Contact Menu

Building a Simple CRUD Application with Express and MongoDB – Part 2

27TH JAN 2016

This article is the second part on creating a CRUD application with Express and MongoDB. We're going to venture deep into the last two operations that were not covered in the first part – **UPDATE** and **DELETE**.

Without further ado, let's start the second part.

CRUD - UPDATE

The **UPDATE** operation is used when you want to change something. It can only be triggered by browsers through a **PUT** request. Like the **POST** request, the **PUT** request can either be triggered through JavaScript or through a `<form>` element.

Let's try triggering a **PUT** request through JavaScript this time since we've already learned how to trigger a request through a `<form>` element while going through the **POST** request in the previous article.

For the purpose of this tutorial, we're going to create a button that, when clicked on, will *replace the last quote written by Master Yoda* with a quote written by Darth Vadar.

To do so, we first add a `button` into the `index.ejs` file:

```
<div>
  <h2>Replace last quote written by Master Yoda with a quote written by Darth Vadar</h2>
  <button id="update"> Darth Vadar invades! </button>
</div>
```

We're also going to create an external JavaScript file to execute the **PUT** request when the button is clicked. According to Express conventions, this file is placed in a folder called `public`

```
$ mkdir public
$ touch public/main.js
```

Then, we have to tell Express to make this `public` folder accessible to the public by using a built-in middleware called `express.static`

```
app.use(express.static('public'))
```

Once this is done, we can add the `main.js` file to the `index.ejs` file:

```
<!-- ... -->
<script src="main.js"></script>
</body>
```

Next, we're going to send the **PUT** request when the button is clicked:

```
// main.js
var update = document.getElementById('update')

update.addEventListener('click', function () {
  // Send PUT Request here
})
```

The easiest way to trigger a **PUT** request in modern browsers is to use the Fetch API. It is only supported on Firefox, Chrome and Opera, so you might want to use a polyfill if you want to use Fetch on an actual project.

We're going to send the following fetch request to the server. Have a quick look at it and I'll explain what it all means:

```
fetch('quotes', {
  method: 'put',
  headers: {'Content-Type': 'application/json'},
  body: JSON.stringify({
    'name': 'Darth Vader',
    'quote': 'I find your lack of faith disturbing.'
  })
})
```

Ready to understand why the Fetch request is written this way? :)

Fetch takes in two parameters. **The first parameter** is a path. In this case, we're sending the request to `/quote`, which will be handled on our server.

The second parameter, `options`**,** is an optional object that allows you to control a number of different settings. The ones we used above are `method`, `headers` and `body`.

`method` is straightforward. We set the `method` to `put` because we're sending a PUT request.

`headers` here refers to HTTP Headers you want to send to the server. It is an object with multiple key-value pairs.

`body` refers to the content you send to the server.

One thing you may notice is that we've set the `Content-Type` to `application/json`. We've also converted Darth Vadar's quote into JSON in the body with `JSON.stringify`. We're making these steps because we're sending Darth Vadar's quote in the JSON format (a standard format for sending information on the web) to the server.

Unfortunately, our server doesn't read JSON data yet. We can teach it to read JSON data by using the `bodyparser.json()` middleware:

```
app.use(bodyParser.json())
```

Once we've done everything above, we will be able to handle this **PUT** request by using the `put` method:

```
app.put('/quotes', (req, res) => {
  // Handle put request
})
```

The next step, then, is to learn how to look for the last quote by Master Yoda and change it to a quote by Darth Vadar in MongoDB.

Updating a Collection in MongoDB

MongoDB Collections come with a method called `findOneAndUpdate` that allows us to change one item from the database. It takes in four parameters – `query`, `update`, `options` and a `callback`.

```
db.collections('quotes').findOneAndUpdate(
  query,
  update,
  options,
  callback
)
```

The first parameter, `query`, allows us to filter the collection through key-value pairs given to it. We can filter the `quotes` collection for Master Yoda's quotes by setting the `name` to `Yoda`.

```
{
  name: 'Yoda'
}
```

The second parameter, `update`**,** tells MongoDB what to do with the update request. It uses MongoDB's update operators like `$set`, `$inc` and `$push`. We will use the `$set` operator since we're changing Yoda's quotes into Darth Vadar's quotes:

```
{
  $set: {
    name: req.body.name,
    quote: req.body.quote
  }
}
```

The third parameter, `options`, is an optional parameter that allows you to define additional stuff. Since we're looking for the last quote by Yoda, we will set `sort` within options to `{_id: -1}`. This allows MongoDB to search through the database, starting from the newest entry.

```
{
  sort: {_id:-1}
}
```

There's a possibility that there aren't any quotes by Master Yoda in our database. MongoDB does nothing by default when this happens. We can force it to create a new entry if we set the `upsert` option, which means insert (or save) if no entries are found, to true:

```
{
  sort: {_id: -1},
  upsert: true
}
```

The final parameter is a callback function that allows you to do something once MongoDB has replaced the final quote by Yoda with a quote by Darth Vadar. In this case, we can send the results back to the fetch request.

```
(err, result) => {
  if (err) return res.send(err)
  res.send(result)
}
```

Here's the entire `findOneAndUpdate` command we've written over the previous few steps:

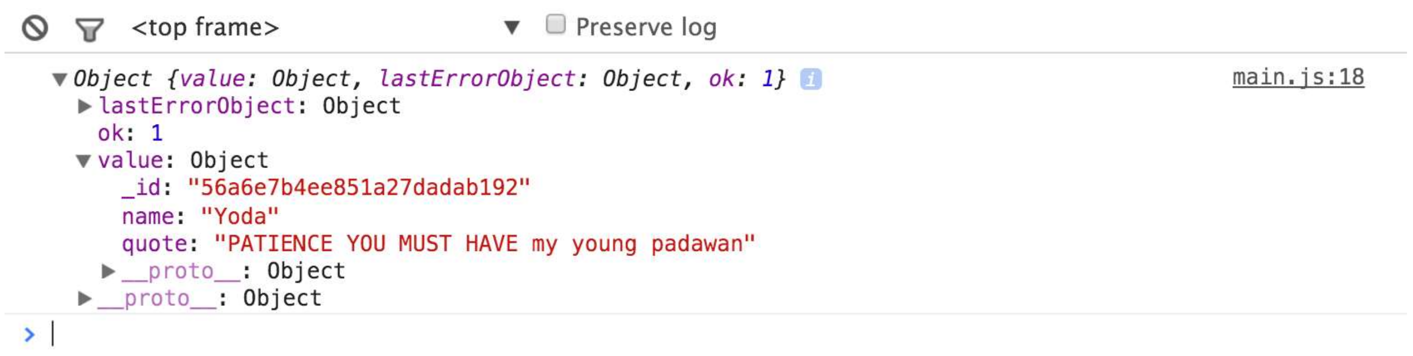
```
app.put('/quotes', (req, res) => {
  db.collection('quotes')
  .findOneAndUpdate({name: 'Yoda'}, {
    $set: {
      name: req.body.name,
      quote: req.body.quote
    }
  }, {
    sort: {_id: -1},
    upsert: true
  }, (err, result) => {
    if (err) return res.send(err)
```

```
      res.send(result)
    })
  })
```

Now, whenever someone clicks on the update button, the browser will send a *PUT* request through Fetch to our Express server. Then, the server responds by sending the changed quote back to fetch. We can then handle the response within by chaining `fetch` with a `then` method. This is possible because Fetch returns a Promise object.

The proper way to check if fetch resolved successfully is to use the `ok` method on the response object. You can then `return res.json()` if you want to read the data that was sent from the server:

```
fetch({ /* request */ })
.then(res => {
  if (res.ok) return res.json()
})
.then(data => {
  console.log(data)
})
```



```
▼Object {value: Object, lastErrorObject: Object, ok: 1}                main.js:18
  ▶lastErrorObject: Object
   ok: 1
  ▼value: Object
     _id: "56a6e7b4ee851a27dadab192"
     name: "Yoda"
     quote: "PATIENCE YOU MUST HAVE my young padawan"
   ▶__proto__: Object
  ▶__proto__: Object
```

If you are working on a fancy webapp, this is the part where you use JavaScript to update the DOM so users can see the new changes immediately. Updating the DOM is out of the scope of this article, so we're just going to refresh the browser to see the changes.

```
fetch({ /* request */ })
.then(res => {
  if (res.ok) return res.json()
})
.then(data => {
  console.log(data)
  window.location.reload(true)
})
```

That's it for the **CREATE** operation! Let's move on to the final one – **DELETE**.

(Before we move on, here's a quick signup form if you'd like to read similar articles from me every Wednesday).

CRUD - DELETE

The **DELETE** operation can only be triggered through a **DELETE** request. It's similar to the **UPDATE** request so it's simple if you understand what we've done earlier.

For this part, let's learn to delete the first quote by Darth Vadar. To do so, we first have to add a button to the `index.ejs` file.

```
<div>
  <h2>Delete Darth Vadar's first quote</h2>
  <button id="delete"> Delete first Darth Vadar quote </button>
</div>
```

Then, we'll trigger a **DELETE** request through Fetch whenever the delete button is clicked:

```
var del = document.getElementById('delete')

del.addEventListener('click', function () {
  fetch('quotes', {
    method: 'delete',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      'name': 'Darth Vader'
    })
  })
  .then(res => {
    if (res.ok) return res.json()
  }).
  then(data => {
    console.log(data)
    window.location.reload()
  })
})
```

We can then handle the event on our server side with the `delete` method:

```
app.delete('/quotes', (req, res) => {
  // Handle delete event here
})
```

Deleting an entry in MongoDB

MongoDB Collections come with a method called `findOneAndDelete` that allows us to remove one item from the database. It takes in three parameters – `query`, `options` and a `callback`. These parameters are exactly the same as the ones we used in `findOneAndUpdate` when updating an entry in MongoDB. The only difference here is that there's no `upsert` within `options`.

```
db.collections('quotes').findOneAndDelete(
  query,
  options,
  callback
)
```

Remember, we are trying to delete the first quote by Darth Vadar. To do so, we filter the `quotes` collection by the name, "Darth Vadar". The `query` parameter is hence:

```
{
  name: req.body.name
}
```

We can skip the `options` parameter since we don't have to reverse the sorting order. Then, we can send a response back to the Fetch request in the callback function.

```
(err, result) => {
  if (err) return res.send(500, err)
  res.send({message: 'A darth vadar quote got deleted'})
})
```

The complete code for the delete handler is as follows:

```
app.delete('/quotes', (req, res) => {
  db.collection('quotes').findOneAndDelete({name: req.body.name},
  (err, result) => {
    if (err) return res.send(500, err)
    res.send({message: 'A darth vadar quote got deleted'})
  })
})
```

Now, whenever somebody clicks on the delete button, the browser will send a *DELETE* request through Fetch to our Express server. Then, the server responds by sending either an error or a message back.

As before, we can reload the website when the fetch is successfully completed.

```
fetch({ /* request */ })
.then(res => {
  if (res.ok) return res.json()
})
.then(data => {
  console.log(data)
  window.location.reload(true)
})
```

That's it for the **DELETE** operation!

Wrapping Up

You have now learned all you need to know about creating simple applications with Node and MongoDB. Now, go forth and create more applications, young padawan. May the force be with you.

If you liked this article and felt that others should see it too, please consider sharing it.

68 Comments **Zell's Blog** 1 **Login**

Recommend 13 Share **Sort by Newest**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Caio Amaral • a month ago

Thank you, great tutorial. Nice job!

Reply • Share ›

Manuel Lazo • a month ago

pretty niceeeeeeeeeeeeeee :D, blessingssss!! :D

Reply • Share ›

Naveen Kumar • 2 months ago

Why can't I access the server from my mobile browser where both server system and mobile connected to same WiFi network.?

Any solution for this purpose?

Reply • Share ›

Phanuwat Phuangsuwan • 3 months ago

Great tutorial, thanks :)

Reply • Share ›

Charlie Zhu • 7 months ago

Howdy Zell,

I put "headers" as "header", missed the "s", debugging is hard.
Is there an good way to debug and troubleshooting?

Thanks,
Charlie

Reply • Share ›

Charlie Zhu • 7 months ago

Good user guide. I completed all demo code

Good user guide. I completed all demo code.

Reply • Share ›

Darth Coder • 7 months ago

I find your lack of semi-colons disturbing

1 • Reply • Share ›

Darth Coder → Darth Coder • 7 months ago

Your work is very satisfactory. It shall help in preparing propaganda for the Galactic Empire on the holonet

Reply • Share ›

jhonsnake • 7 months ago

Amazing tutorial to get started with crud and templating system from express and mongo! congratulations!! very helpful!

Reply • Share ›

chinkush varshney • 7 months ago

very helpful tutorial , thank you

1 • Reply • Share ›

Vikrant Tyagi → chinkush varshney • 7 months ago

yes **@chinkush varshney** thanks for provide this link :)

Reply • Share ›

Prashant P Singh • 8 months ago

Great tutorial. Kudos.

Reply • Share ›

Alex • 8 months ago

Great tutorial! It helped me understand how nodejs works!

I am trying to make an API for a mobile app using Express and I followed this. but when I try doing post request. only the id got saved in the collection. can you help me with this? Thanks!

Reply • Share ›

Zell Liew Mod → Alex • 8 months ago

Email me.

Reply • Share ›

Zeeshan Jadoon • 8 months ago

Really Awesome article.
appreciated.

Reply • Share ›

themistocles • 10 months ago

Terrific tutorial Zell, it must have taken a lot of work.

1 ^ | v • Reply • Share ›

Juxhin Shehu • a year ago

The best tutorial article so far

2 ^ | v • Reply • Share ›

Ellie Nurmukhametova • a year ago

You ar the best! Thank you!!

^ | v • Reply • Share ›

Andrei Smirnov • a year ago

It was very useful tutorial, thanks a lot and keep writing more good ones!

^ | v • Reply • Share ›

Vivek • a year ago

Nice tutorials, thank you :)

^ | v • Reply • Share ›

Zell Liew Mod → Vivek • a year ago

You are welcome.

^ | v • Reply • Share ›

Vivek → Zell Liew • a year ago

Hi Zell, can do same with angular or react so that it will be mean or mern stack?

^ | v • Reply • Share ›

Zell Liew Mod → Vivek • a year ago

It's essentially the same thing on the server side.

^ | v • Reply • Share ›

John Wakler • a year ago

Its very good article,who started learning NODE JS with MONGO .its very understandable for both. Great Job

1 ^ | v • Reply • Share ›

Alan Saberi • a year ago

In findOneAndUpdate and in callback function when I write it in ES6 way. I receive and unexpected token (error in that line. but if I change ()=>{} to function(){}. there will be no such a error.

^ | v • Reply • Share ›

Alan Saberi • a year ago

thanks. Perfect article.

1 ^ | v • Reply • Share ›

Barják László • a year ago

Thanks!

^ | v • Reply • Share ›

Kseniya Ramanchyk • a year ago

Thank you for Part 1 and Part 2, they are very understandable!

1 ^ | v • Reply • Share ›

j.cali • a year ago

Good article

^ | v • Reply • Share ›

pradeep p • a year ago

I was going through lot of articles and other tutorials to get started with MEAN stack development. But I was back to square one even after a week and did not progress much. I stumbled upon this article ,did spend some 1.5 hrs going through each step and following it up closely and everything is falling into place now. Thanks a lot Zell ,this really helped me to start with my MEAN stack development. And really loved the way you documented things step by step. Thanks a ton.

2 ^ | v • Reply • Share ›

Joseph Mtinangi • a year ago

Thanks, very helpful tutorial

^ | v • Reply • Share ›

Amir Shabanov • a year ago

Hey Zell, Great tutorial. Can you continue this tutorial and describe "How to upload images to mLab ?" or "or delete only selected quote"? I'm new in Mean stack

^ | v • Reply • Share ›

antonio ortiz • a year ago

Hey Zell,
What a informative TUT! I am trying to learn Node/Express/Mongo and you managed to clear up some things—buy going at a great pace. And it doesn't hurt injecting "Star wars" into the mix to boot!

I have one question/problem, there seems to be a lag when I click the "Vader" button to replace the "Yoda" quote. It actually doesn't appear until a refresh is done manually.

I checked the console and I am not getting an error—Actually looks like I'm getting the former Yoda quote as expected. See attached...

Here is a link to my code on Github...
https://github.com/antonioO...

^ | v • Reply • Share ›

Zell Liew Mod ➦ antonio ortiz • a year ago

Hm. Can you try and narrow your code down to the place you feel there's an error? See https://zellwk.com/blog/ask...

^ | v • Reply • Share ›

antonio ortiz ➦ Zell Liew • a year ago

Thanks for the reply! This is what I meant.
1)Add Yoda quote
2))Click Vader quote button which will 'Replace last quote written by Master Yoda with a quote written by Darth Vader" DOM does not get updated
3) Upon refreshing browser Darth Vader quote appears.

Because I didn't get any errors and it works technically, I thought maybe it was due to a connection or maybe I configured Mongo incorrectly...
I noticed when I downloaded your src from gitHub and I experienced the same behavior. Which makes me believe it is some DB config issues?

^ | v • Reply • Share ›

Tom C ➦ antonio ortiz • a year ago

Hi Antonio,

To quote directly from this post:

>If you are working on a fancy webapp, this is the part where you use JavaScript to update the DOM so users can see the new changes immediately. Updating the DOM is out of the scope of this article, so we're just going to refresh the browser to see the changes.

Just because the underlying data has changed, doesn't mean the page itself will automatically refresh. You'll need to use javascript to make the displayed data reflect the database on update or delete..or just refresh the page, as you've been doing.

^ | v • Reply • Share ›

benjibutton • a year ago

If you're like me and you're hosting this on a server with a reverse proxy or a base bath other than '/' then change your app.use(express.static('public')) to app.use('/starwars', express.static(__dirname + '/public')). This makes sure you're serving locally instead of over http. This will fix MIME type {text/html} related errors for your main.js Replace starwars with whatever your base path is

1 ^ | ∨ • Reply • Share ›

ironside • a year ago

hello !

please, do you could make available the complete code?

^ | ∨ • Reply • Share ›

Zell Liew Mod ➦ ironside • a year ago

Part 1 can be found here: https://zellwk.com/blog/cru...

^ | ∨ • Reply • Share ›

Adepurana Adiprana • a year ago

hi zell, thanks for your explanations :D
I've followed your tutorial and the update part is working fine, but may I ask something, is there any way to avoid any insertion if we don't find the name we want to update ?

^ | ∨ • Reply • Share ›

Zell Liew Mod ➦ Adepurana Adiprana • a year ago

Yep. Use `find` first, followed by `update`. Just do them separately

1 ^ | ∨ • Reply • Share ›

YINGXIE LI • a year ago

This is the best, detailed, awesome Express, Mongo & Nodejs tutorial I've ever saw! Fantastic!

2 ^ | ∨ • Reply • Share ›

Tumiso Marebane • 2 years ago

Thank you ... This is an awesome Tutorial!!

3 ^ | ∨ • Reply • Share ›

Guilherme Couto • 2 years ago

Thank you. This post is amazing.

^ | ∨ • Reply • Share ›

Zell Liew Mod ➦ Guilherme Couto • 2 years ago

You are welcome

^ | ∨ • Reply • Share ›

Che Armstrong • 2 years ago

Hi,

I noticed in mLab I have the following for each record:

{
"_id": {
"$oid": "5768f1d6fb630c280887e803"
},
"name": "Yoda",
"quote": "Mmmmmm"
}

I'm struggling with update - I want to be able to update by using the ID - is this the $oid? I've tried a few different things but I'm stuck. Can you help?

^ | ⌄ • Reply • Share ›

Che Armstrong ➦ Che Armstrong • 2 years ago

I worked this out...

I need to add a require for ObjectID:

const ObjectId = require('mongodb').ObjectID;

Then I could get the item by ID like this:

app.put('/quotes', function(req, res) {

db.collection('quotes').findOneAndUpdate(
{
"_id": ObjectId("5768f1d6fb630c280887e803")
},
{
$set: {
name: 'test'
}
},

see more

^ | ⌄ • Reply • Share ›

John • 2 years ago

Thank you for this! I would like to make you aware that

```
fetch({ /* request */ })
.then(res => {
  if (res.ok) return res.json()
})
.then(data => {
  console.log(data)
  window.location.reload(true)
})
```

does not update the page upon deletion. The error is:

> undefined:1 Uncaught (in promise) SyntaxError: Unexpected token A in JSON at position 0

This code however, does work:

```
fetch({ /* request */ })
.then(data => {
  console.log(data)
  window.location.reload(true)
})
```

Does anyone know why?

1 ^ | v • Reply • Share ›

Zell Liew Mod ➦ John • 2 years ago

There's an error with your server.js file. Make sure you send the correct response type (json) with res.json() instead of res.send().

2 ^ | v • Reply • Share ›

kanthalion ➦ Zell Liew • 8 months ago

I had the same issue. you have it in your code sample above as `res.send('A Darth Vadar...`
Changing it appropriately fixed that but I think it'd help future readers if you fixed it in the text so they won't have to scroll down here to find the fix.

1 ^ | v • Reply • Share ›

Load more comments

ALSO ON **ZELL'S BLOG**

How to think like a programmer
11 comments • 6 months ago

Breaking the rules | Zell Liew
5 comments • 9 months ago

Need help with your projects?

Hit the button below and tell me more! I'd love to help :)

Hire Zell for my project