Desenvolvimento Web IV

prof. Felippe Scheidt

Objetivos da disciplina

Seguindo a saga do desenvolvimento web, veremos algumas características de aplicações web no contexto “empresarial/corporativo”.

De que maneira esse tipo de aplicação se diferencia de um sistema web “normal”?

- Escopo e dimensão do projeto
- Necessidade de mecanismos mais robustos de segurança.
- Credenciais de usuário com diferentes níveis de autorização
- Banco de dados distribuídos
- Integração com serviços cloud
- Sistema orientado a serviços

Contexto da aplicação

Considerando este fatores, dimensão e natureza do projeto:

Adotar uma arquitetura de software.

Complexidade dos requisitos => Uso de frameworks.

Delegar atividades burocráticas para o framework.

Abstração da camada de persistência.

Automação de testes.

Fácil disponibilidade dos dados através de uma API REST

Integrando as diferentes camadas da aplicação

Camada de persistência (Model)

Interface com usuário (View) (bootstrap, vue.js, react, ...)

Controle de fluxo de execução e roteamento de requisições (Controller)

Exposição de dados para outros módulos do sistema.

Consulta de dados de outros serviços.

Framework == integrar os diferentes níveis e camadas de execução da aplicação.

Por que usar frameworks?

Framework: força o código da aplicação a seguir determinada estrutura estabelecendo uma padronização nas funcionalidades.

Isso é bom? Em geral **sim**, mas pode virar um pesadelo em alguns casos...

Frameworks trabalham com diversos padrões de projetos, destacando-se:

- Injeção de dependência.
- Inversão de controle.

Tecnologias

Qual tecnologia adotar?

Até certo ponto, uma questão de know-how da equipe de desenvolvedores.

Mas o principal fator é analisar os requisitos da aplicação antes de tomar uma decisão.

Nesta disciplina usaremos o Spring framework/Spring boot, pois é uma das soluções Java mais completas e open-source.

https://github.com/spring-projects/spring-framework

https://github.com/spring-projects/spring-boot

Faça uma análise das opções

Escolher o framework mais adequado envolve por exemplo:

Analisar o *expertise* da equipe de desenvolvedores.

A **maturidade** do framework: tempo de mercado.

O tamanho da comunidade de desenvolvedores.

A linguagem de programação.

Multiplataforma.

Recursos da linguagem Java

Algums dos recursos da linguagem Java frequentemente usados:

- ***Annotations***
- Interfaces
- Classes
- API collections
- ***Lambda expression***

Nomeação de pacotes

A escolha do nome do pacote deve seguir a convenção de nomeação de pacotes do java:

Usa o nome *reverso* do domínio da aplicação.

Exemplo:

https://reitoria.ifpr.edu.br/

package br.edu.ifpr.reitoria;

Comparable

Problema: como definir a ordem ou a posição de um objeto em uma lista?

Para tipos primitivos a noção de ordem é trivial, mas para objetos que possuem sua própria definição (classe) precisamos especificar quais os critérios de ordenação.

A interface **Comparable** permite definir isso através da implementação do método `compareTo()`

Assim um objeto consegue comparar a si mesmo com outro objeto.

Annotations

Fornecem informações (*metadados*) sobre o código-fonte.

Isso é útil por exemplo para informar ao compilar que certo código deve ser tratado dentro de um contexto específico (definido na annotation).

Pode ser usado em atributos, métodos, classes, interfaces.

```
@Entity
public class Cliente implements Comparable<Cliente> {
    @Id
    private Integer id;
    @Override
    public int compareTo(Cliente other) {
        return this.id.compareTo(other.id);
```

Implementando o método compareTo

Cada instância da classe Cliente tem a capacidade de se comparar com outra instância da classe cliente:

```
public class Cliente implements Comparable<Cliente> {
    @Override
    public int compareTo(Cliente outro) {
        return this.id.compareTo(outro.id);
        /**
         * compareTo() retorna um numero:
         * >= 1 (objeto 'this' eh maior que objeto 'outro')
         * == 0 (os dois objeto sao 'iguais' => mesma ordem)
         * <= -1 (objeto 'outro' eh maior que 'this')
         */
    }
```

Lambda Expression

Adiciona características da programação funcional ao java.

Função: quando invocada produz o mesmo resultado se passado os mesmos parâmetros.

Não pode depender de valores mutáveis.

Uma função *não deve causar efeitos colaterais externos*: altera uma variável, faz update no banco de dados e envia email.

Lambda Expression

Uma lambda é uma função.

Uma função é uma tarefa computacional que recebe parametros e retorna um valor.

Até o java 8, funções eram somente implementadas através de métodos.

✱ O lambda permite que *funções sejam passadas como **parâmetros***. Essa é uma grande diferença em relação aos métodos tradicionais do java.

Exemplo: lambda expression

Existem diferentes maneiras para iterar uma coleção de objetos no java, por exemplo:

- Iteração externa (old school)
- Iteração externa
- Iteração interna (lambda expression)

Classes

Classes internas (aninhadas)

Classes anônimas

Classes estática