Aula 4 - Organizando o código

Docupedia Export

Author:Ferro Alisson (CtP/ETS)
Date:21-Aug-2023 13:14

Table of Contents

Nas aulas anteriores estávamos vendo como montar uma API,

1 Separando o *Controller*

Até o momento as nossas rotas estava com muita lógica, essa não é a intenção de uma rota, a rota deve receber a requisição e tratar dessa requisição, vamos ver nosso arquivo *routes/person.js*

```
const express = require('express');
const router = express.Router();
const Person = require('../models/Person');

router
    .get('/api/person/first', (req, res) => {
        console.log('Hello from API');
    }
    .get('/api/person', async (req, res) => {
        try {
            const people = await Person.find();
            return res.status(200).send({ data: people });
        } catch (error) {
            return res.status(500).send({ error: error });
        }
    })
    .get('/api/person/:id', async (req, res) => {
        const { id } = req.params;
        if(!id)
        return res.status(400).send({ message: "No id provider" })

        try {
            const person = await Person.findById(id);
            return res.status(200).json(person);
        } catch (error) {
            res.status(500).json({ error: error })
        }
    })
    .post('/api/person', async (req, res) => {
        const { name, lastname, salary } = req.body;
        if(!name || !lastname || !salary)
            return res.status(400).send({ message: "Dados inválidos" })
```

```
        const person = {
            name: name,
            lastname: lastname,
            salary: salary
        }

        try {
            const p = await Person.create(person);
            return res.status(201).send({ message: "Pessoa inserida com sucesso", body: p });
        } catch (error) {
            return res.status(500).send({ error: error });
        }
    })
    .patch('/api/person/:id', async (req, res) => {
        const { id } = req.params;
        if(!id)
            return res.status(400).send({ message: "No id provider" })

        const person = req.body;
        if(!person.salary)
            return res.status(400).send({ message: "No salary provider" })

        try {
            const newPerson = await Person.findByIdAndUpdate(
                id,
                { salary: person.salary }
            );
            return res.status(200).send(newPerson);
        } catch (error) {
            return res.status(500).send({ error: error });
        }
    })
    .delete('/api/person/:id', async (req, res) => {
        const { id } = req.params;
        if(!id)
            return res.status(400).send({ message: "No id provider" });

        try {
            await Person.findByIdAndRemove(id);
```

```
            return res.status(200).send({ message: "Person deleted successfully" })
        } catch (error) {
            console.log(error);
            return res.status(500).send({ message: "Something failled"})
        }
    })

module.exports = router;
```

O primeiro passo aqui é excluir a rota */api/person/first* que foi necessária para demonstrar que a API estava funcionando
O segundo passo é criar um arquivo em *controller/PersonController.js* e la vamos colocar toda a lógica em uma classe PersonController

```
const Person = require('../models/Person');

class PersonController {
    static async create(req, res){
        const { name, lastname, salary } = req.body;
        if(!name || !lastname || !salary)
            return res.status(400).send({ message: "Dados inválidos" })

        const person = {
            name: name,
            lastname: lastname,
            salary: salary
        }

        try {
            const p = await Person.create(person);
            return res.status(201).send({ message: "Pessoa inserida com sucesso", body: p });
        } catch (error) {
            return res.status(500).send({ error: error });
        }
    };

    static async getAllPeople(req, res){
        try {
            const people = await Person.find();
```

```
            return res.status(200).send({ data: people });
        } catch (error) {
            return res.status(500).send({ error: error });
        }
    }

    static async getById(req, res){
        const { id } = req.params;
        if(!id)
        return res.status(400).send({ message: "No id provider" })

        try {
            const person = await Person.findById(id);
            return res.status(200).json(person);
        } catch (error) {
            res.status(500).json({ error: error })
        }
    }
    static async updateById(req,res){
        const { id } = req.params;
        if(!id)
            return res.status(400).send({ message: "No id provider" })

        const person = req.body;
        if(!person.salary)
            return res.status(400).send({ message: "No salary provider" })

        try {
            const newPerson = await Person.findByIdAndUpdate(
                id,
                { salary: person.salary }
            );
            return res.status(200).send(newPerson);
        } catch (error) {
            return res.status(500).send({ error: error });
        }
    };
    static async deleteById(req, res){
        const { id } = req.params;
```

```
        if(!id)
            return res.status(400).send({ message: "No id provider" });

        try {
            await Person.findByIdAndRemove(id);
            return res.status(200).send({ message: "Person deleted successfully" })
        } catch (error) {
            console.log(error);
            return res.status(500).send({ message: "Something failled"})
        }
    }
}

module.exports = PersonController;
```

E agora vamos para o *routes/person.js*

```
const express = require('express');
const PersonController = require('../controller/PersonController');
const router = express.Router();

router
    .get('/api/person', PersonController.getAllPeople)
    .get('/api/person/:id', PersonController.getById)
    .post('/api/person', PersonController.create)
    .patch('/api/person/:id', PersonController.updateById)
    .delete('/api/person/:id', PersonController.deleteById)

module.exports = router;
```

Agora já esta mais limpo, mas ainda podemos realizar uma ultima melhoria no routes, vamos atribuir no startup da API as rotas com o inicio pré-definido.

2 Abstraindo as rotas

Vamos criar um arquivo em *startup/routes.js* e colocar o código

```
const express = require('express');
const person = require('../routes/person');

module.exports = function(app) {
    app.use(express.json());
    app.use('/api/person', person);
}
```

Por fim, precisamos chamar a startup no *index.js* para poder ser usado

```
const express = require('express');
const app = express();
const routes = require('./routes');

require('./startup/db')();
require('./startup/routes')(app);

const port = 8080;

routes(app);

const server = app.listen(port, () => console.log(`Listening on port ${port}`));

module.exports = server;
```

Como foram muitas modificações realizadas, vamos testar novamente todas as rotas no Postman

Desafio: Na API de produtos, realize estes passos para organizar melhor o código