Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 11 – TRATAMENTO DE RETORNO DA API NAS REQUISIÇÕES

Padronizando Retorno da API

Spring

- Não colocamos nenhum retorno no método de listar e de excluir.
- Vejam no "MedicoController.java".
- Devolvemos algo somente no método de listar, porque precisamos retornar algo, no caso, estamos usando a paginação do Spring Boot.
- Porém, os outros métodos estão devolvendo um void. Isso é um problema!
- Ao usarmos o void, não informamos o que o Spring precisa devolver.
- Por isso, por padrão, ele retorna o código 200 OK, se a requisição for processada com sucesso.

Spring

- Porém, há outros códigos HTTP mais customizados dependendo do cenário.
- Por exemplo, no método excluir, o mais adequado seria devolver o código 204, que se refere à requisição processada e sem conteúdo.
- Para devolver o código 204, começaremos a padronizar esses métodos.
- Ao invés de usarmos o void, utilizaremos uma classe do Spring chamada ResponseEntity, sendo uma classe que conseguimos controlar a resposta devolvida pelo framework.
- Vejam no "MedicoController.java".
- Faça um requisição de listar para pegar um id do médico para excluir, note que ao excluir, ele nos devolve o código 200 OK.

```
 @DeleteMapping("/{id}")
 @Transactional
     public void excluir(@PathVariable Long id) {
         var medico = repository.getReferenceById(id);
         medico.excluir();
}
```

- Vamos alterar para :

```
@DeleteMapping("/{id}")
@Transactional
    public ResponseEntity excluir(@PathVariable Long id) {
        var medico = repository.getReferenceById(id);
        medico.excluir();

        return ResponseEntity.noContent().build();
    }
```

Spring

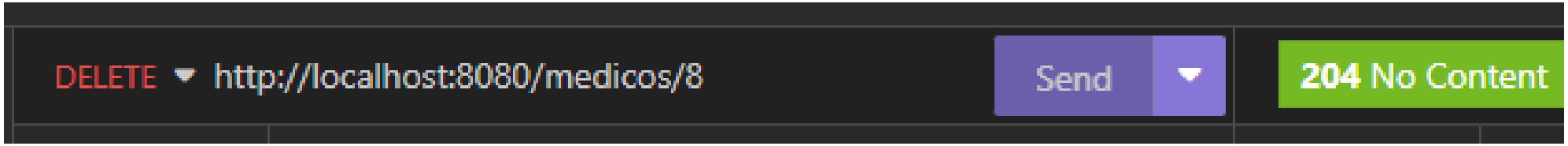
- Assim, o retorno não será mais vazio ("void") e sim um objeto do tipo ResponseEntity.
- Na última linha do método excluir, após a exclusão, adicionamos o return. Contudo, como instanciamos o objeto ResponseEntity?
- Na classe ResponseEntity há métodos estáticos, que podemos usar neste caso.
- No método noContent é usado para excluir.
- noContent() ele não devolve um objeto ResponseEntity, o mesmo cria um objeto e chamamos o build() para construir o objeto ResponseEntity.

Spring

- Vamos entender melhor o ResponseEntity.
- Para controlar o status code e cabeçalhos da resposta que será devolvida pela API, devemos utilizar a classe ResponseEntity, ao invés de devolver diretamente o DTO.
- Quando um método no controller é void ou seu retorno é um DTO, o Spring sempre vai devolver o código HTTP 200 400 500, sem/com cabeçalhos adicionais e sem/com um corpo na resposta.

Spring

- Exclua um médico, note que agora ele retorna



- Retorne ao medicoControler.....

- No método Lista

```
public Page<DadosListagemMedico> listar(
        @PageableDefault(size = 10, sort = {"nome"})
        Pageable paginacao){
```

- Altere para

```
public ResponseEntity<Page<DadosListagemMedico>>
listar(@PageableDefault(size = 10, sort = {"nome"}) Pageable
paginacao) {
```

Spring

- O erro acontece porque precisamos alterar o método para devolver um objeto ResponseEntity
- Faremos os devidos ajustes
- No lugar do return, vamos criar uma variável chamada page

```
return repository.findByAtivoTrue(paginacao).
```

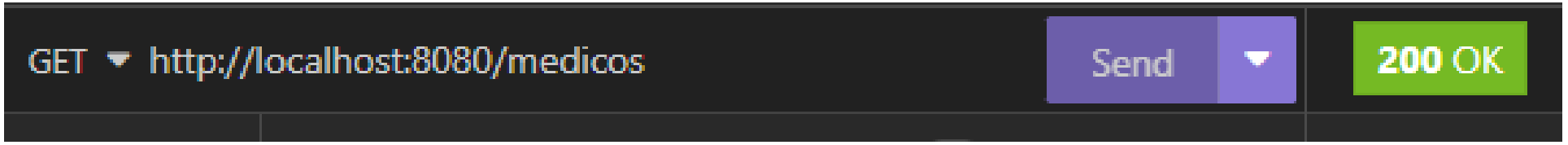
- Ficando assim

```
var page = repository.
```

- Na linha seguinte, faremos o return ResponseEntity.ok(), porque queremos devolver o código 200.
- No parênteses do ok(), passamos o objeto "page".

```
return ResponseEntity.ok(page);
```

- Com isso, no método de listar será devolvido o código 200, e junto na resposta vem o objeto de paginação com os dados dos médicos.
- Realize uma requisição de Listar, note que no corpo está a página “page”



Spring

- Vamos no Método Atualizar.
- Mude de void para ResponseEntity.

```
public ResponseEntity atualizar(@RequestBody @Valid
DadosAtualizacaoMedico dados) {
        var medico = repository.getReferenceById(dados.id());
        medico.atualizarInformacoes(dados);
    }
```

- Neste método, usávamos void, ou seja, não tínhamos retorno nenhum.
- Porém, diferente do método de excluir, no de atualizar não podemos devolver um código 204.

- O mais interessante no método de atualizar é devolver a informação atualizada.
- Como é para atualizar o registro do médico, no final devolveremos os dados do médico atualizado.
- No final do método digitaremos return ResponseEntity.ok() e dentro dos parênteses precisamos passar o objeto,
- Contudo, **não podemos passar o objeto médico**, porque é uma entidade JPA ("Java Persistence API") e **não é recomendado devolver e receber entidades JPA no controller**.

Spring

- Logo, precisamos devolver um DTO ("Data Transfer Object").
- Por exemplo, temos o DadosAtualizacaoMedico que é o nosso DTO.
- Clicando em "DadosAtualizacaoMedico", conseguimos visualizar o código.
- Porém, esse DTO está incompleto, ele possui somente o id, nome, telefone e endereço.
- Sendo o DTO que representa os dados da atualização de um médico, isto é, os dados que o aplicação enviará para atualizar as informações.

Spring

- Logo, precisamos devolver um DTO ("Data Transfer Object").
- Por exemplo, temos o DadosAtualizacaoMedico que é o nosso DTO.

```
@Transactional
public ResponseEntity atualizar(@RequestBody @Valid
DadosAtualizacaoMedico dados) {
        var medico = repository.getReferenceById(dados.id());
        medico.atualizarInformacoes(dados);

        return ResponseEntity.ok(new DadosDetalhamentoMedico(medico));
    }
```

- Perceba que DadosDetalhamentoMedico está escrito em uma cor vermelha bem forte, isso significa que ocorreu erro de compilação.
- Para ajustar isso, selecionaremos "Alt + Enter" no teclado, será exibido um pop-up com diversas opções, escolheremos a "Create record 'DadosDetalhamentoMedico'". Isso para ele criar esse DTO para nós.
- Vamos alterar de "med.voll.api.controller", para "med.voll.api.medico",
- Será criada o record 'DadosDetalhamentoMedico

```
public record DadosDetalhamentoMedico(Medico medico) {


}
```

Spring

- Teremos um record não recebendo o objeto médico e sim as informações do médico.
- No método estamos instanciando um objeto médico, logo podemos criar um construtor que recebe um objeto do tipo medico.
- Esse construtor chama o construtor principal do record passando os parâmetros.

Spring

```
public record DadosDetalhamentoMedico(
        no usages
        Long id, String nome, String email,
        no usages
        String crm, String telefone, Especialidade especialidade,
        no usages
        Endereco endereco) {
```

Spring

```
public DadosDetalhamentoMedico (Medico medico) {
    this(medico.getId(),
            medico.getNome(),
            medico.getEmail(),
            medico.getCrm(),
            medico.getTelefone(),
            medico.getEspecialidade(),
            medico.getEndereco());
}
}
```

- O método de cadastrar será um pouco diferente, porque existe o código 201 do protocolo HTTP que **significa que a requisição foi processada e o novo recurso foi criado**.
- Esse código possui um tratamento especial.
- Usaremos o objeto DadosDetalhamentoMedico. Inclusive, podemos criar um novo endpoint (ou método) no controller, dado que temos somente os quatro métodos do CRUD (Create, Read, Update e Delete). Porém, faltou o método para detalhar.
- Faremos isso mais a frente....

Spring