Teste Técnico – Full Stack

Olá, candidato(a)!

Primeiro gostaríamos de agradecer por seu interesse em fazer parte do time de desenvolvimento da Shopper.com.br.

Estamos construindo o melhor sistema de abastecimento do Brasil e para isso estamos procurando pessoas apaixonadas por usar a tecnologia para criar soluções inovadoras. Esperamos que seja você!

Explicando um pouco do processo seletivo:

Etapa 1 – Teste Técnico

Nessa primeira fase, você irá construir o backend e frontend de uma aplicação para transporte particular. Serão 3 três endpoints e uma integração com a API do Google Maps para o backend e um frontend simples e funcional para complementar o projeto.

Etapa 2 – Apresentação técnica

As melhores entregas serão convidadas para apresentar e participar de uma discussão mais aprofundada sobre o projeto com o time técnico da Shopper.

Etapa 3 - Cultural

Finalizada a validação das suas habilidades técnicas, agora queremos te conhecer melhor como pessoa e profissional. Nesta próxima etapa, você terá a oportunidade de conversar com nosso time, compartilhar seus objetivos de carreira e entender melhor nossa cultura e forma de trabalho. Será um espaço aberto para trocar ideias e esclarecer qualquer dúvida que você possa ter sobre nós.

Além disso, preparamos um bate-papo descontraído para que possamos conhecer seus valores, suas motivações e como você se encaixa em nossa equipe. Queremos ter certeza de que essa oportunidade é ideal para você, tanto quanto você é para nós.

ETAPA 1 – Teste técnico

O que você precisará saber:

- Ler especificações técnicas em inglês e entender requisitos de negócios.
- Desenvolver uma API REST em Node.js com Type Script.
- Noção básica de modelagem de bancos de dados.
- Criar uma imagem e subir um container utilizando Docker.
- O básico do versionamento em um repositório usando Git.

No que você será avaliado:

Sua aplicação será submetida a uma bateria de testes que irão verificar cada um dos critérios de aceite. Por isso é importante que você **leia atentamente e siga rigorosamente** todas as instruções. Sua aplicação deve cumprir **integralmente** os requisitos.

Pontos desejáveis, mas que não são eliminatórios:

- Uma arquitetura limpa (clean code).
- Testes unitários.

Como entregar seu projeto:

- Preencha esse formulário - https://forms.gle/4dTxZxR4CuGVP8DH8.
- A aplicação deve ser completamente dockerizada.
- Deve conter um arquivo docker-compose.yml na raiz do seu repositório.
  Nosso script de teste irá criar um arquivo .env na raiz do repositório no seguinte formato. Sua aplicação deve receber essa variável de ambiente para a execução.

  GOOGLE_API_KEY=<chave da API>

  ATENÇÃO: NÃO compartilhe sua chave pessoal conosco.

- O *docker-compose* deve ser capaz de subir a aplicação e todos os serviços necessários com o comando *docker-compose up*.
- O backend da aplicação deve ficar exposto na **porta 8080**.
- O frontend da aplicação deve ficar exposto na **porta 80**.

Como você deve usar LLMs (Copilot, ChatGPT, Gemini, Llama, etc..),

Gostamos e incentivamos quem busca a inovação para se tornar mais produtivo, porém queremos avaliar você! Utilize a LLM como ferramenta e não como a criadora do seu código.

Você **NÃO** deve fazer:

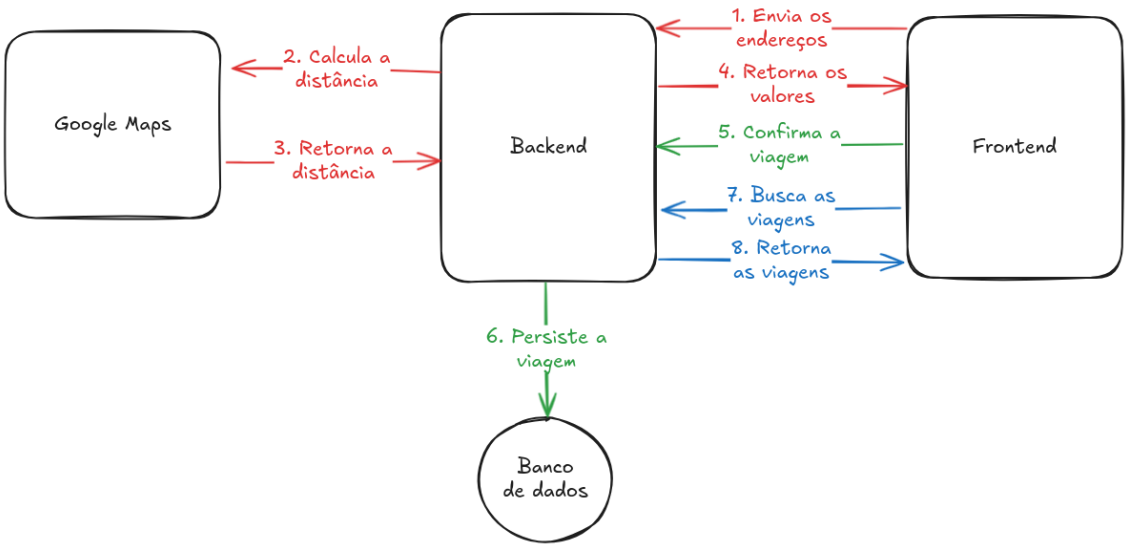
- Copiar esse teste, colar no GPT e apenas copiar o resultado. LLMs geram códigos ruins.

Você pode fazer:

- Usar o GPT para melhorar o código que você criou ou estudar melhores práticas.

CENÁRIO

Vamos desenvolver uma aplicação conceito onde o usuário poderá solicitar uma viagem em carro particular de um ponto A até um ponto B. Ele poderá escolher entre algumas opções de motoristas e valores e confirmar a viagem. Depois também poderá listar o histórico das viagens realizadas. O diagrama abaixo mostra a estrutura geral da aplicação.



DEFINIÇÕES DO BACKEND

O backend deverá ser uma API Rest em NodeJS e Typescript, e terá os seguintes endpoints:

POST /ride/estimate

Responsável por receber a origem e o destino da viagem e realizar os cálculos dos valores da viagem.

Esse endpoint deve fazer as seguintes validações:

- Os endereços de origem e destino recebidos não podem estar em branco.
- O id do usuário não pode estar em branco.
- Os endereços de origem e destino não podem ser o mesmo endereço.

Após as validações, ele deve:

- Calcular a rota entre a origem e destino usando a API Routes do Google Maps.
- Com base no retorno, deve listar os motoristas disponíveis para a viagem de acordo com a quilometragem mínima que aceitam, cada um com seu respectivo valor, usando como base a seguinte tabela:

ID	Nome	Descrição	Carro	Avaliação	Taxa	KM Mínimo
1	Homer Simpson	Olá! Sou o Homer, seu motorista camarada! Relaxe e aproveite o passeio, com direito a rosquinhas e boas risadas (e talvez alguns desvios).	Plymouth Valiant 1973 rosa e enferrujado	2/5 Motorista simpático, mas errou o caminho 3 vezes. O carro cheira a donuts.	R$ 2,50/ km	1
2	Dominic Toretto	Ei, aqui é o Dom. Pode entrar, vou te levar com segurança e rapidez ao seu destino. Só não mexa no rádio, a playlist é sagrada.	Dodge Charger R/T 1970 modificado	4/5 Que viagem incrível! O carro é um show à parte e o motorista, apesar de ter uma cara de poucos amigos, foi super gente boa. Recomendo!	R$ 5,00/ km	5
3	James Bond	Boa noite, sou James Bond. À seu dispor para um passeio suave e discreto. Aperte o cinto e aproveite a viagem.	Aston Martin DB5 clássico	5/5 Serviço impecável! O motorista é a própria definição de classe e o carro é simplesmente magnífico. Uma experiência digna de um agente secreto.	R$ 10,00/ km	10

O endpoint deverá retornar:

- A latitude e longitude dos pontos iniciais e finais.
- A distância e tempo do percurso.
- A lista de motoristas disponíveis ordenados do mais barato para o mais caro, cada um contendo:
  - O ID e nome do motorista.
  - A descrição.
  - O carro.
  - A avaliação.
  - O valor total da corrida.
- A resposta original da rota no Google.

Request Body

```
{
    "customer_id": string,
    "origin": string,
    "destination": string
}
```

Response Body

Status Code	Descrição	Resposta
200	Operação realizada com sucesso	{ "origin": { "latitude": number, "longitude": number }, "destination": { "latitude": number, "longitude": number }, "distance": number, "duration": string, "options": [{

		"id": number,
		"name": string,
		"description": string,
		"vehicle": string,
		"review": {
		"rating": number,
		"comment": string
		},
		"value": number
		}
		],
		"routeResponse": object
		}

400	Os dados fornecidos no corpo da requisição são inválidos	{
		"error_code": "INVALID_DATA",
		"error_description": string
		}

Documentação técnica do Google Maps:

https://developers.google.com/maps/documentation/routes/overview?hl=pt-br

ATENÇÃO: Você precisará obter uma chave de acesso para usar a funcionalidade. Pode ser necessário adicionar um cartão para obter a chave, mas um crédito do Google será automaticamente aplicado e cobrirá o uso necessário para testar. Não realize despesas financeiras para realizar esse teste.

PATCH /ride/confirm
Responsável por confirmar a viagem e gravá-la no histórico.

Esse endpoint deve fazer as seguintes validações:

- Os endereços de origem e destino recebidos não podem estar em branco.

- O id do usuário não pode estar em branco.
- Os endereços de origem e destino não podem ser o mesmo endereço.
- Uma opção de motorista foi informada e é uma opção válida.
- A quilometragem informada realmente é válida para o motorista selecionado.

Após as validações ele deve:

- Salvar no banco de dados os dados da viagem realizada.

Ele **NÃO** deve fazer:

- Recalcular a rota usando a API do Google Maps

Ela irá retornar:

- Resposta de OK ou ERRO dependendo do valor informado.

Request Body

```
{
   "customer_id": string,
   "origin": string,
   "destination": string,
   "distance": number,
   "duration": string,
   "driver": {
       "id": number,
       "name": string
   },
   "value": number
}
```

Response Body:

Status Code	Descrição	Resposta
200	Operação realizada com sucesso	`{ "success": true }`

400	Os dados fornecidos no corpo da requisição são inválidos	<pre>{
		"error_code": "INVALID_DATA",
		"error_description": string
		}</pre>

404	Motorista não encontrado	<pre>{
		"error_code": "DRIVER_NOT_FOUND",
		"error_description": string
		}</pre>

406	Quilometragem inválida para o motorista	<pre>{
		"error_code": "INVALID_DISTANCE",
		"error_description": string
		}</pre>

GET /ride/{customer_id}?driver_id={id do motorista}
Responsável por listar as viagens realizadas por um determinado usuário

Esse endpoint deve fazer as seguintes validações:

- O id do usuário não pode estar em branco.
- Se um id de motorista for informado, ele precisa ser um id válido.

Após as validações ele:

- Buscar as viagens realizadas pelo usuário, ordenando da mais recente para a mais antiga.
- Pode receber um query parameter “driver_id” que, se informado, deve filtrar apenas as viagens realizadas pelo usuário com este motorista.

Ela irá retornar:

- Uma lista com as viagens realizadas.

Response Body**:**

Status Code	**Descrição**	**Resposta**
200	Operação realizada com sucesso	{ "customer_id": string, "rides": [{ "id": number, "date": datetime, "origin": string, "destination": string, "distance": number, "duration": string, "driver": { "id": number, "name": string }, "value": number }] }
400	Motorista invalido	{ "error_code": "INVALID_DRIVER",

		"error_description": string }

404	Nenhum registro encontrado	{
		"error_code": "NO_RIDES_FOUND",
		"error_description": string
		}

DEFINIÇÕES DO FRONTEND

O Frontend deverá ser uma Single Page Application em React e TypeScript e terá as seguintes telas:

Solicitação de viagem

- Deve conter um formulário com os campos para informar o id do usuário, o endereço de origem e o endereço de destino e um botão para estimar o valor da viagem
- Deve fazer a requisição para a API passando os parâmetros necessários, ao receber a resposta deve exibir a tela de opções de viagem

Opções de viagem

- Deve mostrar um mapa estático com a rota retornada na estimativa plotada, indicando o ponto A e o ponto B.
- Deve mostrar a lista de opções de motoristas com:
  - nome.
  - descrição.
  - veículo.
  - avaliação.
  - valor da viagem.
- Para cada motorista deve ter um botão “Escolher”, que irá fazer a requisição para a API e confirmar a viagem.
- Após confirmar a viagem, deve direcionar automaticamente para a tela de histórico de viagens.

Histórico de viagens

- Deve mostrar um campo para informar o id do usuário, um seletor de motorista, com uma opção para mostrar todos e um botão para aplicar o filtro.
- Ao aplicar o filtro, deve exibir a lista das viagens realizadas, com:
    - data e hora da viagem.
    - nome do motorista.
    - origem.
    - destino.
    - distância.
    - tempo.
    - valor.

Tratamento de erros

Em todas as telas, os erros devem ser exibidos para o usuário, permitindo que ele verifique o problema e tente novamente.