

# DESENVOLVIMENTO DE SOFTWARE PARA A WEB 1

## Módulo 6 SpringMVC + Thymeleaf

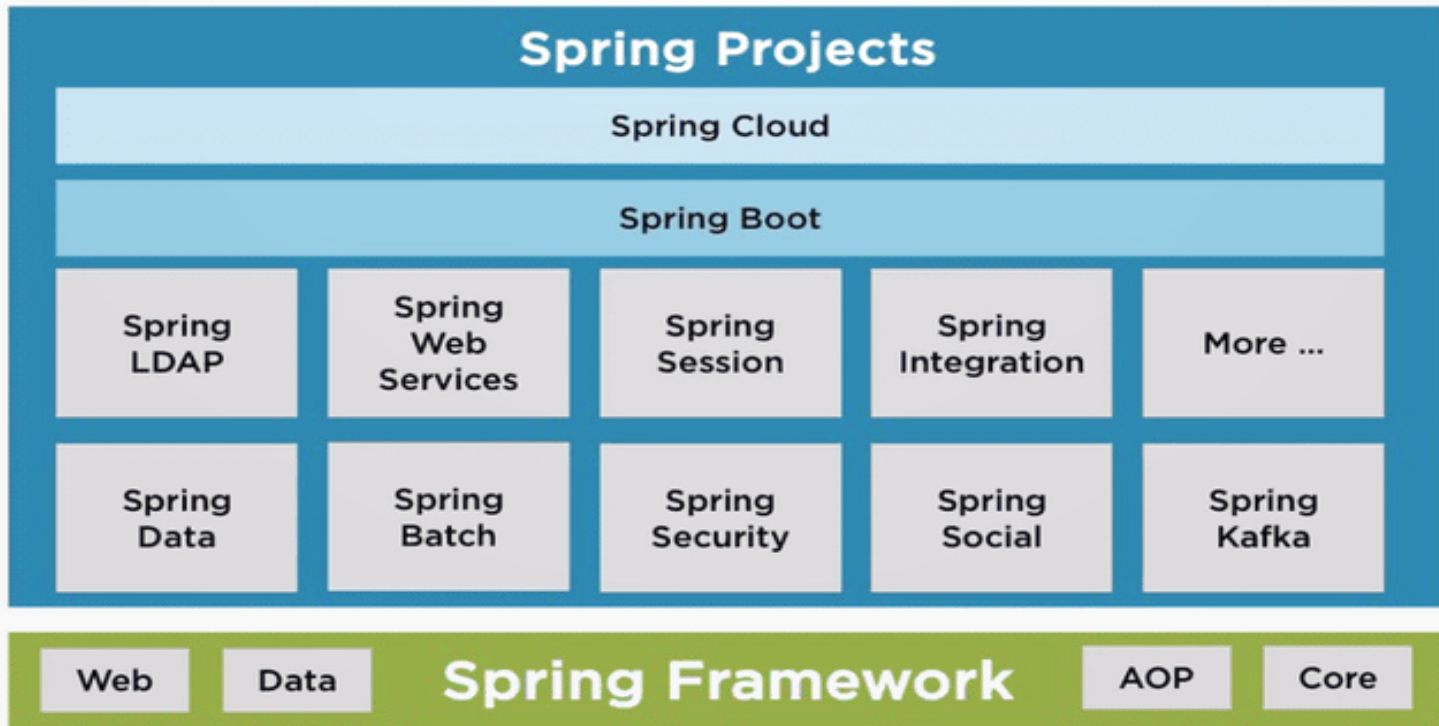
**Delano Medeiros Beder**  
**[delano@dc.ufscar.br](mailto:delano@dc.ufscar.br)**

# Spring Framework


O Spring não é um framework apenas, mas um conjunto de projetos que resolvem várias situações do cotidiano de um programador, ajudando a criar aplicações Java com simplicidade e flexibilidade.


Existem muitas áreas cobertas pelo ecossistema Spring, como **Spring Data JPA** para acesso a banco de dados, **Spring Security** para prover segurança, e diversos outros projetos que vão de *cloud computing* até *big data*.


# Spring Framework



# Spring Boot (inicializr)







**Project**

- ☒ Maven Project
- ☐ Gradle Project

**Language**

- ☒ Java ☐ Kotlin
- ☐ Groovy

**Spring Boot**

- ☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M2) ☐ 2.3.4 (SNAPSHOT)
- ☒ 2.3.3 ☐ 2.2.10 (SNAPSHOT) ☐ 2.2.9
- ☐ 2.1.17 (SNAPSHOT) ☐ 2.1.16

**Project Metadata**

Group

Artifact

**Dependencies** ADD DEPENDENCIES... CTRL + B

No dependency selected

GENERATE CTRL + G EXPLORE CTRL + SPACE SHARE...

O Spring Boot é um projeto que chegou para facilitar o processo de configuração e publicação de nossas aplicações.

A intenção é ter o seu projeto rodando o mais rápido possível e sem complicação.

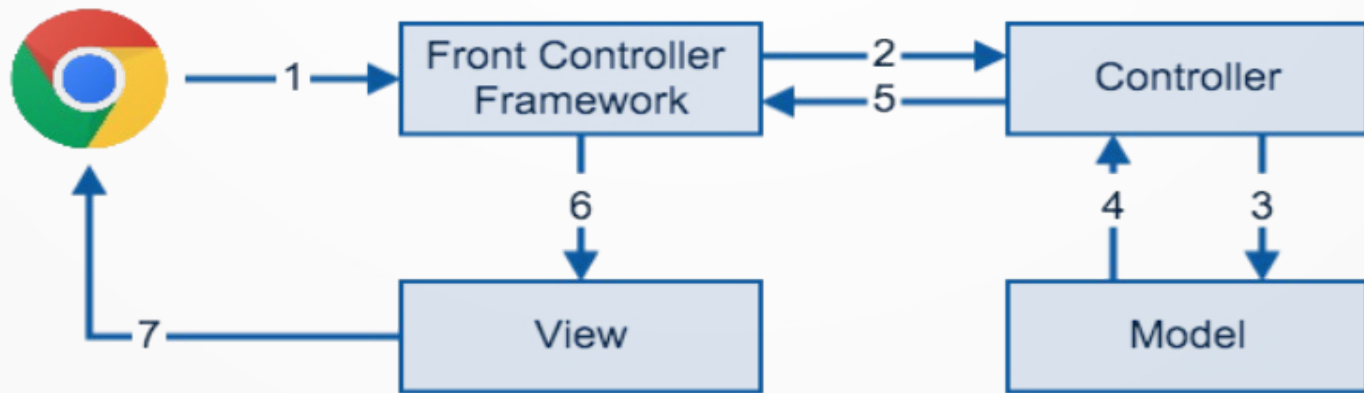
Ele consegue isso favorecendo a convenção sobre a configuração.

Com os módulos informados o Spring-Boot vai reconhecê-los e fornecer uma configuração inicial.

Basta que você diga pra ele quais módulos deseja utilizar:  
WEB, Template, Persistência, Segurança, etc.

# Spring MVC

- Dentre os projetos **Spring**, o **Spring MVC** é o framework que te ajuda no desenvolvimento de aplicações web robustas, flexíveis e com uma clara separação de responsabilidades nos papéis do tratamento da requisição.



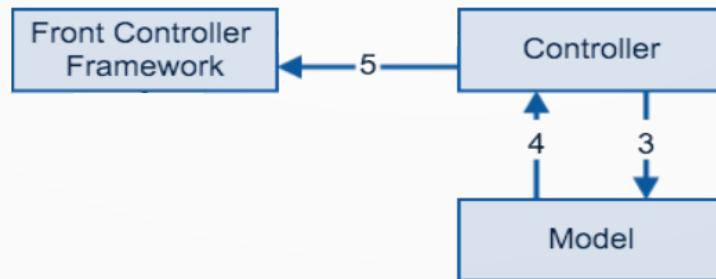
# Spring MVC

1. Acessamos uma URL no navegador que envia a requisição HTTP para o servidor que roda a aplicação web desenvolvido utilizando o **Spring MVC**. Perceba que quem recebe a requisição é o controlador (**Front Controller**) do **Spring MVC**.
2. O controlador do **Spring MVC**, irá procurar qual classe é responsável por tratar essa requisição, entregando a ela os dados enviados pelo navegador. Essa classe faz o papel do **Controlador**.



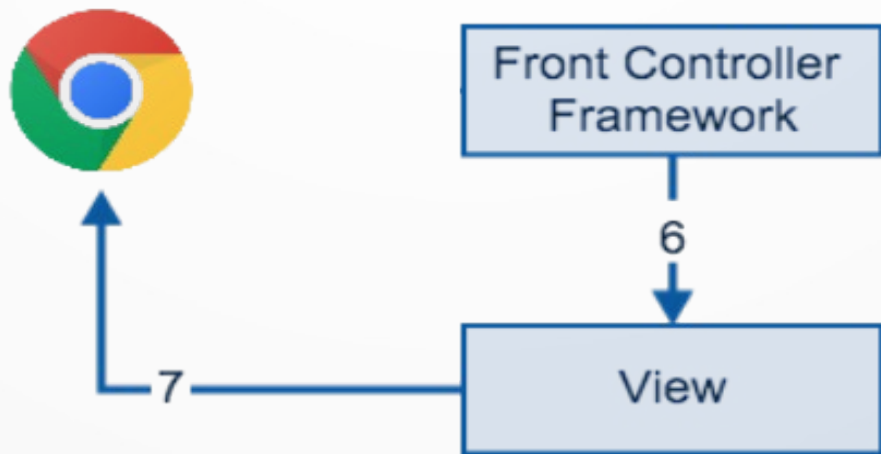
# Spring MVC

3. O **Controlador** passa os dados para o **Modelo**, que por sua vez executa todas as regras de negócio, como cálculos, validações e acesso ao banco de dados.
4. O resultado das operações realizadas pelo **Modelo** é retornado ao **Controlador**.
5. O **Controlador** retorna o nome da **Visão**, junto com os dados que a visão precisa para renderizar a página.



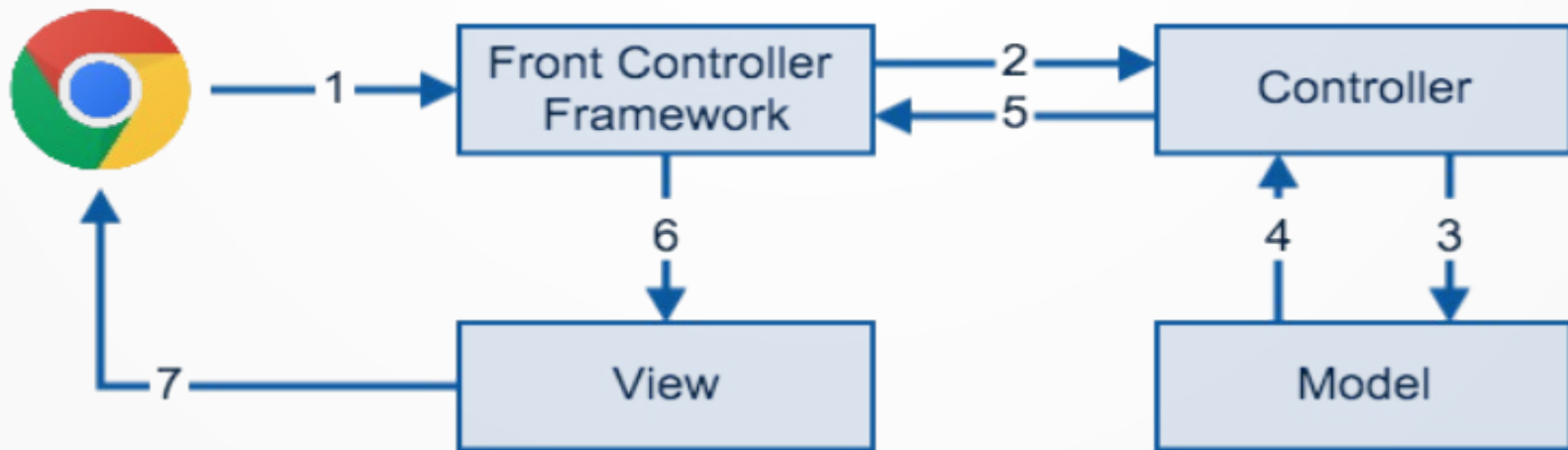
# Spring MVC

6. O **Spring MVC** “encontra” a visão que processa os dados, transformando o resultado em um HTML.
7. Finalmente, o HTML é retornado ao navegador do usuário.





# Spring MVC



# Controlador MVC

## @Controller

Transforma a classe em um bean do tipo *controller* do MVC

```
package br.ufscar.dc.dsw.controller;

import java.text.SimpleDateFormat;

@Controller
public class AloMundoController {

    @GetMapping("/")
    public String index(Model model) {

        SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss - dd MMMM yyyy");

        Calendar cal = Calendar.getInstance();
        model.addAttribute("date", dateFormat.format(cal.getTime()));

        return "index";
    }
}
```

**Dados Modelo**

**@GetMapping**  
Requisições HTTP GET

**Visão**

@GetMapping shortcut para @RequestMapping(method = RequestMethod.GET)

# Thymeleaf

- O Thymeleaf não é um projeto Spring, mas uma biblioteca que foi criada para facilitar a criação da camada de visão com uma forte integração com o Spring, e uma boa alternativa ao JSP.
- O principal objetivo do Thymeleaf é prover uma forma elegante e bem formatada para criarmos nossas páginas. O dialeto do Thymeleaf é bem poderoso como você verá no desenvolvimento da disciplina.

# Thymeleaf (Exemplo Básico)

HomeController.java

```
@RequestMapping("/")
public String home (Model model) {
    model.addAttribute("firstName", "Fulano");
    model.addAttribute("lastName", "Silva");
    return "home";
}
```

templates/home.html

```
<span th:text="'${'Olá'+ firstName + ' ' + lastName}'">Oi</span>
```

Com renderização

Olá Fulano Silva

Sem renderização

Oi

# SpringMVC + Thymeleaf

## Demonstração 1

Alô Mundo MVC

Link YouTube: <https://youtu.be/kKX-8-5mxQc>

# Thymeleaf (Exemplo 18n)

templates/home.html

```
<span th:text="${#{home.hi} + firstName + ' ' + lastName}">Oi</span>
```

messages\_pt.properties

home.hi = Olá

messages\_en.properties

home.hi = Hi

Renderização (pt)

Olá Fulano Silva

Renderização (en)

Hi Fulano Silva

# SpringMVC + Thymeleaf

## Demonstração 2

I18n (Idioma do Navegador)

Link YouTube: [https://youtu.be/OcTU\\_aY7E2g](https://youtu.be/OcTU_aY7E2g)

# SpringMVC + Thymeleaf

## Demonstração 3

I18n (Menu de opções)

Link YouTube: <https://youtu.be/LUBsfvQCDvw>



# Built-in Utilities

<https://www.baeldung.com/spring-thymeleaf-3-expressions>

- `#dates`
  - `#dates.format(date)`, `#dates.day(date)`
- `#numbers`
  - `#numbers.formatInteger(num,3)`, `#numbers.sequence(from, to, step)`
- `#strings`
  - `#strings.isEmpty(str)`, `#strings.contains(str,"hi")`
- `#arrays`
  - `#arrays.length(arr)`, `#arrays.isEmpty(arr)`
- `#lists`
  - `#lists.size(list)`, `#lists.isEmpty(list)`
- `#sets`
  - `#sets.contains(set, element)`, `#sets.isEmpty(set)`

# Condicionais (Operador Elvis)

- O operador Elvis ?: Permite-nos renderizar o texto dentro de um elemento HTML, dependendo do estado atual de uma variável.

```
<td th:text="${teacher.active} ? 'ACTIVE' : 'RETIRED'" />
```

<https://www.baeldung.com/spring-thymeleaf-conditionals>

# Condicionais (th:if & th:unless)

- Os atributos **th:if** e **th:unless** nos permitem renderizar um elemento HTML dependendo de uma condição fornecida

```
<td>  
    <span th:if="${teacher.gender == 'F'}">Female</span>  
    <span th:unless="${teacher.gender == 'F'}">Male</span>  
</td>
```

<https://www.baeldung.com/spring-thymeleaf-conditionals>

# Condicionais (th:switch & th:case)

- Se houver mais de dois resultados possíveis de uma expressão, podemos usar os atributos **th:switch** e **th:case** para a renderização condicional dos elementos HTML

```
<td th:switch="${#lists.size(teacher.courses)}">
  <span th:case="'0'">NO COURSES YET!</span>
  <span th:case="'1'" th:text="${teacher.courses[0]}"></span>
  <div th:case="*">
    <div th:each="course:${teacher.courses}" th:text="${course}"/>
  </div>
</td>
```

O asterisco (\*) é usado para opção *default*

<https://www.baeldung.com/spring-thymeleaf-conditionals>

# Iterações (th:each)

- No Thymeleaf, a iteração é obtida usando o atributo **th:each**.
- Esse atributo itera sobre alguns tipos de dados diferentes, como: objetos que implementam `java.util.Iterable`, objetos que implementam `java.util.Map` e *arrays*.

```
<tr th:each="student: ${students}">
    <td th:text="${student.id}" />
    <td th:text="${student.name}" />
</tr>
```

<https://www.baeldung.com/spring-thymeleaf-iteration>

# Iterações (th:each)

- O Thymeleaf também permite um mecanismo útil para controlar o processo de iteração por meio de uma variável de *status*.
- A variável de status fornece as seguintes propriedades: (i) **index**: o índice de iteração atual, começando com 0 (zero); (ii) **count**: o número de elementos processados até agora; (iii) **size**: o número total de elementos na lista; (iv) **even/odd**: verifica se o índice de iteração atual é par ou ímpar; (v) **first**: verifica se a iteração atual é a primeira; e (vi) **last**: verifica se a iteração atual é a última

```
<tr
  th:each="student, iStat : ${students}"
  th:style="${iStat.odd}? 'font-weight: bold;'"
  th:alt-title="${iStat.even}? 'even' : 'odd'">
  <td th:text="${student.id}" />
  <td th:text="${student.name}" />
</tr>
```

**FIM**