**https://devdotcode.com/how-to-use-sequelize-async-await-to-interact-with-mysql-database-in-node-js/**

**Node.js Rest APIs example with Express, Sequelize & MySQL**

[Express](https://expressjs.com/) is one of the most popular web frameworks for Node.js that supports routing, middleware, view system… [Sequelize](https://sequelize.org/) is a promise-based Node.js ORM that supports the dialects for Postgres, MySQL, SQL Server… In this tutorial, I will show you step by step to build Node.js Restful CRUD API using Express, Sequelize with MySQL database.

You should install MySQL in your machine first. The installation instructions can be found at [Official MySQL installation manual](https://dev.mysql.com/doc/refman/5.7/en/installing.html).

## Node.js Rest CRUD API overview

We will build Rest Apis that can create, retrieve, update, delete and find Tutorials by title.

First, we start with an Express web server. Next, we add configuration for MySQL database, create Tutorial model with Sequelize, write the controller. Then we define routes for handling all CRUD operations (including custom finder).

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

This is our project structure

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

This is our project structure:

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

This is our project structure:

## Create Node.js App

First, we create a folder:

$ mkdir nodejs-express-sequelize-mysql

$ cd nodejs-express-sequelize-mysql

Next, we initialize the Node.js App with a package.json file:

npm init

name: (nodejs-express-sequelize-mysql)

version: (1.0.0)

description: Node.js Rest Apis with Express, Sequelize & MySQL.

entry point: (index.js) server.js

test command:

git repository:

keywords: nodejs, express, sequelize, mysql, rest, api

author: bezkoder

license: (ISC)

Is this ok? (yes) yes

We need to install necessary modules: express, sequelize, mysql2 and cors.  
Run the command:

npm install express sequelize mysql2 cors --save

The package.json file should look like this:

{

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

"version": "1.0.0",

"description": "Node.js Rest Apis with Express, Sequelize & MySQL",

"main": "server.js",

"scripts": {

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

},

"keywords": [

"nodejs",

"express",

"rest",

"api",

"sequelize",

"mysql"

],

"author": "Appolo Tankeh",

"license": "ISC",

"dependencies": {

"cors": "^2.8.5",

"express": "^4.17.1",

"mysql2": "^2.0.2",

"sequelize": "^6.21.0"

}

}

## Setup Express web server

In the root folder, let’s create a new server.js file:

const express = require("express");

const cors = require("cors");

const app = express();

var corsOptions = {

origin: "http://localhost:8081"

};

app.use(cors(corsOptions));

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

app.use(express.json());

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

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

// simple route

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

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

});

// set port, listen for requests

const PORT = process.env.PORT || 8080;

app.listen(PORT, () => {

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

});

What we do are:  
– import express, and cors modules:

* Express is for building the Rest apis
* [cors](https://www.npmjs.com/package/cors) provides Express middleware to enable CORS with various options.

– create an Express app, then add body-parser (json and urlencoded) and corsmiddlewares using app.use() method. Notice that we set origin: http://localhost:8081.  
– define a GET route which is simple for test.  
– listen on port 8080 for incoming requests.

Now let’s run the app with command: node server.js.  
Open your browser with url <http://localhost:8080/>, you will see:

## Configure MySQL database & Sequelize

In the app folder, we create a separate config folder for configuration with db.config.jsfile like this:

module.exports = {

HOST: "localhost",

USER: "root",

PASSWORD: "123456",

DB: "testdb",

dialect: "mysql",

pool: {

max: 5,

min: 0,

acquire: 30000,

idle: 10000

}

};

First five parameters are for MySQL connection.  
pool is optional, it will be used for Sequelize connection pool configuration:

* max: maximum number of connection in pool
* min: minimum number of connection in pool
* idle: maximum time, in milliseconds, that a connection can be idle before being released
* acquire: maximum time, in milliseconds, that pool will try to get connection before throwing error

For more details, please visit [API Reference for the Sequelize constructor](https://sequelize.org/master/class/lib/sequelize.js~Sequelize.html#instance-constructor-constructor).

## Initialize Sequelize

We’re gonna initialize Sequelize in **app**/**models** folder that will contain model in the next step.

Now create **app**/**models**/index.js with the following code:

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

const Sequelize = require("sequelize");

const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {

host: dbConfig.HOST,

dialect: dbConfig.dialect,

operatorsAliases: false,

pool: {

max: dbConfig.pool.max,

min: dbConfig.pool.min,

acquire: dbConfig.pool.acquire,

idle: dbConfig.pool.idle

}

});

const db = {};

db.Sequelize = Sequelize;

db.sequelize = sequelize;

db.tutorials = require("./tutorial.model.js")(sequelize, Sequelize);

module.exports = db;

Don’t forget to call sync() method in *server.js*:

...

const app = express();

app.use(...);

const db = require("./app/models");

db.sequelize.sync()

.then(() => {

console.log("Synced db.");

})

.catch((err) => {

console.log("Failed to sync db: " + err.message);

});

In development, you may need to drop existing tables and re-sync database. Just use force: true as following code:

db.sequelize.sync({ force: true }).then(() => {

console.log("Drop and re-sync db.");

});

## Define the Sequelize Model

In models folder, create tutorial.model.js file like this:

module.exports = (sequelize, Sequelize) => {

const Tutorial = sequelize.define("tutorial", {

title: {

type: Sequelize.STRING

},

description: {

type: Sequelize.STRING

},

published: {

type: Sequelize.BOOLEAN

}

});

return Tutorial;

};

This Sequelize Model represents **tutorials** table in MySQL database. These columns will be generated automatically: *id*, *title*, *description*, *published*, *createdAt*, *updatedAt*.

After initializing Sequelize, we don’t need to write CRUD functions, Sequelize supports all of them:

* create a new Tutorial: [create](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-create)(object)
* find a Tutorial by id: [findByPk](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findByPk)(id)
* get all Tutorials: [findAll](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findAll)()
* update a Tutorial by id: [update](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-update)(data, where: { id: id })
* remove a Tutorial: [destroy](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-destroy)(where: { id: id })
* remove all Tutorials: destroy(where: {})
* find all Tutorials by title: findAll({ where: { title: ... } })

These functions will be used in our Controller.

We can improve the example by adding Comments for each Tutorial. It is the One-to-Many Relationship.

This Sequelize Model represents **tutorials** table in MySQL database. These columns will be generated automatically: *id*, *title*, *description*, *published*, *createdAt*, *updatedAt*.

After initializing Sequelize, we don’t need to write CRUD functions, Sequelize supports all of them:

* create a new Tutorial: [create](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-create)(object)
* find a Tutorial by id: [findByPk](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findByPk)(id)
* get all Tutorials: [findAll](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findAll)()
* update a Tutorial by id: [update](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-update)(data, where: { id: id })
* remove a Tutorial: [destroy](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-destroy)(where: { id: id })
* remove all Tutorials: destroy(where: {})
* find all Tutorials by title: findAll({ where: { title: ... } })

These functions will be used in our Controller.

## Create the Controller

Inside **app**/**controllers** folder, let’s create tutorial.controller.js with these CRUD functions:

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

const db = require("../models");

const Tutorial = db.tutorials;

const Op = db.Sequelize.Op;

// Create and Save a new Tutorial

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

};

// Retrieve all Tutorials from the database.

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

};

// Find a single Tutorial with an id

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

};

// Update a Tutorial by the id in the request

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

};

// Delete a Tutorial with the specified id in the request

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

};

// Delete all Tutorials from the database.

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

};

// Find all published Tutorials

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

};

Let’s implement these functions.

### Create a new object

Create and Save a new Tutorial:

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

// Validate request

if (!req.body.title) {

res.status(400).send({

message: "Content can not be empty!"

});

return;

}

// Create a Tutorial

const tutorial = {

title: req.body.title,

description: req.body.description,

published: req.body.published ? req.body.published : false

};

// Save Tutorial in the database

Tutorial.create(tutorial)

.then(data => {

res.send(data);

})

.catch(err => {

res.status(500).send({

message:

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

});

});

};

### Retrieve objects (with condition)

Retrieve all Tutorials/ find by title from the database:

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

const title = req.query.title;

var condition = title ? { title: { [Op.like]: `%${title}%` } } : null;

Tutorial.findAll({ where: condition })

.then(data => {

res.send(data);

})

.catch(err => {

res.status(500).send({

message:

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

});

});

};

We use req.query.title to get query string from the Request and consider it as condition for findAll() method.

### Retrieve a single object

Find a single Tutorial with an id:

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

const id = req.params.id;

Tutorial.findByPk(id)

.then(data => {

if (data) {

res.send(data);

} else {

res.status(404).send({

message: `Cannot find Tutorial with id=${id}.`

});

}

})

.catch(err => {

res.status(500).send({

message: "Error retrieving Tutorial with id=" + id

});

});

};

### Update an object

Update a Tutorial identified by the id in the request:

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

const id = req.params.id;

Tutorial.update(req.body, {

where: { id: id }

})

.then(num => {

if (num == 1) {

res.send({

message: "Tutorial was updated successfully."

});

} else {

res.send({

message: `Cannot update Tutorial with id=${id}. Maybe Tutorial was not found or req.body is empty!`

});

}

})

.catch(err => {

res.status(500).send({

message: "Error updating Tutorial with id=" + id

});

});

};

### Delete an object

Delete a Tutorial with the specified id:

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

const id = req.params.id;

Tutorial.destroy({

where: { id: id }

})

.then(num => {

if (num == 1) {

res.send({

message: "Tutorial was deleted successfully!"

});

} else {

res.send({

message: `Cannot delete Tutorial with id=${id}. Maybe Tutorial was not found!`

});

}

})

.catch(err => {

res.status(500).send({

message: "Could not delete Tutorial with id=" + id

});

});

};

### Delete all objects

Delete all Tutorials from the database:

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

Tutorial.destroy({

where: {},

truncate: false

})

.then(nums => {

res.send({ message: `${nums} Tutorials were deleted successfully!` });

})

.catch(err => {

res.status(500).send({

message:

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

});

});

};

### Find all objects by condition

Find all Tutorials with published = true:

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

Tutorial.findAll({ where: { published: true } })

.then(data => {

res.send(data);

})

.catch(err => {

res.status(500).send({

message:

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

});

});

};

This controller can be modified a little to return pagination response:

{

"totalItems": 8,

"tutorials": [...],

"totalPages": 3,

"currentPage": 1

}

You can find more details at:  
[Server side Pagination in Node.js with Sequelize and MySQL](https://bezkoder.com/node-js-sequelize-pagination-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 by setting up the routes.

These are our routes:

* /api/tutorials: GET, POST, DELETE
* /api/tutorials/:id: GET, PUT, DELETE
* /api/tutorials/published: GET

Create a turorial.routes.js inside app/routes folder with content like this:

module.exports = app => {

const tutorials = require("../controllers/tutorial.controller.js");

var router = require("express").Router();

// Create a new Tutorial

router.post("/", tutorials.create);

// Retrieve all Tutorials

router.get("/", tutorials.findAll);

// Retrieve all published Tutorials

router.get("/published", tutorials.findAllPublished);

// Retrieve a single Tutorial with id

router.get("/:id", tutorials.findOne);

// Update a Tutorial with id

router.put("/:id", tutorials.update);

// Delete a Tutorial with id

router.delete("/:id", tutorials.delete);

// Delete all Tutorials

router.delete("/", tutorials.deleteAll);

app.use('/api/tutorials', router);

};

You can see that we use a controller from /controllers/tutorial.controller.js.

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

...

require("./app/routes/turorial.routes")(app);

// set port, listen for requests

const PORT = ...;

app.listen(...);

## Test the APIs

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

Server is running on port 8080.

Executing (default): DROP TABLE IF EXISTS `tutorials`;

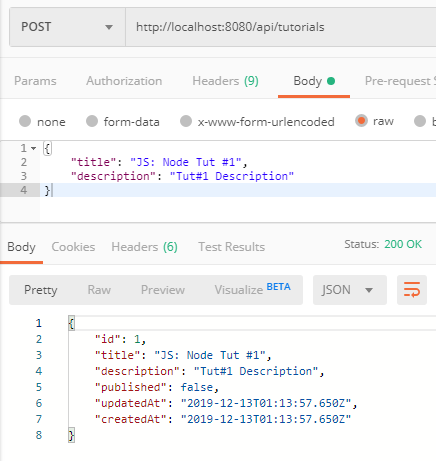
Executing (default): CREATE TABLE IF NOT EXISTS `tutorials` (`id` INTEGER NOT NULL auto\_increment , `title` VARCHAR(255), `description` VARCHAR(255), `published` TINYINT(1), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;

Executing (default): SHOW INDEX FROM `tutorials`

Drop and re-sync db.

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

1. **Create a new Tutorial using POST /tutorials Api**



After creating some new Tutorials, you can check MySQL table:

mysql> select \* from tutorials;

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

| id | title | description | published | createdAt | updatedAt |

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

| 1 | JS: Node Tut #1 | Tut#1 Description | 0 | 2019-12-13 01:13:57 | 2019-12-13 01:13:57 |

| 2 | JS: Node Tut #2 | Tut#2 Description | 0 | 2019-12-13 01:16:08 | 2019-12-13 01:16:08 |

| 3 | JS: Vue Tut #3 | Tut#3 Description | 0 | 2019-12-13 01:16:24 | 2019-12-13 01:16:24 |

| 4 | Vue Tut #4 | Tut#4 Description | 0 | 2019-12-13 01:16:48 | 2019-12-13 01:16:48 |

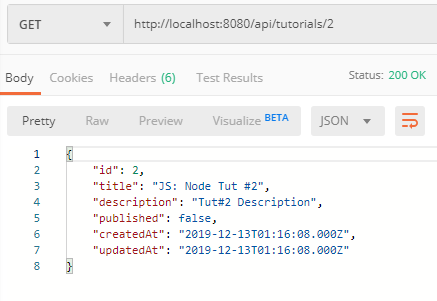
| 5 | Node & Vue Tut #5 | Tut#5 Description | 0 | 2019-12-13 01:16:58 | 2019-12-13 01:16:58 |

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

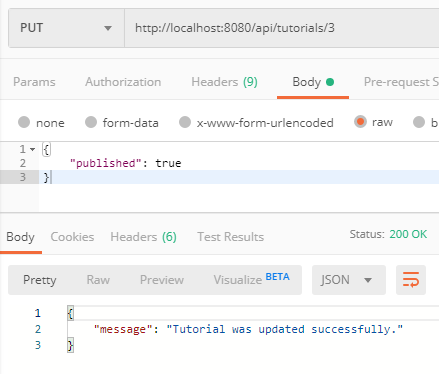
1. **Retrieve all Tutorials using GET /tutorials Api**



1. **Retrieve a single Tutorial by id using GET /tutorials/:id Api**



1. **Update a Tutorial using PUT /tutorials/:id Api**



Check tutorials table after some rows were updated:

mysql> select \* from tutorials;

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

| id | title | description | published | createdAt | updatedAt |

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

| 1 | JS: Node Tut #1 | Tut#1 Description | 0 | 2019-12-13 01:13:57 | 2019-12-13 01:13:57 |

| 2 | JS: Node Tut #2 | Tut#2 Description | 0 | 2019-12-13 01:16:08 | 2019-12-13 01:16:08 |

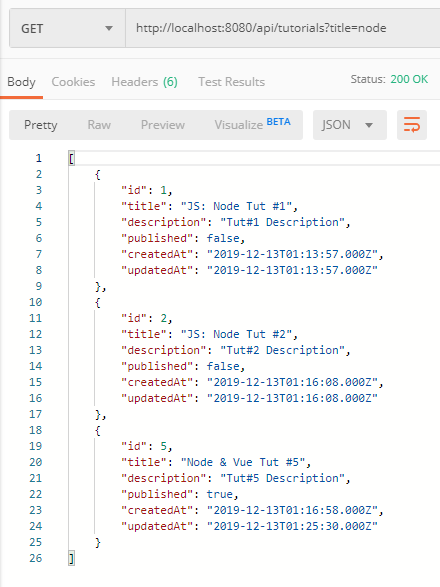
| 3 | JS: Vue Tut #3 | Tut#3 Description | 1 | 2019-12-13 01:16:24 | 2019-12-13 01:22:51 |

| 4 | Vue Tut #4 | Tut#4 Description | 1 | 2019-12-13 01:16:48 | 2019-12-13 01:25:28 |

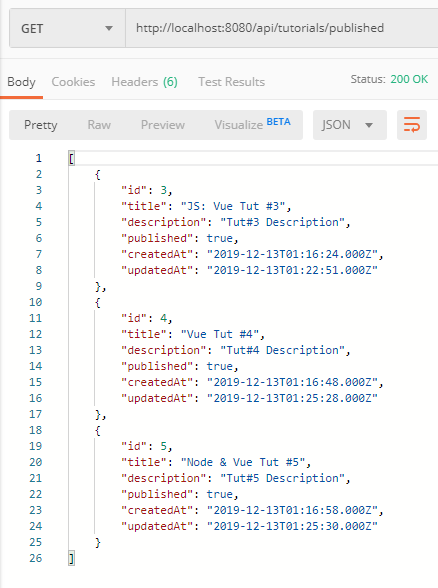
| 5 | Node & Vue Tut #5 | Tut#5 Description | 1 | 2019-12-13 01:16:58 | 2019-12-13 01:25:30 |

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

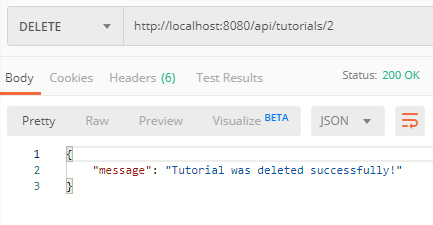
1. **Find all Tutorials which title contains ‘node’: GET /tutorials?title=node**



1. **Find all published Tutorials using GET /tutorials/published Api**



1. **Delete a Tutorial using DELETE /tutorials/:id Api**



Tutorial with id=2 was removed from tutorials table:

mysql> select \* from tutorials;

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

| id | title | description | published | createdAt | updatedAt |

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

| 1 | JS: Node Tut #1 | Tut#1 Description | 0 | 2019-12-13 01:13:57 | 2019-12-13 01:13:57 |

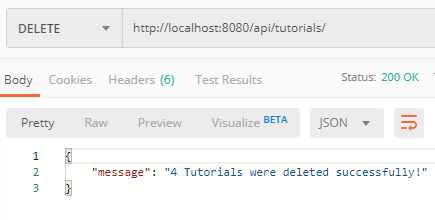
| 3 | JS: Vue Tut #3 | Tut#3 Description | 1 | 2019-12-13 01:16:24 | 2019-12-13 01:22:51 |

| 4 | Vue Tut #4 | Tut#4 Description | 1 | 2019-12-13 01:16:48 | 2019-12-13 01:25:28 |

| 5 | Node & Vue Tut #5 | Tut#5 Description | 1 | 2019-12-13 01:16:58 | 2019-12-13 01:25:30 |

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

1. **Delete all Tutorials using DELETE /tutorials Api**



Now there are no rows in tutorials table:

mysql> SELECT \* FROM tutorials;

Empty set (0.00 sec)

# You can use the. Axios to check it