Estruturando um Router com Express #Backend #Nodejs

Publicado em 5 de fevereiro de 2022

Vitor Ferreira
Full Stack Web Developer | JAVA | React | Node.js | Python

7 artigos + Seguir

Em 17 de janeiro de 2022 iniciei o projeto denominado "Cookmaster", avaliativo na Trybe, uma escola de programação Web que tem exercido um papel importante na formação de novos profissionais para a área de tecnologia.

Esse projeto faz parte do módulo de backend, especificamente para consolidação de conhecimentos de Node.js, Express (pacote que facilita

Dentre muitos desafios que o projeto exigiu, estruturar um Router não era um requerimento formal, mas aproveitei a oportunidade do projeto para aprender como fazê-lo.

Considerando que experienciei algumas dificuldades com esse processo, resolvi compartilhar pois talvez ajude alguém na mesma situação.

No final do artigo você encontrará um link para um vídeo (em ingês) que me ajudou a entender essa estrutura.

Direto ao assunto

1 - Considerando a configuração inicial de um servidor com o Express teremos:

```
index.js > ...
1  const express = require('express');
2
3  const app = express();
4
5  const port = 3000;
6
7  app.get('/', (req, res) => res.send('Hello World!'));
8
9  app.listen(port, () => console.log(`Example app listening on port ${port}!`));
10
```

No exemplo acima a variável app recebe uma instancia do express e através dela podemos chamar as funções que irão tratar as requisições http com seus respectivos verbos (get, put, post, delete...). Nenhuma novidade até aqui...

A medida que sua aplicação cresce e a quantidade de rotas vai ganhando complexidade, a organização se faz necessária:

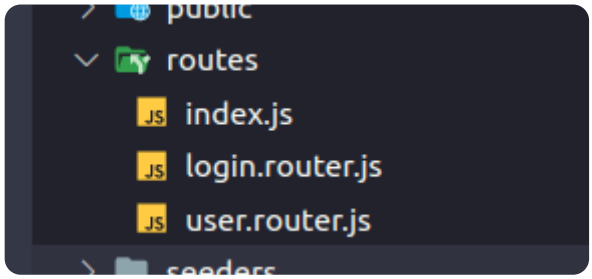
```
index.js > ...
```

```
app.get('/', (req, res) => res.send('Hello World!'));
app.post('/user', () => {});
app.post('/login', () => {});
app.put('/user/:id', () => {});
app.delete('/user/:id', () => {});
app.post('/posts', () => {});
app.put('/posts/:id', () => {});
app.delete('/posts/:id', () => {});

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

2 - Ainda no app, importe a pasta "routes" (linha 3) e utilize-a no método "use" (linha 9):

```
index.js > ...
const express = require('express');

const router = require('./routes');

const app = express();

const port = 3000;

app.use(router);

// AS ROTAS ABAIXO SERÃO ORGANIZADAS NA PASTA ROUTES

// app.get('/', (req, res) => res.send('Hello World!'));
// app.post('/user', () => {});
// app.post('/login', () => {});
// app.put('/user/:id', () => {});
// app.delete('/user/:id', () => {});
// app.post('/posts', () => {});
// app.put('/posts/:id', () => {});
// app.delete('/posts/:id', () => {});

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

3 - Crie a pasta "routes" no seu projeto Node.js:



4 - Na pasta routes, crie um arquivo index.js (imagem acima) e considere o código abaixo como exemplo:

```
index.js M
routes > index.js > ...
const express = require('express'); // importando a lb express
const usersRouter = require('./user.router'); // importando o router da rota /user
const loginRouter = require('./login.router'); // importando o router da rota /login

const router = express.Router(); // criando uma instancia do router disponibilizao no express

router.use('/login', loginRouter); // "pingou" na rota /login chama o router loginRouter (especificado acima)
```

O arquivo index.js contido na pasta routes demonstrado na imagem acima importa o express (linha 1), instancia um router (linha 5) e com ele é possível usar o método "use" mais uma vez (linhas 7 e 8), descriminando a rota "raiz" que será identificada na requisição http e para onde ela será encaminhada.

A essa altura você pode notar que a aplicação ganhou de fato mais uma camada. uma pasta com um arquivo index que vai funcionar como um legítimo Router, gerenciando para onde cada requisição será encaminhada.

5 - Caso seja feita uma requisição do tipo post para a rota "/login" no servidor express (app - imagem 3), essa será transferida para o router instanciado no index da pasta routes, que por sua vez irá verificar que a requisição deve ser transferida para o arquivo denominado como "loginRouter" (linha 7 da imagem acima), chegando lá teremos:

routes > login.router.js > ...

```
1  const express = require('express');
2
3  const loginController = require('../controllers/login.controller');
4
5  const router = express.Router();
6
7  router.post('/', loginController.login);
8
9  module.exports = router;
10
```

Nesse caso simplista a requisição do tipo post na rota "/login" chegou no arquivo ./routes/login.router.js, e foi encaminhada para a camada controller (linha 7 da imagem acima), chegando lá provavelmente será enviada para a camada de services, que por sua vez realizará validações e terá relacionamento com a camada de model e por aí vai...

pode-se importar um middleware de autenticação para verificar se um usuário possui um token para acessar a rota especifica da "raiz", outro para verificar o nível de acesso (administrador, cliente, usuário etc) ou declarar funções para tratar, por exemplo, o upload de imagens.

O exemplo abaixo refere-se ao arquivo **recipes.router.js** da rota "/recipes" desenvolvida no projeto Cookmaster. Considerando uma requisição do tipo PUT na rota "/recipes/:id/image/" o router contido no arquivo index.js da pasta routes primeiramente identificaria a raiz "/recipes", daí encaminharia para o router específico dessa rota (última imagem) e com o restante da string o match iria "cair" na linha 54 (imagem abaixo) que possui middlewares de (i) autenticacao (auth), (ii) validação de "poderes" (checkUserowers), (iv) chamava uma função para upload (upload.single('image')) e (v) finalmente seguia para a camada de controller (recipesController.uploadImageRecipeController).

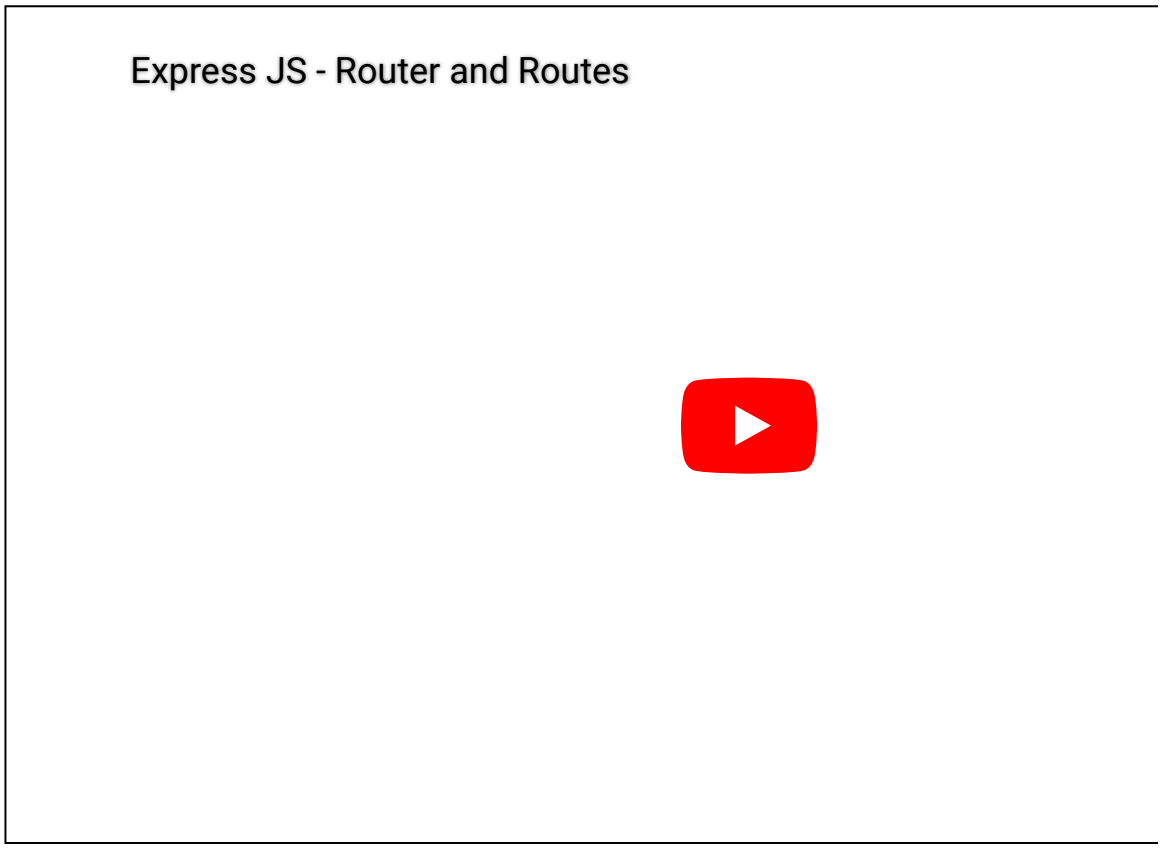
Essa estrutura faz com que a url da requisição sofre intencionamente, a critério do desenvolvedor, um "split" **(["/recipes", "/:id/image/"])**.

```
54  router.put(
55    '/:id/image/',
56    auth,
57    checkUserPowers,
58    upload.single('image'),
59    recipesController.uploadImageRecipeController,
60  );
```

```
24  const express = require('express');
25  const multer = require('multer');
26  const path = require('path');
27  const auth = require('../middlewares/auth');
28  const checkUserPowers = require('../middlewares/checkUserPower');
29  const recipesController = require('../controllers/recipes.controller');
30
31  const router = express.Router();
32
33  const storage = multer.diskStorage({…
39  });
40
41  const upload = multer({ storage });
```

```
será verificado se o token contido no atributo 'authorization'
do headers da requisição é válido.
O middleware 'auth' utiliza-se da função verifyToken contido
no auth.service (pasta services).
*/

// router.use(auth);

router.put(
  '/:id/image/',
  auth,
  checkUserPowers,
  upload.single('image'),
  recipesController.uploadImageRecipeController,
);

router.get('/:id', recipesController.getRecipeByIdController);

router.put('/:id', auth, recipesController.updateRecipeController);

router.delete('/:id', auth, recipesController.deleteRecipeController);

router.post('/', auth, recipesController.addRecipeController);

router.get('/', recipesController.getAllRecipesController);

module.exports = router;
```

Recurso adicional



Espero que o conteúdo tenha sido útil! Até breve!