RESPOSTAS SPRING

1 – Qual a utilidade do Spring Boot?

* É facilitar o processo de configuração e publicação de aplicações

2 – Usando o que foi visto durante o curso, descreva todos os passos que você faria para conseguir receber os dados, validar, fazer com que as informações sejam persistidas no banco de dados e retornar um status 200 para a aplicação cliente em caso de sucesso ou 400 em caso de falha de validação.

* Primeiramente criar um projeto spring no site <https://start.spring.io>
* Configurar ou conferir as dependências necessárias no pom.xml já que na geração do projeto no spring initializr já setamos as dependências iniciais que queriamos.
* Configurar a conexão com o banco de dados através do arquivo application.properties que o spring boot já fornece.
* Criar um class entity com a anotação @Entity aluno, para ela ser persistida no banco de dados como uma tabela, com os atributos id com as anotações @Id para ser um identificador e @GeneratedValue para esse id ser gerado automaticamente, os atributos do tipo String nome e email e o atributo idade tipo Integer.
* Criar uma interface aluno repository que vai estender de JpaRepository para conectar as manipulações ao banco de dados pelo JPA/Hibernate.
* Criar a classe alunoDTO que será a classe de abstração da entity, que vamos usar no controlador, essa classe apenas será uma classe java bean, que terá apenas os atributos que queremos passar na requisição os getters e setters e um método de conversão de dto para entity.
* Criar um aluno controller com as anotações @RestController para dizer que aquela clsse se trata de um controller e a @RequestMapping com o caminha central da nossa uri.
* Dentro de Aluno controller injetar o repository para ter acesso ao bonco e todos os métodos da interface, criar um método post, esse método irá receber a anotação @PostMapping indicando que se trata de um post, e a anotação @Transactional indicando que será feita uma transação pelo JPA para que irá um ResponseEntity que ira retornar um AlunoDTO passando os parâmetros para e método de @RequestBoby que indica que a requisição ira receber os dados pelo seu corpo, @Valid para validar as anotações dos dados que será recebido e um AlunoDto como retorno.
* Criar um objeto Aluno através do método de conversão de um dto para entity para que conseguimos persistir para o banco pelo método save fornecido pelo repository, e retornar um ResponseEntity com o status ok indicando que foi criado um aluno com sucesso no banco dados, que irá retornar o status code 200.
* Para capturar erro utilizaremos um *Controller Advice , vamos criar uma classe que irá receber a anotação @RestControllerAdvice indicando que ela é um controller advice, rest pq estamos trabalhando com um rest controller , nessa classe vamos ter um método que ira capturar esse erro , através da @ExceptionHandler que terá como resposta uma mensagem e o erro capturado e o status code 400 bad request pelo response status.*
* Para acessar um aluno pelo id vamos criar um método dentro de controller, método getById , esse método recebe a anotação @GetMapping("/{id}") juntamente com path id para indicar o id recebido pela URI indicando que se trata de um método get e a anotação @Transactional indicando uma transação, esse método ira receber um ResponseEntity com um AlunoDTO especifico para get com apenas os atributos nome e email abstraídos nele, como parâmetro ira receber a anotação @PathVariable Long id, que indica que a requisição sera pela uri e vai receber um id do tipo Long.
* Criando a classe dto especifica para o get by id, vamos usar os atributos de nome e email que queremos retornar gerar apenas seus getters, como será uma class apenas para consulta, e gerar um construtor que rebera um aluno para realizar essa abstração para dto.
* No método getById novamente vamos criar um Optional de aluno para receber o método findById fornecido pela interface JPA repository, o método find by id devolve um optional, por isso o uso dele, e vamos realizar um if para que se esse aluno estiver presente, se ele existir retornar um ResposnseEntity ok devolvendo um new AlunoDTO pelo aluno.get(), dessa maneira vamos retornar apenas nome e email. Caso contrario retornar um ResponseEntity not found com o status 404, indicando que não foi encontrado o aluno pedido.



**Spring Boot API REST: Construa uma API**

|  |
| --- |
| **Cenário:** |
| Antes de pularmos para a parte da atividade onde você vai precisar descrever como faria determinada implementação, vamos começar pelo básico. Qual a utilidade do Spring Boot?  Agora, precisamos começar a construir um sistema para suportar nosso modelo de avaliações e respostas do Orange Talents. Para esse primeiro momento precisamos cadastrar novos(as) alunos(as) e também conseguir ver os detalhes de cada pessoa cadastrada.  Para a parte do cadastro, todo(a) aluno(a) precisa de um nome, email e idade. O nome não pode ter mais de 30 caracteres, o email também só pode ter no máximo 30 caracteres e a idade precisa ser maior ou igual a 18 anos. É importante que esses dados sejam cadastrados no banco de dados.   * Usando o que foi visto durante o curso,  descreva todos os passos que você faria para conseguir receber os dados, validar, fazer com que as informações sejam persistidas no banco de dados e retornar um status 200 para a aplicação cliente em caso de sucesso ou 400 em caso de falha de validação.   Agora que o cadastro foi feito, é necessário que os detalhes de cada aluno(a) possam ser acessados. Uma restrição importante aqui, a identificação do(a) aluno(a) será feita pelo id do banco de dados e deve fazer parte do endereço de acesso. Para o detalhe, só precisamos exibir o nome e o email.   * Usando o que foi visto durante o curso,  descreva todos os passos que você faria desde conseguir tratar a requisição feita para determinado endereço até retornar as informações do(a) aluno(a) em formato JSON. |

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

* **Peso 1:**Na resposta sobre o motivo do Spring Boot é importante aparecer algo próximo como "Facilitador de configuração"
* **Peso 0.5:***Para o cadastro*: Método no controller mapeado para um endereço acessado via post
* **Peso 2:***Para o cadastro*: Classe criada especificamente para receber os dados. O famoso DTO
* **Peso 0.5:***Para o cadastro*: Uso das anotações padrões da Bean Validation para realizar as validações.
* **Peso 0.5:***Para o cadastro*: Criação de um objeto do tipo Aluno em função dos dados recebidos no método do controller.
* **Peso 0.5:***Para o cadastro*: Configuração do acesso ao banco de dados no application.properties ou yaml.
* **Peso 0.5:***Para o cadastro*: Utilização do EntityManager ou Repository do Spring Data JPA para gravar o objeto do tipo Aluno no banco de dados escolhido.
* **Peso 2: 0.5:***Para o cadastro*: Retorno void no método do controller ou usando ResponseEntity para informar o status 200.
* **Peso 0.5:***Para o detalhe*: Método no controller mapeado para um endereço acessado via get
* **Peso 0.5:***Para o detalhe*: Parâmetro que representa o id do aluno fazendo parte do endereço
* **Peso 0.5:***Para o detalhe*: Utilização de método padrão do Repository ou uso direto do EntityManager para carregar o objeto do tipo Aluno pelo id.
* **Peso 2.0:***Para o detalhe*: Utilização dos dados do objeto aluno para montar um objeto de saída com os dados específicos.
* **Peso 0.5:***Para o detalhe*: Retorno do objeto de saída no método. Também vale a utilização do ResponseEntiy.

|  |
| --- |
| **Resposta do Especialista:** |
| * O Spring boot serve para facilitar a configuração de projetos que querem utilizar a stack do Spring. Através de auto configurações, quase sempre fornecida por starters, a configuração das tecnologias é bem facilitada. * Crio a classe Aluno, adiciono os atributos necessários e também adiciono um id. Adiciono o Id para mapear como chave primária para o banco de dados. * Mapeio a classe Aluno para ser entendida pelo HIbernate. Uso o @Entity em cima da classe e o @Id em cima do atributo id. Poderia colocar também o @Id no email. A @Entity explica para o Hibernate que aquela classe vai ser uma tabela no banco e o @Id explica que aquele atributo vai ser a chave primaria na tabela. Os outros atributos eu não preciso mapear pq ele já vai mapear para colunas do mesmo nome. * Já configuro o acesso ao banco de dados no application.properties com o login/senha e banco de dados local específico. * Crio o controller relativo a alunos * Crio o método que vai receber os dados de cadastro e anoto com @PostMapping * Crio a classe que representa os dados de entrada e anoto cada atributo com as anotações específicas da Bean Validation. Nesse caso poderia usar @Length, @Email, @Min * Crio um método na classe que representa os dados de entrada(DTO) que é responsável por criar um objeto do tipo Aluno a partir dos dados de entrada * Crio a classe que representa o Aluno * Mapeio a classe para a utilização do Hibernate * Volto no controller, recebo injetado o EntityManager e salvo o objeto criado do tipo Aluno. Podia ter usado um Repository também, mas não é necessário. * Deixo o retorno do método como void. * Agora eu crio outro método no controller mapeado para receber get * Coloco o id como parte do endereço * Já defino que o retorno vai ser um objeto do específico para a saída do detalhe apenas com nome e email. * Recebo o id como argumento do método usando a annotation @PathVariable * Utilizo o EntityManager dentro do método para carregar o objeto do tipo Aluno pelo id. * Instancio o objeto do tipo de saída, passo o objeto do tipo aluno como argumento no construtor  e já retorno ele no método. * Agora eu crio a classe que representa a saída, com os atributos específicos, além do construtor. Adiciono os getters, e apenas eles, para expor os dados que preciso. |