Teste técnico para Pessoa Desenvolvedora Back-end Plena

1. Introdução

O objetivo deste teste é avaliar seus conhecimentos técnicos para a vaga que você se candidatou.

Imagine que a sua tarefa é criar uma API que será integrada posteriormente com um front-end desenvolvido por outro time.

Leia o documento abaixo com os detalhes e faça o que foi proposto.

2. Requisitos gerais

Para realizar o teste, colocamos alguns requisitos para padronizar o processo de desenvolvimento e entrega para avaliação.

O teste deverá ser desenvolvido utilizando o framework Laravel, em qualquer versão igual ou acima da 8.

2.1. Usar o Laravel Sail

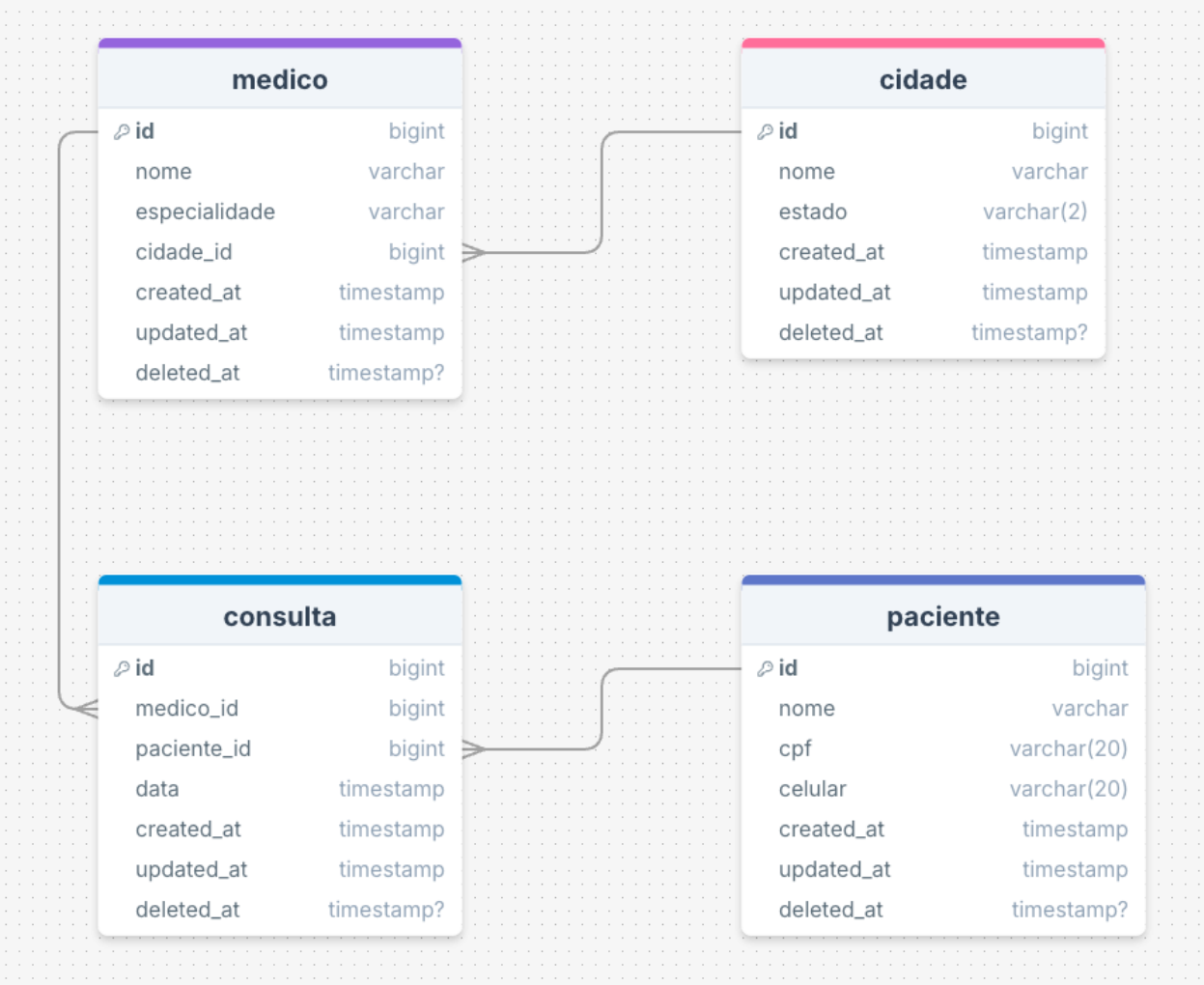
Para executar a aplicação e o banco de dados, pedimos que o projeto seja criado com o Laravel Sail. O único container, além do container da aplicação, que precisa ser instalado é o banco de dados Mysql. As instruções de criação de um projeto com o Laravel Sail estão na documentação.

2.2. Migrations e Seeders

Para criar a estrutura do Banco de dados, com as tabelas necessárias para executar o teste, esperamos que você utilize as Migrations.

Sobre a inserção dos dados, eles devem ser criados através de Factories e/ou Seeders.

O diagrama abaixo representa as tabelas, colunas e relações que o Banco de dados deve possuir:



2.3. Autenticação com o JWT

Para as rotas que necessitam de autenticação, você deverá utilizar o JWT como mecanismo de autenticação dos usuários. Considere que a model `User` do Laravel para gerenciar os usuários.

2.4. Postman

Um dos critérios da avaliação do teste será a execução dos endpoints no Postman. Para isso, criamos esta Collection. Você pode fazer uma cópia dela e utilizá-la para desenvolver o seu teste.

Na Collection está listado todos os endpoints avaliados, métodos e **exemplos de resultados enviados/retornados**.

3. Aplicação

O teste consiste em realizar o desenvolvimento de uma API para listar, adicionar e atualizar as cidades, médicos e pacientes cadastrados no banco de dados.

Veja a tabela abaixo:

Entidade	Listar	Cadastrar	Atualizar
Cidade	✅	🚫	🚫
Médico	✅	🔒	🚫
Paciente	🔒	🔒	🔒
Consulta	🔒	🔒	🚫

Legenda:

✅ Público, sem necessidade de autenticação

🚫 Não necessário no projeto

🔒 Privado, apenas usuário autenticado

A parte de listagem das cidades e médicos deverá ser pública. Porém, para cadastrar um novo médico, listar, cadastrar e atualizar um paciente ou consulta, só será permitido para os usuários autenticados.

Abaixo está descrito as especificações de cada entidade:

3.1. Cidade

3.1.1. Listar cidades

Para listar as cidades cadastradas, deve ser feito uma requisição `GET` para o *endpoint* `/cidades` e não é necessário estar autenticado.

Também deverá ser possível buscar por parte do nome da cidade através do parâmetro `nome` na URL.

A ordenação dos resultados deverá ser em ordem alfabética.

O retorno esperado é um *Array* de objetos JSON do tipo Cidade.

3.2. Médico

3.2.1. Listar médicos

Para listar todos os médicos cadastrados, deve ser feito uma requisição `GET` para o *endpoint* `/medicos`.

Também deverá ser possível buscar por parte do nome do médico através do parâmetro `nome` na URL. Neste filtro, deverá ser desconsiderado a procura por prefixo como "dr" e "dra".

A ordenação dos resultados deverá ser em ordem alfabética.

O retorno esperado é um *Array* de objetos JSON do tipo Medico.

3.2.2. Listar médicos de uma cidade

Para listar apenas os médicos de uma cidade específica, deverá ser feito uma requisição `GET` para o *endpoint* `/cidades/{id_cidade}/medicos`.

Também deverá ser possível buscar por parte do nome do médico através do parâmetro `nome` na URL. Neste filtro, deverá ser desconsiderado a procura por prefixo como "dr" e "dra".

A ordenação dos resultados deverá ser em ordem alfabética.

O retorno esperado é um *Array* de objetos do tipo Medico.

3.2.3. Agendar consulta

Para agendar uma consulta, deverá ser feito uma requisição `POST` para o *endpoint* `/medicos/consulta`.

No *body* da requisição deverá conter os parâmetros `medico_id` (com o ID do médico), `paciente_id` (com o ID do paciente) e a data da consulta no formato timestamp (Y-m-d H:i:s).

Este recurso só deve estar disponível para usuários autenticados.

O retorno esperado é a model Consulta criada.

3.3. Paciente

3.3.1. Listar pacientes do médico

Este é o *endpoint* responsável por retornar todos os pacientes que possuem consultas agendadas e/ou realizadas com o médico. Apenas usuários autenticados tem autorização para acessar esta rota.

Deverá ser feito uma requisição do tipo `GET` para o *endpoint* `/medicos/{id_medico}/pacientes`.

Este endpoint deve contar com um parâmetro opcional chamado `apenas-agendadas` que será um *boolean*. Se o parâmetro existir e for `true`, deverá retornar apenas consultas que ainda não foram realizadas.

Também deverá ser possível buscar por parte do nome do paciente através do parâmetro `nome` na URL.

A ordenação dos resultados deverá ser por ordem crescente da data da consulta.

E o retorno esperado é um array de objetos do tipo Paciente com a relação da model Consulta.

3.3.2. Atualizar paciente

Este *endpoint* será utilizado para alterar os dados de um paciente específico. Apenas usuários autenticados tem autorização para acessar esta rota.

Somente o nome e o celular podem ser alterados.

Este recurso será acessado através da rota `/pacientes/{id_paciente}` com o método `POST`.

O retorno esperado é os dados do paciente atualizados em formato de objeto JSON.

3.3.3. Adicionar paciente

Este é o *endpoint* responsável por adicionar um novo paciente à base de dados. Apenas usuários autenticados tem autorização para acessar esta rota.

Deverá ser feito uma requisição do tipo `POST` para o *endpoint* `/pacientes`.

O retorno esperado é os dados do novo paciente em formato de objeto JSON.