Desenvolvimento Backend

Aula 08 - Upload de arquivos

Material Didático do Instituto Metrópole Digital - IMD

Termo de uso

Apresentação

Nesta aula, você vai aprender como fazer upload de arquivos ao servidor.

Objetivos

- Aprender a realizar upload de arquivos;
- Conhecer sobre o armazenamento dos arquivos no banco de dados;
- Conhecer sobre o controle de upload de arquivos.

Link do video da aula: https://youtu.be/4kUbAGoaw_4

Para realizar esse processo, será utilizada uma biblioteca para facilitar o trabalho e a manipulação de arquivos. É uma biblioteca open source e pode ser encontrada no GitHub .

Instalando o multer

Para instalar o multer, execute o seguinte comando no terminal:

```
    npm install --save multer
```

Utilizando o multer

Para utilizar o multer, o arquivo *posts.rota.js* deverá ser alterado. Inicialmente, adicione as seguintes importações no início do arquivo:

```
    var  multer   =  require ( 'multer' )
    var  upload  =  multer ( {  dest : 'uploads /'  } )
```

Após isso, ainda no arquivo *posts.rota.js*, crie uma nova rota do tipo *post* para adicionar essa postagem. A rota deverá ficar como abaixo:

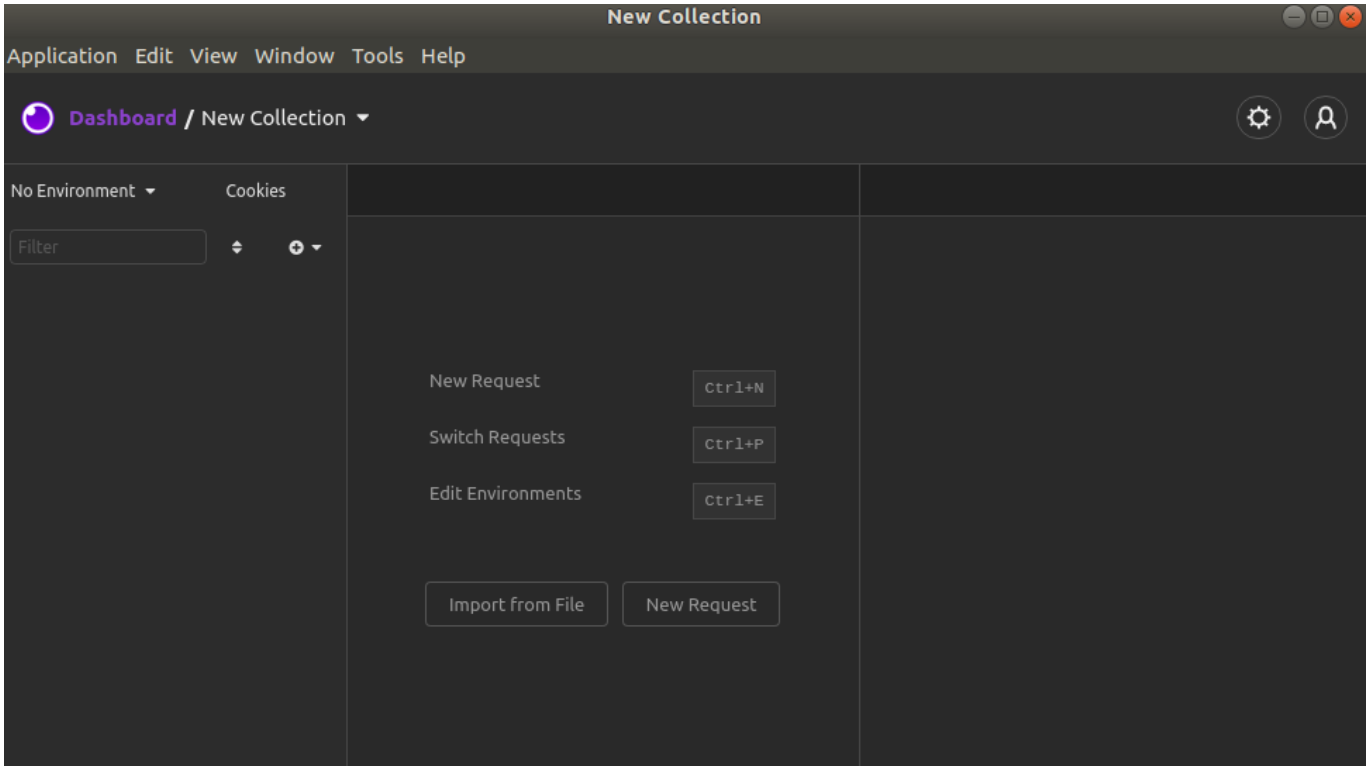
```
    router.post('/upload', upload . single ( 'foto' ), async (req,
res) => {
        console.log(req.file)
        res.json({msg: 'Arquivo enviado com sucesso'})
    })
```

O upload já deve estar funcionando, agora chegou o momento de testes.

Testando o upload de arquivos

Para testar, será utilizada a ferramenta *Insominia*, que simula requisições *http*.

Figura 1 - Ferramenta Insominia



Acompanhe a aula, faça o download e testes os necessários. O download da ferramenta pode ser feito AQUI.

Adaptando banco para armazenar arquivo

Link do video da aula: https://youtu.be/WRHKHzKOT1c

Agora, depois de habilitar o *upload* de arquivos, é preciso preparar o banco de dados para armazenar essa nova informação. É possível fazer isso de diferentes maneiras: você pode armazenar o arquivo diretamente no banco ou armazenar o arquivo no sistema de arquivos e no banco colocar apenas uma referência para esse arquivo.

Dentre todas as vantagens, nesta aula será abordada a segunda forma de armazenamento.

Adicionando um novo campo ao modelo

Para armazenar essa referência à imagem, altere o arquivo de modelo *post.js* para

adicionar um novo campo. Esse arquivo pode ser encontrado na pasta *models*. Após a alteração o arquivo deverá ficar da seguinte forma:

```
    'use strict';
    const {
        Model, INTEGER
    } = require('sequelize');
    module.exports = (sequelize, DataTypes) => {
        class Post extends Model {

            static associate(models) {
                Post.belongsTo(models.Usuario, {foreignKey:
'userId'})
            }

        };
        Post.init({
            titulo: DataTypes.STRING,
            texto: DataTypes.STRING,
            userId: DataTypes.INTEGER,
            foto: DataTypes.STRING
        }, {
            sequelize,
            modelName: 'Post',
        });
        return Post;
    };
```

Realizando a migração

Após alterar o arquivo de modelo, o campo ainda não foi criado no banco de dados especificamente. Para fazer isso, é necessário fazer a migração. Para isso, rode no terminal o seguinte comando:

```
    npx sequelize-cli migration:generate --name add-foto-post
```

Esse comando irá gerar uma arquivo, no qual serão feitas as implementações necessárias. Acompanhe a aula e no final o seu arquivo deverá conter o código abaixo:

```
'use strict';

module.exports = {
```

```
  up: async (queryInterface, Sequelize) => {
    return queryInterface.addColumn('Posts', 'foto', {
      type: Sequelize.STRING
    })
  },

  down: async (queryInterface, Sequelize) => {
    return queryInterface.removeColumn('Posts', 'foto')
  }
};
```

Finalizado o código, rode a migração. Para isso, execute no terminal o comando:

```
npx sequelize db:migrate
```

Melhorando controle do upload

Link do video da aula: https://youtu.be/UL6H4NSYiu0

Até o momento o upload está sendo feito, mas ainda não é possível controlá-lo muito bem. Por exemplo, o arquivo de upload gera um nome aleatório e sem extensão. Para melhorar isso, acompanhe a aula e realize as alterações necessárias.

Utilizando o DiskStorage

O mecanismo de armazenamento em disco ajudará a controlar o local onde será guardado o arquivo. Para isso, é necessário adicionar as seguintes instruções no código do arquivo *posts.rota.js*.

```
    var storage = multer.diskStorage({
            destination: function (req, file, cb) {
                cb(null, 'public/uploads')
            },
            filename: function (req, file, cb) {
                 cb(null, file.fieldname + "-" + Date.now() +
path.extname(file.originalname))
            }
    })
```

Criando o filtro de extensões

Seguindo os passos anteriores, o nome do arquivo e o destino já estão sendo controlados. Porém, o *upload* ainda aceita todos os tipos de arquivo e isso não é interessante para a aplicação.

Para mudar isso, acompanhe a aula e crie, ainda em *posts.rota.js*, a seguinte função:

```
    const fileFilter = (req, file, cb) => {
            const extensoes = /jpeg|jpg/i
            if (extensoes.test(path.extname(file.originalname))){
                    cb(null, true)
            }else{
                    return cb('Arquivo não suportado. Apenas jpg e
jpeg são suportados.')
            }
    }
```

Após isso, altere a inicialização do multer para efetivar as mudanças. Deve ficar dessa maneira:

```
    var upload = multer({ storage: storage, fileFilter: fileFilter
})
```

Permitindo o acesso do usuário ao arquivo

Para finalizar, é interessante permitir que o usuário tenha acesso aos arquivos da pasta *public*. Para fazer isso, acompanhe a aula e adicione o seguinte código ao arquivo *index.js*:

```
    app.use('/static', express.static('public'))
```

Ajustando rota de validação

Link do video da aula: https://youtu.be/h1GOOQEsSqA

Para finalizar sobre o *upload* de arquivos, é essencial que esse *upload* feito seja armazenado no banco de dados para que a aplicação conheça a imagem e registre tudo corretamente.

Armazenando o arquivo no banco

Até o momento o *upload* está sendo feito, mas a aplicação mostra apenas uma mensagem informando o sucesso. Para fazer o armazenamento, acompanhe a aula alterando o arquivo *posts.rota.js* , que deve ficar de acordo com o modelo abaixo:

```
    const express = require('express')
    const router = express.Router()
    const postMid = require('../middleware/validarPost.middleware')
    const { Post, Usuario } = require('../db/models')
    var multer  = require('multer')
    const path = require('path')

    var storage = multer.diskStorage({
            destination: function (req, file, cb) {
                cb(null, 'public/uploads')
            },
            filename: function (req, file, cb) {
                 cb(null, file.fieldname + "-" + Date.now() +
path.extname(file.originalname))
            }
    })

    const fileFilter = (req, file, cb) => {
            const extensoes = /jpeg|jpg/i
            if (extensoes.test(path.extname(file.originalname))){
                    cb(null, true)
            }else{
                    return cb('Arquivo não suportado. Apenas jpg e
jpeg são suportados.')
            }
    }

    var upload = multer({ storage: storage, fileFilter: fileFilter
})

    router.post('/', upload.single('foto'))
    router.post('/', postMid)
    router.put('/', postMid)

    router.get('/', async (req, res) => {
            const posts = await Post.findAll()
            res.json({posts: posts})
    })
```

```
    router.get('/:id', async (req, res) => {
            const post = await Post.findByPk(req.params.id,
                    {include: [{model: Usuario}], raw: true, nest:
true})

            const postProcessado = prepararResultado(post)
            res.json({posts: postProcessado})
    })

    router.post('/:id/upload', upload.single('foto'), async (req,
res) => {
            console.log(req.file)
            const id = req.params.id
            const post = await Post.findByPk(id)
            if (post){
                    post.foto =
`/static/uploads/${req.file.filename}`
                    await post.save()
                    res.json({msg: "Upload realizado com sucesso!"})
            }else{
                    res.status(400).json({msg: "Post não
encontrado!"})
            }
    })

    router.post('/', async (req, res) => {
            const data = req.body
            if (req.file){
                    data.foto =
`/static/uploads/${req.file.filename}`
            }
            const post = await Post.create(data)
            res.json({msg: "Post adicionado com sucesso!"})
    })

    router.delete('/', async (req, res) => {
            const id = req.query.id
            const post = await Post.findByPk(id)
            if (post){
                    await post.destroy()
                    res.json({msg: "Post deletado com sucesso!"})
            }else{
                    res.status(400).json({msg: "Post não
encontrado!"})
```

```
            }
    })

    router.put('/', async (req, res) => {

            const id = req.query.id
            const post = await Post.findByPk(id)

            if (post){
                    post.titulo = req.body.titulo
                    post.texto = req.body.texto
                    await post.save()
                    res.json({msg: "Post atualizado com sucesso!"})
            }else{
                    res.status(400).json({msg: "Post não
encontrado!"})
            }
    })

    function prepararResultado(post){
            const result = Object.assign({}, post)
            if (result.createdAt) delete result.createdAt
            if (result.updatedAt) delete result.updatedAt
            if (result.userId) delete result.userId
            if (result.Usuario){
                    if (result.Usuario.senha) delete
result.Usuario.senha
                    if (result.Usuario.createdAt) delete
result.Usuario.createdAt
                    if (result.Usuario.updatedAt) delete
result.Usuario.updatedAt
            }
            return result
    }

    module.exports = router
```

Validação do *post*

O post pode vir com algum campo em um formato inválido, então, para evitar isso, navegue até o arquivo de *validarPost.middleware.js* e faça as alterações necessárias. Ao final, o código deve ser ficar de acordo com o modelo abaixo:

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

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

    function validarPost(req, res, next){
            const post = req.body
            if (post.userId){
                    post.userId = Number(post.userId)
            }
            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
```

Resumo

Nesta aula, foram vistos conceitos e ferramentas de *upload* de arquivos, em particular a biblioteca *multer*. As lições aprendidas abordaram o *upload* de imagens e o armazenamento desses arquivos no banco de dados.