

**Funcionalidade: Começo do fluxo de pagamento**

|  |
| --- |
| **Cenário:** |
| Descreva os passos que você planeja executar para a implementação da funcionalidade. É muito importante que você descreva aqui antes da implementação em si. |

|  |
| --- |
| **O que seria bom ver nessa resposta?** |

* **Peso 0.5:**Criação do método para receber a requisição de um novo Cliente anotado com a configuração para receber um POST, configurado para validar entrada de dados e também para receber os dados como JSON.
* **Peso 2:**Utilização de uma classe específica para receber os dados do novo Cliente. O famoso DTO.
* **Peso 1:**Deve-se usar os ids para referenciar o Pais e a Estado no DTO
* **Peso 1**: Declaração dos outros atributos na classe
* **Peso 1:**Método de conversão dos dados do novo Cliente para um Cliente de fato dentro na própria classe DTO ou criação de uma classe de conversão de Dados de conversão
* O método de conversão precisa ter acesso ao EntityManager/Repositories. Caso seja uma classe específica de conversão, ela precisa ter um atributo do tipo EntityManager ou os Repositories adequados.
* **Peso 0.5:**Utilização das annotations de validação da Bean Validation na classe do DTO
* **Peso 0.3:**Por não colocar o @NotNull em cima do id do estado
* **Peso 0.2:**Para os outros atributos
* **Peso 1:**Criação da classe que representa o Cliente com as informações
* **Peso 0.8:**Precisa ter a referência aos atributos do tipo Pais e Estado.
* **Peso 0.2:**Outros atributos
* **Peso 0.5:**Utilização das annotations de validação da Bean Validation na classe do Livro
* **Peso 0.3:**Por não colocar o @NotNull em cima do id do estado
* **Peso 0.2:**Para os outros atributos
* **Peso 1:**Utilização do construtor na classe Cliente para deixar claro os dados que são obrigatórios
* **Peso 0.2:**Precisa receber parâmetros do tipo Pais no construtor
* **Peso 0.3:**Receber outros parâmetros
* **Peso 0.5:**Não recebeu o estado no construtor
* **Peso 1.5:**Utilização da annotation @ManyToOne para mapear o relacionamento com Pais e Estado.
* **Peso 1:**Utilização do EntityManager ou Repository específico para a inserção do Cliente
* **Peso 1:**Criação do validador customizado para verificar se o estado, quando passado, é realmente daquele país.
* **Pontuação bônus :**Some a sua nota 1 ponto extra se tiver criado um método no estado que verifica se ele é de determinado país. Este método pode ser chamado dentro do sistema para garantir que um estado sendo associado a um cliente realmente pertence àquele país do cliente.

|  |
| --- |
| **O que penaliza sua resposta?** |

* **Penalidade - 5:**Existência de Service no projeto. Não é necessária dada a especificação. Não tem nada lá que sugira a necessidade da flexibilidade que essa indireção traria.
* **Penalidade - 5:**Existência de biblioteca de geração de código compilado estilo Lombok. Não é necessário porque entendemos que o benefício trazido pelo código gerado não compensa o esforço de entendimento necessário que existe por trás de tal geração.

|  |
| --- |
| **Resposta do Especialista:** |
| * Crio um controller com um método para receber o post com os dados do novo cliente * O argumento do método do controller é do tipo de uma classe no estilo DTO para receber os dados do novo cliente. Ali também eu uso a anotação @Valid e @RequestBody para indicar a necessidade de validação e que o dado vem no corpo da requisição. * Na classe que tem os dados de entrada, utilizo as annotations da Bean Validation para sinalizar as validações necessárias * Ainda na classe que representa os dados de um novo livro, eu utilizo atributos do tipo Integer/Long para representar os ids do país e do estado. Uso os ids simplesmente porque é o identificador necessário para que possamos buscar os objetos relativos através do nosso ORM. * Mais um detalhe aqui. Enquanto o país é obrigatório, o estado não é. O sistema permite cadastrar países sem seus estados, assim como acontece na casa do código. * Dentro do método do controller eu converto os dados do DTO para a criação de um objeto do tipo Cliente. Faço isso através de um método adicionado no DTO que retorna um Cliente em função dos valores dos atributos do DTO. Aqui tem um detalhe especial. Para eu conseguir construir um objeto do tipo Cliente eu preciso de um objeto do tipo Pais e, talvez, outro do tipo Estado. Preciso sair do id do país para um País de fato, assim como preciso sair do id do Estado para o Estado de fato(caso tenha estado). Para fazer isso eu peço um EntityManager no método de conversão, para conseguir acessar o método findById. Se quisesse usar o princípio da segregação pela interface, eu poderia receber como argumento no método de conversão do DTO uma parâmetro do tipo Function e usar um method reference para passar a referência para o método find do EntityManager. * Crio uma validação extra, pelo Spring Validator, para verificar, em caso de definição de estado, se o estado pertence ao país indicado. Em caso de negativa, rejeita request. Preciso também adicionar essa validação no controller com o cadastro do novo cliente. * Crio a classe Cliente com os atributos devidamente anotados com a Bean Validation. Um detalhe é que no construtor eu vou receber uma referência para um objeto do tipo Pais. Não vou receber um estado, dado que estado não é obrigatório. * Crio um método setEstado para receber o estado nos casos de necessidade. * Dentro do método setEstado, além de atribuir a referência para o atributo, eu também invoco o método *estado.pertenceAPais(this.pais)*. Caso isso retorne false, eu solto um IllegalArgumentException indicando que aquele argumento não é correto. Lanço essa exception fazendo uso da classe Assert do Spring. Isso é o que chamamos de self testing. * Também faço o mapeamento na classe Cliente para que o Hibernate consiga persistir os objetos. Importante aqui a utilização @ManyToOne em cima do atributo do tipo Pais e tipo Estado. * Recebo injetado o EntityManager no controller e gravo o livro no banco de dados. * Deixo o Hibernate criar a tabela para mim no banco de dados. Poderia ter criado também o scritpt e executado contra o banco |