UmHelp

CONECTAMOS DIARISTAS COM QUEM QUER UMA CASA BEM CUIDADA

UmHelp é o app de limpeza que se encaixa em qualquer casa, e em qualquer rotina. Queremos cuidar da sua casa, e deixar ela do jeito que você gosta. **Quer fazer parte do nosso time? #VemSerHelper!**

Desafio de Programação Back-End

Olá e muitíssimo obrigado pelo interesse na UmHelp! Somos uma startup em rápido crescimento, e sempre buscamos pessoas boas para compor nossa equipe de tecnologia. Esperamos ter você a bordo conosco em breve!

Pedimos que leia este documento inteiro com atenção para maximizar suas chances de sucesso.

O teste que você irá realizar consiste em elaborar 3 microsserviços que possibilitam o agendamento de serviços de limpeza por parte de um usuário, com a possibilidade da aplicação de desconto promocional.

1. Sobre este documento

- O processo de envio da sua solução e os critérios que serão avaliados pela equipe de tecnologia da UmHelp estão descritos na seção **Avaliação**.
- As regras que o sistema deve seguir estão descritas na seção **Regras de negócio**.
- Algumas restrições adicionais estão descritas na seção **Restrições**.
- A interface que cada microsserviço deve expor está descrita na seção **Serviços**.
- As descrições de interfaces ocasionalmente referenciam entidades descritas na seção **Entidades**.

Caso não tenha certeza sobre qualquer coisa explicada neste documento, fique à vontade para enviar um email para pedrocastilho@da1help.com com suas dúvidas.

2. Avaliação

Seu desafio deverá ser disponibilizado por meio de um repositório Git acessível à equipe da UmHelp. Caso não queira disponibilizar publicamente, entre em contato conosco para que conversemos sobre como podemos adquirir acesso ao seu repositório.

2.1. Critérios

Uma vez disponibilizado a nós, a avaliação do seu código será feita de acordo com os seguintes critérios:

Ausência de bugs

Seu código deve funcionar corretamente, de acordo com a especificação dada por este documento.

Legibilidade e formatação

Lembre que código em geral é lido mais vezes do que ele é escrito. Escreva seu código pensando em quem for lê-lo. Busque minimizar a dificuldade de leitura. Use indentação a seu favor. Siga as boas práticas de formatação de código da linguagem escolhida por você.

Clareza

Seu código deve deixar a sua intenção clara para o leitor. Tanto quanto seja plausível, deve ser possível entender o que o código faz apenas lendo ele.

Extensibilidade

Considere que o código que você está desenvolvendo será apenas uma parte de uma solução maior. Pense no esforço necessário para realizar adições ao seu código e busque minimizá-lo.

Manutenibilidade

Considere que o seu código será mantido por muito tempo após você criá-lo, por várias pessoas. Pense no esforço necessário para realizar mudanças no seu código e busque minimizá-lo.

Desacoplamento

Seu código não deve criar dependências desnecessárias entre módulos ou serviços. Ao alterar uma parte do código, não deveria ser necessário alterar partes sem relação lógica com a funcionalidade da parte alterada.

Boas práticas arquiteturais

Informe-se sobre boas práticas de arquitetura orientada a serviços e use-as. Seu sistema deve ser robusto como um todo. Lembre-se que em uma arquitetura orientada a serviços, em geral não podemos assumir que todos os serviços estarão sempre em um estado válido. Pense em casos excepcionais e busque lidar com eles tanto quanto possível.

2.2 Opcionais

Os pontos a seguir não são obrigatórios, mas apreciamos vê-los:

- Testes unitários
- Sistema deployado em algum provedor de cloud
- Validações e erros bem estruturados

2.3 Após a avaliação

Em geral, iremos avaliar seu código poucos dias após o envio. Uma vez que nossa avaliação esteja completa, iremos enviar um feedback sobre o que achamos. Caso seu código seja aprovado, além do feedback, iremos marcar uma data para uma entrevista presencial com você ou um bate papo por telefone/hangout.

2.4 Entrevista presencial

Na entrevista presencial, conversaremos com você sobre a estrutura do seu código e as escolhas feitas por você. Esteja pronta(o) para explicar as decisões que tomou e conversar sobre alternativas. Alguns tópicos nos quais iremos tocar (mas não os únicos) serão: escolha de bancos de dados, estruturas de interfaces, arquitetura de código. Além disso, poderemos mudar as condições do desafio e perguntar o que você teria feito diferentemente.

3. Regras de negócio

1. Uma vez que um pedido de limpeza é cadastrado, se um desconto foi utilizado, o desconto deve ser consumido (removido da base de descontos). Portanto, o mesmo desconto não deve ser aplicado mais de uma vez.
2. Descontos não devem ser consumidos quando um preço é apenas visualizado.
3. Se um usuário tiver múltiplos descontos cadastrados, o desconto que resulte no menor preço total deve ser aplicado ao pedido de limpeza dele.
4. Uma vez que um pedido de limpeza seja registrado, ele deve ser persistido indefinidamente.
5. O preço base de um serviço de limpeza (sem aplicação de nenhum desconto) deve ser considerado como sendo de R$ 50 por hora.

4. Restrições

1. Os serviços devem usar ao menos 2 linguagens de programação diferentes.
2. Os serviços podem usar um único banco de dados compartilhado, ou bancos de dados separados. Lembre que essa decisão também será questionada.

5. Serviços

A comunicação entre os serviços pode ser feita usando qualquer padrão de comunicação que você deseje. Lembre-se apenas que você terá que justificar suas escolhas durante a entrevista técnica.

Serviço 1: Gateway

Esse serviço deve expor uma interface REST pública, com os seguintes *endpoints*:

- **POST** `/price`:
  Recebe a especificação de um pedido de limpeza (descrita na seção **Entidades**) e retorna o preço de um pedido de limpeza para o usuário especificado, sem registrar o serviço e sem consumir descontos.

- **POST** `/requests`:

Cadastra um novo pedido de limpeza com os parâmetros passados. Retorna os dados do pedido, incluindo o preço final.

- **GET** `/requests`: Retorna todos os pedidos de limpeza atualmente cadastrados.

- **POST** `/discounts`: Cadastra um novo desconto.

Além dos verbos já especificados, os verbos **PUT**, **PATCH** e **DELETE** também devem ser implementados para os endpoints `/requests` e `/discounts`, com comportamento de acordo com a semântica padrão de APIs REST.

Além disso, deve ser possível acessar os pedidos de limpeza individualmente através do uso de algum identificador. Outros endpoints ou verbos podem ser criados caso você julgue necessário, mas tenha em mente os requisitos de segurança da aplicação!

O serviço gateway deve realizar as suas funcionalidades através da comunicação com os outros serviços, especificados a seguir.

Serviço 2: Pedidos de limpeza

Este serviço deve receber requisições de pedidos de limpeza a partir do serviço gateway e calcular os dados que não serão dados no pedido (em especial, o preço do pedido, levando em consideração a aplicação de desconto, caso exista desconto cadastrado no serviço 3. Este serviço também deve ser responsável pelo cadastro persistente de pedidos de limpeza.

Serviço 3: Descontos individuais

Este serviço será responsável por cadastrar descontos individuais. Além disso, ele será consultado pelo serviço 2, que o usará para determinar a existência ou não de descontos. Além disso, quando um pedido de limpeza for cadastrado, este serviço será responsável por garantir que a regra 1 seja respeitada.

Entidades

Pedido de limpeza

Um pedido de limpeza deve conter, ao menos, as seguintes informações:

- Data
- Duração
- Identificação do usuário solicitante

Desconto

Um desconto deve conter, ao menos, as seguintes informações:

- Tipo de desconto (valor absoluto ou porcentagem)
- Valor do desconto
- Identificação do usuário que tem direito ao desconto