way that the `find()` function works is similar to that in the MongoDB command line windows. Remember this? `db.lessons.find().pretty()`
- It is possible to include additional controls in the `find()` function, such as sorting.
- Below we now sort the results by `price` and limit the return to 5 objects.

```
app.get('/collection/:collectionName', (req, res, next) => {
  req.collection.find({}, {limit: 5, sort: [['price', -1]]})
    .toArray((e, results) => {
      if (e) return next(e)
      res.send(results)
    }
  )
})
```

Testing the API with `cURL`

- Browser can only send `GET` request, and this becomes an issue when testing other types of requests such as `POST`, `PUT`, and `DELETE`.
- You can write the front-end code for testing these request, but sometimes it is easier to use a tool.
- As mentioned in the last lecture, one of the common command line tool is `cURL`
  - It comes with the system on macOS, Linux, and Windows 10
  - If you are using an earlier version of Windows, you can download it from here: https://curl.haxx.se/windows/

- It is often a good idea of to use the 'verbose' mode (the `-v` argument), which give you more information: `curl -v localhost:3000`

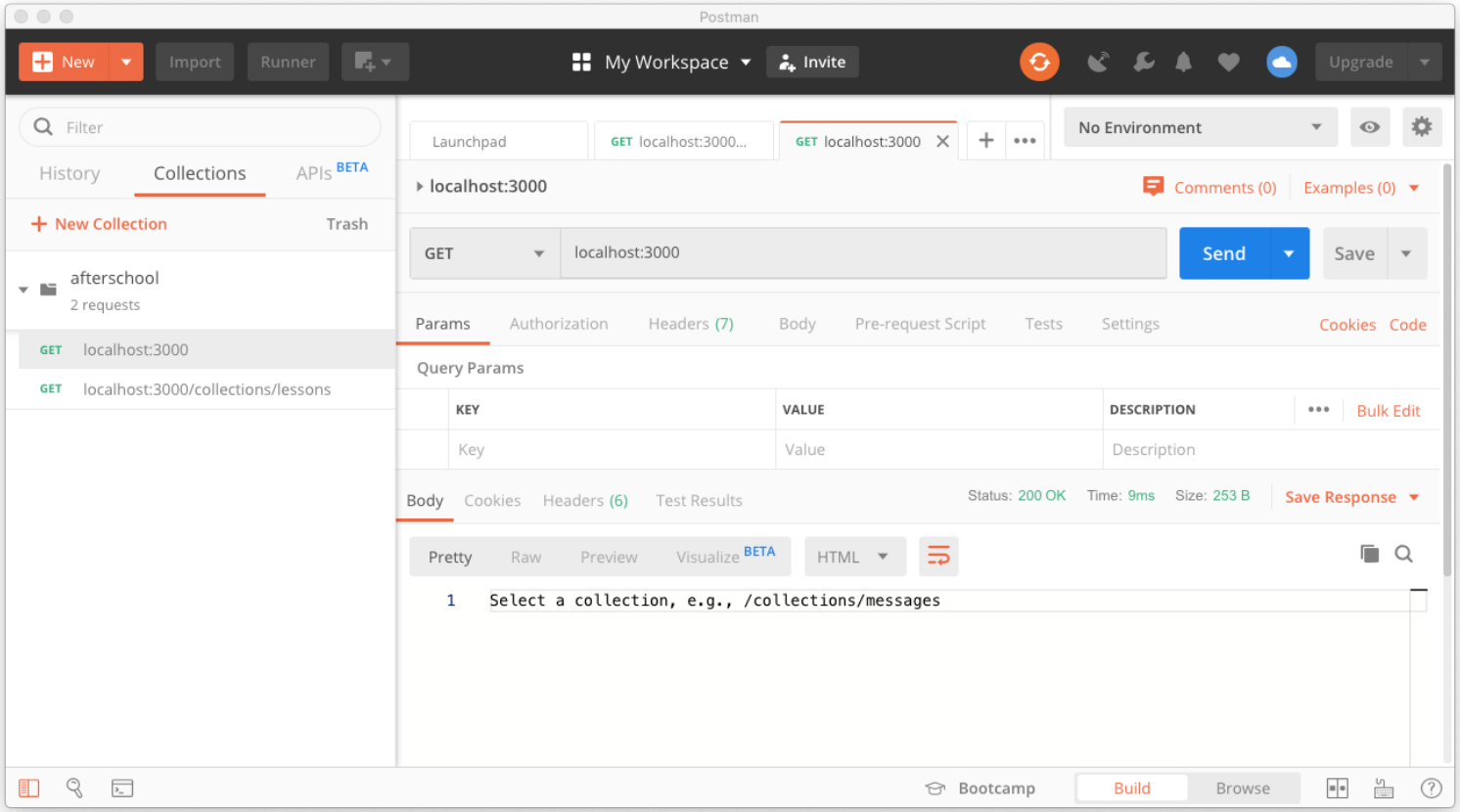
```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 3000 (#0)
> GET / HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/html; charset=utf-8
< Content-Length: 48
< ETag: W/"30-f8awmwEpi419CYjVnylSOlQULfU"
< Date: Fri, 15 Nov 2019 12:38:59 GMT
< Connection: keep-alive
<
* Connection #0 to host localhost left intact
Select a collection, e.g., /collection/messages* Closing connection 0
```

- The default HTTP method is `GET`
- To test other types of request, use `curl -X get/post/put/delete ...`
    - You can use `get` this way, too.
- You can also use the `-d` (data) flag for `POST`:

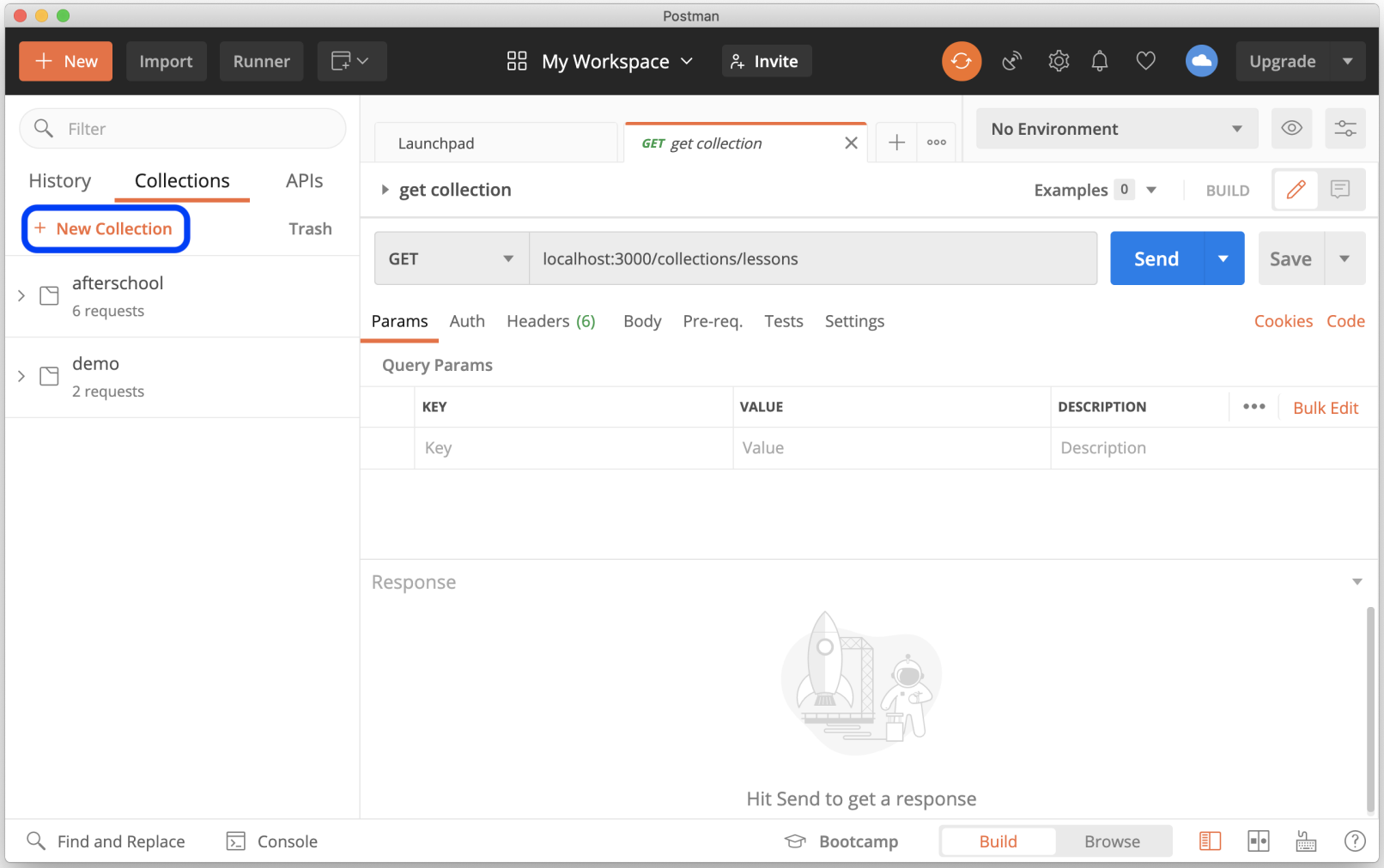
```
curl -d '{"topic":"web development", "location":"hendon", "price":200}'
     -H 'Content-Type: application/json'
     http://localhost:3000/collection/lessons
```

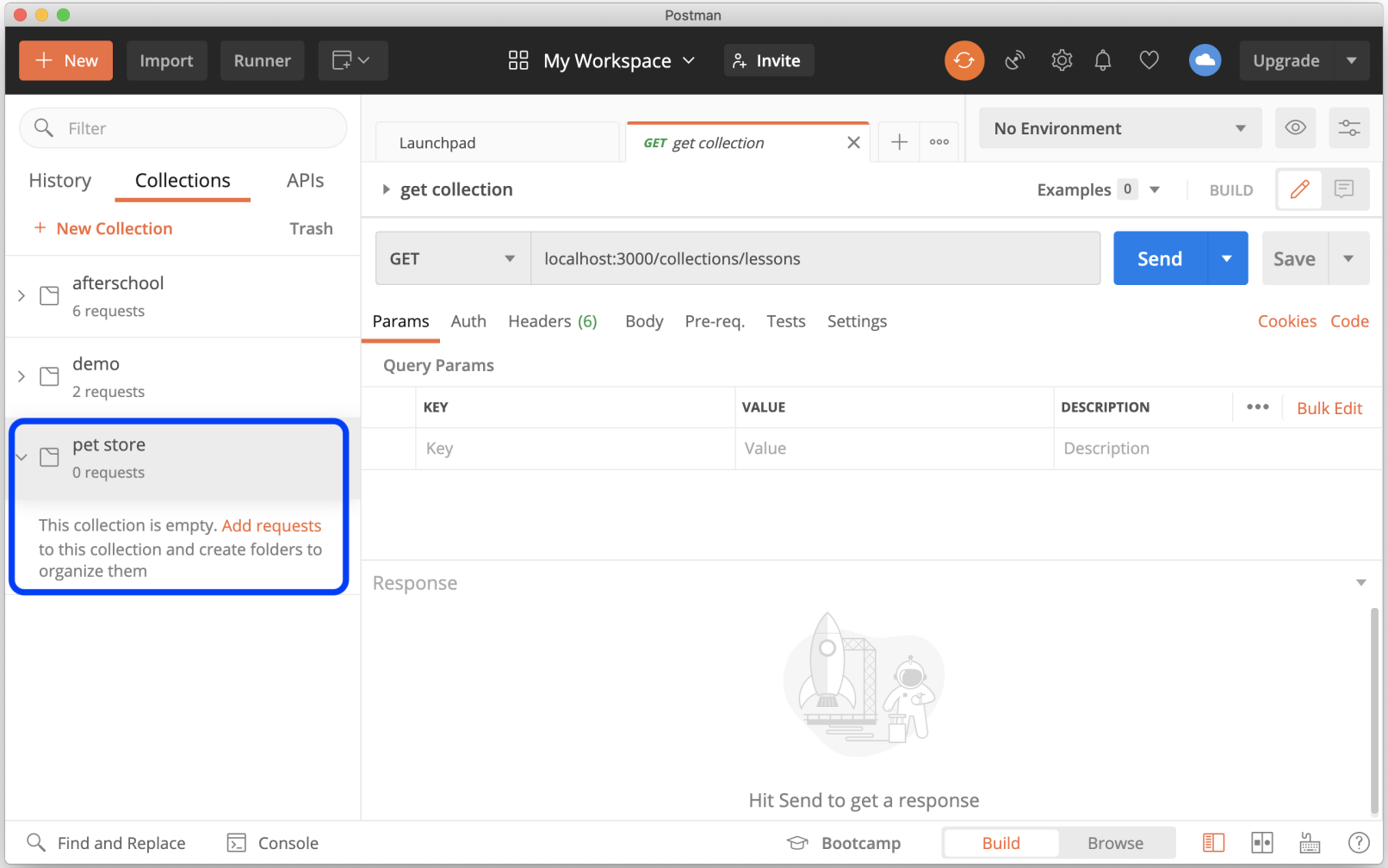
- The `-H` is a 'custom head' flag, which says the data/content is of json format.
- This can get complicated quite quickly.
    - Luckily, there is other more user-friendly tool available.

Postman (https://www.getpostman.com/)

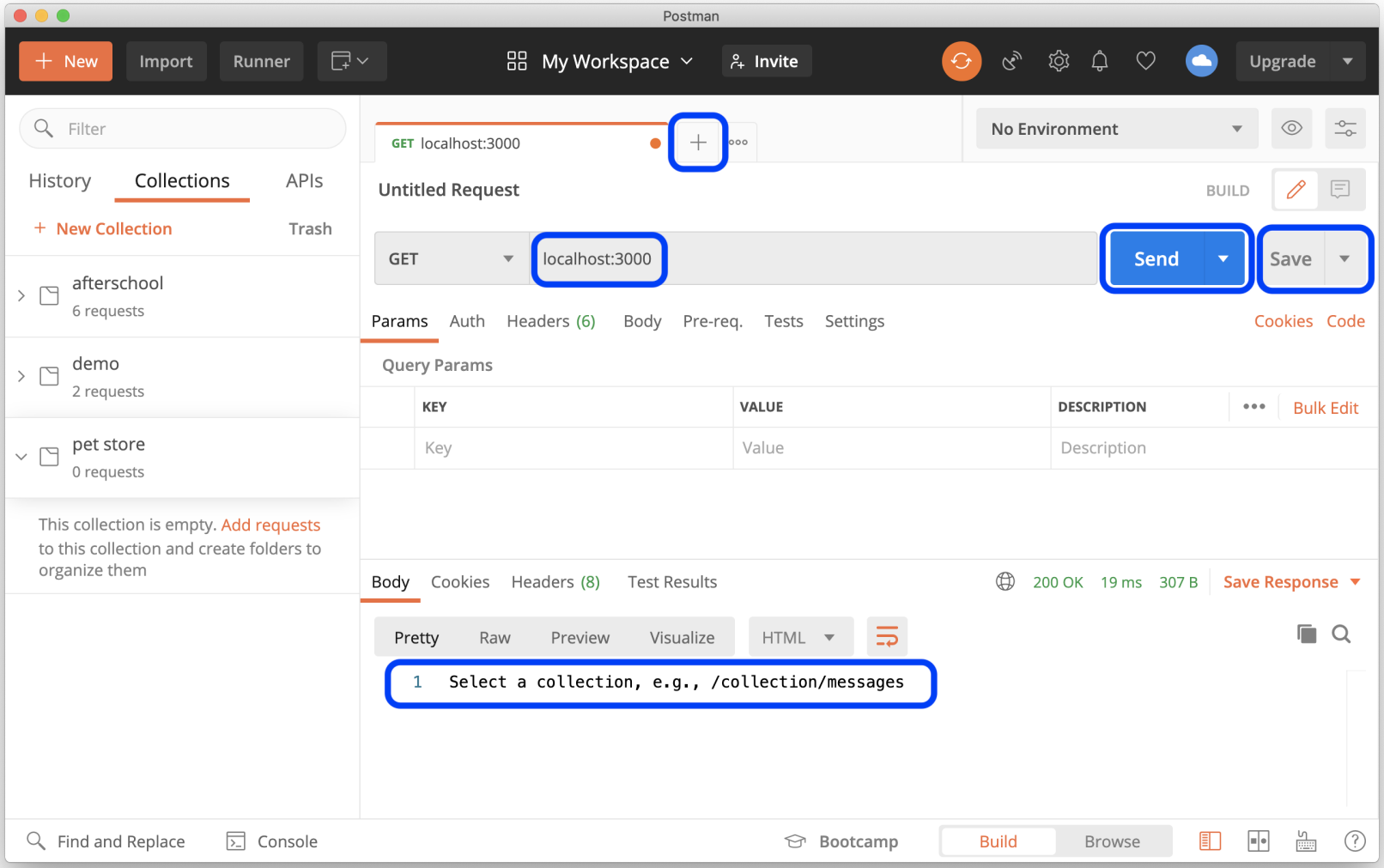


Create a new 'collection'

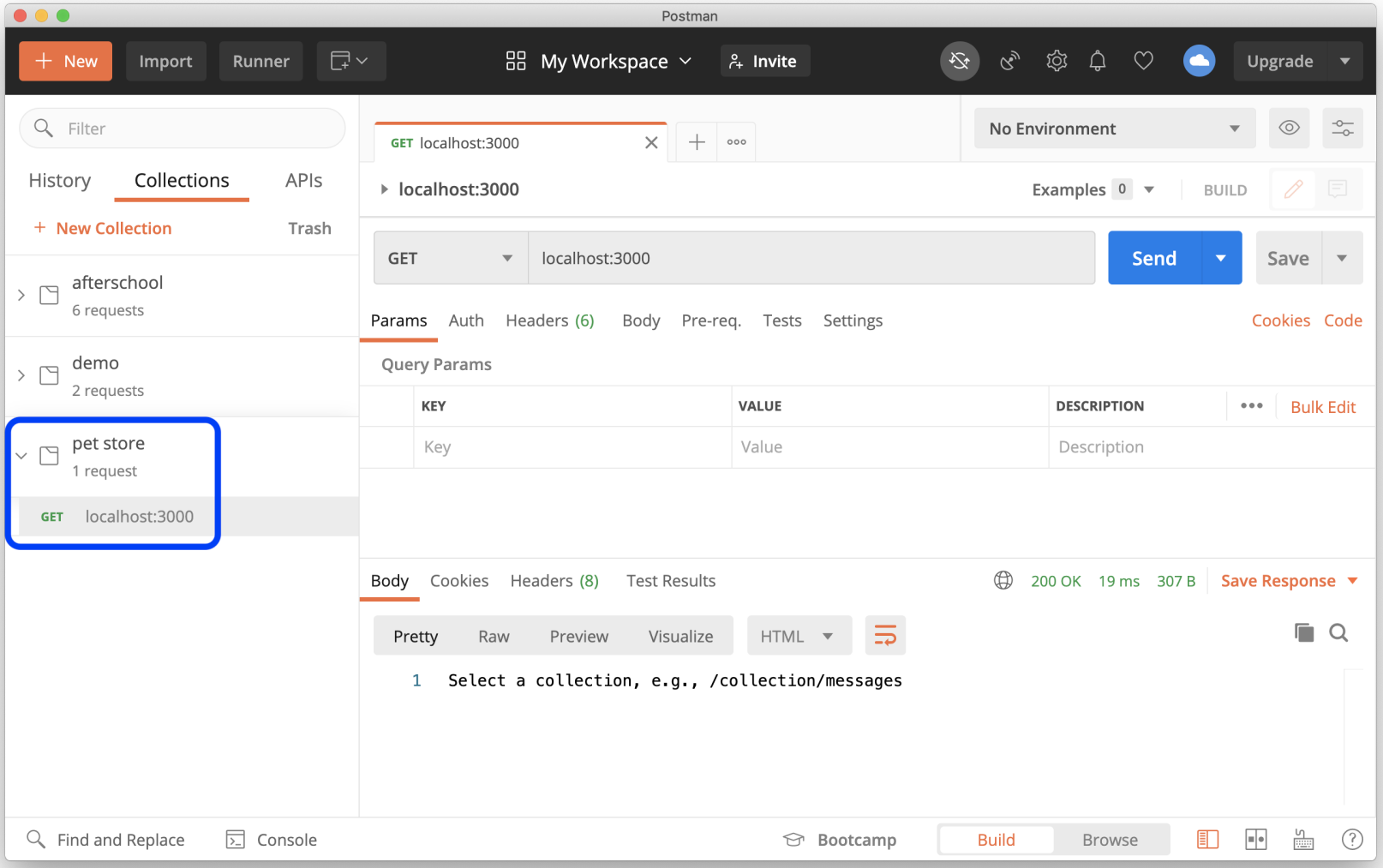



Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
New Collection
Trash
afterschool
6 requests
demo
2 requests
pet store
0 requests
This collection is empty. Add requests to this collection and create folders to organize them
Launchpad
GET get collection
No Environment
get collection
Examples 0
BUILD
GET
localhost:3000/collections/lessons
Send
Save
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Response
Hit Send to get a response
Find and Replace
Console
Bootcamp
Build
Browse

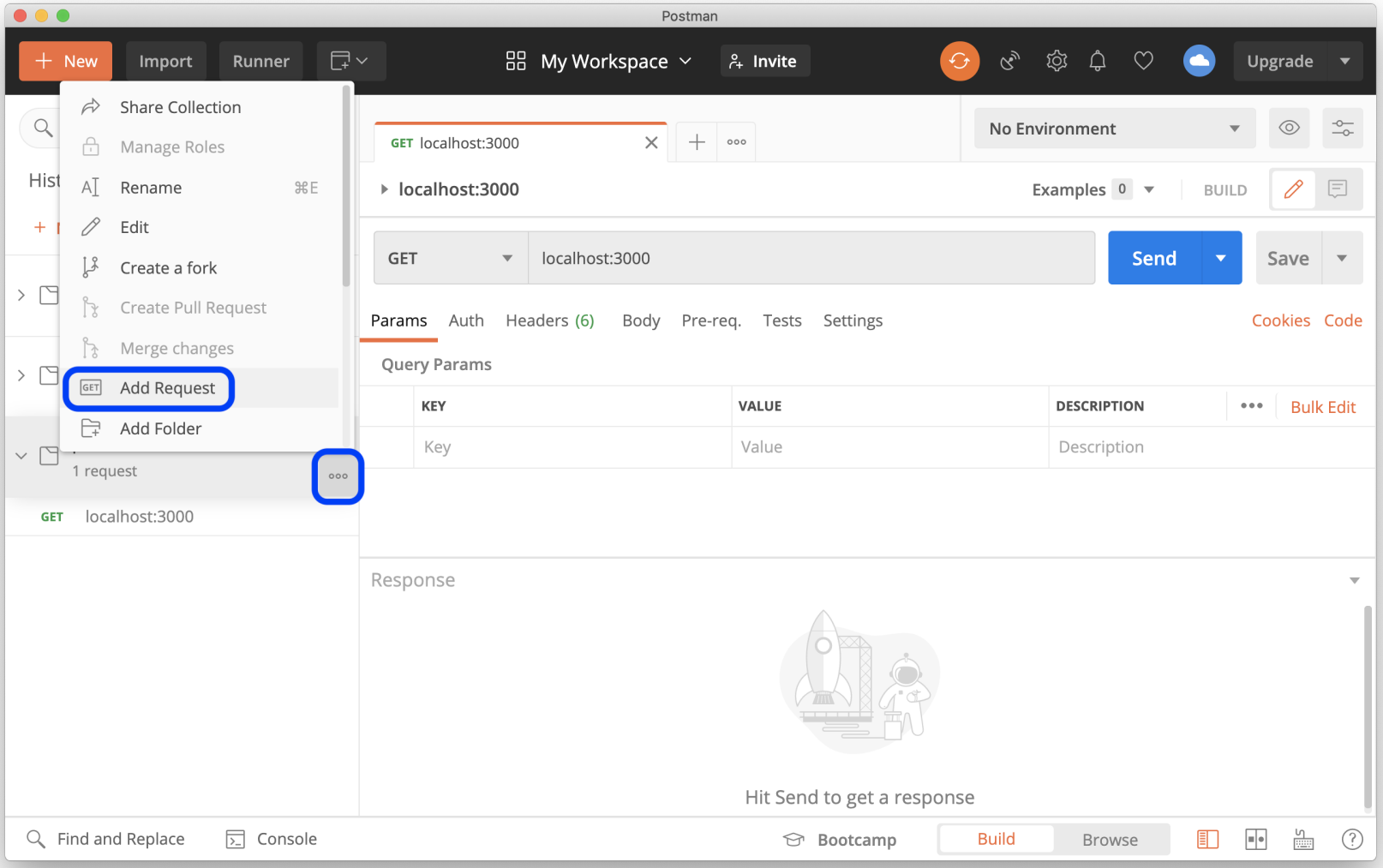
Add new (REST) request

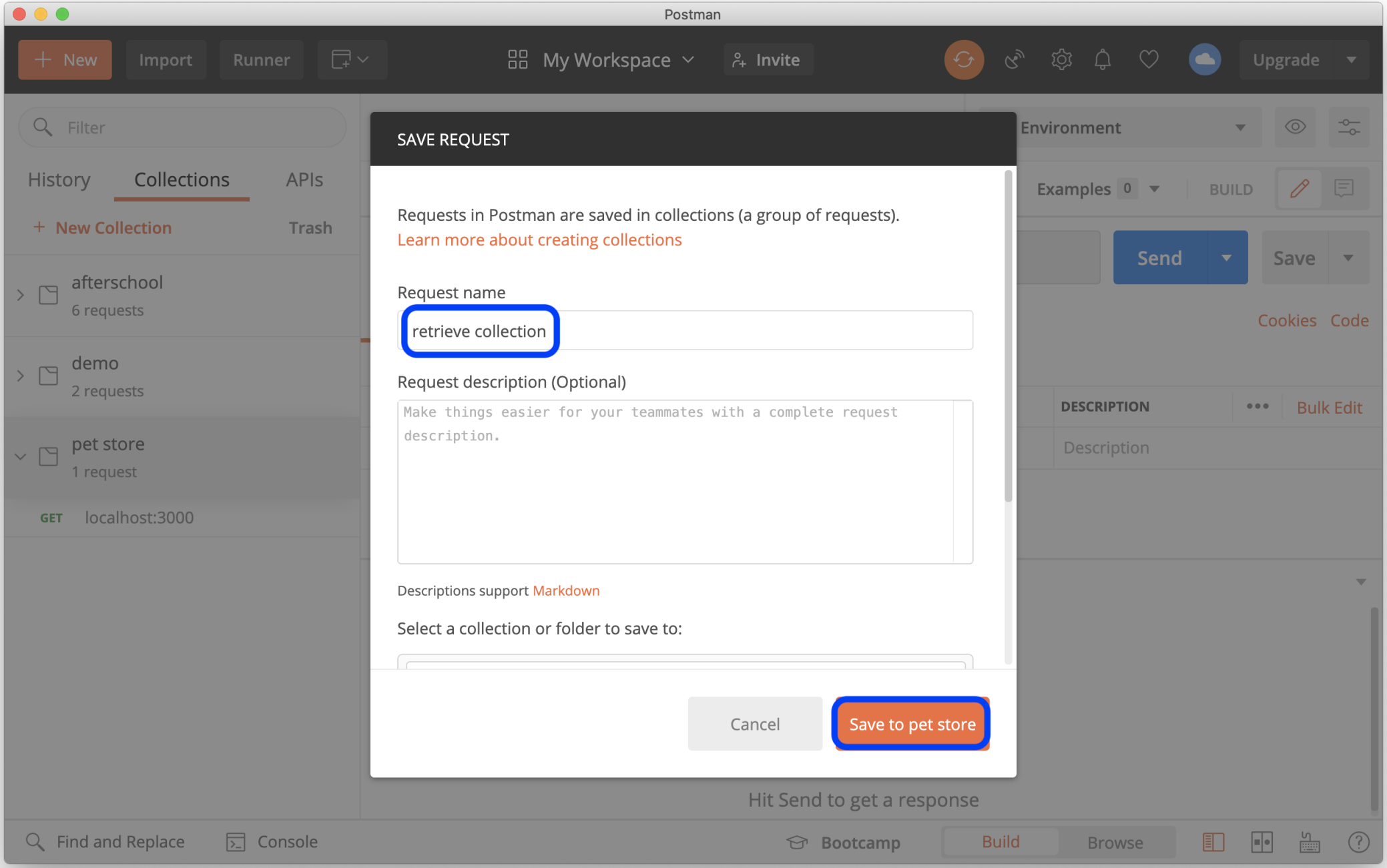


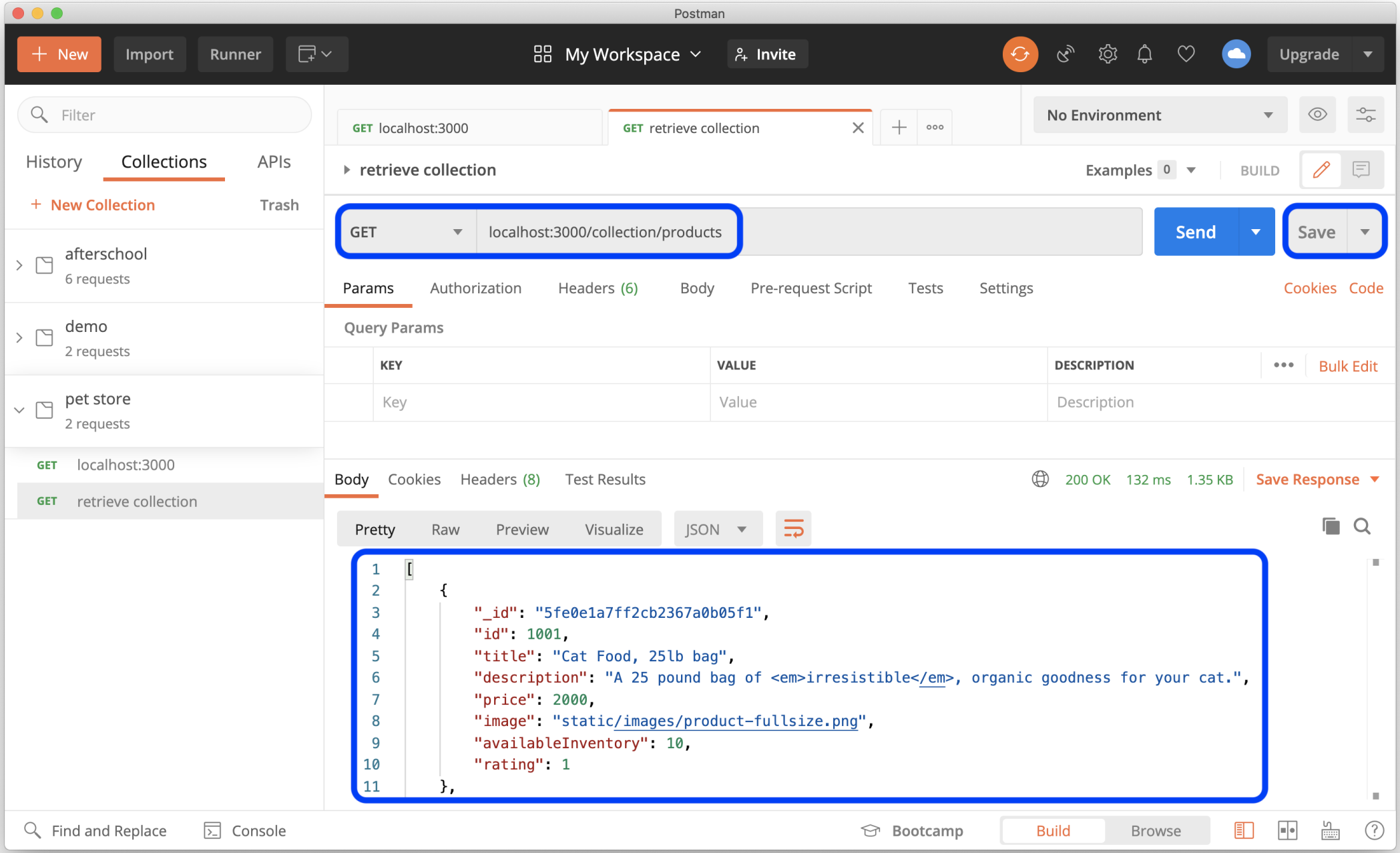
Request saved to 'collection'



Add a second request




Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
New Collection
Trash
afterschool
6 requests
demo
2 requests
pet store
1 request
GET
localhost:3000
SAVE REQUEST
Requests in Postman are saved in collections (a group of requests).
Learn more about creating collections
Request name
retrieve collection
Request description (Optional)
Make things easier for your teammates with a complete request description.
Descriptions support Markdown
Select a collection or folder to save to:
Cancel
Save to pet store
Environment
Examples 0
BUILD
Send
Save
Cookies
Code
DESCRIPTION
Bulk Edit
Description
Hit Send to get a response
Find and Replace
Console
Bootcamp
Build
Browse

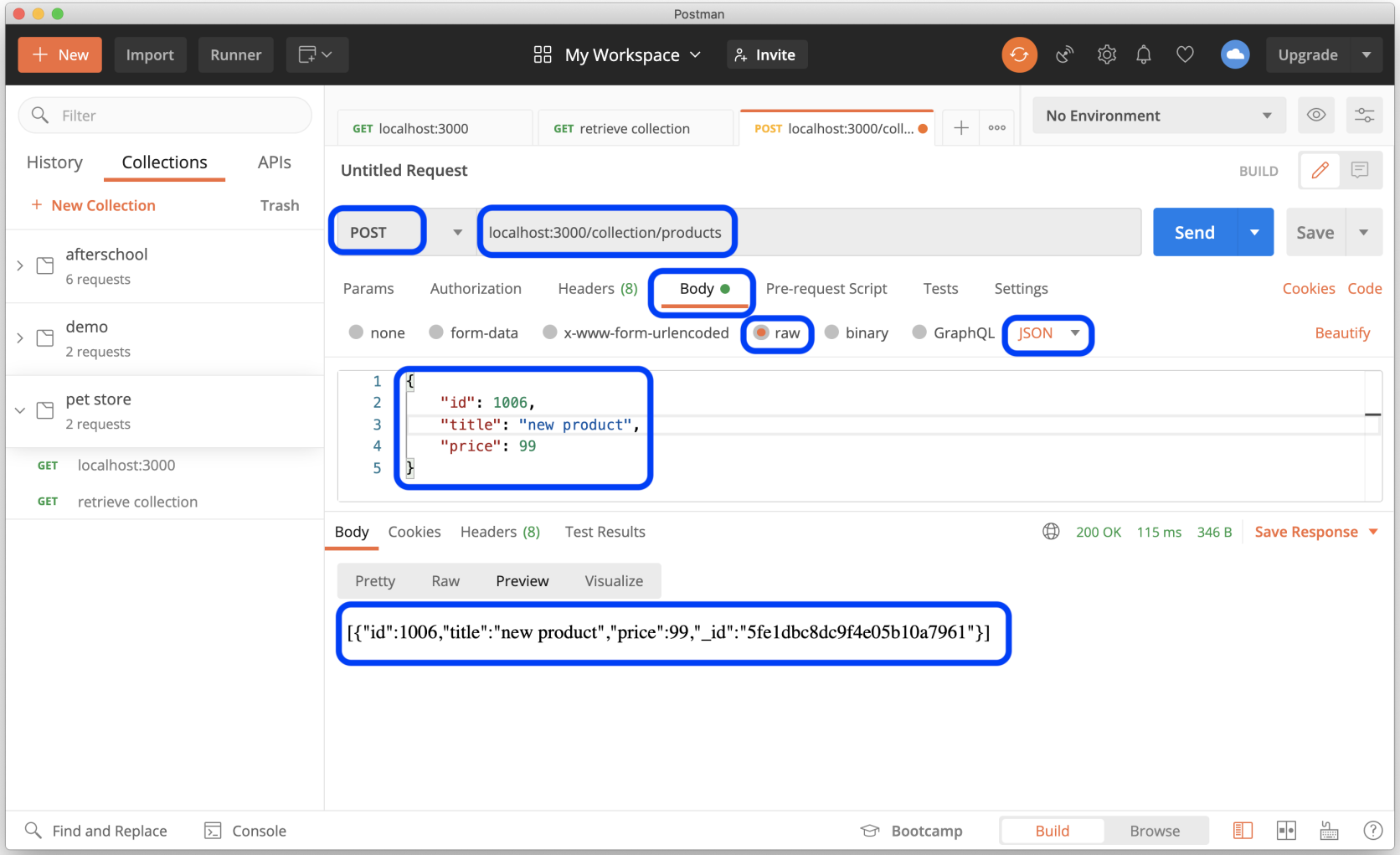

Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
New Collection
Trash
afterschool
6 requests
demo
2 requests
pet store
2 requests
GET localhost:3000
GET retrieve collection
GET localhost:3000
GET retrieve collection
No Environment
retrieve collection
Examples 0
BUILD
GET
localhost:3000/collection/products
Send
Save
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
132 ms
1.35 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 [
2 {
3 "_id": "5fe0e1a7ff2cb2367a0b05f1",
4 "id": 1001,
5 "title": "Cat Food, 25lb bag",
6 "description": "A 25 pound bag of irresistible, organic goodness for your cat.",
7 "price": 2000,
8 "image": "static/images/product-fullsize.png",
9 "availableInventory": 10,
10 "rating": 1
11 },
Find and Replace
Console
Bootcamp
Build
Browse

Creating object with `POST`

- We will use the same URL as before (`/collection/:collectionName`) but the http method changes to `post` .
- Here we pass the `request.body` , i.e., the json object to be inserted, to `req.collection.insert()` function.

```
app.post('/collection/:collectionName', (req, res, next) => {
  req.collection.insert(req.body, (e, results) => {
    if (e) return next(e)
    res.send(results.ops)
  })
})
```

Postman: `POST` request



- Change the method to 'POST' (from 'GET')
- Use the same URL: `localhost:3000/collection/products'
- We need to put the production information in the 'body' of the request
    - Select 'body'
    - Choose 'raw' (data type)
    - Set the data type to 'JSON'
    - Type the JSON object in the field below
- Click 'Send' and you should see the response in the bottom pane if successful.

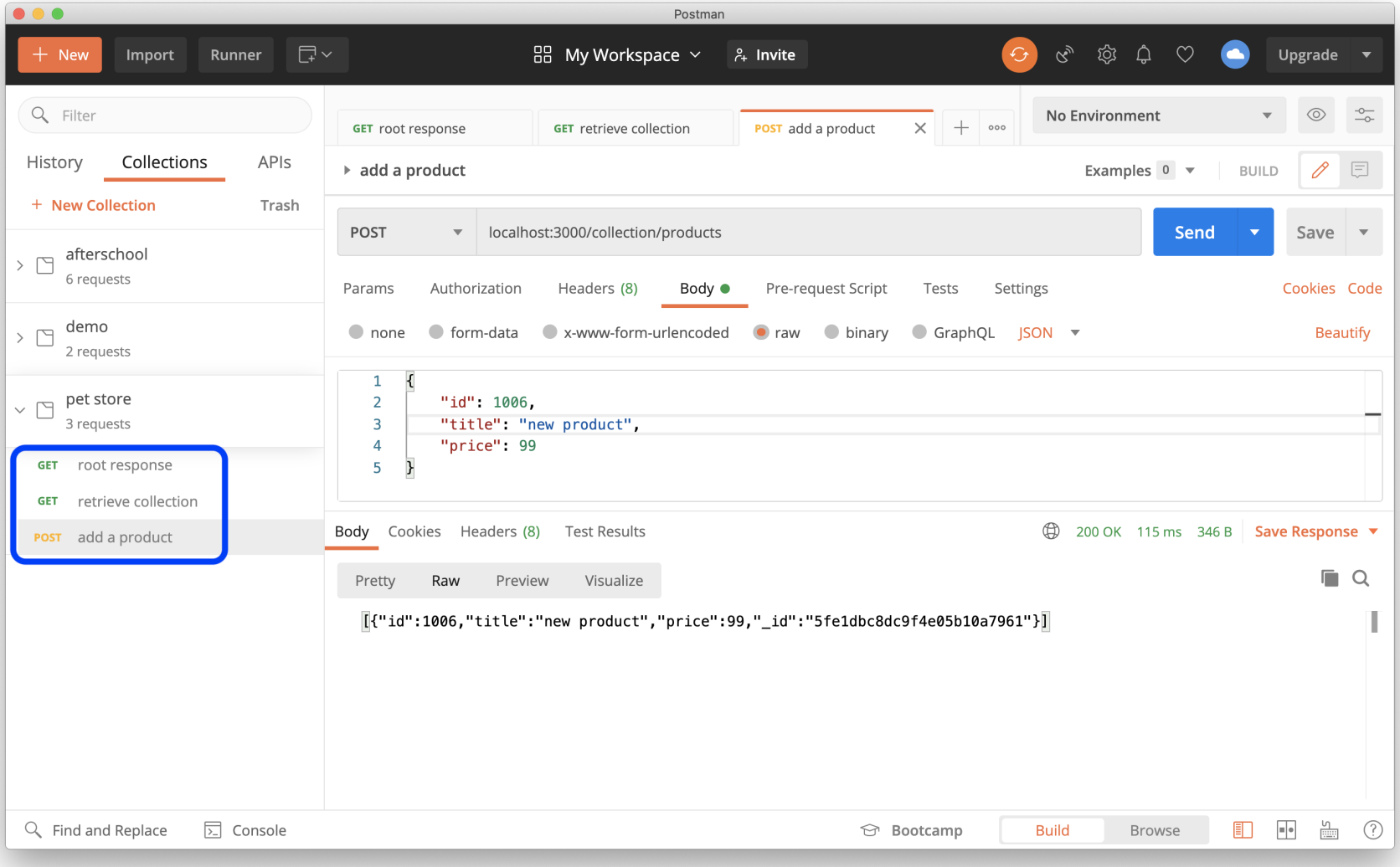
You should see the new product in Mongo Compass

- Not the product information is different from other products.
  - Because we did not enforce a data schema

```
_id: ObjectId("5fe0e1dfff2cb2367a0b05f5")
id: 1005
title: "Laser Pointer"
description: "Drive your cat crazy with this <em>amazing</em> product."
price: 4999
image: "static/images/laser-pointer.jpg"
availableInven…: 25
rating: 1
```

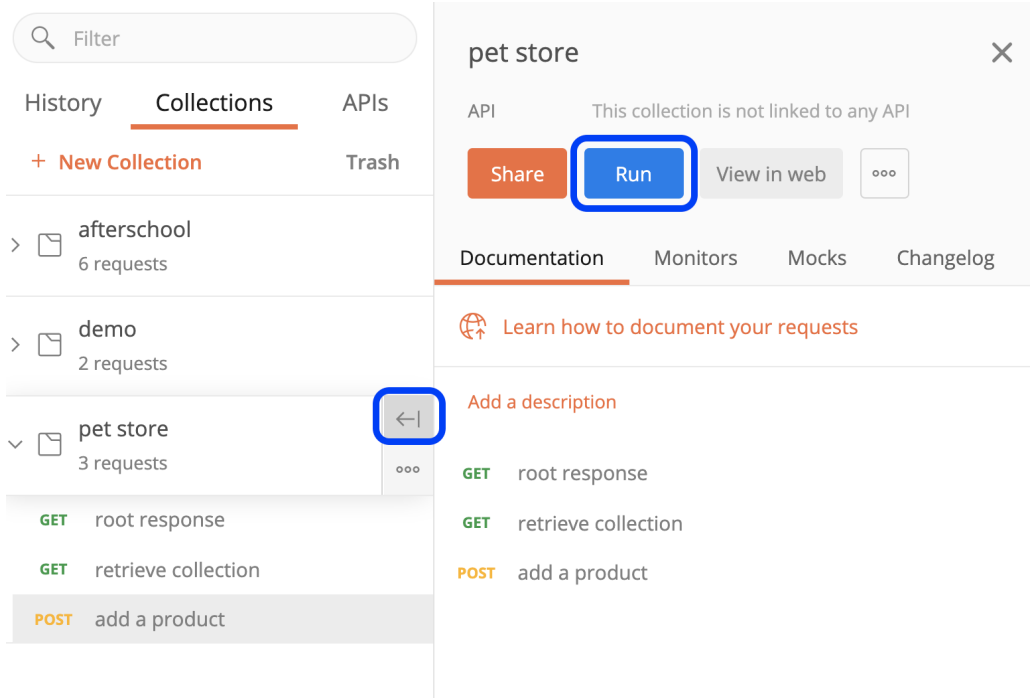
```
_id: ObjectId("5fe1dbc8dc9f4e05b10a7961")
id: 1006
title: "new product"
price: 99
```

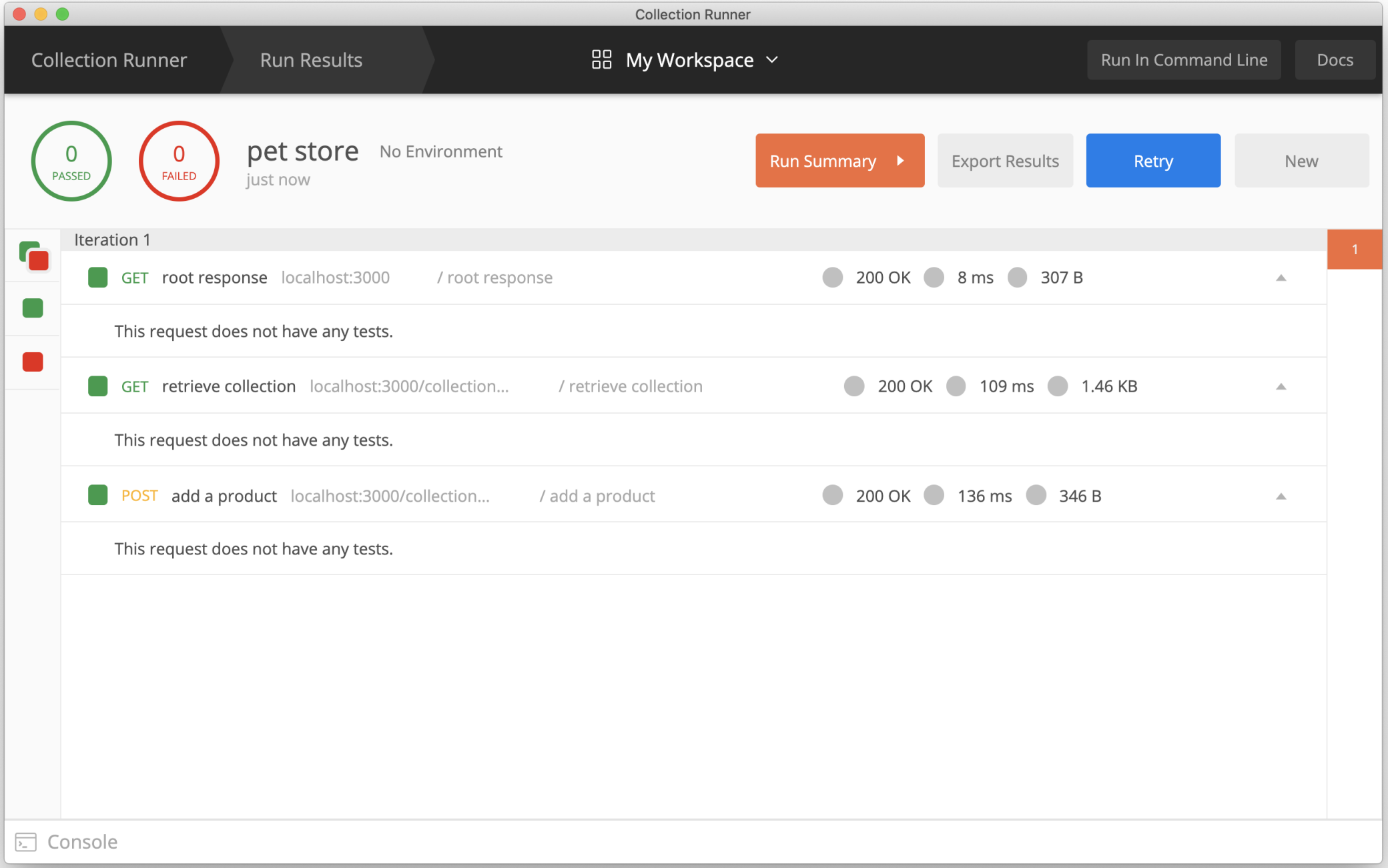
You should have 3 requests in the Postman collection



Re-run the requests

- You can now re-run any of the requests by selecting it in the collection and then 'Send'
  - You need this when you are creating your REST API
  - You want to do this to make sure changes in other requests don't affect this one.
  - Just be aware a new product will be added each time the request is sent
- You can even run all the request in one go




Collection Runner
Collection Runner
Run Results
My Workspace
Run In Command Line
Docs
0
PASSED
0
FAILED
pet store
No Environment
just now
Run Summary
Export Results
Retry
New
Iteration 1
1
GET root response localhost:3000 / root response 200 OK 8 ms 307 B
This request does not have any tests.
GET retrieve collection localhost:3000/collection... / retrieve collection 200 OK 109 ms 1.46 KB
This request does not have any tests.
POST add a product localhost:3000/collection... / add a product 200 OK 136 ms 346 B
This request does not have any tests.
Console

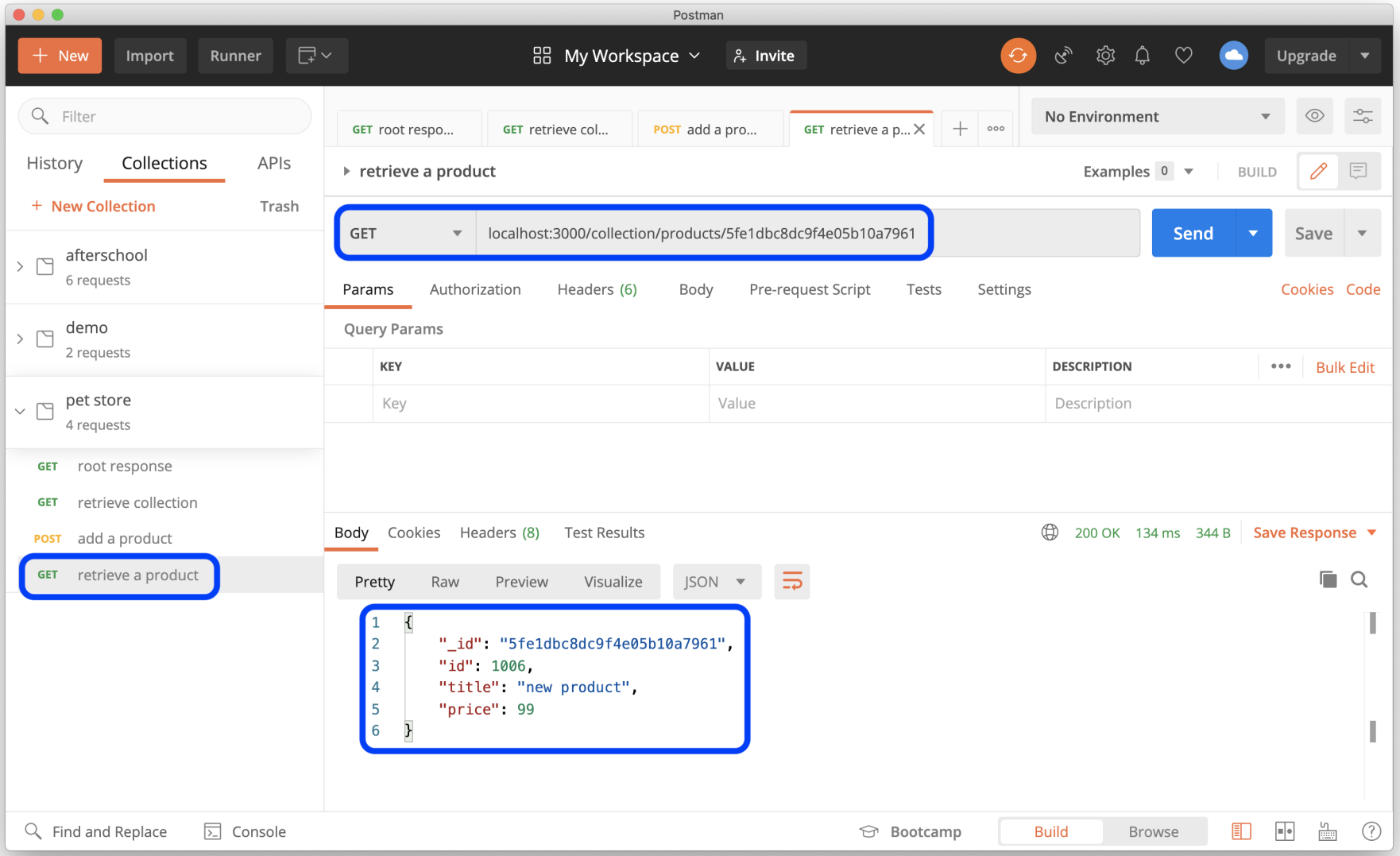
Retrieve an objecct with `GET`

- This time, instead of retrieving an entire MongoDB collection, we only want one object with a specific ID.
- The end point (route) URL will be `/collection/:collectionName/:id`, e.g., `/collection/messages/123`.
- The MongoDB function we will use is `findOne()`, which returns an object instead of a cursor (as `find()`), so we can drop the `toArray()` part.
- We extract the ID from the `:id` part of the URL path with `req.params.id`

```
const ObjectID = require('mongodb').ObjectID;
app.get('/collection/:collectionName/:id', (req, res, next) => {
    req.collection.findOne({ _id: new ObjectID(req.params.id) }, (e, result) => {
        if (e) return next(e)
        res.send(result)
    })
})
```

- The `ObjectID` (first line) is used to create the 'ObjectId' object in MongoDB,
- This is different from the ID string, such as '5dcff5220e9b3a036fa217af'.
- This is later used to generate the 'ObjectId' object in the `findOne` query.

- You need to copy the `objectID` from the Mongo Compass

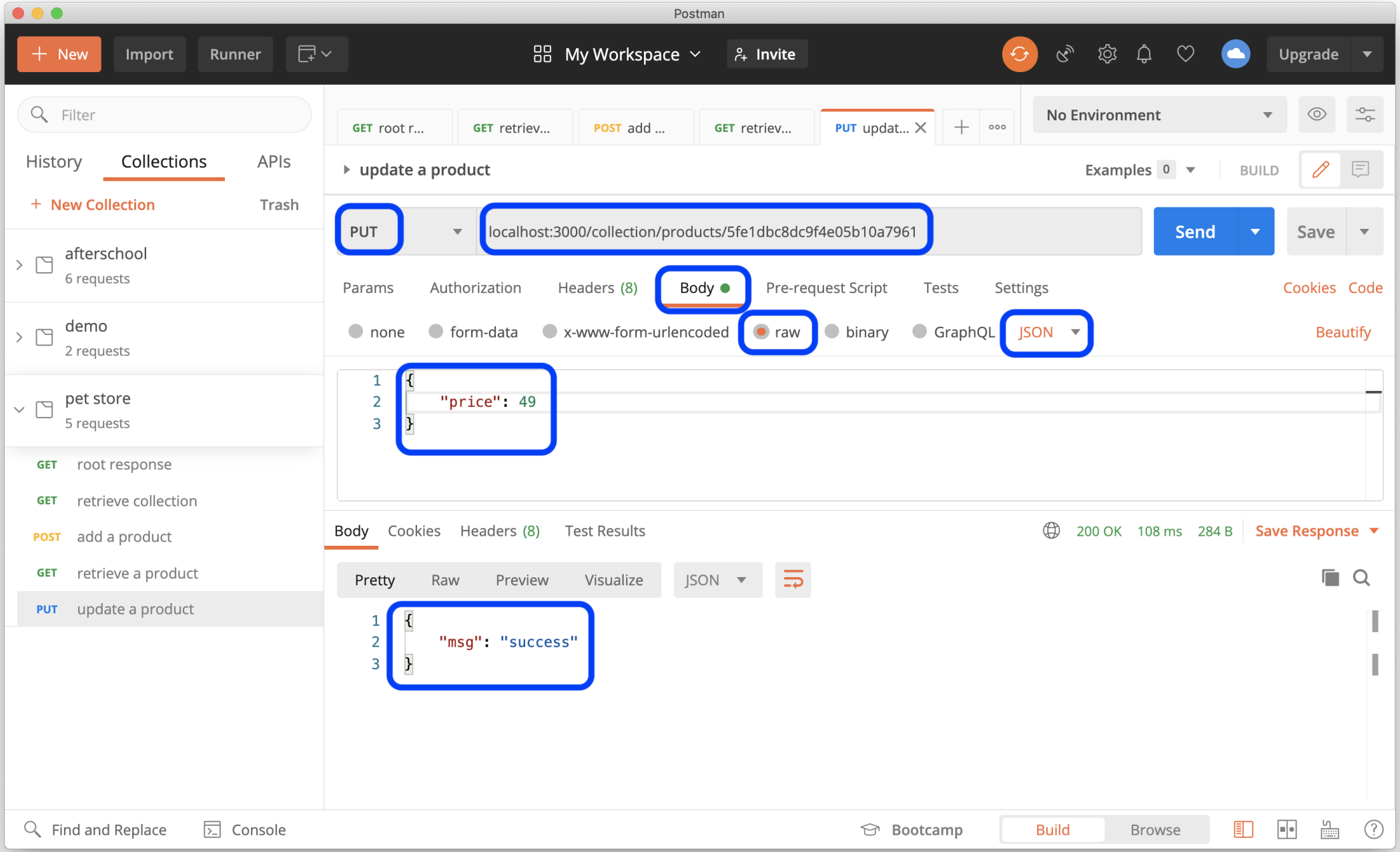


Update object with `PUT`

- The route is `/collection/:collectionName/:id`
  - The request needs to include both the collection name and the document (json object) ID
- In `update()`, we use `{$set:req.body}` to update object.
  - This replace the current document with `req.body`.
- It returns the count of affected objects,
  - This will be 1 when the matching JSON object is found (each document ID is unique).
  - This will be 0 when no matching JSON object is found.

```
app.put('/collection/:collectionName/:id', (req, res, next) => {
  req.collection.update(
    {_id: new ObjectID(req.params.id)},
    {$set: req.body},
    {safe: true, multi: false},
    (e, result) => {
      if (e) return next(e)
      res.send((result.result.n === 1) ? {msg: 'success'} : {msg: 'error'})
    })
})
```

- The second parameter `{safe:true, multi:false}` tells MongoDB to
  - wait for the execution before running the callback function and
  - to process only one (the first) item.
- It sends back 'success' if
  - the callback error is `null` and
  - the number of update documents is 1.


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
New Collection
Trash
afterschool
6 requests
demo
2 requests
pet store
5 requests
GET root response
GET retrieve collection
POST add a product
GET retrieve a product
PUT update a product
GET root r...
GET retriev...
POST add ...
GET retriev...
PUT updat...
No Environment
update a product
Examples 0
BUILD
PUT
localhost:3000/collection/products/5fe1dbc8dc9f4e05b10a7961
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"price": 49
}
Body
Cookies
Headers (8)
Test Results
200 OK
108 ms
284 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"msg": "success"
}
Find and Replace
Console
Bootcamp
Build
Browse

Only need to specify the field that changes

- In the last example, we only changed the price: `{"price": 49}`
- The other fields will remain the same

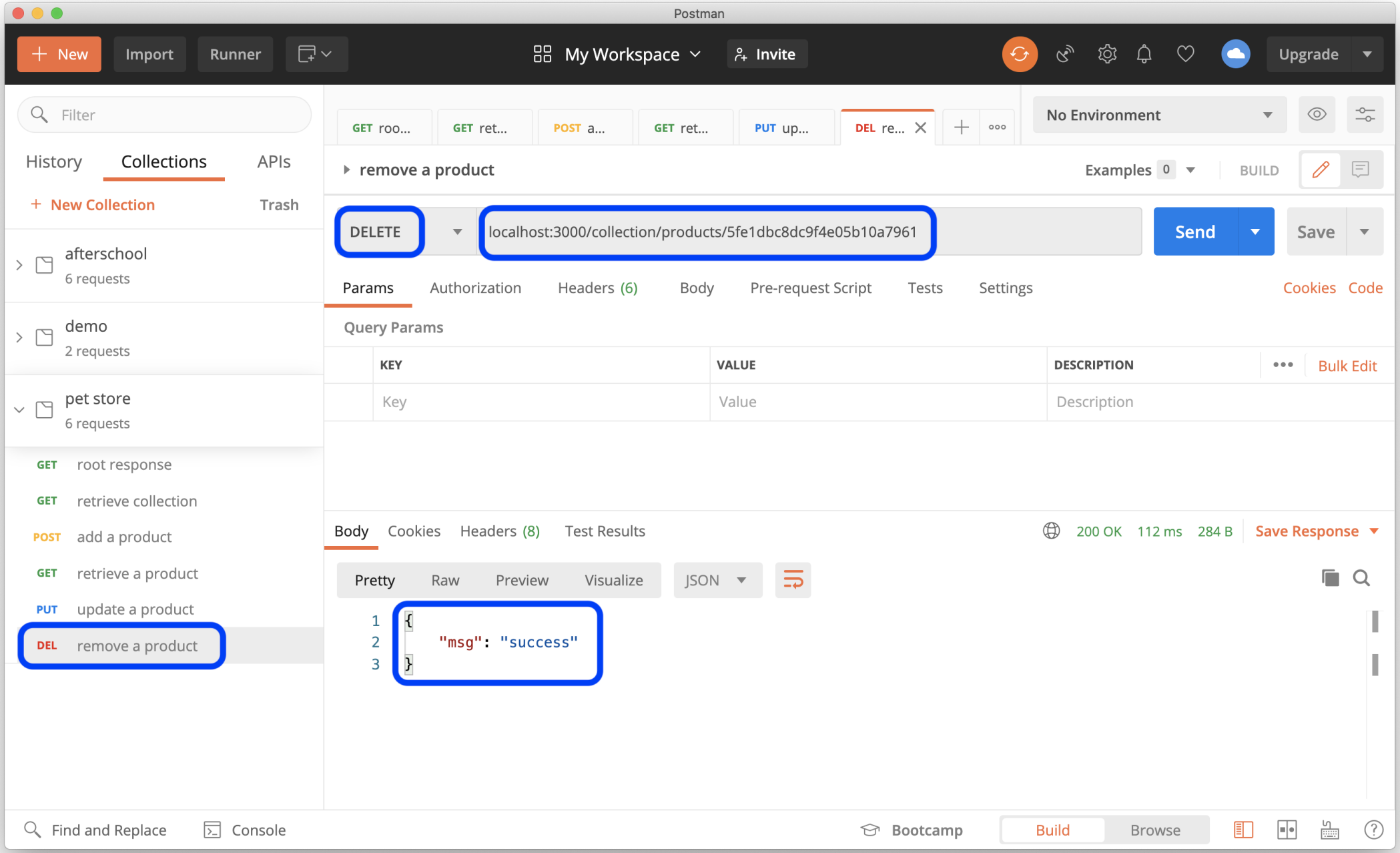
```
_id: ObjectId("5fe1dbc8dc9f4e05b10a7961")
id : 1006
title : " new product  "
price : 49
```

Remove object with `DELETE`

- Lastly, we define the `DELETE /collection/:collectionName/:id` route to remove one document (the `deleteOne` function).

```
app.delete('/collectione/:collectionName/:id', (req, res, next) => {
  req.collection.deleteOne(
    {_id: ObjectID(req.params.id)},
    (e, result) => {
        if (e) return next(e)
        res.send((result.result.n === 1) ? {msg: 'success'} : {msg: 'error'})
  })
})
```

- The callback will have two arguments, with the second having the `result` property. Thus we use `result.result`.
- In the callback, we create an `if/else` to output a custom message (`msg`), which is either `success` for 1 removed document, or `error` for a value different from (1).


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
New Collection
Trash
afterschool
6 requests
demo
2 requests
pet store
6 requests
GET root response
GET retrieve collection
POST add a product
GET retrieve a product
PUT update a product
DEL remove a product
GET roo...
GET ret...
POST a...
GET ret...
PUT up...
DEL re...
No Environment
remove a product
Examples 0
BUILD
DELETE
localhost:3000/collection/products/5fe1dbc8dc9f4e05b10a7961
Send
Save
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
112 ms
284 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "msg": "success"
3 }
Find and Replace
Console
Bootcamp
Build
Browse

All the code together

```
// load Express.js
const express = require('express')
const app = express()

// parse the request parameters
app.use(express.json())

// connect to MongoDB
const MongoClient = require('mongodb').MongoClient;
let db;
MongoClient.connect('mongodb://localhost:27017/', (err, client) => {
    db = client.db('webstore')
})

// get the collection name
app.param('collectionName', (req, res, next, collectionName) => {
    req.collection = db.collection(collectionName)
    // console.log('collection name:', req.collection)
    return next()
})
```

```
// dispaly a message for root path to show that API is working
app.get('/', function (req, res) {
    res.send('Select a collection, e.g., /collection/messages')
})

// retrieve all the objects from an collection
app.get('/collection/:collectionName', (req, res) => {
    req.collection.find({}).toArray((e, results) => {
        if (e) return next(e)
        res.send(results)
    })
})
```

```
// retrieve an object by mongodb ID
const ObjectID = require('mongodb').ObjectID;
app.get('/collection/:collectionName/:id', (req, res, next) => {
    req.collection.findOne(
        { _id: new ObjectID(req.params.id) },
        (e, result) => {
            if (e) return next(e)
            res.send(result)
    })
})

// add an object
app.post('/collection/:collectionName', (req, res, next) => {
    req.collection.insert(req.body, (e, results) => {
        if (e) return next(e)
        res.send(results.ops)
    })
})
```

```
// update an object by ID
app.put('/collection/:collectionName/:id', (req, res, next) => {
    req.collection.update(
        { _id: new ObjectID(req.params.id) },
        { $set: req.body },
        { safe: true, multi: false },
        (e, result) => {
            if (e) return next(e)
            res.send((result.result.n === 1) ?
                {msg: 'success'} : { msg: 'error'})
    })
})
```

```
// delete an object by ID
app.delete('/collection/:collectionName/:id', (req, res, next) => {
    req.collection.deleteOne(
        { _id: ObjectID(req.params.id) },
        (e, result) => {
            if (e) return next(e)
            res.send((result.result.n === 1) ?
                {msg: 'success'} : {msg: 'error'})
    })
})

app.listen(3000)
```

Reading

Practical Node.js - Chapter 8

MongoDB Node.js driver tutorial