Create APIs in NodeJS

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:

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

Prerequisites and required applications

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. You should have basic understanding of nodejs.

ExpressJS is one of the most trending web frameworks for node.js. It is built on top of node.js http module, and adds support for routing, middleware, view system etc. It is very simple and minimal, unlike other frameworks.

MySQL is an open-source relational database management system. Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

EcmaScript (ES) is a standardised scripting language for JavaScript (JS). The current ES version supported in modern browsers is ES5. However, ES6 tackles a lot of the limitations of the core language, making it easier for devs to code

Postman is an API(application programming interface) development tool which helps to build, test and modify APIs.It has the ability to make various types of HTTP requests(GET, POST, PUT, PATCH etc.).

IDE (integrated development environment) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools, and a debugger. In case of mine, I prefer to use visual studio code.

Our project structure will be like:

Create a Project

Now it’s time to create our project. Create a directory name *NodeMysqlApp*. Then navigate to *NodeMysqlApp* directory. Command are as below

```
// Create directory
mkdir NodeMysqlApp

// then Navigate to NodeMysqlApp
cd NodeMysqlApp
```

Initialise and Configure Our Project

To initialise, run the command in the project folder *npm init* that will ask a few questions to avoid that you can run *npm init -y* .

```
PROBLEMS    TERMINAL    OUTPUT    DEBUG CONSOLE

PS E:\TestProject\WebRTC\Node JS\Projects\NodeMysqlApp> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (nodemysqlapp) nodemysqlapp
version: (1.0.0) 1.0.0
description: This is app for creating rest api using node
entry point: (index.js) server.js
test command: echo \"Error: no test specified\" && exit 1
git repository:
keywords: nodejs, expressjs, api, restfulapi, javascript, es6
author: Shashikala Patel
license: (ISC) ISC
About to write to E:\TestProject\WebRTC\Node JS\Projects\NodeMysqlApp\package.json:

{
  "name": "nodemysqlapp",
  "version": "1.0.0",
  "description": "This is app for creating rest api using node",
  "main": "server.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [
    "nodejs",
    "expressjs",
    "api",
    "restfulapi",
    "javascript",
    "es6"
  ],
  "author": "Shashikala Patel",
  "license": "ISC"
}


Is this OK? (yes) ▌
```

Finally *package.json* looks like below

```
{
  "name": "nodemysqlapp",
  "version": "1.0.0",
  "description": "This is app for creating rest api using node",
  "main": "server.js",
```

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "nodejs",
    "expressjs",
    "api",
    "restfulapi",
    "javascript",
    "es6"
  ],
  "author": "Shashikala Patel",
  "license": "ISC"
}
```

Install express and other dependencies

1. **Express** is top framework of node js. Install using below command :

   `npm install express --save`

2. **Body Parser** is Node.js body parsing middleware. Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

   `npm install body-parser --save`

3. **MySQL** is open source database use to interacting with database and manipulating the records.

   `npm install mysql --save`

4. **Nodemon** is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected. Use -dev flag to save in devDependencies and --save will save the dependencies in package.json file.

   `npm install --save-dev nodemon`

Now `package.json` looks like as below

package.json ×

package.json > [ ] keywords

```
1   {
2     "name": "nodemysqlapp",
3     "version": "1.0.0",
4     "description": "This is app for creating rest api using node",
5     "main": "server.js",
      ▷ Debug
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1"
8     },
9     "keywords": [
10      "nodejs",
11      "expressjs",
12      "api",
13      "restfulapi",
14      "javascript",
15      "es6"
16    ],
17    "author": "Shashikala Patel",
18    "license": "ISC",
19    "dependencies": {
20      "body-parser": "^1.19.0",
21      "express": "^4.17.1",
22      "mysql": "^2.18.1"
23    },
24    "devDependencies": {
25      "nodemon": "^2.0.7"
26    }
27  }
28
```

Start the web server

As we earlier we have created enter point of application is server.js, we will create *server.js* file at the root of project folder.

```
touch server.js
```

Add some code in `server.js` file

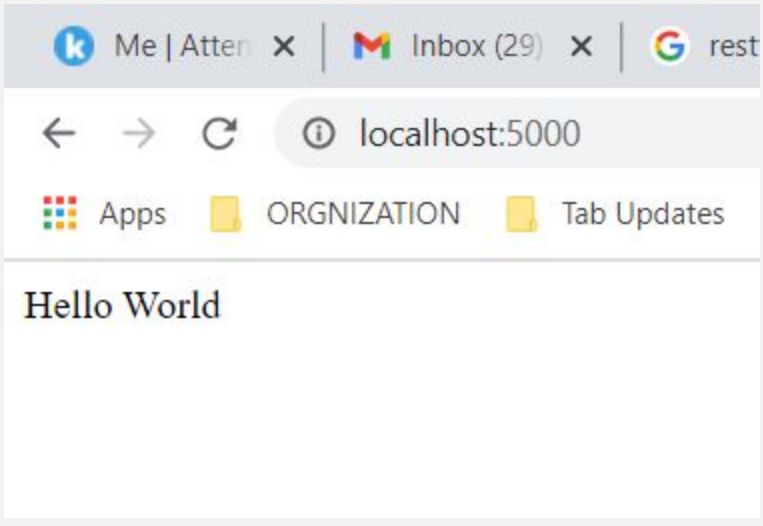
```
const express = require('express');
const bodyParser = require('body-parser');
// create express app
const app = express();
// Setup server port
const port = process.env.PORT || 5000;
// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }))
// parse requests of content-type - application/json
app.use(bodyParser.json())
// define a root route
app.get('/', (req, res) => {
  res.send("Hello World");
});
// listen for requests
app.listen(port, () => {
  console.log(`Server is listening on port ${port}`);
});
```

Now run the web server using `node server.js` or `node server` command :

`node server.js OR node server`

```
PS E:\TestProject\WebRTC\Node JS\Projects\NodeMysqlApp> node server
Server is listening on port 5000
```

Now open your favourite browser and navigate to http://localhost:5000 . Browser will show `Hello World` . That's great now our server is running.



In previous step we had installed `nodemon` . If we want run the server using nodemon then we have to use the `nodemon server.js` or `nodemon server` command. Let's do some change in `package.json` file , add a line of code in `scripts` object of package.json file.

```
"start": "nodemon server"
```

package.json × server.js

package.json > { } scripts > start

```
1   {
2     "name": "nodemysqlapp",
3     "version": "1.0.0",
4     "description": "This is app for creating rest api using node",
5     "main": "server.js",
      ▷ Debug
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1",
8       "start": "nodemon server"
9     },
10    "keywords": [
11      "nodejs",
12      "expressjs",
13      "api",
14      "restfulapi",
15      "javascript",
16      "es6"
17    ],
18    "author": "Shashikala Patel",
19    "license": "ISC",
20    "dependencies": {
21      "body-parser": "^1.19.0",
22      "express": "^4.17.1",
23      "mysql": "^2.18.1"
24    },
25    "devDependencies": {
26      "nodemon": "^2.0.7"
27    }
28  }
29
```

Now simply run `npm start` to run the server that will auto restart the serve when detect any change in files.

```
npm start
```

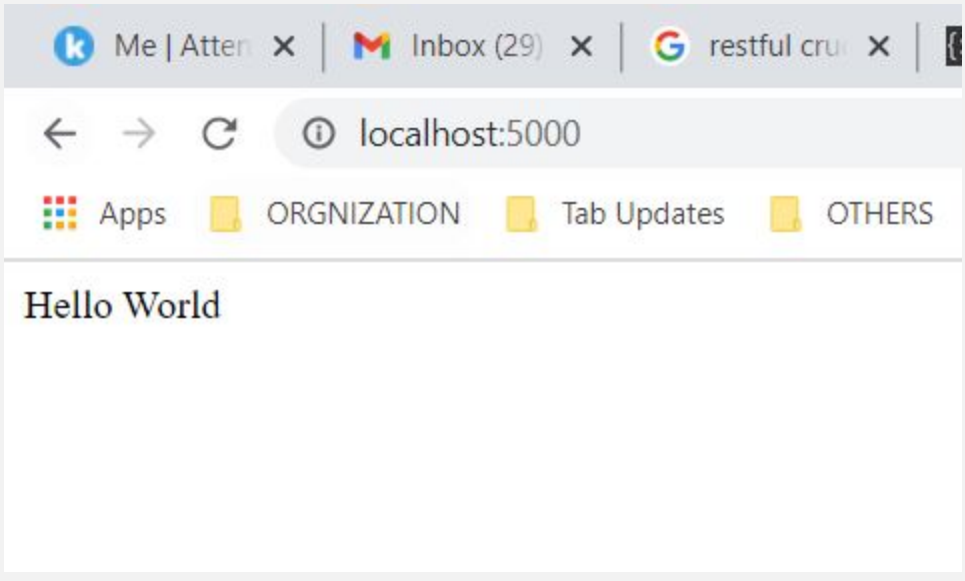
package.json server.js

server.js > app.get('/') callback

```javascript
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = process.env.PORT || 5000;

// parse requests of content-type
app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())

// define a root route
app.get('/', (req, res) => {
  res.send("Hello World");
});

// listen for requests
app.listen(port, () => {
  console.log(`Server is listening on port ${port}`);
});
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE

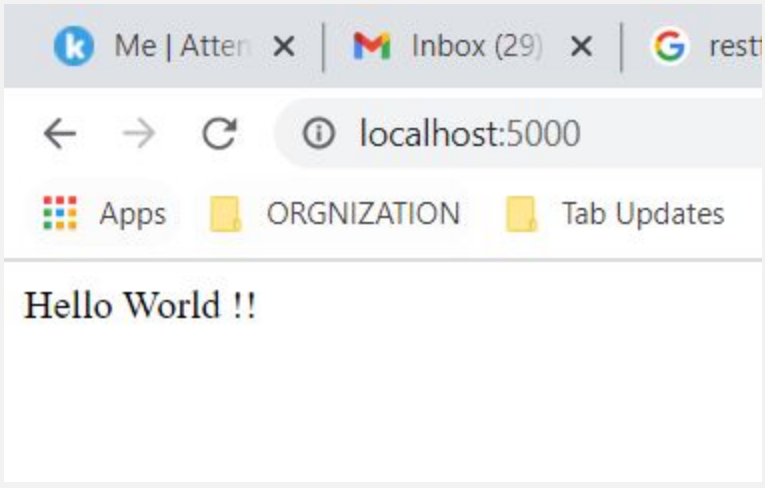
```
Server is listening on port 5000
PS E:\TestProject\WebRTC\Node JS\Projects\NodeMysqlApp> npm start

> nodemysqlapp@1.0.0 start E:\TestProject\WebRTC\Node JS\Projects\NodeMysqlApp
> nodemon server

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server is listening on port 5000
```


Me | Atten
Inbox (29)
restful cru
localhost:5000
Apps
ORGNIZATION
Tab Updates
OTHERS
Hello World

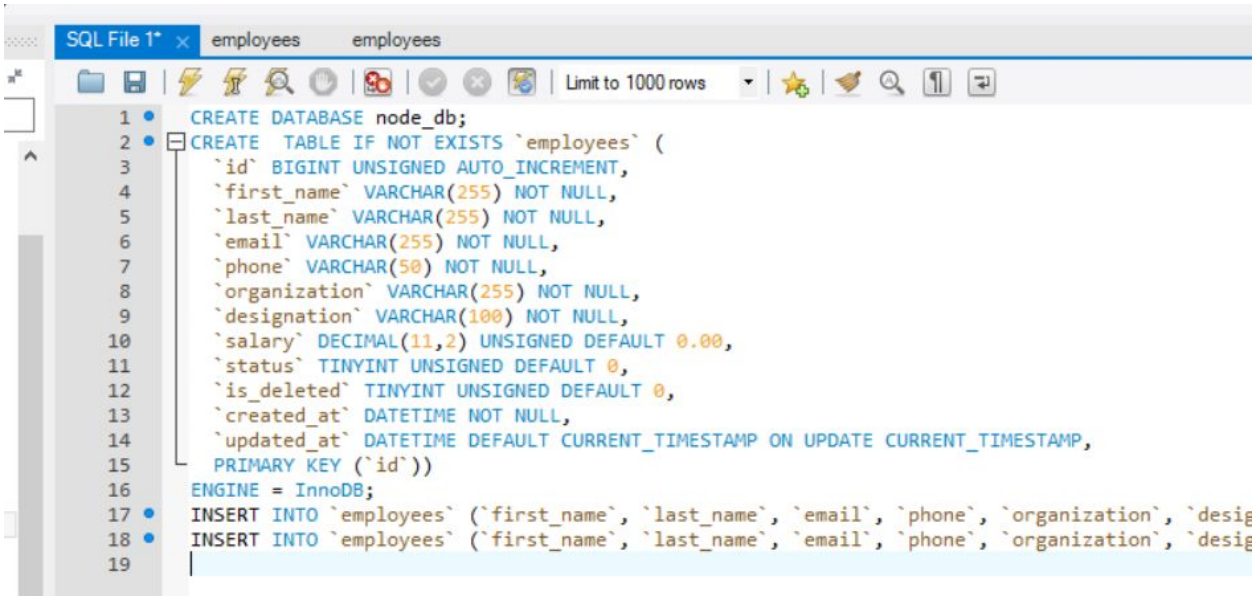
```
9
10  // define a root route
11  app.get('/', (req, res) => {
12    res.send("Hello World !!");
13  });
14
15  // listen for requests
16  app.listen(port, () => {
17    console.log(`Server is listening on port ${port}`);
18  });
```

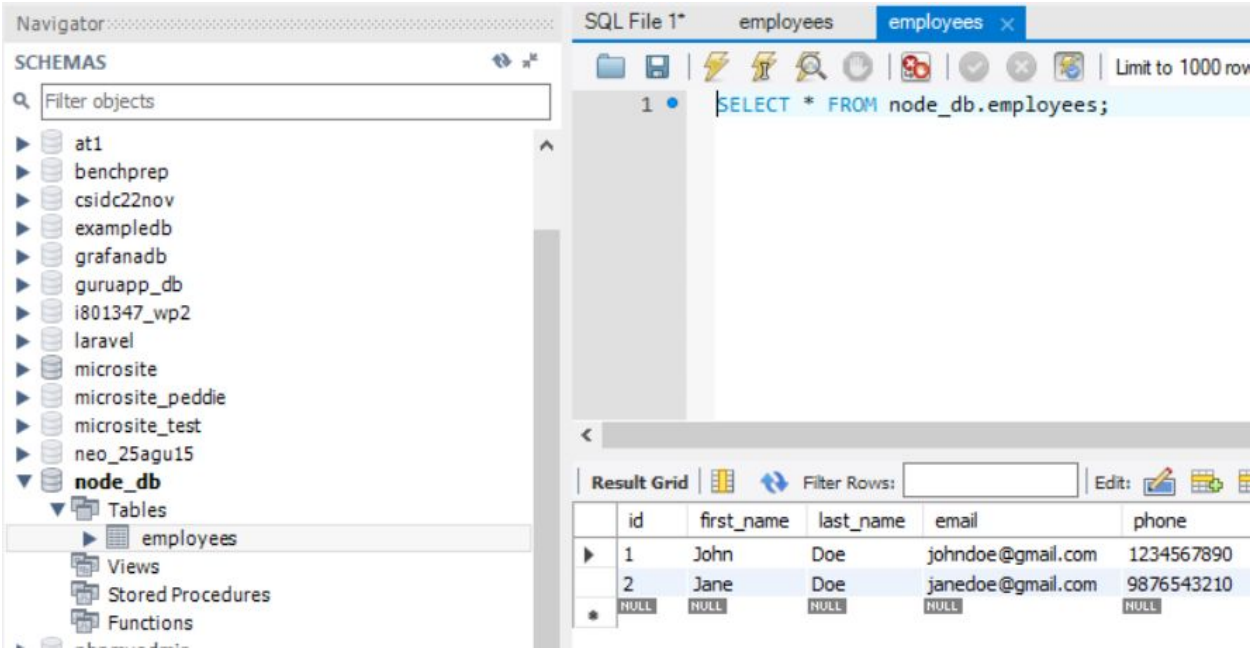
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE

```
Server is listening on port 5000
PS E:\TestProject\WebRTC\Node JS\Projects\NodeMysqlApp> npm start

> nodemysqlapp@1.0.0 start E:\TestProject\WebRTC\Node JS\Projects\NodeMys
> nodemon server

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server is listening on port 5000
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Server is listening on port 5000
```



Create database






```
CREATE DATABASE node_db;
CREATE  TABLE IF NOT EXISTS `employees` (
  `id` BIGINT UNSIGNED AUTO_INCREMENT,
  `first_name` VARCHAR(255) NOT NULL,
  `last_name` VARCHAR(255) NOT NULL,
  `email` VARCHAR(255) NOT NULL,
  `phone` VARCHAR(50) NOT NULL,
  `organization` VARCHAR(255) NOT NULL,
  `designation` VARCHAR(100) NOT NULL,
  `salary` DECIMAL(11,2) UNSIGNED DEFAULT 0.00,
  `status` TINYINT UNSIGNED DEFAULT 0,
  `is_deleted` TINYINT UNSIGNED DEFAULT 0,
  `created_at` DATETIME NOT NULL,
  `updated_at` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
```

```
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
INSERT INTO `employees` (`first_name`, `last_name`, `email`, `phone`, `organization`, `designation`, `salary`, `status`, `is_deleted`, `created_at`) VALUES ('John', 'Doe', 'johndoe@gmail.com', '1234567890', 'BR Softech Pvt Ltd', 'Full Stack Developer', '500.00', '1', '0', '2019-11-19 03:30:30');
INSERT INTO `employees` (`first_name`, `last_name`, `email`, `phone`, `organization`, `designation`, `salary`, `status`, `is_deleted`, `created_at`) VALUES ('Jane', 'Doe', 'janedoe@gmail.com', '9876543210', 'RG Infotech Jaipur', 'PHP Developer', '450.00', '1', '0', '2019-11-19 03:35:30');
```

Make database connection

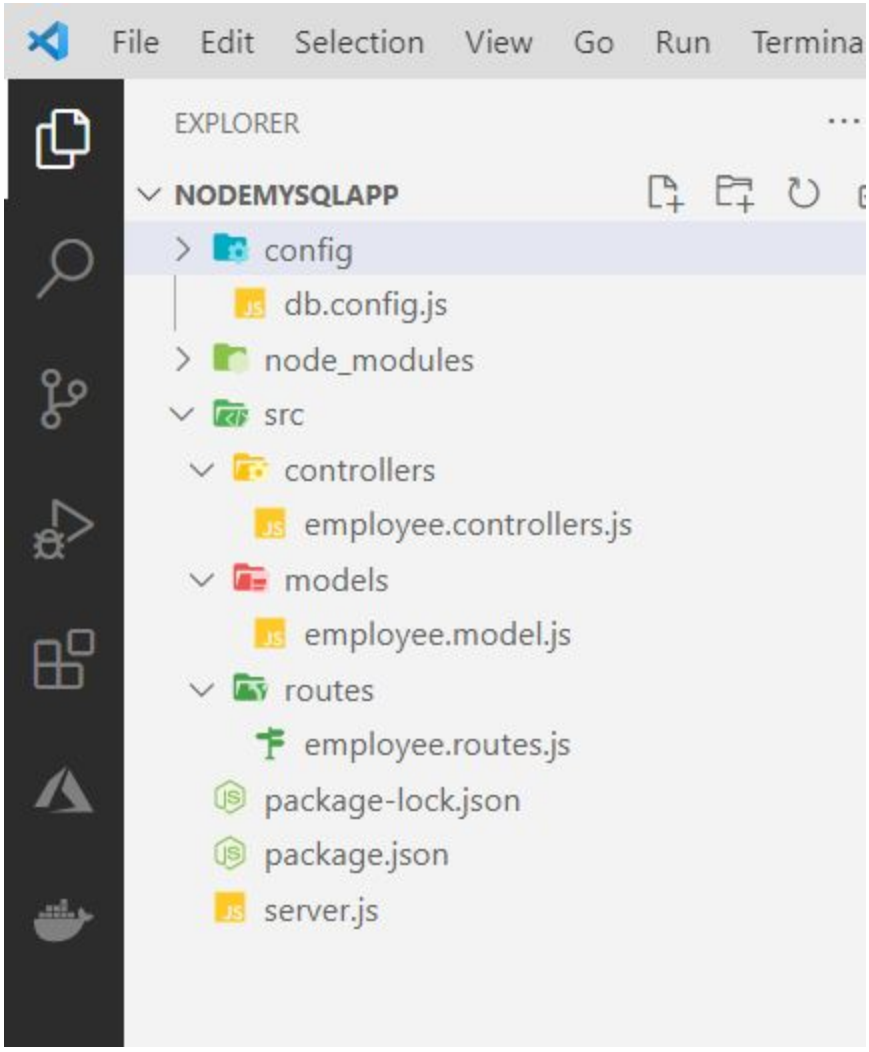
To make connectivity with the database in our project we'll make seperate file. So create a *config* folder at root and make a *db.config.js* file inside the *config* folder.

```
mkdir config
cd config
touch db.config.js
```

Now open `db.config.js` and add code below for creating mysql connection.

```
'use strict';
const mysql = require('mysql');
//local mysql db connection
const dbConn = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : '',
  database : 'node_db'
});
dbConn.connect(function(err) {
  if (err) throw err;
  console.log("Database Connected!");
});
module.exports = dbConn;
```

Project Folder Structure
Now folder structure of project like as below



Complete `employee.model.js` file is here -

package.json | server.js | db.config.js | employee.controllers.js | employee.routes.js | employee.model.js

src > models > employee.model.js > Employee

```
"use strict";
var dbConn = require("./../../config/db.config");

//Employee object create
var Employee = function (employee) {
  this.first_name = employee.first_name;
  this.last_name = employee.last_name;
  this.email = employee.email;
  this.phone = employee.phone;
  this.organization = employee.organization;
  this.designation = employee.designation;
  this.salary = employee.salary;
  this.status = employee.status ? employee.status : 1;
  this.created_at = new Date();
  this.updated_at = new Date();
};

Employee.create = function (newEmp, result) {
  dbConn.query("INSERT INTO employees set ?", newEmp, function (err, res) {
    if (err) {
      console.log("error: ", err);
      result(err, null);
    } else {
      console.log(res.insertId);
      result(null, res.insertId);
    }
  });
};

Employee.findById = function (id, result) {
  dbConn.query(
    "Select * from employees where id = ? ",
    id,
```

Here is complete `employee.controller.js` file -

package.json | server.js | db.config.js | employee.controllers.js | employee.routes

src > controllers > employee.controllers.js > findAll > findAll

```javascript
"use strict";
const Employee = require("../models/employee.model");

exports.findAll = function (req, res) {
  Employee.findAll(function (err, employee) {
    console.log("controller");
    if (err) res.send(err);
    console.log("res", employee);
    res.send(employee);
  });
};

exports.create = function (req, res) {
  const new_employee = new Employee(req.body);
  //handles null error
  if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
    res
      .status(400)
      .send({ error: true, message: "Please provide all required field" });
  } else {
    Employee.create(new_employee, function (err, employee) {
      if (err) res.send(err);
      res.json({
        error: false,
        message: "Employee added successfully!",
        data: employee,
      });
    });
  }
};

exports.findById = function (req, res) {
  Employee.findById(req.params.id, function (err, employee) {
    if (err) res.send(err);
```

Here is complete `employee.routes.js` file -

package.json | server.js | db.config.js | employee.controllers.js | employee.routes.js

src > routes > employee.routes.js > ...

```
const router = express.Router();
const employeeController = require("../controllers/employee.controller");

// Retrieve all employees
router.get("/", employeeController.findAll);

// Create a new employee
router.post("/", employeeController.create);

// Retrieve a single employee with id
router.get("/:id", employeeController.findById);

// Update a employee with id
router.put("/:id", employeeController.update);

// Delete a employee with id
router.delete("/:id", employeeController.delete);

module.exports = router;
```

Now finally complete `server.js` file here :

package.json | server.js | db.config.js | employee.controllers.js

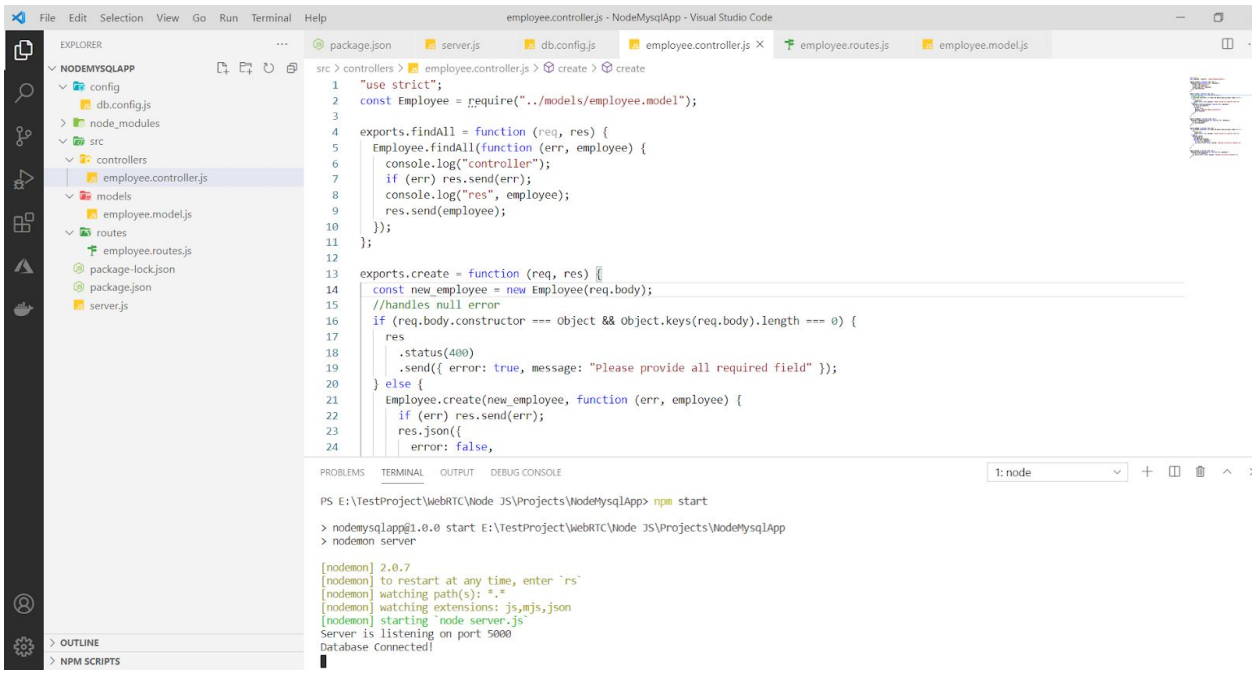
server.js > app.get('/') callback

```javascript
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = process.env.PORT || 5000;
const employeeRoutes = require('./src/routes/employee.routes')

// parse requests of content-type
app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())

// define a root route
app.get('/', (req, res) => {
  res.send("Hello World");
});

// using as middleware
app.use('/api/v1/employees', employeeRoutes)

// listen for requests
app.listen(port, () => {
  console.log(`Server is listening on port ${port}`);
});
```

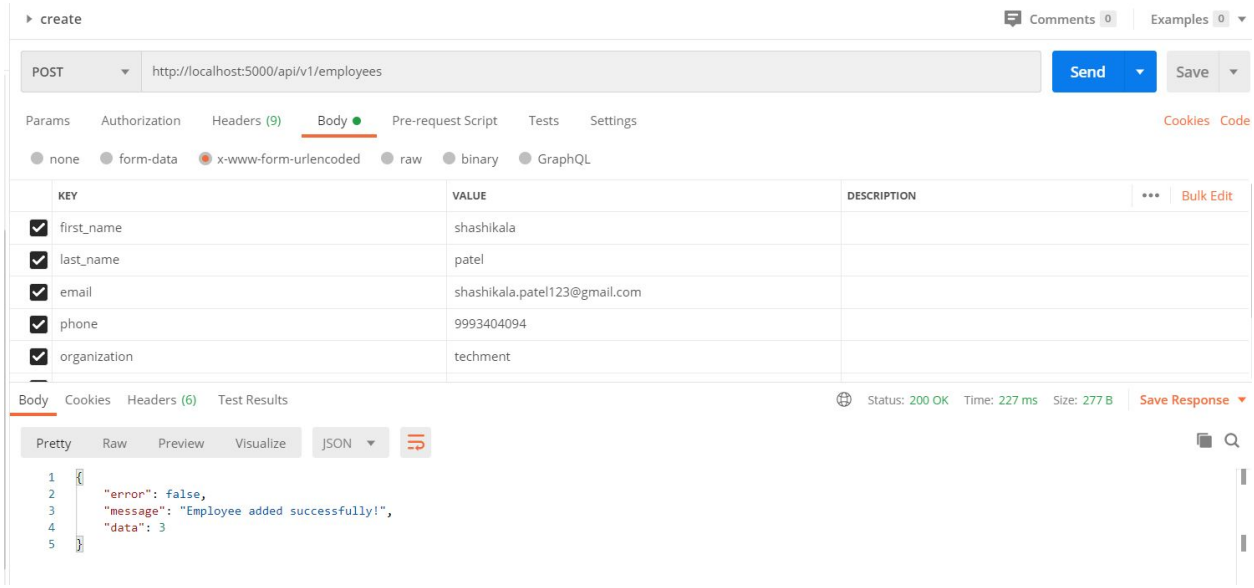
API End Points

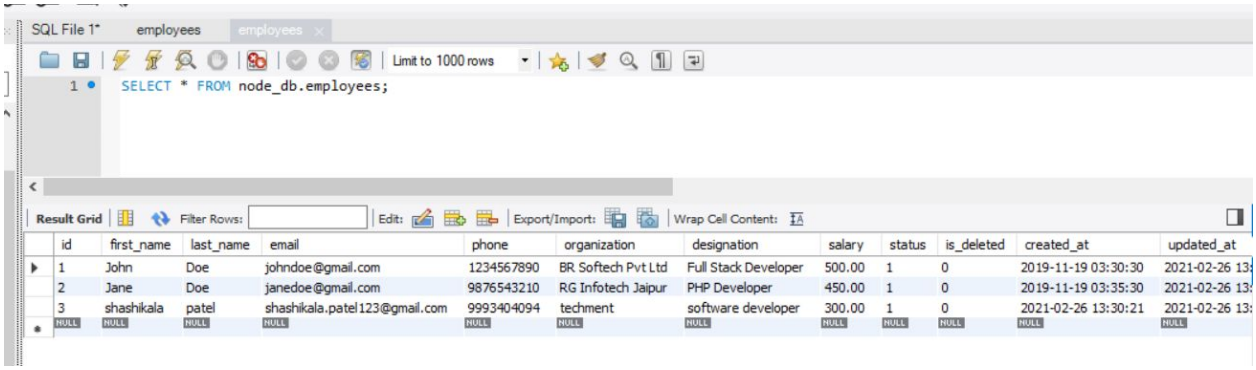
1. GET /api/v1/employees: will give all employees stored in database
2. GET /api/v1/employees/<employee_id>: will give a specific employee with employee_id.
3. POST /api/v1/employees : create a employee
4. PATCH /api/v1/employees/<employee_id>: update a employee partially
5. DELETE /api/v1/employees/<employee_id>: delete a employee
6. PUT /api/v1/employees/<employee_id>: update a employee completely

APIs Test in Postman



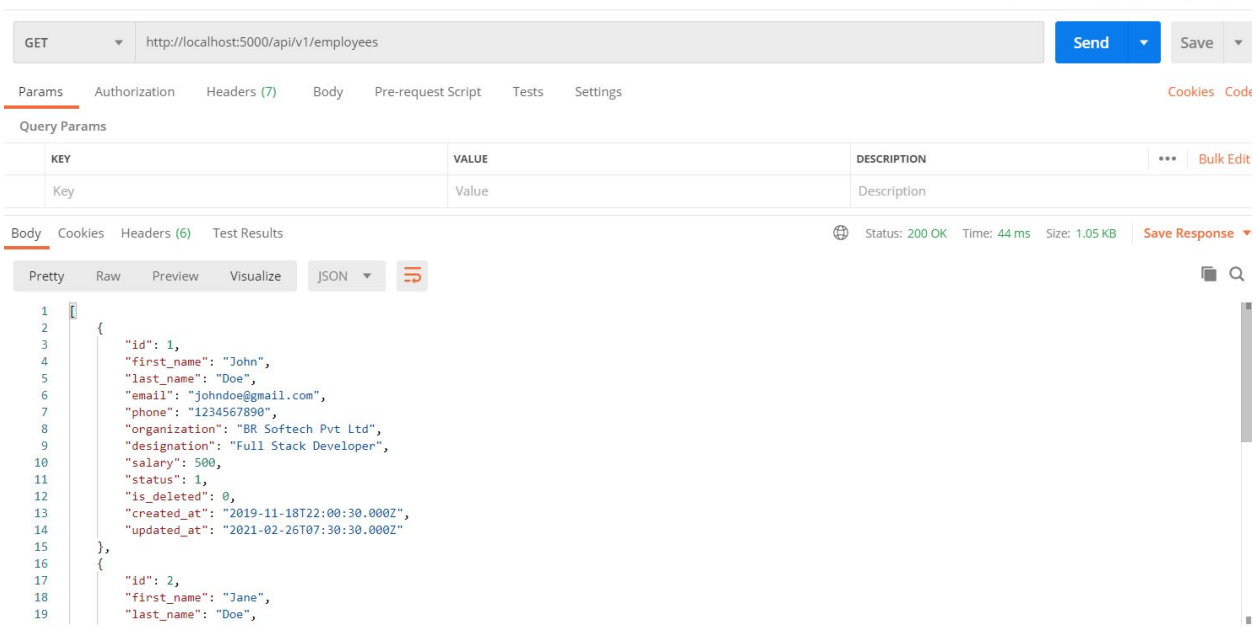
1. Creating a new employee http://localhost:5000/api/v1/employees using POST method





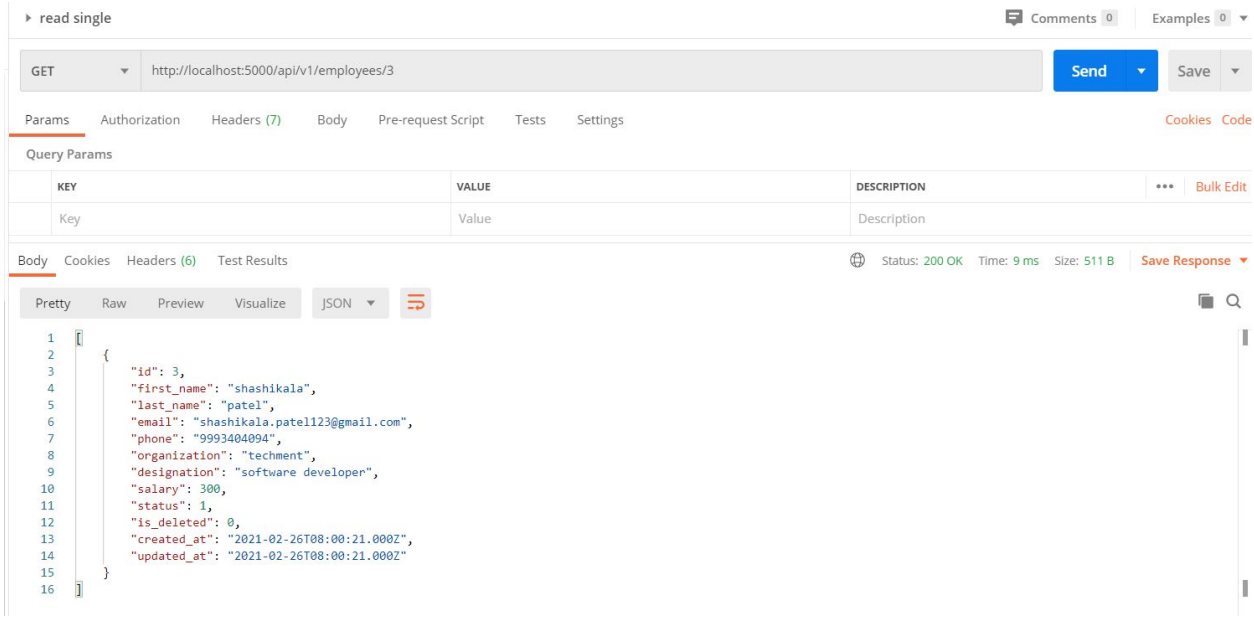
id	first_name	last_name	email	phone	organization	designation	salary	status	is_deleted	created_at	updated_at
1	John	Doe	johndoe@gmail.com	1234567890	BR Softech Pvt Ltd	Full Stack Developer	500.00	1	0	2019-11-19 03:30:30	2021-02-26 13:
2	Jane	Doe	janedoe@gmail.com	9876543210	RG Infotech Jaipur	PHP Developer	450.00	1	0	2019-11-19 03:35:30	2021-02-26 13:
3	shashikala	patel	shashikala.patel123@gmail.com	9993404094	techment	software developer	300.00	1	0	2021-02-26 13:30:21	2021-02-26 13:
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2. Get all employees list http://localhost:5000/api/v1/employees using GET method

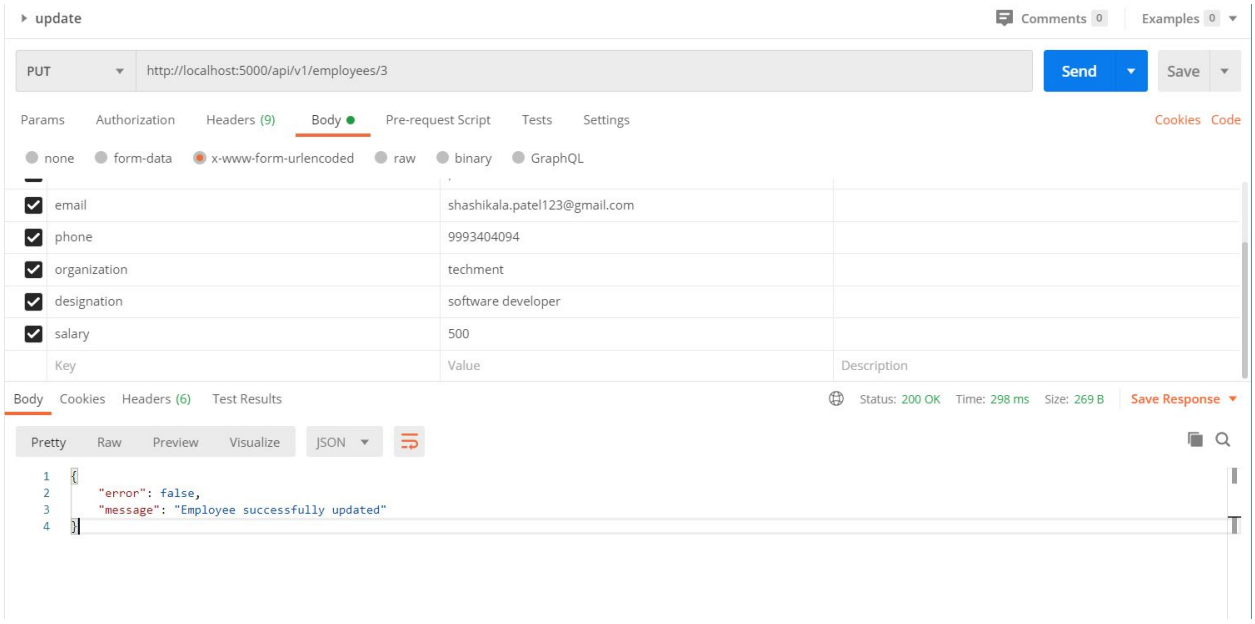


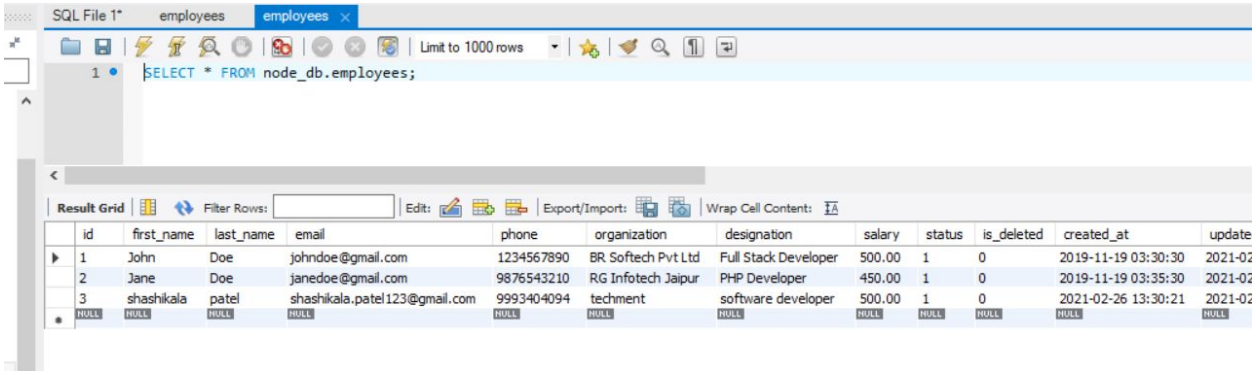

```
[
    {
        "id": 1,
        "first_name": "John",
        "last_name": "Doe",
        "email": "johndoe@gmail.com",
        "phone": "1234567890",
        "organization": "BR Softech Pvt Ltd",
        "designation": "Full Stack Developer",
        "salary": 500,
        "status": 1,
        "is_deleted": 0,
        "created_at": "2019-11-18T22:00:30.000Z",
        "updated_at": "2021-02-26T07:30:30.000Z"
    },
    {
        "id": 2,
        "first_name": "Jane",
        "last_name": "Doe",
```

3. Get specific employee http://localhost:5000/api/v1/employees/id using GET method



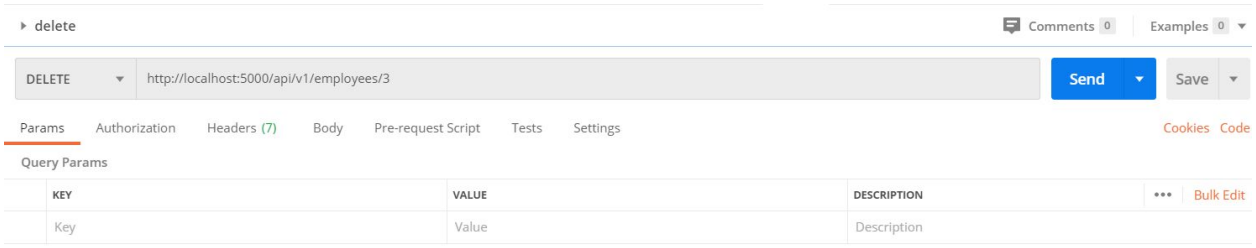
4. Update specific employee http://localhost:5000/api/v1/employees/id using PUT method

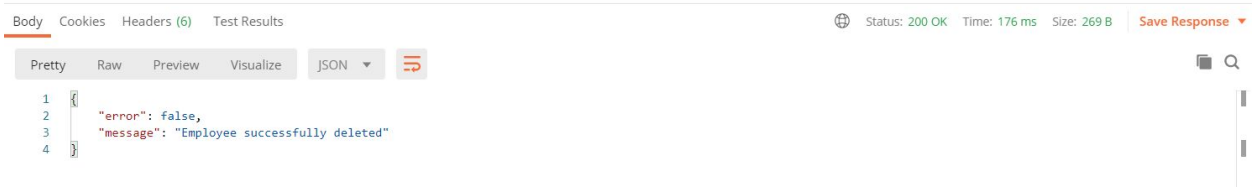


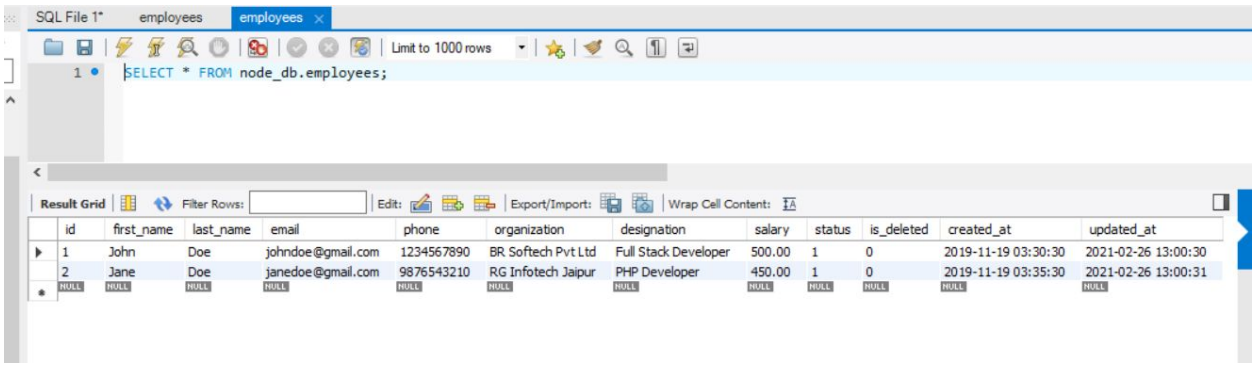


id	first_name	last_name	email	phone	organization	designation	salary	status	is_deleted	created_at	update
1	John	Doe	johndoe@gmail.com	1234567890	BR Softech Pvt Ltd	Full Stack Developer	500.00	1	0	2019-11-19 03:30:30	2021-02
2	Jane	Doe	janedoe@gmail.com	9876543210	RG Infotech Jaipur	PHP Developer	450.00	1	0	2019-11-19 03:35:30	2021-02
3	shashikala	patel	shashikala.patel123@gmail.com	9993404094	techment	software developer	500.00	1	0	2021-02-26 13:30:21	2021-02
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5. Delete specific employee http://localhost:5000/api/v1/employees/id using DELETE method







id	first_name	last_name	email	phone	organization	designation	salary	status	is_deleted	created_at	updated_at
1	John	Doe	johndoe@gmail.com	1234567890	BR Softech Pvt Ltd	Full Stack Developer	500.00	1	0	2019-11-19 03:30:30	2021-02-26 13:00:30
2	Jane	Doe	janedoe@gmail.com	9876543210	RG Infotech Jaipur	PHP Developer	450.00	1	0	2019-11-19 03:35:30	2021-02-26 13:00:31
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL