Desenvolvimento Web III

Professor Jefferson Chaves
jefferson.chaves@ifpr.edu.br

ATIVIDADE AVALIATIVA

TRABALHO SEMESTRAL

Os objetivos acadêmicos desta atividade são o desenvolvimento da capacidade de projetar e implementar aplicações web utilizando o padrão arquitetural em camadas, promovendo a consolidação de conceitos fundamentais vistos em aula durante o semestre. Além disso, serão utilizados conceitos de persistência de dados, manipulação de informações e controle de acesso. A atividade também procura estimular a aplicação de boas práticas de conceitos de modelagem orientada a objetos, uso de enumerações para controle de estados, e validação de dados em formulários.

Para tanto, neste trabalho, você desenvolverá uma aplicação web para gestão de fitas VHS, com controle de categorias.

DESCRIÇÃO DA ATIVIDADE

Você conheceu uma pessoa saudosa, que ainda tem uma coleção de fitas VHS, cujo controle é feito em uma planilha. Você deve implementar uma aplicação para gestão das fitas VHS dessa pessoa. Sua aplicação deve permitir o gerenciamento das fitas (inserção, deleção, atualização), além de exibir uma listagem com os dados dessas fitas.

Cada VHS possui também uma **categoria**, como: Ação, Comédia, Drama, Terror, Animação, entre outras. O sistema deve permitir o gerenciamento dessas categorias, possibilitando que o usuário as cadastre, edite ou exclua.

Na listagem de fitas, devem ser apresentados:

- Identificador
- Título da fita
- Imagem (opcional)
- Diretor
- Categoria
- Data de cadastro
- Status da fita (disponível, alugada ou indisponível)

São exigidos alguns requisitos mínimos para essa aplicação:

- Deve ser possível modificar o status do VHS.
- Excluir uma fita que não está mais funcionando ou foi perdida.
- A aplicação deve permitir o **gerenciamento de categorias**, incluindo:
  - Cadastro de novas categorias.
  - Edição e exclusão de categorias existentes.
- O **status da fita** deve ser representado por uma **enumeração**, com os estados:
  - DISPONÍVEL
  - EMPRESTADA
  - INDISPONÍVEL
- A aplicação deve possuir controle de usuários:
  - Somente um usuário devidamente autenticado pode acessar o sistema.
- O formulário de cadastro das fitas deve conter **validação básica dos campos**.
- A aplicação deve seguir o padrão arquitetural **em camadas**.
- Os dados devem ser persistidos em uma base de dados relacional utilizando-se ORM.

ENTREGA DA ATIVIDADE

A entrega da atividade deve ser realizada de acordo com os seguintes critérios:

- A atividade pode ser realizada sozinho em um dupla.
- A atividade deve ser entregue utilizando o GitHub. Crie um repositório com o nome: `controle-de-locadora-vhs`.
- Você deve entregar um pequeno relatório (1 ou 2 páginas) junto com o projeto, descrevendo brevemente o processo de desenvolvimento, as tecnologias utilizadas, as funcionalidades implementadas e os resultados obtidos. Você deve incluir informações de como esse projeto pode ser baixado e instalado. Você pode substituir o relatório por um arquivo README.md no
- Envie o link do repositório como resposta a esta atividade.

VAMOS DAR O PRIMEIRO PASSO

Esse trecho do trabalho é apenas um exemplo. Seguir os passos seguintes é opcional.

1 - Configuração do Projeto

Para iniciar o projeto, inicialmente você deve criar um projeto Spring e adicionar as seguintes dependências:

- Spring Web (arquitetura web);
- Thymeleaf (para construção de HTMLS dinâmicos)
- Spring Data JPA (ferramenta de ORM)
- Mysql Driver SQL
- Lombok (Opcional: evita a escrita de códigos getters, setters e etc)
- Spring Boot DevTools (Implanta a aplicação automáticamente)

Quando usamos a dependência `Spring Data JPA`, somos obrigados a configurar a conexão com banco de dados no arquivo `application.properties`. Um modelo pode ser encontrado aqui.

2 - Criação dos Pacotes

Com o projeto criado, você pode criar os pacotes (pastas) que já sabemos que irão ser necessários:

- `entidades` (camada com as classes que representam as coisas do mundo real da aplicação. VHS, Categoria e Usuário são exemplos de entidades);
- `servicos` (camada com as classes que validam das regras de negócio da aplicação. Por exemplo, é possível emprestar uma fita indisponível?);
- `repositorios` (camada com as classe que interagem com o banco de dados);
- `controladores` (camada com as classe que recebem as requisições e devolvem uma resposta)
- outras camadas podem ser necessárias (filtros, úteis e etc);

3 - Implementando a primeira Entidade

Para termos um exemplo, vamos inicialmente criar a classe VHS, que representará a fita de vídeo. A classe pode ser vista na próxima página.

Repare na anotação `@Entity`. É essa anotação que orienta o Spring a criar as tabelas no banco de dados de acordo com a classe. Essa anotação exige um `id` para a tabela, anotado logo em seguida.

Repare também que os atributos `category` e `status` **estão comentados**. Isso ocorre pois, como não temos a classe `Category` nem tampouco a enumeração `TapeStatus` o Spring irá acusar um erro e não executará. Quando essas classes forem criadas, é necessário remover tais comentários.

No pacote `entidades`, implemente a classe `VHS`, conforme modelo abaixo:

```java
@Entity
@Data
public class VHS {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String imageUrl; // Caminho ou URL da imagem (opcional)
    private LocalDate registrationDate;

    //@ManyToOne
    //@JoinColumn(name = "category_id")
    //private Category category;

    //@Enumerated(EnumType.STRING)
    //private TapeStatus status;
}
```

4 - Implementando a primeira classe de serviço

Seguindo a arquitetura em camadas, vamos criar uma classe de serviços para a entidade `VHS`. É comum termos classes de serviço associadas a classes de entidade. Por exemplo, se existe uma classe `VHS`, é provável que tenhamos uma classe `VHSService`. Cabe destacar que isso não é uma regra, mas uma convenção. Podem existir camadas de serviço que não estão associadas a uma entidade específica ou que estejam associadas a mais de uma entidade. Considere uma classe `LoginService`, que realiza operações de `autenticação`, `login` e `logout` de um usuário. Essa classe pode estar associada a uma entidade `Usuário`, a uma classe de `Log`, etc.

No pacote `servicos`, implemente a classe `VHSService`, conforme modelo abaixo:

```
@Service
public class VHSService {

    @Autowired
    VHSRepository vhsRepository;

    public List<VHS> findAll() {
        return vhsRepository.findAll();
    }
}
```

O Spring irá, provavelmente, apontar alguns erros, pois a classe `VHSRepository` ainda não existe. Vamos criá-la no próximo item.

Observe que a classe está anotada com `@Service`. Esse estereótipo indica que essa será uma classe que pode ser gerenciada pelo Spring.

Inicialmente, adicionamos o método `findAll()`. No futuro você irá implementar os outros métodos. Esse método simplesmente faz uma chamada para um método de mesmo nome da classe `VHSRepository`.

5 - Implementando a primeira classe de repositório

Um classe repositório é uma classe cujo objetivo é realizar as operações com o banco de dados. O Spring, por meio da dependência `Spring Data JPA` cria abstrações para acesso às interações mais comuns com a base de dados, tais como inserir, buscar, atualizar e remover registros no banco de dados.

No pacote `repositorios`, implemente a interface `VHSRepository`, conforme modelo abaixo:

```
@Repository
public interface VHSRepository extends JpaRepository<VHS, Long> {}
```

Essa interface estende a interface `JpaRepository` que espera como parâmetros de tipo o tipo da classe e o tipo da chave primária da classe.

Essa interface pode parecer estranha, pois não tem nenhum método. Isso porque os métodos estão definidos pela interface `JpaRepository`. Contudo, você pode implementar as assinaturas para seus métodos personalizados, conforme visto em aula.

Aproveite e verifique a classe `VHSService`. Se ainda houver uma indicação de erro, realize as importações necessárias.

6 - Implementando a primeira classe controladora

A classe Controller funciona como um orquestrador entre a interface do usuário (View) e a lógica de negócio ou dados (Model). Ele é responsável por receber as requisições do usuário, como cliques, envio de formulários ou acesso a URLs, chamar os métodos que processarão os dados e enviar a resposta ao cliente. A resposta pode ser uma página, um arquivo JSON, uma mensagem ou outro tipo de dado.

É por meio de classes desse tipo que “expomos” funcionalidades para os clientes. Essas funcionalidades quando expostas, podem ser acessadas por “rotas”, que nada mais são que endereços web específicos para cada funcionalidades. Essas funcionalidades podem receber parâmetros que podem ser dados vindos da URL ou de formulários.

No pacote `controladores`, implemente a classe `VHSController`, conforme modelo abaixo:

```
@Controller
@RequestMapping("/vhs")
public class VHSController {

    @Autowired
    VHSService vhsService;

    @GetMapping
    public String findAll(Model model){

        List<VHS> vhsList = vhsService.findAll();
        model.addAttribute("vhsList", vhsList);

        return "vhs-list";
    }
}
```

Logo no começo da classe, observe a anotação `@RequestMapping("/vhs")`. Essa anotação indica ao Spring que todas as rotas mapeadas nessa classe, deverão começar o prefixo `"/vhs"`. Em seguida, vamos analisar a linha:

```
@Autowired
VHSService vhsService;
```

A anotação `@Autowired` já apareceu anteriormente. Essa anotação faz parte do sistema de Inversão de Controle (IoC) do Spring e basicamente instrui o Spring a instanciar o objeto `vhsService`, sem a necessidade de usar o comando `new` por exemplo. Deixo como sugestão, realizar uma pesquisa do porque se usar IoC, quais suas vantagens e desvantagens.

A Inversão de Controle (IoC) permite que, ao invés de sua classe criar os objetos que ela precisa, quem faz isso é o framework (como o Spring). Sua classe apenas recebe tudo pronto.
A Injeção de Dependência (DI) é o jeito que o framework usa para te entregar esses objetos. É possível receber os dados pelo construtor, por um método ou até colocar direto dentro de um atributo.

Vamos analisar o método `findAll`:

```
@GetMapping
public String findAll(Model model){

    List<VHS> vhsList = vhsService.findAll();
    model.addAttribute("vhsList", vhsList);

    return "vhs-list";
}
```

Esse método é exposto pela anotação `@GetMapping`. Embora essa anotação não tenha uma rota especificada, ela recebe qualquer requisição do tipo `Get` para a rota `"/vhs"` que endereça a classe. Seria possível e aceitável criar uma rota explícita para essa funcionalidade, tal como `@GetMapping("/all")` mas não foi o caso aqui. O método `findAll` recebe como parâmetro um objeto `model` que permite repassar os dados da controller para um arquivo HTML. Esse objeto é injetado pela IoC do Spring. E por fim, a linha `return "vhs-list"` deve retornar o nome da tela que será carregada. Um exemplo de tela por ser encontrada aqui.

A conclusão do projeto é com você. Use os materiais postados na plataforma, os projetos postados no repositório da disciplina e nos exemplos feitos em aula.