Building a Simple CRUD Application with Express and MongoDB

22ND JAN 2016

If you know me you would have known that I started learning about the web without having gone through any computing courses in university previously. I didn't dare to venture into server-side programming for a long time because of my background.

I remember when I eventually picked up the courage to try, I had such a hard time understanding the documentations for Express, MongDB and Node that I gave up.

One year later, I finally understood how to work with these tools. Then, I decided to write a comprehensive tutorial so you won't have to go through the same headache I went through.

CRUD, Express and MongoDB

CRUD, Express and MongoDB are big words for a person who has never touched any server-side programming in their life. Let's quickly introduce what they are before we diving into the tutorial.

Express is a framework for building web applications on top of Node.js. It simplifies the server creation process that is already available in Node. In case you were wondering, Node

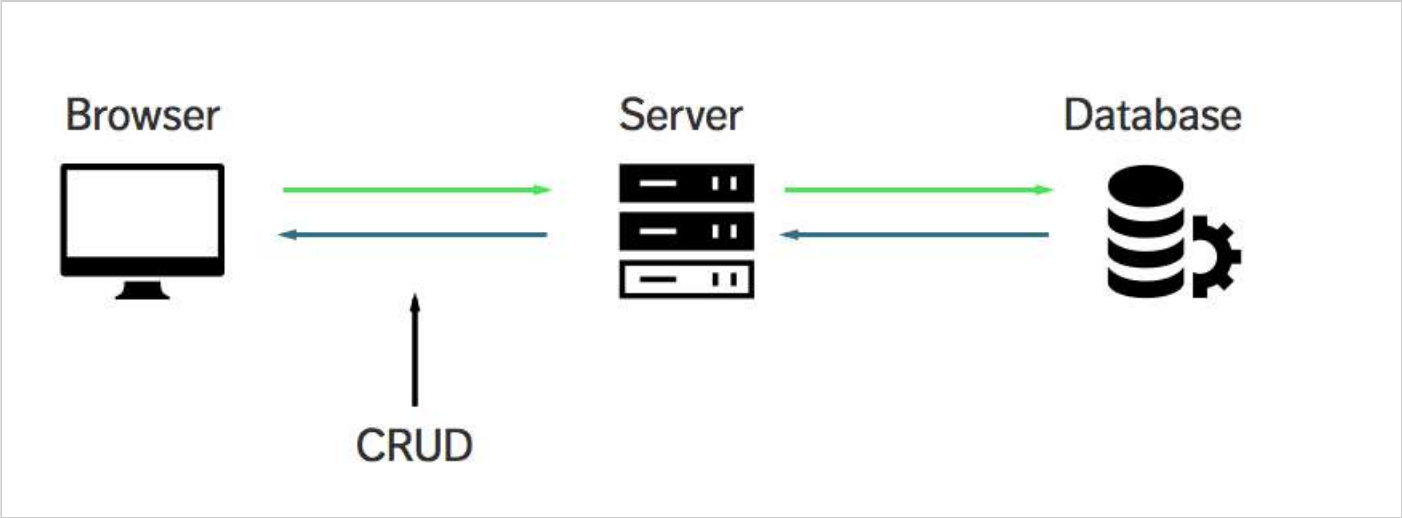
allows you to use JavaScript as your server-side language.

MongoDB is a database. This is the place where you store information for your web websites (or applications).

CRUD is an acronym for Create, Read, Update and Delete. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE respectively). This is what each operation does:

- **Create (POST)** - Make something
- **Read (GET)**_- Get something
- **Update (PUT)** - Change something
- **Delete (DELETE)**- Remove something

If we put CRUD, Express and MongoDB together into a single diagram, this is what it would look like:



Does CRUD, Express and MongoDB makes more sense to you now?

Great. Let's move on.

What we're building

We're going to build an application simple list application that allows you to keep track of things within a list (like a Todo List for example).

Well, a todo list is kind of boring. How about we make a list of quotes from Star wars characters instead? Awesome, isn't it? Feel free to take a quick look at the demo before continuing with the tutorial. Also, this is where you can find the finished code for the application.

By the way, what we're building isn't a sexy single page app. We're mainly focusing on how to use CRUD, Express and Mongo DB in this tutorial, so, more server-side stuff. I'm not going to emphasize on the styles.

You'll need two things to get started with this tutorial:

1. You aren't afraid of typing commands into a shell. Check out this article if you currently are.
2. You need to have Node installed.

To check if you have Node installed, open up your command line and run the following code:

```
$ node -v
```

You should get a version number if you have Node installed. If you don't, you can install Node either by downloading the installer from Node's website or downloading it through package managers like Homebrew (Mac) and Chocolatey (Windows).

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

Getting started

Start by creating a folder for this project. Feel free to call it anything you want. Once you navigate into it, run the `npm init` command.

This command creates a `package.json` file which helps you manage dependencies that we install later in the tutorial.

```
$ npm init
```

Just hit enter through everything that appears. I'll talk about the ones you need to know as we go along.

Running Node for the first time in your life

The simplest way to use node is to run the `node` command, and specify a path to a file. Let's create a file called `server.js` to run node with.

```
$ touch server.js
```

When the execute the `server.js` file, we want to make sure it's running properly. To do so, simply write a `console.log` statement in `server.js`:

```
console.log('May Node be with you')
```

Now, run `node server.js` in your command line and you should see the statement you logged:

```
[crud-express-mongo] node server.js
May Node be with you
```

Great. Let's move on and learn how to use Express now.

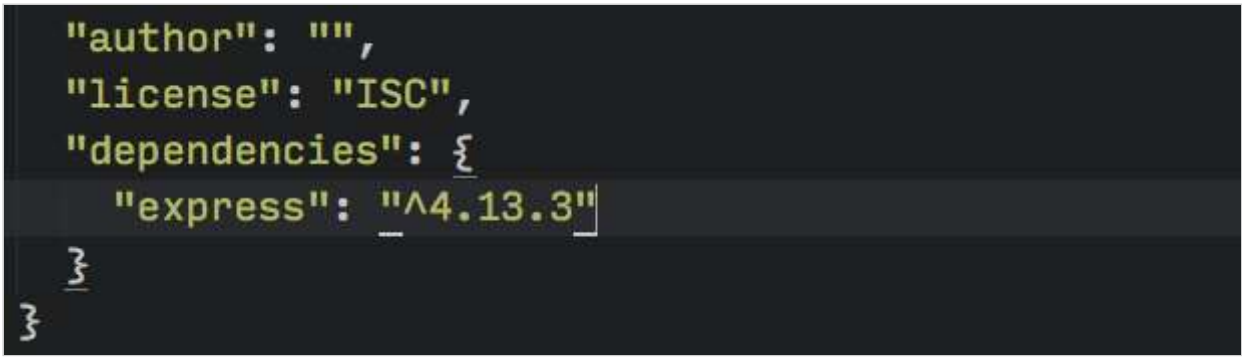
Using Express

We first have to install Express before we can use it in our application. Installing Express is pretty easy. All we have to do is run an install command with Node package manager (npm),which comes bundled with Node.

Run the `npm install express --save` command in your command line:

```
$ npm install express --save
```

Once you're done, you should see that npm has saved Express as a dependency in `package.json`.



Next, we use express in `server.js` by requiring it.

```
const express = require('express');
const app = express();
```

The first thing we want to do is to create a server where browsers can connect to. We can do so with the help of a `listen` method provided by Express:

```
app.listen(3000, function() {
  console.log('listening on 3000')
})
```

Now, run `node server.js` and navigate to `localhost:3000` on your browser. You should see a message that says "cannot get /".

That's a good sign. It means **we can now communicate to our express server through the browser**. This is where we begin CRUD operations.

CRUD - READ

The **READ** operation is performed by browsers whenever you visit a webpage. Under the hood, browsers sends a **GET** request to the server to perform a READ operation. The reason we see the "cannot get /" error is because we have yet to send anything back to the browser from our server.

In Express, we handle a **GET** request with the `get` method:

```
app.get(path, callback)
```

The first argument, `path`, is the path of the GET request. It's anything that comes after your domain name.

When we're visiting `localhost:3000`, our browsers are actually looking for `localhost:3000/`. The path argument in this case is `/`.

The second argument is a callback function that tells the server what to do when the path is matched. It takes in two arguments, a request object and a response object:

```
app.get('/', function (request, response) {
  // do something here
})
```

For now, let's write "Hello World" back to the browser. We do so by using a `send` method that comes with the response object:

```
app.get('/', function(req, res) {
  res.send('Hello World')
})
// Note: request and response are usually written as req and res respectively.
```

I'm going to start writing in ES6 code and show you how to convert to ES6 along the way as well. First off, I'm replacing the `function()` with the ES6 arrow function. The below code is the same as the above code:

```
app.get('/', (req, res) => {
  res.send('hello world')
})
```

Now, restart your server by doing the following:

1. Stop the current server by hitting `CTRL + C` in the command line.
2. Run `node server.js` again.

Then, navigate to `localhost:3000` on your browser. You should be able to see a string that says "Hello World".

Great. Let's change our app so we serve an `index.html` page back to the browser instead. To do so, we use the `sendFile` method that's provided by the `res` object.

```
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html')
  // Note: __dirname is directory that contains the JavaScript source code. Try logging it
and see what you get!
  // Mine was '/Users/zellwk/Projects/demo-repos/crud-express-mongo' for this app.
})
```

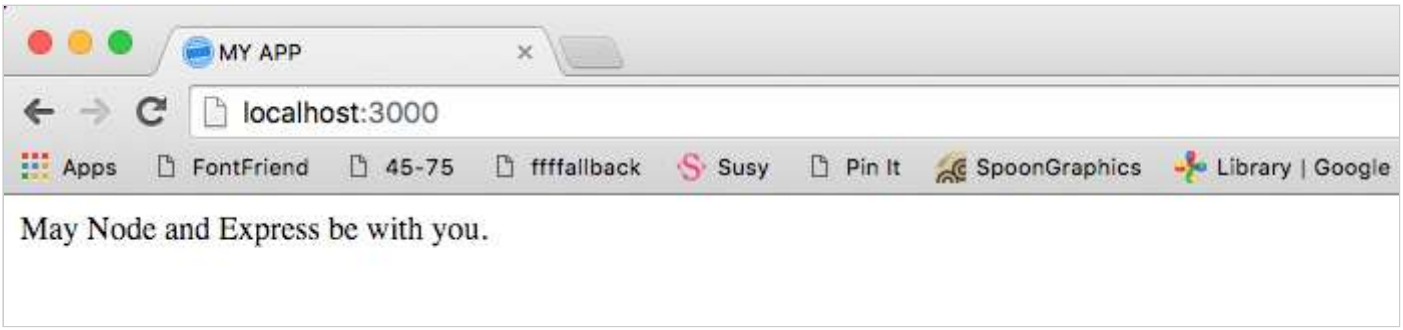
In the `sendFile` method above, we told Express to serve an `index.html` file that can be found in the root of your project folder. We don't have that file yet. Let's make it now.

```
touch index.html
```

Let's put some text in our `index.html` file as well:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>MY APP</title>
</head>
<body>
  May Node and Express be with you.
</body>
</html>
```

Restart your server and refresh your browser. You should be able to see the results of your HTML file now.



This is how Express handles a **GET** request (**READ** operation) in a nutshell.

At this point, you probably have realized that you need to restart your server whenever you make a change to `server.js`. This is process is incredibly tedious, so let's take a quick detour and streamline it by using a package called nodemon.

Enter Nodemon

Nodemon restarts the server automatically whenever you save a file that the server uses. We can install Nodemon by using the following command:

```
$ npm install nodemon --save-dev
```

Note: The reason we're using a `--save-dev` flag here is because we're only using Nodemon when we're developing. This flag would save Nodemon as a `devDependency` in your `package.json` file.

Moving on, Nodemon behaves exactly the same as node, which means we can run our server by calling `nodemon server.js`. However, we can't do it in the command line right now because Nodemon isn't installed with a `-g` flag.

There's one other way to run Nodemon – we can execute Nodemon from the `node_modules` folder. The code looks like this:

```
$ ./node_modules/.bin/nodemon server.js
```

That's a handful to type. One way to make it simpler is to create a `script` key in `package.json`.

```
{
  // ...
  "scripts": {
    "dev": "nodemon server.js"
  }
  // ...
}
```

Now, you can run `npm run dev` to trigger `nodemon server.js`.

Back to the main topic. We're going to cover the **CREATE** operation next.

CRUD - CREATE

The **CREATE** operation is performed only by the browser if a **POST** request is sent to the server. This POST request can triggered either with JavaScript or through a `<form>` element.

Let's find out how to use a `<form>` element to create new entries for our star wars quote app for this part of the tutorial.

To do so, you first have to create a `<form>` element and add it to your `index.html` file. You need to have three things on this form element:

1. An `action` attribute
2. a `method` attribute
3. and `name` attributes on all `<input>` elements within the form

```
<form action="/quotes" method="POST">
  <input type="text" placeholder="name" name="name">
  <input type="text" placeholder="quote" name="quote">
  <button type="submit">Submit</button>
</form>
```

The `action` attribute tells the browser where to navigate to in our Express app. In this case, we're navigating to `/quotes`. The `method` attribute tells the browser what to request to send. In this case, it's a POST request.

On our server, we can handle this POST request with a `post` method that Express provides. It takes the same arguments as the GET method:

```
app.post('/quotes', (req, res) => {
  console.log('Hellooooooooooooooooo!')
})
```

Restart your server (hopefully you've set up Nodemon so it restarts automatically) and refresh your browser. Then, enter something into your form element. You should be able to see `Hellooooooooooooooooo!` in your command line.

```
[nodemon] starting `node server.js`
listening on 3000
Hellooooooooooooooooo!
```

Great, we know that Express is handling the form for us right now. The next question is, how do we get the input values with Express?

Turns out, Express doesn't handle reading data from the `<form>` element on it's own. We have to add another package called body-parser to gain this functionality.

```
$ npm install body-parser --save
```

Express allows us to add middlewares like body-parser to our application with the `use` method. You'll hear the term middleware a lot when dealing with Express. These things are basically plugins that change the request or response object before they get handled by our application. **Make sure you place body-parser before your CRUD handlers!**

```
const express = require('express')
const bodyParser= require('body-parser')
const app = express()

app.use(bodyParser.urlencoded({extended: true}))

// All your handlers here...
```

The `urlencoded` method within body-parser tells body-parser to extract data from the `<form>` element and add them to the `body` property in the `request` object.

Now, you should be able to see everything in the form field within the `req.body` object. Try doing a `console.log` and see what it is!

```
app.post('/quotes', (req, res) => {
  console.log(req.body)
})
```

You should be able to get an object similar to the following in your command line:

```
{ name: 'Yoda',
  quote: 'Train yourself to let go of everything you fear to lose' }
```

Hmmm. Master Yoda has spoken! Let's make sure we remember Yoda's words. It's important. We want to be able to retrieve it the next time we load our index page.

Enter the database, MongoDB.

MongoDB

We first have to install MongoDB through npm if we want to use it as our database.

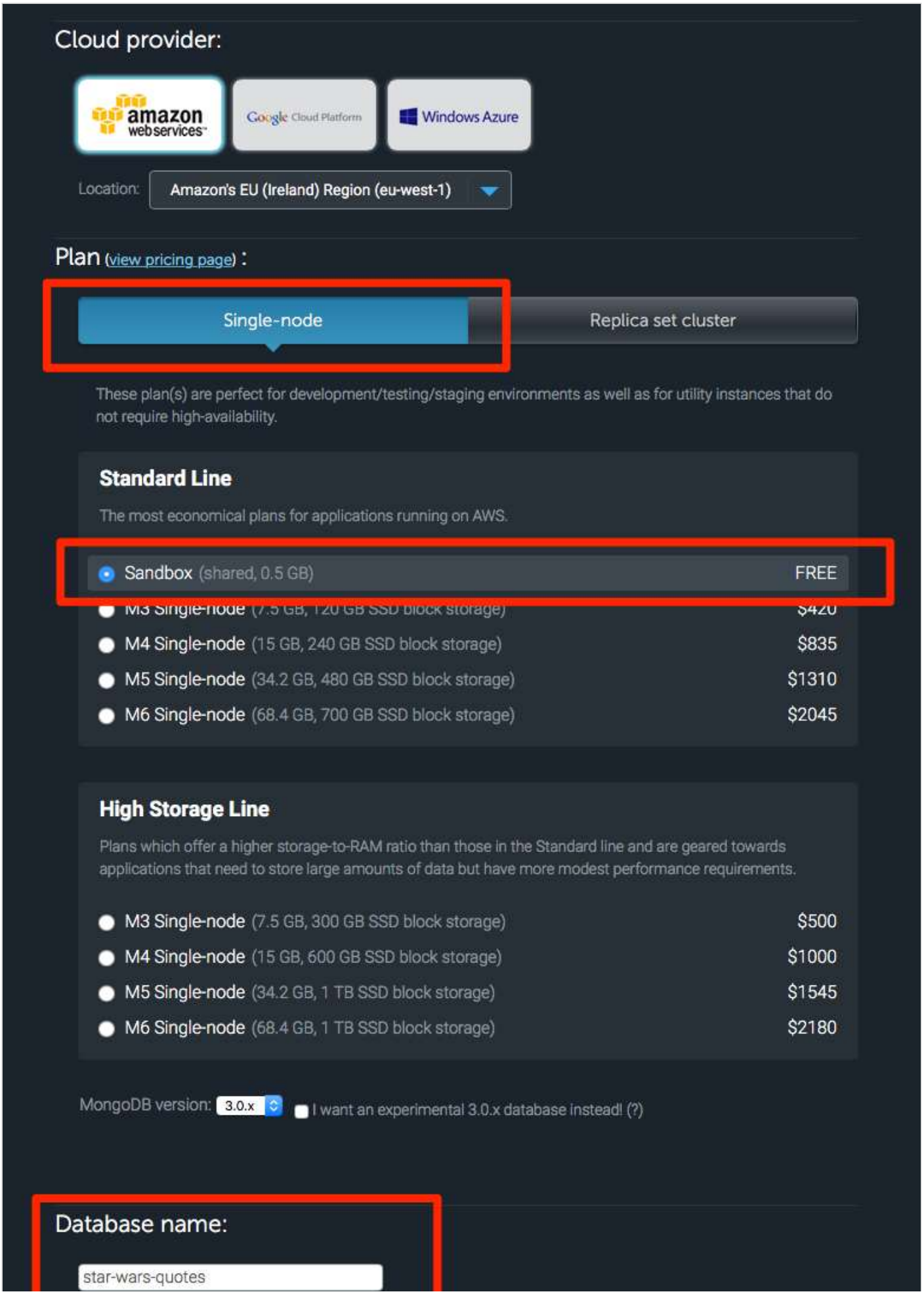
```
npm install mongodb --save
```

Once installed, we can connect to MongoDB through the `Mongo.Client`'s connect method as shown in the code below:

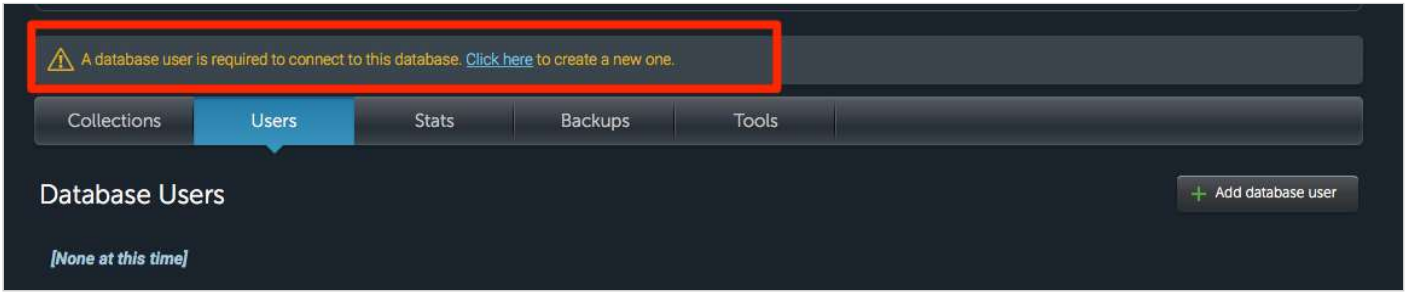
```
const MongoClient = require('mongodb').MongoClient

MongoClient.connect('link-to-mongodb', (err, database) => {
  // ... start the server
})
```

The next part is to get the correct link to our database. Most people store their databases on cloud services like MongoLab. We're going to do same as well.

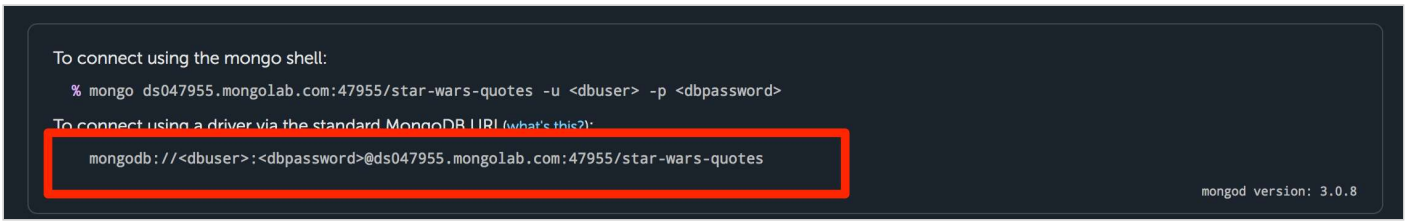
So, go ahead and create an account with MongoLab. (It's free). Once you're done, create a new MongoDB Deployment and set the plan to `sandbox`.


Cloud provider:
amazon web services
Google Cloud Platform
Windows Azure
Location:
Amazon's EU (Ireland) Region (eu-west-1)
Plan (view pricing page) :
Single-node
Replica set cluster
These plan(s) are perfect for development/testing/staging environments as well as for utility instances that do not require high-availability.
Standard Line
The most economical plans for applications running on AWS.
Sandbox (shared, 0.5 GB) FREE
M4 Single-node (15 GB, 240 GB SSD block storage) $835
M5 Single-node (34.2 GB, 480 GB SSD block storage) $1310
M6 Single-node (68.4 GB, 700 GB SSD block storage) $2045
High Storage Line
Plans which offer a higher storage-to-RAM ratio than those in the Standard line and are geared towards applications that need to store large amounts of data but have more modest performance requirements.
M3 Single-node (7.5 GB, 300 GB SSD block storage) $500
M4 Single-node (15 GB, 600 GB SSD block storage) $1000
M5 Single-node (34.2 GB, 1 TB SSD block storage) $1545
M6 Single-node (68.4 GB, 1 TB SSD block storage) $2180
MongoDB version: 3.0.x
I want an experimental 3.0.x database instead! (?)
Database name:
star-wars-quotes

Once you're done creating the deployment, head into it and create a database user and database password. **Remember the database user and database password** because you're going to use it to connect the database you've just created.



Finally, grab the MongoDB url and add it to your `MongoClient.connect` method. Make sure you use your database user and password!




```
MongoClient.connect('your-mongodb-url', (err, database) => {
  // ... do something here
})
```

Next, we want to start our servers only when the database is connected. Hence, let's move `app.listen` into the `connect` method. We're also going to create a `db` variable to allow us to use the database when we handle requests from the browser.

```
var db

MongoClient.connect('your-mongodb-url', (err, database) => {
  if (err) return console.log(err)
  db = database
  app.listen(3000, () => {
```

```
    console.log('listening on 3000')
  })
})
```

We're done setting up MongoDB. Now, let's create a `quotes` collection to store quotes for our application.

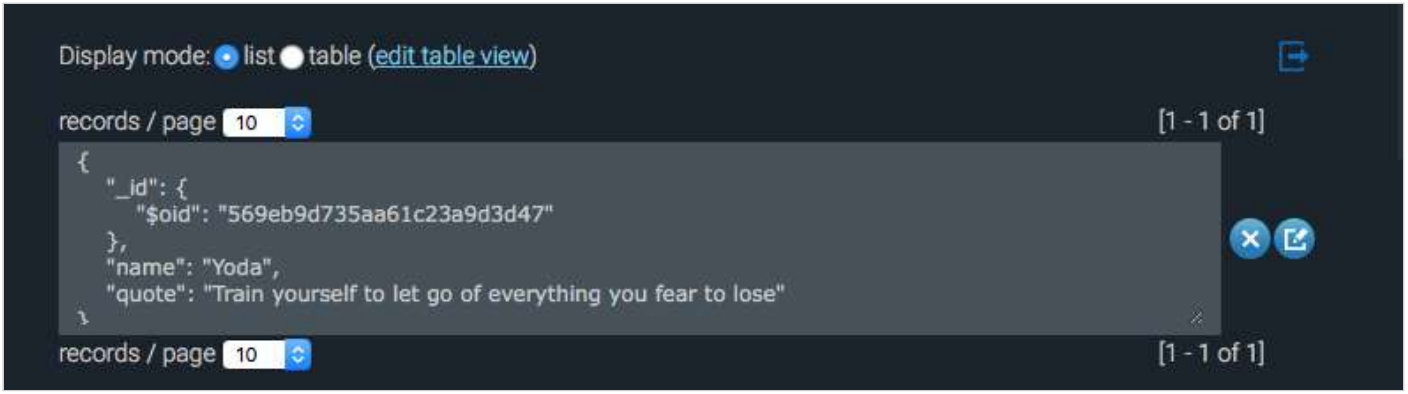
By the way, **a collection is a named location to store stuff**. You can create as many collections as you want. It can be things like "products", "quotes", "groceries", or any other labels you choose.

We can create the `quotes` collection by using the string `quotes` while calling MongoDB's `db.collection()` method. While creating the quotes collection, we can also save our first entry into MongoDB with the `save` method simultaneously.

Once we're done saving, we have to redirect the user somewhere (or they'll be stuck waiting forever for our server to move). In this case, we're going to redirect them back to `/`, which causes their browsers to reload.

```
app.post('/quotes', (req, res) => {
  db.collection('quotes').save(req.body, (err, result) => {
    if (err) return console.log(err)

    console.log('saved to database')
    res.redirect('/')
  })
})
```

Now, if you enter something into the `<form>` element, you'll be able to see an entry in your MongoDB collection.



Whoohoo! Since we already have some quotes in the collection, why not try showing them to our user when they land on our page?

Showing quotes to users

We have to do two things to show the quotes stored in MongoLab to our users.

1. Get quotes from MongoLab
2. Use a template engine to display the quotes

Let's go one step at a time.

We can get the quotes from MongoLab by using the `find` method that's available in the `collection` method.

```
app.get('/', (req, res) => {
  var cursor = db.collection('quotes').find()
})
```

The `find` method returns a `cursor` (A Mongo Object) that probably doesn't make sense if you `console.log` it out.

```
Readable {
  connection: null,
  server: null,
  disconnectHandler:
   { s: { storedOps: [], storeOptions: [Object], topology: [Object] },
     length: [Getter] },
  bson: {},
  ns: 'star-wars-quotes.quotes',
  cmd:
   { find: 'star-wars-quotes.quotes',
     limit: 0,
     skip: 0,
     query: {},
     slaveOk: true,
     readPreference: { preference: 'primary', tags: undefined, options: undefine
d } },
```

The good news is, this `cursor` object contains all quotes from our database. It also contains a bunch of other properties and methods that allow us to work with data easily. One such

method is the `toArray` method.

The `toArray` method takes in a callback function that allows us to do stuff with quotes we retrieved from MongoLab. Let's try doing a `console.log()` for the results and see what we get!

```
db.collection('quotes').find().toArray(function(err, results) {
  console.log(results)
  // send HTML file populated with quotes here
})
```

```
{ _id: 569eb9d735aa61c23a9d3d47,
  name: 'Yoda',
  quote: 'Train yourself to let go of everything you fear to lose' }
```

Great! You now see an array of quotes (I only have one right now). We've completed the first part – getting data from MongoLab. The next part is to generate a HTML that contains all our quotes.

We can't serve our `index.html` file and expect quotes to magically appear because there's no way to add dynamic content to a HTML file. What we can do instead, is to use template engines to help us out. Some popular template engines include Jade, Embedded JavaScript and Nunjucks.

I've written extensively about the how and why of template engines in a separate post. You might want to check it out if you have no idea what template engines are. I personally use (and recommend) Nunjucks as my template engine of choice. Feel free to check out the post to find out why.

For this tutorial, we're going to use Embedded JavaScript (`ejs`) as our template engine because it's the easiest to start with. You'll find it familiar from the get-go since you already know HTML and JavaScript.

We can use EJS by first installing it, then setting the `view engine` in Express to `ejs`.

```
$ npm install ejs --save
```

```
app.set('view engine', 'ejs')
```

Once the `view engine` is set, we can begin generating the HTML with our quotes. This process is also called **rendering**. We can use the `render` object built into the `response` object `render` to do so. It has the following syntax:

```
res.render(view, locals)
```

The first parameter, `views`, is the name of the file we're rendering. This file must be placed within a `views` folder.

The second parameter, locals, is an object that passes data into the view.

Let's first create an `index.ejs` file within the `views` folder so we can start populating data.

```
mkdir views
touch views/index.ejs
```

Now, place the following code within `index.ejs`.

```
<ul class="quotes">
  <% for(var i=0; i<quotes.length; i++) {%>
    <li class="quote">
      <span><%= quotes[i].name %></span>
      <span><%= quotes[i].quote %></span>
    </li>
  <% } %>
</ul>
```

See what I mean when I say you'll find it familiar? In EJS, you can write JavaScript within `<%` and `%>` tags. You can also output JavaScript as strings if you use the `<%=` and `%>` tags.

Here, you can see that we're basically looping through the `quotes` array and create strings with `quotes[i].name` and `quotes[i].quote`.

One more thing to do before we move on from the `index.ejs` file. Remember to copy the `<form>` element from the `index.html` file into this file as well. The complete `index.ejs` file so far should be:

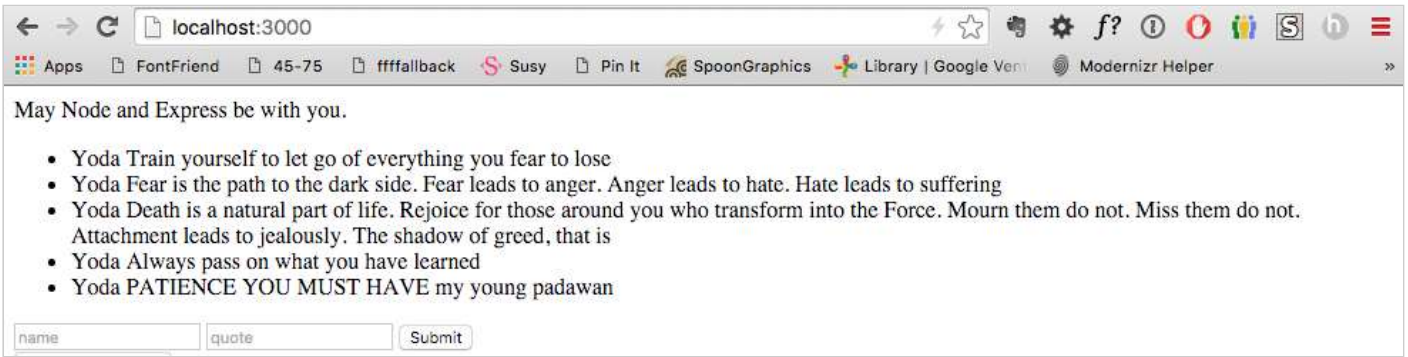
```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title>MY APP</title>
</head>
<body>
  May Node and Express be with you.

  <ul class="quotes">
  <% for(var i=0; i<quotes.length; i++) {%>
    <li class="quote">
      <span><%= quotes[i].name %></span>
      <span><%= quotes[i].quote %></span>
    </li>
  <% } %>
  </ul>

  <form action="/quotes" method="POST">
    <input type="text" placeholder="name" name="name">
    <input type="text" placeholder="quote" name="quote">
    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

Finally, we have to render this `index.ejs` file when handling the **GET** request. Here, we're setting the results (an array) as the `quotes` array we used in `index.ejs` above.

```
app.get('/', (req, res) => {
  db.collection('quotes').find().toArray((err, result) => {
    if (err) return console.log(err)
    // renders index.ejs
    res.render('index.ejs', {quotes: result})
  })
})
```

Now, refresh your browser and you should be able to see Master Yoda's quotes.



Um. You maybe only have one quote if you followed the tutorial step by step until this point. The reason I have multiple quotes is because I silently added more as I worked on the application.

Wrapping Up

We've covered a lot of ground in just 3000 words. Here are a few bullets to sum it all up. You have...

- Created an Express Server
- Learned to execute CREATE and READ operations
- Learned to save and read from MongoDB
- Learned to use a template engine like Embedded JS.

There are two more operations to go, but we'll leave it to the next post. Catch you there!

Oh, let me know in the comments if you found this tutorial useful!

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

cameron_hermens • 8 days ago

Just completed this tutorial, which was great. However, I (and a few others below) noticed an error for anyone running MongoDB >= 3.0.0:

```
MongoClient.connect('your-mongodb-url', (err, database) => {
if (err) return console.log(err)
db = database
app.listen(3000, () => {
console.log('listening on 3000')
})
})
```

This function no longer returns the database, but returns the client instead. It should be changed to the following:

```
MongoClient.connect('your-mongodb-url', (err, client) => {
if (err) return console.log(err)
db = client('star-wars-quotes')
app.listen(3000, () => {
console.log('listening on 3000')
})
})
```


Parrots repeat things • 8 days ago

This is great.

Yogesh Nogia • 15 days ago

the best

Reply • Share ›

Or Gilad • 20 days ago

One of the best readable tutorial i`v ever read. Great work!

Reply • Share ›

Kamil Naja • 24 days ago

Thanks for this tutorial, works very well!!!

Reply • Share ›

Temicka Nunyabidness • a month ago

Hey MongoLab website is down so I'm stuck at that section on the blog because MongoLab's website is down! GRRRR everything was going so well until I got stuck there

Reply • Share ›

Temicka Nunyabidness • a month ago

You saved my life with this site

Reply • Share ›

Manuel Lazo • a month ago

Ive been taking an online course about nodejs and mongodb and i decided to make an stop and practice a little about mongo. suddenly, i found your web post and it was really helpful and understandable :D, thanks a lot and keep going!!! :D

Reply • Share ›

Naveen Kumar • 2 months ago

Nice tutorial. :) The force is strong with you :)

2 Reply • Share ›

Tash Navarro • 2 months ago

HI .. I'm struggling with TypeError: db.collection is not a function pointing exactly at :
app.post('/quotes', (req, res) => {
db.collection('quotes').save(req.body, (err,result) => {
if (err) return console.log(err)
console.log('saved to database')
res.redirect('/')
if (err) return console.log(err)
})
I can't figure our what happened. I've been debugging this since last night. also my mongodb's version is same with mlab.

Reply • Share ›

Caio Amaral ➔ Tash Navarro • a month ago

The MongoClient.connect() method has changed.

The callback function now receive (err, client) instead (err, database) .

You can can fix it changing the line

db = database

To

db = database.db('star-wars-quotes')

1 • Reply • Share ›

James Hughes ➚ Caio Amaral • 21 days ago

Using (err, client) ini the callback function didn't work for me.
This worked for me:

I had the callback function as: (err, database)
and changed the line:
db = database
to: db = database.db('starwarsquotesapp');

• Reply • Share ›

Aleph Santos ➚ Tash Navarro • a month ago

This works here:
db.db('yourDatabaseNameHere').collection('quotes').find().toArray(function(err, results) {

1 • Reply • Share ›

Jenny Penn • 2 months ago

why must the form be at the index.html? i tried to change it to my app.component.html and res.sendFile(__dirname + '/app.component.html'). I have no error but it would not submit

• Reply • Share ›

Steve Adler • 2 months ago

Thank you for this tutorial. I'm as green as moss and as dumb as a stump. I'm having a tough time understanding where to put the new code segments you keep adding in relationship to the existing code. It would be really great to see where the new code goes in context to the existing code. I able to figure it out until you tried to post to Mongo, where I get "db.collection is not a function". I just don;t know where to include it so it knows that 'db' is a database. :(Is there a place to download the entire source code so I can cheat and see what you did? Thanks again!

• Reply • Share ›

Andy B ➚ Steve Adler • 2 months ago

I'm having the exact same problem. "db.collection is not a function". A look at the whole code would be really useful. I've never used anything mentioned in this article before so I wouldn't say it's suitable for *everyone* until the whole code can be seen somewhere. Would really appreciate it if you could put it up, Zell. Apart from this road bump though, the rest of the article is really easy to follow :-D!

• Reply • Share ›

Andy B ➚ Andy B • 2 months ago

This may help:

The error is in the mongodb library. Try to install mongodb: 2.2.33 version. Delete your node_modules directory and add (or edit)

"dependencies": {
"mongodb": "^2.2.33"}

To your package.json

Then

npm install

and there you are

From: https://stackoverflow.com/q...

1 • Reply • Share ›

Tash Navarro ➚ Andy B • 2 months ago

this works for me.thanks. just npm remove mongodb and install it again to overwrite dependencies in package.json.

• Reply • Share ›

Zell Liew Mod ➚ Steve Adler • 2 months ago

Not at the moment. Lemme put something up maybe this week or next.

4 • Reply • Share ›

Radwa • 2 months ago

great tutorial, thanks

5 • Reply • Share ›

Karan Patel • 2 months ago

Awesome explanation
Keep it Up

5 • Reply • Share ›

Ivana Natalia • 3 months ago

really helpfull for me. I am really a beginner but this tutorial is the best and clearly explained. Even i use mongodb local, i can understand.
Thank youuu

• Reply • Share ›

Danilo Gonzales, Jr. • 3 months ago

The best tutorial I have done, it it explains a lot plain and simple. Thank you very much!!!

The best tutorial I have done, it it explains a lot plain and simple. Thank you very much!!!
However, I used my own local mongodb setup at localhost:27017 and didn't bother to setup mongodb in the cloud.
Those who would want to learn about NodeJS, ExpressJS and MongoDB, this is the simplest and the most easy to understand tutorial...

Reply • Share ›

Tamil Vanan • 3 months ago

Very good article and gives lot of confidence those who are starting the app with express and mongodb, clean article.

Reply • Share ›

Jat Singh • 3 months ago

Very Beautifully explained. only found little confusion of Mongodb js code where it was need to put but later figured out, as tried making this first app with express and mongodb. Thanks.

8 Reply • Share ›

Test Coder • 3 months ago

Thank you so!!!!! much for this

Reply • Share ›

Stefano Rubini • 3 months ago

Thanks a lot!!!!

Reply • Share ›

Ayumi • 4 months ago

Thank you so much for the tutorial! Very useful!!

Reply • Share ›

Bhavatharini Sridharan • 4 months ago

Great tutorial. Covers a lot of the essentials to get you going without overwhelming details. Had to dig elsewhere to understand some concepts, but really enjoyed this one.

Reply • Share ›

Chaud • 4 months ago

CRUD operations are actually "Create Retrieve Update Delete" and they are atomic operations executed against a database. The communication between the client and server isn't CRUD; rather it is a stateless, request / response session.

The client in your diagram "Browser" is the presentation layer, or the rendered "View" in MVC (Model View Controller).

The "Server" in your diagram is the business logic layer (BLL), and data access layer (DAL). In your diagram they are fused together under "Server". The BLL would comprise the Controller in MVC and the DAL would be the Model.

The DAL usually presents an ORM (Object Relationship Mapping) interface to the business logic

layer, so that it can focus on it's terminology and domain specific objects like members, accounts, transactions, etc...

The DAL then translates those business objects, into the database operations necessary to "Create Retrieve Update and Delete" the individual components, pieces parts of the business objects. So, CRUD are the basic operations that allow the DAL to communicate with the database.

see more

2 • Reply • Share ›

Jagwant • 4 months ago

Hey! Just wanted to thank you for the amazing article. I had always found express to be a mystical topic and had trouble wrapping my head around the concepts.
Clean and well written indeed! Thanks.!

• Reply • Share ›

jaybird1905 • 4 months ago

You demystified a pretty complex, powerful topic. This is finally got my foot in the door in a way that I could understand how to put it all together. This is awesome man, thank you!

1 • Reply • Share ›

Max Q • 4 months ago

Hey, I'm way way late, but wanted to say thanks a ton for this tutorial - a few months back, it really got me started on Node/Express/Mongo. Keep up the awesome posts!

11 • Reply • Share ›

Shailesh Salekar • 5 months ago

Good article, looking for more same articles. thank you

• Reply • Share ›

Nadir • 5 months ago

Thank you for this article ! Helpful for a beginner like me attracted by the powerful NodeJs.

• Reply • Share ›

lidev • 5 months ago

Man! I always feared server side and you explained it so simply. Thank you. God bless the internet.

• Reply • Share ›

saad qamar • 5 months ago

perfectly explained...(Y)

• Reply • Share ›

Mohammad Khalid ➦ saad qamar • 5 months ago

nice ;)

• Reply • Share ›

Abbas Ali MuGhal • 5 months ago

how can i render the data in the react component ?

• Reply • Share ›

Faraaz Motiwala ➚ Abbas Ali MuGhal • 5 months ago

Make a request to the endpoint and store the response data into a property on the state.

• Reply • Share ›

Yofriadi Yahya • 6 months ago

what if i place views folder inside src folder, error in console. how to work around this?

• Reply • Share ›

Moch. Faisal Rasid • 6 months ago

thank you, I love nodejs

• Reply • Share ›

Alexandra Estrada • 7 months ago

great tutorial! :)

• Reply • Share ›

Charlie Zhu • 7 months ago

New mongoLab link: https:// mlab . com

2 • Reply • Share ›

Konrad Holubek • 7 months ago

This was really great and very helpful. Thank you !!

1 • Reply • Share ›

Nick Shulhin • 7 months ago

Thanks! Great tutorial, very useful!

• Reply • Share ›

Tom Larry • 7 months ago

It complains about this line "res.render('index.ejs', {quotes: result})", Error: Failed to lookup view "views/index.ejs" in views directory. Already made sure ejs module is installed, and I have this line "app.set('view engine', 'ejs');".
Same if replace index.ejs with views/index.ejs, and /views/index.ejs.

• Reply • Share ›

sekar • 7 months ago

REALLY awesome tutorial for beginers.....

• Reply • Share ›

cFred • 7 months ago

Thanks , you were great.

Reply • Share ›

Atul Gaikwad • 7 months ago

any other tutorial using mongoose schema

Reply • Share ›

Load more comments

ALSO ON **ZELL'S BLOG**

Writing modular CSS (Part 2)

65 comments • 10 months ago

Martin Schulz — > > Components in summary>

Promises in JavaScript

8 comments • 8 months ago

Vladimir Maxymenko — Thank you, Zell! I'm

Need help with your projects?

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

Hire Zell for my project