Desafio Técnico - Backend

Pessoa Desenvolvedora Júnior/Pleno
Engenharia 2024

Objetivo:

A equipe de engenharia da RD Station tem alguns princípios nos quais baseamos nosso trabalho diário. Um deles é: **projete seu código para ser mais fácil de entender, não mais fácil de escrever.**

Portanto, para nós, é mais importante um código de fácil leitura do que um que utilize recursos complexos e/ou desnecessários.

O que gostaríamos de ver:

- O código deve ser fácil de ler. Clean Code pode te ajudar;
- Notas gerais e informações sobre a versão da linguagem e outras informações importantes para executar seu código;
- Código que se preocupa com a performance (complexidade de algoritmo);
- O seu código deve cobrir todos os casos de uso presentes no README, mesmo que não haja um teste implementado para tal;
- A adição de novos testes é sempre bem-vinda;
- Você deve enviar para nós o link do repositório público com a aplicação desenvolvida (GitHub, BitBucket, etc.).

O Desafio - Carrinho de compras

O desafio consiste em uma API para gerenciamento de um carrinho de compras de e-commerce.

Você deve desenvolver utilizando a linguagem Ruby e framework Rails, uma API Rest que terá 3 endpoints que deverão implementar as seguintes funcionalidades:

1. Registrar um produto no carrinho:

Criar um endpoint para inserção de produtos no carrinho.

Se não existir um carrinho para a sessão, criar o carrinho e salvar o ID do carrinho na sessão.

Adicionar o produto no carrinho e devolver o payload com a lista de produtos do carrinho atual.

ROTA: `/cart` Payload:

```
None
{
  "product_id": 345, // id do produto sendo adicionado
  "quantity": 2, // quantidade de produto a ser adicionado
}
```

Response

```
None
{
  "id": 789, // id do carrinho
  "products": [
    {
      "id": 645,
      "name": "Nome do produto",
      "quantity": 2,
      "unit_price": 1.99, // valor unitário do produto
      "total_price": 3.98, // valor total do produto
    },
    {
```

```
      "id": 646,
      "name": "Nome do produto 2",
      "quantity": 2,
      "unit_price": 1.99,
      "total_price": 3.98,
    },
  ],
  "total_price": 7.96 // valor total no carrinho
}
```

2. Listar itens do carrinho atual:

Criar um endpoint para listar os produtos no carrinho atual.

ROTA: `/cart`

Response:

```
None
{
  "id": 789, // id do carrinho
  "products": [
    {
      "id": 645,
      "name": "Nome do produto",
      "quantity": 2,
      "unit_price": 1.99, // valor unitário do produto
      "total_price": 3.98, // valor total do produto
    },
    {
      "id": 646,
      "name": "Nome do produto 2",
      "quantity": 2,
      "unit_price": 1.99,
      "total_price": 3.98,
    },
  ],
  "total_price": 7.96 // valor total no carrinho
}
```

3. Alterar a quantidade de produtos no carrinho:

Um carrinho pode ter N produtos, se o produto já existir no carrinho, apenas a quantidade dele deve ser alterada

ROTA: `/cart/add_item`

Payload

```
None
{
  "product_id": 1230,
  "quantity": 1
}
```

Response:

```
None
{
  "id": 1,
  "products": [
    {
      "id": 1230,
      "name": "Nome do produto X",
      "quantity": 2, // considerando que esse produto já estava no carrinho
      "unit_price": 7.00,
      "total_price": 14.00,
    },
    {
      "id": 01020,
      "name": "Nome do produto Y",
      "quantity": 1,
      "unit_price": 9.90,
      "total_price": 9.90,
    },
  ],
  "total_price": 23.9
}
```

4. Remover um produto do carrinho:

Criar um endpoint para excluir um produto do do carrinho.

ROTA: `/cart/:product_id`

Detalhes adicionais:

- Verifique se o produto existe no carrinho antes de tentar removê-lo;
- Se o produto não estiver no carrinho, retorne uma mensagem de erro apropriada;

- Após remover o produto, retorne o payload com a lista atualizada de produtos no carrinho;
- Certifique-se de que o endpoint lida corretamente com casos em que o carrinho está vazio após a remoção do produto.

5. Excluir carrinhos abandonados:

Um carrinho é considerado abandonado quando estiver sem interação (adição ou remoção de produtos) há mais de 3 horas.

Quando este cenário ocorrer, o carrinho deve ser marcado como abandonado.

Se o carrinho estiver abandonado há mais de 7 dias, remover o carrinho.

Utilize um Job para gerenciar (marcar como abandonado e remover) carrinhos sem interação.

Configure a aplicação para executar este Job nos períodos especificados acima.

Detalhes adicionais:

- O Job deve ser executado regularmente para verificar e marcar carrinhos como abandonados após 3 horas de inatividade.
- O Job também deve verificar periodicamente e excluir carrinhos que foram marcados como abandonados por mais de 7 dias.

Como resolver:

- *Implementação:*

Você deve usar como base o código disponível neste repositório e expandi-lo para que atenda as funcionalidade descritas acima.
Há trechos parcialmente implementados e também sugestões de locais para algumas das funcionalidades sinalizados com um `# TODO`. Você pode segui-los ou fazer da maneira que julgar ser a melhor a ser feita, desde que atenda os contratos de API e funcionalidades descritas.

- *Testes:*

Existem testes pendentes, eles estão marcados como Pending, e devem ser implementados para garantir a cobertura dos trechos de código implementados por você. Alguns testes já estão passando e outros estão com erro. Com a sua implementação os testes com erro devem passar a funcionar. A adição de novos testes é sempre bem-vinda, mas sem alterar os já implementados.

O que esperamos:

- Implementação dos testes faltantes e de novos testes para os métodos/serviços/entidades criados;
- Construção das 4 rotas solicitadas;
- Implementação de um job para controle dos carrinhos abandonados.

Itens adicionais / Legais de ter:

- Utilização de factory na construção dos testes;
- Desenvolvimento do docker-compose / dockerização da app.

A aplicação já possui um Dockerfile, que define como a aplicação deve ser configurada dentro de um container Docker. No entanto, para completar a dockerização da aplicação, é necessário criar um arquivo `docker-compose.yml`. O arquivo irá definir como os vários serviços da aplicação (por exemplo, aplicação web, banco de dados, etc.) interagem e se comunicam.

Adicione tratamento de erros para situações excepcionais válidas, por exemplo: garantir que um produto não possa ter quantidade negativa.

Se desejar, você pode adicionar a configuração faltante no arquivo `docker-compose.yml` e garantir que a aplicação rode de forma correta utilizando Docker.

Informações técnicas:

Dependências:

- ruby 3.3.1
- rails 7.1.3.2
- postgres 16
- redis 7.0.15

Como executar o projeto

Executando a app sem o docker:

Dado que todas as ferramentas estão instaladas e configuradas.

Instalar as dependências do:

- bundle install

Executar o sidekiq:

- bundle exec sidekiq

Executar projeto:

- bundle exec rails server

Executar os testes:

- bundle exec rspec

Como enviar seu projeto:

Salve seu código em um versionador de código (GitHub, GitLab, Bitbucket) e nos envie o link público. Se achar necessário, informe no README as instruções para execução ou qualquer outra informação relevante para correção/entendimento da sua solução.