Node.js MySQL tutorial: a step-by-step getting started guide with Express js REST API

23-Nov-2020 | (about a 26 minute 👓)

Node.js and MySQL mix very well together. In this Node.js MySQL tutorial, we will look into how to get started with both of them step-by-step to build REST API for quotes using Express js.

Table of contents

Why Node.js MySQL tutorial

Node.js has been popularly coupled with NoSQL databases, especially Mongo DB. If you move an older codebase to Node.js you will not have the option to choose a database. Most older systems or even new ones use a relational database management system like MySQL. This leads to a need for a step-by-step Node.js MySQL tutorial.

If you are starting a new project don't blindly use a NoSQL like Mongo DB. This hilarious but accurate video from 2010 about MySQL vs Mongo still makes perfect sense.

> Use a NoSQL database if your project absolutely needs it else a relational database would suffice.

Prerequisites for Node.js MySQL tutorial

1. You have Node.js installed on your machine (or a docker container that can run Node.js). We will be using Node.js version 12 and npm 6.14.
2. You are familiar with how Node.js works generally and also know about Node.js frameworks. We will use Express js for this guide.
3. Some git knowledge would be really helpful
4. You have access to a MySQL instance either running locally or remotely. I would suggest using remote MySQL. You should know how RDBMS works.
5. You can code with an IDE, I will be using VS code but you can use any editor or IDE for this Node.js MySQL tutorial.

Node.js MySQL tutorial steps

We will be building a simple REST API with Express js that can give out quotes. Before diving deeper into the steps, I would really recommend you to take a refresher on REST(REpresentational State Transfer). It will be best to read on REST verbs and run some cURL commands to POST APIs.

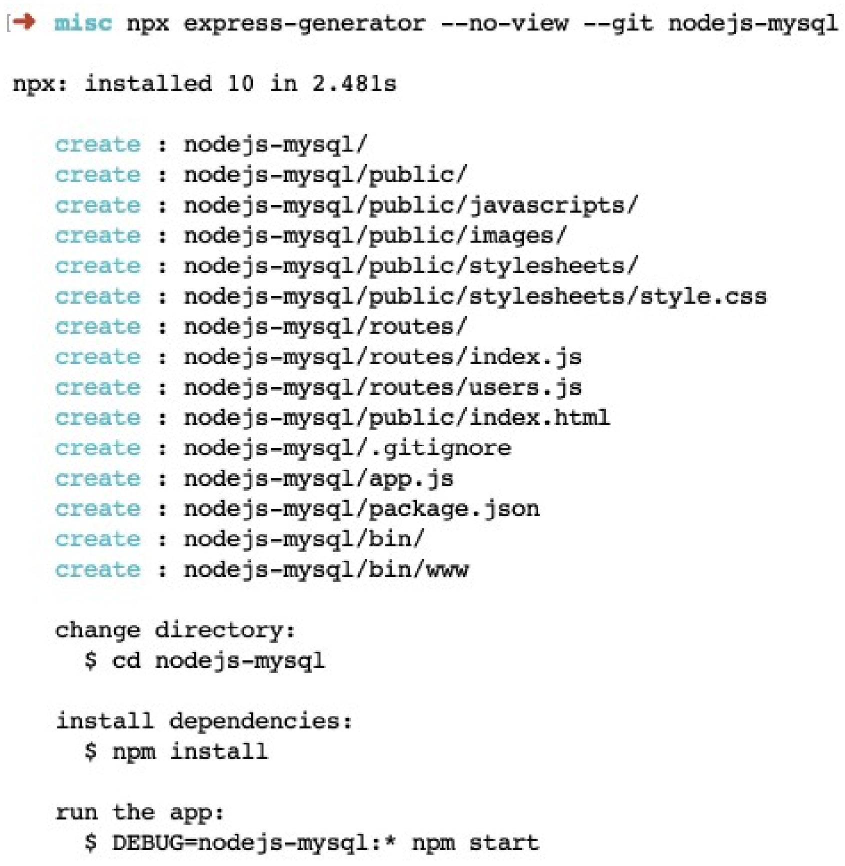
Given you have Node.js running (either on the machine or with Docker), we can start with setting up Express first:

Setup Express js for Node.js MySQL tutorial

To set up express, we will use the express-generator. You can generate an express js app without any view engine for this Node.js MySQL tutorial with the following command:

Copy

```
npx express-generator --no-view --git nodejs-mysql
```



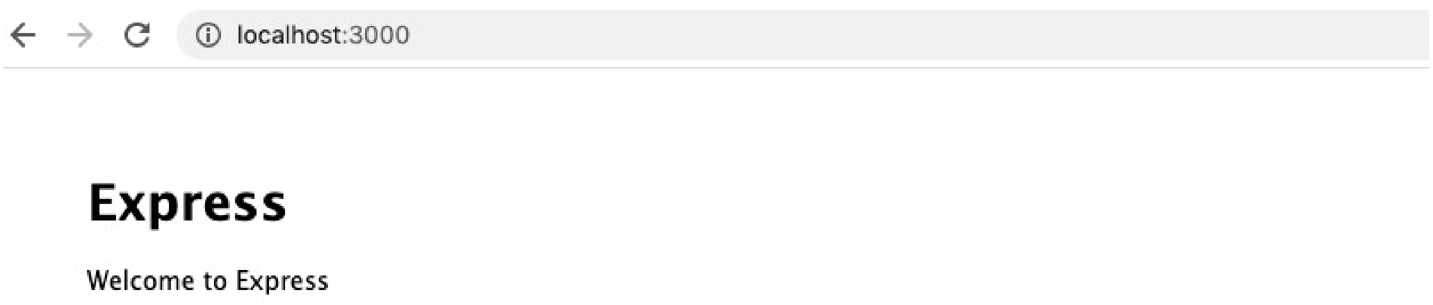
```
➜ misc npx express-generator --no-view --git nodejs-mysql

npx: installed 10 in 2.481s

   create : nodejs-mysql/
   create : nodejs-mysql/public/
   create : nodejs-mysql/public/javascripts/
   create : nodejs-mysql/public/images/
   create : nodejs-mysql/public/stylesheets/
   create : nodejs-mysql/public/stylesheets/style.css
   create : nodejs-mysql/routes/
   create : nodejs-mysql/routes/index.js
   create : nodejs-mysql/routes/users.js
   create : nodejs-mysql/public/index.html
   create : nodejs-mysql/.gitignore
   create : nodejs-mysql/app.js
   create : nodejs-mysql/package.json
   create : nodejs-mysql/bin/
   create : nodejs-mysql/bin/www

   change directory:
     $ cd nodejs-mysql

   install dependencies:
     $ npm install

   run the app:
     $ DEBUG=nodejs-mysql:* npm start
```

To quickly check the output execute the following:

Copy

```
cd nodejs-mysql && npm install && DEBUG=nodejs-mysql:* npm start
```

You should see the below output on your browser:



Express

Welcome to Express

The generated basic express app can be seen in this pull request.

Delete public folder

As we are building a REST API for quote in this Node.js tutorial with Express we will delete the public folder. We don't need any CSS or js for the purpose of this tutorial as we will deal with JSON.

To delete the public folder run the following command:

Copy

```
rm -rf public
```

Delete existing routes and create a new route for quotes

At this point, we will make some changes to the routes. We will delete the `routes/users.js` file that we don't need anymore. We will then add the `routes/quoets.js` file that looks like below:

Copy

```
const express = require('express');
const router = express.Router();

/* GET quotes listing. */
router.get('/', function(req, res, next) {
  res.json({
    data: [
      {
        quote: 'There are only two kinds of languages: the ones people complain abo
        author: 'Bjarne Stroustrup'
      }
    ],
    meta: {
      page: 1
    }
  });
});

module.exports = router;
```

For now the `/routes` will give a static output of only one quote as shown above. We will make it dynamic in the next steps.

After that, we will need to link up the quotes route with the `app.js` file. After the linking my `app.js` file looks like below:

Copy

```
const express = require('express');
const cookieParser = require('cookie-parser');
const logger = require('morgan');

const indexRouter = require('./routes/index');
const quotesRouter = require('./routes/quotes');

const app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());

app.use('/', indexRouter);
app.use('/quotes', quotesRouter);

module.exports = app;
```

One more thing to do is, change the `index.js` file to show a JSON output in place of rendering a view/HTML template. After this change the `index.js` file looks like below:

Copy

```
const express = require('express');
const router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.json({message: 'ok'});
});

module.exports = router;
```

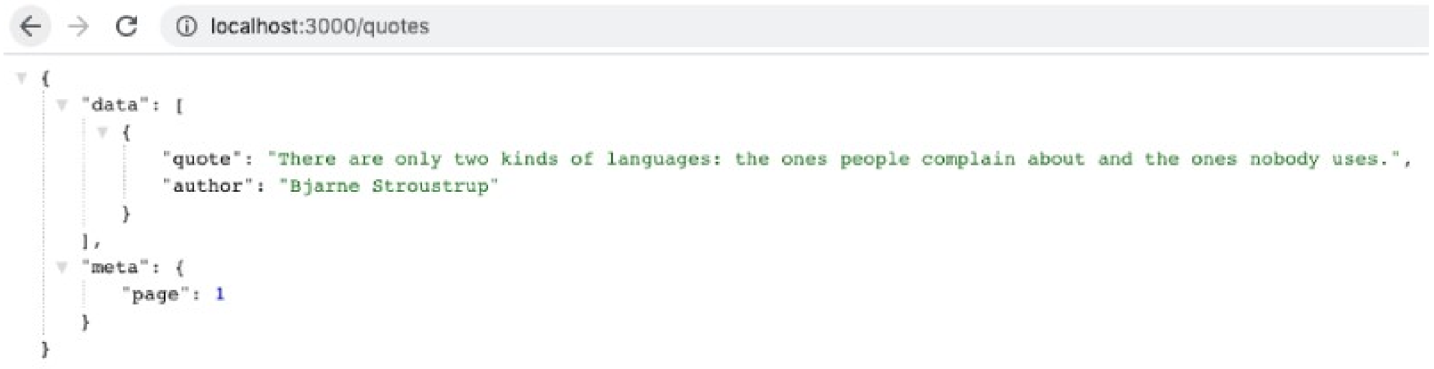
I have also changed all the `var` to `const` as it makes more sense. If you want to view all the changes in one go, it is available in this pull request.

You can see the last changes by running the following on your command line:

Copy

```
DEBUG=nodejs-mysql:* npm start
```

You should see the below when you hit `http://localhost:3000/quotes` on your browser:


localhost:3000/quotes

```
{
  "data": [
    {
      "quote": "There are only two kinds of languages: the ones people complain about and the ones nobody uses.",
      "author": "Bjarne Stroustrup"
    }
  ],
  "meta": {
    "page": 1
  }
}
```

Setup MySQL with quote table

At this juncture, for this Node.js MySQL tutorial, we will create a Quotes API. The consumers can fetch quotes and add new ones too. To enable this we will use a single table called `quote` , its structure is given below:

Copy

```
CREATE TABLE `quote`
( `id` INT(11) NOT NULL AUTO_INCREMENT ,
 `quote` VARCHAR(255) NOT NULL ,
 `author` VARCHAR(255) NOT NULL ,
 `created_at` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ,
 `updated_at` DATETIME on update CURRENT_TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTA
 PRIMARY KEY (`id`),
 INDEX `idx_author` (`author`), UNIQUE `idx_quote_uniqie` (`quote`)
)
 ENGINE = InnoDB CHARSET=utf8mb4 COLLATE utf8mb4_general_ci;
```

This is a simple table with 5 columns. The first one is `id` the auto-incremented primary key. Then there is `quote` and `author` . Both of these columns are varchar.

After that there is `created_at` and `updated_at` . Both date columns have a default value of `CURRENT_TIMESTAMP` so we don't need to send these values from code. Also not that, updated_at will be auto-updated when the row is updated because of this: `on update CURRENT_TIMESTAMP` .

You can run the above to create the `quote` table, to fill up some good programming-related quotes run the following insert SQL query too:

Copy

```
INSERT INTO `quote` (`id`, `quote`, `author`) VALUES
(1, 'There are only two kinds of languages: the ones people complain about and the
(3, 'Any fool can write code that a computer can understand. Good programmers write
(4, 'First, solve the problem. Then, write the code.', 'John Johnson'),
(5, 'Java is to JavaScript what car is to Carpet.', 'Chris Heilmann'),
(6, 'Always code as if the guy who ends up maintaining your code will be a violent
(7, "I'm not a great programmer; I'm just a good programmer with great habits.", 'K
(8, 'Truth can only be found in one place: the code.', 'Robert C. Martin'),
(9, "If you have to spend effort looking at a fragment of code and figuring out wha
(10, 'The real problem is that programmers have spent far too much time worrying ab
(11, 'SQL, Lisp, and Haskell are the only programming languages that I’ve seen wher
(12, 'Deleted code is debugged code.', 'Jeff Sickel'),
(13, 'There are two ways of constructing a software design: One way is to make it s
(14, 'Simplicity is prerequisite for reliability.', 'Edsger W. Dijkstra'),
(15, 'There are only two hard things in Computer Science: cache invalidation and na
(16, 'Measuring programming progress by lines of code is like measuring aircraft bu
```

The above query will add 15 quotes to the table, it should be a good starting point for the get quotes API. Notice that id 2 is missing and this is done on purpose. You can view the database init file here too.

Wire up Node.js with MySQL

At this stage, we will connect to MySQL from Node.js express application. To do this task, we will use Mysql2 library. If you are asking why not the default Mysql. It is because of 2 main reasons:

- Mysql2 has a wrapper for promises out of the box
- Mysql2 supports prepared statements which are faster and safer

If you want a head to head comparison of these two libraries please head to npm compare.

Install mysql2 in the express app

To start querying the MySQL database with Nodejs, we will first install the `mysql2` library with npm:

Copy

```
npm install --save mysql2
```

The above command will install the `mysql2` library and also add it to the `package.json` file. The next step is to update the `/quotes` route to show quotes from the database rather than a static quote.

Show the quotes - Get API

When you hit `http://localhost:3000` after starting the express js app, you can see something like below:

Copy

```
{
  "data": [
    {
    "quote": "There are only two kinds of languages: the ones people complain about
    "author": "Bjarne Stroustrup"
    }
  ],
  "meta": {
    "page": 1
  }
}
```

Let’s pull similar data from our database table `quote` and improvise more on it.

To show multiple quotes we will change the static response with a dynamic one. For that, we will need to connect to the database. Let’s create a config file that has the database credentials like below called `config.js` at the root of the project (besides app.js):

Copy

```
const env = process.env;

const config = {
  db: { /* do not put password or any sensitive info here, done only for demo */
    host: env.DB_HOST || 'remotemysql.com',
    user: env.DB_USER || '2ZE90yGC6G',
    password: env.DB_PASSWORD || 'JZFqXibSmX',
    database: env.DB_NAME || '2ZE90yGC6G',
    waitForConnections: true,
    connectionLimit: env.DB_CONN_LIMIT || 2,
    queueLimit: 0,
    debug: env.DB_DEBUG || false
  },
  listPerPage: env.LIST_PER_PAGE || 10,
};

module.exports = config;
```

We have created a `config.js` file that has the credentials for the database taken from the environment variable. If the environment variables are not set we use the fallback values.

After that, create `db.js` file in `/services` folder which should look like the following:

Copy

```
const mysql = require('mysql2/promise');
const config = require('../config');
const pool = mysql.createPool(config.db);

async function query(sql, params) {
  const [rows, fields] = await pool.execute(sql, params);

  return rows;
}

module.exports = {
  query
}
```

In this simple DB wrapper, we create a pool of connections for MySQL. As our config has `connectionLimit` of 2 it will create a maximum of 2 connections to the database. Then there is a straightforward `query` method exposed out that can run the SQL query with given params.

After that, create a `/services/quotes.js` file with the following contents:

Copy

```
const db = require('../services/db');

async function getMultiple(){
  const data = await db.query('SELECT id, quote, author FROM quote');
  const meta = {page: 1};

  return {
    data,
    meta
  }
}

module.exports = {
  getMultiple
}
```

Till this point, it is a very simple file that includes the `db` service created above. Then there is a `getMultiple` function exposed out with module.exports. Beware this will query all the records on the database which should be 15 at this stage. We will make it paginated in the next step.

Consequently we will wire up the `getMultiple` function in `/services/quotes.js` with the route `/quotes` in the `/routes/quotes.js` file like below:

Copy

```
const express = require('express');
const router = express.Router();
const quotes = require('../services/quotes');

/* GET quotes listing. */
router.get('/', async function(req, res, next) {
  try {
    res.json(await quotes.getMultiple());
  } catch (err) {
    console.error(`Error while getting quotes `, err.message);
    next(err);
  }
});

module.exports = router;
```

I think this is the right time to install `nodemon` , I would recommend to install it globally like below:

Copy

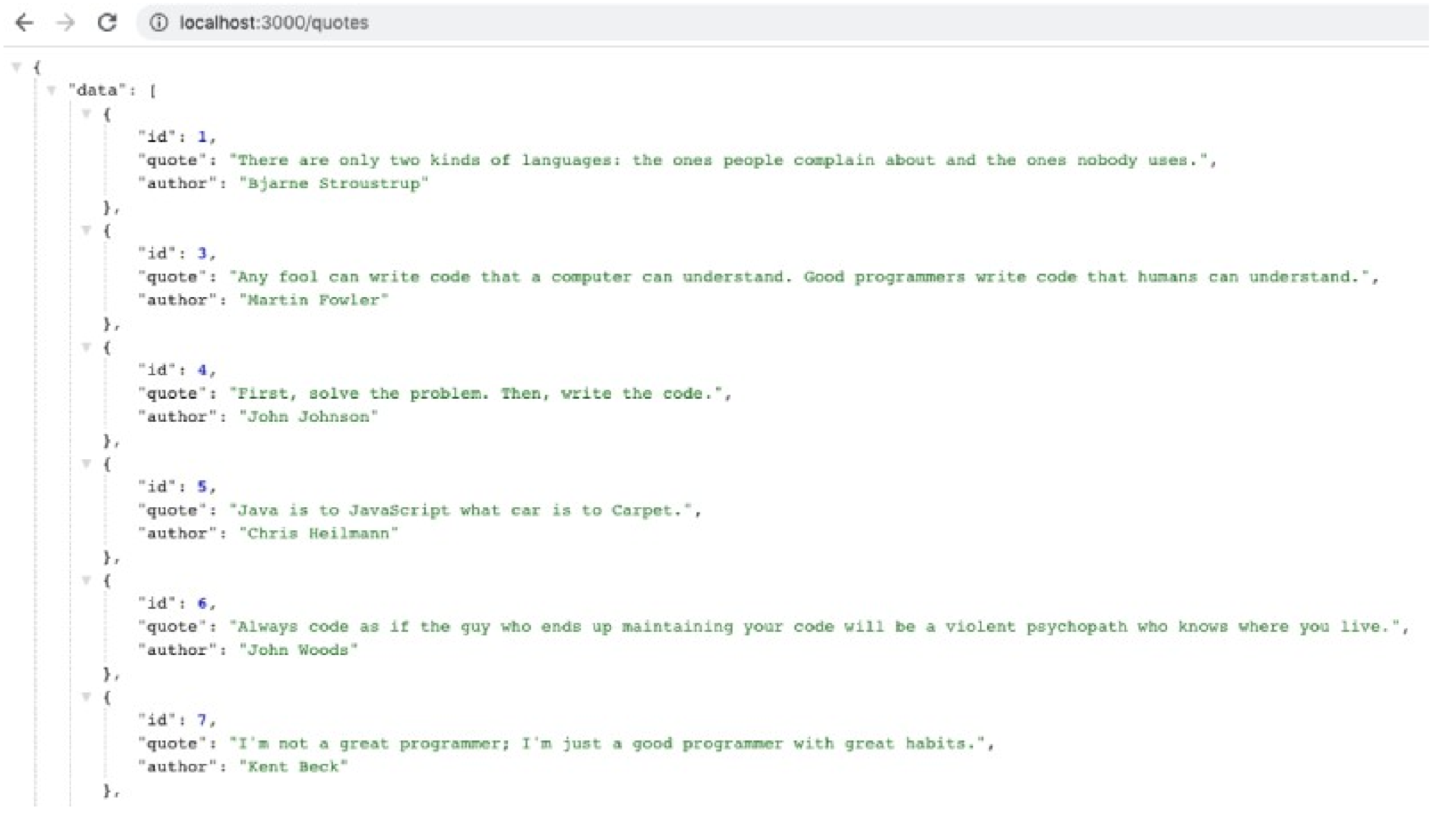
```
npm install -g nodemon #you might need sudo depending on your config
```

With nodemon, you can restart the Node.js server on each code change, which is very helpful while developing. You can run the app now with `nodemon` like below to check the result on a browser:

Copy

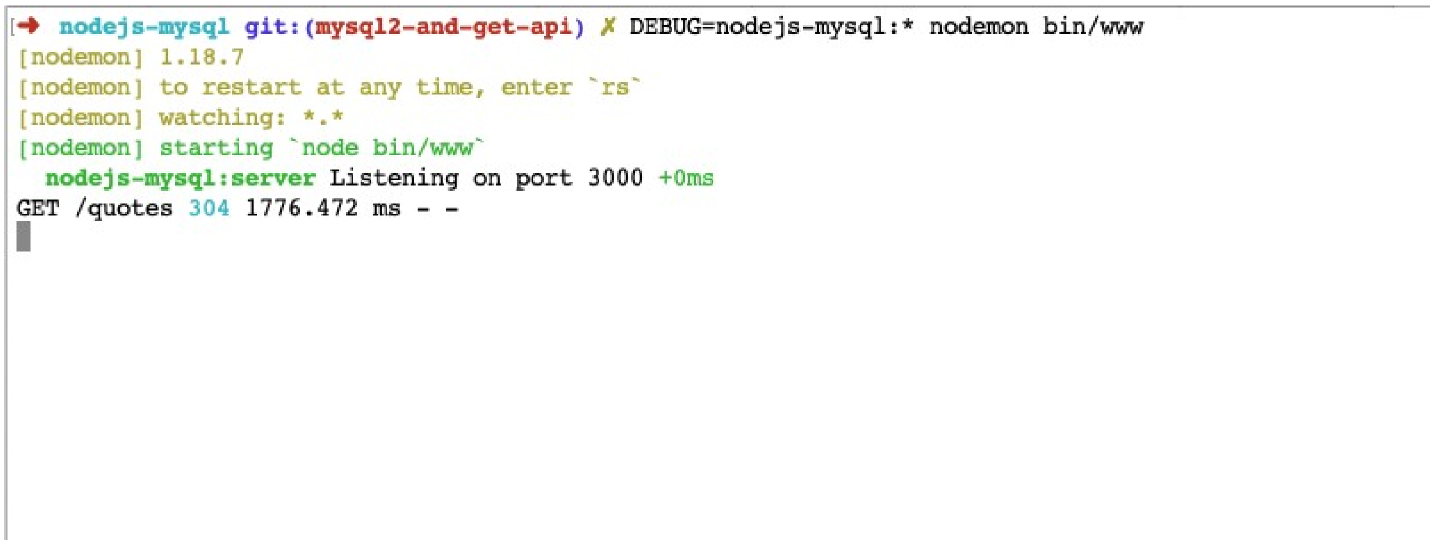
```
DEBUG=nodejs-mysql:* nodemon bin/www
```

When you hit `http://localhost:3000` on the browser, you should see a similar output or lots of `JSON` on the browser:



```
localhost:3000/quotes

{
  "data": [
    {
      "id": 1,
      "quote": "There are only two kinds of languages: the ones people complain about and the ones nobody uses.",
      "author": "Bjarne Stroustrup"
    },
    {
      "id": 3,
      "quote": "Any fool can write code that a computer can understand. Good programmers write code that humans can understand.",
      "author": "Martin Fowler"
    },
    {
      "id": 4,
      "quote": "First, solve the problem. Then, write the code.",
      "author": "John Johnson"
    },
    {
      "id": 5,
      "quote": "Java is to JavaScript what car is to Carpet.",
      "author": "Chris Heilmann"
    },
    {
      "id": 6,
      "quote": "Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.",
      "author": "John Woods"
    },
    {
      "id": 7,
      "quote": "I'm not a great programmer; I'm just a good programmer with great habits.",
      "author": "Kent Beck"
    },
```

If you go back and check your console where you ran `nodemon` you should be able to see something like below:



```
➜ nodejs-mysql git:(mysql2-and-get-api) ✗ DEBUG=nodejs-mysql:* nodemon bin/www
[nodemon] 1.18.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node bin/www`
  nodejs-mysql:server Listening on port 3000 +0ms
GET /quotes 304 1776.472 ms - -
```

If you change any file the server will restart because of nodemon. In the next step of this Node.js MySQL tutorial with express js, we will paginate the results 10 quotes on the page. If you want to see the code changes we did to get the quotes from the database it is here in this pull request.

Paginate the quotes for Node.js MySQL tutorial

At this juncture, we will start to paginate the quotes 10 quotes per page. The has already been put in place in the `/config.js` file at line no. 14 as `listPerPage: env.LIST_PER_PAGE || 10,` we will use it now.

We will add a `/helper.js` on the root that should look like below:

Copy

```
function getOffset(currentPage = 1, listPerPage) {
  return (currentPage - 1) * [listPerPage];
}

function emptyOrRows(rows) {
  if (!rows) {
    return [];
  }
  return rows;
}

module.exports = {
  getOffset,
  emptyOrRows
}
```

We will use this helper.js file to calculate the offset. The other function will return an empty array if the rows variable is empty, else it will return rows.

Next we will update the query to get quotes in `/services/quotes.js` service. The change quote service looks like below:

Copy

```
const db = require('./db');
const helper = require('../helper');
const config = require('../config');

async function getMultiple(page = 1){
  const offset = helper.getOffset(page, config.listPerPage);
  const rows = await db.query(
    'SELECT id, quote, author FROM quote LIMIT ?,?',
    [offset, config.listPerPage]
  );
  const data = helper.emptyOrRows(rows);
  const meta = {page};

  return {
    data,
    meta
  }
}

module.exports = {
  getMultiple
}
```

The main difference for the pagination feature compared to the older quotes service is the query has offset and limit passed to it. Notice that we are using a prepared statement which makes the query secure from SQL injection. You can read more about SQL injection prevention with prepared statements in this stackoverflow answer.

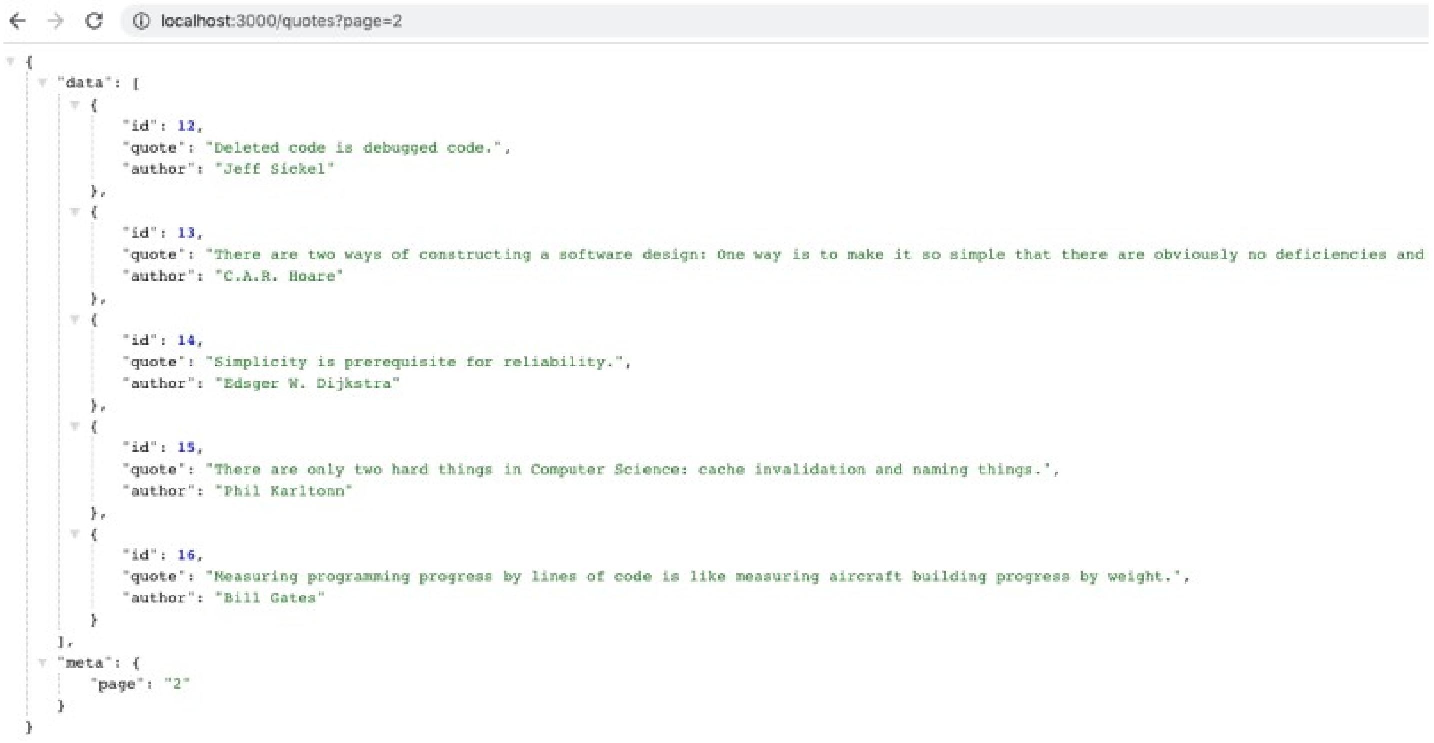
The other file changed to get the pagination feature is `/routes/quotes.js`. The new quotes.js route looks like below now:

Copy

```
const express = require('express');
const router = express.Router();
const quotes = require('../services/quotes');

/* GET quotes listing. */
router.get('/', async function(req, res, next) {
  try {
    res.json(await quotes.getMultiple(req.query.page));
  } catch (err) {
    console.error(`Error while getting quotes `, err.message);
    next(err);
  }
});

module.exports = router;
```

The only change here is we are passing the `page` query parameter to the `getMultiple` function. This will enable pagination like `/quotes?page=2` etc.

If you run the app and hit the browser with `http://localhost:3000/quotes?page=2` you will see 5 quotes like below:



localhost:3000/quotes?page=2

```
{
  "data": [
    {
      "id": 12,
      "quote": "Deleted code is debugged code.",
      "author": "Jeff Sickel"
    },
    {
      "id": 13,
      "quote": "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and
      "author": "C.A.R. Hoare"
    },
    {
      "id": 14,
      "quote": "Simplicity is prerequisite for reliability.",
      "author": "Edsger W. Dijkstra"
    },
    {
      "id": 15,
      "quote": "There are only two hard things in Computer Science: cache invalidation and naming things.",
      "author": "Phil Karltonn"
    },
    {
      "id": 16,
      "quote": "Measuring programming progress by lines of code is like measuring aircraft building progress by weight.",
      "author": "Bill Gates"
    }
  ],
  "meta": {
    "page": "2"
  }
}
```

So what happened here, the main change is in the way we construct the `SELECT` query. Depending on the page number we calculate an offset and pass a different query:

- For page 1, the query is `SELECT id, quote, author FROM quote LIMIT 0,10`
- For page 2, the query becomes `SELECT id, quote, author FROM quote LIMIT 10,10`

As you see the offset calculation makes it possible to get the next set of 10 quotes where 10 is the no. of items we want to list as per our config. This might be a big change to comprehend in one go, please have a look at this pull request for all the code that has changed for the pagination feature to come to life.

Next steps

As you have a basic GET API up and running you can add more features to it like:

- Add a new route like `/quotes/{id}` to get a single quote by id
- You can add quote filter/search capability like by the author for instance
- You can also search by word using SQL Like `%computer%` can give all quotes that have the word computer in it
- To make matters exciting for practice, add a new column called `category` and update the API.

I will leave it up to you on what else you would want to build on top of the newly created GET quotes REST API endpoint. The next step is to create a POST API to create new quotes.

Save new quote - POST API for Node.js MySQL tutorial

To create new quotes we will need a Post API. Before we get on with it, let's clear out our assumptions:

1. We will not use a sophisticated validation library like Joi for this demo.
2. We will keep the response codes as simple as possible
3. We will not build PUT (update) and DELETE endpoints. As you can run the INSERT query UPDATE and DELETE will be similar with a difference of the quote ID/IDs being passed in the request body.

Let’s get cracking with the code for the POST quotes API. The first thing is we will add the POST quotes route to `/routes/quotes.js` file just above `module.exports = router` line:

Copy

```
/* POST quotes */
router.post('/', async function(req, res, next) {
  try {
    res.json(await quotes.create(req.body));
  } catch (err) {
    console.error(`Error while posting quotes `, err.message);
    next(err);
  }
});

module.exports = router;
```

After that we will add `validateCreate` and `create` functions in the `/services/quotes.js` service file and expose `create` in module.exports like below:

Copy

```
function validateCreate(quote) {
  let messages = [];

  console.log(quote);

  if (!quote) {
    messages.push('No object is provided');
  }

  if (!quote.quote) {
    messages.push('Quote is empty');
  }

  if (!quote.author) {
    messages.push('Quote is empty');
  }

  if (quote.quote && quote.quote.length > 255) {
    messages.push('Quote cannot be longer than 255 characters');
  }

  if (quote.author && quote.author.length > 255) {
    messages.push('Author name cannot be longer than 255 characters');
  }

  if (messages.length) {
    let error = new Error(messages.join());
    error.statusCode = 400;

    throw error;
  }
}

async function create(quote){
  validateCreate(quote);

  const result = await db.query(
    'INSERT INTO quote (quote, author) VALUES (?, ?)',
    [quote.quote, quote.author]
  );

  let message = 'Error in creating quote';

  if (result.affectedRows) {
    message = 'Quote created successfully';
  }
  return {message};
}

module.exports = {
  getMultiple,
  create
}
```

I know the validation is a bit primitive but it does the job for now. A better way to do it would be to use Joi or a similar validation library. Next up let's add a new error handler to show our validation or other errors as JSON responses in the `/app.js` file like below:

Copy

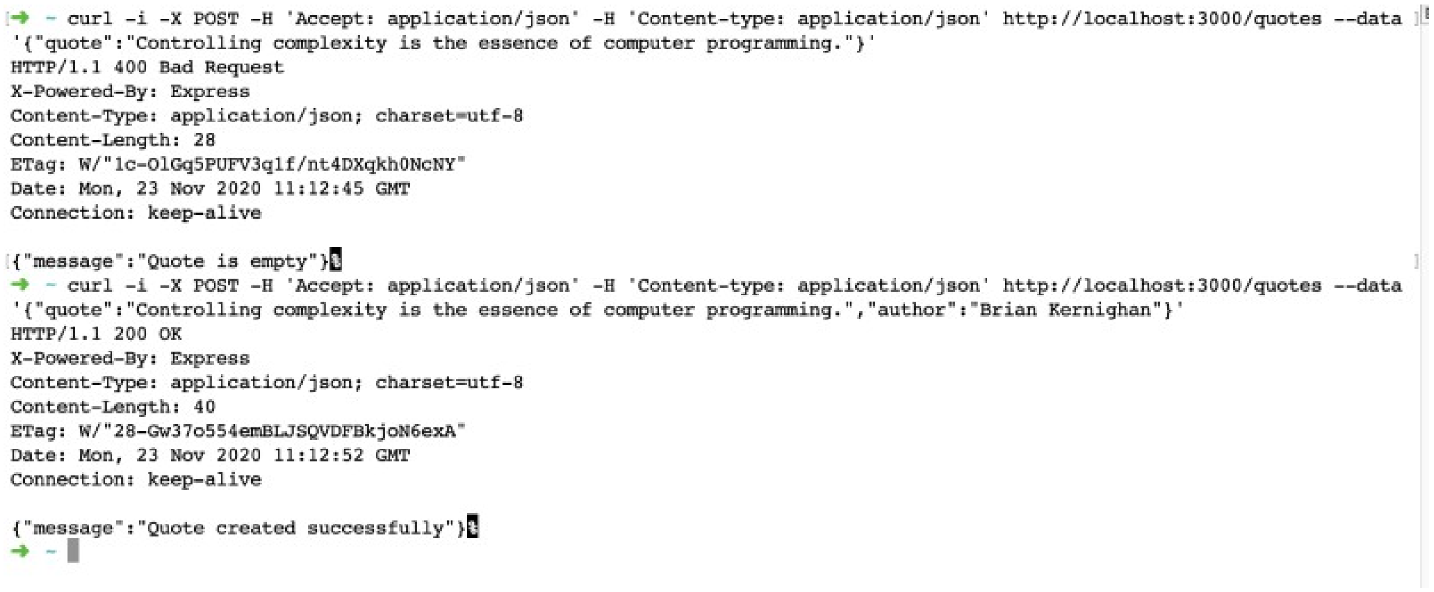
```
app.use((err, req, res, next) => {
  const statusCode = err.statusCode || 500;
  console.error(err.message, err.stack);
  res.status(statusCode).json({'message': err.message});

  return;
})

module.exports = app;
```

Make sure to put it just above the `module.exports = app` line so that they get executed after the routes. Now you can start your app and try the cURL commands below:

Copy

```
curl -i -X POST -H 'Accept: application/json' -H 'Content-type: application/json' h

curl -i -X POST -H 'Accept: application/json' -H 'Content-type: application/json' h
```

Below is the output of both cURL commands:


```
➜ ~ curl -i -X POST -H 'Accept: application/json' -H 'Content-type: application/json' http://localhost:3000/quotes --data
'{"quote":"Controlling complexity is the essence of computer programming."}'
HTTP/1.1 400 Bad Request
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 28
ETag: W/"1c-OlGq5PUFV3q1f/nt4DXqkh0NcNY"
Date: Mon, 23 Nov 2020 11:12:45 GMT
Connection: keep-alive

{"message":"Quote is empty"}
➜ ~ curl -i -X POST -H 'Accept: application/json' -H 'Content-type: application/json' http://localhost:3000/quotes --data
'{"quote":"Controlling complexity is the essence of computer programming.","author":"Brian Kernighan"}'
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 40
ETag: W/"28-Gw37o554emBLJSQVDFBkjoN6exA"
Date: Mon, 23 Nov 2020 11:12:52 GMT
Connection: keep-alive

{"message":"Quote created successfully"}
➜ ~
```

Depending on the configs and the database records you might get a `duplicate entry` error. Just change the quote to something different and try. The code changes for the POST quote API is in this pull request.

You can find lots of quotes to try here.

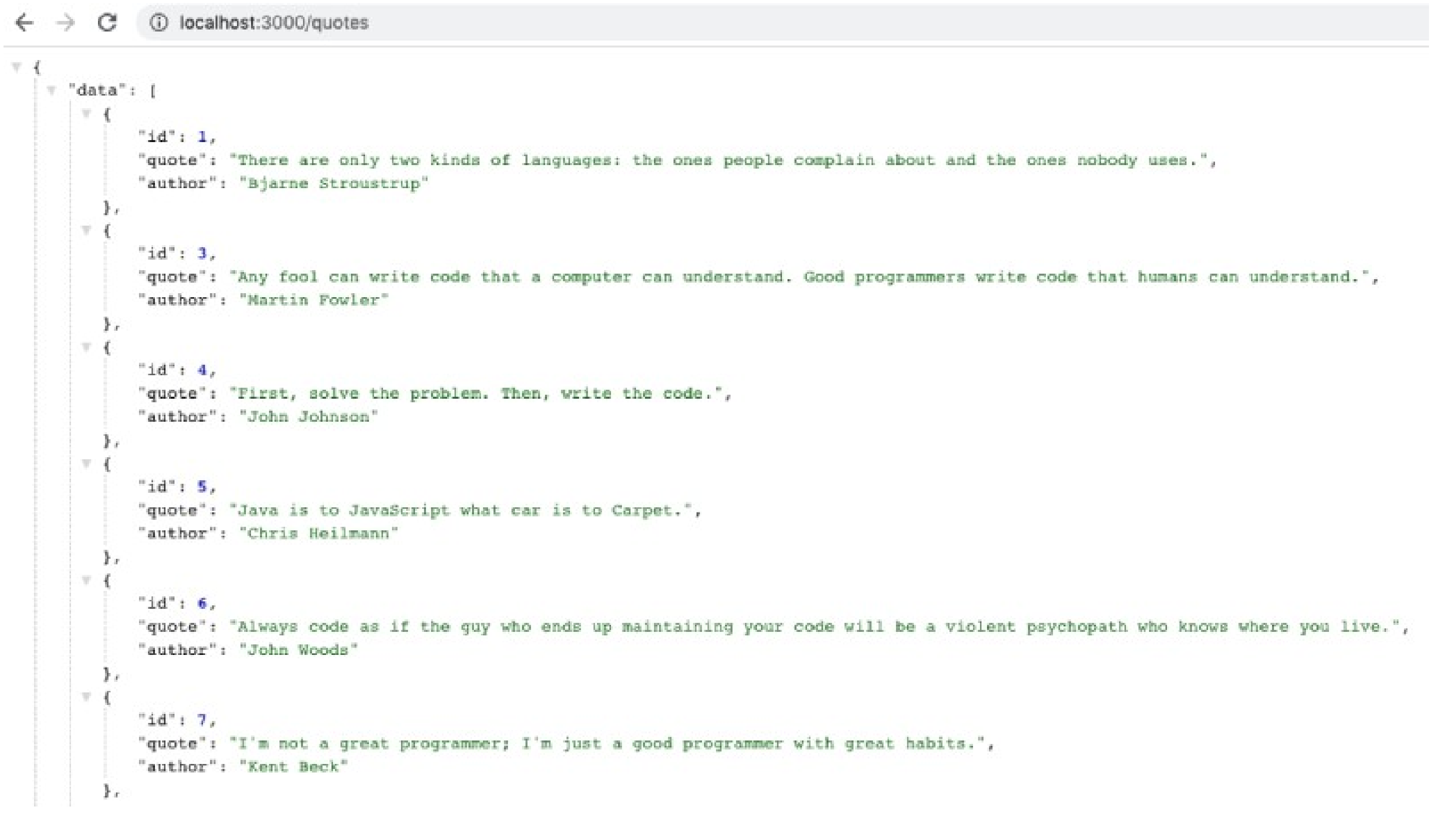
So there you have it, a basic RESTful API for quotes that can crate new quotes with a POST endpoint. There is a GET endpoint to fetch quotes with pagination.

If you want to try a hosted DB you can check this Node.js HarperDB tutorial.

TLDR; I want to run it quickly

As all the code is in a public Github repository, you can get started in no time running the following commands:

1. Clone the repository: `git clone git@github.com:geshan/nodejs-mysql.git`
2. Then run `cd nodejs-mysql`
3. After that execute `npm install`
4. Consequently run: `npm start`
5. Then hit `https://localhost:3000/quote` on your favorite browser
6. You should see something like below:


localhost:3000/quotes

```
{
  "data": [
    {
      "id": 1,
      "quote": "There are only two kinds of languages: the ones people complain about and the ones nobody uses.",
      "author": "Bjarne Stroustrup"
    },
    {
      "id": 3,
      "quote": "Any fool can write code that a computer can understand. Good programmers write code that humans can understand.",
      "author": "Martin Fowler"
    },
    {
      "id": 4,
      "quote": "First, solve the problem. Then, write the code.",
      "author": "John Johnson"
    },
    {
      "id": 5,
      "quote": "Java is to JavaScript what car is to Carpet.",
      "author": "Chris Heilmann"
    },
    {
      "id": 6,
      "quote": "Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.",
      "author": "John Woods"
    },
    {
      "id": 7,
      "quote": "I'm not a great programmer; I'm just a good programmer with great habits.",
      "author": "Kent Beck"
    },
```

You can look at the code and try to see how the whole thing is pieced together in this Node.js MySQL tutorial with Express Js for a quotes REST API. This API can act as a good base for a Node.js microservice.

You can dockerize the REST API app following this step-by-step tutorial. After that app is dockerized you can host it easily on something like Google Cloud Run. If you want a quick start testing ground without Docker, I would recommend Glitch. You can try 3 Node.js free hosting options too.

Conclusion

Creating a REST API with Node.js and MySQL is not that difficult.

> Some things are not taken care of in this Node.js MySQL tutorial with Express but this is an excellent good starting point.

Geshan Manandhar | 23-Nov-2020 Please share:

« How to use Docker with Node.js a step-by-step tutorial

How to make PHPUnit Code Coverage 2+ times faster with Pcov compared to Xdebug »

Comments

Post a Comment

RECENT POSTS

Using Jest beforeEach to write better unit tests in JavaScript, with code example

How to use TypeScript optional parameters with example code

10 useful Docker commands to get things done with a real-life example

How to append contents to a file using Node.js

Using Node.js readline to create a basic CLI app with Async await example

READ MORE ON

devops	**24**
misc	**47**
software engineering	**68**
talks	**19**
web development	**63**

All categories

SIDE PROJECTS

AU Tech Jobs

Nepal News English

Join the Newsletter

Receive exclusive content and links about software engineering and web development every month's first Monday.

The first issue has been published.

You email

Subscribe

We hate spam as much as you do. Unsubscribe at any time.

Copyright © 2022 - Geshan Manandhar