Como criar um CRUD de livros para web

Para fazer o CRUD que antes funcionava no terminal funcionar para a web, não é necessário mudar muitos arquivos do projeto. Esse é um tutorial simples de como fazer isso.

Criando os arquivos

1- Começamos criando um projeto Spring Boot, no site https://start.spring.io/ . Depois de baixar, abra a pasta onde contém a pasta “src” no vs code ou em outro programa de sua preferência e coloque o seu projeto antigo dentro da pasta: src/main/java/com/nome_do_dominio/nome_do_projeto.

2- Após isso, você deve adicionar à todos os seus arquivos java a linha “package com.nome_do_dominio.nome_do_projeto” antes do código, para indicar que esses arquivos fazem parte do projeto. Certifique-se que o caminho da pasta corresponde exatamente ao nome do pacote, senão o código não compila. Exemplo: se o pacote é com.livros.projeto, o caminho deve ser src/main/java/com/livros/projeto .

3- Crie, dentro de “src/main/java/com/nome_do_dominio/nome_do_projeto”, um arquivo que vai ser responsável por iniciar a aplicação. Ele é o ponto de entrada da aplicação Spring Boot. No caso do meu grupo o arquivo é LivrosApplication.java.

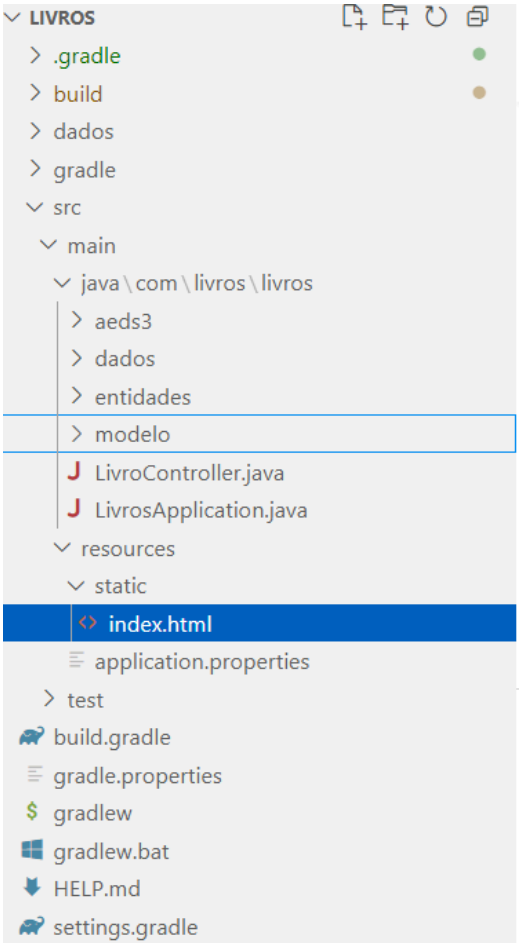
4- Também dentro de “src/main/java/com/nome_do_dominio/nome_do_projeto”, crie outro arquivo, que vai ser responsável pelas requisições. No caso do meu grupo ele se chama LivroController.java.

5- Em “src/main/resources/static” crie o seu index.html e seus arquivos javascript e css, caso necessário.

6- Juntos, esses arquivos criados nos números 3, 4 e 5 serão os responsáveis por fazer o programa funcionar na web. Se você já tem arquivos backend, como por exemplo .db, arquivos que estabelecem como os dados serão armazenados, etc, não será necessário alterá-los.

Estrutura

Já foi dito como estruturar o projeto, mas para ficar mais claro, um exemplo de estrutura pode ser visto abaixo:



Application

O código desse arquivo é bem simples. Ele pode ser escrito assim:

```
package com.livros.livros;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class LivrosApplication {
    public static void main(String[] args) {
        SpringApplication.run(LivrosApplication.class, args);
    }
}
```

Como dito antes, sua função é apenas iniciar a aplicação.

Controller

Esse é o controlador REST, que irá expor os endpoints. Na página web, quando o usuário quiser fazer algo, como por exemplo pesquisar um livro, o javascript irá até esse endpoint e o Controller irá fazer a requisição ao backend, assim devolvendo o livro procurado. No código do Controller você deve escrever o tipo de mapping (post, get, delete, put, patch ou request), o que será chamado e o que será devolvido. Um exemplo de get pode ser visto abaixo:

```
@GetMapping("/isbn/{isbn}")
    public Livro buscarPorISBN(@PathVariable String isbn) {
        try {
            if (!Livro.isValidISBN13(isbn)) {
              throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
              "ISBN inválido. Deve conter exatamente 13 dígitos.");
            }
            Livro livro = arqLivros.readISBN(isbn);
            if (livro == null) {
                throw new ResponseStatusException(HttpStatus.NOT_FOUND,
                "Livro não encontrado com ISBN: " + isbn);
            }
            return livro;
        } catch (ResponseStatusException e) {
            throw e;
        } catch (Exception e) {
            e.printStackTrace();
            throw new
            ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
            "Erro interno ao buscar o livro.");
        }
    }
```

Pode-se observar que esse código chama o método .readISBN, do ArquivoLivros.java (arqLivros). No caso da resposta ser null, ele passa a mensagem de que o livro não foi encontrado para o javascript. Se o livro for encontrado, seus dados são passados para o javascript.

index.html

Aqui, se você decidir usar html, css e javascript apenas nesse arquivo, ele será responsável por customizar a página e fazer os acessos aos endpoints, além de estabelecer a URL. No caso da URL, o padrão é http://localhost:8080, mas como no caso do meu grupo no Controller nós colocamos que a URL base terminaria em

/livros, a nossa seria assim: http://localhost:8080/livros. Em alguns casos a porta 8080 pode estar sendo usada por outro programa, então, caso queira, para trocar de porta você deve criar um arquivo (ou editar se ele já existir) chamado “application.properties”, dentro de “resources” e escrever:

```
spring.application.name=nome_da_aplicacao
server.port=8081
```

ou alguma outra porta que você prefira.

Um exemplo de função para buscar um livro por ISBN em javascript pode ser visto abaixo:

```
function buscarPorISBN() {
	let isbn = document.getElementById("isbnSearch").value;
	if (isbn.length !== 13 || !/^\d+$/.test(isbn)) {
	     document.getElementById("resultadoISBN").textContent = "ISBN
inválido (devem ser 13 números)";
	return;
	}
	fetch(`${baseUrl}/isbn/${isbn}`)
	.then(res => {
	if (!res.ok) {
	  if (res.status === 400 || res.status === 404) {
	    throw new Error("Livro não encontrado");
	  }else {
	    throw new Error("Internal server error");
	  }
	}
	return res.json();
	})
	.then(data => {
	  const output = `
	  <strong>Título:</strong> ${data.titulo}<br>
	  <strong>Autor:</strong> ${data.autor}<br>
	  <strong>ISBN:</strong> ${data.isbn}<br>
	  <strong>Edição:</strong> ${data.edicao}<br>
	  <strong>Data de Lançamento:</strong> ${data.dataLancamento}<br>
	   <strong>Preço:</strong> R$ ${data.preco.toFixed(2).replace(".",
",")}
	  `;
	  document.getElementById("resultadoISBN").innerHTML = output;
	})
	.catch(err => {
```

```
                document.getElementById("resultadoISBN").textContent  =
err.message;
      });
    }
```

Como rodar o projeto?

Abra o terminal, ir até a pasta onde src está localizado e digitar: ./gradlew bootRun --args='--server.port=8081' , especificando a porta, ou apenas ./gradlew bootRun, sem especificar. Depois disso, basta abrir o navegador e ir em http://localhost:8080 ou em outro http que você determinou.