**How to Make CRUD API in Node JS + MySQL?**

A CRUD (**Create, Read, Update, Delete**) API is a fundamental component of many web applications, allowing you to perform basic database operations. In this article, we’ll guide you through the process of building a CRUD API using Node.js and MySQL.

We’ll use the popular Express.js framework for creating the API and MySQL as our database. This step-by-step tutorial will help you create a fully functional API.

Before getting started, make sure you have the following prerequisites in place:

**Node.js and npm**: Ensure that you have Node.js installed on your machine. You can download it from the official Node.js website. npm, the Node Package Manager, typically comes with Node.js.

**MySQL**: You should have MySQL installed on your machine or access to a MySQL server.

**A Text Editor or IDE**: You’ll need a code editor or integrated development environment (IDE) of your choice to write and manage your Node.js application.

**Building a CRUD API with Node.js and MySQL**

**Step 1: Set Up a Node.js Project**

The first step is to create a new directory for your project and navigate to it in your terminal:

mkdir node-mysql-crud-api

cd node-mysql-crud-api

Next, initialize a new Node.js project:

npm init -y

This command creates a package.json file with default settings.

**Step 2: Install Dependencies**

You’ll need to install the necessary packages for building the API and connecting to MySQL. Open your terminal and run the following command:

npm i express express-validator mysql2

**Express:** is used for building the API.

**Mysql2:** is the MySQL driver for Node.js.

**Express-validator:** is used for data validation.

**Step 3: Enable ES6 Import**

In this project, we will use es6 import, so you have to add "type": "module" in the package.json.

**Step 4: Creation of MySQL Database and Table**

In this project we will perform CRUD operations by creating, reading, updating and deleting posts. Therefore we have to design the database and table accordingly.

First, create a database with the name you want, I named it **node\_crud\_api**.

After that, we need to create a table called posts, and there will be 6 columns inside the table –

**Id:** Post ID.

**Title:** Title of the post.

**Content:** Post content.

**Author:** Author name of the post

**Created\_at:** Post creation date and time.

**Updated\_at:** Date and time of Post update.

After creating the database, use the following SQL code to create the posts table and its structure.

CREATE TABLE `posts` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`title` varchar(100) NOT NULL,

`content` text NOT NULL,

`author` varchar(30) NOT NULL,

`created\_at` date NOT NULL DEFAULT current\_timestamp(),

`updated\_at` date NOT NULL DEFAULT current\_timestamp(),

PRIMARY KEY (`id`)

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

**Step 5: Application folder structure**

The following image gives you a complete look of the node-mysql-crud-api folder structure.

Node MySQL CRUD API project folder structure

So we need to create the files shown in the above image to create this Node JS CRUD API.

**Step 6: Set Up the MySQL Database Connection**

To interact with the MySQL database, you need to establish a connection. Therefore the database.js contains the code for the database connection.

Replace the placeholders in the following code with your database credentials:

database.js

import mysql from "mysql2";

const connection = mysql.createPool({

host: "localhost",

user: "root",

password: "",

database: "node\_crud\_api",

});

export default connection.promise();

**Step 7: Implementation of CRUD Endpoints**

routes.js – contains all the routes (CRUD endpoints) with validation rules.

Controller.js – The Controller class contains all the callbacks that handle all the requests and perform all **CRUD operations**.

**Step 8: Create the Express Application**

index.js: where the execution of the application will start.

index.js

import express from "express";

import routes from "./routes.js";

const app = express();

const PORT = 3000;

app.use(express.json());

app.use(routes);

// Error Handling

app.use((err, req, res, next) => {

err.statusCode = err.statusCode || 500;

err.message = err.message || "Internal Server Error";

res.status(err.statusCode).json({

message: err.message,

});

});

app.listen(PORT, () => console.log(`Server is running on port ${PORT}`));

**Step 9: Start the Application**

You can now start the application:

**Node index.js**

Your CRUD API is now running on **http://localhost:3000**. You can use tools like Postman or Thunder Client VS Code extension to interact with the API and perform CRUD operations on your MySQL database.

**Step 10: Testing of the Node JS CRUD API**

This guide provides a basic knowledge of building a CRUD API with Node.js and MySQL. Depending on your specific requirements, you may want to add more features, authentication, and error handling to enhance the functionality and robustness of your API.

**Node .JS CREATE**

const http=require('http');

http.createServer((req,res)=>{

const mysql=require('mysql');

const Connection=mysql.createConnection({

    host:"localhost",

    user:"root",

    password:"",

    database:"olivier"

});

Connection.connect(function(err){

    if(err){

        throw err;

    }

    Connection.query("create table user2(id int,name varchar(100))");

    res.end('table is created');

    console.log('table is created');

});

}).listen(2000);

**Node.JS READ**

const http = require('http');

const mysql = require('mysql');

// MySQL database configuration

const dbConfig = {

  host: 'localhost',

  user: 'root',

  password: '',

  database: 'login'

};

// Create a MySQL connection pool

const pool = mysql.createPool(dbConfig);

// Create an HTTP server

const server = http.createServer((req, res) => {

  // Set the response headers

  res.writeHead(200, { 'Content-Type': 'text/html' });

  // Check if the request URL is for data

  if (req.url === '/data') {

    // Use the connection pool to query the database

    pool.getConnection((err, connection) => {

      if (err) {

        console.error('Error getting MySQL connection:', err);

        res.writeHead(500, { 'Content-Type': 'text/plain' });

        res.end('Internal Server Error');

        return;

      }

      // Query the database and retrieve data

      connection.query('SELECT \* FROM user', (error, results) => {

        connection.release(); // Release the connection back to the pool

        if (error) {

          console.error('Error executing MySQL query:', error);

          res.writeHead(500, { 'Content-Type': 'text/plain' });

          res.end('Internal Server Error');

        } else {

          // Send the data as a JSON response

          res.end(JSON.stringify(results));

        }

      });

    });

  } else {

    // Send a basic HTML response

    res.end('<h1>Hello, this is a Node.js server!</h1>');

  }

});

// Set the server to listen on port 3000

const PORT = 2000;

server.listen(PORT, () => {

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

});

**Node.JS UPDATE**

const http = require('http');

let mysql = require('mysql');

// Create an HTTP server

const server = http.createServer((req, res) => {

  // Set the response headers

  res.writeHead(200, { 'Content-Type': 'text/html' });

let connection = mysql.createConnection({

  host: 'localhost',

  port: 2000,

  user: 'root',

  password: '',

  database: 'login',

});

connection.connect((err) => {

  if (err) return console.error(err.message);

  let sql = `UPDATE user

           SET user\_name = kalisa?

           WHERE id = 1`;

  let data = [false, 1];

  connection.query(sql, data, (error, results, fields) => {

    if (error) return console.error(error.message);

    console.log('Rows affected:', results.affectedRows);

  });

});

  // close the database connection

  connection.end();

}).listern(2000);

**Node.JS DELETE**

const http = require('http');

let mysql = require('mysql');

// Create an HTTP server

const server = http.createServer((req, res) => {

  // Set the response headers

  res.writeHead(200, { 'Content-Type': 'text/html' });

let connection = mysql.createConnection({

  host: 'localhost',

  port: 2000,

  user: 'root',

  password: '',

  database: 'login',

});

connection.connect((err) => {

  if (err) return console.error(err.message);

  let sql = `DELETE FROM user

           WHERE id = 1`;

  let data = [1];

  connection.query(sql, data, (error, results, fields) => {

    if (error) return console.error(error.message);

    console.log('Rows affected:', results.affectedRows);

  });

});

  // close the database connection

  connection.end();

}).listern(2000);