Descrição do Teste:

Este teste tem como objetivo avaliar o seu conhecimento em PHP num contexto de API Rest. O teste vai desde conceitos básicos, como Programação Orientada a Objetos (POO), até o uso de frameworks avançados, como o CakePHP. Abaixo, serão detalhados em tópicos, as instruções que você deverá seguir para implementar as funcionalidades requisitadas.

Para remeter o seu teste, publique as alterações em um repositório público de sua preferência (GitHub, BitBucket, GitLab, etc.) e nos envie o link no email leandro@mundowap.com.br com o assunto "Teste para vaga de desenvolvedor back-end".

Conhecimentos básicos para execução do teste:

- Github;
- Docker;
- PHP;
- Programação Orientada a Objetos (POO);
- MVC;
- Composer;
- MySQL;
- CakePHP;
- REST;

Requisitos:

Os itens que serão descritos abaixo são **cruciais** para a consideração do teste, caso algum deles não seja cumprido, a avaliação poderá ser desconsiderada. **É expressamente proibido o uso de cópia ou geração via inteligência artificial dos códigos. Caso seja identificado, o teste será imediatamente desclassificado.**

- Utilize o **Composer** na instalação de pacotes externos;
- Utilize o padrão de projeto **MVC**.
    - Como se trata de uma API, as Views serão uma exceção nesse requisito;
- Organize as rotas utilizando os padrões **RESTful**;
- Criar **migrations** para fazer a construção do banco de dados;
- Aplicar a **Programação Orientada a Objetos (POO)** no projeto;
- Escrever e organizar o código atendendo às recomendações da **PSR-4**.

O que será avaliado no seu teste?

- Qualidade de leitura do código escrito, como tipagem de métodos e variáveis, comentários, declarações/importações de variáveis, classes e métodos não utilizados, entre outros;
- Utilização de recursos, funções do CakePHP, ao invés de criar bibliotecas internas;
- Lógica empregada na implementação;
- Fluxo de execução e performance, evitando loops e processamentos desnecessários.
- Tratativas e saídas na resposta da requisição dos erros (exemplo: parâmetros da requisição inválidos, erro de banco de dados, etc.).

O que será considerado um diferencial?

- Utilizar um sistema de Events e Listeners, exemplo: BeforeSave e AfterSave;
- Utilizar um sistema de Validators, de modo que todos os dados que entram através do cliente sejam devidamente verificados;

Instruções para Implementação:

- Leia o arquivo “README.md” na raiz do projeto base;
- Utilizando Migrations, crie as tabelas no banco de dados conforme o arquivo “*db_structure.sql*” na raiz do projeto base;
- Baseado nas tabelas criadas no banco de dados, crie os Models;
- Crie um Controller e rotas para disponibilizar ações de CRUD (criar, consultar, atualizar e deletar) referentes a tabela “stores”;
- A tabela “stores” deverá seguir as seguintes regras:
    - No cadastro ou na atualização, o campo “id” não deve ser acessível ao cliente para preenchimento;
    - O campo “name” não deve permitir valores vazios e no cadastro, deverá ser obrigatório;
    - No cadastro ou na atualização, sempre que o campo “name” for alterado, deverá ser consultado se já existe um outro registro com o mesmo valor, caso exista, um erro deverá ser emitido com a seguinte mensagem “Nome em uso”;
    - No cadastro ou na atualização, deve ser obrigatório cadastrar um registro na tabela “addresses” de forma aninhada (na mesma requisição);
    - Os registros na tabela de endereços não deverão ser editados, sempre substituídos, ou seja, sempre que o endereço de uma loja precisar ser alterado, deverá ser excluído o registro de endereço atual e criado um novo.
      Exemplo: suponhamos que no banco de dados tenhamos a loja de id 1 vinculada ao endereço de id 11. Nesse contexto, o esperado é que ao receber dados do endereço na requisição de edição da loja (id 1), ao invés de editar o registro de endereço atual (id 11), deverá ser feita a substituição do mesmo por um registro novo, ou seja, o endereço atual (id 11) será excluído e um novo registro vinculado a loja que está sendo editada (id 1) será criado na tabela de endereços com os novos dados recebidos da requisição. Deste modo será mantida a regra do relacionamento, onde uma loja tem apenas um registro de endereço;
    - Na exclusão, o registro vinculado a tabela “addresses” também deverá ser excluído (cascata, porém como o banco não tem chaves estrangeiras, o comportamento deve ser implementado no model);
    - Na consulta, deverá ser disponibilizado os dados do registro vinculado a tabela “addresses” para cada registro (join);
    - A tabela “addresses” deverá seguir as seguintes regras:
    - No cadastro, somente os campos “postal_code”, “street_number” e “complement” devem ser acessíveis para preenchimento;
    - No cadastro, os campos “postal_code” e “street_number” deverão ser obrigatórios e não podem conter valores vazios;
    - Sempre que o campo “postal_code” for alterado, deverá ser consultado o valor recebido utilizando integrações com APIs de CEP para buscar as demais informações do endereço e completar o cadastro. As consultas deverão seguir o seguinte fluxo: primeiro, faça a consulta utilizando a API República Virtual, caso os dados não sejam

encontrados, faça a consulta utilizando a api Via CEP, caso os dados ainda não sejam encontrados, um erro deverá ser emitido com a mensagem “CEP não encontrado”;

- Na consulta, deverá existir na resposta da requisição, para cada registro, a chave “postal_code_masked”, que corresponde ao valor da coluna “postal_code”, com uma máscara de CEP aplicada;

Agora é botar a mão na massa e dar o seu melhor. Boa sorte, candidato(a)!

Surgiu alguma dúvida? Entre em contato pelo e-mail leandro@mundowap.com.br