

Rota chamando o middleware image-upload Antes de chamar o método editUser

```
const router = require("express").Router();
const UserController = require("../controllers/UserController");

// middlewares
const verifyToken = require("../helpers/verify-token");
const { imageUpload } = require("../helpers/image-upload");

router.patch(
  "/edit/:id",
  verifyToken,
  imageUpload.single("image"),
  UserController.editUser
);


module.exports = router;
```

1. `const router = require("express").Router();`: Importa o módulo `Router` do Express para definir as rotas.
2. `const UserController = require("../controllers/UserController");`: Importa o controlador `UserController` de `../controllers/UserController`, que contém a lógica para manipular requisições relacionadas aos usuários.
3. `const verifyToken = require("../helpers/verify-token");`: Importa o middleware `verifyToken` de `../helpers/verify-token`. Este middleware é responsável por verificar se o token de autenticação é válido.
4. `const { imageUpload } = require("../helpers/image-upload");`: Importa a função `imageUpload` de `../helpers/image-upload`. Esta função é um middleware que cuida do upload de imagens.
5. `router.patch("/edit/:id", verifyToken, imageUpload.single("image"), UserController.editUser);`: Define uma rota PATCH para edição de usuário em `/edit/:id`. Utiliza os middlewares `verifyToken` e `imageUpload.single("image")` para autenticar e lidar com o upload de imagem, respectivamente. Quando a rota é acionada, o método `editUser` do `UserController` é chamado para processar a requisição.
6. `module.exports = router;`: Exporta o router para que ele possa ser utilizado em outros arquivos.



`image-upload.js`

javascript

 Copy code

```
const multer = require("multer");
const path = require("path");

const imageStorage = multer.diskStorage({
  destination: function (req, file, cb) {
    let folder = "";

    console.log(req);

    if (req.baseUrl.includes('users')) {
      folder = "users";
    } else if (req.baseUrl.includes('pets')) {
      folder = "pets";
    }
    cb(null, `public/images/${folder}/`);
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  },
});

const imageUpload = multer({
  storage: imageStorage,
  fileFilter(req, file, cb) {
    if (!file.originalname.match(/\.(png|jpg)$/)) {
      // upload only png and jpg format
      return cb(new Error("Por favor, envie apenas png ou jpg!"));
    }
    cb(undefined, true);
  },
});

module.exports = { imageUpload };
```



1. `const multer = require("multer");`: Importa o módulo `multer`, que é uma middleware para manipular dados de formulários multipartes, que geralmente são usados para upload de arquivos.
2. `const path = require("path");`: Importa o módulo `path`, que fornece utilitários para trabalhar com caminhos de arquivos e diretórios.
3. `const imageStorage = multer.diskStorage({ ... });`: Define a configuração de armazenamento para o multer. Esta configuração determina onde os arquivos enviados serão salvos no servidor e como seus nomes serão gerados.
4. `destination`: Uma função que determina o diretório onde os arquivos serão salvos com base na requisição (`req`). Se a URL base incluir `'users'`, os arquivos serão salvos na pasta `'public/images/users/'`; caso contrário, se incluir `'pets'`, serão salvos na pasta `'public/images/pets/'`.
5. `filename`: Uma função que determina o nome do arquivo que será salvo. Neste caso, o nome é composto pelo timestamp atual concatenado com a extensão do arquivo original.
6. `const imageUpload = multer({ ... });`: Configura o middleware multer para usar a configuração de armazenamento definida anteriormente. Também define um filtro de arquivo para permitir apenas arquivos com extensão `.png` ou `.jpg`.
7. `module.exports = { imageUpload };`: Exporta a função `imageUpload` para que possa ser utilizada em outros arquivos.

o que significa essa linha: `imageUpload.single("image")`

A linha `imageUpload.single("image")` é um método do middleware `multer` que especifica que apenas um arquivo com o nome `"image"` será aceito na requisição HTTP. Vamos analisar mais detalhadamente:

- `imageUpload`: É o middleware configurado para lidar com o upload de imagens. Ele foi definido anteriormente como um middleware multer configurado com as opções de armazenamento e filtro de arquivo.
- `.single("image")`: Este método indica que o middleware deve esperar um único arquivo no corpo da requisição com o campo de nome `"image"`. Quando uma requisição é feita para a rota que usa este middleware, o multer processará o upload do arquivo e o armazenará no local especificado na configuração de armazenamento.

Resumindo, essa linha especifica que, ao acessar a rota onde ela é definida, o middleware multer `imageUpload` será acionado para lidar com o upload de um único arquivo com o campo de nome `"image"` na requisição HTTP.

Função cb

```
const multer = require("multer");
const path = require("path");

// Destination to store image
const imageStorage = multer.diskStorage({
  destination: function (req, file, cb) {
    let folder = "";

    if (req.baseUrl.includes('users')) {
      folder = "users";
    } else if (req.baseUrl.includes('pets')) {
      folder = "pets";
    }
    cb(null, `public/images/${folder}/`);
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  },
});

const imageUpload = multer({
  storage: imageStorage,
  fileFilter(req, file, cb) {
    if (!file.originalname.match(/\.(png|jpg)$/)) {
      // upload only png and jpg format
      return cb(new Error("Por favor, envie apenas png ou jpg!"));
    }
    cb(undefined, true);
  },
});

module.exports = { imageUpload };
```

Na página abaixo temos a explicação mais detalhada da função cb

1. ``destination: function (req, file, cb) { ... }``:

- Aqui, ``cb`` é uma função de retorno de chamada fornecida pelo multer. Ela é usada para comunicar o resultado de uma operação assíncrona de volta ao multer.
- Quando a função ``destination`` é chamada, ela determina o diretório de destino onde o arquivo será armazenado com base nos parâmetros recebidos (``req``, ``file``) e executa a função de retorno de chamada ``cb``.
- O primeiro argumento passado para ``cb`` é ``null``, indicando que não ocorreu um erro durante a execução da função.
- O segundo argumento passado para ``cb`` é o caminho para o diretório de destino calculado pela função ``destination``.

2. ``filename: (req, file, cb) => { ... }``:

- Assim como no caso anterior, ``cb`` é uma função de retorno de chamada.
- Quando a função ``filename`` é chamada, ela determina o nome do arquivo que será salvo com base nos parâmetros recebidos (``req``, ``file``) e executa a função de retorno de chamada ``cb``.
- O primeiro argumento passado para ``cb`` é ``null``, indicando que não ocorreu um erro durante a execução da função.
- O segundo argumento passado para ``cb`` é o nome do arquivo gerado.

3. ``fileFilter(req, file, cb) { ... }``:

- Neste caso, ``cb`` também é uma função de retorno de chamada.
- Quando a função ``fileFilter`` é chamada, ela verifica se o arquivo enviado atende aos critérios estabelecidos (no caso, se é um arquivo PNG ou JPG) e executa a função de retorno de chamada ``cb`` com o resultado da verificação.
- Se o arquivo não atender aos critérios, ``cb`` é chamado com um erro como primeiro argumento, informando que o upload não é permitido.
- Se o arquivo atender aos critérios, ``cb`` é chamado com ``undefined`` como primeiro argumento, indicando que o upload é permitido.

Em resumo, em todas as ocorrências, a função ``cb`` é usada para comunicar o resultado de uma operação de volta ao multer, seja para fornecer um caminho de destino para salvar um arquivo, um nome de arquivo gerado, ou para indicar se o upload do arquivo é permitido ou não.