Teste Técnico – Mobile

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 frontend de uma aplicação para transporte particular. Será um aplicativo simples e funcional que se integrará com 3 três endpoints de uma API simulada.

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 - Fit 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 aplicação nativa de iOS com swift ou Android com kotlin
- 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/hitvwDAyrZDDr2556).
- A entrega será apenas do código fonte, não é necessário gerar um APK ou IPA da aplicação.
- O projeto deverá compilar e rodar no emulador do Android Studio ou Xcode na versão mais recente.
- A aplicação se conectará ao backend simulado da aplicação no endereço: **https://xd5zl5kk2yltomvw5fb37y3bm40vsyrx.lambda-url.sa-east-1.on.aws**.

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.

DEFINIÇÕES DO BACKEND

O backend é uma API simulada, ela possui respostas padronizadas que são retornadas de acordo com os parâmetros enviados.
Para acessar o backend, utilize a URL de base abaixo:

https://xd5zl5kk2yltomvw5fb37y3bm40vsyrx.lambda-url.sa-east-1.on.aws

A API possui os seguintes endpoints que devem ser utilizados pela aplicação:

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

Esse endpoint possui os seguintes cenários:

Origem	Destino	ID de cliente	Cenário
Qualquer	Qualquer	Qualquer	Sucesso com nenhum motorista disponível
Av. Pres. Kenedy, 2385 - Remédios, Osasco - SP, 02675-031	Av. Paulista, 1538 - Bela Vista, São Paulo - SP, 01310-200	Qualquer	Sucesso com 3 motoristas disponíveis
Av. Thomas Edison, 365 - Barra Funda,	Av. Paulista, 1538 - Bela Vista, São Paulo	Qualquer	Sucesso com 2 motoristas disponíveis

São Paulo - SP, 01140-000	- SP, 01310-200		
Av. Brasil, 2033 - Jardim America, São Paulo - SP, 01431-001	Av. Paulista, 1538 - Bela Vista, São Paulo - SP, 01310-200	Qualquer	Sucesso com 1 motorista disponível
Qualquer	Igual a origem	Qualquer	Erro do endereço de destino igual ao de origem
Qualquer	Nulo	Qualquer	Erro de destino em branco
Nulo	Qualquer	Qualquer	Erro de origem em branco
Qualquer	Qualquer	Nulo	Erro de id de cliente em branco

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 }, }`

		<pre>"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
		}</pre>

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

PATCH /ride/confirm
Responsável por confirmar a viagem.

Para usar endpoint, considere a tabela de motoristas abaixo para saber quando o ID do motorista é válido e quando a distância é válida:

ID do motorista	KM Mínimo aceito
1	1
2	5
3	10

Esse endpoint possui os seguintes cenários:

Origem	Destino	ID de cliente	ID do motorista	Distância	Cenário
Qualquer	Qualquer	Qualquer	Válido	Válida	Sucesso
Qualquer	Qualquer	Qualquer	Válido	Inválida	Erro de distância inválida para o motorista
Qualquer	Qualquer	Qualquer	Inválido	Válida	Erro de motorista inválido
Qualquer	Qualquer	Qualquer	Nulo	Válida	Erro de motorista não informado
Qualquer	Igual a origem	Qualquer	Válido	Válida	Erro de endereço de destino igual a origem
Qualquer	Nulo	Qualquer	Válido	Válida	Erro de destino em branco
Nulo	Qualquer	Qualquer	Válido	Válida	Erro de origem

					em branco
Qualquer	Qualquer	Nulo	Válido	Válida	Erro de id de cliente em branco

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	`{ "error_code": "INVALID_DATA", "error_description": string }`

404	Motorista não encontrado	{
		"error_code": "DRIVER_NOT_FOUND",
		"error_description": string
		}

406	Quilometragem inválida para o motorista	{
		"error_code": "INVALID_DISTANCE",
		"error_description": string
		}

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

Esse endpoint possui os seguintes cenários (considere os mesmos IDs de motoristas do endpoint de confirmação da viagem):

ID de cliente	**ID de motorista**	**Cenário**
CT01	Nulo ou válido	Sucesso com lista aleatória de viagens
Qualquer	Nulo ou válido	Erro sem viagens salvas
CT01	Inválido	Erro de motorista inválido

Response Body:

Status Code	**Descrição**	**Resposta**

200	Operação realizada com sucesso	<pre>{
		"customer_id": string,
		"rides": [
		{
		"id": number,
		"date": datetime,
		"origin": string,
		"destination": string,
		"distance": number,
		"duration": string,
		"driver": {
		"id": number,
		"name": string
		},
		"value": number
		}
		]
		}</pre>

400	Motorista invalido	<pre>{
		"error_code": "INVALID_DRIVER",
		"error_description": string
		}</pre>

404	Nenhum registro encontrado	<pre>{</pre>

		<pre>"error_code": "NO_RIDES_FOUND", "error_description": string }</pre>

DEFINIÇÕES DO FRONTEND

O Frontend deverá ser uma aplicação nativa, considere os pontos abaixo:

- No Android:
  - Utilize Kotlin para o desenvolvimento
  - Você pode escolher criar as telas em XML ou em Compose, como preferir
- No iOS:
  - Utilize Swift para o desenvolvimento
  - Você pode escolher criar as telas em UIKit ou SwiftUI, como preferir
- Funcionalidades e animações:
  - Iremos avaliar os requisitos descritos neste documento, garanta que eles estejam funcionando corretamente
  - Você pode adicionar animações, transições de tela ou layouts mais refinados se quiser, porém eles não serão considerados como eliminatórios durante a avaliação

A aplicação deve possuir as seguintes telas e comportamentos:

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 e feedback ao usuário

- Em todas as telas, os erros devem ser exibidos para o usuário, permitindo que ele verifique o problema e tente novamente.
- Sempre que houver algum processamento mais lento, como uma requisição, deve ser apresentado para o usuário que o aplicativo está executando uma ação.
- A aplicação deve impedir que o usuário realize ações repetidas por engano ao tocar várias vezes em um botão (*debounce*).