Documentação API Blog

Adriel Marcelo Costa Modollo

31/10/21

Sumário

Estrutura do projeto

O sistema possuí uma estrutura onde temos nosso “config” para gerenciar todas nossas rotas através do comando node: module.exports, note que a função dá acesso a todas rotas através do caminho informado e a variável “file”. Através deste arquivo de configuração conseguimos conectar todas rotas, agilizando o processo de criação e deixando o código mais limpo, assim eliminando a necessidade de sobrecarregar com muitas informações somente um arquivo. Como por exemplo, poderíamos colocar para cada novo “routes” uma chamada dentro do nosso “index.js”, que nada mais é onde está nossa conexão com mongodb e acesso as portas de conexão. Desta forma ao invés de se preocupar toda vez com uma nova chamada, simplesmente declaramos uma vez “setupRoutes(app)” e a chamada de requisição “const setupRouters = require("**caminho**");” desta forma o caminho fica configurado e nosso arquivo de “config” faz todo o resto do trabalho, bastando a partir deste momento criar todas rotas que desejar, lembrando somente de usar o module.exports = nomeDaRota.

Essas rotas são controladas através do nosso arquivo “controllers” que é onde criamos toda parte lógica do projeto, assim conseguimos gerenciar as funções de forma mais fácil e ajudar em implementações futura e deixar isso bem dividido, para sabermos onde está sendo implementado a parte lógica. Note que sua funcionalidade é bem simples, onde declaramos nossas funções e em seguida exportamos elas para se comunicar com nossos arquivos de rotas.

Desta forma o sistema é estruturado, o que incluí alguns dos conceitos principais de SOLID para nos ajudar no dia a dia e facilitar no desenvolvimento para que possa se manter e se estender de forma estável.

Gerenciador de pacotes

O Yarn foi instalado de modo padrão: npm install --global yarn

Optamos pela dependência Yarn por trazer um ganho em nosso tempo de download de pacotes, seus pacotes utilizados foi:

```
yarn add express mongoose dotenv multer
```

(Que permite o servidor expresso para conectar com nosso mongoose e Multer facilita o processo de upload de imagens).

```
npm init
```

(Onde estruturamos no package.json e configurações iniciais por padrão)

```
yarn add nodemon
```

(Automatiza o reinicio da aplicação para reconhecer novas modificações).

```
yarn add bit crypt
```

(Para proteger a visualização no postman de nossa senha registrada no mongodb.)

```
yarn global add node-gyp
```

(Como havia me deparado com alguns erros do bitcrypt, foi necessário executar o comando acima para solucionar os problemas do pacote bitcrypt)

```
yarn add eslint-config-standard eslint-plugin-import eslint-plugin-node
eslint-plugin-promise eslint-plugin-standard
```

(Linter corrigi falhas de indentação, como formatação ou até mesmo algum “;” que possa estar incluído de forma despercebida).

Após instalar todas essas dependências a parte já criamos nosso package .json, agora a partir deste momento é necessário somente executar o comando:

```
yarn
```

Mongodb

Foi configurado com o banco de forma padrão, passando nossa URL dentro de nosso arquivo .env e recuperando essas informações para conexão através de nosso index.js, utilizando nossa função connect.

.env

Renomeie o arquivo **example .env** para **.env** e insira as conexões de cluster do mongodb no lugar da URL de exemplo.

Docker

Dockerfile

Arquivo de configuração onde é declarado nossa versão node, local do projeto, arquivo que será copiado, comando de execução para instalação de

pacotes, o que será copiado, no caso é utilizado “. . “para copiar todo projeto, expose para atribuirmos a porta, terminal e comando de execução do projeto.

docker-compose.yml

Services

Declaramos nossos serviços, neste caso só utilizamos o mongo e atribuímos no volume, que é nosso armazenamento e a porta que será conectada.

Server

Declaramos todos restante do conteúdo, ou seja, nome do nosso container Docker, o que será construído nesta imagem, no caso atribuí “.” para construir a imagem de todo projeto, nossa porta configurada e conexão com nossa variável “MONGO_URL” porém com os parâmetros de conexão Docker, como será criado e local do volume da imagem Docker, reinicio automático configurado com sempre e porta de conexão do projeto e link mongo.

Iniciar Docker

Execute o comando: docker-compose up

Models

Category:

Name: Apenas atribuído o nome de categoria como string e requirido como campo obrigatório para envio.

Post:

Title: atribuído título como string e campo obrigatório, além de unique para evitar o armazenamento de mesmos valores.

Desc: atribuído desc como string e campo obrigatório, além de unique para evitar o armazenamento de mesmos valores.

Photo: atribuído photo como string e não obrigatório

Username: atribuído username como string e campo obrigatório.

Categories: atribuído categories como array e não obrigatório.

User:

Username: atribuído username como string obrigatória e unique para evitar armazenamento de mesmos valores.

Email: atribuído email como string obrigatória e unique para evitar armazenamento de mesmos valores.

Password: atribuído password como string obrigatória.

Routes

Autenticação (Auth)

Cadastro:

Método post, para envio e registro de nossa requisição através do caminho: “/register”, a função assíncrona tem todos os parâmetros de usuário, ou seja, username, email e password. Neste momento utilizamos nosso pacote bcrypt para ocultar nossa senha e evitar retornar essa senha, mesmo que esteja criptografada.

Exemplo: localhost:3001/auth/register

Login:

Método post, para envio e login no sistema através de nossa requisição no caminho: “/login”, a função assíncrona tem parâmetros de usuário, ou seja, username e password. Neste momento utilizamos nosso pacote bcrypt para ocultar nossa senha e evitar retornar essa senha, mesmo que esteja criptografada e realizamos a verificação para ver se o username e password está correto e pode ter acesso ao sistema!

Exemplo: localhost:3001/auth/login

Categorias (Categories)

Cadastro:

Método post, para envio e registro de nossa requisição através do caminho: “/”, a função assíncrona tem todos os parâmetros de categorias, ou seja, utiliza-se somente de name, por isso conseguimos passar na requisição desta forma, onde todos os campos serão validados.

Exemplo: localhost:3001/categories/register

Recuperar informações:

Método get, onde conseguimos recuperar as informações de todas categorias solicitada, exemplo: localhost:3001/categories/consult. Desta forma irá retornar qualquer registro!

Postagens (Posts)

Cadastro:

Método post, para envio e registro de nossa requisição através do caminho: “/”, a função assíncrona tem todos os parâmetros de posts, ou seja, utiliza-se name, desc e username, pois estes campos são os obrigatórios, porém pode-se utilizar também os não obrigatórios photo e categories.

Exemplo: localhost:3001/posts/register

Atualizar (Posts)

Método put, para envio e atualização de dados de postagem através de nossa requisição que se encontra no caminho: “/:id”, a função assíncrona tem todos os parâmetros de posts, ou seja, utiliza-se name, desc e username, pois estes campos são os obrigatórios, porém pode-se utilizar também os não obrigatórios photo e categories. Lembrando que é essencial preencher esses campos obrigatórios para o pleno funcionamento do envio. Para atualizarmos utilizamos findByIdAndUpdate que localiza no ID e consegue atualizar os dados conforme o solicitado

Exemplo: localhost:3001/posts/update/idDoPost

Deletar (Posts)

Método delete, para exclusão através de nosso ID, ou seja, é necessário somente informar nosso ID de postagem para que aja uma request, porém é feita a validação onde permite somente o usuário que criou a postagem deletar. Necessário informar o username responsável pela postagem, title e desc que será modificado.

Exemplo: localhost:3001/posts/exclude/colocarIDpost

Recuperar informações por ID (Posts)

Método get, onde conseguimos recuperar as informações de postagens através do ID, desta forma basta informar da seguinte maneira: localhost:3001/posts/consultID/idPost.

Recuperar informações com condição (Posts)

Método get, onde é conseguimos recuperar todas as postagens feita por cada usuário ou por categoria. Exemplo de requisição: localhost:3001/posts/consult?user=username

Usuários (Users)

Atualizar (Users)

Método put, para envio e atualização de dados de usuário através de nossa requisição que se encontra no caminho: “/:id”, a função assíncrona tem todos os parâmetros de posts, ou seja, utiliza-se username, email e password. Lembrando que é essencial preencher esses campos obrigatórios para o pleno funcionamento do envio. Para atualizarmos utilizamos findByIdAndUpdate que localiza no ID e consegue atualizar os dados conforme o solicitado

Exemplo: localhost:3001/users/update/idDoUsuário

Deletar (Users)

Método delete, para exclusão através de nosso ID, ou seja, é necessário somente informar nosso ID de usuário e autenticar a validação com os campos userID, username e password.

Exemplo: localhost:3001/users/exclude/colocarIDusuario

Recuperar informações por ID (Users)

Método get, onde conseguimos recuperar as informações de usuário através do ID, desta forma basta informar da seguinte maneira: localhost:3001/users/consultID/colocarIDusuario.

Index

Conectamos todas nossas routes e models através de require, informando os respectivos caminho de cada arquivo, desta forma enviando a requisição http através da porta informada 3001, onde está conectado ao mongodb com a função connect e url no arquivo .env. Note que também existe duas funções que foi aplicada para caso um dia houver uma parte de client, sendo possível enviar imagens. Essas funções é upload e storage que se utiliza do pacote multer que facilita este processo.

Realizando testes com postman

Necessário estar selecionado “Raw” e “Json” para conseguirmos enviar as requests. Caso preferir você pode baixar os testes utilizado pronto.

Caso tenha optado por baixar o arquivo pronto: import > upload files > selecione o arquivo > import.

Users:

localhost:3001/auth/register – Método post para cadastro de usuário

```
{
    "username":"seuUsuário",
    "email":"email@servidor.com",
    "password":"suaSenha"
}
```

localhost:3001/auth/login - Método post para acesso ao usuário

```
{
    "username": " seuUsuário ",
    "password": "suaSenha"
}
```

localhost:3001/users/update/colocarIDusuario – Método put para atualização de dados do usuário

```
{
    "userId":"idDoUsuário",
    "username": "seuUsuário",
    "password": "suaSenha"
}
```

localhost:3001/users/consultID/colocarIDusuario – Método get para recuperar usuários

localhost:3001/users/exclude/colocarIDusuario – Método delete para exclusão de usuário.

```
{
    "userId":"idDoUsuário",
    "username": "seuUsuário",
    "password": "suaSenha"
}
```

Posts

localhost:3001/posts/register – Método post para criação de postagem.

```
{
    "username": "seuUsuário",
    "title": "titulo",
    "desc": "descrição"
}
```

localhost:3001/posts/update/colocarIDpost - Método put para atualização de postagem

```
{
    "username": "seuUsuário",
    "title": "titulo",
    "desc": "descrição"
}
```

localhost:3001/posts/exclude/colocarIDpost – Método delete para exclusão de postagem

```
{
    "username": "seuUsuário",
    "title": "titulo",
    "desc": "descrição"
}
```

localhost:3001/posts/consultID/colocarIDpost – Método get para recuperar postagem por ID

localhost:3001/posts/consult – Método get para recuperar todas postagens

localhost:3001/posts/consult?user=username – Método get para recuperar postagem por usuário

Categories

localhost:3001/categories/register – Método post para cadastrar categoria

```
{
    "name": "nome da categoria"
}
```

localhost:3001/categories/consult – Método get para buscar todas categorias

localhost:3001/upload?file=suaImagem.extensão_da_imagem – Método post para envio de imagens