# Documentação API Blog

**Adriel Marcelo Costa Modollo**

**31/10/21**

Sumário

## Estrutura do projeto

O sistema consiste em uma estrutura baseada em MVC, porém como é utilizado somente para api, a parte de visualização foi retirada. No caso agora temos nosso caminho de rotas (Routes) e criação do nosso banco(models). Além dessas principais funcionalidades, temos nossos arquivos de configuração como: .json, .env e nosso arquivo index.js que concluí a comunicação com nossas rotas e models.

## 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)

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.

# 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:5000/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:5000/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:5000/categories

### Recuperar informações:

Método get, onde conseguimos recuperar as informações de todas categorias solicitada, exemplo: localhost:5000/categories. 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:5000/posts

## 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:5000/posts/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:5000/posts/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:5000/posts/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:5000/posts?user=adrielmodollo

## 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:5000/posts/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:5000/user/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:5000/users/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 5000, 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:5000/auth/register – Método post para cadastro de usuário

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

localhost:5000/auth/login - Método post para acesso ao usuário

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

localhost:5000/users/colocarIDusuario – Método put para atualização de dados do usuário

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

localhost:5000/users/colocarIDusuario – Método get para recuperar usuários

localhost:5000/users/colocarIDusuario – Método delete para exclusão de usuário.

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

## Posts

localhost:5000/posts – Método post para criação de postagem.

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

localhost:5000/posts/colocarIDpost - Método put para atualização de postagem

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

localhost:5000/posts/colocarIDpost – Método delete para exclusão de postagem

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

localhost:5000/posts/colocarIDpost – Método get para recuperar postagem por ID

localhost:5000/posts – Método get para recuperar todas postagens

localhost:5000/posts?user=username – Método get para recuperar postagem por usuário

## Categories

localhost:5000/categories – Método post para cadastrar categoria

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

localhost:5000/categories – Método get para buscar todas categorias

localhost:5000/upload?file=suaImagem.extensão_da_imagem – Método post para envio de imagens