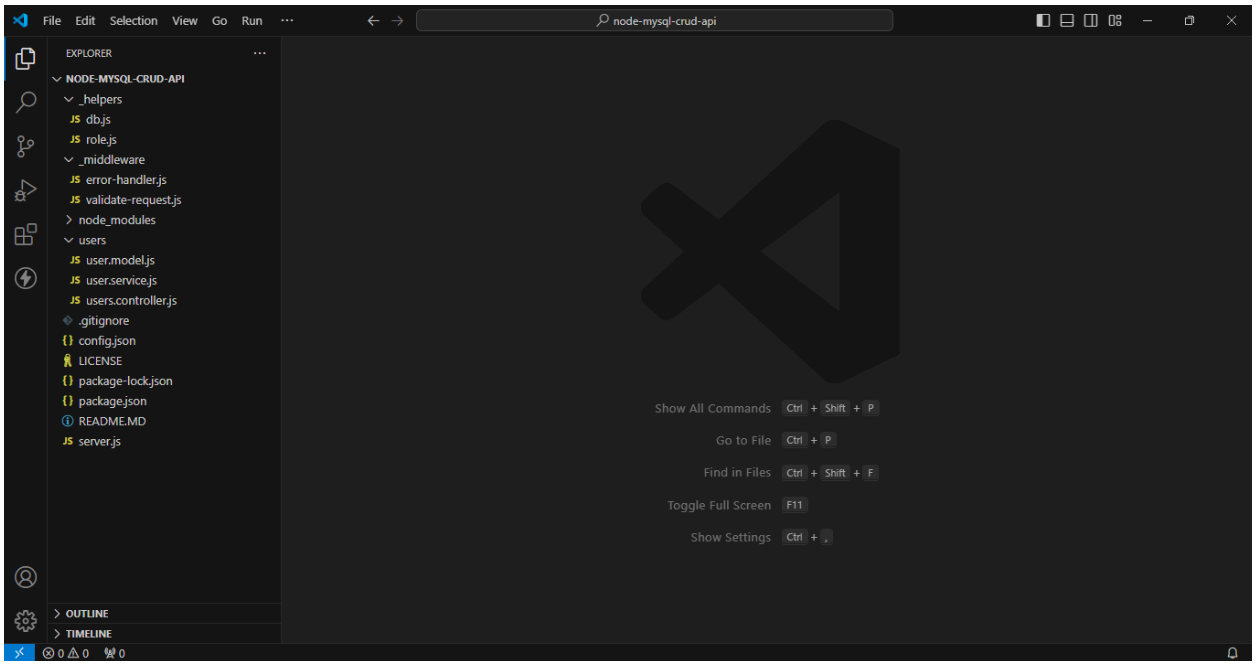
Mathew James M. Dagle IT - INTPROG32 BSIT - 3

TypeScript, Node.js, Sequelize (ORM) and MySQL - CRUD API.

Step 1: Getting started with NodeJS.

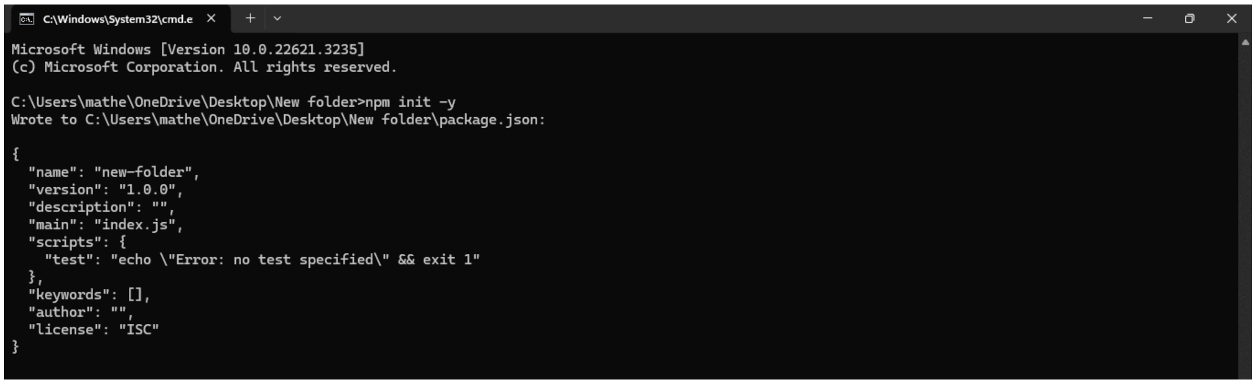
Note: to use the commands within the node package manager you need to have the NodeJS installed on your system. For reference click the link as follows: https://nodejs.org/.

Kickstart your project directory by following the same structure below:



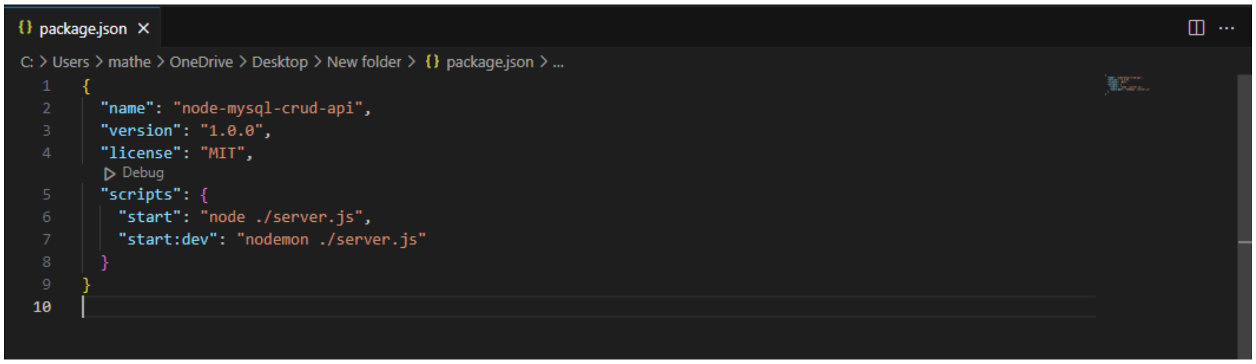
```
NODE-MYSQL-CRUD-API
  _helpers
    db.js
    role.js
  _middleware
    error-handler.js
    validate-request.js
  node_modules
  users
    user.model.js
    user.service.js
    users.controller.js
  .gitignore
  config.json
  LICENSE
  package-lock.json
  package.json
  README.MD
  server.js
```

Install and initialize the node package manager by typing the following commands on your terminal just within your project folder.



```
Microsoft Windows [Version 10.0.22621.3235]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mathe\OneDrive\Desktop\New folder>npm init -y
Wrote to C:\Users\mathe\OneDrive\Desktop\New folder\package.json:

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

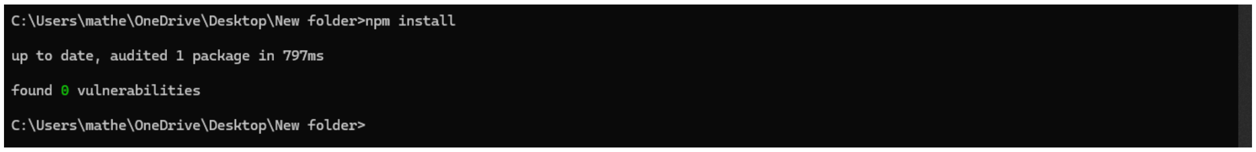
Edit your package.json file to look like this.



package.json
C: > Users > mathe > OneDrive > Desktop > New folder > package.json > ...

```json
{
  "name": "node-mysql-crud-api",
  "version": "1.0.0",
  "license": "MIT",
  "scripts": {
    "start": "node ./server.js",
    "start:dev": "nodemon ./server.js"
  }
}

```

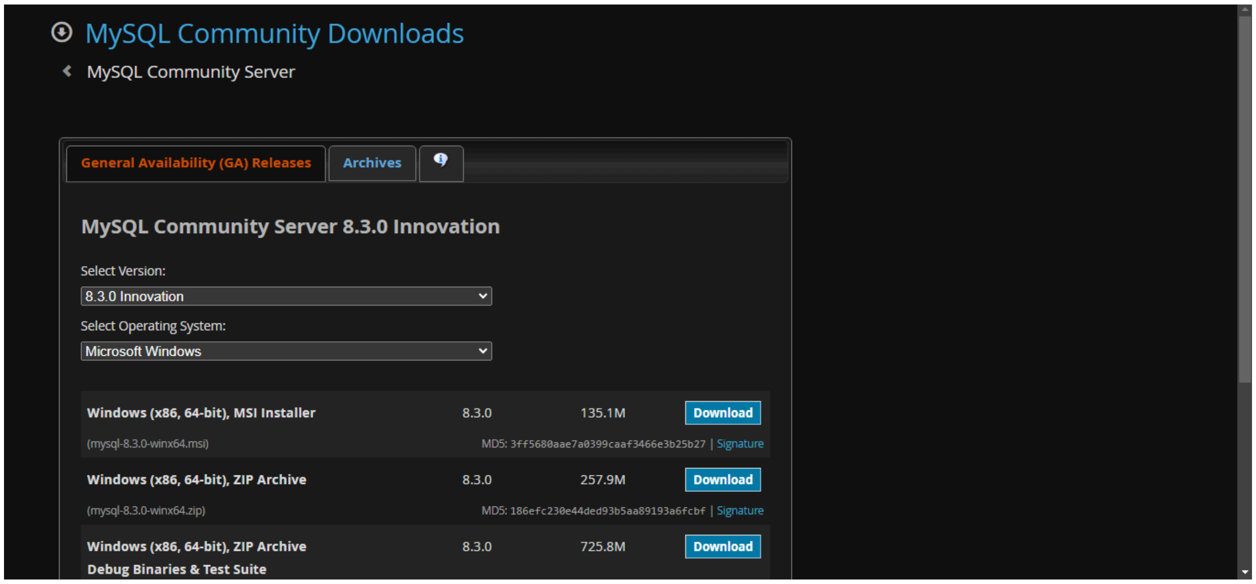
And then afterwards run npm install or npm i.



```
C:\Users\mathe\OneDrive\Desktop\New folder>npm install

up to date, audited 1 package in 797ms

found 0 vulnerabilities

C:\Users\mathe\OneDrive\Desktop\New folder>
```

Step 2: Setting up MySQL

You must have a MySQL server instance for the API to connect to. The Community Server version is available to download for free from https://dev.mysql.com/downloads/mysql/.

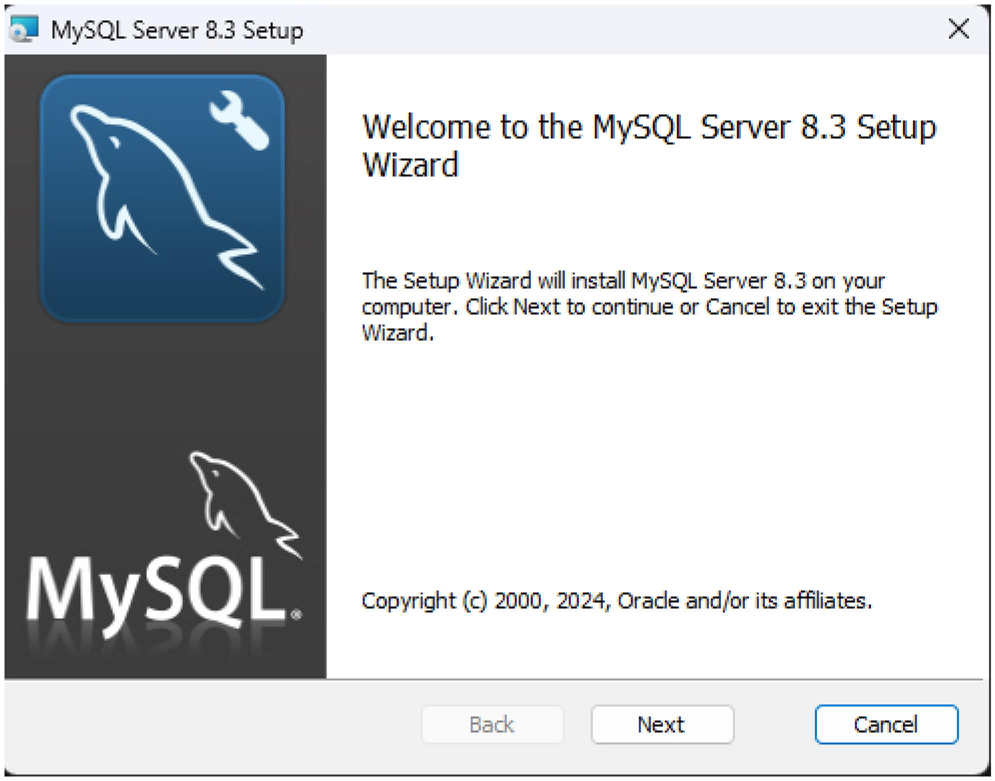


MySQL Community Server 8.3.0 Innovation

Select Version: 8.3.0 Innovation

Select Operating System: Microsoft Windows

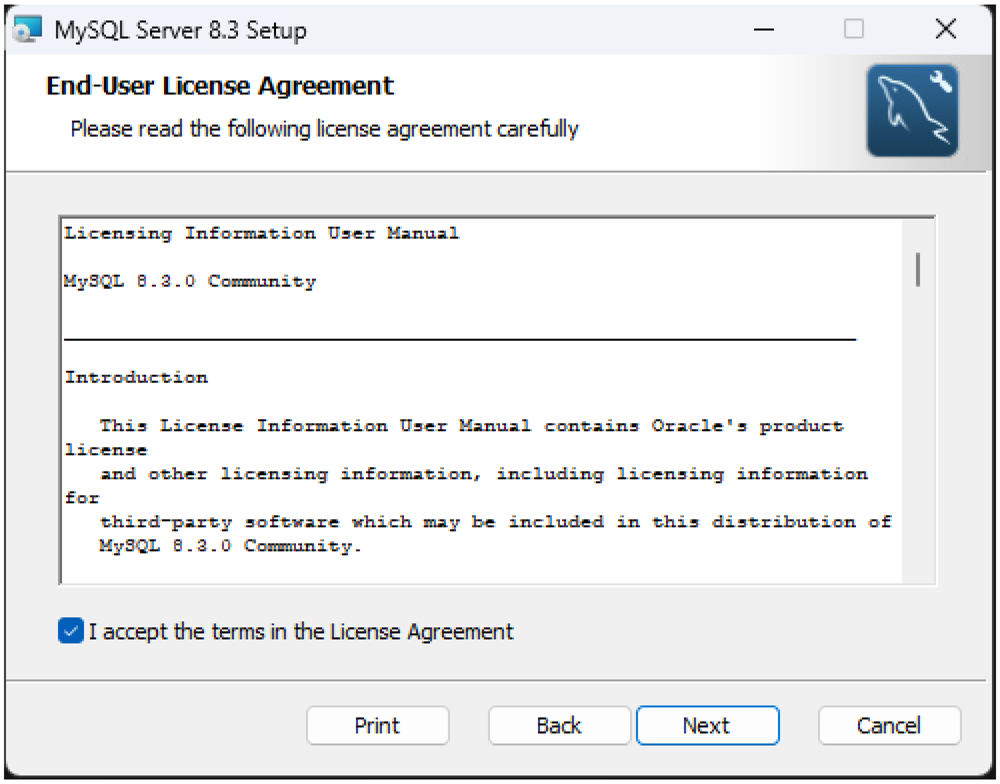
Windows (x86, 64-bit), MSI Installer	8.3.0	135.1M	Download
(mysql-8.3.0-winx64.msi)	MD5: 3ff5680aae7a0399caaf3466e3b25b27 | Signature		
Windows (x86, 64-bit), ZIP Archive	8.3.0	257.9M	Download
(mysql-8.3.0-winx64.zip)	MD5: 186efc230e44ded93b5aa89193a6fcbf | Signature		
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite	8.3.0	725.8M	Download



Welcome to the MySQL Server 8.3 Setup Wizard

The Setup Wizard will install MySQL Server 8.3 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

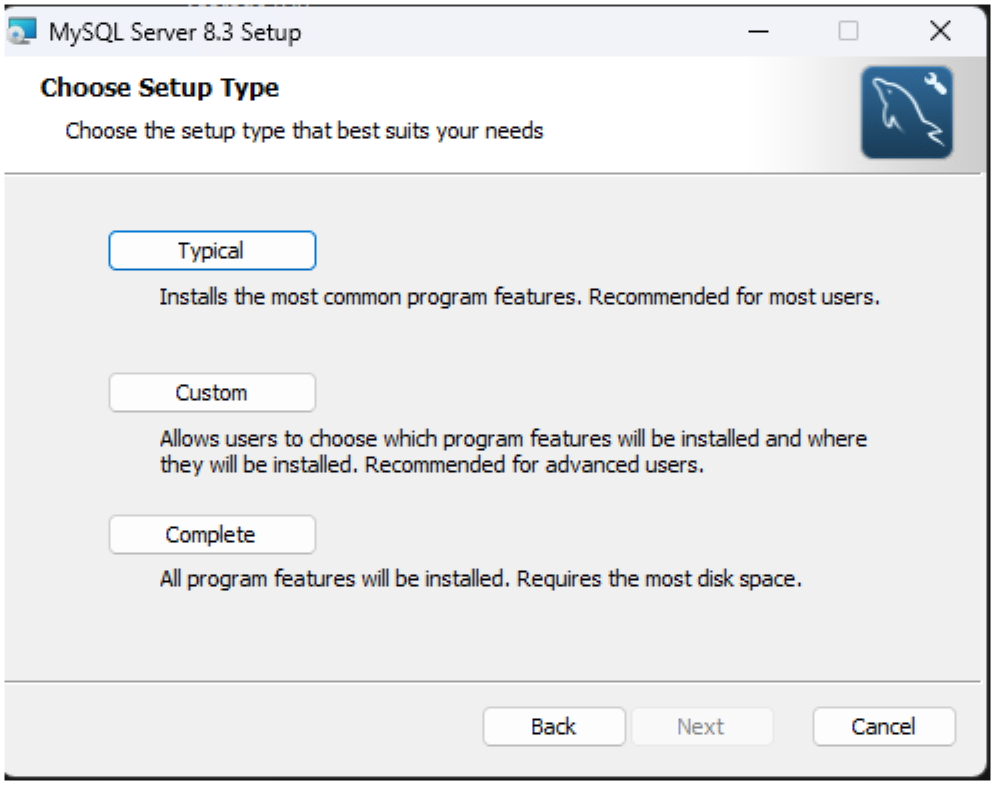


End-User License Agreement

Please read the following license agreement carefully

```
Licensing Information User Manual

MySQL 8.3.0 Community

____________________________________________________________

Introduction

   This License Information User Manual contains Oracle's product
license
   and other licensing information, including licensing information
for
   third-party software which may be included in this distribution of
   MySQL 8.3.0 Community.
```

[✓] I accept the terms in the License Agreement



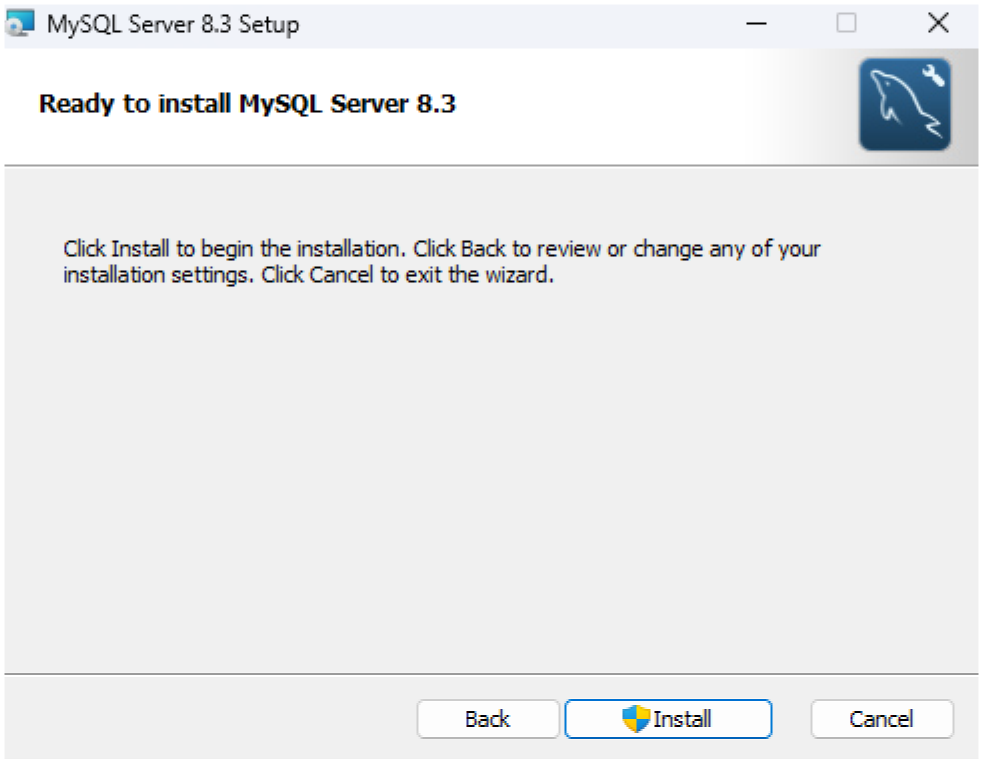
Choose Setup Type

Choose the setup type that best suits your needs

Installs the most common program features. Recommended for most users.

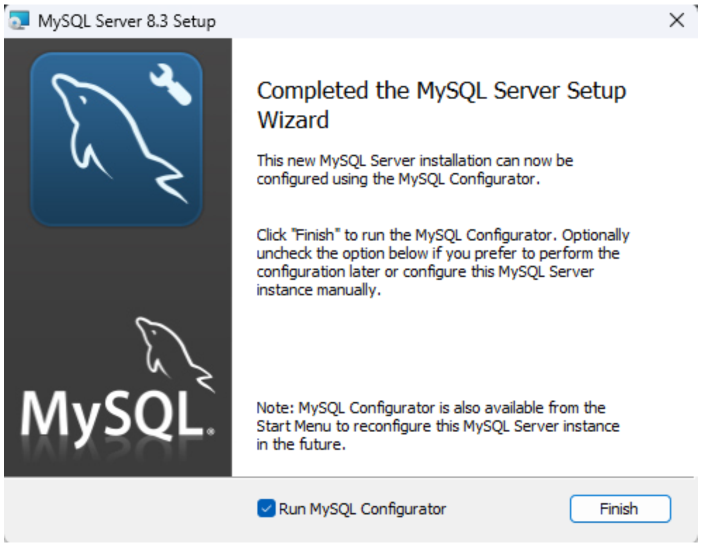
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

All program features will be installed. Requires the most disk space.



Ready to install MySQL Server 8.3

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.



Completed the MySQL Server Setup Wizard

This new MySQL Server installation can now be configured using the MySQL Configurator.

Click "Finish" to run the MySQL Configurator. Optionally uncheck the option below if you prefer to perform the configuration later or configure this MySQL Server instance manually.

Note: MySQL Configurator is also available from the Start Menu to reconfigure this MySQL Server instance in the future.

[✓] Run MySQL Configurator

Further Post-Installation instructions available at https://dev.mysql.com/doc/refman/8.0/en/installing.html.

Step 3: Installing project dependencies.

Your Node.js project requires dependencies to use Sequelize and be able to interact with MySql. On your terminal, install them like so:

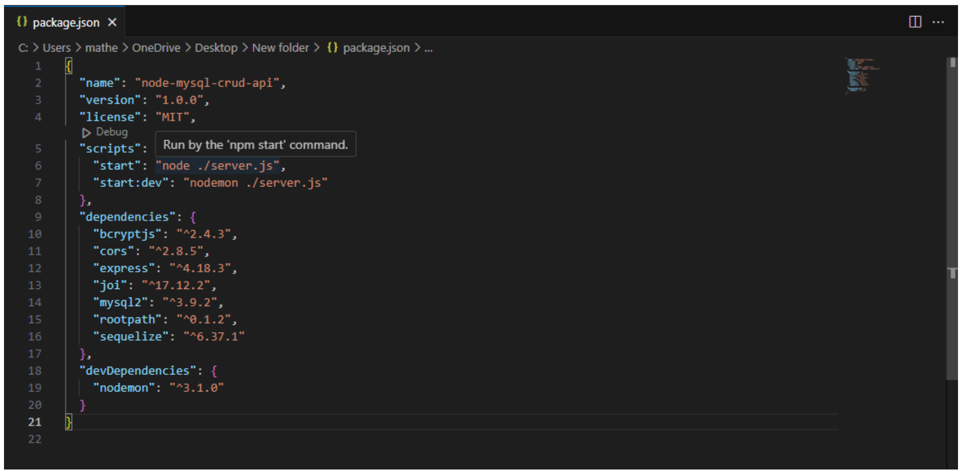
- npm i sequelize mysql2

You will also need the help of other dependencies for your project to work:

- npm i express cors bcryptjs joi rootpath

Install this package to power up your development workflow:

- npm i -D nodemon


{} package.json ×
C: > Users > mathe > OneDrive > Desktop > New folder > {} package.json > ...
▷ Debug
Run by the 'npm start' command.

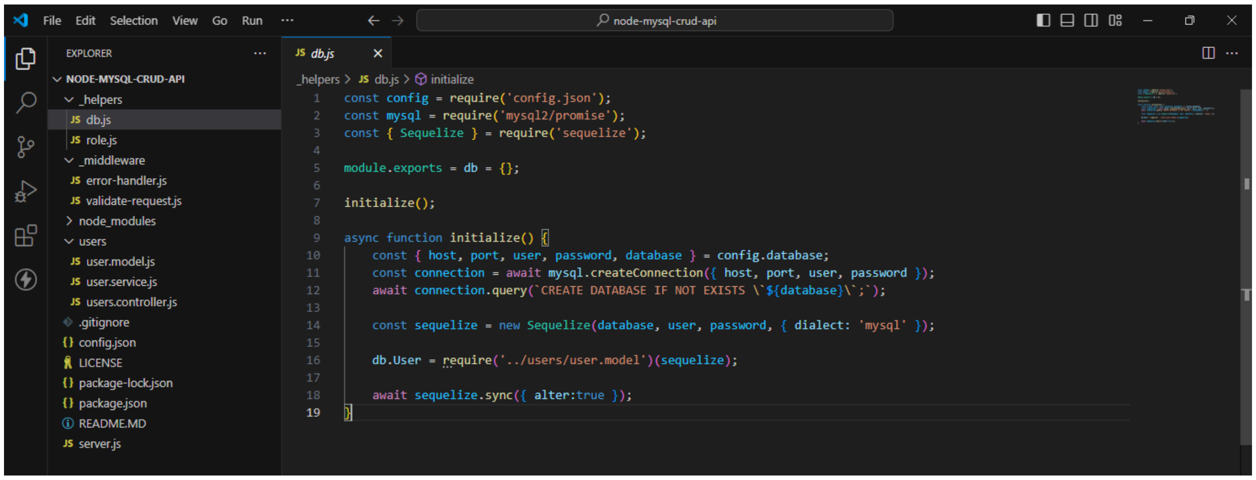
```json
{
  "name": "node-mysql-crud-api",
  "version": "1.0.0",
  "license": "MIT",
  "scripts": {
    "start": "node ./server.js",
    "start:dev": "nodemon ./server.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "express": "^4.18.3",
    "joi": "^17.12.2",
    "mysql2": "^3.9.2",
    "rootpath": "^0.1.2",
    "sequelize": "^6.37.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

By now, your package.json file should be updated to look like this.

Step 4: Filling up the contents of your JavaScript and JSON files.

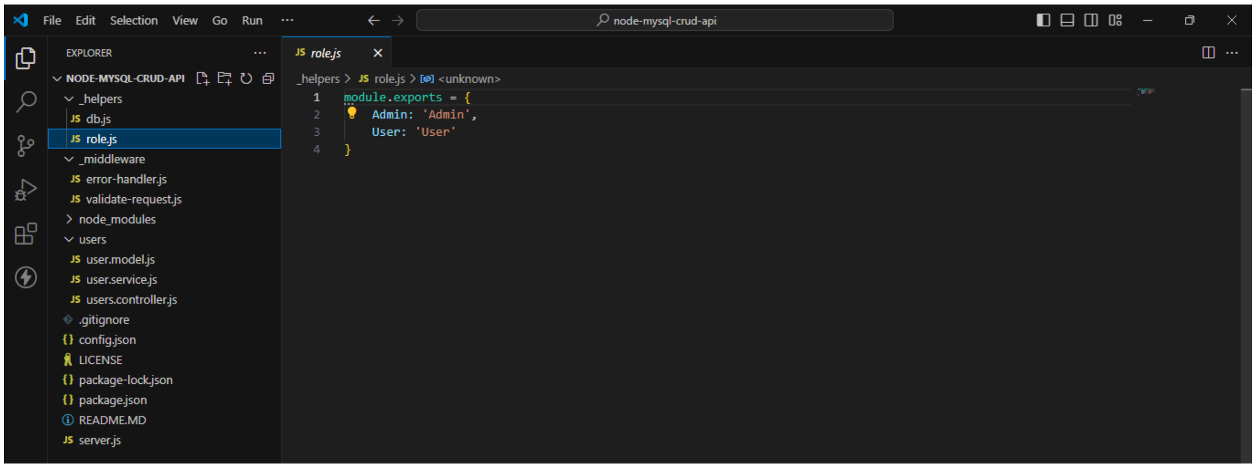
Helpers Folder

Path: /_helpers/db.js



```javascript
const config = require('config.json');
const mysql = require('mysql2/promise');
const { Sequelize } = require('sequelize');

module.exports = db = {};

initialize();

async function initialize() {
    const { host, port, user, password, database } = config.database;
    const connection = await mysql.createConnection({ host, port, user, password });
    await connection.query(`CREATE DATABASE IF NOT EXISTS \`${database}\`;`);

    const sequelize = new Sequelize(database, user, password, { dialect: 'mysql' });

    db.User = require('../users/user.model')(sequelize);

    await sequelize.sync({ alter:true });
}
```

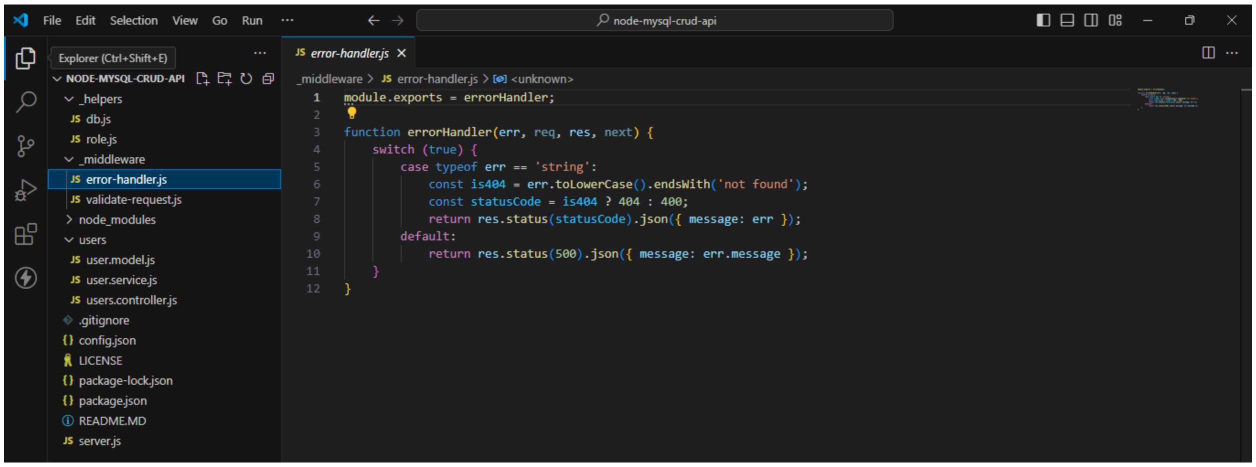
Path: /_helpers/role.js



```javascript
module.exports = {
    Admin: 'Admin',
    User: 'User'
}
```

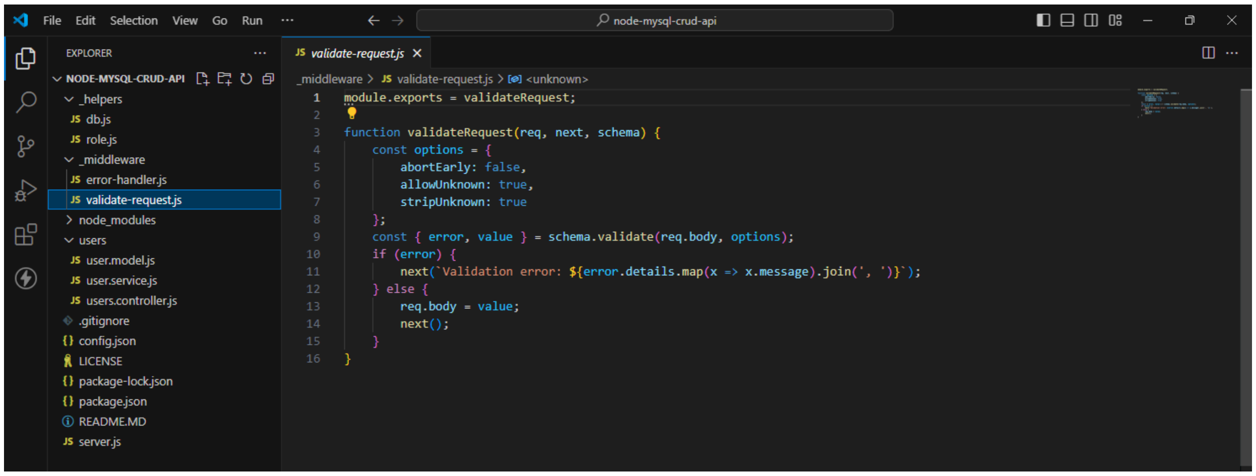
Express.js Middleware Folder

Path: /_middleware/error-handler.js



```
module.exports = errorHandler;

function errorHandler(err, req, res, next) {
    switch (true) {
        case typeof err == 'string':
            const is404 = err.toLowerCase().endsWith('not found');
            const statusCode = is404 ? 404 : 400;
            return res.status(statusCode).json({ message: err });
        default:
            return res.status(500).json({ message: err.message });
    }
}
```

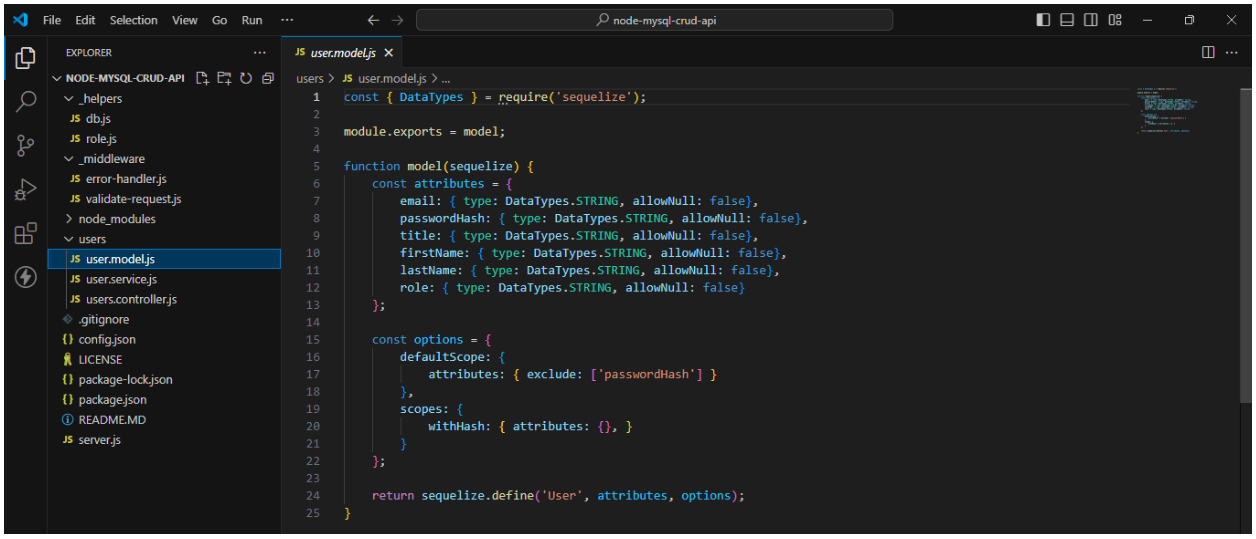
Path: /_middleware/validate-request.js



```
module.exports = validateRequest;

function validateRequest(req, next, schema) {
    const options = {
        abortEarly: false,
        allowUnknown: true,
        stripUnknown: true
    };
    const { error, value } = schema.validate(req.body, options);
    if (error) {
        next(`Validation error: ${error.details.map(x => x.message).join(', ')}`);
    } else {
        req.body = value;
        next();
    }
}
```

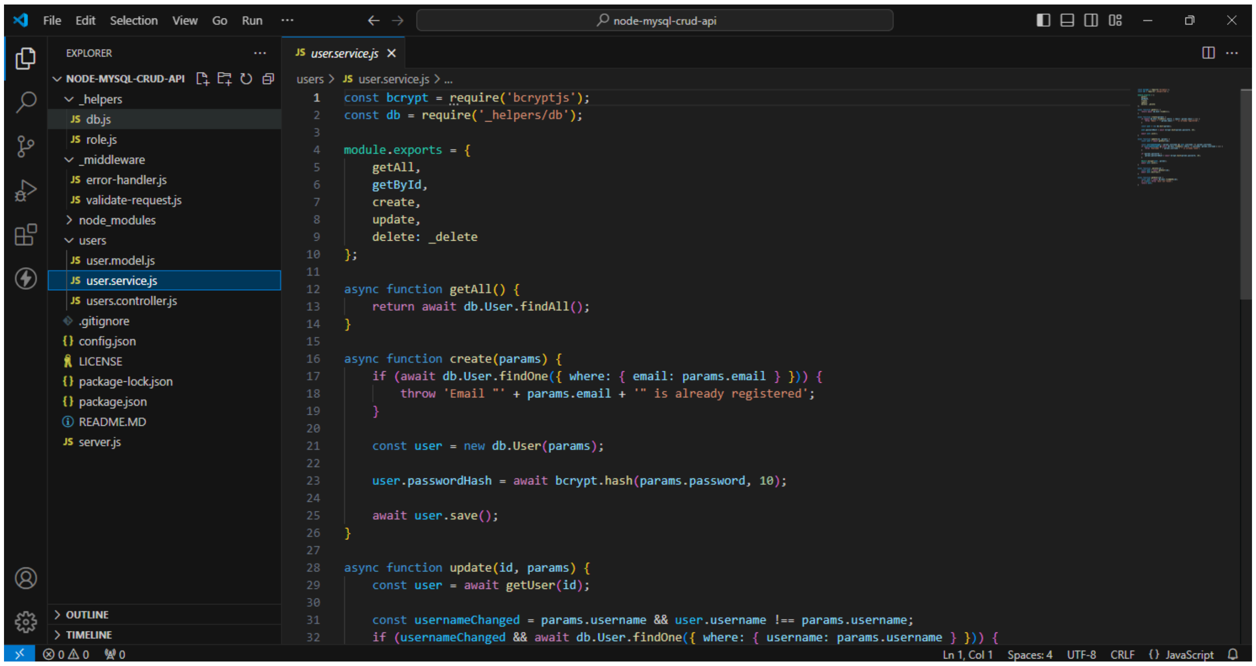
Users Feature Folder

Path: /users/user.model.js

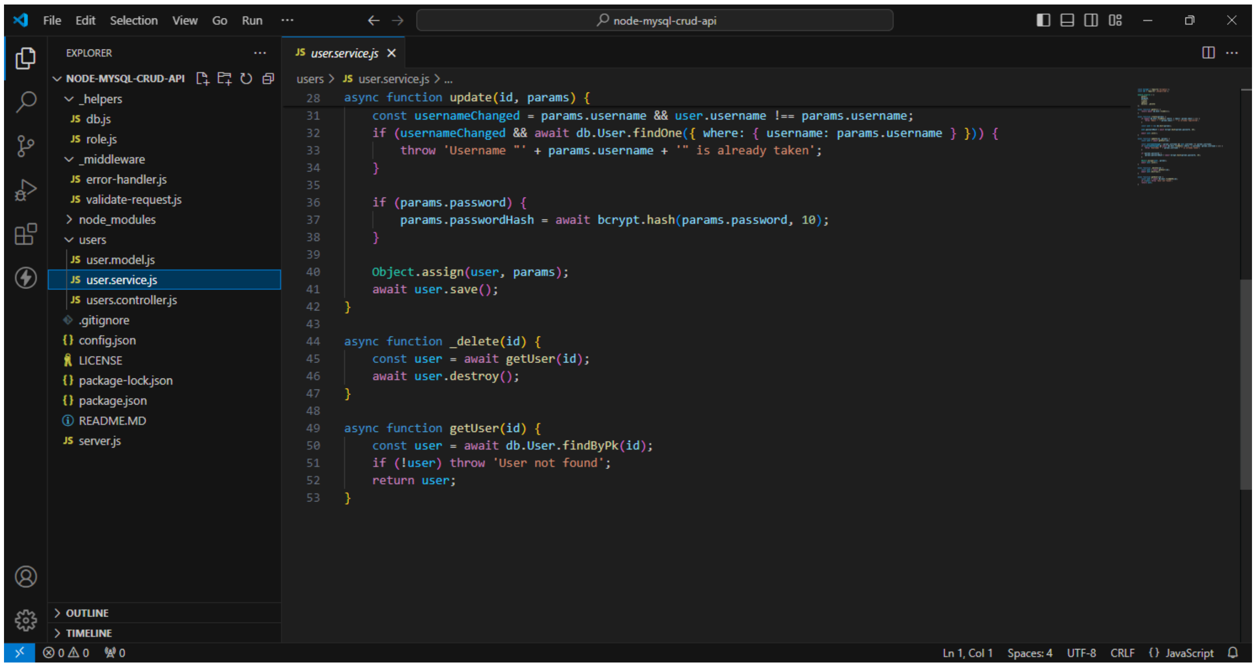


```javascript
const { DataTypes } = require('sequelize');

module.exports = model;

function model(sequelize) {
    const attributes = {
        email: { type: DataTypes.STRING, allowNull: false},
        passwordHash: { type: DataTypes.STRING, allowNull: false},
        title: { type: DataTypes.STRING, allowNull: false},
        firstName: { type: DataTypes.STRING, allowNull: false},
        lastName: { type: DataTypes.STRING, allowNull: false},
        role: { type: DataTypes.STRING, allowNull: false}
    };

    const options = {
        defaultScope: {
            attributes: { exclude: ['passwordHash'] }
        },
        scopes: {
            withHash: { attributes: {}, }
        }
    };

    return sequelize.define('User', attributes, options);
}
```

Path: /users/user.service.js

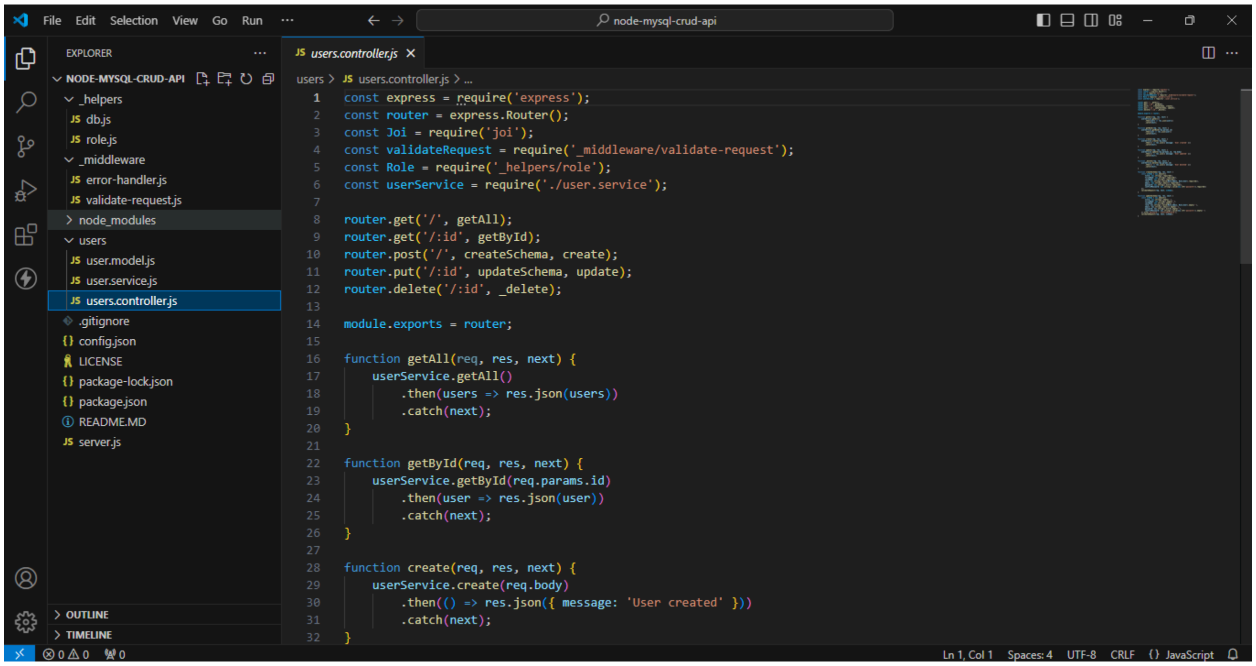


```javascript
const bcrypt = require('bcryptjs');
const db = require('_helpers/db');

module.exports = {
    getAll,
    getById,
    create,
    update,
    delete: _delete
};

async function getAll() {
    return await db.User.findAll();
}

async function create(params) {
    if (await db.User.findOne({ where: { email: params.email } })) {
        throw 'Email "' + params.email + '" is already registered';
    }

    const user = new db.User(params);

    user.passwordHash = await bcrypt.hash(params.password, 10);

    await user.save();
}

async function update(id, params) {
    const user = await getUser(id);

    const usernameChanged = params.username && user.username !== params.username;
    if (usernameChanged && await db.User.findOne({ where: { username: params.username } })) {
```

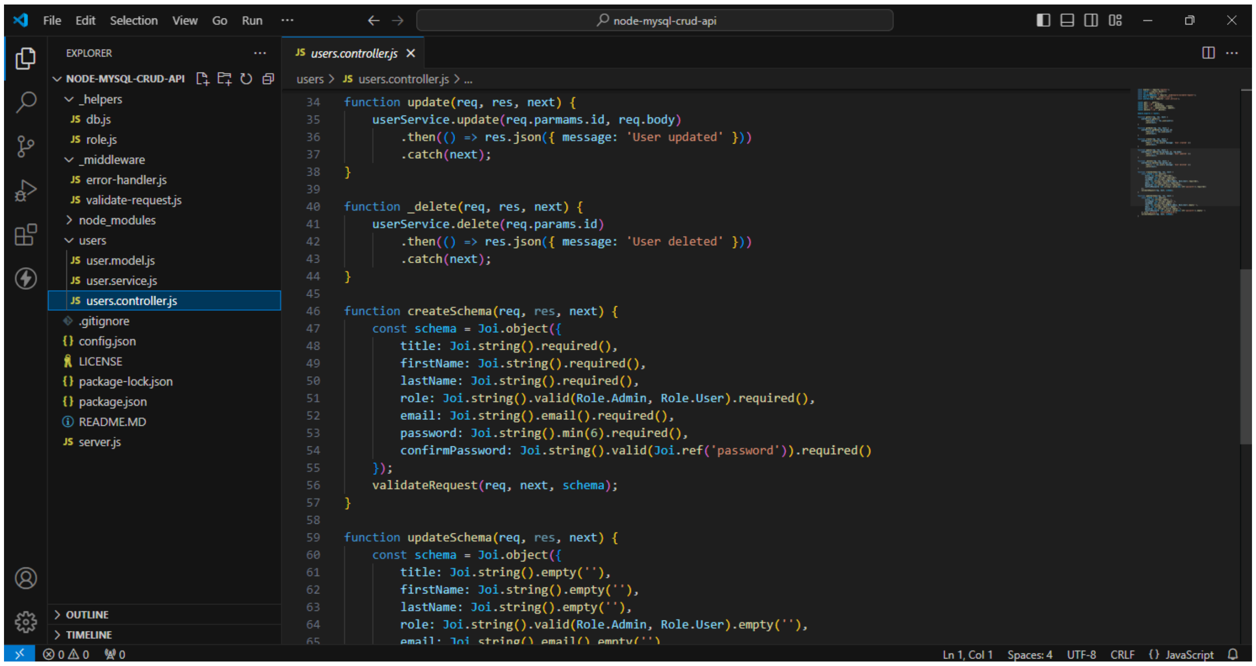


```
async function update(id, params) {
    const usernameChanged = params.username && user.username !== params.username;
    if (usernameChanged && await db.User.findOne({ where: { username: params.username } })) {
        throw 'Username "' + params.username + '" is already taken';
    }

    if (params.password) {
        params.passwordHash = await bcrypt.hash(params.password, 10);
    }

    Object.assign(user, params);
    await user.save();
}

async function _delete(id) {
    const user = await getUser(id);
    await user.destroy();
}

async function getUser(id) {
    const user = await db.User.findByPk(id);
    if (!user) throw 'User not found';
    return user;
}
```

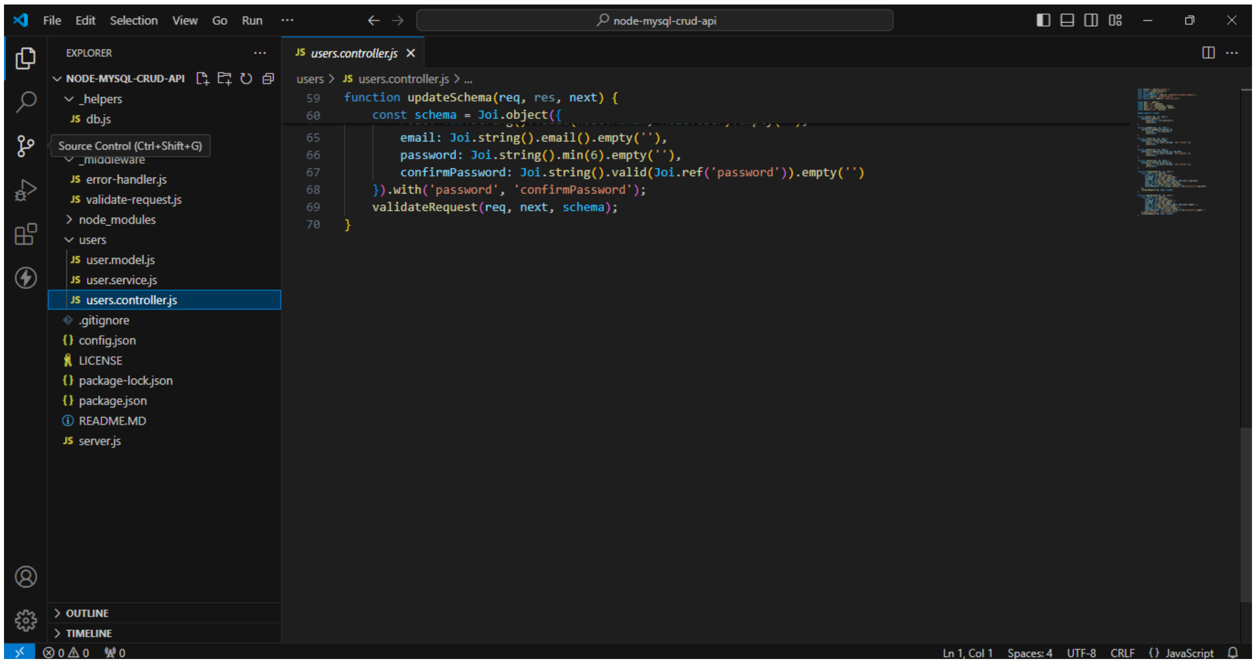
Path: /users/users.controller.js



```
const express = require('express');
const router = express.Router();
const Joi = require('joi');
const validateRequest = require('_middleware/validate-request');
const Role = require('_helpers/role');
const userService = require('./user.service');

router.get('/', getAll);
router.get('/:id', getById);
router.post('/', createSchema, create);
router.put('/:id', updateSchema, update);
router.delete('/:id', _delete);

module.exports = router;

function getAll(req, res, next) {
    userService.getAll()
        .then(users => res.json(users))
        .catch(next);
}

function getById(req, res, next) {
    userService.getById(req.params.id)
        .then(user => res.json(user))
        .catch(next);
}

function create(req, res, next) {
    userService.create(req.body)
        .then(() => res.json({ message: 'User created' }))
        .catch(next);
}
```



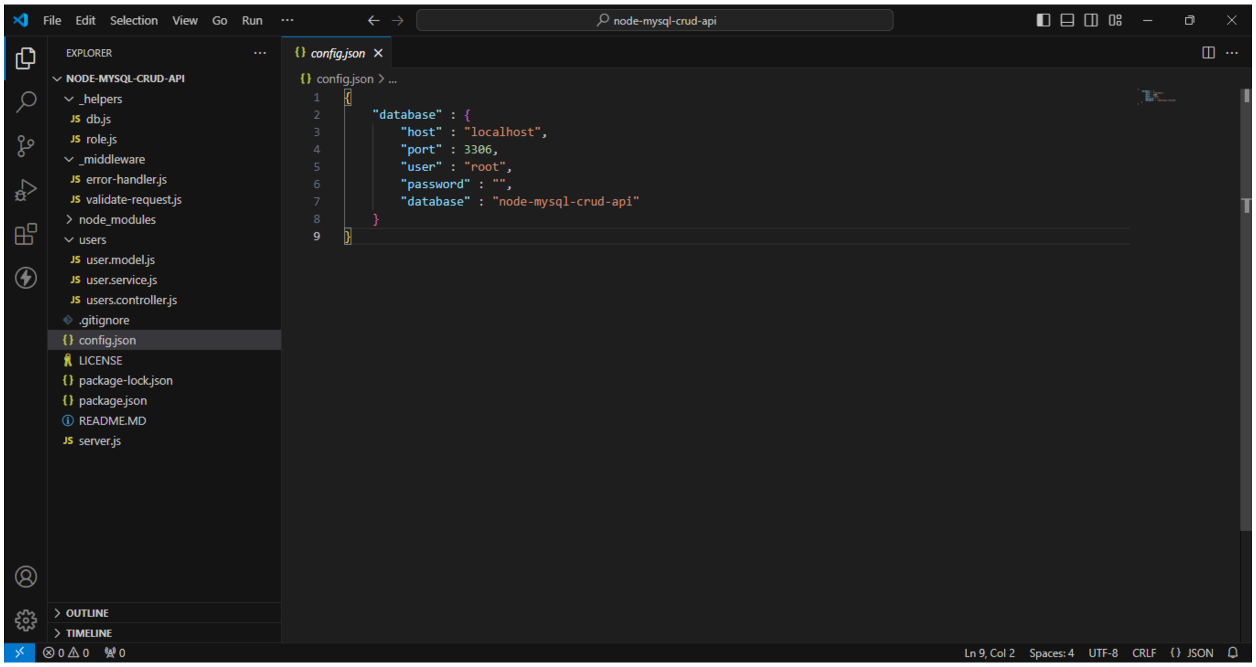
```
function update(req, res, next) {
    userService.update(req.parmams.id, req.body)
        .then(() => res.json({ message: 'User updated' }))
        .catch(next);
}

function _delete(req, res, next) {
    userService.delete(req.params.id)
        .then(() => res.json({ message: 'User deleted' }))
        .catch(next);
}

function createSchema(req, res, next) {
    const schema = Joi.object({
        title: Joi.string().required(),
        firstName: Joi.string().required(),
        lastName: Joi.string().required(),
        role: Joi.string().valid(Role.Admin, Role.User).required(),
        email: Joi.string().email().required(),
        password: Joi.string().min(6).required(),
        confirmPassword: Joi.string().valid(Joi.ref('password')).required()
    });
    validateRequest(req, next, schema);
}

function updateSchema(req, res, next) {
    const schema = Joi.object({
        title: Joi.string().empty(''),
        firstName: Joi.string().empty(''),
        lastName: Joi.string().empty(''),
        role: Joi.string().valid(Role.Admin, Role.User).empty(''),
```



```
function updateSchema(req, res, next) {
    const schema = Joi.object({
        email: Joi.string().email().empty(''),
        password: Joi.string().min(6).empty(''),
        confirmPassword: Joi.string().valid(Joi.ref('password')).empty('')
    }).with('password', 'confirmPassword');
    validateRequest(req, next, schema);
}
```

Api Config

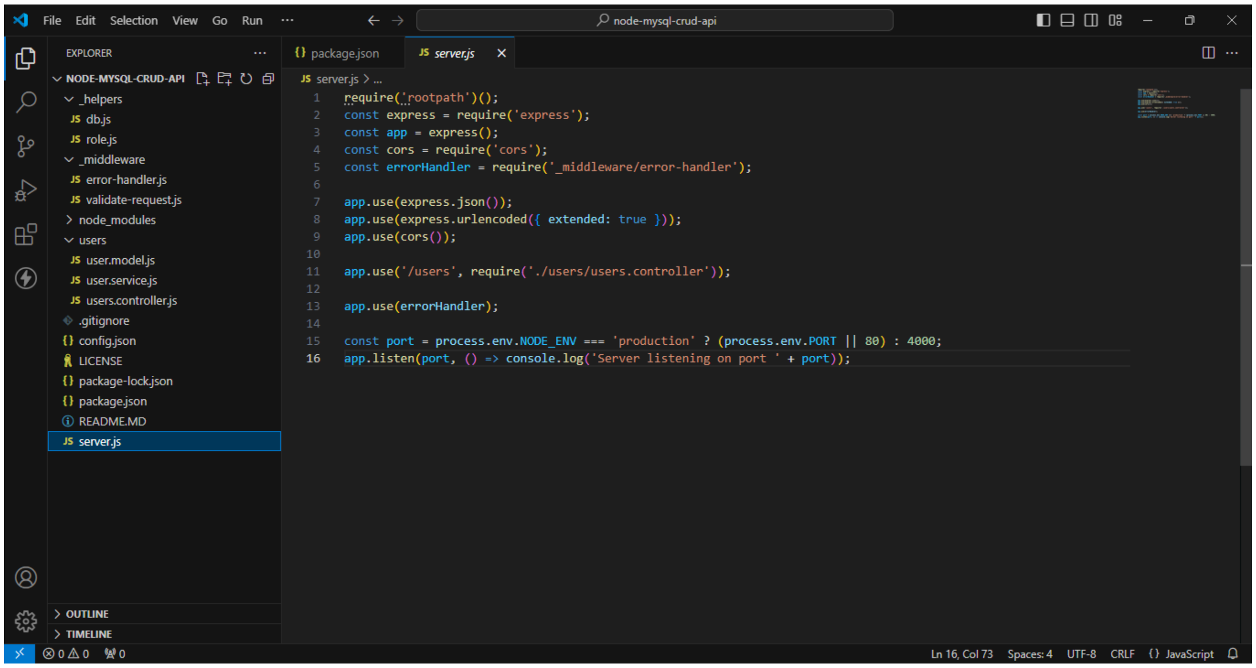
Path: /config.json



```json
{
    "database" : {
        "host" : "localhost",
        "port" : 3306,
        "user" : "root",
        "password" : "",
        "database" : "node-mysql-crud-api"
    }
}
```

Server Startup File

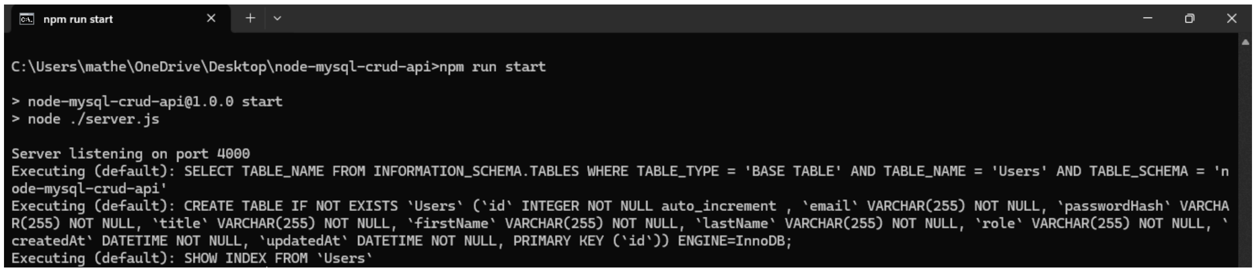
Path: /server.js



```javascript
require('rootpath')();
const express = require('express');
const app = express();
const cors = require('cors');
const errorHandler = require('_middleware/error-handler');

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

app.use('/users', require('./users/users.controller'));

app.use(errorHandler);

const port = process.env.NODE_ENV === 'production' ? (process.env.PORT || 80) : 4000;
app.listen(port, () => console.log('Server listening on port ' + port));
```

Step 4: Testing the Node + MySQL CRUD API

- Make sure that all required npm packages are installed.
- Update the database credentials in /config.json to connect to your MySQL server instance, and ensure MySQL server is running.
- Start the API by running npm start (or npm run start:dev to start with nodemon) from the command line in the project root folder, you should see the message Server listening on port 4000.

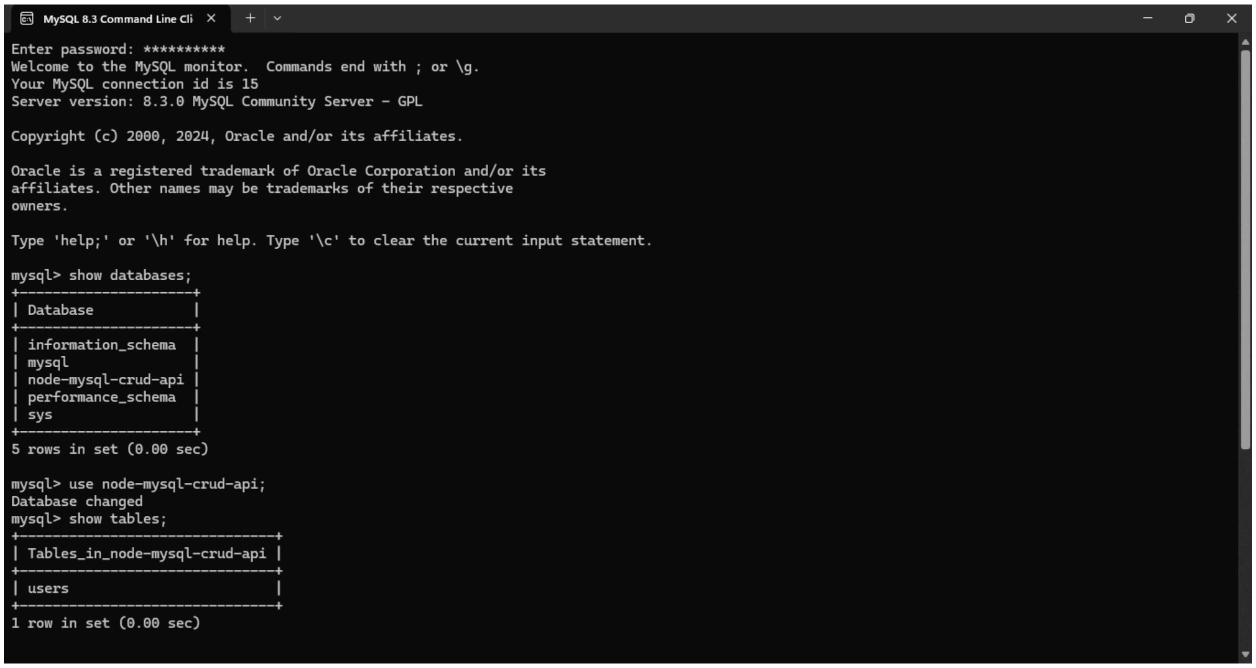


npm run start

```
C:\Users\mathe\OneDrive\Desktop\node-mysql-crud-api>npm run start

> node-mysql-crud-api@1.0.0 start
> node ./server.js

Server listening on port 4000
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Users' AND TABLE_SCHEMA = 'n
ode-mysql-crud-api'
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER NOT NULL auto_increment , `email` VARCHAR(255) NOT NULL, `passwordHash` VARCHA
R(255) NOT NULL, `title` VARCHAR(255) NOT NULL, `firstName` VARCHAR(255) NOT NULL, `lastName` VARCHAR(255) NOT NULL, `role` VARCHAR(255) NOT NULL, `
createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Users`
```

MySQL Command Line Interface (Showing the database and table automatically created)

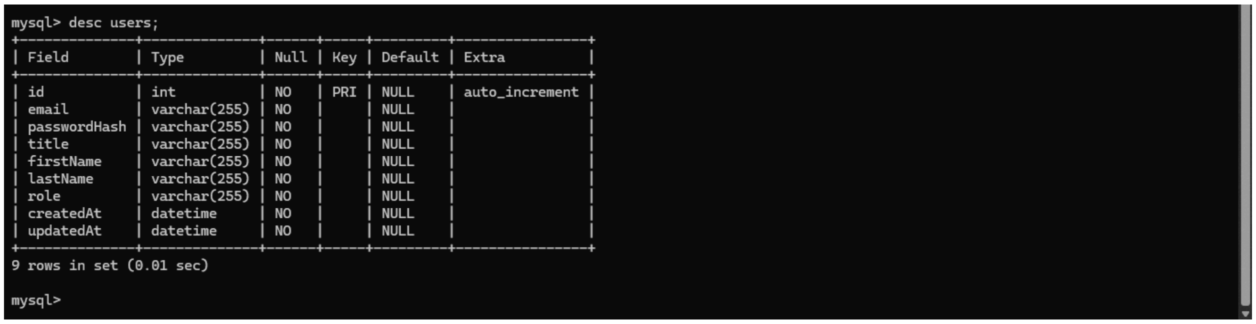
To show available databases, type in “show databases;”
And “show tables;” to look for the tables created.



MySQL 8.3 Command Line Cli

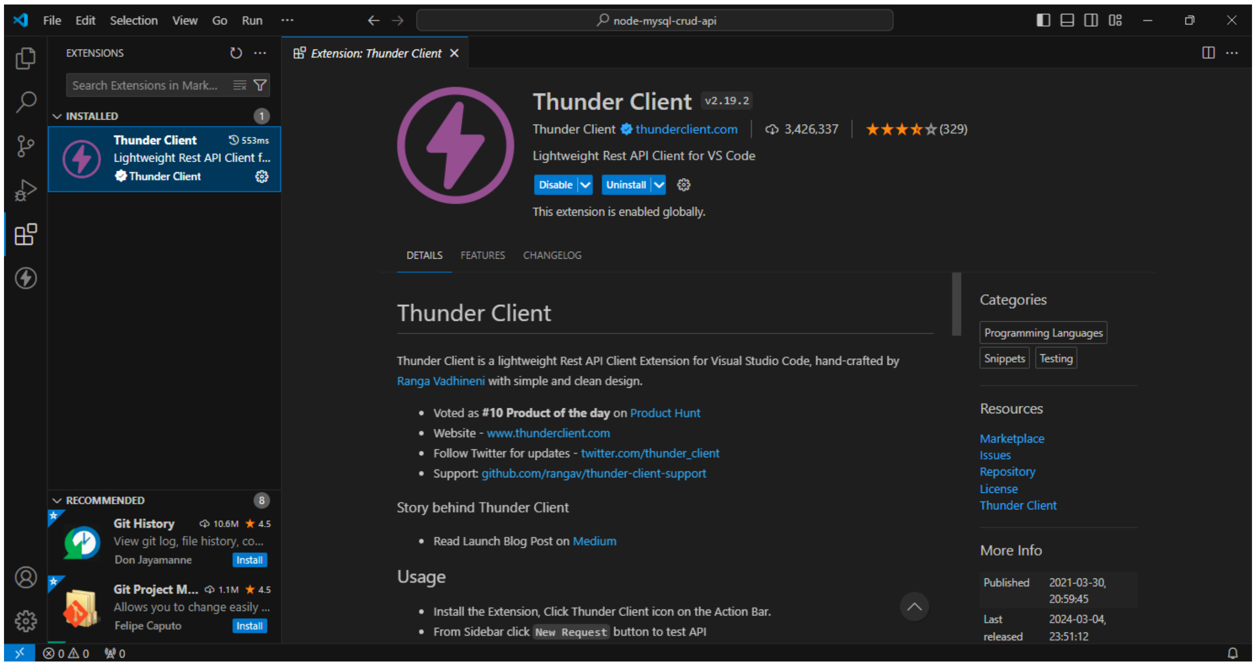
```
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+---------------------+
| Database            |
+---------------------+
| information_schema  |
| mysql               |
| node-mysql-crud-api |
| performance_schema  |
| sys                 |
+---------------------+
5 rows in set (0.00 sec)

mysql> use node-mysql-crud-api;
Database changed
mysql> show tables;
+-------------------------------+
| Tables_in_node-mysql-crud-api |
+-------------------------------+
| users                         |
+-------------------------------+
1 row in set (0.00 sec)
```

“desc users;” is use to describe table definition



```
mysql> desc users;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int          | NO   | PRI | NULL    | auto_increment |
| email        | varchar(255) | NO   |     | NULL    |                |
| passwordHash | varchar(255) | NO   |     | NULL    |                |
| title        | varchar(255) | NO   |     | NULL    |                |
| firstName    | varchar(255) | NO   |     | NULL    |                |
| lastName     | varchar(255) | NO   |     | NULL    |                |
| role         | varchar(255) | NO   |     | NULL    |                |
| createdAt    | datetime     | NO   |     | NULL    |                |
| updatedAt    | datetime     | NO   |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
9 rows in set (0.01 sec)

mysql>
```

- You can test the API directly with a tool such as Postman or VSCode extension ThunderClient. This time around we use Thunder Client.



Thunder Client v2.19.2

Thunder Client thunderclient.com | 3,426,337 | ★★★★★ (329)

Lightweight Rest API Client for VS Code

This extension is enabled globally.

Thunder Client

Thunder Client is a lightweight Rest API Client Extension for Visual Studio Code, hand-crafted by Ranga Vadhineni with simple and clean design.

- Voted as **#10 Product of the day** on Product Hunt
- Website - www.thunderclient.com
- Follow Twitter for updates - twitter.com/thunder_client
- Support: github.com/rangav/thunder-client-support

Story behind Thunder Client

- Read Launch Blog Post on Medium

Usage

- Install the Extension, Click Thunder Client icon on the Action Bar.
- From Sidebar click `New Request` button to test API

Categories

Programming Languages | Snippets | Testing

Resources

Marketplace
Issues
Repository
License
Thunder Client

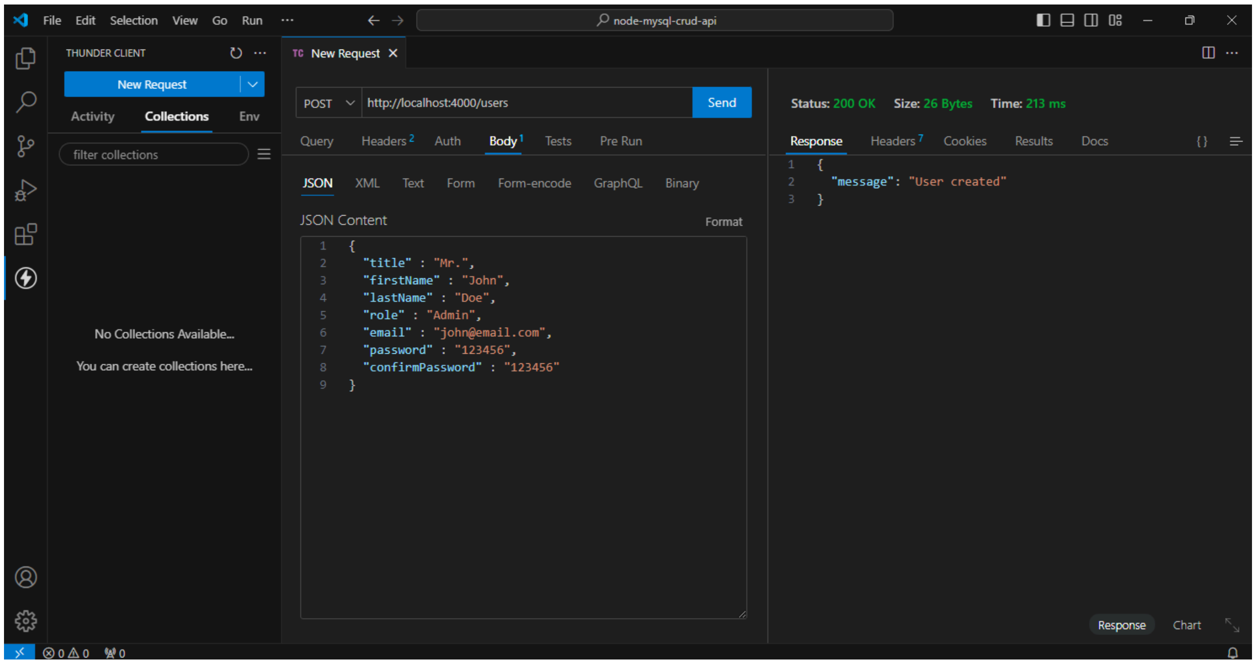
More Info

Published	2021-03-30, 20:59:45
Last released	2024-03-04, 23:51:12

Making Requests in Thunder Client

- Click on the new request button
- You can configure the request url and what type of request you want to send on the page next to the activity/collections/env tab.
- If you are done with the configurations then click the send button.

Creating a new user


File Edit Selection View Go Run ···
node-mysql-crud-api
THUNDER CLIENT
New Request
Activity Collections Env
filter collections
No Collections Available...
You can create collections here...
New Request
Query Headers 2 Auth Body 1 Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content
Format
Response Headers 7 Cookies Results Docs
Response Chart

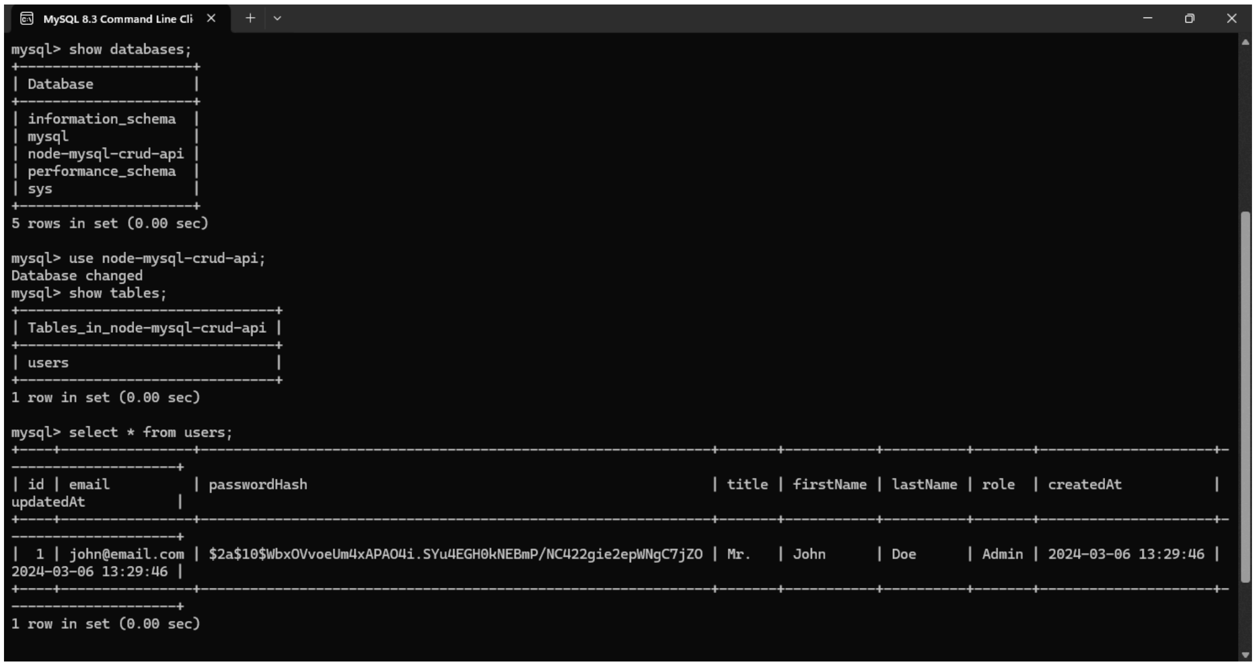
POST http://localhost:4000/users Send

```
{
  "title" : "Mr.",
  "firstName" : "John",
  "lastName" : "Doe",
  "role" : "Admin",
  "email" : "john@email.com",
  "password" : "123456",
  "confirmPassword" : "123456"
}
```

Status: 200 OK Size: 26 Bytes Time: 213 ms

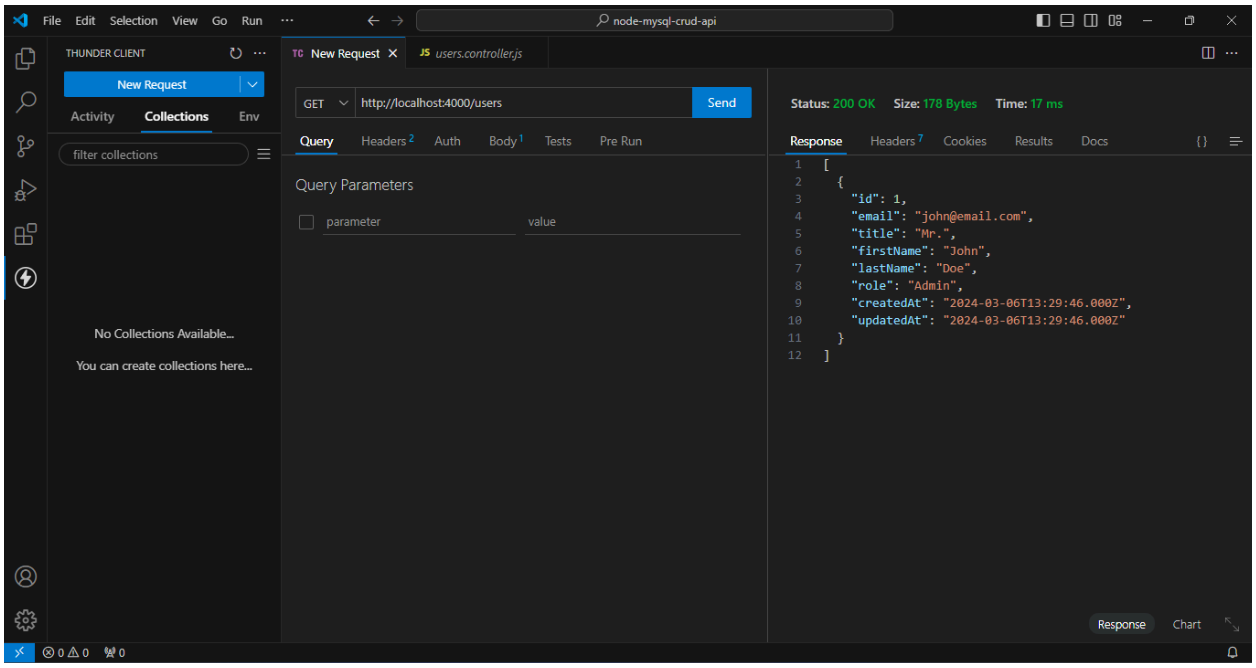
```
{
  "message": "User created"
}
```

Changes reflected on the database after creating a user.



```
mysql> show databases;
+---------------------+
| Database            |
+---------------------+
| information_schema  |
| mysql               |
| node-mysql-crud-api |
| performance_schema  |
| sys                 |
+---------------------+
5 rows in set (0.00 sec)

mysql> use node-mysql-crud-api;
Database changed
mysql> show tables;
+-------------------------------+
| Tables_in_node-mysql-crud-api |
+-------------------------------+
| users                         |
+-------------------------------+
1 row in set (0.00 sec)

mysql> select * from users;
+----+----------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------+-
--------------------+
| id | email          | passwordHash                                                 | title | firstName | lastName | role  | createdAt           |
updatedAt           |
+----+----------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------+-
--------------------+
|  1 | john@email.com | $2a$10$WbxOVvoeUm4xAPAO4i.SYu4EGH0kNEBmP/NC422gie2epWNgC7jZO | Mr.   | John      | Doe      | Admin | 2024-03-06 13:29:46 |
2024-03-06 13:29:46 |
+----+----------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------+-
--------------------+
1 row in set (0.00 sec)
```

Get all users

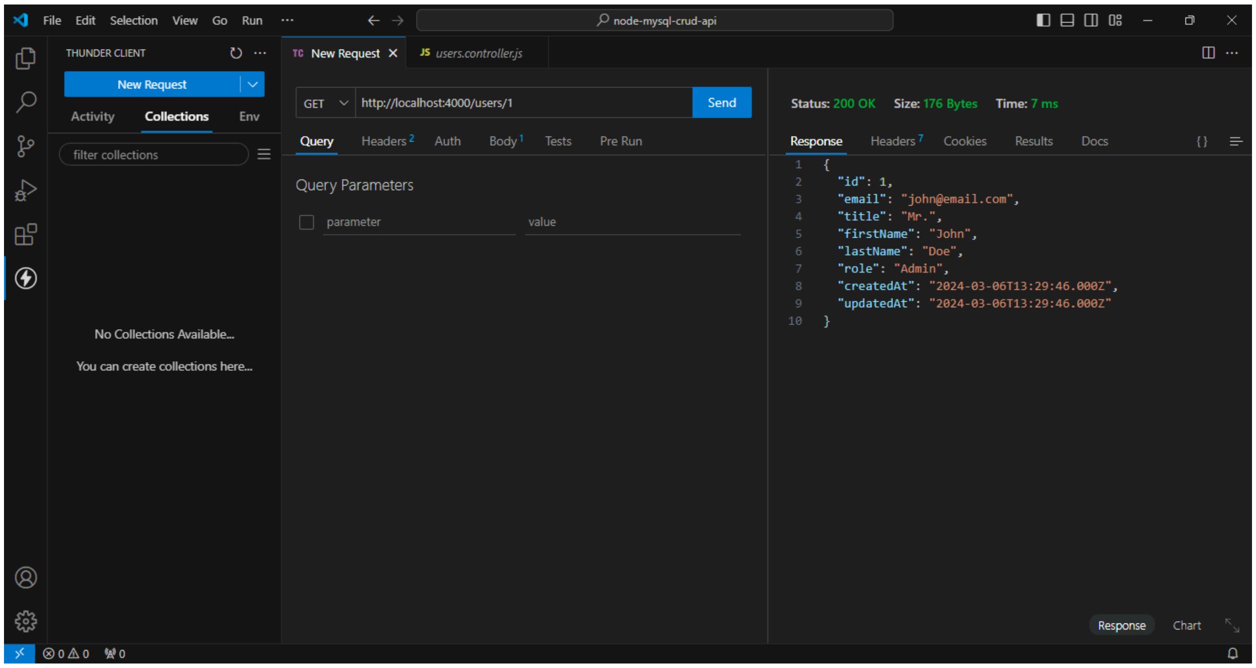


GET http://localhost:4000/users Send

Status: 200 OK Size: 178 Bytes Time: 17 ms

```
[
  {
    "id": 1,
    "email": "john@email.com",
    "title": "Mr.",
    "firstName": "John",
    "lastName": "Doe",
    "role": "Admin",
    "createdAt": "2024-03-06T13:29:46.000Z",
    "updatedAt": "2024-03-06T13:29:46.000Z"
  }
]
```

Get a user by ID

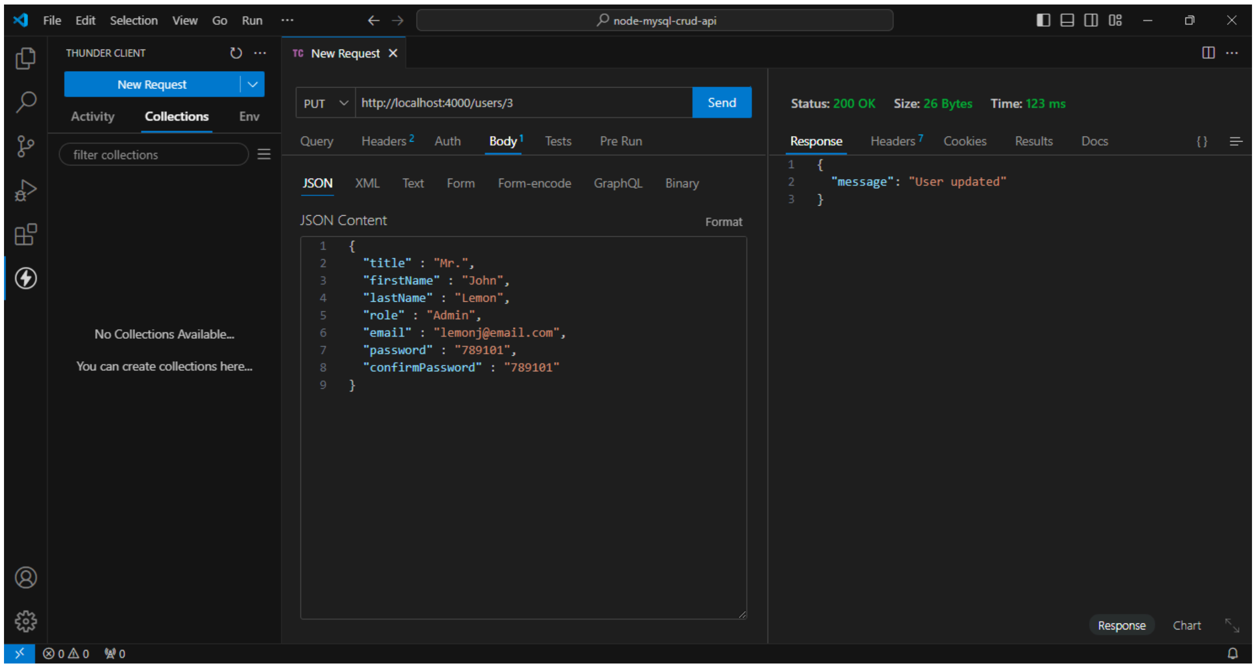


GET http://localhost:4000/users/1 Send

Status: 200 OK Size: 176 Bytes Time: 7 ms

```
{
  "id": 1,
  "email": "john@email.com",
  "title": "Mr.",
  "firstName": "John",
  "lastName": "Doe",
  "role": "Admin",
  "createdAt": "2024-03-06T13:29:46.000Z",
  "updatedAt": "2024-03-06T13:29:46.000Z"
}
```

Update a user by ID



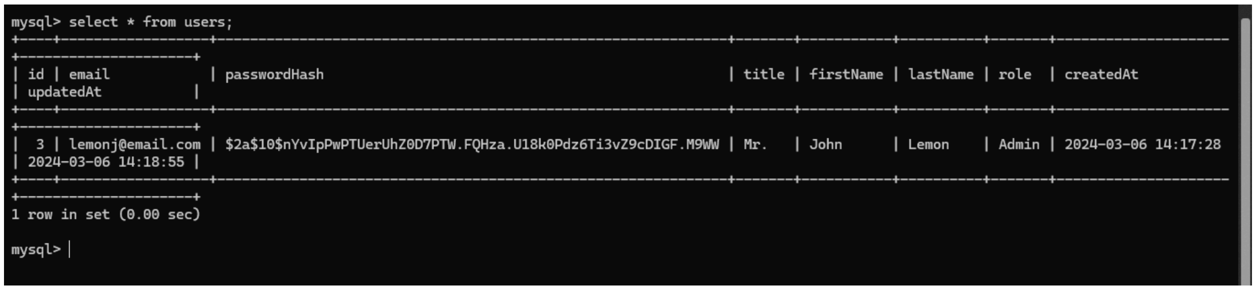
PUT http://localhost:4000/users/3

```
{
  "title" : "Mr.",
  "firstName" : "John",
  "lastName" : "Lemon",
  "role" : "Admin",
  "email" : "lemonj@email.com",
  "password" : "789101",
  "confirmPassword" : "789101"
}
```

Status: 200 OK Size: 26 Bytes Time: 123 ms

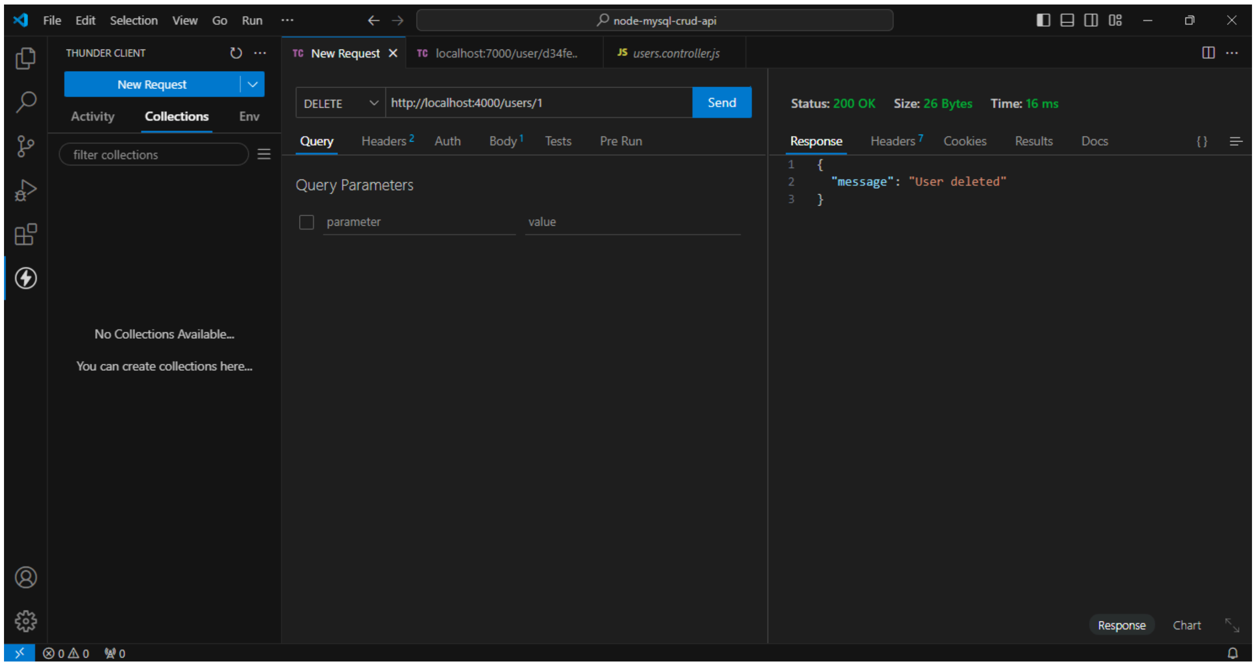
```
{
  "message": "User updated"
}
```

Changes reflected on the database after updating a user.



```
mysql> select * from users;
+----+------------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------
+---------------------+
| id | email            | passwordHash                                                 | title | firstName | lastName | role  | createdAt
| updatedAt           |
+----+------------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------
+---------------------+
|  3 | lemonj@email.com | $2a$10$nYvIpPwPTUerUhZ0D7PTW.FQHza.U18k0Pdz6Ti3vZ9cDIGF.M9WW | Mr.   | John      | Lemon    | Admin | 2024-03-06 14:17:28
| 2024-03-06 14:18:55 |
+----+------------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------
+---------------------+
1 row in set (0.00 sec)

mysql> |
```

Delete a user by ID

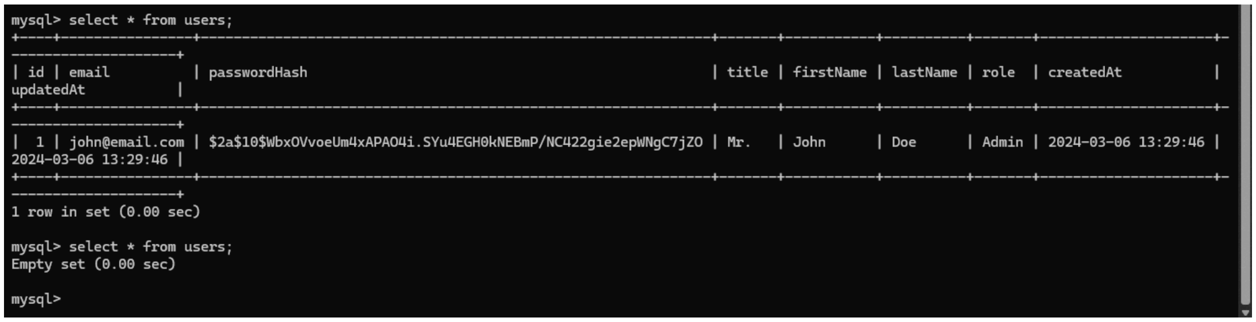


DELETE http://localhost:4000/users/1

Status: 200 OK Size: 26 Bytes Time: 16 ms

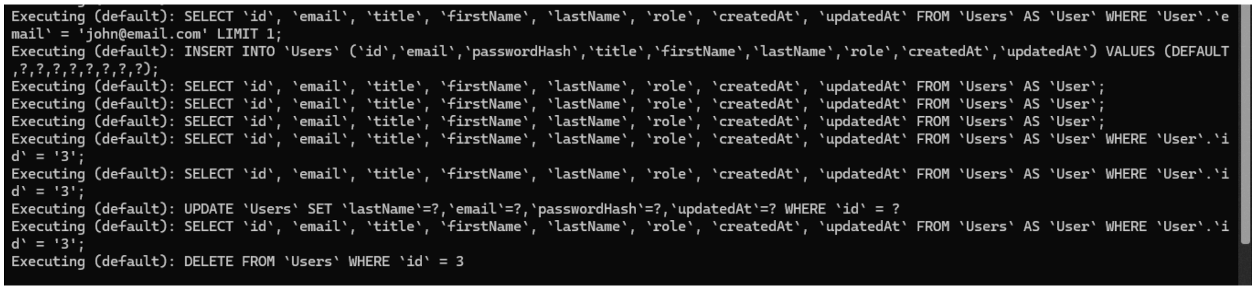
```
{
  "message": "User deleted"
}
```

Changes reflected on the database after deleting a user.



```
mysql> select * from users;
+----+----------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------+-
---------------------+
| id | email          | passwordHash                                                 | title | firstName | lastName | role  | createdAt           |
updatedAt           |
+----+----------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------+-
---------------------+
|  1 | john@email.com | $2a$10$WbxOVvoeUm4xAPAO4i.SYu4EGH0kNEBmP/NC422gie2epWNgC7jZO | Mr.   | John      | Doe      | Admin | 2024-03-06 13:29:46 |
2024-03-06 13:29:46 |
+----+----------------+--------------------------------------------------------------+-------+-----------+----------+-------+---------------------+-
---------------------+
1 row in set (0.00 sec)

mysql> select * from users;
Empty set (0.00 sec)

mysql>
```

Terminal Logs after performing CRUD operations.



```
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User` WHERE `User`.`e
mail` = 'john@email.com' LIMIT 1;
Executing (default): INSERT INTO `Users` (`id`,`email`,`passwordHash`,`title`,`firstName`,`lastName`,`role`,`createdAt`,`updatedAt`) VALUES (DEFAULT
,?,?,?,?,?,?,?,?);
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User` WHERE `User`.`i
d` = '3';
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User` WHERE `User`.`i
d` = '3';
Executing (default): UPDATE `Users` SET `lastName`=?,`email`=?,`passwordHash`=?,`updatedAt`=? WHERE `id` = ?
Executing (default): SELECT `id`, `email`, `title`, `firstName`, `lastName`, `role`, `createdAt`, `updatedAt` FROM `Users` AS `User` WHERE `User`.`i
d` = '3';
Executing (default): DELETE FROM `Users` WHERE `id` = 3
```