

tryber / **sd-010-a-cookmaster** Publicgenerated from [betrybe/sd-0x-cookmaster-v2](#)

☆ 0 stars 🍴 11 forks

☆ Star

👁 Watch ▼

Code

Issues

Pull requests 126

Actions

Projects

Wiki

Security

Insights

🔗 master ▼

...



vinicius-vasconcelos ...

on 20 Sep

[View code](#)

☰ README.md



Termos e acordos

Ao iniciar este projeto, você concorda com as diretrizes do Código de Ética e Conduta e do Manual da Pessoa Estudante da Trybe

Boas vindas ao repositório do projeto Cookmaster!

Você já usa o GitHub diariamente para desenvolver os exercícios, certo? Agora, para desenvolver os projetos, você deverá seguir as instruções a seguir. Fique atento a cada passo, e se tiver qualquer dúvida, nos envie por Slack! #vqv 🚀

Aqui você vai encontrar os detalhes de como estruturar o desenvolvimento do seu projeto a partir deste repositório, utilizando uma branch específica e um Pull Request para colocar seus códigos.

Sumário

- [Boas vindas ao repositório do projeto Cookmaster!](#)
- [Habilidades](#)
- [Entregáveis](#)

- O que deverá ser desenvolvido
- Desenvolvimento
- Data de Entrega
- Instruções para entregar seu projeto
 - Antes de começar a desenvolver
 - Durante o desenvolvimento
- Como desenvolver
 - Todos os seus endpoints devem estar no padrão REST
 - Conexão com o Banco
 - Coleções
 - Linter
 - Testes
 - Dica: desativando testes
- Requisitos do projeto
 - Requisitos Obrigatórios
 - 1 - Crie um endpoint para o cadastro de usuários
 - 2 - Crie um endpoint para o login de usuários
 - 3 - Crie um endpoint para o cadastro de receitas
 - 4 - Crie um endpoint para a listagem de receitas
 - 5 - Crie um endpoint para visualizar uma receita específica
 - 6 - Crie uma query em mongo que insira uma pessoa usuária com permissões de admin
 - 7 - Crie um endpoint para a edição de uma receita
 - 8 - Crie um endpoint para a exclusão de uma receita
 - 9 - Crie um endpoint para a adição de uma imagem a uma receita
 - 10 - Crie um endpoint para acessar a imagem de uma receita
 - 11 - Crie testes de integração que cubram no mínimo 30% dos arquivos em `src` , com um mínimo de 50 linhas cobertas
 - Requisitos Bônus
 - 12 - Crie um endpoint para cadastro de pessoas administradoras
 - 13 - Crie testes de integração que cubram no mínimo 60% dos arquivos em `src` , com um mínimo de 100 linhas cobertas
 - 14 - Crie testes de integração que cubram no mínimo 90% dos arquivos em `src` , com um mínimo de 150 linhas cobertas
 - Depois de terminar o desenvolvimento (opcional)
- Revisando um pull request
- Avisos finais

Habilidades

Neste projeto, você será capaz de:

- Entender o que há por dentro de um token de autenticação;
- Gerar tokens a partir de informações como login e senha;
- Autenticar rotas do Express, usando o token JWT;
- Fazer upload de arquivos em APIs REST;
- Salvar arquivos no servidor através de uma API REST;
- Consultar arquivos do servidor através de uma api REST.
- Realizar testes de integração

Entregáveis

Para entregar o seu projeto você deverá criar um Pull Request neste repositório.

Lembre-se que você pode consultar nosso conteúdo sobre [Git & GitHub](#) sempre que precisar!

O que deverá ser desenvolvido

Você vai desenvolver seu app utilizando a arquitetura MSC!

Neste novo projeto deverá ser possível fazer o cadastro e login de pessoas usuárias, onde apenas essas pessoas poderão acessar, modificar e deletar as receitas que cadastrou.

Desenvolvimento

Você vai desenvolver todas as camadas da aplicação (Models, Service e Controllers) a partir do seu código no projeto cookmaster.

Através dessa aplicação, será possível realizar as operações básicas que se pode fazer em um determinado banco de dados: Criação, Leitura, Atualização e Exclusão (ou CRUD , para as pessoas mais íntimas 😊).

Para realizar qualquer tipo de alteração no banco de dados (como cadastro, edição ou exclusão de receitas) será necessário autenticar-se. Além disso, as pessoas usuárias devem poder ser clientes ou administradores. Pessoas clientes apenas poderão disparar ações nas receitas que ele mesmo criou. Já uma pessoa administradora pode disparar qualquer ação em qualquer receita.

A autenticação deverá ser feita via `JWT`.

O código para cadastro de pessoas usuárias deve ser criado por você utilizando os conhecimentos adquiridos nesse bloco.

Deverá ser possível adicionar uma imagem à uma receita, utilizando o upload de arquivos fornecido pelo `multer`.

⚠ Dicas Importantes ⚠:

- Não haverá front-end neste projeto, portanto não se preocupe com a visualização, mas apenas com as funcionalidades e organização do código.
- Sua API deve ser desenvolvida dentro da pasta `./src`, seus testes de integração, na pasta `./src/integration-tests`;
- Para permitir que as imagens sejam acessadas através da API, você deve utilizar o middleware `static` do `express`, da seguinte forma:

```
// ./src/api/app.js

const path = require('path');
// ...

// /images é o caminho/end-point da API onde as imagens estarão disponíveis
// path.join(__dirname, '..', 'uploads') é o caminho da pasta onde o multer
// a pasta `uploads` está em `./src/uploads` e não deve ser renomeada ou r
app.use('/images', express.static(path.join(__dirname, '..', 'uploads')));

// ...
```

Data de Entrega

- Serão `3` dias de projeto.
- Data de entrega para avaliação final do projeto: `30/09/2021 - 14:00h`.

Instruções para entregar seu projeto

Antes de começar a desenvolver

1. Clone o repositório

- `git clone https://github.com/tryber/sd-010-a-cookmaster.git` .
- Entre na pasta do repositório que você acabou de clonar:
 - `cd sd-010-a-cookmaster`

2. Instale as dependências [**Caso existam**]

- `npm install`

3. Crie uma branch a partir da branch `master`

- Verifique que você está na branch `master`
 - Exemplo: `git branch`
- Se não estiver, mude para a branch `master`
 - Exemplo: `git checkout master`
- Agora crie uma branch à qual você vai submeter os `commits` do seu projeto
 - Você deve criar uma branch no seguinte formato: `nome-de-usuario-nome-do-projeto`
 - Exemplo: `git checkout -b joaozinho-sd-010-a-cookmaster`

4. Adicione as mudanças ao `stage` do Git e faça um `commit`

- Verifique que as mudanças ainda não estão no `stage`
 - Exemplo: `git status` (deve aparecer listada a pasta *joaozinho* em vermelho)
- Adicione o novo arquivo ao `stage` do Git
 - Exemplo:
 - `git add .` (adicionando todas as mudanças - *que estavam em vermelho* - ao `stage` do Git)
 - `git status` (deve aparecer listado o arquivo *joaozinho/README.md* em verde)
- Faça o `commit` inicial
 - Exemplo:
 - `git commit -m 'iniciando o projeto x'` (fazendo o primeiro `commit`)
 - `git status` (deve aparecer uma mensagem tipo *nothing to commit*)

5. Adicione a sua branch com o novo `commit` ao repositório remoto

- Usando o exemplo anterior: `git push -u origin joaozinho-sd-010-a-cookmaster`

6. Crie um novo Pull Request (PR)

- Vá até a página de *Pull Requests* do [repositório no GitHub](#)
- Clique no botão verde "New pull request"
- Clique na caixa de seleção "Compare" e escolha a sua branch **com atenção**
- Clique no botão verde "Create pull request"
- Adicione uma descrição para o *Pull Request* e clique no botão verde "Create pull request"
- **Não se preocupe em preencher mais nada por enquanto!**
- Volte até a [página de Pull Requests do repositório](#) e confira que o seu *Pull Request* está criado

Durante o desenvolvimento

- Faça `commits` das alterações que você fizer no código regularmente.
- Lembre-se de sempre após um (ou alguns) `commits` atualizar o repositório remoto.
- Os comandos que você utilizará com mais frequência são:
 - i. `git status` (para verificar o que está em vermelho - fora do stage - e o que está em verde - no stage)
 - ii. `git add` (para adicionar arquivos ao stage do Git)
 - iii. `git commit` (para criar um commit com os arquivos que estão no stage do Git)
 - iv. `git push -u nome-da-branch` (para enviar o commit para o repositório remoto na primeira vez que fizer o `push` de uma nova branch)
 - v. `git push` (para enviar o commit para o repositório remoto após o passo anterior)

Como desenvolver

 **Leia-os atentamente e siga à risca o que for pedido.** 

Observações importantes:

- O não cumprimento de um requisito, total ou parcialmente, impactará em sua avaliação;
- O projeto deve rodar na porta **3000**;
- A testagem local depende da API estar rodando (utilize `npm run dev` para facilitar o processo);

Todos os seus endpoints devem estar no padrão REST

- Use os verbos HTTP adequados para cada operação.
- Agrupe e padronize suas URL em cada recurso.
- Garanta que seus endpoints sempre retornem uma resposta, havendo sucesso nas operações ou não.
- Retorne os códigos de status corretos (recurso criado, erro de validação, autorização, etc).

Há dois arquivos em `./src/api/` no repositório: `server.js` e `app.js`, **ambos não devem ser renomeados ou apagados**.

Em `app.js` o seguinte trecho de código não deve ser removido:

```
app.get('/', (request, response) => {  
  response.send();  
});
```

Isso está configurado para o avaliador funcionar corretamente.

Conexão com o Banco

A conexão do banco local deverá conter os seguintes parâmetros:

```
const MONGO_DB_URL = 'mongodb://localhost:27017/Cookmaster';  
const DB_NAME = 'Cookmaster';
```

Para o avaliador funcionar altere a conexão do banco para:

```
const MONGO_DB_URL = 'mongodb://mongodb:27017/Cookmaster';  
const DB_NAME = 'Cookmaster';
```

Coleções

O banco terá duas coleções: usuários e receitas.

A coleção de usuários deverá ter o seguinte nome: `users`.

Os campos da coleção `users` terão este formato:

```
{ "name" : "Erick Jacquin", "email" : "erickjacquin@gmail.com", "password" : "
```

A resposta do insert para ser retornada após a criação é esta:

```
{ "_id" : ObjectId("5f46914677df66035f61a355"), "name" : "Erick Jacquin", "ema
```

(O `_id` será gerado automaticamente pelo mongodb)

A coleção de receitas deverá ter o seguinte nome: `recipes` .

Os campos da coleção `recipes` terão este formato:

```
{ "name" : "Receita do Jacquin", "ingredients" : "Frango", "preparation" : "10
```

A resposta do insert para ser retornada após a criação é esta:

```
{ "_id" : ObjectId("5f46919477df66035f61a356"), "name" : "string", "ingredient
```

(O `_id` será gerado automaticamente pelo mongodb, e o `userId` será gerado com o id do usuário que criou a receita)

Linter

Usaremos o [ESLint](#) para fazer a análise estática do seu código.

Este projeto já vem com as dependências relacionadas ao *linter* configuradas no arquivos `package.json` .

Para poder rodar os `ESLint` em um projeto basta executar o comando `npm install` dentro do projeto e depois `npm run lint` . Se a análise do `ESLint` encontrar problemas no seu código, tais problemas serão mostrados no seu terminal. Se não houver problema no seu código, nada será impresso no seu terminal.

⚠️ PULL REQUESTS COM ISSUES DE LINTER NÃO SERÃO AVALIADAS. ATENTE-SE PARA RESOLVÊ-LAS ANTES DE FINALIZAR O DESENVOLVIMENTO! ⚠️

Você pode também instalar o plugin do `ESLint` no `VSCode` , basta ir em `extensions` e baixar o [plugin ESLint](#) .

Testes

Todos os requisitos do projeto serão testados **automaticamente**. Cada endpoint possui vários requisitos e os testes para cada requisito de um endpoint estão no arquivo de teste correspondente.

Por exemplo: Os requisitos relacionados ao endpoint `/users` estão no arquivo `users.test.js`.

Para executar os testes localmente, digite no terminal o comando `npm test`.

Inicialmente todos os testes falharão:

```
Test Suites: 3 failed, 3 total
Tests:       37 failed, 37 total
Snapshots:   0 total
Time:        2.843 s, estimated 3 s
Ran all test suites.
```

Dica: desativando testes

Especialmente no início, quando a maioria dos testes está falhando, a saída após executar os testes é bastante poluída. Você pode desabilitar temporariamente um teste utilizando a função `skip` junto à função `it`. Como o nome indica, esta função "pula" um teste:

```
it.skip('Será validado que o campo "email" é obrigatório', async () => {
  await frisby
    .post(`${url}/users/`,
      {
        name: 'Erick Jacquin',
        password: '12345678',
      })
    .expect('status', 400)
    .then((response) => {
      const { body } = response;
      const result = JSON.parse(body);
      expect(result.message).toBe('Invalid entries. Try again.');
```

Uma estratégia é pular todos os testes no início e ir implementando um teste de cada vez, removendo dele a função `skip`.

```
> sd-0x-project-cookmaster-v2@1.0.0 test
> jest --runInBand --detectOpenHandles --forceExit "users"

PASS test/users.test.js
  1 - Crie um endpoint para o cadastro de usuários
    ✓ Será validado que o campo "name" é obrigatório (62 ms)
    ✓ Será validado que o campo "email" é obrigatório (27 ms)
    ✓ Será validado que não é possível cadastrar usuário com o campo email inválido (19 ms)
    ✓ Será validado que o campo "senha" é obrigatório (20 ms)
    ✓ Será validado que o campo "email" é único (69 ms)
    ✓ Será validado que é possível cadastrar usuário com sucesso (18 ms)
    ✓ Será validado que é possível ao cadastrar usuário, o valor do campo "role" tenha o valor "user" (21 ms)
  2 - Crie um endpoint para o login de usuários
    ○ skipped Será validado que o campo "email" é obrigatório
    ○ skipped Será validado que o campo "password" é obrigatório
    ○ skipped Será validado que não é possível fazer login com um email inválido
    ○ skipped Será validado que não é possível fazer login com uma senha inválida
    ○ skipped Será validado que é possível fazer login com sucesso

Test Suites: 1 passed, 1 total
Tests:       5 skipped, 7 passed, 12 total
Snapshots:   0 total
Time:        1.698 s, estimated 2 s
Ran all test suites matching /users/i.
```

Você também pode rodar apenas um arquivo de teste, por exemplo:

```
npm test users.test.js
```

```
> sd-0x-project-cookmaster-v2@1.0.0 test
> jest --runInBand --detectOpenHandles --forceExit "users.test.js"

PASS test/users.test.js
  1 - Crie um endpoint para o cadastro de usuários
    ✓ Será validado que o campo "name" é obrigatório (61 ms)
    ✓ Será validado que o campo "email" é obrigatório (17 ms)
    ✓ Será validado que não é possível cadastrar usuário com o campo email inválido (21 ms)
    ✓ Será validado que o campo "senha" é obrigatório (18 ms)
    ✓ Será validado que o campo "email" é único (83 ms)
    ✓ Será validado que é possível cadastrar usuário com sucesso (17 ms)
    ✓ Será validado que é possível ao cadastrar usuário, o valor do campo "role" tenha o valor "user" (14 ms)
  2 - Crie um endpoint para o login de usuários
    ✓ Será validado que o campo "email" é obrigatório (15 ms)
    ✓ Será validado que o campo "password" é obrigatório (12 ms)
    ✓ Será validado que não é possível fazer login com um email inválido (14 ms)
    ✓ Será validado que não é possível fazer login com uma senha inválida (15 ms)
    ✓ Será validado que é possível fazer login com sucesso (29 ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        1.502 s
Ran all test suites matching /users.test.js/i.
```

⚠ Lembre-se de não entregar o projeto com nenhum teste ignorado. **Testes ignorados serão tratados como testes falhando.** ⚠

⚠ Não apague, em hipótese alguma, qualquer teste ou arquivo deste repositório. ⚠

Requisitos do projeto

Requisitos Obrigatórios

1 - Crie um endpoint para o cadastro de usuários

- A rota deve ser (/users).
- No banco um usuário precisa ter os campos Email, Senha, Nome e Role.
- Para criar um usuário através da API, todos os campos são obrigatórios, com exceção do Role.
- O campo Email deve ser único.
- Usuários criados através desse endpoint devem ter seu campo Role com o atributo *user*, ou seja, devem ser usuários comuns, e não admins.
- O body da requisição deve conter o seguinte formato:

```
{
  "name": "string",
  "email": "string",
  "password": "string"
}
```

- Não use `bcrypt` ou outra biblioteca para encriptar a senha, para que o avaliador funcione corretamente.

Além disso, as seguintes verificações serão feitas:

- [Será validado que o campo "name" é obrigatório]

Se o usuário não tiver o campo "name" o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/users`. The request body is a JSON object: `{ "email": "erickjacquin@gmail.com", "password": "12345678" }`. The response is a 400 Bad Request with a status of 1.92 ms and 41 B. The response body is: `{ "message": "Invalid entries. Try again." }`.

- [Será validado que o campo "email" é obrigatório]

Se o usuário não tiver o campo "email" o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/users`. The request body is a JSON object: `{ "name": "Batista", "password": "12345678" }`. The response is a 400 Bad Request with a status of 1.92 ms and 41 B. The response body is: `{ "message": "Invalid entries. Try again." }`.

- [Será validado que não é possível cadastrar usuário com o campo email inválido]

Se o usuário tiver o campo email inválido o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/users`. The request body is a JSON object: `{ "name": "Batista", "email": "brunobatista@", "password": "12345678" }`. The response is a 400 Bad Request with a status of 2.01 ms and 41 B. The response body is: `{ "message": "Invalid entries. Try again." }`.

- [Será validado que o campo "senha" é obrigatório]

Se o usuário não tiver o campo "senha" o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/users`. The request body is a JSON object: `{ "name": "Batista", "email": "erickjacquin@gmail.com" }`. The response is a 400 Bad Request with a status of 1.91 ms and 41 B. The response body is: `{ "message": "Invalid entries. Try again." }`.

- [Será validado que o campo "email" é único]

Se o usuário cadastrar o campo "email" com um email que já existe, o resultado retornado deverá ser conforme exibido abaixo, com um status http 409 :

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/users`. The request body is a JSON object: `{ "name": "Batista", "email": "erickjacquin@gmail.com", "password": "12345678" }`. The response is a 409 Conflict with a status of 4.13 ms and 38 B. The response body is: `{ "message": "Email already registered" }`.

- [Será validado que é possível cadastrar usuário com sucesso]

Se o usuário for cadastrado com sucesso o resultado retornado deverá ser conforme exibido abaixo, com um status http 201 :

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/users`. The request body is a JSON object: `{ "name": "Batista", "email": "brunobatista@gmail.com", "password": "12345678" }`. The response is a 201 Created with a status of 3.58 ms and 107 B. The response body is: `{ "user": { "name": "Batista", "email": "brunobatista@gmail.com", "role": "user", "_id": "5f4697be77df66035f61a357" } }`.

- [Será validado que é possível ao cadastrar usuário, o valor do campo "role" tenha o valor "user"]

Se o usuário for criado com sucesso o resultado retornado deverá ser conforme exibido abaixo, com um status http 201 :

```
POST http://localhost:3000/users Send 201 Created 3.58 ms 107 B Just Now
```

```
JSON Auth Query Header 1 Docs
```

```
1 {
2   "name": "Batista",
3   "email": "brunobatista@gmail.com",
4   "password": "12345678"
5 }
```

```
Preview Header 6 Cookie Timeline
```

```
1 {
2   "user": {
3     "name": "Batista",
4     "email": "brunobatista@gmail.com",
5     "role": "user",
6     "_id": "5f4697be77df66035f61a357"
7   }
8 }
```

2 - Crie um endpoint para o login de usuários

- A rota deve ser (/login).
- A rota deve receber os campos Email e Senha e esses campos devem ser validados no banco de dados.
- Na configuração do JWT **não use variáveis de ambientes** para não ter conflito com o avaliador.
- Um token JWT deve ser gerado e retornado caso haja sucesso no login. No seu payload deve estar presente o id, email e role do usuário.
- O body da requisição deve conter o seguinte formato:

```
{
  "email": "string",
  "password": "string"
}
```

Além disso, as seguintes verificações serão feitas:

- [Será validado que o campo "email" é obrigatório]

Se o login não tiver o campo "email" o resultado retornado deverá ser conforme exibido abaixo, com um status http 401 :

```
POST http://localhost:3000/login Send 401 Unauthorized 1.99 ms 39 B Just Now
```

```
JSON Auth Query Header 1 Docs
```

```
1 {
2   "password": "12345678"
3 }
```

```
Preview Header 6 Cookie Timeline
```

```
1 {
2   "message": "All fields must be filled"
3 }
```

- [Será validado que o campo "password" é obrigatório]

Se o login não tiver o campo "password" o resultado retornado deverá ser conforme exibido abaixo, com um status http 401 :

POST http://localhost:3000/login Send 401 Unauthorized 1.03 ms 39 B A Minute Ago

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "erickjacquin@gmail.com"
3 }
```

Preview Header 6 Cookie Timeline

```
1 {
2   "message": "All fields must be filled"
3 }
```

- [Será validado que não é possível fazer login com um email inválido]

Se o login tiver o email inválido o resultado retornado deverá ser conforme exibido abaixo, com um status http 401 :

POST http://localhost:3000/login Send 401 Unauthorized 10.2 ms 44 B Just Now

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "erickjacquin@",
3   "password": "12345678"
4 }
```

Preview Header 6 Cookie Timeline

```
1 {
2   "message": "Incorrect username or password"
3 }
```

- [Será validado que não é possível fazer login com uma senha inválida]

Se o login tiver a senha inválida o resultado retornado deverá ser conforme exibido abaixo, com um status http 401 :

POST http://localhost:3000/login Send 401 Unauthorized 1.99 ms 44 B Just Now

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "erickjacquin@gmail.com",
3   "password": "12"
4 }
```

Preview Header 6 Cookie Timeline

```
1 {
2   "message": "Incorrect username or password"
3 }
```

- [Será validado que é possível fazer login com sucesso]

Se foi feito login com sucesso o resultado retornado deverá ser conforme exibido abaixo, com um status http 200 :

POST http://localhost:3000/login Send 200 OK 3.19 ms 259 B Just Now

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "erickjacquin@gmail.com",
3   "password": "12345678"
4 }
```

Preview Header 6 Cookie Timeline

```
1 {
2   "token":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7Ii9pZCI6IjV
    mNDY5MTQ2NzdkZjY2MDM1ZjYxYTM1NSIsImVtYWlsIjo1ZXJpY2tqYmNxdWlu
    QGdtYWlsLmNvbSIsInJvbGU101j1c2VyIn0sImhhdCI6MTU5ODQ2NTIyMSw1Z
    XhwIjo5NTk4NDcyNDIxfQ.kuaGJIZHMHG5Xb7h4g1I-
    goSpz4QnP_Ki5MpmN4Q3Gs"
3 }
```

3 - Crie um endpoint para o cadastro de receitas

- A rota deve ser (/recipes).
- A receita só pode ser criada caso o usuário esteja logado e o token JWT validado.
- No banco, a receita deve ter os campos Nome, Ingredientes, Modo de preparo, URL da imagem e Id do Autor.

- Nome, ingredientes e modo de preparo devem ser recebidos no corpo da requisição, com o seguinte formato:

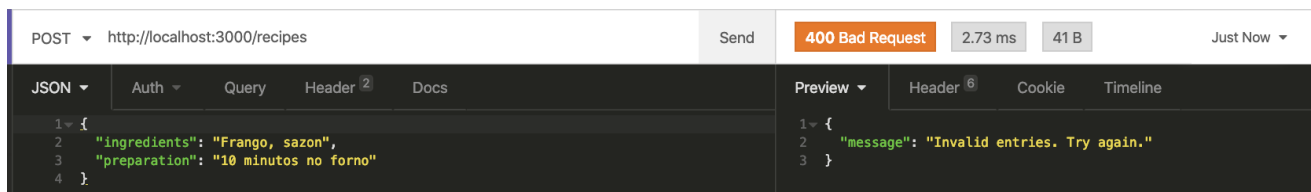
```
{
  "name": "string",
  "ingredients": "string",
  "preparation": "string"
}
```

- O campo dos ingredientes pode ser um campo de texto aberto.
- O campo ID do autor, deve ser preenchido automaticamente com o ID do usuário logado, que deve ser extraído do token JWT.
- A URL da imagem será preenchida através de outro endpoint

Além disso, as seguintes verificações serão feitas:

- [Será validado que não é possível cadastrar receita sem o campo "name"]**

Se a receita não tiver o campo "name" o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :



POST http://localhost:3000/recipes

Send 400 Bad Request 2.73 ms 41 B Just Now

JSON Auth Query Header 2 Docs

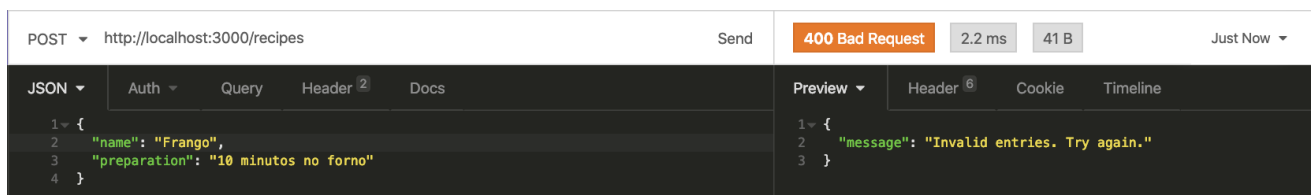
Preview Header 6 Cookie Timeline

```
1 {
2   "ingredients": "Frango, sazão",
3   "preparation": "10 minutos no forno"
4 }
```

```
1 {
2   "message": "Invalid entries. Try again."
3 }
```

- [Será validado que não é possível cadastrar receita sem o campo "ingredients"]**

Se a receita não tiver o campo "ingredients" o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :



POST http://localhost:3000/recipes

Send 400 Bad Request 2.2 ms 41 B Just Now

JSON Auth Query Header 2 Docs

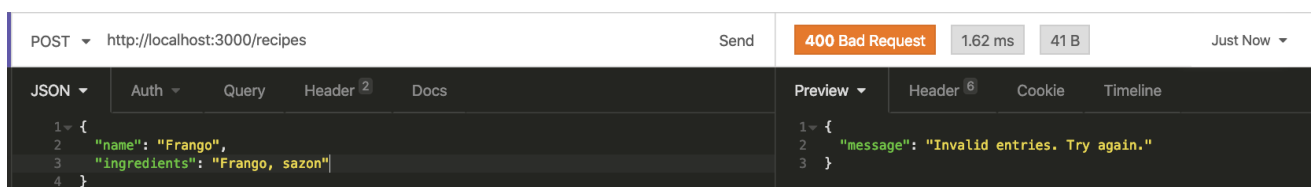
Preview Header 6 Cookie Timeline

```
1 {
2   "name": "Frango",
3   "preparation": "10 minutos no forno"
4 }
```

```
1 {
2   "message": "Invalid entries. Try again."
3 }
```

- [Será validado que não é possível cadastrar receita sem o campo "preparation"]**

Se a receita não tiver o campo "preparation" o resultado retornado deverá ser conforme exibido abaixo, com um status http 400 :



POST http://localhost:3000/recipes

Send 400 Bad Request 1.62 ms 41 B Just Now

JSON Auth Query Header 2 Docs

Preview Header 6 Cookie Timeline

```
1 {
2   "name": "Frango",
3   "ingredients": "Frango, sazão"
4 }
```

```
1 {
2   "message": "Invalid entries. Try again."
3 }
```


- [Será validado que não é possível cadastrar uma receita com token invalido]

Se a receita não tiver o token válido o resultado retornado deverá ser conforme exibido abaixo, com um status http 401 :

Insomnia - cadastro de receitas

POST http://localhost:3000/recipes Send 401 Unauthorized 1.35 ms Just Now

JSON Auth Query Header 2 Docs Preview Header 6 Cookie Timeline

Authorization erer

Content-Type application/json

```
1 {
2   "message": "jwt malformed"
3 }
```

- [Será validado que é possível cadastrar uma receita com sucesso]

O resultado retornado para cadastrar a receita com sucesso deverá ser conforme exibido abaixo, com um status http 201 :

POST http://localhost:3000/recipes Send 201 Created 2.38 ms 163 B Just Now

JSON Auth Query Header 2 Docs Preview Header 6 Cookie Timeline

```
1 {
2   "name": "Frango",
3   "ingredients": "Frango, sazón",
4   "preparation": "10 minutos no forno"
5 }
```

```
1 {
2   "recipe": {
3     "name": "Frango",
4     "ingredients": "Frango, sazón",
5     "preparation": "10 minutos no forno",
6     "userId": "5f46a9b577df66035f61a36d",
7     "_id": "5f46a9c177df66035f61a36e"
8   }
9 }
```

4 - Crie um endpoint para a listagem de receitas

- A rota deve ser (/recipes).
- A rota pode ser acessada por usuários logados ou não

Além disso, as seguintes verificações serão feitas:

- [Será validado que é possível listar todas as receitas sem estar autenticado]

O resultado retornado para listar receitas com sucesso deverá ser conforme exibido abaixo, com um status http 200 :

GET http://localhost:3000/recipes Send 200 OK 2.64 ms 307 B Just Now

Body Auth Query Header 2 Docs Preview Header 6 Cookie Timeline

```
1 [
2   {
3     "_id": "5f46a9c177df66035f61a36e",
4     "name": "Frango",
5     "ingredients": "Frango, sazón",
6     "preparation": "10 minutos no forno",
7     "userId": "5f46a9b577df66035f61a36d"
8   },
9   {
10    "_id": "5f46aa6177df66035f61a36f",
11    "name": "Frango",
12    "ingredients": "Frango, sazón",
13    "preparation": "10 minutos no forno",
14    "userId": "5f46a9b577df66035f61a36d"
15  }
16 ]
```


- **[Será validado que é possível listar todas as receitas estando autenticado]**

O resultado retornado para listar receitas com sucesso deverá ser conforme exibido abaixo, com um status http 200 :

```

GET http://localhost:3000/recipes 200 OK 2.64 ms 307 B Just Now
Body Auth Query Header 2 Docs Preview Header 6 Cookie Timeline
1 [
2   {
3     "_id": "5f46a9c177df66035f61a36e",
4     "name": "Frango",
5     "ingredients": "Frango, sazón",
6     "preparation": "10 minutos no forno",
7     "userId": "5f46a9b577df66035f61a36d"
8   },
9   {
10    "_id": "5f46aa6177df66035f61a36f",
11    "name": "Frango",
12    "ingredients": "Frango, sazón",
13    "preparation": "10 minutos no forno",
14    "userId": "5f46a9b577df66035f61a36d"
15  }
16 ]
  
```

5 - Crie um endpoint para visualizar uma receita específica

- A rota deve ser (/recipes/:id).
- A rota pode ser acessada por usuários logados ou não

Além disso, as seguintes verificações serão feitas:

- **[Será validado que é possível listar uma receita específica sem estar autenticado]**

O resultado retornado para listar uma receita com sucesso deverá ser conforme exibido abaixo, com um status http 200 :

```

GET http://localhost:3000/recipes/5f46a9c177df66035f61a36e 200 OK 2.56 ms 152 B Just Now
Body Auth Query Header 2 Docs Preview Header 6 Cookie Timeline
1 {
2   "_id": "5f46a9c177df66035f61a36e",
3   "name": "Frango",
4   "ingredients": "Frango, sazón",
5   "preparation": "10 minutos no forno",
6   "userId": "5f46a9b577df66035f61a36d"
7 }
  
```

- **[Será validado que é possível listar uma receita específica estando autenticado]**

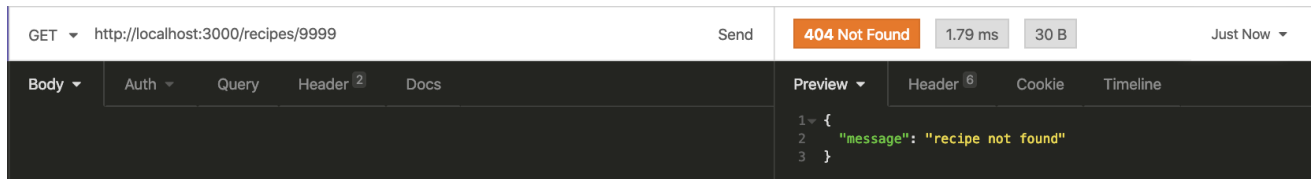
O resultado retornado para listar uma receita com sucesso deverá ser conforme exibido abaixo, com um status http 200 :

```

GET http://localhost:3000/recipes/5f46a9c177df66035f61a36e 200 OK 2.56 ms 152 B Just Now
Body Auth Query Header 2 Docs Preview Header 6 Cookie Timeline
1 {
2   "_id": "5f46a9c177df66035f61a36e",
3   "name": "Frango",
4   "ingredients": "Frango, sazón",
5   "preparation": "10 minutos no forno",
6   "userId": "5f46a9b577df66035f61a36d"
7 }
  
```

- **[Será validado que não é possível listar uma receita que não existe]**

O resultado retornado para listar uma receita que não existe deverá ser conforme exibido abaixo, com um status http 404 :



6 - Crie uma query em mongo que insira uma pessoa usuária com permissões de admin

Crie um arquivo `seed.js` na raiz do projeto com uma query do Mongo DB capaz de inserir um usuário na coleção `users` com os seguintes valores:

```
{ name: 'admin', email: 'root@email.com', password: 'admin', role: 'admin' }
```

Obs.: Esse usuário tem o poder de criar, deletar, atualizar ou remover qualquer receita, independente de quem a cadastrou. Isso será solicitado ao longo dos próximos requisitos.

Além disso, as seguintes verificações serão feitas:

- [Será validado que o projeto tem um arquivo de seed, com um comando para inserir um usuário root e verifico que é possível fazer login]

Será validado no arquivo `seed.js` existe a query para criar um usuário root

7 - Crie um endpoint para a edição de uma receita

- A rota deve ser (`/recipes/:id`).
- A receita só pode ser atualizada caso o usuário esteja logado e o token `JWT` validado.
- A receita só pode ser atualizada caso pertença ao usuário logado, ou caso esse usuário seja um admin.
- O corpo da requisição deve receber o seguinte formato:

```
{
  "name": "string",
  "ingredients": "string",
  "preparation": "string"
}
```

Além disso, as seguintes verificações serão feitas:

- [Será validado que não é possível editar receita sem estar autenticado]

O resultado retornado para editar receita sem autenticação deverá ser conforme exibido abaixo, com um status http 401 :

The screenshot shows a REST client interface. The top bar indicates a PUT request to `http://localhost:3000/recipes/5f46a9c177df66035f61a36e` with a status of **401 Unauthorized**, a response time of 1.26 ms, and a body size of 32 B. The request body is a JSON object: `{ "name": "editar", "ingredients": "string", "preparation": "string" }`. The response body is: `{ "message": "missing auth token" }`.

- [Será validado que não é possível editar receita com token inválido]

O resultado retornado para editar receita com token inválido deverá ser conforme exibido abaixo, com um status http 401 :

The screenshot shows a REST client interface. The top bar indicates a PUT request to `http://localhost:3000/recipes/5f46a9c177df66035f61a36e` with a status of **401 Unauthorized**, a response time of 1.96 ms, and a body size of 27 B. The request body is the same JSON object as before. The response body is: `{ "message": "jwt malformed" }`.

- [Será validado que é possível editar receita estando autenticado]

O resultado retornado para editar uma receita com sucesso deverá ser conforme exibido abaixo, com um status http 200 :

The screenshot shows a REST client interface. The top bar indicates a PUT request to `http://localhost:3000/recipes/5f46a9c177df66035f61a36e` with a status of **200 OK**, a response time of 6.38 ms, and a body size of 132 B. The request body is the same JSON object. The response body is a JSON object: `{ "_id": "5f46a9c177df66035f61a36e", "name": "editar", "ingredients": "string", "preparation": "string", "userId": "5f46a9b577df66035f61a36d" }`.

- [Será validado que é possível editar receita com usuário admin]

O resultado retornado para editar uma receita com sucesso deverá ser conforme exibido abaixo, com um status http 200 :

This screenshot is identical to the previous one, showing a successful 200 OK response for the PUT request to edit a recipe.

8 - Crie um endpoint para a exclusão de uma receita

- A rota deve ser (/recipes/:id).
- A receita só pode ser excluída caso o usuário esteja logado e o token JWT validado.

- A receita só pode ser excluída caso pertença ao usuário logado, ou caso o usuário logado seja um admin.

Além disso, as seguintes verificações serão feitas:

- [Será validado que não é possível excluir receita sem estar autenticado]

O resultado retornado para excluir uma receita sem autenticação deverá ser conforme exibido abaixo, com um status http 401 :

DELETE ▼ http://localhost:3000/recipes/5f46a9c177df66035f61a36e					Send	401 Unauthorized	1.79 ms	32 B	Just Now ▼
Body ▼	Auth ▼	Query	Header ²	Docs	Preview ▼	Header ⁶	Cookie	Timeline	
					<pre>1 - { 2 "message": "missing auth token" 3 }</pre>				

- [Será validado que é possível excluir receita estando autenticado]

O resultado retornado para excluir uma receita com sucesso deverá ser conforme exibido abaixo, com um status http 204 :

DELETE ▾		http://localhost:3000/recipes/5f46a9c177df66035f61a36e		Send	204 No Content	3.41 ms	0 B	Just Now ▾
Body ▾	Auth ▾	Query	Header ²	Docs	Preview ▾	Header ⁴	Cookie	Timeline
					No body returned for response			

- [Será validado que é possível excluir receita com usuário admin]

O resultado retornado para excluir uma receita com sucesso deverá ser conforme exibido abaixo, com um status http 204 :

DELETE ▾	http://localhost:3000/recipes/5f46a9c177df66035f61a36e	Send	204 No Content	3.41 ms	0 B	Just Now ▾		
Body ▾	Auth ▾	Query	Header ²	Docs	Preview ▾	Header ⁴	Cookie	Timeline
					No body returned for response			

9 - Crie um endpoint para a adição de uma imagem a uma receita

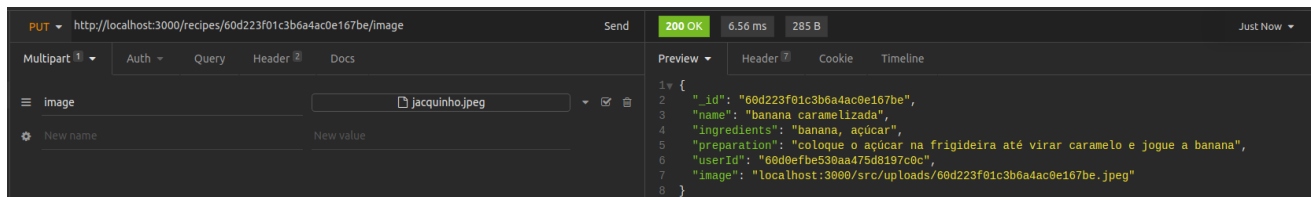
- A rota deve ser (/recipes/:id/image/).
- A imagem deve ser lida do campo image .
- O endpoint deve aceitar requisições no formato multipart/form-data .
- A receita só pode ser atualizada caso o usuário esteja logado e o token jwt validado.
- A receita só pode ser atualizada caso pertença ao usuário logado ou caso o usuário logado seja admin.

- O upload da imagem deverá ser feito utilizando o `Multer`.
- O nome do arquivo deve ser o ID da receita, e sua extensão `.jpeg`.
- A URL completa para acessar a imagem através da API deve ser gravada no banco de dados, junto com os dados da receita.

Além disso, as seguintes verificações serão feitas:

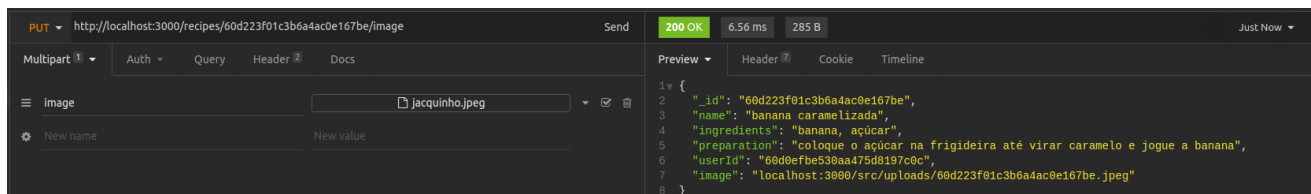
- [Será validado que é possível enviar foto com usuário autenticado]

O resultado retornado para adicionar uma foto na receita com sucesso deverá ser conforme exibido abaixo, com um status `http 200`:



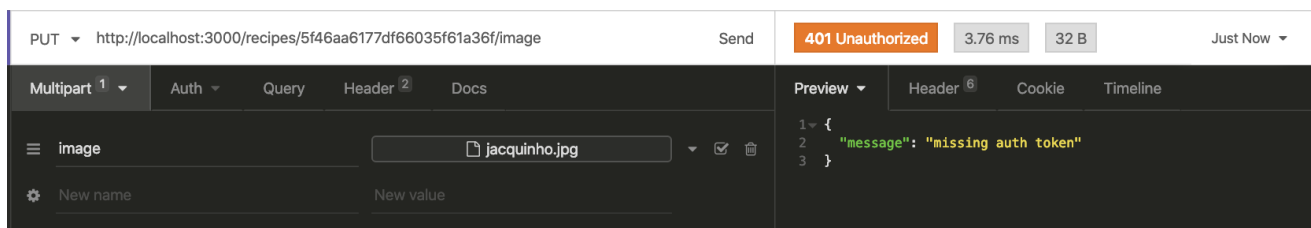
- [Será validado que ao enviar foto, o nome da imagem é alterada para o id da receita]

O resultado retornado para adicionar uma foto na receita com sucesso deverá ser conforme exibido abaixo, com um status `http 200`:



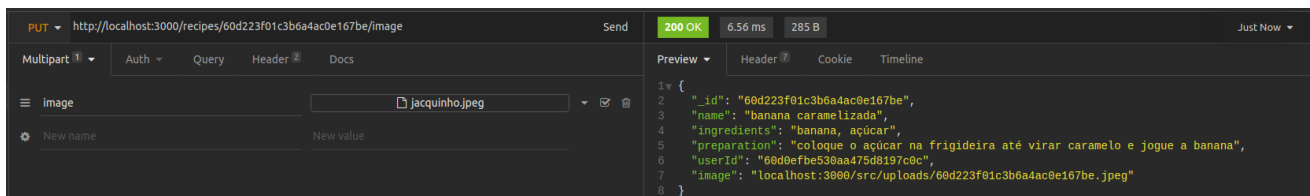
- [Será validado que não é possível enviar foto sem estar autenticado]

O resultado retornado para adicionar uma foto na receita com sucesso deverá ser conforme exibido abaixo, com um status `http 401`:



- [Será validado que é possível enviar foto com usuário admin]

O resultado retornado para adicionar uma foto na receita com sucesso deverá ser conforme exibido abaixo, com um status `http 200`:



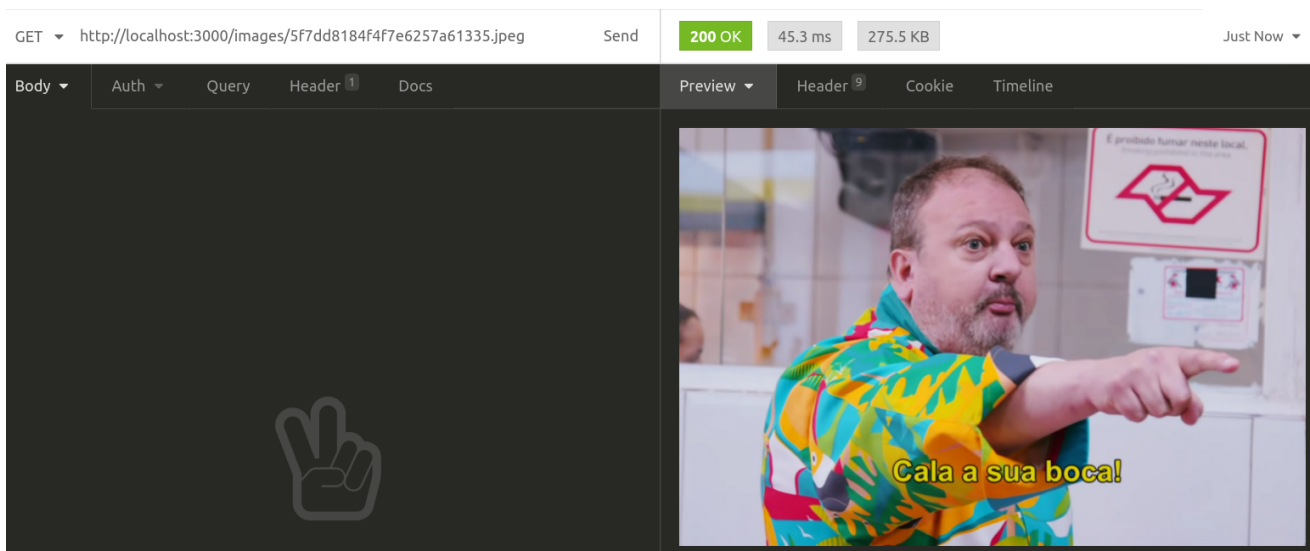
10 - Crie um endpoint para acessar a imagem de uma receita

- As imagens devem estar disponíveis através da rota `/images/<id-da-receita>.jpeg` na API.

Além disso, as seguintes verificações serão feitas:

- [Será validado que é retornada uma imagem como resposta]

O resultado retornado deverá ser do tipo imagem, com um status http `200` :



11 - Crie testes de integração que cubram no mínimo 30% dos arquivos em `src`, com um mínimo de 50 linhas cobertas

- Os testes de integração devem ser criados na pasta `./src/integration-tests`, essa pasta **não pode ser renomeada ou removida**;
- O arquivo `change.me.test.js` pode ser alterado, renomeado ou removido;
- Os testes devem ser criados usando o instrumental e boas práticas apresentado nos conteúdos de testes do course;
- Para rodar os testes, utilize o comando `npm run dev:test`;
- Para visualizar a cobertura, utilize o comando `npm run dev:test:coverage`;

Além disso, as seguintes verificações serão feitas:

- [Será validado que o teste cobre o valor esperado]

Nenhum teste pode ser pulado; O resultado do percentual total de cobertura deve ser igual ou maior que 30 ; O resultado do numero total de linhas cobertas deve ser igual ou maior que 50 .

Requisitos Bônus

12 - Crie um endpoint para cadastro de pessoas administradoras

- A rota deve ser (/users/admin).
- Só será possível adicionar um admin caso esta ação esteja sendo feita por outro admin, portanto, deve ser validado se há um admin logado.
- Por padrão, as requisições pra esse endpoint devem adicionar um usuário com a role *admin*.
- O corpo da requisição deve ter o seguinte formato:

```
{
  "name": "string",
  "email": "string",
  "password": "string"
}
```

Além disso, as seguintes verificações serão feitas:

- [Será validado que não é possível cadastrar um usuário admin, sem estar autenticado como um usuário admin]

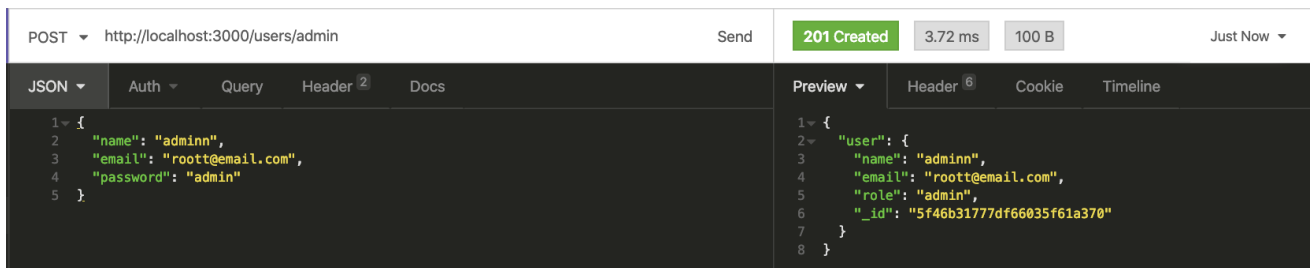
Se o usuário admin não é criado com sucesso o resultado retornado deverá ser conforme exibido abaixo, com um status http 403 :

The screenshot shows a REST client interface. The top bar indicates a POST request to `http://localhost:3000/users/admin` with a status of **403 Forbidden**, a response time of 2.56 ms, and a body size of 49 B. The left sidebar shows the request headers: `Authorization: yHKUx8x7eCZA588WzWbTGgOt0a8zQmTg` and `Content-Type: application/json`. The right pane shows the response body in JSON format:

```
1 {
2   "message": "Only admins can register new admins"
3 }
```

- [Será validado que é possível cadastrar um usuário admin]

Se o usuário admin é criado com sucesso o resultado retornado deverá ser conforme exibido abaixo, com um status http 201 :



13 - Crie testes de integração que cubram no mínimo 60% dos arquivos em `src` , com um mínimo de 100 linhas cobertas

- Os testes de integração devem ser criados na pasta `./src/integration-tests` , essa pasta **não pode ser renomeada ou removida**;
- O arquivo `change.me.test.js` pode ser alterado, renomeado ou removido;
- Os testes devem ser criados usando o instrumental e boas práticas apresentado nos conteúdos de testes do course;
- Para rodar os testes, utilize o comando `npm run dev:test` ;
- Para visualizar a cobertura, utilize o comando `npm run dev:test:coverage` ;

Além disso, as seguintes verificações serão feitas:

- [Será validado que o teste cobre o valor esperado]

Nenhum teste pode ser pulado; O resultado do percentual total de cobertura deve ser igual ou maior que `60` ; O resultado do numero total de linhas cobertas deve ser igual ou maior que `100` .

14 - Crie testes de integração que cubram no mínimo 90% dos arquivos em `src` , com um mínimo de 150 linhas cobertas

- Os testes de integração devem ser criados na pasta `./src/integration-tests` , essa pasta **não pode ser renomeada ou removida**;
- O arquivo `change.me.test.js` pode ser alterado, renomeado ou removido;
- Os testes devem ser criados usando o instrumental e boas práticas apresentado nos conteúdos de testes do course;
- Para rodar os testes, utilize o comando `npm run dev:test` ;
- Para visualizar a cobertura, utilize o comando `npm run dev:test:coverage` ;

Além disso, as seguintes verificações serão feitas:

- [Será validado que o teste cobre o valor esperado]

Nenhum teste pode ser pulado; O resultado do percentual total de cobertura deve ser igual ou maior que 90 ; O resultado do numero total de linhas cobertas deve ser igual ou maior que 150 .

Depois de terminar o desenvolvimento (opcional)

Para sinalizar que o seu projeto está pronto para o "Code Review" dos seus colegas, faça o seguinte:

- Vá até a página **DO SEU Pull Request**, adicione a label de "code-review" e marque seus colegas:
 - No menu à direita, clique no link "**Labels**" e escolha a label **code-review**;
 - No menu à direita, clique no link "**Assignees**" e escolha **o seu usuário**;
 - No menu à direita, clique no link "**Reviewers**" e digite `students` , selecione o time `tryber/students-sd-00` .

Caso tenha alguma dúvida, [aqui tem um video explicativo](#).

Revisando um pull request

Use o conteúdo sobre [Code Review](#) para te ajudar a revisar os *Pull Requests*.

#VQV

Avisos finais

Ao finalizar e submeter o projeto, não se esqueça de avaliar sua experiência preenchendo o formulário. Leva menos de 3 minutos!

Link: [FORMULÁRIO DE AVALIAÇÃO DE PROJETO](#)

O avaliador automático não necessariamente avalia seu projeto na ordem em que os requisitos aparecem no readme. Isso acontece para deixar o processo de avaliação mais rápido. Então, não se assuste se isso acontecer, ok?



Releases

Packages

No packages published

Publish your first package

Contributors 2

-  **jeanpsv** Jean Paulo Silva Vasconcelos
-  **vinicius-vasconcelos** Vinicius-Vasconcelos

Languages

-  **JavaScript** 100.0%