RESTful APIs

The first code snippet configures an Express.js server with CRUD endpoints for user management, along with a global error handler. It assigns unique ports to each worker process in a clustered environment, with the server listening on the designated port. The second snippet demonstrates clustering in Node.js using the `cluster` module, where multiple worker processes are forked based on available CPU cores. Each worker forwards incoming requests to other workers in a round-robin manner, enhancing server performance by leveraging multiple CPU cores efficiently.

App.js

const express = require('express');

const cluster = require('cluster');

const dotenv = require('dotenv')

const { get\_oneuser, get\_allusers, create\_user, update\_user, delete\_user } = require('./controllers');

dotenv.config()

const app = express()

const port = process.env.PORT || 4000

app.use(express.json())

app.get('/api/users', get\_allusers);

app.get('/api/users/:id', get\_oneuser);

app.post('/api/users', create\_user);

app.put('/api/users/:id', update\_user);

app.delete('/api/users/:id', delete\_user);

*//global error handler*

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

*if* (req.err) {

        res.status(500).send({ error: req.err })

    }

*else* {

        res.status(404).send({ error: "404 page not found !" })

    }

})

*if* (cluster.isWorker) {

    app.set('port', port + cluster.worker.id);

}

const server = app.listen(app.get('port') || port, () => {

    console.log(`Server listening on port ${server.address().port}`);

});

Server.js

const cluster = require("cluster");

const http = require("http");

const os = require("os");

const express = require("express");

const dotenv = require('dotenv')

cluster.setupPrimary({

    exec: "./app.js"

});

dotenv.config()

const app = express();

const port = process.env.PORT || 4000;

const no\_of\_cpus = os.cpus().length;

const workers = [];

*for* (let i = 0; i < no\_of\_cpus; i++) {

    workers[i] = cluster.fork();

}

cluster.on("message", (worker, data) => {

*for* (let i = 0; i < workers.length; i++) {

*if* ((worker.id - 1) != i) {

            workers[i].send(data);

        }

    }

});

let lastWorker = 0;

*//middlewares*

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

    lastWorker = (lastWorker + 1) % no\_of\_cpus;

    req.currentWorker = workers[lastWorker];

    console.log(`forawrded to ${port + req.currentWorker.id}`);

    next();

});

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

    const request = http.request({

        host: '127.0.0.1',

        port: port + req.currentWorker.id,

        method: req.method,

        path: req.url,

        headers: req.headers

    }, response => {

        res.writeHead(response.statusCode, response.headers);

        response.pipe(res);

    });

    req.pipe(request);

});

app.listen(port, () => {

    console.log("Server running on multiple instances");

});

Controller.js

const cluster = require("cluster")

const { v4: uuidv4 } = require('uuid');

const validator = require('validator');

const users = []

process.on("message", (message) => {

*if* (message.action == "post") {

        users.push(message.body)

    }

*else* *if* (message.action == "put") {

        users[message.useri] = message.body

    }

*else* *if* (message.action == "delete") {

        users.splice(message.useri, 1)

    }

})

const get\_allusers = async (req, res) => {

    res.status(200).send(users);

}

const get\_oneuser = async (req, res) => {

    const uid = req.params.id

*if* (!validator.isUUID(uid)) {

        res.status(400).send("invalid id")

*return*

    }

    const useri = users.findIndex(user => user.id === uid)

*if* (users[useri]) {

        res.status(200).send(users[useri])

    }

*else* {

        res.status(404).send("user not found")

    }

}

const create\_user = async (req, res) => {

    const uid = uuidv4()

    const user\_data = req.body

*if* (!user\_data.username || !user\_data.age || !user\_data.hobbies) {

        res.status(400).send("invalid data")

*return*

    }

    let newuser = {

        id: uid,

        username: user\_data.username,

        age: user\_data.age,

        hobbies: user\_data.hobbies

    }

    users.push(newuser)

*if* (cluster.worker) {

        process.send({

            action: "post",

            body: newuser

        })

    }

    res.status(201).send(newuser)

}

const update\_user = async (req, res) => {

    const uid = req.params.id

*if* (!validator.isUUID(uid)) {

        res.status(400).send("invalid id")

*return*

    }

    const useri = users.findIndex((user) => user.id === uid)

*if* (useri !== -1) {

        var user\_data = req.body

        var newName = user\_data.username || users[useri].username

        var newAge = user\_data.age || users[useri].age

        var newHobbies = user\_data.hobbies || users[useri].hobbies

        users[useri] = {

            id: uid,

            username: newName,

            age: newAge,

            hobbies: newHobbies

        }

*if* (cluster.worker) {

            process.send({

                action: "put",

                useri: useri,

                body: users[useri]

            })

        }

        res.status(200).send(users[useri])

    }

*else* {

        res.status(404).send("user not found")

    }

}

const delete\_user = async (req, res) => {

    const uid = req.params.id

*if* (!validator.isUUID(uid)) {

        res.status(400).send("invalid id")

*return*

    }

    const useri = users.findIndex((user) => user.id === uid)

*if* (useri !== -1) {

        users.splice(useri, 1)

*if* (cluster.worker) {

            process.send({

                action: "delete",

                useri: useri

            })

        }

        res.status(204).send()

    } *else* {

        res.status(404).send("user not found")

    }

}

module.exports = { get\_oneuser, get\_allusers, create\_user, update\_user, delete\_user }

"dependencies": {

    "dotenv": "^16.0.3",

    "express": "^4.18.2",

    "uuid": "^9.0.0",

    "validator": "^13.9.0",

    "nodemon": "^2.0.22"