Desafio Módulo 3 - Back-end

Seu papel é construir uma RESTful API que permita:

- Fazer Login
- Cadastrar Usuário
- Detalhar Usuário
- Editar Usuário
- Listar produtos
- Detalhar produtos
- Cadastrar produtos
- Editar produtos
- Remover produtos
- **EXTRA:** Filtrar produtos por categoria

Importante: Lembre-se sempre que cada usuário só pode ver e manipular seus próprios dados e seus próprios produtos. Não atender a este pré-requisito é uma falha de segurança gravíssima!

Importante 2: O diretório ".github" e seu conteúdo não podem ser alterados e muito menos excluídos

Importante 3: Sempre que a validação de uma requisição falhar, responda com código de erro e mensagem adequada à situação, ok?

Exemplo:

```
// Quando é informado um id de usuário que não existe:
// HTTP Status 404
{
    mensagem: "Usuário não encontrado!"
}
```

Banco de dados

Você precisa criar um Banco de Dados PostgreSQL chamado `market_cubos` contendo as seguintes tabelas e colunas:
ATENÇÃO! Os nomes das tabelas e das colunas a serem criados devem seguir exatamente os nomes listados abaixo.

- usuarios
  - id
  - nome
  - nome_loja (o nome da loja deste vendedor)
  - email (campo único)
  - senha
- produtos
  - id
  - usuario_id
  - nome
  - quantidade
  - categoria
  - preco
  - descricao
  - imagem (campo texto para URL da imagem na web)

IMPORTANTE: Na raiz do seu repositório forkado deverá ser criado um arquivo SQL que deverá ser o script que cria as tabelas corretamente.

Requisitos obrigatórios

- A API a ser criada deverá acessar o banco de dados a ser criado "market_cubos" para persistir e manipular os dados de usuários e produtos utilizados pela aplicação.
- O campo `id` das tabelas no banco de dados deve ser auto incremento, chave primária e não deve permitir edição uma vez criado.
- Seu código deverá estar organizado, delimitando as responsabilidades de cada arquivo adequadamente. Ou seja, é esperado que ele tenha, no mínimo:
  - Um arquivo index.js
  - Um arquivo servidor.js
  - Um arquivo conexao.js
  - Um arquivo de rotas

    - Um pasta com controladores
      ATENÇÃO!: os arquivos iniciais já existentes neste repositório original (index.js, servidor.js e conexao.js) **não deverão ser renomeados e nem movidos dentro da estrutura de pastas do projeto**. O arquivo **conexao.js** deverá ser alterado **apenas** os valores das propriedades que definem **as credenciais de acesso** ao seu banco de dados.
- Qualquer valor monetário deverá ser representado em centavos (Ex.: R$ 10,00 reais = 1000)
- Evite códigos duplicados. Antes de copiar e colar, pense se não faz sentido esse pedaço de código estar centralizado numa função.

Status Codes

Abaixo, listamos os possíveis ***status codes*** esperados como resposta da API.

```
// 200 (OK) = requisição bem sucedida
// 201 (Created) = requisição bem sucedida e algo foi criado
// 204 (No Content) = requisição bem sucedida, sem conteúdo no corpo da resposta
// 400 (Bad Request) = o servidor não entendeu a requisição pois está com uma sintaxe/formato ir
// 401 (Unauthorized) = o usuário não está autenticado (logado)
// 403 (Forbidden) = o usuário não tem permissão de acessar o recurso solicitado
// 404 (Not Found) = o servidor não pode encontrar o recurso solicitado
```

Endpoints

Cadastrar usuário

`POST  /usuario`

Essa é a rota que será utilizada para cadastrar um novo usuario no sistema.

- **Requisição**
  Sem parâmetros de rota ou de query.
  O corpo (body) deverá possuir um objeto com as seguintes propriedades (respeitando estes nomes):
    - nome
    - email
    - senha
    - nome_loja
- **Resposta**
  Em caso de **sucesso**, não deveremos enviar conteúdo no corpo (body) da resposta.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu

corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.

- **REQUISITOS OBRIGATÓRIOS**
  - Validar os campos obrigatórios:
    - nome
    - email
    - senha
    - nome_loja
  - Validar se o e-mail informado já existe
  - Criptografar a senha antes de persistir no banco de dados
  - Cadastrar o usuário no banco de dados

Exemplo de requisição

```
// POST /usuario
{
    "nome": "José",
    "email": "jose@lojadasflores.com.br",
    "senha": "j1234",
    "nome_loja": "Loja das Flores"
}
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
// Sem conteúdo no corpo (body) da resposta


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Já existe usuário cadastrado com o e-mail informado."
}
```

Login do usuário

`POST  /login`

Essa é a rota que permite o usuario cadastrado realizar o login no sistema.

- **Requisição**
  Sem parâmetros de rota ou de query.
  O corpo (body) deverá possuir um objeto com as seguintes propriedades (respeitando estes nomes):

    - email
    - senha
- **Resposta**
  Em caso de **sucesso**, o corpo (body) da resposta deverá possuir um objeto com apenas uma propriedade **token** que deverá possuir como valor o token de autenticação gerado.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
- **REQUISITOS OBRIGATÓRIOS**
    - Validar os campos obrigatórios:
        - email
        - senha
    - Verificar se o e-mail existe
    - Validar e-mail e senha
    - Criar token de autenticação com id do usuário

Exemplo de requisição

```
// POST /login
{
    "email": "jose@lojadasflores.com.br",
    "senha": "j1234"
}
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNjIzMjQ5NjIxLCJleHAiOjE2M
}
```

```
// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Usuário e/ou senha inválido(s)."
}
```

ATENÇÃO: Todas as funcionalidades (endpoints) a seguir, a partir desse ponto, deverão exigir o token de

autenticação do usuário logado, recebendo no header com o formato Bearer Token. Portanto, em cada funcionalidade será necessário validar o token informado.

Validações do token

- **REQUISITOS OBRIGATÓRIOS**
  - Validar se o token foi enviado no header da requisição (Bearer Token)
  - Verificar se o token é válido
  - Consultar usuário no banco de dados pelo id contido no token informado

Detalhar usuário

`GET /usuario`

Essa é a rota que será chamada quando o usuario quiser obter os dados do seu próprio perfil.
Atenção!: O usuário deverá ser identificado através do ID presente no token de autenticação.

- **Requisição**
  Sem parâmetros de rota ou de query.
  Não deverá possuir conteúdo no corpo da requisição.
- **Resposta**
  Em caso de **sucesso**, o corpo (body) da resposta deverá possuir um objeto que representa o usuário encontrado, com todas as suas propriedades (exceto a senha), conforme exemplo abaixo, acompanhado de ***status code*** apropriado.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
  Dica: neste endpoint podemos fazer uso do status code 401 (Unauthorized).

Exemplo de requisição

```
// GET /usuario
// Sem conteúdo no corpo (body) da requisição
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
{
    "id": 1,
    "nome": "José",
    "email": "jose@lojadasflores.com.br",
    "nome_loja": "Loja das Flores"
}


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Para acessar este recurso um token de autenticação válido deve ser enviado."
}
```

Atualizar usuário

`PUT /usuario`

Essa é a rota que será chamada quando o usuário quiser realizar alterações no seu próprio usuário.
Atenção!: O usuário deverá ser identificado através do ID presente no token de autenticação.

- **Requisição**
  Sem parâmetros de rota ou de query.
  O corpo (body) deverá possuir um objeto com as seguintes propriedades (respeitando estes nomes):
  - nome
  - email
  - senha
  - nome_loja
- **Resposta**
  Em caso de **sucesso**, não deveremos enviar conteúdo no corpo (body) da resposta.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
  Dica: neste endpoint podemos fazer uso do status code 401 (Unauthorized)
- **REQUISITOS OBRIGATÓRIOS**
  - Validar os campos obrigatórios:
    - nome
    - email
    - senha
    - nome_loja
  - Validar se o novo e-mail já existe no banco de dados para outro usuário

        - Caso já exista o novo e-mail fornecido para outro usuário no banco de dados, a alteração não deve ser permitida (o campo de email deve ser sempre único no banco de dados)
    - Criptografar a senha antes de salvar no banco de dados
    - Atualizar as informações do usuário no banco de dados

Exemplo de requisição

```
// PUT /usuario
{
    "nome": "José de Abreu",
    "email": "jose_abreu@gmail.com",
    "senha": "j4321",
    "nome_loja": "Loja das Flores Cheirosas"
}
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
// Sem conteúdo no corpo (body) da resposta


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "O e-mail informado já está sendo utilizado por outro usuário."
}
```

Listar produtos do usuário logado

`GET /produtos`

Essa é a rota que será chamada quando o usuario logado quiser listar todos os seus produtos cadastrados.
Lembre-se: Deverão ser retornados **apenas** produtos associados ao usuário logado, que deverá ser identificado através do ID presente no token de validação.

- **Requisição**
  Sem parâmetros de rota ou de query.
  Não deverá possuir conteúdo no corpo (body) da requisição.
- **Resposta**
  Em caso de **sucesso**, o corpo (body) da resposta deverá possuir um array dos objetos (produtos) encontrados.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu

corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
Dica: neste endpoint podemos fazer uso do status code 401 (Unauthorized).

- **REQUISITOS OBRIGATÓRIOS**
  - O usuário deverá ser identificado através do ID presente no token de validação
  - O endpoint deverá responder com um array de todos os produtos associados ao usuário. Caso não exista nenhum produto associado ao usuário deverá responder com array vazio.

Exemplo de requisição

```
// GET /produtos
// Sem conteúdo no corpo (body) da requisição
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
[
    {
        "id": 1,
        "usuario_id": 1,
        "nome": "Camisa preta",
        "quantidade": 12,
        "categoria": "Camisas",
        "preco": 4990,
        "descricao": "Camisa de malha com acabamento fino.",
        "imagem": "https://bit.ly/3ctikxq",
    },
    {
        "id": 2,
        "usuario_id": 1,
        "nome": "Calça jeans azul",
        "quantidade": 8,
        "categoria": "Calças",
        "preco": 4490,
        "descricao": "Calça jeans azul.",
        "imagem": "https://bit.ly/3ctikxq",
    },
]
```

```
// HTTP Status 200 / 201 / 204
[]
```

```
// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Para acessar este recurso um token de autenticação válido deve ser enviado."
}
```

Detalhar um produto do usuário logado

`GET  /produtos/:id`

Essa é a rota que será chamada quando o usuario logado quiser obter um dos seus produtos cadastrados.
Lembre-se: Deverá ser retornado **apenas** produto associado ao usuário logado, que deverá ser identificado através do ID presente no token de validação.

- **Requisição**
  Deverá ser enviado o ID do produto no parâmetro de rota do endpoint.
  O corpo (body) da requisição não deverá possuir nenhum conteúdo.
- **Resposta**
  Em caso de **sucesso**, o corpo (body) da resposta deverá possuir um objeto que representa o produto encontrado, com todas as suas propriedades, conforme exemplo abaixo, acompanhado de ***status code*** apropriado.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
  Dica: neste endpoint é possível utilizar os ***status codes*** 401 (Unauthorized) e 403 (Forbidden), além dos outros mais comuns que costumamos utilizar.
- **REQUISITOS OBRIGATÓRIOS**
  - Validar se existe produto para o id enviado como parâmetro na rota e se este produto pertence ao usuário logado.

Exemplo de requisição

```
// GET /produtos/44
// Sem conteúdo no corpo (body) da requisição
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
{
    "id": 1,
    "usuario_id": 1,
    "nome": "Camisa preta",
    "quantidade": 8,
    "categoria": "Camisa",
    "preco": 4990,
    "descricao": "Camisa de malha com acabamento fino.",
    "imagem": "https://bit.ly/3ctikxq"
}
```

```
// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Não existe produto cadastrado com ID 44."
}
```

```
// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "O usuário logado não tem permissão para acessar este produto."
}
```

Cadastrar produto para o usuário logado

`POST  /produtos`

Essa é a rota que será utilizada para cadastrar um produto associado ao usuário logado.
Lembre-se: Deverá ser possível cadastrar **apenas** produtos associados ao próprio usuário logado, que deverá ser identificado através do ID presente no token de validação.

- **Requisição**
  Sem parâmetros de rota ou de query.
  O corpo (body) da requisição deverá possuir um objeto com as seguintes propriedades (respeitando estes nomes):
  - nome
  - quantidade
  - categoria
  - preco
  - descricao
  - imagem
- **Resposta**
  Em caso de **sucesso**, não deveremos enviar conteúdo no corpo (body) da resposta.

Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
Dica: neste endpoint podemos fazer uso do status code 401 (Unauthorized).

- **REQUISITOS OBRIGATÓRIOS**
  - Validar os campos obrigatórios:
    - nome
    - quantidade
    - preco
    - descricao
  - Validar se a quantidade do produto é maior que zero.
  - Cadastrar o produto associado ao usuário logado.

Exemplo de requisição

```
// POST /produtos
{
    "nome": "Camisa preta",
    "quantidade": 8,
    "categoria": "Camisa",
    "preco": 4990,
    "descricao": "Camisa de malha com acabamento fino.",
    "imagem": "https://bit.ly/3ctikxq"
}
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
// Sem conteúdo no corpo (body) da resposta


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "O preço do produto deve ser informado."
}


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Para cadastrar um produto, o usuário deve estar autenticado."
}
```

Atualizar produto do usuário logado

`PUT /produtos/:id`

Essa é a rota que será chamada quando o usuario logado quiser atualizar um dos seus produtos cadastrados.
Lembre-se: Deverá ser possível atualizar **apenas** produtos associados ao próprio usuário logado, que deverá ser identificado através do ID presente no token de validação.

- **Requisição**
  Deverá ser enviado o ID do produto no parâmetro de rota do endpoint.
  O corpo (body) da requisição deverá possuir um objeto com as seguintes propriedades (respeitando estes nomes):
  - nome
  - quantidade
  - categoria
  - preco
  - descricao
  - imagem
- **Resposta**
  Em caso de **sucesso**, não deveremos enviar conteúdo no corpo (body) da resposta.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
  Dica: neste endpoint é possível utilizar os ***status codes*** 401 (Unauthorized) e 403 (Forbidden), além dos outros mais comuns que costumamos utilizar.
- **REQUISITOS OBRIGATÓRIOS**
  - Validar se existe produto para o id enviado como parâmetro na rota e se este produto pertence ao usuário logado.
  - Validar os campos obrigatórios:
    - nome
    - quantidade
    - preco
    - descricao
  - Atualizar o produto no banco de dados

Exemplo de requisição

```
// PUT /produtos/2
{
    "nome": "Calça jeans preta",
    "quantidade": 22,
    "categoria": "Calças",
    "preco": 4490,
    "descricao": "Calça jeans preta.",
    "imagem": "https://bit.ly/3ctikxq"
}
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
// Sem conteúdo no corpo (body) da resposta


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "O usuário autenticado não ter permissão para alterar este produto."
}
```

Excluir produto do usuário logado

`DELETE  /produtos/:id`

Essa é a rota que será chamada quando o usuario logado quiser excluir um dos seus produtos cadastrados.
Lembre-se: Deverá ser possível excluir **apenas** produtos associados ao próprio usuário logado, que deverá ser identificado através do ID presente no token de validação.

- **Requisição**
  Deverá ser enviado o ID do produto no parâmetro de rota do endpoint.
  O corpo (body) da requisição não deverá possuir nenhum conteúdo.
- **Resposta**
  Em caso de **sucesso**, não deveremos enviar conteúdo no corpo (body) da resposta.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
  Dica: neste endpoint é possível utilizar os ***status codes*** 401 (Unauthorized) e 403 (Forbidden), além dos outros mais comuns que costumamos utilizar.
- **REQUISITOS OBRIGATÓRIOS**:
  - Validar se existe produto para o id enviado como parâmetro na rota e se este produto pertence ao usuário logado.

- Excluir o produto no banco de dados.

Exemplo de requisição

```
// DELETE /produtos/88
// Sem conteúdo no corpo (body) da requisição
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
// Sem conteúdo no corpo (body) da resposta
```

```
// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Não existe produto para o ID 88."
}
```

```
// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "O usuário autenticado não tem permissão para excluir este produto."
}
```

EXTRA

ATENÇÃO!: Esta parte extra não é obrigatória e recomendamos que seja feita apenas quando terminar toda a parte obrigatória acima.

Filtrar produtos por categoria

Na funcionalidade de listagem de produtos do usuário logado (**GET /produtos**), deveremos incluir um parâmetro do tipo query **categoria** para que seja possível consultar apenas produtos de uma categoria específica.
Lembre-se: Deverão ser retornados **apenas** produtos associados ao usuário logado, que deverá ser identificado através do ID presente no token de validação.

- **Requisição**
  Parâmetro opcional do tipo query **categoria**.
  Não deverá possuir conteúdo no corpo (body) da requisição.

- **Resposta**
  Em caso de **sucesso**, o corpo (body) da resposta deverá possuir um array dos objetos (produtos) encontrados.
  Em caso de **falha na validação**, a resposta deverá possuir ***status code*** apropriado, e em seu corpo (body) deverá possuir um objeto com uma propriedade **mensagem** que deverá possuir como valor um texto explicando o motivo da falha.
  Dica: neste endpoint podemos fazer uso do status code 401 (Unauthorized).
- **REQUISITOS OBRIGATÓRIOS**
  - O usuário deverá ser identificado através do ID presente no token de validação
  - O endpoint deverá responder com um array de todos os produtos associados ao usuário que sejam da categoria passada no parâmetro query. Caso não exista nenhum produto associado ao usuário deverá responder com array vazio.

Exemplo de requisição

```
// GET /produtos?categoria=camisas
// Sem conteúdo no corpo (body) da requisição
```

Exemplos de resposta

```
// HTTP Status 200 / 201 / 204
[
    {
        "id": 1,
        "usuario_id": 1,
        "nome": "Camisa preta",
        "quantidade": 12,
        "categoria": "Camisas",
        "preco": 4990,
        "descricao": "Camisa de malha com acabamento fino.",
        "imagem": "https://bit.ly/3ctikxq",
    },
    {
        "id": 2,
        "usuario_id": 1,
        "nome": "Calça jeans azul",
        "quantidade": 8,
        "categoria": "Calças",
        "preco": 4490,
        "descricao": "Calça jeans azul.",
        "imagem": "https://bit.ly/3ctikxq",
    },
]
```

```
// HTTP Status 200 / 201 / 204
[]


// HTTP Status 400 / 401 / 403 / 404
{
    "mensagem": "Para acessar este recurso um token de autenticação válido deve ser enviado."
}
```

Aulas úteis:

- Modelagem de Dados
- A relação um para muitos
- Criando tabelas com relacionamentos
- CRUD SQL
- Programação Assíncrona
- Funções async com await
- Revisão Programação Assíncrona
- Conexão NodeJs com PostgreSQL
- Configurando conexão com o banco
- Executando comandos SQL a partir da API
- Autenticação e Criptografia
- Revisão Módulo 3
- Login retornando token
- Filtro de autenticação lendo token do header
- Utilizando recursos com token no header
- Revisão ao vivo Módulo 3

LEMBRE-SE: Feito é melhor que perfeito!!!

tags: `back-end` `módulo 3` `nodeJS` `PostgreSQL` `API REST` `desafio`