Desafio Técnico Estágio — AdoteUmPet

Objetivo

Construir uma aplicação completa para **gerenciar pets para adoção** e **consultar informações de raças**.

- **Backend**: API REST em **qualquer linguagem** (Node, Python, Go, Java, .NET, etc.), com **MySQL ou PostgreSQL**.
- **Frontend**: **React** ou **VueJS**.

Requisitos do Backend

1) Banco de Dados (MySQL ou PostgreSQL)

Tabela `pets`:

- `id` (UUID)
- `name` (VARCHAR)
- `species` (ENUM ou CHECK: `'dog' | 'cat'`)
- `breed` (VARCHAR)
- `age_years` (INT)
- `shelter_city` (VARCHAR)
- `shelter_lat` (DECIMAL(10,7))
- `shelter_lng` (DECIMAL(10,7))
- `status` (ENUM/CHECK: `'available' | 'adopted'`, default `'available'`)
- `created_at` (TIMESTAMP, default now)

2) Endpoints

Método	Rota	Descrição
POST	`/pets`	Cadastrar pet (validações de campos + ranges de lat/l
GET	`/pets`	Listar com filtros (`name`, `species`, `breed`, `shelter_city`, `status`), **pag** e **ordenar** (`sortBy`, `order`). Retornar `{ total, page,`
GET	`/pets/:id`	Detalhes do pet.
GET	`/breeds/:species`	Integração com **TheDogAPI** (dog) ou **TheCatAPI** (cat). por nome. Normalizar a resposta para `{ name, origin` `energy_level, image_url }`.

3) Tecnologias no backend

- **Linguagem**: livre (documentar como rodar).
- **BD**: MySQL **ou** PostgreSQL (documentar migrações).
- É permitido ORM (Sequelize/Prisma/TypeORM/Knex/EF/JPA/etc.) ou SQL puro.
- Expor variáveis via `.env` (ex.: chaves de TheDogAPI/TheCatAPI).

Requisitos do Frontend

1) Páginas

- **Listagem de Pets**: filtros (nome, espécie, raça, cidade, status), paginação e ordenação.
- **Cadastro de Pet**: formulário com validações e feedback.
- **Detalhe do Pet**: informações completas + **mapa** (Leaflet/Google Maps) com localização do abrigo.
- **Explorar Raças**: escolher espécie, buscar por nome, exibir temperamento/origem/energia + imagem.

2) Funcionalidades

- **Dropdown** de cidades brasileiras (lista estática simples é suficiente).
- **Gráfico (Chart.js)**: distribuição de **idade** (faixas 0–1, 2–3, 4–6, 7+).
- **Mapa**: Leaflet (preferível) ou Google Maps na página de detalhe.

3) Técnico

- HTML5, CSS3, JS.
- React (Context/Zustand/Redux) ou Vue (Pinia/Vuex) para estado.
- `.env` com `VITE_API_URL` / `NEXT_PUBLIC_API_URL` (ou equivalente).

Critérios de Avaliação

- **Funcionalidade**: requisitos atendidos.
- **Qualidade de código**: organização, padrões, legibilidade (TS é um plus).
- **UX/UI**: responsivo, acessível, consistente.
- **Boas práticas**: camadas, tratamento de erros, .env, migrações/seed.
- **Integração**: consumo correto da API própria e da TheDogAPI/TheCatAPI.
- **Dados & Estado**: filtros, paginação, cache (se houver), UX de loading/empty/error.

Diferenciais

- **Testes** (backend e frontend) com **cobertura > 60%** (relatório).
- **Cache** de `/breeds/:species` e/ou filtros de pets.
- **Docker Compose** para subir **api + db + adminer/pgadmin**.
- **CI** (lint + testes).
- **Swagger/OpenAPI** no backend.
- **Seed idempotente** a partir do CSV.

Observações

- Pode usar bibliotecas/APIs e **IA** (desde que explique seu código).
- Layout livre.
- Dúvidas: testes.rh@appmoove.com.br
- Boa sorte! 🐶 🐱

Entrega

Prazo para entrega: 72 horas

- Enviar **link do GitHub** ou **.zip** para testes.rh@appmoove.com.br
- Incluir `README.md` com:
  - **Como rodar** (passo a passo) em sua linguagem escolhida.
  - Requisitos (versões), dependências e variáveis de ambiente.
  - **Como escolher o BD** (MySQL ou Postgres) e rodar **migrações/seed**.
  - Como iniciar **backend** e **frontend**.
  - Como rodar **testes** e gerar **cobertura**.
  - (Opcional) **Swagger** e exemplos de requisição.