Lista 28 – Paginação, Filtros e Thymeleaf

Esta lista usará o *back-end* de cadastro de funcionários que vocês entregaram na lista 26.

A tarefa desta lista consiste em adicionar a funcionalidade de busca paginada com filtro visto na aula do dia 03.03. Para isso se baseiem no projeto exemplo_livro_paginacao_filtro (03.03 - final).zip. Vocês devem adaptá-lo para que funcione com a tabela de funcionários dos exercícios da lista 26.

A apresentação atualizada aula 50b - projeto básico spring (09.02.2023).pdf também pode auxiliar. Ela mostra os passos para se fazer a consulta com o exemplo feito em aula da tabela de livros:

1. *Slide* 116: Adição do método de busca com suporte a paginação. A busca deverá ser feita na tabela de funcionários pelo campo `nome`.
2. *Slide* 117: Declaração do método que será implementado, que faz a busca paginada com filtro.
3. *Slide* 118: Implementação, na classe de serviço, do método que faz a busca paginada com filtro.
4. *Slide* 119: Adição do método no controlador REST que acrescenta um *end-point* de busca paginada com filtro. Lembrem de alterar o *end-point* para outro nome, como `/buscar-paginas` de modo a não conflitar com o *end-point* `/paginas` já existente. Este novo *end-point* pode ser usado para vocês testarem com o navegador ou Postman se as consultas estão funcionando, como por exemplo:
   - http://localhost:8080/api/funcionarios/buscar-paginas
   - http://localhost:8080/api/funcionarios/buscar-paginas?tamPagina=3
   - http://localhost:8080/api/funcionarios/buscar-paginas?tamPagina=3&pagina=1
   - http://localhost:8080/api/funcionarios/buscar-paginas?tamPagina=3&pagina=0&ordenacao=titulo
   - http://localhost:8080/api/funcionarios/buscar-paginas?tamPagina=3&pagina=0&ordenacao=paginas&direcao=DESC
5. *Slide* 120: Criação de um *controller* convencional que deverá ser feito no *end-point* `/funcionarios` e implementação da consulta paginada com filtro bem semelhante ao que foi feito no controlador REST. Não é necessário apagar a consulta implementada no item anterior, vocês podem manter as duas no código.
6. *Slide* 121: Adição da dependência do Thymeleaf.
7. *Slide* 122 em diante: Consulta usando Thymeleaf no arquivo $paginacao.html$.
8. Acrescente a seguinte dependência no $pom.xml$ para que os ícones de edição e exclusão sejam mostrados nas consultas.
```
<dependency>
	<groupId>org.webjars.bowergithub.iconic</groupId>
	<artifactId>open-iconic</artifactId>
	<version>1.1.1</version>
</dependency>
```
9. Acrescente um novo campo `endereco` na entidade `Funcionario`. Modifique a classe `FuncionarioDTO` para que ela suporte este novo campo. Altere todo o restante do sistema, como por exemplo os métodos `alterar()`, `commandLineRunner()`, `popularTabelaInicial()`, etc, a fim de levar em consideração este novo campo. Finalmente, modifique o arquivo $paginacao.html$ para que o novo campo apareça na tabela com os resultados da consulta.
10. Como exercício extra, modifique a funcionalidade de filtragem da busca para que ela seja aplicada no campo `endereco` em vez de no campo `nome`.
11. Após verificar que o item 10 acima está funcionando, desfaça essas modificações de forma que a busca funcione com o campo `nome` como anteriormente 😊.

Após finalizar o sistema desenvolvido acima e se certificar que ele está funcionando corretamente, crie um repositório no GitHub para armazená-lo na nuvem. Utilize o GitBash para criar um repositório local e conectá-lo ao repositório no GitHub. Faça o *commit* com a versão atual do sistema e então realize o *push* para enviar o projeto para o repositório remoto.