



Ignite



Node.js



Desafio 02 - Trabalhando com middlewares

Sobre o desafio

Template da aplicação

Middlewares da aplicação

checksExistsUserAccount

checksCreateTodosUserAvailability

checksTodoExists

findUserById

Especificação dos testes

Testes dos middlewares

17 Entrega



Sobre o desafio

Nesse desafio você irá trabalhar mais a fundo com middlewares no Express. Dessa forma você será capaz de fixar mais ainda os conhecimentos obtidos até agora.

Para facilitar um pouco mais do conhecimento da regra de negócio, você irá trabalhar com a mesma aplicação do desafio anterior: uma aplicação para gerenciar tarefas (ou *todos*) mas com algumas mudanças.

Será permitida a criação de um usuário com `name` e `username`, bem como fazer o CRUD de *todos*:

- Criar um novo *todo*;
- Listar todos os *todos*;
- Alterar o `title` e `deadline` de um *todo* existente;
- Marcar um *todo* como feito;
- Excluir um *todo*;

Tudo isso para cada usuário em específico. Além disso, dessa vez teremos um plano grátis onde o usuário só pode criar até dez *todos* e um plano Pro que irá permitir criar *todos* ilimitados, isso tudo usando middlewares para fazer as validações necessárias.

A seguir veremos com mais detalhes o que e como precisa ser feito 🚀

Template da aplicação

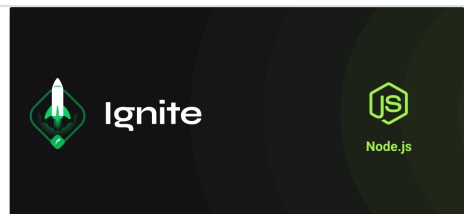
Para realizar esse desafio, criamos para você esse modelo que você deve utilizar como um template do GitHub.

O template está disponível na seguinte URL:

rocketseat-education/ignite-template-trabalhando-co...

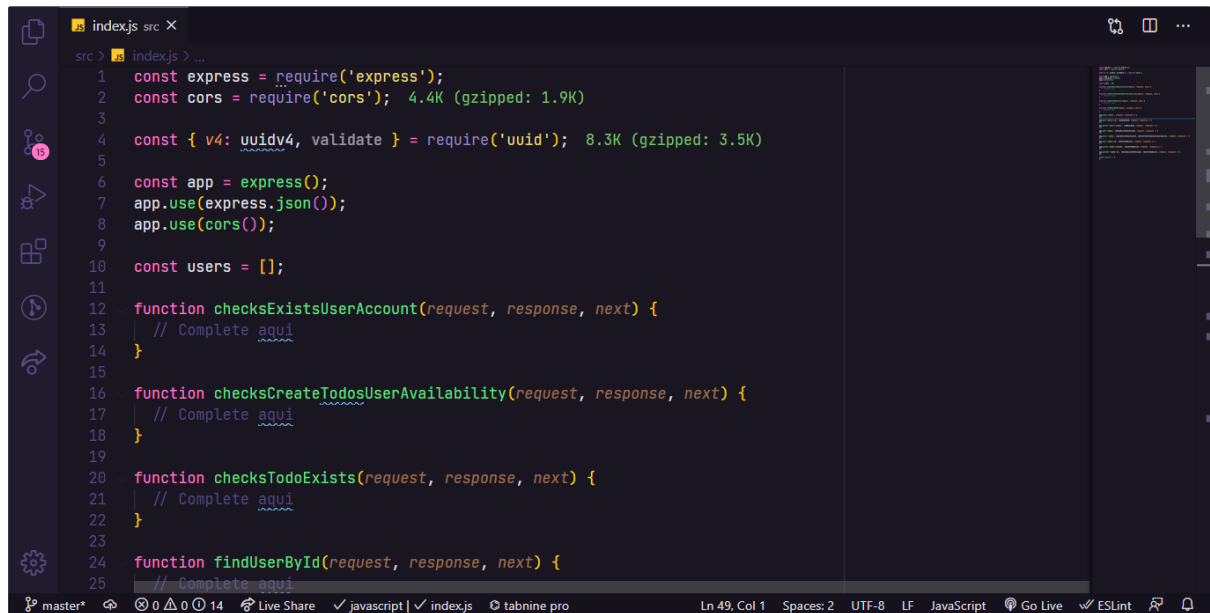
Ignite] Desafio 02 - Trilha Node.js. Contribute to rocketseat-education/ignite-template-trabalhando-com-middlewares

 <https://github.com/rocketseat-education/ignite-template-t...>



Dica: Caso não saiba utilizar repositórios do GitHub como template, temos um guia em [nosso FAQ](#).

Agora navegue até a pasta criada, abra no Visual Studio Code e por último abra o arquivo `index.js`. Lembre-se de executar o comando `yarn` no seu terminal para instalar todas as dependências e você terá o seguinte código:



```
1 const express = require('express');
2 const cors = require('cors');
3
4 const { v4: uuidv4, validate } = require('uuid');
5
6 const app = express();
7 app.use(express.json());
8 app.use(cors());
9
10 const users = [];
11
12 function checksExistsUserAccount(request, response, next) {
13   // Complete aqui
14 }
15
16 function checksCreateTodosUserAvailability(request, response, next) {
17   // Complete aqui
18 }
19
20 function checksTodoExists(request, response, next) {
21   // Complete aqui
22 }
23
24 function findUserById(request, response, next) {
25   // Complete aqui
26 }
```

Middlewares da aplicação

Com o template já clonado e o arquivo `index.js` aberto, você deve completar onde não possui código com o código para atingir os objetivos de cada teste.

Nesse desafio não será necessário alterar o código de nenhuma rota, **apenas dos middlewares**. Os testes irão também testar o funcionamento das rotas mas o resultado depende apenas da dos middlewares.

Aqui teremos uma breve descrição do que cada middleware deve fazer e na seção Especificação dos testes você verá com mais detalhes o que precisa ser feito para satisfazer cada teste.

checksExistsUserAccount

Esse middleware é responsável por receber o username do usuário pelo header e validar se existe ou não um usuário com o username passado. Caso exista, o usuário deve ser repassado para o request e a função next deve ser chamada.

checksCreateTodosUserAvailability

Esse middleware deve receber o **usuário** já dentro do request e chamar a função next apenas se esse usuário ainda estiver no **plano grátis** e **ainda não possuir 10 todos** cadastrados ou se ele já estiver com o **plano Pro** ativado.

checksTodoExists

Esse middleware deve receber o **username** de dentro do header e o **id** de um *todo* de dentro de `request.params`. Você deve validar o usuário, validar que o **id** seja um uuid e também validar que esse **id** pertence a um *todo* do usuário informado.

Com todas as validações passando, o *todo* encontrado deve ser passado para o `request` assim como o usuário encontrado também e a função `next` deve ser chamada.

findUserById

Esse middleware possui um funcionamento semelhante ao middleware `checksExistsUserAccount` mas a busca pelo usuário deve ser feita através do `id` de um usuário passado por parâmetro na rota. Caso o usuário tenha sido encontrado, o mesmo deve ser repassado para dentro do `request.user` e a função `next` deve ser chamada.

Especificação dos testes

Em cada teste, tem uma breve descrição no que sua aplicação deve cumprir para que o teste passe.



Caso você tenha dúvidas quanto ao que são os testes, e como interpretá-los, dê uma olhada em [nosso FAQ](#)

Para esse desafio, temos os seguintes testes:

Testes dos middlewares

- **Should be able to find user by username in header and pass it to request.user**

Para que esse teste passe, você deve permitir que o middleware `checksExistsUserAccount` receba um `username` pelo header do request e caso um usuário com o mesmo `username` exista, ele deve ser colocado dentro de `request.user` e, ao final, retorne a chamada da função `next`.

Atente-se bem para o nome da propriedade que armazenará o objeto `user` no request.

- **Should not be able to find a non existing user by username in header**

Para que esse teste passe, no middleware `checksExistsUserAccount` você deve retornar uma resposta com status `404` caso o `username` passado pelo header da requisição não pertença a nenhum usuário. Você pode também retornar uma mensagem de erro mas isso é opcional.

- **Should be able to let user create a new todo when is in free plan and have less than ten todos**

Para que esse teste passe, você deve permitir que o middleware `checksCreateTodosUserAvailability` receba o objeto `user` (considere sempre que o objeto existe) da `request` e chame a função `next` somente no caso do usuário estar no **plano grátis e ainda não possuir 10 *todos* cadastrados** ou se ele **já estiver com o plano Pro ativado**.

💡 Você pode verificar se o usuário possui um plano Pro ou não a partir da propriedade `user.pro`. Caso seja `true` significa que o plano Pro está em uso.

- **Should not be able to let user create a new todo when is not Pro and already have ten todos**

Para que esse teste passe, no middleware `checksCreateTodosUserAvailability` você deve retornar uma resposta com status `403` caso o usuário recebido pela requisição esteja no **plano grátis e já tenha 10 *todos* cadastrados**. Você pode também retornar uma mensagem de erro mas isso é opcional.

- **Should be able to let user create infinite new todos when is in Pro plan**

Para que esse teste passe, você deve permitir que o middleware `checksCreateTodosUserAvailability` receba o objeto `user` (considere sempre que o objeto existe) da `request` e chame a função `next` caso o usuário já esteja com o plano Pro.

💡 Se você satisfizer os dois testes anteriores antes desse, ele já deve passar também.

- **Should be able to put user and todo in request when both exists**

Para que esse teste passe, o middleware `checksTodoExists` deve receber o `username` de dentro do header e o `id` de um *todo* de dentro de `request.params`. Você deve validar que o usuário exista, validar que o `id` seja um uuid e também validar que esse `id` pertence a um *todo* do usuário informado.

Com todas as validações passando, o *todo* encontrado deve ser passado para o `request` assim como o usuário encontrado também e a função `next` deve ser chamada.

É importante que você coloque dentro de `request.user` o usuário encontrado e dentro de `request.todo` o *todo* encontrado.

- **Should not be able to put user and todo in request when user does not exists**

Para que esse teste passe, no middleware `checksTodoExists` você deve retornar uma resposta com status `404` caso não exista um usuário com o `username` passado pelo header da requisição.

- **Should not be able to put user and todo in request when todo id is not uuid**

Para que esse teste passe, no middleware `checksTodoExists` você deve retornar uma resposta com status `400` caso o `id` do `todo` passado pelos parâmetros da requisição não seja um UUID válido (por exemplo `1234abcd`).

- **Should not be able to put user and todo in request when todo does not exists**

Para que esse teste passe, no middleware `checksTodoExists` você deve retornar uma resposta com status `404` caso o `id` do `todo` passado pelos parâmetros da requisição não pertença a nenhum `todo` do usuário encontrado.

- **Should be able to find user by id route param and pass it to request.user**

Para que esse teste passe, o middleware `findUserById` deve receber o `id` de um usuário de dentro do `request.params`. Você deve validar que o usuário exista, repassar ele para `request.user` e retornar a chamada da função `next`.

- **Should not be able to pass user to request.user when it does not exists**

Para que esse teste passe, no middleware `findUserById` você deve retornar uma resposta com status `404` caso o `id` do usuário passado pelos parâmetros da requisição não pertença a nenhum usuário cadastrado.

Todos os demais testes são os mesmos testes encontrados no desafio 01 com algumas (ou nenhuma) mudanças.



Vale reforçar que esse desafio é focado apenas em middlewares e você não precisa modificar o conteúdo das rotas para que os testes passem 💜



Entrega

Esse desafio deve ser entregue a partir da plataforma da Rocketseat. Envie o link do repositório que você fez suas alterações. Após concluir o desafio, além de ter mandado o código para o GitHub, fazer um post no LinkedIn é uma boa forma de demonstrar seus conhecimentos e esforços para evoluir na sua carreira para oportunidades futuras.

Feito com  por Rocketseat  Participe da nossa [comunidade aberta!](#)