### Exercício 9.6 - Tutorial Passo-a-Passo

Interface de Utilizador (UI) com Java, usando Vaadin Flow

## **Introdução ao Uso do Vaadin Flow para fazer a UI**

Se o **Vaadin Flow** for servido na mesma origem que a API Spring Boot (como é o caso ao adicionarmos a interface diretamente no projeto da solução do exercício 9.3), **não será necessário configurar o CORS**. Isto porque a aplicação estará totalmente integrada e todos os recursos (UI e endpoints da API) serão disponibilizados a partir da mesma origem (domínio e porta). Esta característica é uma das principais vantagens do **Vaadin Flow** na integração com o Spring Boot.

## **Porquê o Vaadin Flow?**

O **Vaadin Flow** oferece várias vantagens na integração com o Spring Boot, destacando-se pela facilidade de desenvolvimento e manutenção em aplicações full-stack. Aqui estão os principais benefícios relevantes para este contexto:

### 1. Mesma Origem, Sem Problemas de CORS

* Quando o Vaadin é integrado no mesmo projeto Spring Boot, o servidor único serve tanto a interface do utilizador (UI) quanto a API backend.
* Como ambos são entregues a partir do mesmo domínio e porta, **não há necessidade de configurar o CORS**, evitando potenciais problemas de segurança e tempo gasto em configurações adicionais.

### 2. Integração Nativa com Spring Boot

* O Vaadin Flow oferece suporte nativo para Spring Boot, com bibliotecas dedicadas como o vaadin-spring-boot-starter.
* Permite a utilização de componentes Spring, como serviços e repositórios, diretamente na camada de interface do utilizador.
* A anotação @Route simplifica a definição de páginas e a navegação.

### 3. Menos Dependência de JavaScript

* O Vaadin Flow permite desenvolver interfaces de utilizador modernas **apenas com Java**, sem necessidade de escrever JavaScript, HTML ou CSS separadamente (embora seja possível personalizar).
* Isto é ideal para equipas que preferem uma abordagem baseada em Java ou que já possuem competências em Java.

### 4. Componentes Prontos para Produção

* O Vaadin fornece uma vasta biblioteca de componentes UI (grids, formulários, botões, etc.) que já incluem funcionalidades como validação, responsividade e acessibilidade.
* Esses componentes reduzem o tempo de desenvolvimento, permitindo construir rapidamente interfaces completas.

### 5. Atualizações em Tempo Real

* Com o suporte a WebSockets e eventos de servidor, o Vaadin Flow permite atualizar automaticamente a interface do utilizador quando os dados no backend mudam.
* Isso facilita a construção de aplicações altamente interativas sem esforço adicional.

### 6. Arquitetura Simplificada

* A UI é construída como parte da aplicação Spring Boot, eliminando a necessidade de configurar pipelines de construção separados para o frontend (como no caso de frameworks SPA tradicionais, por exemplo, React ou Angular).
* Isto reduz a complexidade e o esforço de configuração.

## Em Resumo

Ao utilizar o Vaadin Flow para criar a interface da solução do exercício 9.3, todos os recursos são servidos a partir de uma única origem. Isso não só elimina a necessidade de configurar o CORS, mas também simplifica o processo de desenvolvimento e integração. Além disso, a forte integração com o Spring Boot e os componentes UI prontos para produção tornam o Vaadin uma escolha eficiente e produtiva para aplicações full-stack.

Vamos ver como implementar a UI nas páginas seguintes.

## Passo 1: Configurar o Vaadin no Projeto

**Abrir o ficheiro pom.xml:** No projeto da solução do exercício 9.3, localize o ficheiro **pom.xml**.

**Adicionar a dependência do Vaadin:** No bloco <dependencies>, adicione a seguinte dependência:

* <dependency>  
   <groupId>com.vaadin</groupId>  
   <artifactId>vaadin-spring-boot-starter</artifactId>  
   <version>24.6.1</version> <!-- Confirme a versão mais recente -->  
  </dependency>

**Atualizar as dependências do Maven:** Execute o seguinte comando para baixar as novas dependências:

./mvnw clean install

## 

## Passo 2: Criar o Pacote para a Interface de Utilizador

**Criar o pacote ui:** Navegue para o diretório src/main/java/com/example/livros/ e crie uma nova pasta chamada ui.

**Criar a classe LivroView:** Dentro do pacote ui, crie o ficheiro LivroView.java e faça copy-paste do código apresentado na página seguinte:

* package com.example.livros.ui;
* import com.example.livros.model.Livro;
* import com.example.livros.service.LivroService;
* import com.vaadin.flow.component.button.Button;
* import com.vaadin.flow.component.grid.Grid;
* import com.vaadin.flow.component.notification.Notification;
* import com.vaadin.flow.component.textfield.TextField;
* import com.vaadin.flow.component.orderedlayout.VerticalLayout;
* import com.vaadin.flow.router.Route;
* import org.springframework.beans.factory.annotation.Autowired;
* @Route("livros")
* public class LivroView extends VerticalLayout {
* private final LivroService livroService;
* private final Grid<Livro> grid = new Grid<>(Livro.class);
* private final TextField tituloField = new TextField("Título");
* private final TextField autorField = new TextField("Autor");
* private final Button saveButton = new Button("Guardar");
* @Autowired
* public LivroView(LivroService livroService) {
* this.livroService = livroService;
* configureGrid();
* configureForm();
* add(tituloField, autorField, saveButton, grid);
* updateGrid();
* }
* private void configureGrid() {
* grid.setColumns("id", "titulo", "autor");
* grid.getColumns().forEach(column -> column.setAutoWidth(true));
* }
* private void configureForm() {
* saveButton.addClickListener(event -> {
* String titulo = tituloField.getValue();
* String autor = autorField.getValue();
* if (titulo.isEmpty() || autor.isEmpty()) {
* Notification.show("Por favor, preencha todos os campos.");
* return;
* }
* Livro novoLivro = new Livro();
* novoLivro.setTitulo(titulo);
* novoLivro.setAutor(autor);
* livroService.guardarLivro(novoLivro);
* Notification.show("Livro guardado com sucesso!");
* tituloField.clear();
* autorField.clear();
* updateGrid();
* });
* }
* private void updateGrid() {
* grid.setItems(livroService.listarLivros());
* }
* }

## Passo 3: Verificar o Serviço LivroService

1. **Confirmar que o método listarLivros existe:** Abra o ficheiro LivroService.java no pacote service e certifique-se de que o método listarLivros está implementado para devolver uma lista de livros:

* public List<Livro> listarLivros() {  
   return livroRepository.findAll();  
  }

1. **Confirmar que o método guardarLivro existe:** Certifique-se de que há um método guardarLivro para persistir novos livros:

* public Livro guardarLivro(Livro livro) {  
   return livroRepository.save(livro);  
  }

## Passo 4: Configurar o Ficheiro application.properties

1. **Definir a porta para desenvolvimento:** Abra o ficheiro application.properties na pasta src/main/resources e adicione a seguinte linha (caso ainda não exista):

* server.port=8080

1. **Configuração adicional (se necessário):** Certifique-se de que as propriedades do datasource estão corretamente configuradas.

## Passo 5: Testar a Aplicação

1. **Executar a aplicação:** Utilize o comando:

* ./mvnw spring-boot:run

1. **Aceder à interface no navegador:** Abra o navegador e aceda ao endereço:

* http://localhost:8080/livros
* Verifique se a interface está a funcionar e se permite adicionar, listar e guardar livros.

## Passo 6: Refinar a Interface (Opcional)

* Adicione validações ou melhorias, como filtros de pesquisa ou paginação.
* Personalize o estilo visual com o CSS do Vaadin.