

Programação I

**Persistência de Dados
Spring / Hibernate**

Samuel da Silva Feitosa

Aula 23

Persistência de Dados

- Persistência de dados é fazer com que dados sejam **armazenados / gravados no computador**, possibilitando sua **recuperação** em outra execução do programa.
 - O armazenamento desses dados pode acontecer de diferentes maneiras.
- Exemplos:
 - Bancos de dados.
 - Arquivos no disco do computador.
 - Armazenamento em nuvem.
 - Etc.

Arquitetura Utilizada

- Nesta aula vamos utilizar uma arquitetura moderna de desenvolvimento, muito utilizada no mercado atualmente.
 - Vamos desenvolver WEB Services REST, que vão ser responsáveis por disponibilizar um CRUD (Create, Read, Update e Delete) em um banco de dados em memória.
 - Este formato é o padrão atual de novos projetos WEB (backend).
- Para a interface com o usuário utilizaremos a biblioteca Swing que já estamos utilizando.
 - Geralmente, são utilizadas bibliotecas de JavaScript (React, Angular, Vue, etc.) para desenvolvimento do frontend em aplicações WEB.
 - Em nosso projeto, vamos ‘consumir’ os WEB Services REST via Java.

Projeto - Lista de Tarefas

- Vamos criar uma aplicação muito simples, que permite ao usuário informar uma determinada tarefa, a qual será adicionada a uma lista de tarefas em um banco de dados.
- Para o backend, vamos criar a estrutura e os WEB Services para permitir a criação, leitura, atualização e exclusão de tarefas.
- Para o frontend, vamos criar as telas com os campos para entrada de dados e os botões responsáveis por interagir com o backend.

Preparação

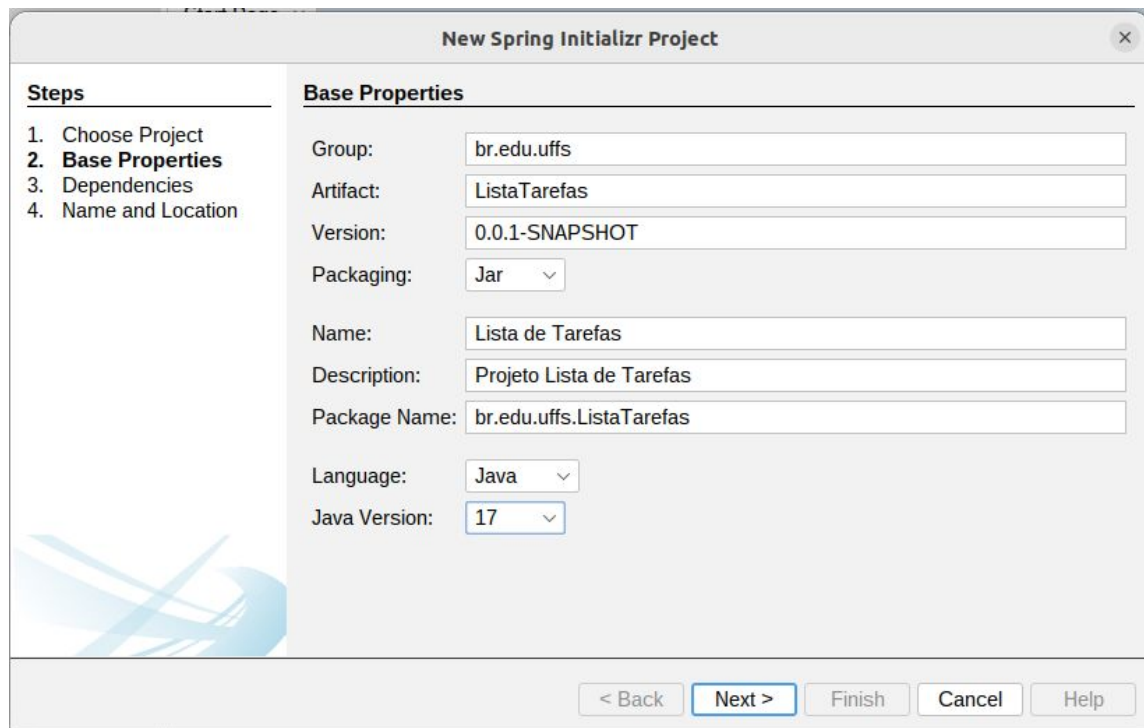
- Primeiramente, para facilitar o processo, vamos baixar o **plugin** do Netbeans para trabalhar com o SpringBoot.
 - **Spring** é um framework criado com o objetivo de facilitar o desenvolvimento de aplicações, explorando para isso, os conceitos de Inversão de Controle e Injeção de Dependências.
 - Não precisa de um servidor para rodar, utiliza apenas aquilo que é necessário para o projeto.
 - **Spring Boot** é um framework (ou extensão do Spring) que facilita a criação de aplicações Spring, possibilitando a execução imediata.
 - Permite gastar o mínimo de tempo possível configurando o projeto.
- Link: <https://plugins.netbeans.apache.org/catalogue/?id=4>
 - Utilizar a opção Tools → Plugins → Downloaded → Add Plugins.

Criação do Projeto (1)

- Por simplicidade, vamos criar um único projeto para o *backend* e para o *frontend*.
 - Em projetos reais, é comum ter equipes diferentes para cada parte do sistema, e dessa forma, projetos separados para cada parte.
- Neste projeto, vamos utilizar o gerenciador de dependências Maven.
 - Sendo assim, ao criar o novo projeto, vamos escolher as opções:
 - Java with Maven -> Spring boot inicializr project.
 - O projeto será iniciado para a criação de WEB Services automaticamente.

Criação do Projeto (2)

- Informações básicas para o projeto.



New Spring Initializr Project

Steps

1. Choose Project
2. **Base Properties**
3. Dependencies
4. Name and Location

Base Properties

Group:

Artifact:

Version:

Packaging:

Name:

Description:

Package Name:

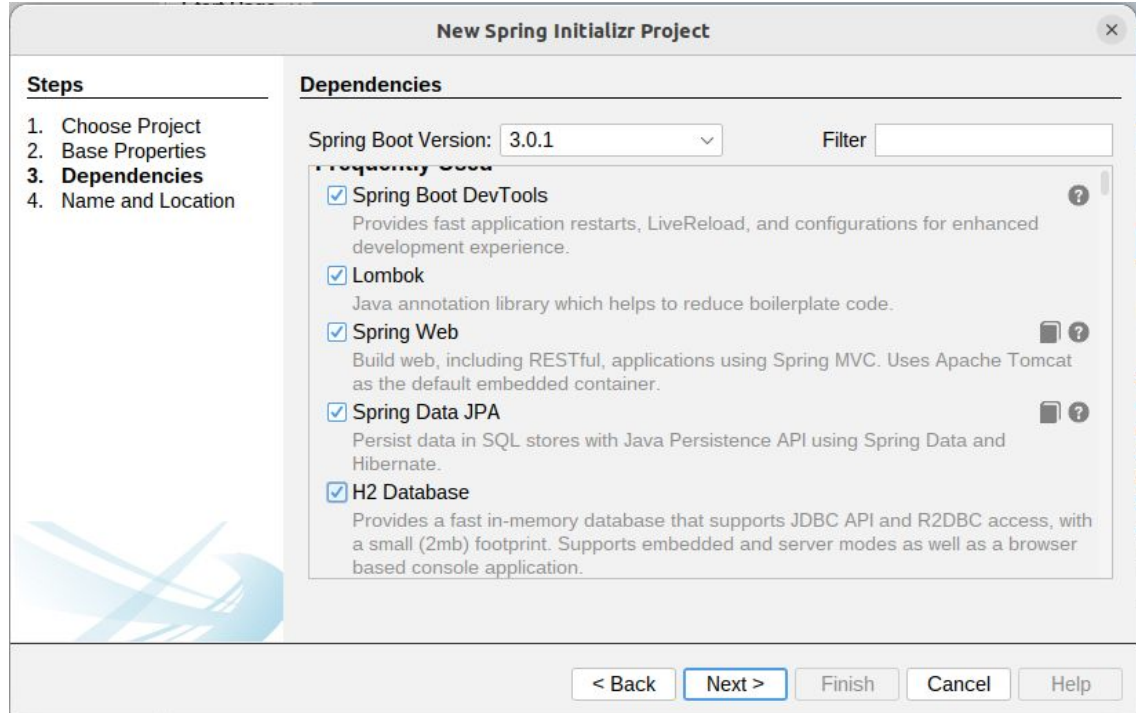
Language:

Java Version:

< Back **Next >** Finish Cancel Help

Seleção de Dependências

- Vamos selecionar as dependências que serão utilizadas neste projeto.
 - Spring Boot DevTools
 - Lombok
 - Spring Web
 - Spring Data JPA
 - H2 Database
- Após isso, vamos nomear o projeto como 'ListaTarefas'.



Desenvolvimento do BackEnd

- Após a criação, o projeto está pronto para a codificação das regras de negócio do sistema.
 - Na primeira execução, todas as dependências serão baixadas e inseridas no projeto automaticamente pelo Maven.
- O projeto já terá pacote *br.edu.uffs.ListaTarefas* criado.
- Vamos criar mais três pacotes para organizar as nossas classes:
 - Controllers: onde ficarão os códigos com a lógica do sistema.
 - Entities: onde serão descritas as entidades do sistema.
 - Repositories: usados para fornecer as principais funcionalidades de acesso, inserção, remoção e listagem de informações do banco de dados.

Entity - Tarefa

- Em nossa aplicação, teremos uma única entidade.
 - Notem o uso de diversas anotações (iniciadas com @).
 - Estas anotações, em sua maior parte, servem para instruir o mecanismo a realizar a geração de código automaticamente.

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Tarefa {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Long id;
    String tarefa;
}
```

- Notem também a não criação de construtores, getters e setters, etc.

Repository - TarefaRepository

- Da mesma forma, teremos um único *repository* para a Tarefa.
 - Este repositório é implementado como uma Interface, extendendo a interface JpaRepository.
 - Notem que, como estamos querendo usar apenas operações padrão, não precisamos inserir nenhum código na interface, e ela herdará todos os métodos diretamente.

```
public interface TarefaRepository extends JpaRepository<Tarefa, Long> {  
  
}
```

TarefaController

- Nesta classe, vamos criar um WEB Service REST.
 - Cada ação será mapeada para uma rota de execução.
 - Estas rotas são controladas automaticamente pelo Spring.
 - Chamamos o *repository* para executar as ações no banco de dados.

```
@RestController
@AllArgsConstructor
public class TarefaController {
    TarefaRepository repository;

    @GetMapping("/tarefa")
    public List<Tarefa> getAllTarefas() {
        return repository.findAll();
    }

    @GetMapping("/tarefa/{id}")
    public Tarefa getTarefaById(@PathVariable Long id) {
        return repository.findById(id).get();
    }

    @PostMapping("/tarefa")
    public Tarefa saveTarefa(@RequestBody Tarefa tarefa) {
        return repository.save(tarefa);
    }

    @DeleteMapping("/tarefa/{id}")
    public void deleteTarefa(@PathVariable Long id) {
        repository.deleteById(id);
    }
}
```

Testando nosso WEB Service

- Nosso WEB Service já está pronto, lendo as informações e salvando-as no banco de dados.
- Podemos testar se tudo está OK através do **Postman**, que é uma ferramenta que permite enviar requisições HTTP remotas e locais.
- Vamos testar os 4 *endpoints* que criamos para nossa aplicação:
 - Get All
 - Save Tarefa
 - Get Tarefa
 - Delete Tarefa

Desenvolvimento do Frontend (1)

- Com o *backend* desenvolvido, vamos desenvolver as telas (*frontend*) para interagir com o WEB Service criado.
- Para isso, vamos criar o pacote *br.edu.uffs.ListaTarefasApp*, que vai conter a aplicação visual desenvolvida.
 - Neste pacote, vamos adicionar um novo *JFrame Form* com o nome *ListaTarefasApp*.

Desenvolvimento do Frontend (2)

- A tela desenvolvida será conforme a imagem:
 - Um painel contendo um campo para cadastro de uma nova tarefa.
 - Um painel contendo uma tabela onde as tarefas cadastradas devem ser exibidas.
- A aplicação deverá comunicar com o WEB Service REST da seguinte forma:
 - Enviar os dados da tarefa via POST para salvar no banco de dados.
 - Comunicar via GET com o serviço para obter as tarefas cadastradas.
 - Enviar o ID de uma tarefa para removê-la.

The image shows a mockup of a web application interface for task management, divided into two main sections:

Adicionar uma Tarefa

This section contains a form for adding a new task. It includes a label "Nova Tarefa" above a text input field with the placeholder text "edtTarefa". Below the input field are two buttons: "btnCadastrar" and "Cadastrar Tarefa".

Listagem de Tarefas

This section displays a list of tasks. It features a button "btnRemover" and a button "Remover Selecionados". Below these buttons is a table with two columns: "ID" and "Tarefa". The table body contains a single row with the text "tblTarefas" in the "Tarefa" column, indicating where the task list would be populated.

Evento do botão 'Cadastrar'

- Criar o objeto.
- Ler o campo.
- Enviar os dados lidos através de uma requisição POST / REST.

```
private void btnCadastrarActionPerformed(java.awt.event.ActionEvent evt) {  
    Tarefa tarefa = new Tarefa();  
  
    if (edtTarefa.getText().isBlank()) {  
        JOptionPane.showMessageDialog(this, "Tarefa não preenchida!");  
  
        return;  
    }  
  
    tarefa.setTarefa(edtTarefa.getText());  
  
    try {  
        RestTemplate restTemplate = new RestTemplate();  
        restTemplate.postForObject(WS_URL, tarefa, Tarefa.class);  
        JOptionPane.showMessageDialog(this, "Tarefa salva com sucesso!");  
    } catch (RestClientException e) {  
        JOptionPane.showMessageDialog(this, e.getMessage());  
    }  
}
```


Preenchimento da tabela de Tarefas

- Obter todas as informações das tarefas via GET.
- Apresentar as informações na tabela.
- Além disso, é preciso chamar essa função em dois locais:
 - Na inicialização da classe.
 - Ao inserir uma nova tarefa.

```
private void loadTableData() {  
    RestTemplate restTemplate = new RestTemplate();  
  
    ResponseEntity<Tarefa[]> response =  
        restTemplate.getForEntity(WS_URL, Tarefa[].class);  
  
    Tarefa[] tarefas = response.getBody();  
  
    DefaultTableModel tableModel =  
        (DefaultTableModel) tblTarefas.getModel();  
  
    tableModel.setNumRows(0);  
  
    for (Tarefa tarefa : tarefas) {  
        Object[] row = { tarefa.getId(), tarefa.getTarefa() };  
        tableModel.addRow(row);  
    }  
}
```

Remoção de tarefas

- Verificar as linhas selecionadas.
- Percorrer o vetor resultante e enviar requisição DELETE com cada ID.
- Recarregar tabela com os dados após a remoção.

```
private void btnRemoverActionPerformed(java.awt.event.ActionEvent evt) {  
    int rows[] = tblTarefas.getSelectedRows();  
  
    if (rows.length == 0) {  
        JOptionPane.showMessageDialog(this, "Nenhuma tarefa selecionada!");  
        return;  
    }  
  
    for (int row : rows) {  
        RestTemplate restTemplate = new RestTemplate();  
        Long id = (Long) tblTarefas.getModel().getValueAt(row, 0);  
        restTemplate.delete(WS_URL + "/" + id);  
    }  
  
    loadTableData();  
    JOptionPane.showMessageDialog(this, "Tarefas removidas com sucesso!");  
}
```

Considerações Finais

- Nesta aula estudamos a forma ‘moderna’ de realizar a integração entre sistemas (frontend e backend).
- Utilizamos REST como o padrão de comunicação.
- Estudamos como fazer persistência de dados usando Spring / JPA e um banco de dados em memória.
 - O backend faz a persistência em banco de dados.
 - O frontend consome uma API, que pode estar rodando localmente ou em nuvem.
- Existem várias outras formas de fazer persistência de dados.
 - Salvar informações em arquivos, utilizar JDBC para salvar um banco de dados local, etc.