Desenvolvimento Backend

Aula 05 - Roteamento, Middleware e Validação

Material Didático do Instituto Metrópole Digital - IMD

Termo de uso

Apresentação

Nesta aula, você verá como manipular rotas no Express, como validar dados de entrada e ainda o conceito de *middleware*.

Objetivos

- Entender como manipular rotas no Express;
- Aprender a validar dados de entrada;
- Conhecer o que são e como criar *middlewares* no Express.

Manipulando rotas com Express

Link do video da aula: https://youtu.be/bluQCwXP_TY

Nesta aula, você verá como controlar rotas, como validar dados e também conhecere o conceito de *middlewares* do Express.

Router

O Express fornece um facilitador para organizar as rotas em diferentes arquivos, deixando o código mais organizado. Para usar esse conceito, crie um arquivo novo chamado `usuarios.rota.js` na pasta `/rotas` do seu projeto com o seguinte conteúdo:

```
const express = require('express')
const router = express.Router()
const { v4: uuidv4 } = require('uuid')

const usuarios = {}

router.get('/:id', (req, res) => {
    res.json({usuarios: usuarios[req.params.id]})
})

router.put('/', (req, res) => {
    const id = req.query.id
    if (id && usuarios[id]){
        const usuario = req.body
        usuario.id = id
        usuarios[id] = usuario
        res.json({msg: "Usuário atualizado com sucesso!"})
    }else{
```

```
        res.status(400).json({msg: "Usuário não encontrado!"})
    }
})

router.delete('/', (req, res) => {
    const id = req.query.id
    if (id && usuarios[id]){
        delete usuarios[id]
        res.json({msg: "Usuário deletado com sucesso!"})
    }else{
        res.status(400).json({msg: "Usuário não encontrado!"})
    }
})

router.post('/', (req, res) => {
    const usuario = req.body
    const idUsuario = uuidv4()
    usuario.id = idUsuario
    usuarios[idUsuario] = usuario
    res.json({msg: "Usuário adicionado com sucesso!"})
})

router.get('/', (req, res) => {
    res.json({usuarios: Object.values(usuarios)})
})

module.exports = router
```

Perceba que em vez de criar as rotas diretamente na instância da aplicação (`app`), estamos criando no `express.Router()`. Com isso, é possível dividir e organizar melhor o código.

Uma vez criado o arquivo de rotas, altere o arquivo principal da aplicação (index.js) para incluir o arquivo de rotas e indicar o *path* base.

```
const express = require('express')
const rotaUsuario = require('./rotas/usuario.rota')

const app = express()
app.use(express.json())

app.use('/usuarios', rotaUsuario)

app.get('/', (req, res) => {
```

```
    res.json({msg: "Hello from Express!"})
})

app.listen(8080, () => {
    console.log('Servidor pronto na porta 8080')
})
```

Seguindo no nosso exemplo, vamos criar a API de um blog. Para isso, vamos agora criar as operações relativas às postagens do blog replicando a solução dadas as rotas do `/usuarios`.

Crie então o arquivo `post.rota.js` na pasta `/rotas` com o seguinte conteúdo:

```
const express = require('express')
const router = express.Router()
const { v4: uuidv4 } = require('uuid')

const posts = {}

router.get('/:id', (req, res) => {
    res.json({posts: posts[req.params.id]})
})

router.put('/', (req, res) => {
    const id = req.query.id
    if (id && posts[id]){
        const post = req.body
        post.id = id
        posts[id] = post
        res.json({msg: "Post atualizado com sucesso!"})
    }else{
        res.status(400).json({msg: "Post não encontrado!"})
    }
})

router.delete('/', (req, res) => {
    const id = req.query.id
    if (id && posts[id]){
        delete posts[id]
        res.json({msg: "Post deletado com sucesso!"})
    }else{
        res.status(400).json({msg: "Post não encontrado!"})
    }
})
```

```
router.post('/', (req, res) => {
    const post = req.body
    const idPost = uuidv4()
    post.id = idPost
    posts[idPost] = post
    res.json({msg: "Post adicionado com sucesso!"})
})

router.get('/', (req, res) => {
    res.json({posts: Object.values(posts)})
})

module.exports = router
```

Altere agora o arquivo principal (index.js) para incluir a nova rota:

```
const express = require('express')
const rotaUsuario = require('./rotas/usuario.rota')
const rotaPost = require('./rotas/posts.rota')

const app = express()
app.use(express.json())

app.use('/usuarios', rotaUsuario)
app.use('/posts', rotaPost)

app.get('/', (req, res) => {
    res.json({msg: "Hello from Express!"})
})

app.listen(8080, () => {
    console.log('Servidor pronto na porta 8080')
})
```

> Para a explicação detalhada, não deixe de assistir a videoaula.

Validando dados

Link do video da aula: https://youtu.be/9rYYumSAfDU

Agora que já temos o arquivo de rotas organizado, vamos ver como validar os dados de entrada.

Para a validação, utilizaremos uma biblioteca muito conhecida no Javascript, a Ajv.

Instalação

Para instalar o Ajv, execute:

```
$ npm install ajv --save
```

Para validar dados de entrada mais complexos, execute:

```
$ npm install ajv-formats --save
```

Feita a instalação, precisamos agora criar os *schemas* para controlar o formato de entrada válido.

Criando *schemas*

Os *schemas* são definições das estruturas esperadas para determinado dado. O Ajv utiliza um padrão internacional formalizado na RFC 8927 para as definições dos *schemas*.

Acesse a documentação da biblioteca Ajv através deste link para acesso a vários exemplos de definições.

Para o nosso exemplo de usuário, crie um arquivo chamado `usuario.schema.js` na pasta `/schemas` com o seguinte conteúdo:

```
module.exports = {
    type: "object",
    properties: {
        email: {type: "string", format: "email"},
        senha: {type: "string"}
    },
    required: ["email", "senha"],
    additionalProperties: false
}
```

Feito o *schema*, podemos adicionar a validação no arquivo `usuario.rota.js`:

```
const Ajv = require('ajv')
```

```
const ajv = new Ajv()
const addFormats = require("ajv-formats")
addFormats(ajv)

router.post('/', (req, res) => {
    const usuario = req.body

        const validate = ajv.compile(usuarioSchema)
        const valid = validate(usuario)

        if (valid){
          const idUsuario = uuidv4()
      usuario.id = idUsuario
      usuarios[idUsuario] = usuario
      res.json({msg: "Usuário adicionado com sucesso!"})
        }else{
           res.status(400).json({msg: "Dados inválidos", errors:
validate.errors})
        }

})
```

Para explicação detalhada desse exemplo, não deixe de assistir a videoaula.

Criando Middleware

Link do video da aula: https://youtu.be/mIRjY6UHY8g

Nesta aula, você verá como trabalhar com middlewares. Mas o que é um *middleware* no Express?

O *middleware* é uma ação que pode ser anexada para executar antes da chamada final de uma rota.

O *middleware* pode, por exemplo, validar os dados de entrada para garantir que a ação só seja executada com dados válidos. Dessa forma, o código fica mais desacoplado, evitando repetir a lógica de validação nas rotas.

Criando o *middleware*

Para criar o middleware de validação dos usuários, crie o arquivo chamado `validarUsuario.middleware.js` na pasta `/middleware` com o seguinte conteúdo:

```
const Ajv = require('ajv')
const ajv = new Ajv()
const addFormats = require("ajv-formats")
const usuarioSchema = require('../schema/usuario.schema')

addFormats(ajv)

function validarUsuario(req, res, next){
    const usuario = req.body
    const validate = ajv.compile(usuarioSchema)
    const valid = validate(usuario)
    if (valid){
        next()
    }else{
        res.status(400).json({msg: "Dados inválidos", erros:
validate.errors})
    }
}

module.exports = validarUsuario
```

Perceba que o *middleware* nada mais é do que uma função com os parâmetros `req`, `res`, e `next`. No exemplo acima, a requisição é interrompida caso o dado seja inválido e segue adiante caso esteja tudo bem.

Adicionando o middleware à rota

Agora que o *middleware* está pronto, falta indicar nas definições das rotas que queremos utilizá-lo. Para isso, edite o arquivo de rotas do usuário, adicionando o *middleware* para as rotas POST e PUT, conforme o exemplo abaixo:

```
const express = require('express')
const router = express.Router()
const { v4: uuidv4 } = require('uuid')
const usuarioMid =
require('../middleware/validarUsuario.middleware')

const usuarios = {}

router.post('/', usuarioMid)
router.put('/', usuarioMid)
```

Não deixe de conferir a videoaula para uma explicação detalhada.

Gerenciando postagens

Link do video da aula: https://youtu.be/Y2XBMhnr6oo

Nesta videoaula, vamos replicar a solução de validação de usuários feita agora para as operações de postagens.

Schema da Postagem

Cada post no blog, inicialmente, terá apenas um título e um texto. O título poderá ser uma string de 5 a 100 caracteres e o texto deve ser um campo livre. Para isso, podemos utilizar o seguinte *schema* do Ajv.

```
module.exports = {
    type: "object",
    properties: {
        titulo: {type: "string", maxLength: 100, minLength: 5},
        texto: {type: "string"}
    },
    required: ["titulo", "texto"],
    additionalProperties: false
}
```

O Ajv utiliza um padrão para definição do *schema* que segue a RFC 8927

Middleware

Definido o *schema* é hora de criar o *middleware* de validação de postagens. Para isso, crie um arquivo chamado `validarPost.middleware.js` na pasta `/middleware`, com o conteúdo a seguir. Acompanhe a videoaula para explicação detalhada do código.

```
const Ajv = require('ajv')
const ajv = new Ajv()
const postSchema = require('../schema/post.schema')

function validarPost(req, res, next){
    const post = req.body
    const validate = ajv.compile(postSchema)
    const valid = validate(post)
    if (valid){
```

```
        next()
    }else{
        res.status(400).json({msg: "Dados inválidos", erros:
validate.errors})
    }
}

module.exports = validarPost
```

Alteração na rota

Com o *middleware* criado, devemos adicioná-lo nas rotas POST e PUT, conforme o trecho de código abaixo:

```
const express = require('express')
const router = express.Router()
const { v4: uuidv4 } = require('uuid')
const postMid = require('../middleware/validarPost.middleware')

const posts = {}

router.post('/', postMid)
router.put('/', postMid)
```

O código acima foi extraído do arquivo `posts.rota.js` e tem o objetivo de adicionar os *middlewares* de validação nas rotas de adicionar (POST) e editar (PUT) um post.

Resumo

Nesta aula, você viu como manipular rotas para deixar seu programa organizado e bem dividido. Além disso, viu como utilizar a biblioteca Ajv para criar validações poderosas de dados de entrada, garantindo que seu sistema lide apenas com informações em um padrão preestabelecido. Por fim, conheceu o conceito de *middlewares* e como ele ajuda a separar as responsabilidades dos arquivos do seu projeto.