**Teste Técnico – Tech Lead**

Olá! Tudo bem?

Muito obrigada pelo seu interesse na Estapar.

Segue abaixo o nosso desafio técnico. Caso tenha qualquer dúvida, fique à vontade para entrar em contato.

# Estapar Backend Developer Test (V1.4)

## Objetivo

Construa um sistema backend simples para gerenciar um estacionamento: controlar vagas disponíveis, entrada/saída de veículos e calcular receita.

As garagens tem um único grupo de cancelas que ficam na entrada da gagagem, os setores são divisões lógicas e não físicas para a organização do contratante do seu pool de vagas.

## O que fazer

- Use Java 21, Kotlin 2.1.x .

- Use Spring, Micronaut.

- Use MySQL.

- Use git para versionamento.

- Pode usar AI se quiser.

## Simulador

Inicie o simulador:

```bash

docker run -d --network="host" cfontes0estapar/garage-sim:1.0.0

```

Busque a configuração da garagem com `GET /garage`. O simulador enviará eventos de veículos para seu webhook.

## Requisitos Funcionais

- Ao iniciar, busque e armazene os dados da garagem/vagas do simulador.

- Implemente uma API REST:

- `GET /revenue` — receita total por setor e data.

- Aceite POSTs no webhook `http://localhost:3003/webhook` para eventos ENTRY, PARKED e EXIT.

## Regras de Negócio

- Ao entrar um veículo, marque uma vaga como ocupada.

- Ao sair, marque a vaga como disponível e calcule o valor:

- Primeiros 30 minutos são grátis.

- Após 30 minutos, cobre uma tarifa fixa por hora, inclusive a primeira hora (use `basePrice` da garagem, arredonde para cima).

- Se o estacionamento estiver cheio, não permita novas entradas até liberar uma vaga.

### Regra de preço dinâmico.

1. Com lotação menor que 25%, desconto de 10% no preço, na hora da entrada.

2. Com lotação menor até 50%, desconto de 0% no preço, na hora da entrada.

3. Com lotação menor até 75%, aumentar o preço em 10%, na hora da entrada.

4. Com lotação menor até 100%, aumentar o preço em 25%, na hora da entrada.

### Regra de lotação

Com 100% de lotação, fechar o setor e só permitir mais carros com a saida de um já estacionado.

## O que será avaliado

- Clareza e estrutura do código

- Uso de REST e banco de dados

- Tratamento de eventos e regras de negócio

- Tratamento básico de erros e testes

---

## Detalhes da API

### Eventos do Webhook

POST para `http://localhost:3003/webhook`:

\*\*ENTRY\*\*

```json

{

"license\_plate": "ZUL0001",

"entry\_time": "2025-01-01T12:00:00.000Z",

"event\_type": "ENTRY"

}

```

Response:

HTTP 200

\*\*PARKED\*\*

```json

{

"license\_plate": "ZUL0001",

"lat": -23.561684,

"lng": -46.655981,

"event\_type": "PARKED"

}

```

Response:

HTTP 200

\*\*EXIT\*\*

```json

{

"license\_plate": "ZUL0001",

"exit\_time": "2025-01-01T12:00:00.000Z",

"event\_type": "EXIT"

}

```

Response:

HTTP 200

### Configuração da Garagem

GET `/garage`:

EXEMPLO:

```json

{

"garage": [

{

"sector": "A",

"basePrice": 10.0,

"max\_capacity": 100

}

],

"spots": [

{

"id": 1,

"sector": "A",

"lat": -23.561684,

"lng": -46.655981

}

]

}

```

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# API do Projeto a ser implementada

## REST API

### Consulta faturamento

\*\*GET\*\*

`/revenue`

Request

```JSON

{

"date": "2025-01-01",

"sector": "A"

}

```

Response

```JSON

{

"amount": 0.00,

"currency": "BRL",

"timestamp": "2025-01-01T12:00:00.000Z"

}

```