Boas-vindas! Caro aspirante a dev!
Na primeira aula falamos sobre testes unitários, aprendemos a instalar a biblioteca Jest e criamos nossos primeiros casos de testes. Nesta aula, vamos nos aprofundar um pouco mais com múltiplos casos de testes e testes de integração. E aí, animado?

Aula 02 – Tipos de testes e Testes de integração:

Esta é nossa segunda aula do módulo de Testes, nela abordaremos os seguintes tópicos:

- Tipos de teste
- Testes de integração

Ao final desta aula, você estará apto(a), a:

- Conhecer os diferentes tipos de teste existentes;
- Entender de forma específica o que é e como realizar testes de integração.

Exercícios:

1. Com suas palavras, liste e explique os principais tipos de teste que podem ser realizados em um projeto de software. Obs.: Não precisa listar todos, só os mais comuns.

2. Quando existem vários casos de teste (ou seja, vários **test()** ou **it()**), em cada um dos quais se espera diferentes resultados (sucesso ou erro), é possível tornar a ordem dos testes aleatória ? E é possível paralelizar os testes ? Quais são os benefícios dessas abordagens em relação ao tradicional (um teste de cada vez), e quais são os desafios a serem ajustados para possibilitar essas abordagens ? Explique.

3. Qual a importância dos testes de integração e o que de fato é testado com eles ? Explique!

4. Mostre como se configura o Jest para que ele inclua no seu relatório de testes um relatório de “coverage” (cobertura), que indica quantas linhas do código foram testadas, quais funções foram testadas, quantos “branches” (if, else, while) foram testados, etc.

5. Seja bem-vindo(a) ao Discountopia, uma API de ecommerce que tem você como mais novo(a) desenvolvedor(a) ! Você recebeu o código-fonte do projeto (que está em anexo nessa aula) e foi dado(a) sua primeira tarefa para se ambientar com a aplicação: escrever o teste de integração da rota **PATCH /orders/:order_id/items/:product_id.**

Essa rota recebe:

A. o ID de uma ordem (carrinho de compras) pela URL

B. o ID de um produto pela URL

C. o ID do usuário autenticado, por meio do header “Authorization”, que contém um JWT (ou seja, o JWT vem pelo header Authorization, não por meio de cookies)

D. Um objeto **{quantity}** por meio do body da requisição. Onde **quantity** é um número representando a nova quantidade do produto que o usuário deseja que o carrinho tenha

A regra de negócio (ou seja, o que a rota faz) é: substituir a quantidade do produto especificado, na ordem especificada, pela nova quantidade descrita por **quantity**, e responder ao cliente com status 200 e body **{data: null, err: null}** indicando que não houve erro, e que não há dados a mostrar.

A regra de negócio também prevê os casos de erro e como a rota responde. Em todo caso, o body da resposta é **{data: null, err: {name: string, message: string}}**, ou seja, **data** é nulo e **err** é um objeto contendo o nome do erro e uma mensagem (a rota está implementada, veja no código). Os casos de erro previstos são:

A. Usuário não está autenticado: status 401 (unathorized)

B. Erro de input: **order_id** não é um número inteiro positivo: status 400 (bad request)

C. Erro de input: **product_id** não é um número inteiro positivo: status 400 (bad request)

D. Erro de input: **quantity** não é um número inteiro positivo: status 400 (bad request)

E. A Ordem não existe: status 404 (not found)

F. O produto não existe: status 404 (not found)

G. O usuário não é o proprietário da ordem: status 403 (forbidden)

H. O produto não está incluso na ordem: 404 (not found)

Quando mais de um caso de erro acontece ao mesmo tempo, vale o primeiro erro. Por exemplo, se o usuário não está autenticado e **order_id** é negativo, a API responde status 401.

Como explicado, o código dessa rota já está pronto (router => controller => service => repository). Seu trabalho é escrever o teste da rota no arquivo **__tests__/integration/orders.test.js** , no qual já está codificado o teste de outra rota (não-relacionado). Para mais detalhes sobre o projeto, veja o código e o vídeo explicativo do projeto no anexo da aula. Você deve usar o **superfetch** (pois já está sendo usado no projeto).

6. Agora que você se acostumou com o projeto, recebeu uma segunda tarefa mais difícil: codificar uma nova rota e ainda seus testes de integração. A rota em questão é **POST /orders/:order_id/discount-codes**

   Essa rota recebe:

   A. O ID de uma ordem pela URL

   B. Um objeto **{code}** pelo body da requisição. Onde **code** é um código de cupom como “25OFF2023EDTECH”

   C. o ID do usuário autenticado, por meio do header “Authorization”, que contém um JWT

   A regra de negócio (ou seja, o que a rota faz) é: procurar se existe um código de desconto com o código fornecido e adicionar esse código de desconto na ordem especificada. A rota deve responder ao cliente com status 200 e body **{data: null, err: null}** indicando que não houve erro, e que não há dados a mostrar.

   A regra de negócio também prevê os casos de erro e como a rota deve responder. Em todo caso, o body da resposta é **{data: null, err: {name: string, message: string}}**, ou seja, **data** é nulo e **err** é um objeto contendo o nome do erro e uma mensagem. Os casos de erro previstos são:

   A. Usuário não está autenticado: status 401 (unathorized)

   B. Erro de input: **order_id** não é um número inteiro positivo: status 400 (bad request)

   C. Erro de input: **code** não é uma string: status 400 (bad request)

   D. A Ordem não existe: status 404 (not found)

   E. O usuário não é o proprietário da ordem: status 403 (forbidden)

   F. A ordem já está fechada: status 403 (forbidden)

   G. Não existe código de desconto com o código fornecido: status 404 (not found)

H. O código de desconto está expirado (passou da data de validade): status 403 (forbidden)

I. O código de desconto não pode ser usado porque o valor total da ordem (desconsiderando qualquer desconto) é menor que o "valor mínimo" onde esse código pode ser usado: status 403 (forbidden)

J. O código de desconto não pode ser usado porque ele é específico a algumas categorias de produto (por exemplo "Eletrônicos" e "Eletrodomésticos"), mas nenhum produto na ordem pertence a alguma dessas categorias: status 403 (forbidden)

K. O código de desconto já foi usado na mesma ou em outra ordem do usuário: 403 (forbidden)

Quando mais de um caso de erro acontece ao mesmo tempo, vale o primeiro erro. Por exemplo, se o usuário não está autenticado e **order_id** é negativo, a API deve responder com status 401. Para mais detalhes sobre o projeto, veja o código e o vídeo explicativo do projeto no anexo da aula. Você deve usar o **superfetch** (pois já está sendo usado no projeto)

Em síntese:

Ao final desta aula você saberá como realizar múltiplos testes para diferentes cenários esperados, bem como criar testes de integração.

Lembre-se que o foco deste estudo é sempre aprofundar o seu conhecimento em web e, portanto, não se limite às referências deste módulo e mergulhe no conhecimento sobre Testes, pertinentes a esta aula.

Referências:

A. Os diferentes tipos de testes em software | Atlassian

B. Integration Testing: What is, Types with Example

C. Globais · Jest

D. Opções CLI do Jest

E. Aprenda a testar rotas de api com Jest em NodeJS (CodarMe no youtube): https://www.youtube.com/watch?v=HzjHDsoHwB4

F. Uma história de testes com Jest (básico): https://alluring-beluga-3dd.notion.site/Uma-hist-ria-de-testes-com-Jest-b-sico-6ef0d86483fb4514851f1397c4a6b3cb

    a. Veja a **Parte 2**