Node.js CRUD Application with PostgreSQL

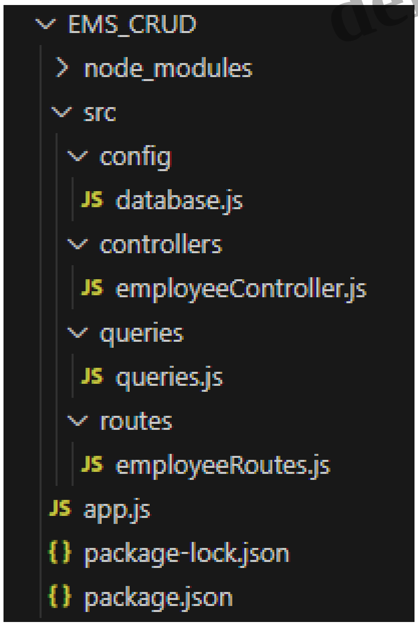
Description

Introduction and Example Overview

Node.js is an open source, cross-platform Javascript runtime that allows us to run Javascript code outside of the browser scope. Node.js is primarily used to write Javascript code for backend to perform activities like database interaction, file system access, network access etc. Using Express.js along with Node.js is like cherry on the cake because it makes the developers life easy to write backend code. In this post we are going to see a very basic example which is a good starting point to learn how to create Node.js CRUD Application with PostgreSQL. Make sure that you have Postgres database client installed on your machine.

Node.js CRUD Example Project Structure Overview

Below is the project structure we are going to follow to maintain modularity. Since it is a basic Node.js CRUD example we have not created the service layer and repository layer, hence we have written the database interaction logic in controller layer.



```
EMS_CRUD
  node_modules
  src
    config
      database.js
    controllers
      employeeController.js
    queries
      queries.js
    routes
      employeeRoutes.js
  app.js
  package-lock.json
  package.json
```

- **config -> database.js :** config folder contains all the files related to project configuration. Here in database.js we have kept database connectivity details.
- **routes -> emplyeeRoutes.js :** routes folder contains all the routes i.e. endpoints.
- **controller -> emplyeeController.js :** controller folder contains all the files which act as a handler when the reaches here from route layer.
- **queries -> queries.js :** queries folder contains all the queries if you are planning to create an

application with native SQL queries. If you are using Sequelize which is an ORM, then the queries folder might not be required.

- **app.js :** This is an entry point which will execute when we start our node application.
- **package.json :** package.json file contains all the dependencies and project metadata information.

Creating Node.js app using npm

If you are in your VS code terminal, make sure you are in the correct directory that you want to make as node.js application. In order to create a node.js app, run **npm init** command. It will ask you to enter the project name, description and other details. But if you want to keep all the options as default, then run **npm init -y** command. It will generate a package.json file for you.

```
PS C:\Users\GS-2865\Desktop\EMS_CRUD> npm init -y
Wrote to C:\Users\GS-2865\Desktop\EMS_CRUD\package.json:

{
  "name": "EMS_CRUD",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

We need to install a bunch of dependencies to make our node.js applicable capable for database interaction. Hit below command on your VS code terminal.

```
npm install express pg postgres --save
```

Below is how the package.json file looks like after installing the dependencies.

```
{
  "name": "ems_crud",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "4.18.2",
    "pg": "^8.11.3",
    "postgres": "3.3.5",
    "uuid": "9.0.1"
  }
}
```

Setting up Express Web Server

The file naming convention used for setting up express web server is either index.js or server.js or app.js. We have imported express dependency and employeeRoute internal module as we are going to call route layer from this file. We have also setup a base route which the service consumer has to add while invoking.

```
const express = require('express')
const employeeRoute = require('./src/routes/employeeRoutes')

const app = express();

//Makes sure the returned object is in json format
app.use(express.json())

//base route
app.use('/api/v1/employees', employeeRoute);

//Sets the port number and startup message when the server starts
const port = 8080;
app.listen(port, () => console.log(`Listening to port ${port}`))
```

PostgreSQL database connection details in database.js config file

Connection details to Postgres is present in database.js config file as shown below. It contains the username, password, port and host details. We will export the details so that it can be used in the controller layer. Usually it is used in repository layer.

```
const Pool = require("pg").Pool;

const pool = new Pool({
    user: "postgres",
    host: "localhost",
    database: "employee",
    password: "password",
    port: 5432
});
```

```
module.exports = pool;
```

Creating Route containing the endpoints.

When client hits the request to an endpoint using Http methods such as GET, POST, PUT, DELETE, we need to specific how the server will respond. For this we need to configure the routes. Below is how the employeeRoute.js looks like

```
const Router = require('express')
const employeeController = require('../controllers/employeeController.js')
const router = Router();

router.get('/', employeeController.getEmployees);
router.get('/:id', employeeController.getByEmployeeId);
router.post('/', employeeController.createEmployee)
router.delete('/:id', employeeController.deleteEmployee);

module.exports = router;
```

We have given just '/' for all the http methods. We extracted out the common part of the url and placed it in app.js i.e. *app.use('/api/v1/employees', employeeRoute);*

We imported employeeController.js because we are going to invoke the methods present in the controller.

Creating the Controller

For every operation we will use a common query method skeleton.

```
pool.query(argument1, [argument2], (error,results) => {
        if(error) throw error;
        resp.status(201).send("Employee created successfully");
    });
```

The query method accepts three arguments. The first being the actual SQL native query, second argument is an array of each property that needs to be mapped in the placeholder present in the SQL query, and third argument is the callback function which gets the result if the database call is successful or throws the error if any exception occurs.

Create Employee

```
const createEmployee = (req, resp) => {
    const {id, first_name, last_name, designation, phone_number} = req.body;
    pool.query(queries.createEmployee, [id, first_name, last_name, designation
        if(error) throw error;
        resp.status(201).send("Employee created successfully");
    });
};
```

To create an Employee record, first we will extract our each property from the request object and then pass it to query method. Upon success we return 201 CREATED Response code and a message along with it stating “*Employee created successfully*“

Delete Employee

```
const deleteEmployee = (req, resp) => {
    const id = parseInt(req.params.id);
    pool.query(queries.deleteEmployee, [id], (error,results) => {
        if(error) throw error;
        resp.status(200).send("Employee deleted successfully");
    });
};
```

To delete an Employee first we need to get the path variable i.e. employeeId. We get it using ***req.params.id***. Upon successful deletion we return 200 response code and a message stating that *Employee deleted successfully*.

Get All employees

```
const getEmployees = (req, resp) => {

    pool.query(queries.getEmployees, (error,results) => {
        if(error) throw error;
        resp.status(200).send(results.rows);
        console.log("All Employees returned successfully");
    });
};
```

This query method only accepts two arguments as compared to the previous one’s since we don’t have to extract anything from the request and the SQL query also does not have any condition to retrieve the records. Upon successful execution we return 200 response code along with all the records.

Get single Employee

```
const getByEmployeeId = (req, resp) => {
    const id = parseInt(req.params.id);
    pool.query(queries.getEmployeeById, [id], (error,results) => {
        if(error) throw error;
        resp.status(200).send(results.rows);
        console.log(`Employee with ${id} returned successfully`);
    });
}
```

To fetch an Employee first we need to get the path variable i.e. *employeeId*. We get it using ***req.params.id*** and convert it into Integer datatype. Upon successful fetching of employee record we return 200 response code and the fetched employee record.

Please find the complete ***employeeController.js*** class.

```
const pool = require('../config/database');
const queries = require('../queries/queries')

//create Employee object
const createEmployee = (req, resp) => {

    const {id, first_name, last_name, designation, phone_number} = req.body;
    pool.query(queries.createEmployee, [id, first_name, last_name, designation
        if(error) throw error;
        resp.status(201).send("Employee created successfully");
    });
};

//Delete Employee object
const deleteEmployee = (req, resp) => {
    const id = parseInt(req.params.id);
    pool.query(queries.deleteEmployee, [id], (error,results) => {
        if(error) throw error;
        resp.status(200).send("Employee deleted successfully");
    });
};

//Retrieve all Employees
const getEmployees = (req, resp) => {

    pool.query(queries.getEmployees, (error,results) => {
        if(error) throw error;
        resp.status(200).send(results.rows);
        console.log("All Employees returned successfully");
    });
};

//Retrieve Employee object based on the id
const getByEmployeeId = (req, resp) => {
    const id = parseInt(req.params.id);
    pool.query(queries.getEmployeeById, [id], (error,results) => {
        if(error) throw error;
        resp.status(200).send(results.rows);
        console.log(`Employee with ${id} returned successfully`);
    });
}

//Exported methods to be used on route layer.
module.exports = {
    createEmployee,
    deleteEmployee,
    getByEmployeeId,
    getEmployees
}
```

Creating the Employee table

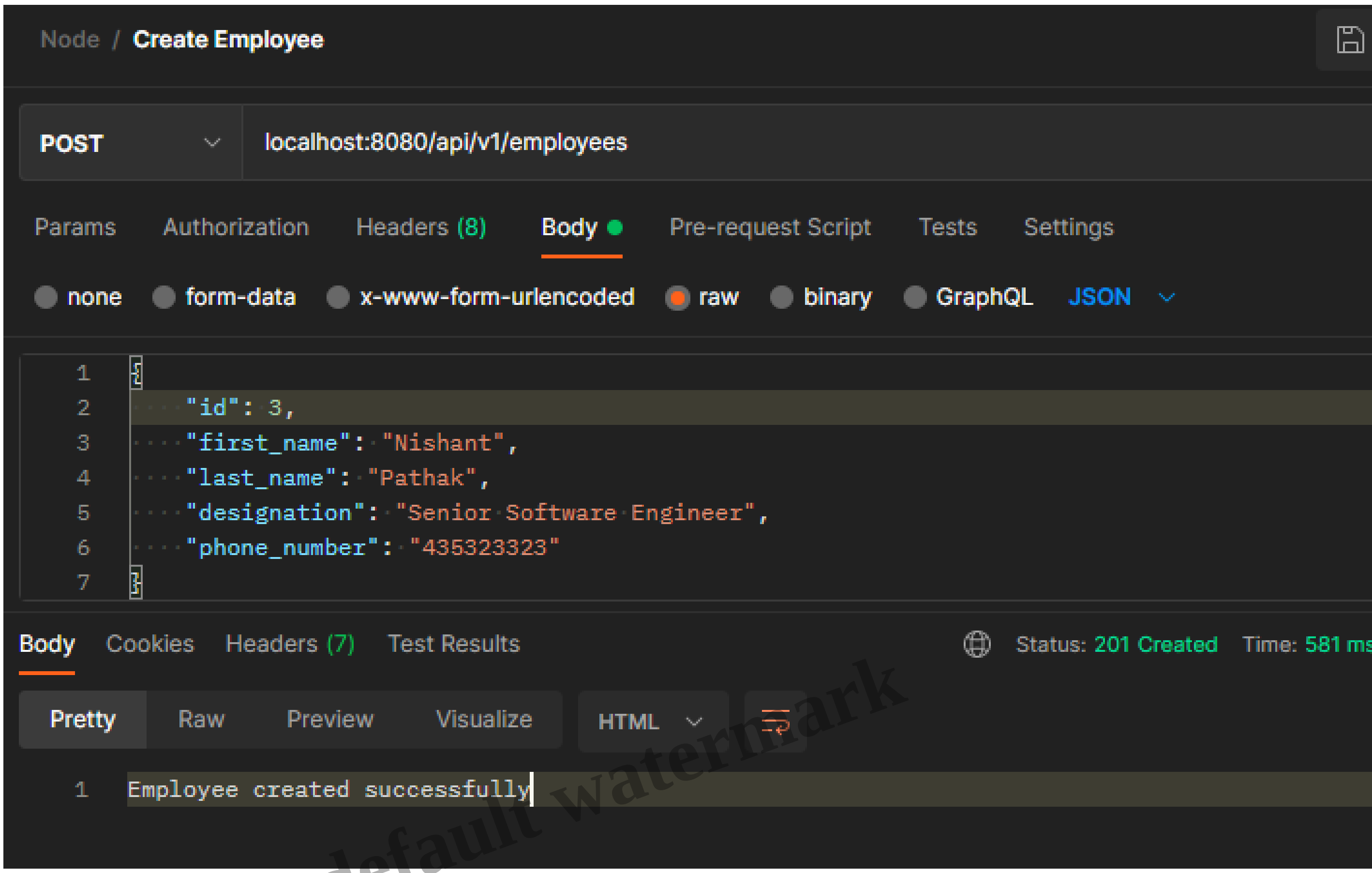
Before executing your node application, copy paste the below script in Postgres query editor which will create employee database table.

```
CREATE TABLE public.employee
(
    id integer NOT NULL,
    first_name character varying COLLATE pg_catalog."default",
    last_name character varying COLLATE pg_catalog."default",
    designation character varying COLLATE pg_catalog."default",
    phone_number character varying COLLATE pg_catalog."default",
    CONSTRAINT employee_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.employee
    OWNER to postgres;
```

Testing our Node.js REST APIs

Finally its time to test our changes. We have used postman client application to test the changes, you can use any client application of your choice. For each REST operation we have placed the screenshot that contains the request and its corresponding result.

Creating Employee object



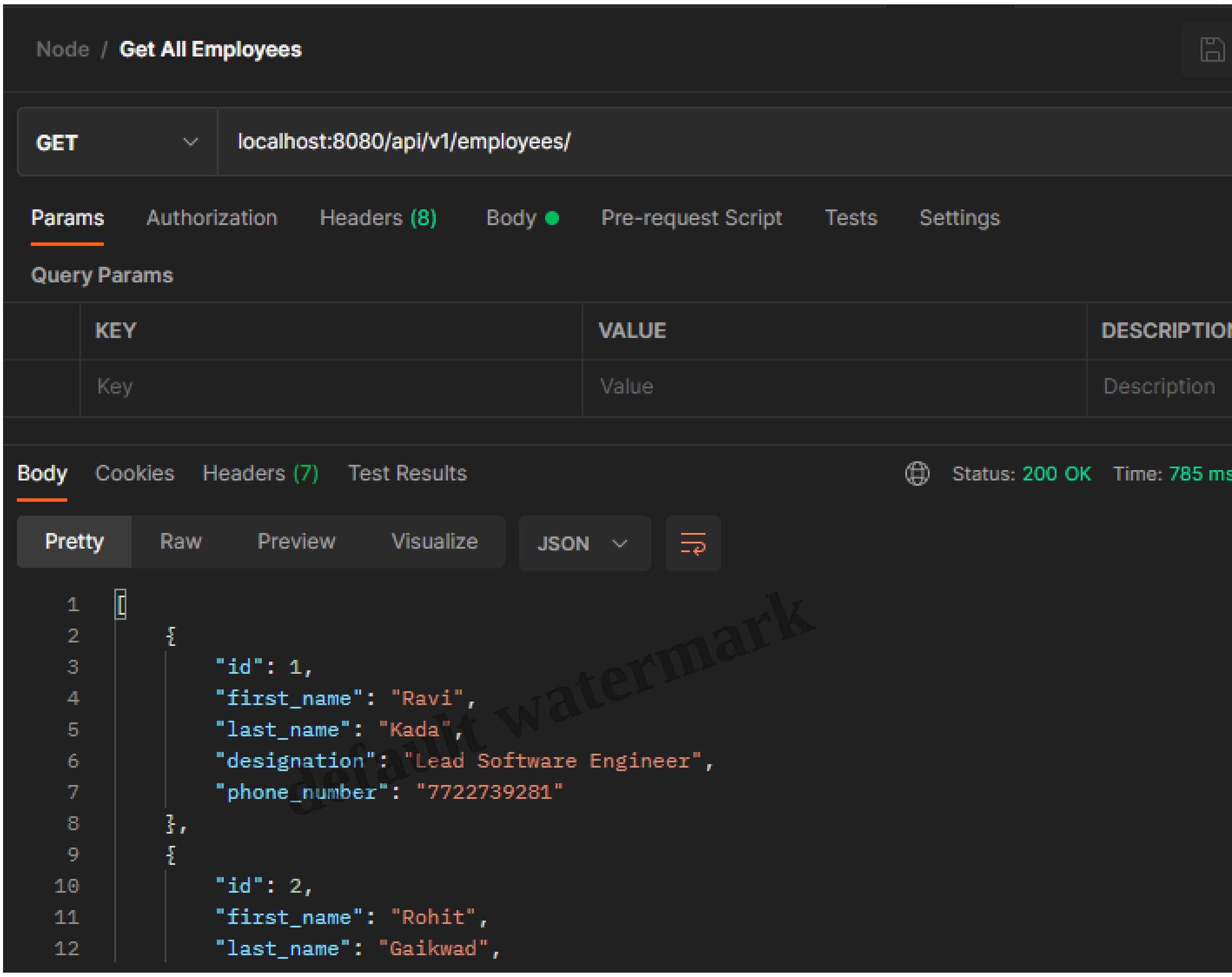
POST localhost:8080/api/v1/employees

```
{
    "id": 3,
    "first_name": "Nishant",
    "last_name": "Pathak",
    "designation": "Senior Software Engineer",
    "phone_number": "435323323"
}
```

Status: 201 Created Time: 581 ms

```
Employee created successfully
```

Retrieving all Employee objects



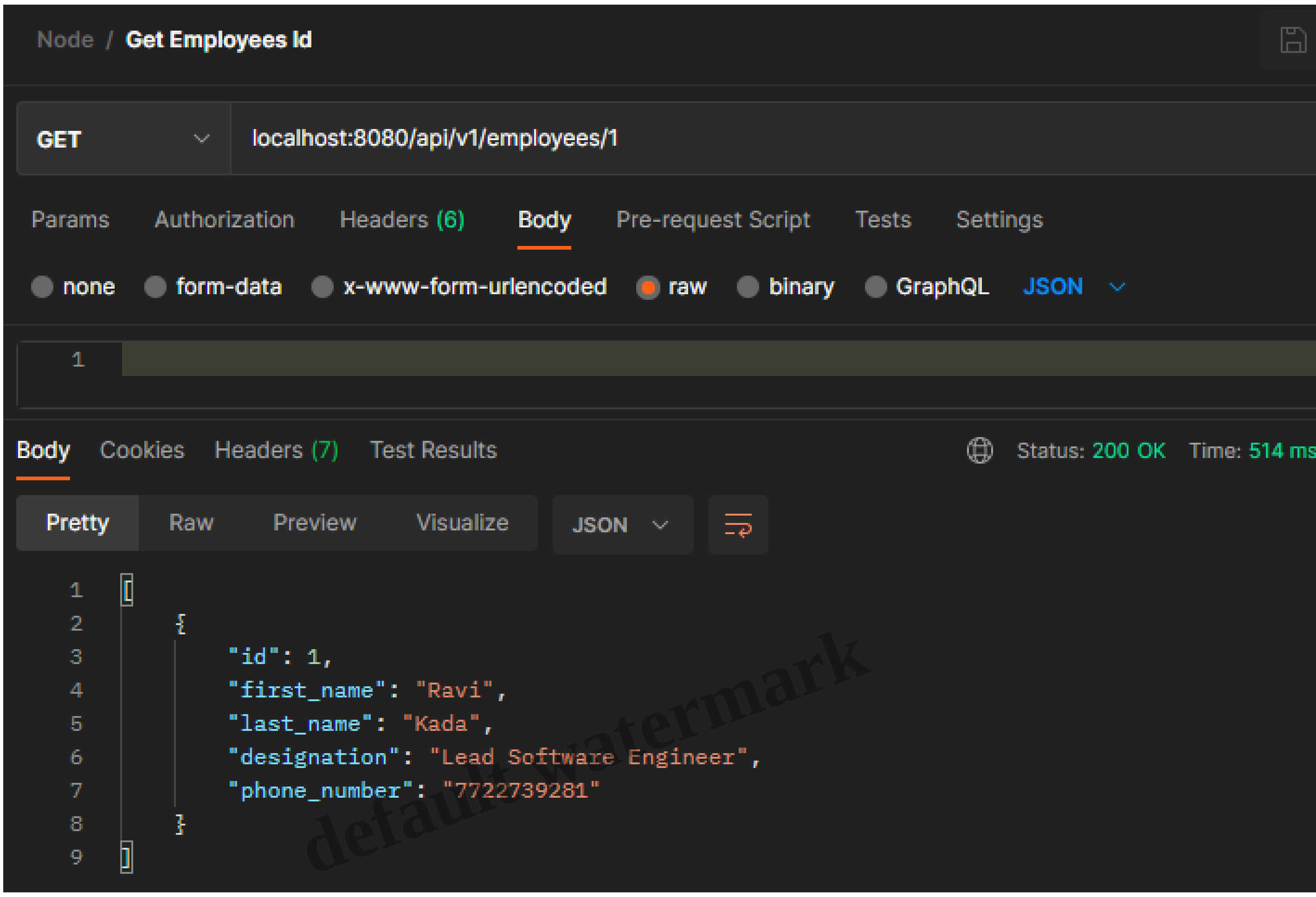
GET localhost:8080/api/v1/employees/

KEY	VALUE	DESCRIPTIO
Key	Value	Description

Status: 200 OK Time: 785 ms

```
[
    {
        "id": 1,
        "first_name": "Ravi",
        "last_name": "Kada",
        "designation": "Lead Software Engineer",
        "phone_number": "7722739281"
    },
    {
        "id": 2,
        "first_name": "Rohit",
        "last_name": "Gaikwad",
```

Retrieving single Employee object


Node / Get Employees Id

GET localhost:8080/api/v1/employees/1

Params Authorization Headers (6) Body Pre-request Script Tests Settings

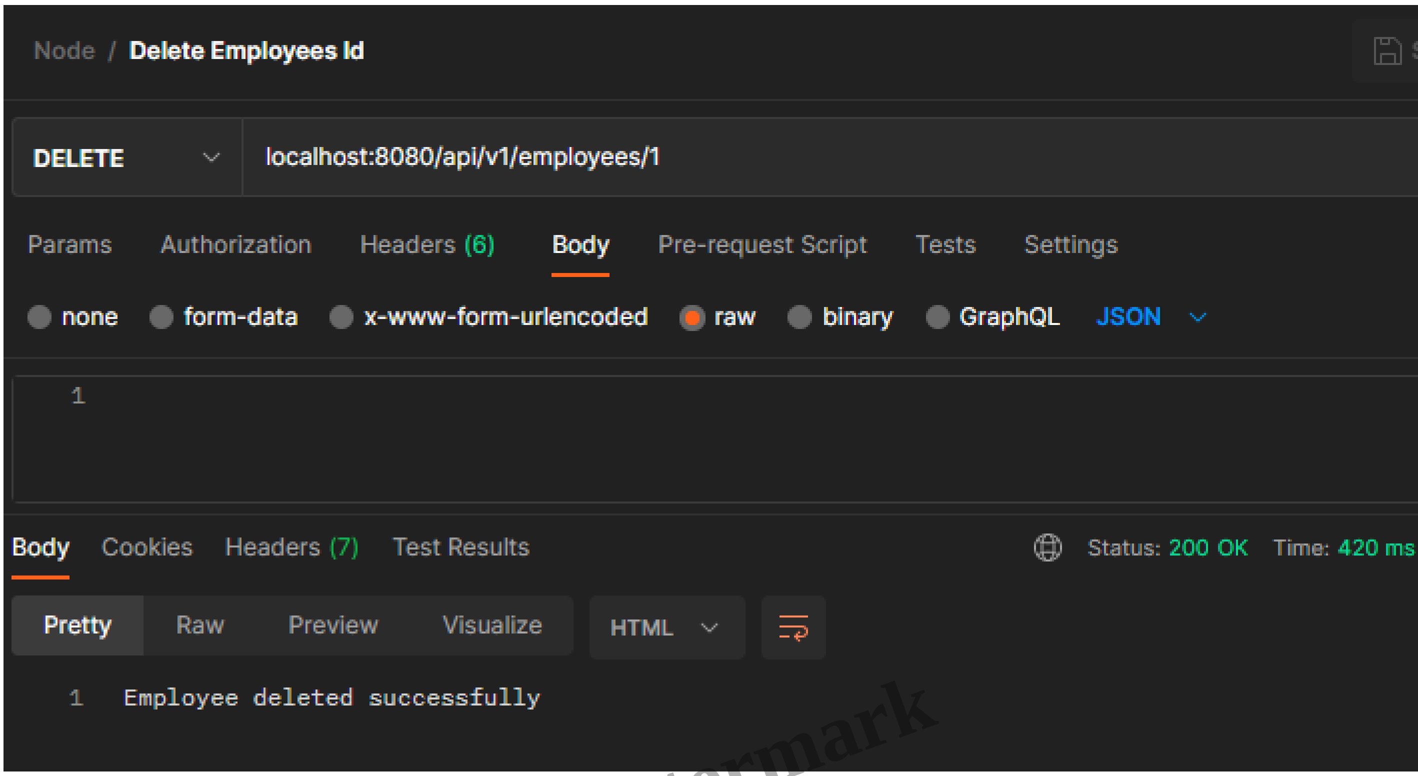
none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (7) Test Results Status: 200 OK Time: 514 ms

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "first_name": "Ravi",
        "last_name": "Kada",
        "designation": "Lead Software Engineer",
        "phone_number": "7722739281"
    }
]
```

Deleting Employee object using



DELETE localhost:8080/api/v1/employees/1

Status: 200 OK Time: 420 ms

```
Employee deleted successfully
```

Conclusion

In this post we learnt how to create a Node.js + Express.js CRUD application which interacts with the Postgres database to perform database operation. If you liked this post, feel free to share it in the community. If you are preparing for node.js interview then you can take a look at node.js interview post
.

Category

1. Nodejs

Date Created
September 23, 2023
Author
kk-ravi144gmail-com