1. **Build Node.js Rest APIs with Express & MySQL**

[Last modified: April 15, 2020](https://bezkoder.com/node-js-rest-api-express-mysql/) [bezkoder](https://bezkoder.com/author/bezkoder/) [Node.js](https://bezkoder.com/category/node-js/)

[Express](https://expressjs.com/) is one of the most popular web frameworks for Node.js that supports routing, middleware, view system… This tutorial will guide you through the steps of building Node.js Restful CRUD API using Express and interacting with MySQL database.

Before reading the tutorial, please install MySQL in your machine. The installation instructions can be found at [Official MySQL installation manual](https://dev.mysql.com/doc/refman/5.7/en/installing.html).

Related Posts:  
– [Node.js Rest APIs example with Express & MySQL (including Sequelize)](https://bezkoder.com/node-js-express-sequelize-mysql/)  
– [Deploying/Hosting Node.js app on Heroku with MySQL database](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/)  
– [Node.js – JWT Authentication & Authorization example](https://bezkoder.com/node-js-jwt-authentication-mysql/)

Fullstack:  
– [Vue.js + Node.js + Express + MySQL](https://bezkoder.com/vue-js-node-js-express-mysql-crud-example/)  
– [Angular 8 + Node.js + Express + MySQL](https://bezkoder.com/angular-node-express-mysql/)  
– [React + Node.js + Express + MySQL](https://bezkoder.com/react-node-express-mysql/)

Node.js & MySQL Associations:  
– [One-to-Many Relationship example](https://bezkoder.com/sequelize-associate-one-to-many/)

Contents [[hide](https://bezkoder.com/node-js-rest-api-express-mysql/)]

* [Application overview](https://bezkoder.com/node-js-rest-api-express-mysql/#Application_overview)
* [Create Node.js application](https://bezkoder.com/node-js-rest-api-express-mysql/#Create_Nodejs_application)
* [Setup Express web server](https://bezkoder.com/node-js-rest-api-express-mysql/#Setup_Express_web_server)
* [Create MySQL table](https://bezkoder.com/node-js-rest-api-express-mysql/#Create_MySQL_table)
* [Configure & Connect to MySQL database](https://bezkoder.com/node-js-rest-api-express-mysql/#Configure_038_Connect_to_MySQL_database)
* [Define the Model](https://bezkoder.com/node-js-rest-api-express-mysql/#Define_the_Model)
* [Define Routes](https://bezkoder.com/node-js-rest-api-express-mysql/#Define_Routes)
* [Create the Controller](https://bezkoder.com/node-js-rest-api-express-mysql/#Create_the_Controller)
  + [Create a new object](https://bezkoder.com/node-js-rest-api-express-mysql/#Create_a_new_object)
  + [Retrieve all objects](https://bezkoder.com/node-js-rest-api-express-mysql/#Retrieve_all_objects)
  + [Retrieve a single object](https://bezkoder.com/node-js-rest-api-express-mysql/#Retrieve_a_single_object)
  + [Update an object](https://bezkoder.com/node-js-rest-api-express-mysql/#Update_an_object)
  + [Delete an object](https://bezkoder.com/node-js-rest-api-express-mysql/#Delete_an_object)
  + [Delete all objects](https://bezkoder.com/node-js-rest-api-express-mysql/#Delete_all_objects)
* [Test the APIs](https://bezkoder.com/node-js-rest-api-express-mysql/#Test_the_APIs)
* [Conclusion](https://bezkoder.com/node-js-rest-api-express-mysql/#Conclusion)
* [Further Reading](https://bezkoder.com/node-js-rest-api-express-mysql/#Further_Reading)
* [Source code](https://bezkoder.com/node-js-rest-api-express-mysql/#Source_code)

**Application overview**

We will build Rest Apis for creating, retrieving, updating & deleting Customers.

First, we start with an Express web server. Next, we add configuration for MySQL database, create Customer model, write the controller. Then we define routes for handling all CRUD operations:

| **Methods** | **Urls** | **Actions** |
| --- | --- | --- |
| GET | /customers | get all Customers |
| GET | /customers/42 | get Customer with id=42 |
| POST | /customers | add new Customer |
| PUT | /customers/42 | update Customer with id=42 |
| DELETE | /customers/42 | remove Customer with id=42 |
| DELETE | /customers | remove all Customers |

Finally, we’re gonna test the Rest Apis using Postman.

Our project structure will be like:

**Create Node.js application**

Open terminal/console, then create a folder for our application:

$ mkdir nodejs-express-mysql

$ cd nodejs-express-mysql

Initialize the Node.js application with a *package.json* file:

npm init

name: (nodejs-express-mysql)

version: (1.0.0)

description: Node.js Restful CRUD API with Node.js, Express and MySQL

entry point: (index.js) server.js

test command:

git repository:

keywords: nodejs, express, mysql, restapi

author: bezkoder

license: (ISC)

Is this ok? (yes) yes

Next, we need to install necessary modules: express, mysql and body-parser.  
Run the command:

npm install express mysql body-parser --save

The *package.json* file should look like this:

{

"name": "nodejs-express-mysql",

"version": "1.0.0",

"description": "Node.js Restful CRUD API with Node.js, Express and MySQL",

"main": "server.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [

"nodejs",

"express",

"mysql",

"restapi"

],

"author": "bezkoder",

"license": "ISC",

"dependencies": {

"body-parser": "^1.19.0",

"express": "^4.17.1",

"mysql": "^2.17.1"

}

}

**Setup Express web server**

Now, in the root folder, we create a new file named *server.js*:

const express = require("express");

const bodyParser = require("body-parser");

const app = express();

// parse requests of content-type: application/json

app.use(bodyParser.json());

// parse requests of content-type: application/x-www-form-urlencoded

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

// simple route

app.get("/", (req, res) => {

res.json({ message: "Welcome to bezkoder application." });

});

// set port, listen for requests

app.listen(3000, () => {

console.log("Server is running on port 3000.");

});

What we do are:  
– import express and body-parser modules. Express is for building the Rest apis, and [body-parser](https://www.npmjs.com/package/body-parser) helps to parse the request and create the req.body object that we will need to access in our routes.  
– create an Express app, then add body-parser middlewares using app.use() method.  
– define a GET route which is simple for test.  
– listen on port 3000 for incoming requests.

Now we can run the app with command: node server.js.  
Open your browser, enter the url <http://localhost:3000/>, you will see:

**Create MySQL table**

Before connecting Node.js Application with MySQL, we need a table first.  
So run the SQL script below to create customers table:

CREATE TABLE IF NOT EXISTS `customers` (

id int(11) NOT NULL PRIMARY KEY AUTO\_INCREMENT,

email varchar(255) NOT NULL,

name varchar(255) NOT NULL,

active BOOLEAN DEFAULT false

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

**Configure & Connect to MySQL database**

We’re gonna have a separate folder for configuration. Let’s create *config* folder in the *app* folder, under application root folder, then create *db.config.js* file inside that *config* folder with content like this:

module.exports = {

HOST: "localhost",

USER: "root",

PASSWORD: "123456",

DB: "testdb"

};

Now create a database connection that uses configuration above.  
The file for connection is *db.js*, we put it in *app/models* folder that will contain model in the next step.

const mysql = require("mysql");

const dbConfig = require("../config/db.config.js");

// Create a connection to the database

const connection = mysql.createConnection({

host: dbConfig.HOST,

user: dbConfig.USER,

password: dbConfig.PASSWORD,

database: dbConfig.DB

});

// open the MySQL connection

connection.connect(error => {

if (error) throw error;

console.log("Successfully connected to the database.");

});

module.exports = connection;

**Define the Model**

In *models* folder, create a file called *customer.model.js*. We’re gonna define constructor for Customer object here, and use the database connection above to write CRUD functions:

* create a new Customer
* find a Customer by id
* get all Customers
* update a Customer by id
* remove a Customer
* remove all Customers

This is the content inside *customer.model.js*:

const sql = require("./db.js");

// constructor

const Customer = function(customer) {

this.email = customer.email;

this.name = customer.name;

this.active = customer.active;

};

Customer.create = (newCustomer, result) => {

sql.query("INSERT INTO customers SET ?", newCustomer, (err, res) => {

if (err) {

console.log("error: ", err);

result(err, null);

return;

}

console.log("created customer: ", { id: res.insertId, ...newCustomer });

result(null, { id: res.insertId, ...newCustomer });

});

};

Customer.findById = (customerId, result) => {

sql.query(`SELECT \* FROM customers WHERE id = ${customerId}`, (err, res) => {

if (err) {

console.log("error: ", err);

result(err, null);

return;

}

if (res.length) {

console.log("found customer: ", res[0]);

result(null, res[0]);

return;

}

// not found Customer with the id

result({ kind: "not\_found" }, null);

});

};

Customer.getAll = result => {

sql.query("SELECT \* FROM customers", (err, res) => {

if (err) {

console.log("error: ", err);

result(null, err);

return;

}

console.log("customers: ", res);

result(null, res);

});

};

Customer.updateById = (id, customer, result) => {

sql.query(

"UPDATE customers SET email = ?, name = ?, active = ? WHERE id = ?",

[customer.email, customer.name, customer.active, id],

(err, res) => {

if (err) {

console.log("error: ", err);

result(null, err);

return;

}

if (res.affectedRows == 0) {

// not found Customer with the id

result({ kind: "not\_found" }, null);

return;

}

console.log("updated customer: ", { id: id, ...customer });

result(null, { id: id, ...customer });

}

);

};

Customer.remove = (id, result) => {

sql.query("DELETE FROM customers WHERE id = ?", id, (err, res) => {

if (err) {

console.log("error: ", err);

result(null, err);

return;

}

if (res.affectedRows == 0) {

// not found Customer with the id

result({ kind: "not\_found" }, null);

return;

}

console.log("deleted customer with id: ", id);

result(null, res);

});

};

Customer.removeAll = result => {

sql.query("DELETE FROM customers", (err, res) => {

if (err) {

console.log("error: ", err);

result(null, err);

return;

}

console.log(`deleted ${res.affectedRows} customers`);

result(null, res);

});

};

module.exports = Customer;

Customer model is simple, it contains fields: email, name & active.

We use database connection query() method to execute MySQL script: INSERT, SELECT, UPDATE, DELETE. You can find more details about mysql module at: <https://www.npmjs.com/package/mysql>.

**Define Routes**

When a client sends request for an endpoint using HTTP request (GET, POST, PUT, DELETE), we need to determine how the server will reponse. It’s why we’re gonna setup the routes.

These are routes we define:

* /customers: GET, POST, DELETE
* /customers/:customerId: GET, PUT, DELETE

Create a *routes* folder inside *app* folder with content like this:

module.exports = app => {

const customers = require("../controllers/customer.controller.js");

// Create a new Customer

app.post("/customers", customers.create);

// Retrieve all Customers

app.get("/customers", customers.findAll);

// Retrieve a single Customer with customerId

app.get("/customers/:customerId", customers.findOne);

// Update a Customer with customerId

app.put("/customers/:customerId", customers.update);

// Delete a Customer with customerId

app.delete("/customers/:customerId", customers.delete);

// Create a new Customer

app.delete("/customers", customers.deleteAll);

};

You can see that we use a controller from /controllers/customer.controller.js. It contains methods for handling CRUD operations and will be created in the next step.

We also need to include routes in *server.js* (right before app.listen()):

...

require("./app/routes/customer.routes.js")(app);

app.listen(...);

**Create the Controller**

Now we create a *controllers* folder inside *app* folder, then we have a file named *customer.controller.js*. Our controller will be written inside this with CRUD functions:

* create
* findAll
* findOne
* update
* delete
* deleteAll

const Customer = require("../models/customer.model.js");

// Create and Save a new Customer

exports.create = (req, res) => {

};

// Retrieve all Customers from the database.

exports.findAll = (req, res) => {

};

// Find a single Customer with a customerId

exports.findOne = (req, res) => {

};

// Update a Customer identified by the customerId in the request

exports.update = (req, res) => {

};

// Delete a Customer with the specified customerId in the request

exports.delete = (req, res) => {

};

// Delete all Customers from the database.

exports.deleteAll = (req, res) => {

};

Let’s implement these functions.

**Create a new object**

Create and Save a new Customer:

exports.create = (req, res) => {

// Validate request

if (!req.body) {

res.status(400).send({

message: "Content can not be empty!"

});

}

// Create a Customer

const customer = new Customer({

email: req.body.email,

name: req.body.name,

active: req.body.active

});

// Save Customer in the database

Customer.create(customer, (err, data) => {

if (err)

res.status(500).send({

message:

err.message || "Some error occurred while creating the Customer."

});

else res.send(data);

});

};

**Retrieve all objects**

Retrieve all Customers from the database:

exports.findAll = (req, res) => {

Customer.getAll((err, data) => {

if (err)

res.status(500).send({

message:

err.message || "Some error occurred while retrieving customers."

});

else res.send(data);

});

};

**Retrieve a single object**

Find a single Customer with a customerId:

exports.findOne = (req, res) => {

Customer.findById(req.params.customerId, (err, data) => {

if (err) {

if (err.kind === "not\_found") {

res.status(404).send({

message: `Not found Customer with id ${req.params.customerId}.`

});

} else {

res.status(500).send({

message: "Error retrieving Customer with id " + req.params.customerId

});

}

} else res.send(data);

});

};

**Update an object**

Update a Customer identified by the customerId in the request:

exports.update = (req, res) => {

// Validate Request

if (!req.body) {

res.status(400).send({

message: "Content can not be empty!"

});

}

Customer.updateById(

req.params.customerId,

new Customer(req.body),

(err, data) => {

if (err) {

if (err.kind === "not\_found") {

res.status(404).send({

message: `Not found Customer with id ${req.params.customerId}.`

});

} else {

res.status(500).send({

message: "Error updating Customer with id " + req.params.customerId

});

}

} else res.send(data);

}

);

};

**Delete an object**

Delete a Customer with the specified customerId in the request:

exports.delete = (req, res) => {

Customer.remove(req.params.customerId, (err, data) => {

if (err) {

if (err.kind === "not\_found") {

res.status(404).send({

message: `Not found Customer with id ${req.params.customerId}.`

});

} else {

res.status(500).send({

message: "Could not delete Customer with id " + req.params.customerId

});

}

} else res.send({ message: `Customer was deleted successfully!` });

});

};

**Delete all objects**

Delete all Customers from the database:

exports.deleteAll = (req, res) => {

Customer.removeAll((err, data) => {

if (err)

res.status(500).send({

message:

err.message || "Some error occurred while removing all customers."

});

else res.send({ message: `All Customers were deleted successfully!` });

});

};

**Test the APIs**

Run our Node.js application with command: node server.js.  
The console shows:

Server is running on port 3000.

Successfully connected to the database.

Using Postman, we’re gonna test all the Apis above.

1. **Create a new Customer using POST /customers Api**

After creating some new Customers, we can check MySQL table:

mysql> SELECT \* FROM customers;

+----+--------------------+--------+--------+

| id | email | name | active |

+----+--------------------+--------+--------+

| 1 | bezkoder@gmail.com | zKoder | 1 |

| 2 | jack123@gmail.com | Jack | 0 |

| 3 | drhelen@gmail.com | Helen | 0 |

+----+--------------------+--------+--------+

1. **Retrieve all Customers using GET /customers Api**

1. **Retrieve a single Customer by id using GET /customers/:customerId Api**

1. **Update a Customer using PUT /customers/:customerId Api**

Check customers table after a row was updated:

mysql> SELECT \* FROM customers;

+----+--------------------+----------+--------+

| id | email | name | active |

+----+--------------------+----------+--------+

| 1 | bezkoder@gmail.com | zKoder | 1 |

| 2 | jack123@gmail.com | Jack | 0 |

| 3 | drhelen@gmail.com | Dr.Helen | 1 |

+----+--------------------+----------+--------+

1. **Delete a Customer using DELETE /customers/:customerId Api**

Customer with id=2 was removed from customers table:

mysql> SELECT \* FROM customers;

+----+--------------------+----------+--------+

| id | email | name | active |

+----+--------------------+----------+--------+

| 1 | bezkoder@gmail.com | zKoder | 1 |

| 3 | drhelen@gmail.com | Dr.Helen | 1 |

+----+--------------------+----------+--------+

1. **Delete all Customers using DELETE /customers Api**

Now there are no rows in customers table:

mysql> SELECT \* FROM customers;

Empty set (0.00 sec)

**Conclusion**

Today, we’ve learned how to create Node.js Rest Apis with an Express web server. We also know way to add configuration for MySQL database, create a model, write a controller and define routes for handling all CRUD operations.

You can find more interesting thing in the next tutorial:  
[Deploying/Hosting Node.js app on Heroku with MySQL database](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/)

If you want to use Sequelize to reduce boilerplate code, there is a post for this:  
[Node.js Rest APIs example with Express, Sequelize & MySQL](https://bezkoder.com/node-js-express-sequelize-mysql/)

Happy learning! See you again.

**Further Reading**

* [Express.js Routing](http://expressjs.com/en/guide/routing.html)
* <https://www.npmjs.com/package/express>
* <https://www.npmjs.com/package/body-parser>
* <https://www.npmjs.com/package/mysql>

**Source code**

….

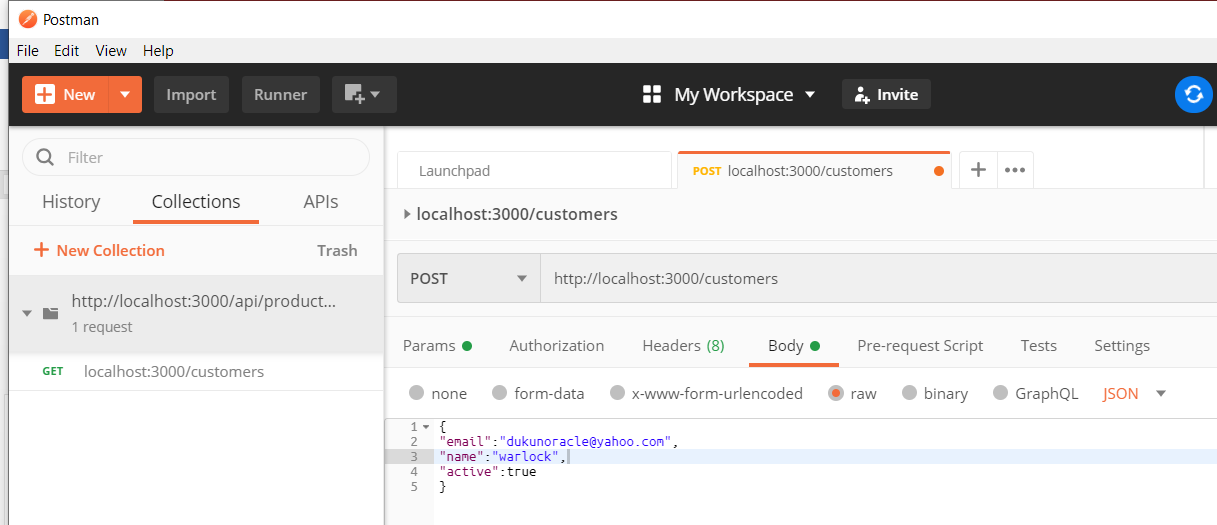
Install postman

Sign in

[Dukunoracle@yahoo.com](mailto:Dukunoracle@yahoo.com)

dukunoracle

warlock11x /postman11x



<http://localhost:3000/customers>

post

json

{

"email":"dukunoracle@yahoo.com",

"name":"warlock",

"active":true

}

B) DEPLOY

**Deploying/Hosting Node.js app on Heroku with MySQL database**

[Last modified: April 15, 2020](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/) [bezkoder](https://bezkoder.com/author/bezkoder/) [Deployment](https://bezkoder.com/category/deployment/), [Node.js](https://bezkoder.com/category/node-js/)

[Heroku](https://www.heroku.com/) is one of the most popular cloud platform as a service (PaaS). We can use it to deploy and manage our application in simple & convenient way. This tutorial guides you through the steps of deploying Node.js app on Heroku with ClearDB and MySQL database. The deployment will be done using Git and [Heroku CLI](https://devcenter.heroku.com/articles/heroku-cli).

Related Posts:  
– [Build Node.js Rest APIs with Express & MySQL](https://bezkoder.com/node-js-rest-api-express-mysql/)  
– [Node.js Rest APIs example with Express, Sequelize & MySQL](https://bezkoder.com/node-js-express-sequelize-mysql/)

Contents [[hide](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/)]

* [Create a Node.js app for deployment](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Create_a_Nodejs_app_for_deployment)
* [Download and Install Heroku](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Download_and_Install_Heroku)
* [Log in to Heroku](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Log_in_to_Heroku)
* [Setup Git and Create a Heroku app](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Setup_Git_and_Create_a_Heroku_app)
* [Deploy the app to Heroku](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Deploy_the_app_to_Heroku)
* [Configure MySQL Database for Node.js on Heroku app](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Configure_MySQL_Database_for_Nodejs_on_Heroku_app)
  + [Setup ClearDB](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Setup_ClearDB)
  + [Configure MySQL connection](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Configure_MySQL_connection)
  + [Config Node.js to connect ClearDB MySQL on Heroku](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Config_Nodejs_to_connect_ClearDB_MySQL_on_Heroku)
  + [Create MySQL table on ClearDB](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Create_MySQL_table_on_ClearDB)
* [Test the Heroku app on local](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Test_the_Heroku_app_on_local)
* [Test the Node.js Rest Apis on Heroku](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Test_the_Nodejs_Rest_Apis_on_Heroku)
  + [Error code=H10 desc=”App crashed”](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Error_codeH10_desc8221App_crashed8221)
  + [Error R10: failed to bind to $PORT](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Error_R10_failed_to_bind_to_PORT)
  + [Error Connection lost: The server closed the connection](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Error_Connection_lost_The_server_closed_the_connection)
* [Conclusion](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Conclusion)
* [Further Reading](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Further_Reading)
* [Source code](https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/#Source_code)

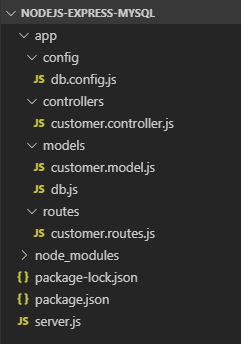
**Create a Node.js app for deployment**

In previous tutorial, we’ve built a [Restful CRUD API with Node.js, Express and MySQL](https://bezkoder.com/node-js-rest-api-express-mysql/).

The routes for handling all CRUD operations looks like this:

| **Methods** | **Urls** | **Actions** |
| --- | --- | --- |
| GET | /customers | get all Customers |
| GET | /customers/42 | get Customer with id=42 |
| POST | /customers | add new Customer |
| PUT | /customers/42 | update Customer with id=42 |
| DELETE | /customers/42 | remove Customer with id=42 |
| DELETE | /customers | remove all Customers |

Our project structure’s like:



You can clone the sample app directly from [Github](https://github.com/bezkoder/nodejs-express-mysql).  
We’re gonna deploy the app using Git and Heroku CLI.

**Download and Install Heroku**

Heroku CLI is a command line application that we can use to create, deploy and manage Heroku apps.  
Now go to [Heroku Dev Center](https://devcenter.heroku.com/articles/heroku-cli#download-and-install), follow the instructions to download & install Heroku CLI.

**Log in to Heroku**

First we need a Heroku account and it can be created from the [Signup Page](https://signup.heroku.com/).  
After having an account, open command prompt, run the command:

$ heroku login

It will prompt us to enter our Heroku account (email & password) in a browser window. Then we will receive:

heroku: Press any key to open up the browser to login or q to exit:

Opening browser to https://cli-auth.heroku.com/auth/browser/xxx

Logging in... done

Logged in as bezkoder@gmail.com

**Setup Git and Create a Heroku app**

Point the command prompt to our project root directory, then create Git repository:

$ cd nodejs-express-mysql

$ git init

$ git add .

$ git commit -m "initial commit"

Create Heroku app with command: heroku create [app-name].

$ heroku create bezkoder-nodejs-mysql

Creating ⬢ bezkoder-nodejs-mysql... done

https://bezkoder-nodejs-mysql.herokuapp.com/ | https://git.heroku.com/bezkoder-nodejs-mysql.git

Now we have initialized a new Heroku app, we also create a remote repository at Heroku:  
https://git.heroku.com/bezkoder-nodejs-mysql.git

If we don’t pass a name, just run the command: heroku create, Heroku will generate a random name:

$ heroku create

Creating app... done, ⬢ tranquil-woodland-06325

https://tranquil-woodland-06325.herokuapp.com/ | https://git.heroku.com/tranquil-woodland-06325.git

And we need to rename the app using heroku apps:rename.

$ heroku apps:rename --app tranquil-woodland-06325 bezkoder-nodejs-mysql

You can check the result of this step with git remote -v:

$ git remote -v

heroku https://git.heroku.com/bezkoder-nodejs-mysql.git (fetch)

heroku https://git.heroku.com/bezkoder-nodejs-mysql.git (push)

**Deploy the app to Heroku**

We can easily deploy our Node.js app to Heroku by pushing the code to the remote repository that we created at the previous step. Heroku will automatically detects that this is a Node.js app and builds it accordingly.

$ git push heroku master

Enumerating objects: 16, done.

Counting objects: 100% (16/16), done.

Delta compression using up to 4 threads

Compressing objects: 100% (13/13), done.

Writing objects: 100% (16/16), 3.41 KiB | 290.00 KiB/s, done.

Total 16 (delta 0), reused 0 (delta 0)

remote: Compressing source files... done.

remote: Building source:

remote:

remote: -----> Node.js app detected

remote:

remote: -----> Creating runtime environment

remote:

remote: NPM\_CONFIG\_LOGLEVEL=error

remote: NODE\_ENV=production

remote: NODE\_MODULES\_CACHE=true

remote: NODE\_VERBOSE=false

remote:

remote: -----> Installing binaries

remote: engines.node (package.json): unspecified

remote: engines.npm (package.json): unspecified (use default)

remote:

remote: Resolving node version 10.x...

remote: Downloading and installing node 10.16.3...

remote: Using default npm version: 6.9.0

remote:

remote: -----> Installing dependencies

remote: Installing node modules (package.json)

remote: added 59 packages from 48 contributors and audited 171 packages in 2.351s

remote: found 0 vulnerabilities

remote:

remote:

remote: -----> Build

remote:

remote: -----> Pruning devDependencies

remote: audited 171 packages in 1s

remote: found 0 vulnerabilities

remote:

remote:

remote: -----> Caching build

remote: - node\_modules

remote:

remote: -----> Build succeeded!

remote: -----> Discovering process types

remote: Procfile declares types -> (none)

remote: Default types for buildpack -> web

remote:

remote: -----> Compressing...

remote: Done: 20M

remote: -----> Launching...

remote: Released v3

remote: https://bezkoder-nodejs-mysql.herokuapp.com/ deployed to Heroku

remote:

remote: Verifying deploy... done.

To https://git.heroku.com/bezkoder-nodejs-mysql.git

\* [new branch] master -> master

But currently our Node.js app doesn’t connect with any MySQL database. We need to come to next step to make it work.

**Configure MySQL Database for Node.js on Heroku app**

**Setup ClearDB**

Heroku provides PostgreSQL as default database engine for our application. In this tutorial, we’re gonna work with MySQL database, so service provider called [ClearDB](https://elements.heroku.com/addons/cleardb) will be used.

To get started, install ClearDB add-on to our application with command:

$ heroku addons:create cleardb:ignite

Creating cleardb:ignite on ⬢ bezkoder-nodejs-mysql... free

Created cleardb-cubed-17749 as CLEARDB\_DATABASE\_URL

Use heroku addons:docs cleardb to view documentation

**Configure MySQL connection**

After installing the Add-ons, we can get our database URL by running the command:

$ heroku config | grep CLEARDB\_DATABASE\_URL

If you get the error on Window Os:  
'grep' is not recognized as an internal or external command, operable program or batch file.  
You can use this command instead:

$ heroku config | findstr CLEARDB\_DATABASE\_URL

CLEARDB\_DATABASE\_URL: mysql://b7e2437887xxxa:0200xxx6@us-cdbr-iron-east-02.cleardb.net/heroku\_7643ec736354xxx?reconnect=true

Copy the value of the CLEARDB\_DATABASE\_URL config variable and use it in the following command:

$ heroku config:set DATABASE\_URL='mysql://b7e2437887xxxa:0200xxx6@us-cdbr-iron-east-02.cleardb.net/heroku\_7643ec736354xxx?reconnect=true'

Setting DATABASE\_URL and restarting ⬢ bezkoder-nodejs-mysql... done, v6

DATABASE\_URL: 'mysql://b7e2437887xxxa:0200xxx6@us-cdbr-iron-east-02.cleardb.net/heroku\_7643ec736354xxx?reconnect=true'

**Config Node.js to connect ClearDB MySQL on Heroku**

In the previous step, we get DATABASE\_URL that contains:

* b7e2437887xxxa: username
* 0200xxx6: password
* us-cdbr-iron-east-02.cleardb.net: host
* heroku\_7643ec736354xxx: database

Now we’re gonna use these parameters to config our database connection in Node.js app.  
Open *db.config.js* file, change the configuration like this:

module.exports = {

HOST: "us-cdbr-iron-east-02.cleardb.net",

USER: "b7e2437887xxxa",

PASSWORD: "0200xxx6",

DB: "heroku\_7643ec736354xxx"

};

Don’t forget to push the updated code to Heroku remote repository.

$ git add .

$ git commit

$ git push heroku master

**Create MySQL table on ClearDB**

Before testing our app on Heroku, we need to create MySQL table named customers on ClearDB. Using the connect parameters above, open another command prompt, run the command:

mysql --host=us-cdbr-iron-east-02.cleardb.net --user=b7e2437887xxxa --password=0200xxx6 --reconnect heroku\_7643ec736354xxx

Create the table with SQL script like this:

mysql> CREATE TABLE IF NOT EXISTS `customers` (

id int(11) NOT NULL PRIMARY KEY AUTO\_INCREMENT,

email varchar(255) NOT NULL,

name varchar(255) NOT NULL,

active BOOLEAN DEFAULT false

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

Check the table:

mysql> DESCRIBE customers;

+--------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------+--------------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| email | varchar(255) | NO | | NULL | |

| name | varchar(255) | NO | | NULL | |

| active | tinyint(1) | YES | | 0 | |

+--------+--------------+------+-----+---------+----------------+

**Test the Heroku app on local**

To test our app in local machine, run the command:

$ heroku local

9:50:46 AM web.1 | > nodejs-express-mysql@1.0.0 start D:\Projects\NodeJs\nodejs-express-mysql

9:50:46 AM web.1 | > node server.js

9:50:47 AM web.1 | Server is running on port 5000.

9:50:48 AM web.1 | Successfully connected to the database.

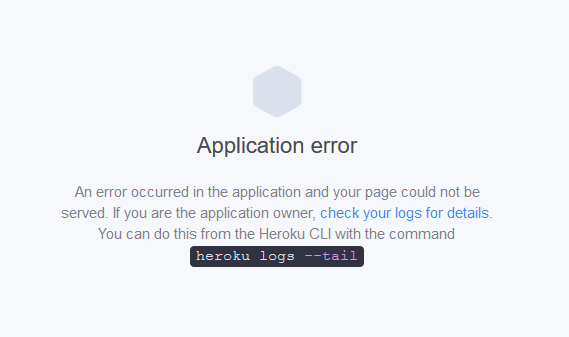
Now you can make CRUD operations with our Rest Apis using the url: http://localhost:5000/customers.

But for https://bezkoder-nodejs-mysql.herokuapp.com/customers, we’ve not done yet and will get some errors.

**Test the Node.js Rest Apis on Heroku**

Run the command: heroku run node server.js.

Currently, if you make any request to https://bezkoder-nodejs-mysql.herokuapp.com/customers, you will see:



Run the command heroku logs --tail, there are 2 errors that occur frequently:

**Error code=H10 desc=”App crashed”**

heroku[router]: at=error code=H10 desc="App crashed" method=GET path=/... host=...

**Error R10: failed to bind to $PORT**

Error R10 (Boot timeout) -> Web process failed to bind to $PORT within 60 seconds of launch

This is because our Node.js app is configured to bind to port 3000, but Heroku uses the $PORT environment variable, and it is dynamic.

To fix this issue, just change our code like this:

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}.`);

});

**Error Connection lost: The server closed the connection**

We may also get this database connection error. This is because the MySQL server on ClearDB closed the connection. You can detect the disconnect event, then recreate the connection.

Another way to fix this is to use connection pooling. Open *db.js* and change its code to new code like this:

const mysql = require("mysql");

const dbConfig = require("../config/db.config.js");

var connection = mysql.createPool({

host: dbConfig.HOST,

user: dbConfig.USER,

password: dbConfig.PASSWORD,

database: dbConfig.DB

});

module.exports = connection;

**Note:** After fixing the issues, you should update code to Heroku remote repository.

$ git add .

$ git commit

$ git push heroku master

Now we can make CRUD operations to Rest Apis on Heroku server with command above: heroku run node server.js.

Or restart the server with: heroku restart.

**Conclusion**

Today we’ve learned how to deploy Node.js app on Heroku, we also know way to config ClearDB add-on to work with MySQL Database, then we test the Heroku app on local and Heroku server with fixing Error H10, R10, database connection lost.

There are many things to do, but hope you can cover them all.  
Happy learning! See you again.

**Further Reading**

* [Heroku: Deploying with Git](https://devcenter.heroku.com/articles/git)
* [ClearDB MySQL | Heroku Dev Center](https://devcenter.heroku.com/articles/cleardb)
* [MySQL pooling connection](https://github.com/mysqljs/mysql#pooling-connections)

**Source code**

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

https://devcenter.heroku.com/articles/heroku-cli#download-and-install