Desafio Técnico – Backend Júnior

⚠️ Aviso antes de começar

- Leia com atenção este documento todo e tente seguir ao máximo as instruções;
- Crie um repositório no seu GitHub sem citar nada relacionado à DirectoAI;
- Faça seus commits no seu repositório;
- Envie o link do seu repositório para o email do recrutador responsável, ou qualquer outro canal de comunicação;
- Você poderá consultar o Google, Stackoverflow ou algum projeto particular na sua máquina;
- Dê uma olhada no tópico de **"Materiais úteis"**;
- Dê uma olhada em como será a entrevista no tópico **"Para o dia da entrevista técnica"**;
- Fique à vontade para perguntar qualquer dúvida aos recrutadores;
- Fique tranquilo, respire. **Boa sorte!** 🎯

🎯 Objetivo

Criar uma **API de gerenciamento de campanhas publicitárias** simples, utilizando **Golang com Fiber** como provider HTTP, estruturada seguindo boas práticas de organização de pastas e rodando dentro de containers via **Docker Compose**.

📌 Requisitos Técnicos

1. **Linguagem**: Go (>= 1.21).
2. **Framework HTTP**: Fiber.
3. **Banco de Dados**: PostgreSQL.
4. **Docker**: A aplicação deve rodar com `docker-compose up`
5. **Padrão de Pastas**: Baseado em Standard Go Project Layout.

📁 Estrutura Sugerida (Não obrigatória)

```
project/
|— cmd/
|   └— server/          # Arquivo main.go
|
|— internal/
|   ├— campaign/        # Lógica de domínio de campanhas
|   |   ├— handler.go   # Handlers HTTP (Fiber)
|   |   ├— service.go   # Regras de negócio
|   |   └— repository.go # Comunicação com o banco
|   |
|   ├— user/            # Autenticação e usuários
|   |   ├— handler.go
|   |   └— repository.go
|   |
|   ├— middleware/
|   |   └— auth.go      # Middleware de autenticação
|   |
|   ├— database/
|   |   └— postgres.go  # Conexão com PostgreSQL
|   |
|   └— router/
|       └— router.go    # Definição das rotas
|
|— docker-compose.yml
|— Dockerfile
|— go.mod
|— go.sum

```

📚 Funcionalidades Requeridas

A API deve expor endpoints para gerenciar **Campanhas** e **Autenticação**.

🔐 Autenticação

- **Login**
  `POST /login`
  Body esperado:

```
{
  "username": "admin",
  "password": "123456"
}
```

  → Deve retornar um **token JWT** válido.

- **Middleware de Autenticação**
  Criar um middleware que valide o token JWT.

Todos os endpoints de **campanhas** devem exigir autenticação.

📢 Campanhas

- **Criar campanha**
  `POST /campaigns`
  Body esperado:

```
{
  "name": "Campanha Teste",
  "budget": 1000,
  "status": "active"
}
```

- **Listar campanhas**
  `GET /campaigns`
- **Buscar campanha por ID**
  `GET /campaigns/:id`
- **Atualizar campanha**
  `PUT /campaigns/:id`
- **Deletar campanha**
  `DELETE /campaigns/:id`

Estrutura da Tabela no PostgreSQL

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL
);

CREATE TABLE campaigns (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    budget NUMERIC(12,2) NOT NULL,
    status VARCHAR(20) NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

🐳 Docker

`docker-compose.yml` esperado

- Serviço **app** (Go + Fiber).

- Serviço **db** (Postgres 15).

Exemplo mínimo:

```yaml
version: "3.9"
services:
  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
      POSTGRES_DB: ads
    ports:
      - "5432:5432"
    volumes:
      - db_data:/var/lib/postgresql/data

  app:
    build: .
    command: ["./server"]
    depends_on:
      - db
    ports:
      - "8080:8080"
    environment:
      DB_HOST=db
      DB_PORT=5432
      DB_USER=admin
      DB_PASSWORD=admin
      DB_NAME=ads
      JWT_SECRET=mysecret

volumes:
  db_data:

```

✅ Critérios de Avaliação

- Organização do projeto seguindo boas práticas (ex: golang-standards/projectlayout)
- Clareza e legibilidade do código (Clean Code)
- Aplicação de SOLID e Design Patterns
- Cobertura e qualidade dos testes (mínimo 80% na camada de domínio/serviço)
- Uso de cache (Redis ou in-memory) e estrutura dockerizada (Docker Compose)
- Git com histórico de commits organizado (sugestão: Conventional Commits)

- Capacidade de aplicar lógica de negócio de forma clara e escalável
- Documentação clara e objetiva (Swagger ou README detalhado)
- Diferenciais implementados (goroutines, mensageria, GRPC, etc.)

🚫 O que não será avaliado

- Interface do frontend (não é necessário estilizar);
- Frameworks específicos no React (Next, Tailwind etc);
- Deploy em produção;
- Performance real em ambiente de alta carga.

🚀 Diferenciais (não obrigatórios)

- Testes unitários básicos (`testing` ou `testify`).
- Documentação com Swagger/OpenAPI.
- Uso de variáveis de ambiente via `.env`.
- Hash de senha no cadastro de usuários.

🗓️ Para o dia da entrevista técnica

Na data marcada pelo recrutador, tenha sua aplicação rodando na sua máquina local para execução dos testes e para nos mostrar os pontos desenvolvidos e possíveis questionamentos.

Faremos um **code review junto com você**, como se você já fosse do nosso time. Você poderá explicar o que pensou, como arquitetou e como pode evoluir o projeto.

👉 Entregável:

Link para repositório no GitHub/GitLab com instruções no `README.md` para rodar via Docker.