

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 packages

- 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

The screenshot shows the MongoDB Atlas interface. At the top, there's a navigation bar with a leaf icon, a dropdown for 'Kai's Org - 2020-02-19', a gear icon for 'Access Manager', 'Support', and 'Billing' links, and 'All Clusters' and 'Kai' dropdowns. Below the navigation is a secondary navigation bar with 'Project 0', a dropdown, three dots, 'Atlas' (which is highlighted with a green underline), 'Realm', 'Charts', and user icons.

The main content area has a left sidebar under 'DATA STORAGE' with 'Clusters' (selected and highlighted in green), 'Triggers', and 'Data Lake'. Under 'SECURITY' are 'Database Access', 'Network Access', and 'Advanced'.

The main panel title is 'Clusters' and the subtitle is 'KAIS ORG - 2020-02-19 > PROJECT 0'. It shows a list of clusters:

- Cluster0** (Version 4.2.11) is highlighted with a blue box. Its 'Metrics' and 'Collections' tabs are visible, and the 'Connect Cluster' button is circled in orange.
- SANDBOX** (Version 4.2.11) is also listed.

A callout box for Cluster0 states: 'This is a Shared Tier Cluster. If you need a database that's better for high-performance production applications, upgrade to a dedicated cluster.' It includes an 'Upgrade' button.

Metrics and status cards are displayed on the right:

- Operations R: 0 W: 0** (100.0/s) Last 6 Hours
- Logical Size 0.0 B** (512.0 MB max) Last 30 Days
- Connections 6** (500 max) Last 6 Hours

- Choose 'Connect your application'

Connect to Cluster0

✓ Setup connection security

Choose a connection method

Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.



Connect with the mongo shell

Interact with your cluster using MongoDB's interactive Javascript interface



Connect your application

Connect your application to your cluster using MongoDB's native drivers



Connect using MongoDB Compass

Explore, modify, and visualize your data with MongoDB's GUI



Go Back

Close

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

Connect to Cluster0

The screenshot shows the MongoDB connection setup interface. At the top, there are three green checkmark icons with arrows pointing right: 'Setup connection security', 'Choose a connection method', and 'Connect'. Below this, step 1 is titled 'Select your driver and version' with dropdown menus for 'DRIVER' (set to 'Node.js') and 'VERSION' (set to '3.6 or later'). Step 2 is titled 'Add your connection string into your application code' and includes an unchecked checkbox for 'Include full driver code example'. A connection string 'mongodb+srv://kai:<password>@cluster0.4cxsh.mongodb.net/<dbname>?retryWrites=' is displayed in a text input field with a 'Copy' button. A note below it says: 'Replace <password> with the password for the kai user. Replace <dbname> with the name of the database that connections will use by default. Ensure any option params are URL encoded.' Both the connection string and the note are highlighted with orange rounded rectangles.

✓ Setup connection security ✓ Choose a connection method Connect

1 Select your driver and version

DRIVER VERSION

Node.js 3.6 or later

2 Add your connection string into your application code

Include full driver code example

mongodb+srv://kai:<password>@cluster0.4cxsh.mongodb.net/<dbname>?retryWrites=

Copy

Replace <password> with the password for the kai user. Replace <dbname> with the name of the database that connections will use by default. Ensure any option params are URL encoded.

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-f8awmwEpi419CYjVny\lS0lQULfU"
< 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
```

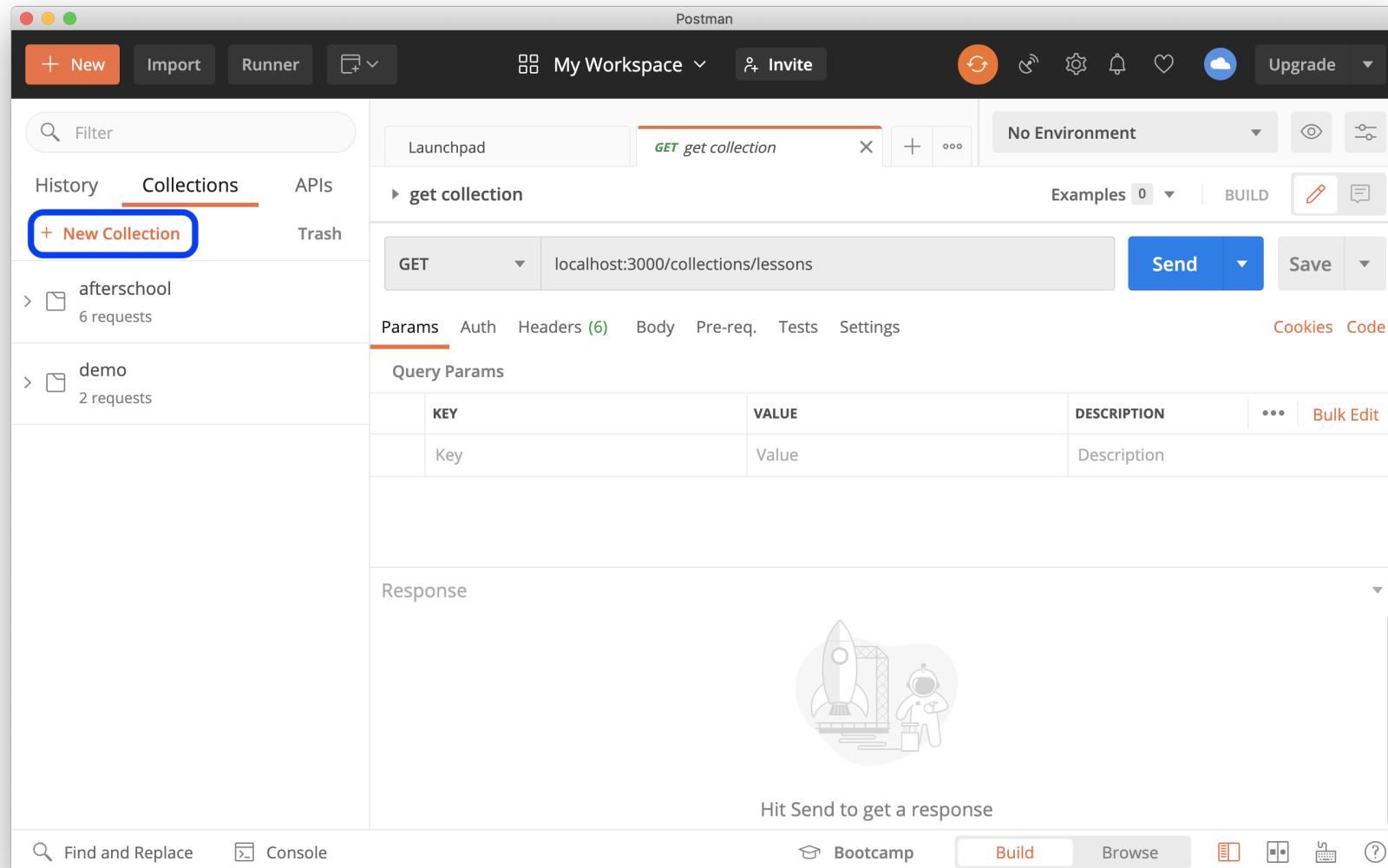
- The `-H` is a 'custom head' flag, which says the data/content is of json format.
- This can get complicated quite quickly.
 - Luckily, there is other more user-friendly tool available.

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



A screenshot of the Postman application interface. The top navigation bar includes "New", "Import", "Runner", "My Workspace" (selected), "Invite", "Upgrade", and various status icons. The left sidebar shows "History", "Collections" (selected), "APIs BETA", and a "New Collection" button. Under "Collections", there is a folder named "afterschool" containing 2 requests: "localhost:3000" and "localhost:3000/collections/lessons". The main workspace displays a collection titled "localhost:3000" with a single GET request to "localhost:3000". The request details show "Params", "Authorization", "Headers (7)", "Body", "Pre-request Script", "Tests", "Settings", "Cookies", and "Code" tabs. The "Params" tab is active, showing a table with one row: "Key" (Value) and "Value" (Description). Below the request details, the "Body" tab is selected, showing "Pretty", "Raw", "Preview", "Visualize BETA", and "HTML" options. A message at the bottom says "1 Select a collection, e.g., /collections/messages". The bottom navigation bar includes "Bootcamp", "Build" (selected), "Browse", and other icons.

Create a new 'collection'



The screenshot shows the Postman application interface. The top navigation bar includes 'New', 'Import', 'Runner', 'My Workspace' (selected), 'Invite', and 'Upgrade'. The left sidebar has tabs for 'History', 'Collections' (selected), 'APIs', and 'Trash', with a button '+ New Collection' highlighted by a blue box. Below the sidebar are two collections: 'afterschool' (6 requests) and 'demo' (2 requests). The main workspace shows a 'Launchpad' collection with a 'get collection' endpoint. The endpoint details are: Method: GET, URL: localhost:3000/collections/lessons, Buttons: 'Send' (blue) and 'Save'. Below the URL are tabs for 'Params', 'Auth', 'Headers (6)', 'Body', 'Pre-req.', 'Tests', 'Settings', 'Cookies', and 'Code'. A 'Query Params' table is present with columns: KEY, VALUE, DESCRIPTION, and 'Bulk Edit'. The table has one row: Key (Value) and Description. The bottom section is labeled 'Response' with a placeholder message 'Hit Send to get a response' and a small rocket ship icon. The footer contains 'Find and Replace', 'Console', 'Bootcamp' (selected), 'Build' (highlighted in orange), 'Browse', and other icons.

Postman

+ New Import Runner ↗ My Workspace Invite Upgrade

Filter

History Collections APIs + New Collection Trash

Launchpad **get collection** X No Environment Examples 0 BUILD

▶ get collection

GET localhost:3000/collections/lessons Send Save

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies Code

Query Params

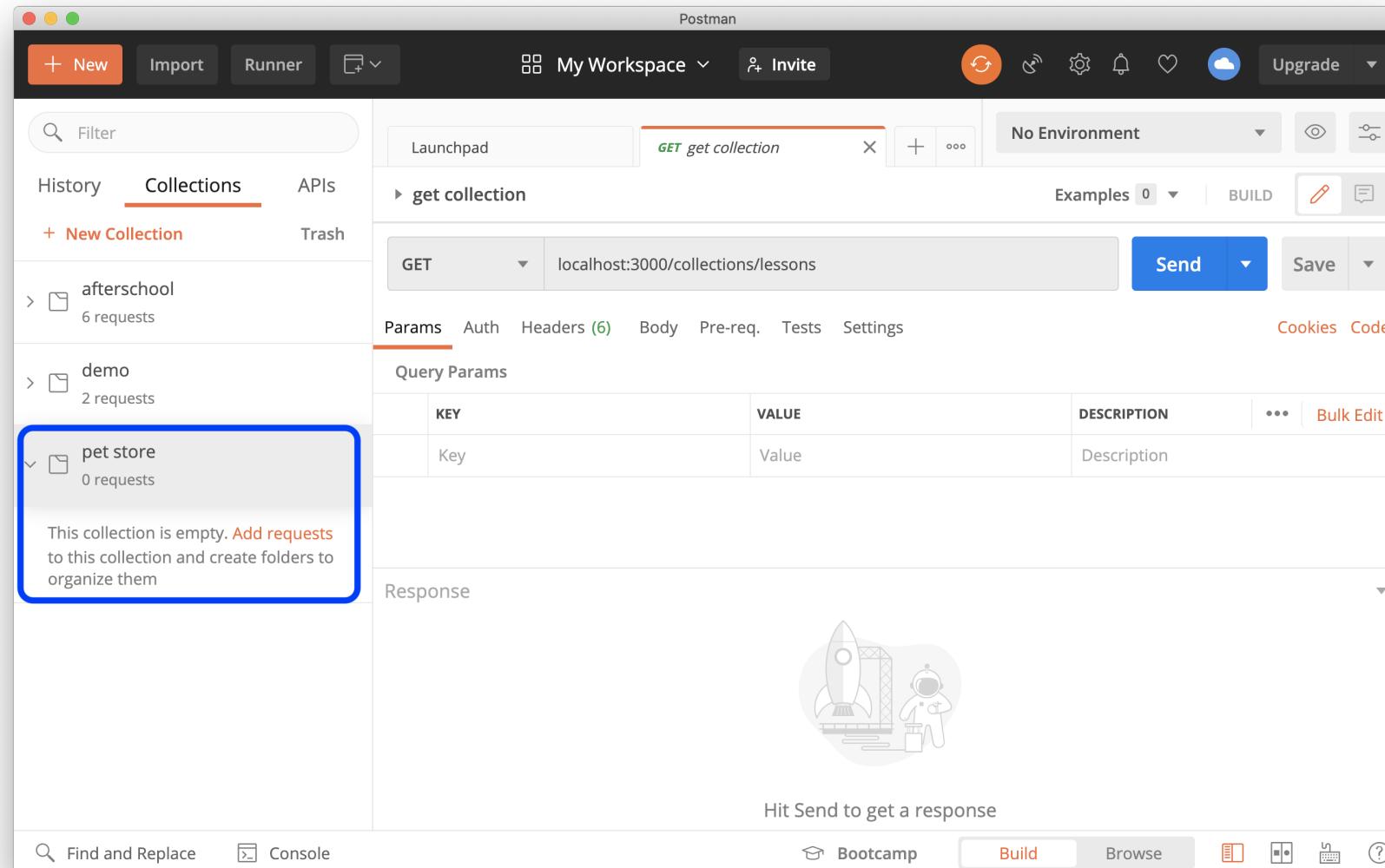
KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

This collection is empty. [Add requests](#) to this collection and create folders to organize them

Response

Hit Send to get a response

Find and Replace Console Bootcamp Build Browse



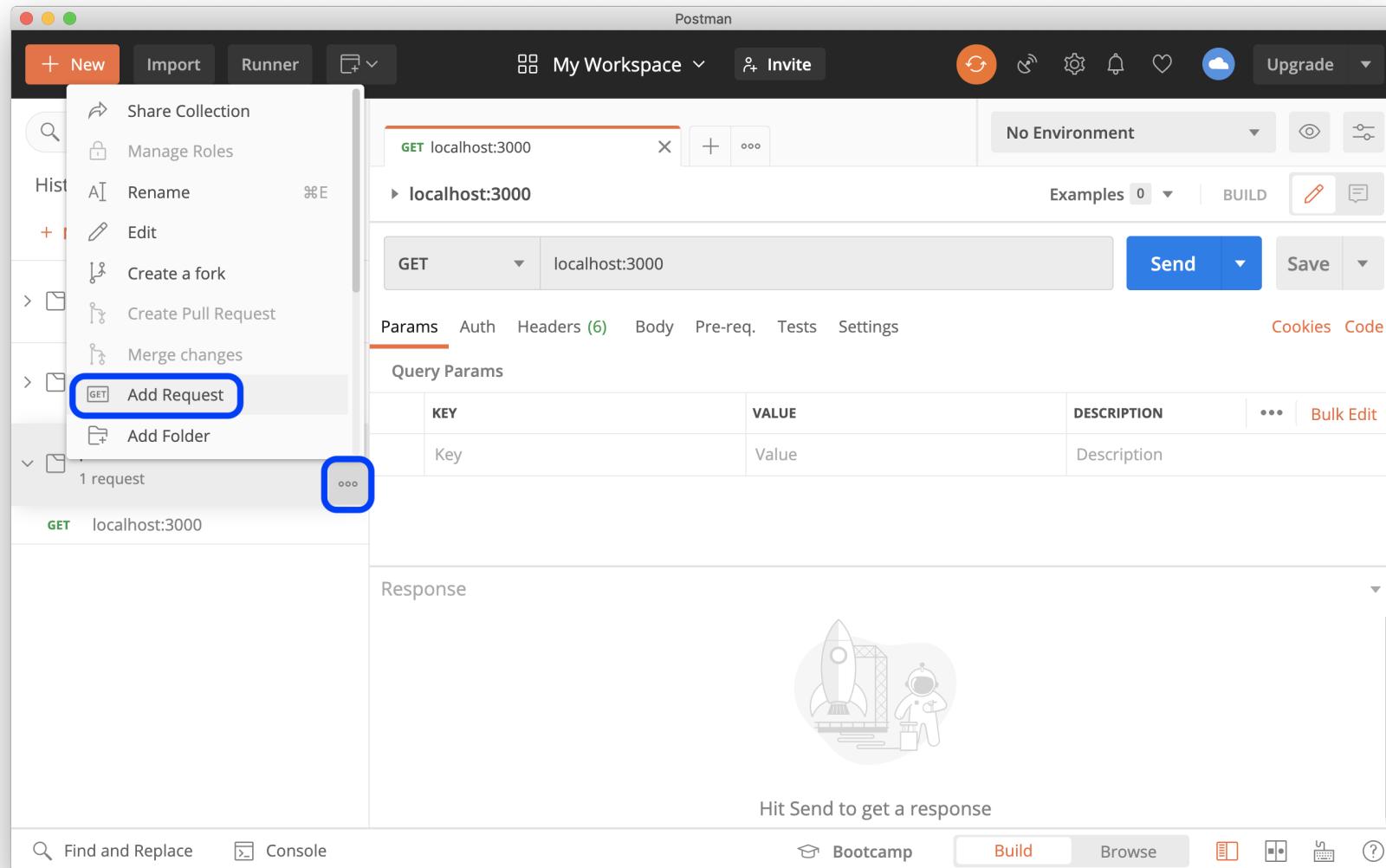
Add new (REST) request

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with collections like 'afterschool' (6 requests), 'demo' (2 requests), and 'pet store' (0 requests). A message indicates the collection is empty and prompts to 'Add requests'. The main workspace shows an 'Untitled Request' for a 'GET' method to 'localhost:3000'. The 'Send' and 'Save' buttons are highlighted with blue boxes. The 'Params' tab is selected, showing a table for 'Query Params' with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. The 'Body' tab is also visible. At the bottom, a search bar contains the placeholder 'Select a collection, e.g., /collection/messages'. The bottom navigation bar includes 'Find and Replace', 'Console', 'Bootcamp', 'Build' (highlighted in orange), 'Browse', and other icons.

Request saved to 'collection'

The screenshot shows the Postman application interface. The left sidebar is titled 'Collections' and lists three collections: 'afterschool' (6 requests), 'demo' (2 requests), and 'pet store' (1 request). The 'pet store' collection is highlighted with a blue border. A request for 'localhost:3000' is selected within the 'pet store' collection. The main workspace displays a 'GET' request to 'localhost:3000'. The 'Params' tab is active, showing a table for 'Query Params' with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. The table has one row with 'Key' and 'Value' fields. Below the table, tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results' are visible. At the bottom, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'HTML'. The status bar at the bottom shows '200 OK 19 ms 307 B' and a 'Save Response' button.

Add a second request



The screenshot shows the Postman application interface. On the left, a sidebar menu is open with various options like 'New', 'Import', 'Runner', and 'Share Collection'. A blue box highlights the 'Add Request' option under the 'Hist' section. Another blue box highlights the 'Add Request' button in the main request configuration area. The main workspace shows a single request entry: 'GET localhost:3000'. The 'Headers' tab is selected, showing '(6)' entries. Below it is the 'Query Params' table:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

At the bottom, there's a 'Response' section with a rocket icon and the text 'Hit Send to get a response'. The bottom navigation bar includes 'Find and Replace', 'Console', 'Bootcamp' (which is currently selected), 'Build', 'Browse', and other icons.

Postman

+ New Import Runner Filter My Workspace Invite Upgrade

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 (highlighted with a blue box)

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** (highlighted with a blue box)

Hit Send to get a response

Find and Replace Console Bootcamp Build Browse

Environment Examples 0 BUILD Cookies Code

Send Save

23

The screenshot shows the Postman application interface. A modal dialog titled "SAVE REQUEST" is open in the center. Inside the dialog, there is a message about saving requests in collections, a "Request name" input field containing "retrieve collection" (which is highlighted with a blue box), and a "Request description (Optional)" text area. Below the text area, it says "Descriptions support Markdown". At the bottom of the dialog, there are "Cancel" and "Save to pet store" buttons, with "Save to pet store" also highlighted with a blue box. The background of the application shows a sidebar with "Collections" selected, displaying three collections: "afterschool", "demo", and "pet store". The "pet store" collection is currently expanded, showing one request. At the bottom of the screen, there are "Find and Replace" and "Console" tabs, along with navigation icons for "Bootcamp", "Build", and "Browse". The overall theme of the application is dark gray with orange highlights.

Postman

+ New Import Runner  My Workspace  Invite       Upgrade 

Filter  History Collections APIs + New Collection Trash

GET localhost:3000 GET retrieve collection    No Environment Examples 0 BUILD  

retrieve collection

GET localhost:3000/collection/products  

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 

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

The screenshot shows the Postman application interface. In the center, there's a request builder for an 'Untitled Request'. The method is set to 'POST' and the URL is 'localhost:3000/collection/products'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "id": 1006,  
3   "title": "new product",  
4   "price": 99  
5 }
```

The 'Body' tab also includes options for 'raw', 'JSON', 'GraphQl', and 'Binary'. Below the body, the response section shows a status of '200 OK' and the raw response data:

```
[{"id":1006,"title":"new product","price":99,"_id":"5fe1dbc8dc9f4e05b10a7961"}]
```

The left sidebar lists collections: 'afterschool' (6 requests), 'demo' (2 requests), and 'pet store' (2 requests). There are also links for 'localhost:3000' and 'retrieve collection'.

- 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

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Collections' with three items: 'afterschool' (6 requests), 'demo' (2 requests), and 'pet store' (3 requests). Below these are three specific requests: 'GET root response', 'GET retrieve collection', and 'POST add a product'. The 'POST add a product' request is currently selected, indicated by a blue border around its row in the sidebar.

The main workspace shows the configuration for the selected POST request:

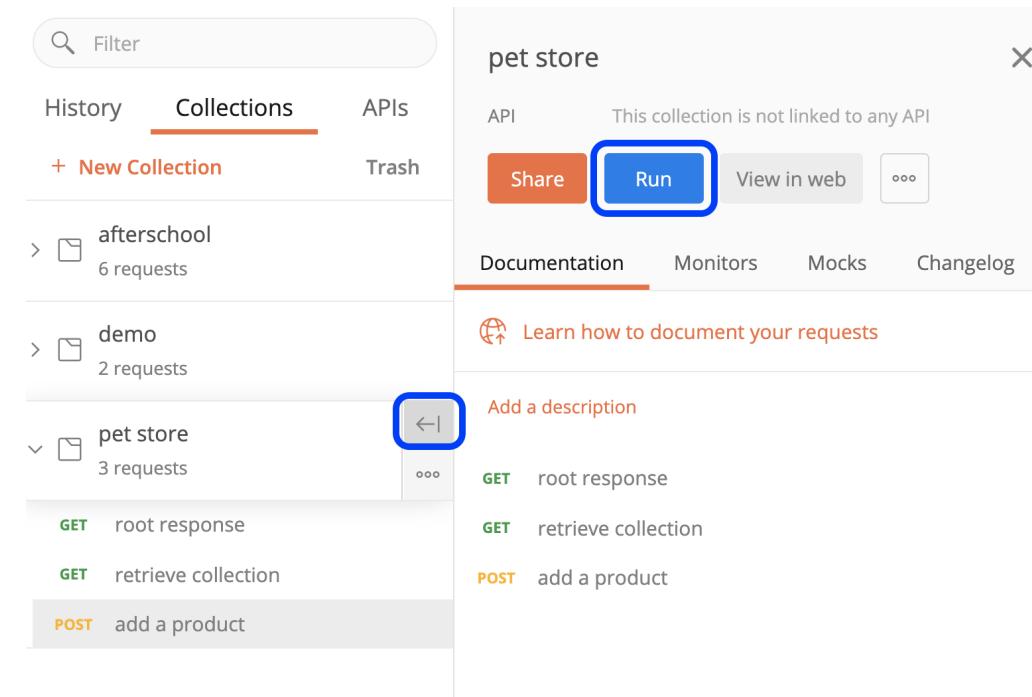
- Method:** POST
- URL:** localhost:3000/collection/products
- Body:** (highlighted in green)
- JSON:** (selected)
- Body Content:**

```
1 {
2   "id": 1006,
3   "title": "new product",
4   "price": 99
5 }
```

At the bottom of the workspace, the response status is shown as 200 OK with 115 ms and 346 B.

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

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

Method	Request	Response	Time	Size
GET	root response localhost:3000 / root response	200 OK	8 ms	307 B
GET	retrieve collection localhost:3000/collection... / retrieve collection	200 OK	109 ms	1.46 KB
POST	add a product localhost:3000/collection... / add a product	200 OK	136 ms	346 B

This request does not have any tests.

This request does not have any tests.

This request does not have any tests.

Console

Retrieve an object 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

The screenshot shows the Postman application interface. On the left, the 'Collections' sidebar is open, displaying several collections: 'afterschool' (6 requests), 'demo' (2 requests), and 'pet store' (4 requests). Below these are API endpoints: 'GET root response', 'GET retrieve collection', 'POST add a product', and 'GET retrieve a product'. The 'GET retrieve a product' endpoint is highlighted with a blue box.

In the main workspace, a GET request is configured to 'localhost:3000/collection/products/5fe1dbc8dc9f4e05b10a7961'. The 'Body' tab is selected, showing a JSON response:

```
1 {  
2   "_id": "5fe1dbc8dc9f4e05b10a7961",  
3   "id": 1006,  
4   "title": "new product",  
5   "price": 99  
6 }
```

The response status is 200 OK, with a duration of 134 ms and a size of 344 B. A 'Save Response' button is visible.

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 Filter My Workspace Invite Upgrade

No Environment Examples 0 BUILD

History Collections APIs + New Collection

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

PUT localhost:3000/collection/products/5fe1dbc8dc9f4e05b10a7961 Send Save

Params Authorization Headers (8) Body (green dot) Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

1 {
2 "price": 49
3 }

Body Cookies Headers (8) Test Results 200 OK 108 ms 284 B Save Response

Pretty Raw Preview Visualize JSON

1 {
2 "msg": "success"
3 }

Find and Replace Console Bootcamp Build Browse

The screenshot shows the Postman application interface. On the left, the sidebar lists collections: 'afterschool' (6 requests), 'demo' (2 requests), and 'pet store' (5 requests). Below these are various API endpoints: 'root response', 'retrieve collection', 'add a product', 'retrieve a product', and 'update a product'. The 'update a product' endpoint is currently selected and highlighted with a blue border. In the main workspace, a 'PUT' request is configured with the URL 'localhost:3000/collection/products/5fe1dbc8dc9f4e05b10a7961'. The 'Body' tab is active, showing a JSON payload with a single key-value pair: 'price': 49. The 'JSON' dropdown at the bottom of the body editor is also highlighted with a blue border. The response section shows a successful 200 OK status with a response body containing the message 'msg': 'success'. The bottom navigation bar includes 'Find and Replace', 'Console', 'Bootcamp', 'Build' (which is currently selected), 'Browse', and other icons.

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

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

Remove object with `DELETE`

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

```
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'}
    }
})
```

- 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 GET roo... GET ret... POST a... GET ret... PUT up... DEL re... No Environment Examples 0 BUILD

History Collections APIs + New Collection

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

remove a product

DELETE localhost:3000/collection/products/5fe1dbc8dc9f4e05b10a7961

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

Pretty Raw Preview Visualize JSON

```
1 {  
2   "msg": "success"  
3 }
```

Find and Replace Console Build

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