**Aula XXV - Desenvolvimento Web III**

**SpringWeb - MVC**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fontes:

<https://spring.io/projects/spring-framework>

<https://www.tutorialspoint.com/spring_boot/index.htm>

<https://medium.com/@mvalho/spring-boot-como-criar-uma-aplica%C3%A7%C3%A3o-do-zero-parte-1-2c3098c0e900>

<http://www.dclick.com.br/2010/09/15/o-que-e-o-maven-e-seus-primeiros-passos-com-a-ferramenta/>

<https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/>

<https://becode.com.br/o-que-e-api-rest-e-restful/>

<https://www.w3schools.com/xml/>

<https://netbeans.org/kb/71/web/quickstart-webapps-spring_pt_BR.html#setting>

<https://docs.spring.io/spring/>

<https://docs.spring.io/spring/docs/3.0.x/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Contexto**

O **Spring Boot** é uma estrutura baseada em Java de código aberto usada para criar um Micro Service. **É desenvolvido pela Equipe Pivotal.** É fácil criar aplicativos de Spring autônomos e prontos para produção usando o Spring Boot. O Spring Boot contém um suporte abrangente à infraestrutura para o desenvolvimento de um micro serviço e permite que você desenvolva aplicativos prontos para a empresa que você pode "simplesmente executar" .

**Pequeno Glossário para contextualização:**

**Maven:** Apache Maven, ou Maven, é uma ferramenta de automação de compilação utilizada primariamente em projetos Java. Para saber mais, leia sobre, neste link: <http://www.dclick.com.br/2010/09/15/o-que-e-o-maven-e-seus-primeiros-passos-com-a-ferramenta/>

**Gradle:** Gradle é um sistema de automação de compilação open source que se baseia nos conceitos de **Apache Ant** e **Apache Maven** e introduz uma linguagem de domínio específico baseada em Groovy em vez do XML usado pelo Apache Maven para declarar a configuração do projeto. Para saber mais, leia sobre, neste link: <https://medium.com/collabcode/come%C3%A7ando-com-o-gradle-4e96fd12d933>

**API:** A sigla API corresponde às palavras em inglês **“Application Programming Interface“**. No português **“Interface de Programação de Aplicações”**. Elas são uma forma de integrar sistemas, possibilitando benefícios como a segurança dos dados, facilidade no intercâmbio entre informações com diferentes linguagens de programação e a monetização de acessos. Para saber mais, leia neste link: <https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/>

**REST:** **Representational State Transfer**, em português Transferência Representacional de Estado, é um estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de web services.

**Web RESTful:** A diferença é apenas gramatical. Em outras palavras, sistemas que utilizam os princípios **REST são chamados de RESTful**. **REST: conjunto de princípios de arquitetura.** **RESTful: capacidade de determinado sistema aplicar os princípios de REST.** Para saber mais, você pode ler neste link: <https://becode.com.br/o-que-e-api-rest-e-restful/>

**XML:** XML, do inglês eXtensible Markup Language, é uma linguagem de marcação recomendada pela W3C para a criação de documentos com dados organizados hierarquicamente, tais como textos, banco de dados ou desenhos vetoriais. A linguagem XML é classificada como extensível porque permite definir os elementos de marcação. Para saber mais, leia neste link: <https://www.w3schools.com/xml/>

**Micro Service:** É uma arquitetura que permite aos desenvolvedores desenvolver e implantar serviços de forma independente. Cada serviço em execução tem seu próprio processo e isso atinge o modelo leve para suportar aplicativos de negócios.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SpringWeb - MVC**

O Spring Boot verifica o classpath do projeto buscando por algumas bibliotecas, dependendo da biblioteca que ele encontrar ele aplica um configuração padrão para que possamos desenvolver a nossa aplicação.Por exemplo, se ele encontra as bibliotecas do Spring JPA e de algum banco de dados ele já sobe aquelas configurações padrões que todos nós vemos nas documentações oficiais como: nome de usuário do banco, endereço, porta, senha, driver e tudo mais. Se ele encontra as biblioteca do Spring Web: todas as configurações, locais das páginas web, container, porta.

Você pode escolher o Spring Boot por causa dos recursos e benefícios que ele oferece, conforme os tópicos abaixo:

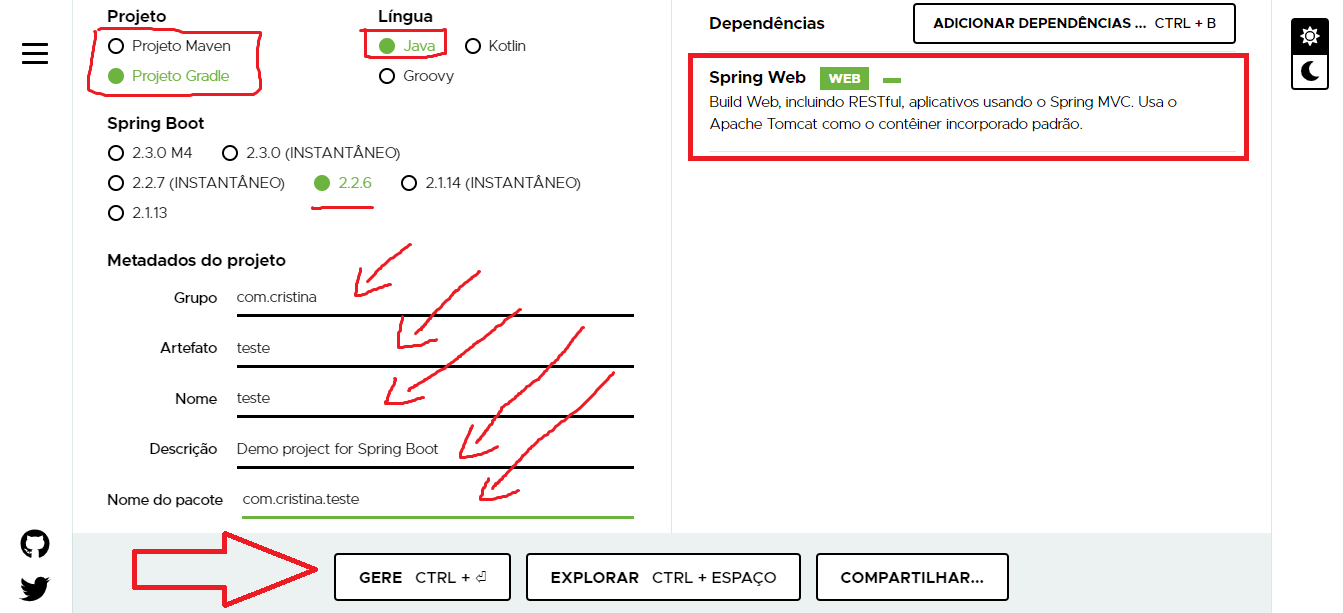
* **Ele fornece uma maneira flexível de configurar Java Beans, configurações XML e transações de banco de dados.**
* **Ele fornece um poderoso processamento em lote e gerencia os pontos de extremidade REST.**
* **No Spring Boot, tudo é configurado automaticamente; nenhuma configuração manual é necessária.**
* **Oferece aplicação de mola baseada em anotações**
* **Facilita o gerenciamento de dependências**
* **Inclui contêiner de servlet incorporado.**

**Passos para execução:**

Vamos acessar o inicializar do Spring, por intermédio deste endereço. <https://start.spring.io/> A interface é bem simples e acho que não é necessário muita explicação. Existe a versão mais avançada mas é desnecessário para esse início e veremos mais adiante.



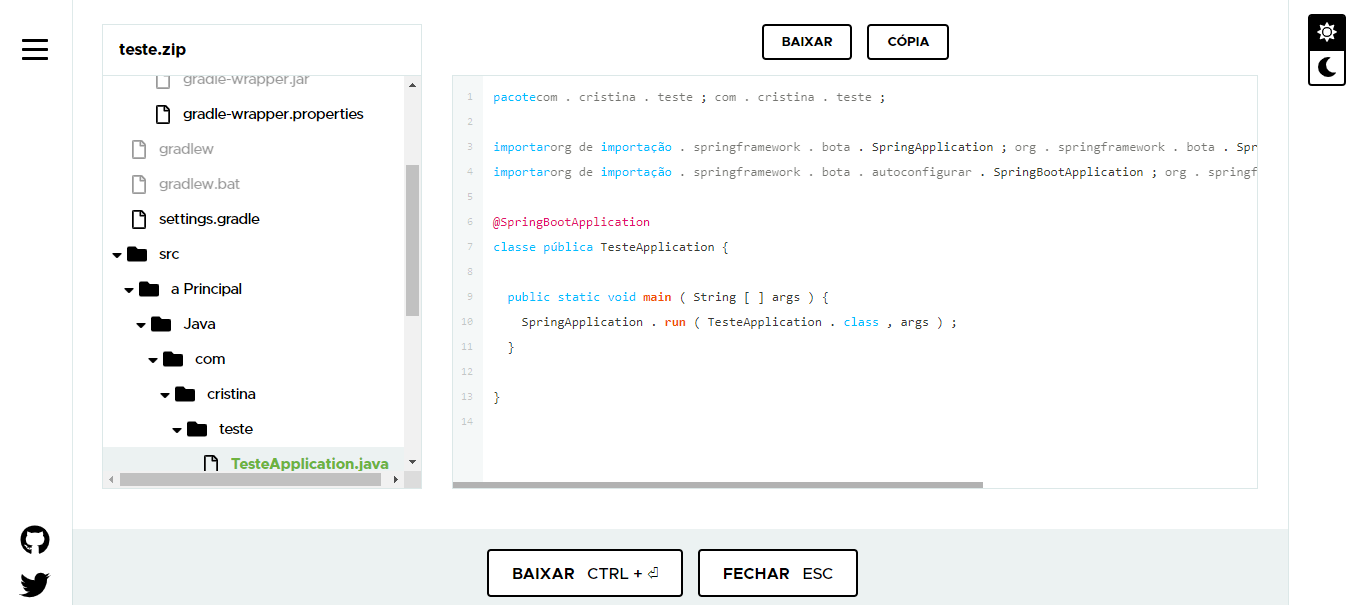
Vamos fazer esta configuração:



e clicar neste botão! **Explorar**



Certo, vamos aos pontos importantes dessa estrutura. Vista do ponto de tudo ter sido criado pelo **Spring Boot Initializr**. Foi automaticamente criado uma main class que será a classe usada para subir toda a aplicação com o **Spring Boot**, ela se chama **CristinaApplication.java** e é anotada com **@SpringBootApplication** essa anotação diz ao **Spring Boot** que essa classe deverá ser usada como base para configurar toda nossa aplicação ela seria uma junção de algumas anotações usadas pelo **Spring** para configurar uma aplicação. Uma **test class** para nossa **main clas**s, que por convenção leva o nome da **classe+Test** e o arquivo **application.properties**. Esse arquivo de propriedades é bem importante para aplicação, pois é nele que colocaremos alguma configurações. Esse arquivo também pode ser substituído por um arquivo YAML, **o Spring Boot vai reconhecê-lo caso você mude a extensão de .properties para .yml.** As duas classes de **test e principal foram criadas dentro de uma estrutura de pastas, que é comumente usada por quem desenvolve em Java. Então aquela tarefa de sempre criar a estrutura básica de pastas, já é feita pelo Spring Initializr.**

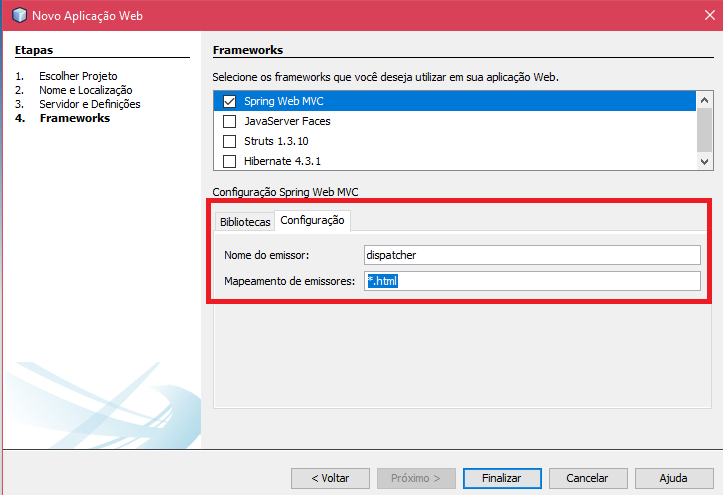


Estes são os processos para trabalhar com ele pelo inicializador e pode ser usado pelo IntelliJ

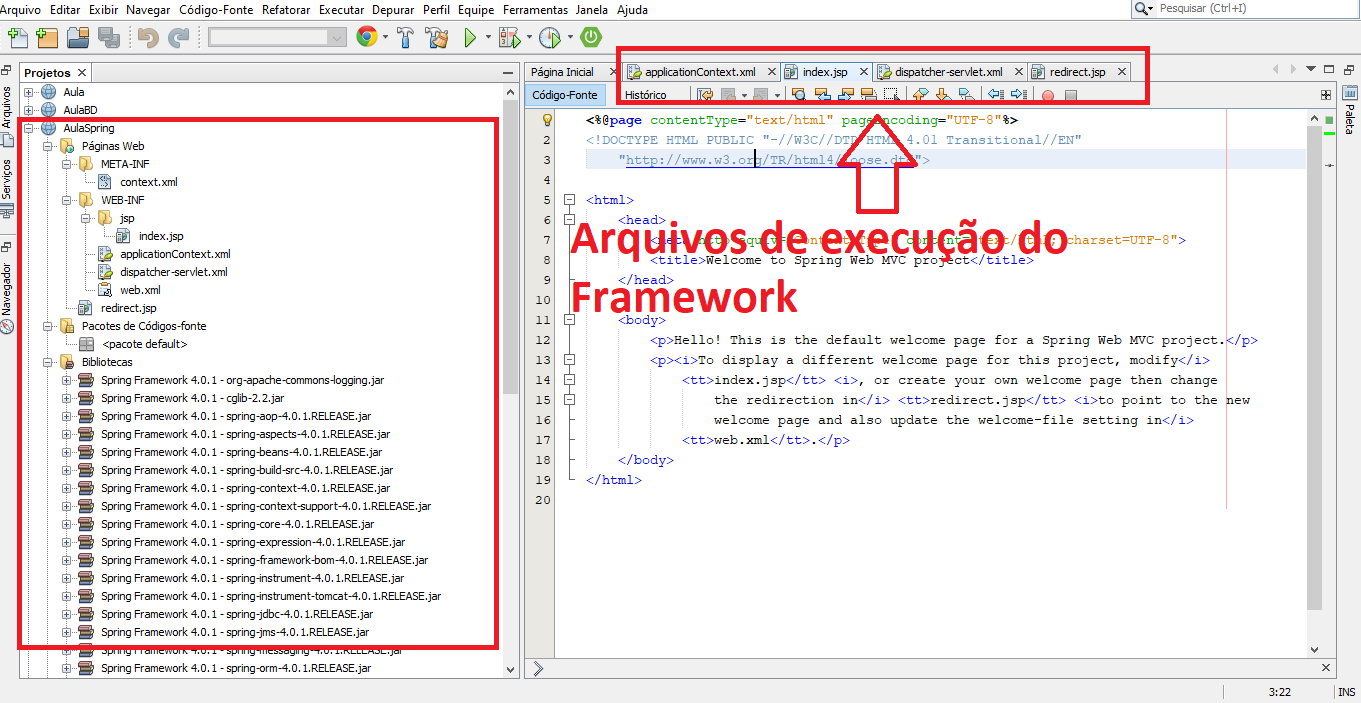
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**No NetBeans** ou outra IDE, vamos fazer as configurações que já fizemos em aula anterior, você pode rever, clicando neste [**Link da Aula XXIII**](https://docs.google.com/document/d/1RwWZjDmC8iiOLUiTWBqvkxNC9Qq0SGKlZhgkVGFvUmk/edit?usp=sharing)**.**

Este projeto que abriremos agora, vai se chamar **AulaSpring** e, a única alteração que faremos na configuração do Framework será a alteração da extensão do arquivo HTML para os padrões de **DOCTYPE do HTML 5**. Então, vamos clicar na aba **Configuração** e observe que o assistente permite especificar o nome e o **mapeamento do servlet Spring Dispatcher,** vamos alterar a extensão de **htm para html.**



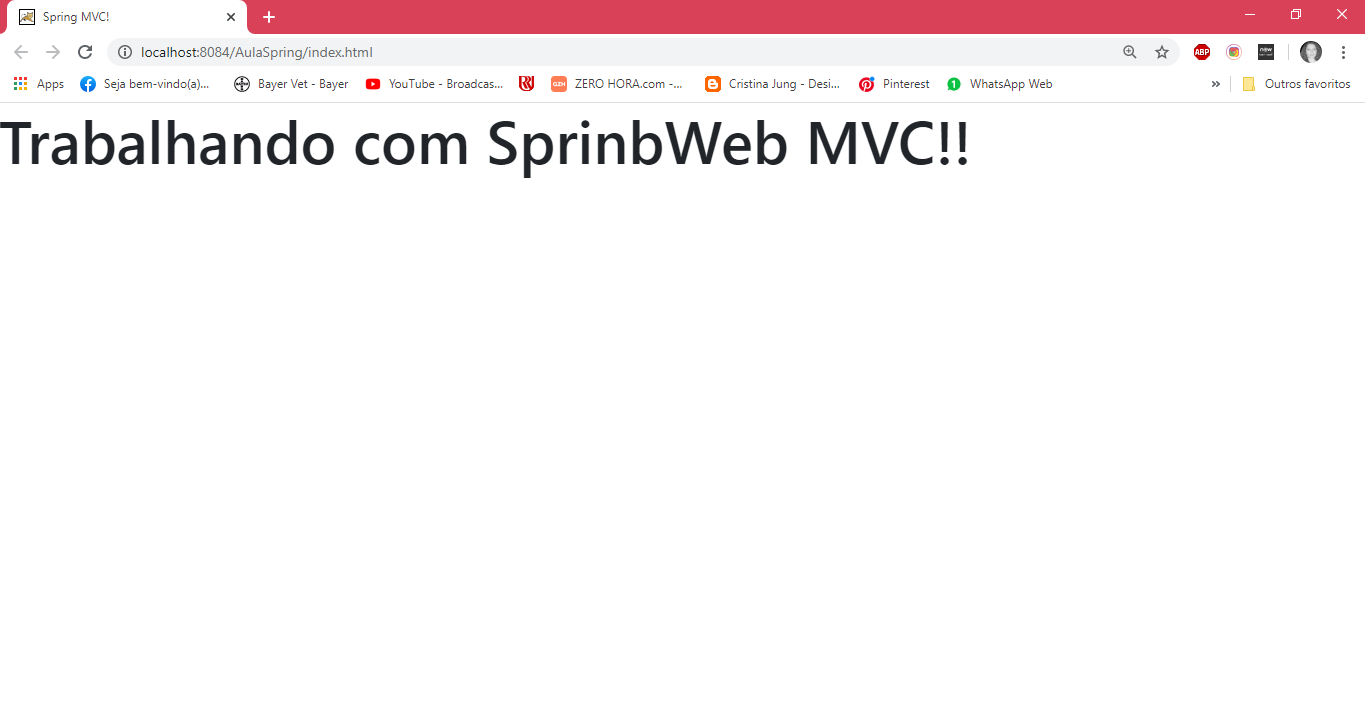
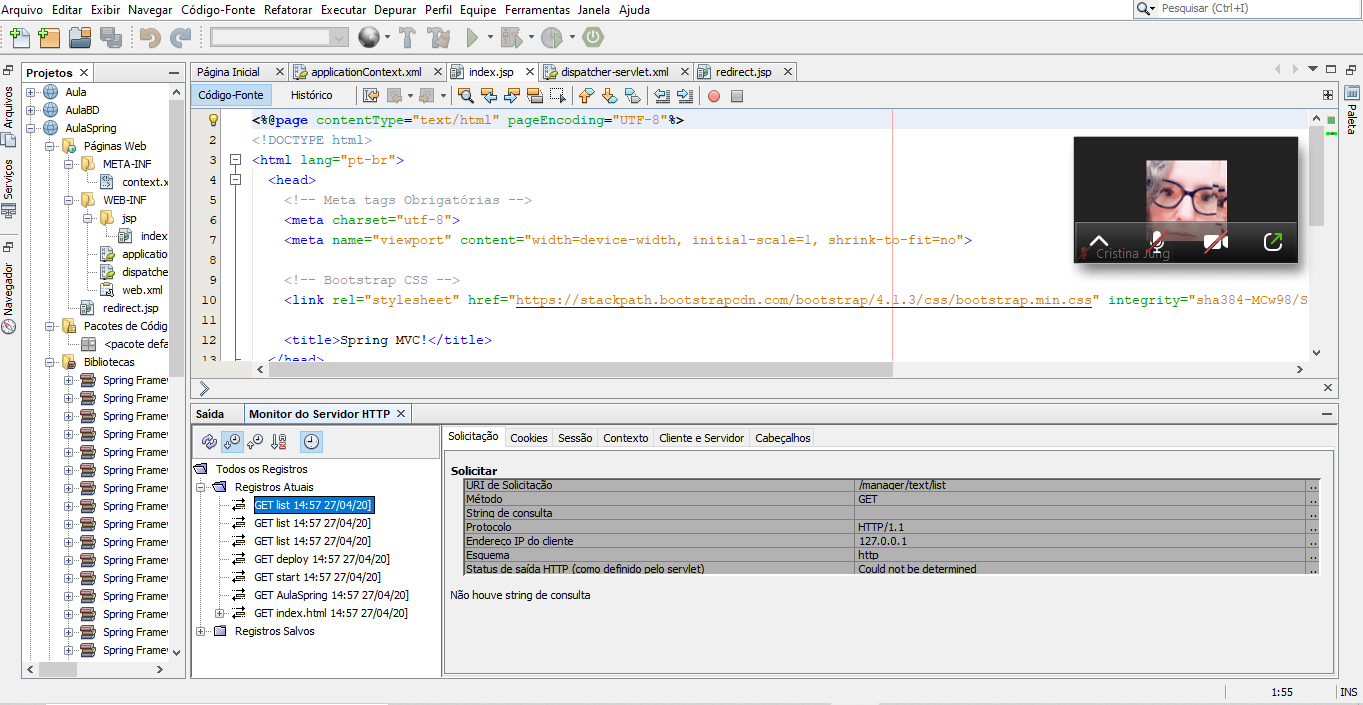
Pressione Finalizar. A IDE irá cria um projeto para o aplicativo inteiro, **incluindo todos os metadados, assim como o script de construção do projeto Ant que você pode inspecionar na janela Arquivos**. Você pode ver a estrutura do modelo na janela Projetos. Importante verificae que também são abertos 4 arquivos por padrão no editor do IDE: **dispatcher-servlet.xml, applicationContext.xml, redirect.jsp e index.jsp.**



No arquivo index.jsp, iremos alterar o CDN para a versão do Bootstrap.

|  |
| --- |
| 1. **Acesse o Link:** [**https://getbootstrap.com.br/**](https://getbootstrap.com.br/) 2. **Clique no Menu Documentação;** 3. **Vá até a opção Template Inicial, clique sobre o link Copiar** 4. **Substitua o DOCTYPE do arquivo pelo CDN do Bootstrap. Ficará desta forma:** |

Próximo passo para que possamos ver se está configurado e executando direito, será clicar sobre o botão Executar (com a seta verde), veremos o TomCat ou o GlassFish subindo. Todos os Status serão exibidos na janela de Output da IDE.

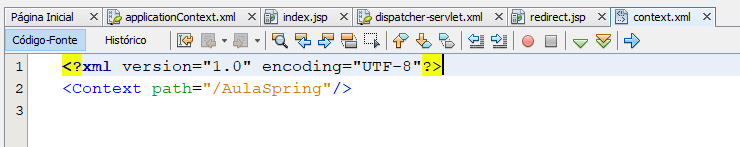
Ótimo, se apareceu a saída, seja com TomCat ou GalssFish, toda a estrutura está funcionando corretamente. 

***Quando você executa o projeto no IDE, o projeto é compilado e implantado no servidor e, em seguida, aberto no navegador padrão. Além disso, o IDE fornece um recurso Implantar ao salvar, que está ativado por padrão para projetos da Web. Quando você salva arquivos no editor, o projeto é automaticamente recompilado e implantado no servidor. Para visualizar alterações, você pode, simplesmente, atualizar as páginas no navegador.*** *Fonte:* [*https://netbeans.org/kb/71/web/quickstart-webapps-spring\_pt\_BR.html*](https://netbeans.org/kb/71/web/quickstart-webapps-spring_pt_BR.html)

**Vamos ver agora o que cada arquivo da estrutura do SpringWeb MVC é responsável:**

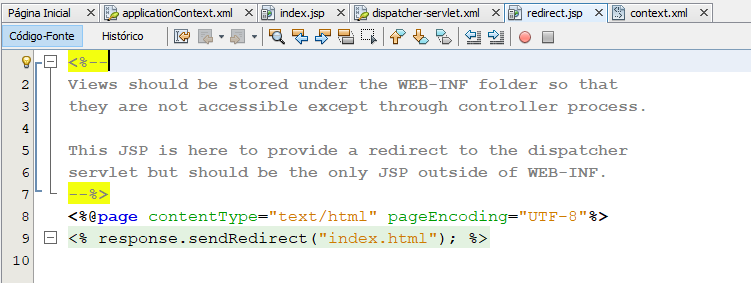
**Vamos começar pelo context.xml:**

Arquivo em XML que exibe o caminho do nosso projeto.



**redirect.jsp:**

O ponto de entrada padrão do aplicativo é **redirect.jsp**. Em redirect.jsp, existe uma instrução de redirecionamento que aponta todas as solicitações para **index.html**:



**dispatcher-servlet.xml:**

Aqui os arquivos HTML são setados (lembrem-se que estou me referindo a HTML porque nas configurações do nosso Spring nós definimos que a extensão padrão de todos estes arquivos serão HTML). O nome completamente qualificado do servlet dispatcher, conforme mostrado acima, é org.springframework.web.servlet.DispatcherServlet. Essa classe é contida na biblioteca do Spring, que foi adicionada ao classpath do projeto quando o projeto foi criado. Você pode verificar isso na janela Projetos, fazendo uma busca detalhada a partir do nó Bibliotecas. Localize o arquivo spring-webmvc-3.0.0.RELEASE.jar e, em seguida, expanda para encontrar org.springframework.web.servlet > DispatcherServlet.

***Fonte: Docs do Spring --*** [***https://docs.spring.io/spring/***](https://docs.spring.io/spring/)

**A classe DispatcherServlet manipula as solicitações de entrada com base nas definições de configuração encontradas em dispatcher-servlet.xml.**

Observe o código abaixo:

|  |
| --- |
| **<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">** |

**Três beans são definidos neste arquivo: indexController, viewResolver e urlMapping.** Quando a DispatcherServlet recebe uma solicitação que coincide com \*.htm tal como index.html, ela procura por um controle no urlMapping que possa acomodar a solicitação. **Acima, você vê que existe uma propriedade mappings que vincula /index.htm a indexController.**

O ambiente de tempo de execução procura a definição de um bean chamado indexController, que é convenientemente fornecido pelo projeto de esqueleto. Observe que indexController estende **ParameterizableViewController**. Essa é uma outra classe fornecida pelo Spring, que simplesmente retorna uma visualização. Acima, observe que **p:viewName="index"** especifica o nome da visualização lógica, que é resolvida por meio de **viewResolver colocando-se o prefixo /WEB-INF/jsp/ e o sufixo .jsp.** **Isso permite que o tempo de execução localize o arquivo no diretório do aplicativo e responda com a visualização da página de boas-vindas (/WEB-INF/jsp/index.jsp).**

**Visão geral do aplicativo**

(fonte: <https://netbeans.org/kb/71/web/quickstart-webapps-spring_pt_BR.html#setting>)

**O aplicativo criado é composto de duas páginas JSP** **(que se chamam visualizações na terminologia MVC)**. A primeira visualização contém um formulário HTML com um campo de entrada solicitando o nome do usuário. A segunda visualização é uma página que exibe simplesmente uma mensagem hello contendo o nome do usuário.

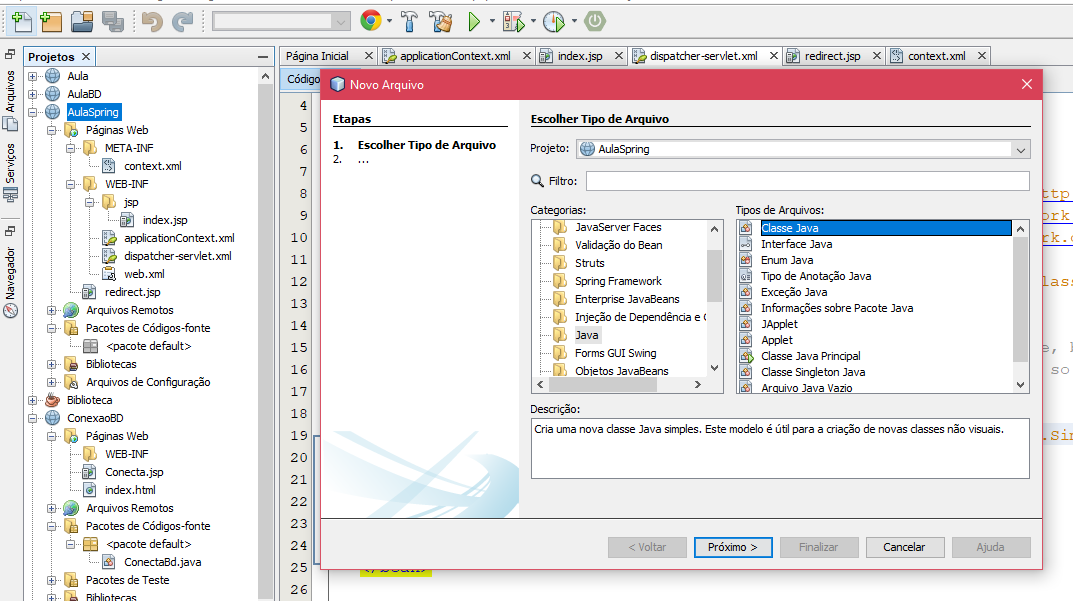
**As visualizações são gerenciadas por um controlador, que recebe as solicitações para o aplicativo e decide quais visualizações retornar.** Ele também passa para as visualizações quaisquer informações que precisam exibir **(isso é chamado de modelo).** Esse controlador do aplicativo se chama HelloController.

**Em uma aplicação Web complexa, a lógica comercial não é contida diretamente no controlador. Em vez disso, outra entidade, denominada serviço, é utilizada pelo controlador sempre que ele precisar realizar a lógica corporativa. Em nosso aplicativo, a lógica corporativa está limitada ao ato de processamento da mensagem olá, sendo assim, para essa finalidade, você cria um HelloService.**

**Criando uma aplicação:**

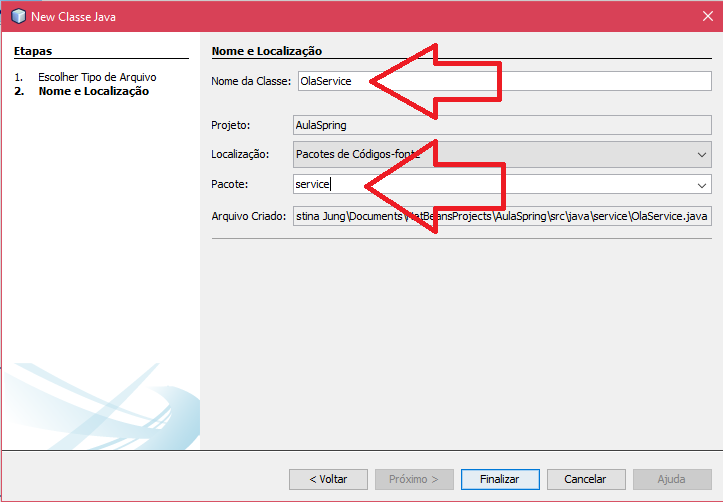
Agora que nosso ambiente está configurado adequadamente, vamos começar a criar a nossa aplicação Java.

**Vamos criar uma nova classe Java:**

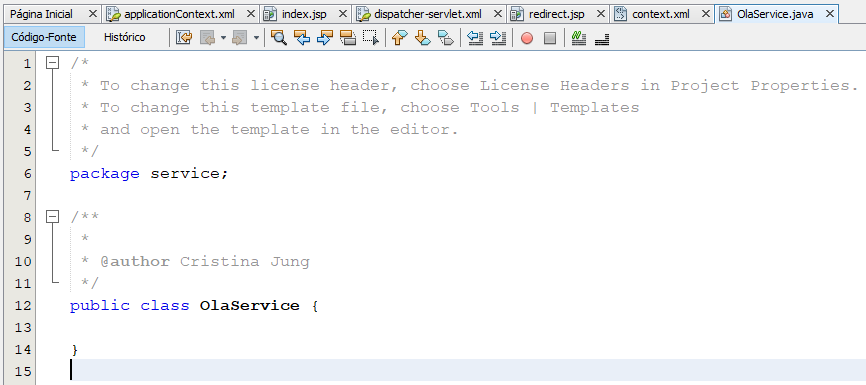


Na janela de assistente para Nova classe Java exibido, digite **OlaService** para Nome da classe e digite **service** para Nome do pacote a fim de criar um novo pacote para a classe.

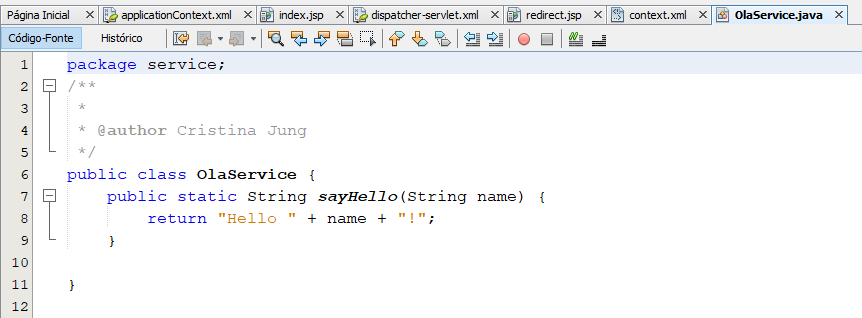
Clique em Terminar. O IDE cria a nova classe e a abre no editor.



A classe OlaService realiza um serviço muito simples. **Ela pega um nome como parâmetro, prepara e retorna uma String que inclui o nome. No editor, crie o seguinte método sayHello() para a classe (alterações em negrito).**

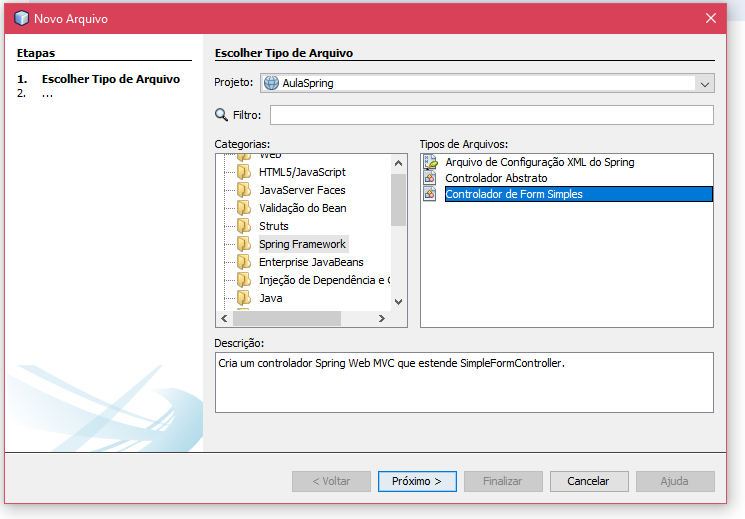


Vamos declarar os seguintes parâmetros!

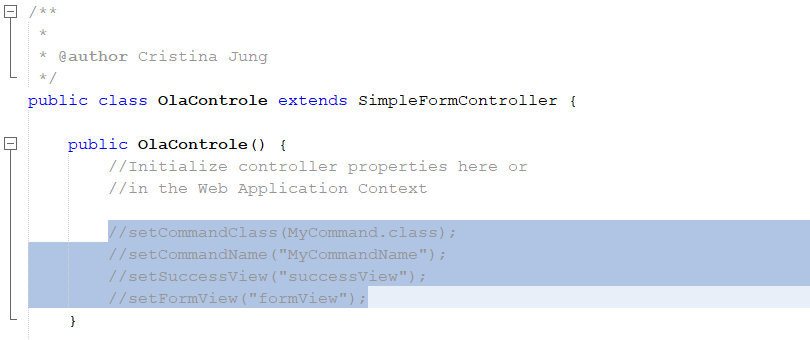


**Agora é a vez de passarmos para a aplicação do Framework! Vamos implementar o controlador do Modelo.**

Abra a janela do assistente para **Novo arquivo.** Em **Categorias**, selecione **Spring Framework** em **Tipos de arquivos e Controlador de formulário simplificado**.



1. Se você está trabalhando com o **NetBeans**, ele fornece modelos para vários artefatos do Spring, incluindo o arquivo de configuração do Spring XML, o AbstractController e o SimpleFormController.
2. Clique em Próximo.
3. Chame a classe de **OlaController** e crie um novo pacote para ela, digitando **controller** no campo de texto Pacote.
4. Clique em Terminar.
5. A IDE cria a nova classe e a abre no editor. Especifique as propriedades do controlador, removendo o comentário dos métodos setter exibidos por padrão no modelo da classe. Para retirar os comentários do trecho de código, realce o código como na imagem a seguir e pressione **Ctrl-/ ou retire as barras de comentários.**



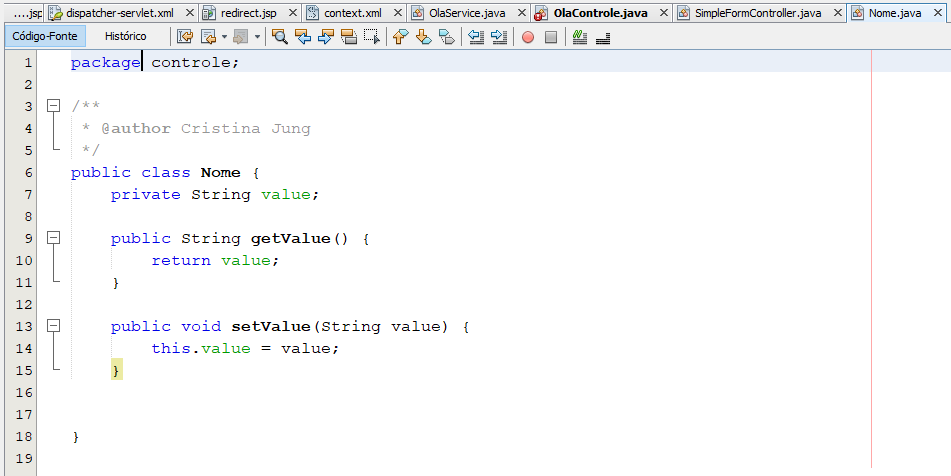
1. Faças as alterações a seguir (alterações em negrito). Depois pressione CTRL + SHIFT + I para abrir o corretor de importação. Verifique se a importação abaixo foi instalada.

|  |
| --- |
| importe [org.springframework.web.servlet.ModelAndView](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/servlet/ModelAndView.html); |

Na documentação da **API**, essa classe **"representa um modelo e uma visualização retornados por um manipulador, a serem resolvidos por um DispatcherServlet. A visualização pode tomar a forma de um nome de visualização do String, que precisará ser resolvido por um objeto ViewResolver, como alternativa, um objeto Visualização pode ser especificado diretamente. O modelo é um Mapa, que permite o uso de vários objetos, classificados por nome"**.

|  |
| --- |
| public HelloController() {  [setCommandClass](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/servlet/mvc/BaseCommandController.html#setCommandClass(java.lang.Class))(**Nome**.class);  [setCommandName](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/servlet/mvc/BaseCommandController.html#setCommandName(java.lang.String))("**nome**");  [setSuccessView](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/servlet/mvc/SimpleFormController.html#setSuccessView(java.lang.String))("**hello**View");  [setFormView](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/servlet/mvc/SimpleFormController.html#setFormView(java.lang.String))("**name**View");  } |

1. Acesse a janela Projetos, clique com o botão direito do mouse no nó do projeto e escolha Novo - Classe Java. O assistente para Nova classe Java é exibido.
2. Indique Name para o Nome da classe, e para Pacote, selecione **controler** na lista suspensa.
3. Clique em Terminar. A classe Nome é criada e aberta no editor.
4. **Para a classe Nome, crie um campo denominado valor e, em seguida, crie os métodos do acessor (por exemplo, os métodos getter e setter) para esse campo. Inicie declarando o campo valor:**



1. Vamos acessar agora o arquivo: HelloController e vamos declarar um campo privado chamado HelloService.

|  |
| --- |
| **private HelloService helloService;**  **public void setHelloService(HelloService helloService) {**  **this.helloService = helloService;**  **}** |

1. Agora, vamos importar a instrução a seguir é adicionada ao início do arquivo. Você pode pressionar CTRL + SHIFT + I para a janela de Importação aparecer e incluir a instrução. Se ela não aparecer é só digitar.

|  |
| --- |
| import service.HelloService; |

1. Próxima etapa, vamos registrar o **HelloService** em applicationContext.xml. Abra **applicationContext.xml** no editor e insira a seguinte declaração de bean:

|  |
| --- |
| <bean name="helloService" class="service.HelloService" /> |

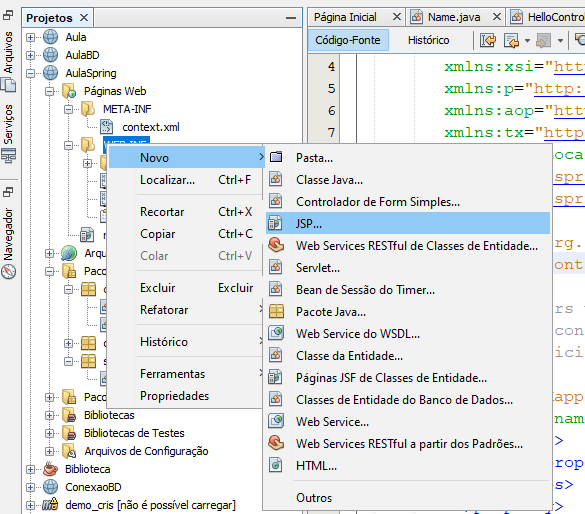
1. Registre **HelloController** em **dispatcher-servlet.xml**. Abra dispatcher-servlet.xml no editor e insira a seguinte declaração de bean:

|  |
| --- |
| <bean class="controller.HelloController" p:helloService-ref="helloService"/> |

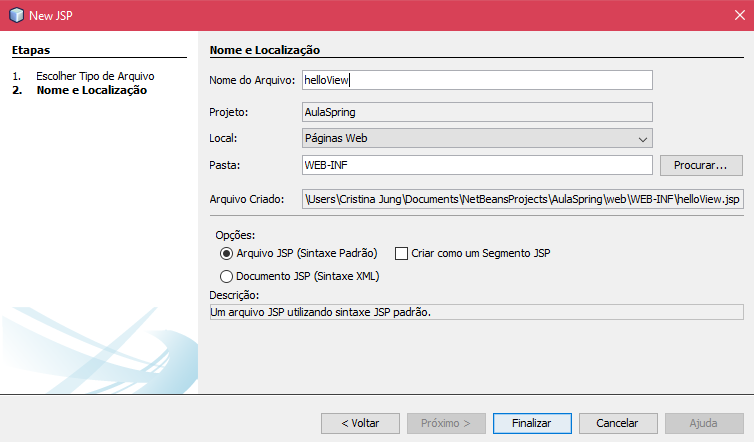
**Implementando as visualizações!**

**Para implementar a visualização desse projeto, vamos precisar criar duas páginas JSP.** A primeira, vamos chamar de **nameView.jsp,** irá ter a função de página de boas-vindas e vai permitir que um usuário insira um nome. A outra página, **helloView.jsp**, vai exibir uma mensagem de saudação que inclui o nome de entrada.

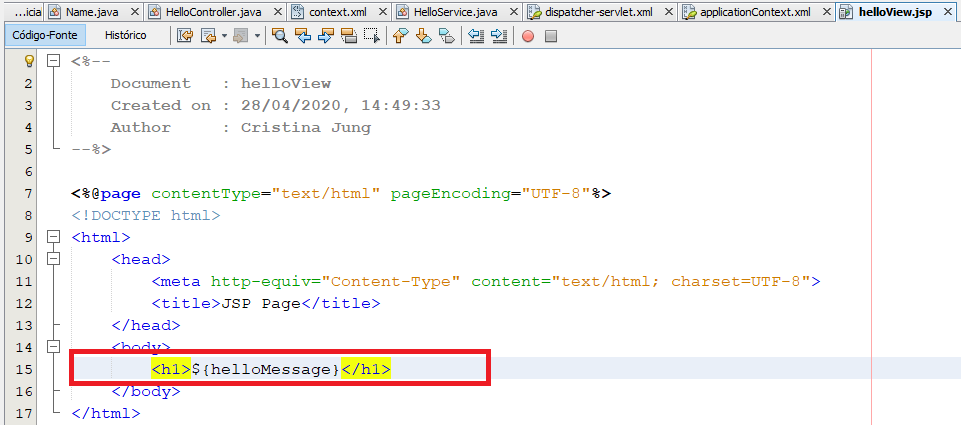
1. Vamos começar criando **helloView.jsp**. Na janela Projetos, clique com o botão direito do mouse no nó **WEB-INF --> jsp e escolha Novo > JSP**.



1. O assistente para Novo arquivo JSP se abre. Vamos colocar o nome de **helloView**.
2. Clique em **Finalizar**. A nova página JSP é criada na pasta jsp e é aberta no editor.



1. No editor, altere o título do arquivo para Olá e altere a mensagem de saída para recuperar o **helloMessage** do objeto ModelandView criado em HelloControll



1. Vamos criar outra página JSP, agora com nome de **nameView**. É só repetir os procedimentos que fizemos para criar o arquivo helloView. Esta tag importa a biblioteca de marcas Spring, que contém marcas úteis durante a implementação de visualizações como páginas JSP.

|  |
| --- |
| <%@taglib uri="http://www.springframework.org/tags" prefix="spring" %> |

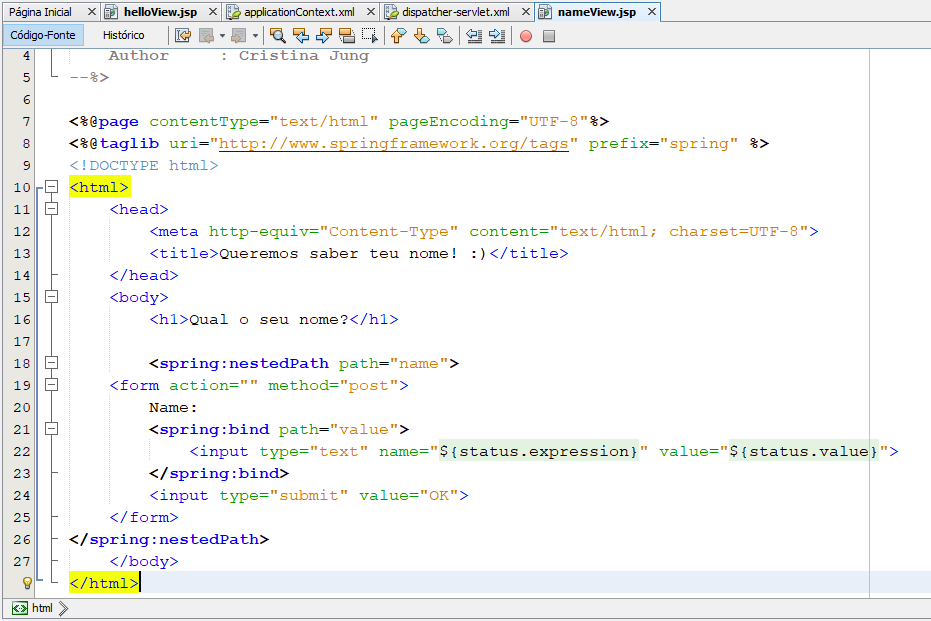
1. Faça as seguintes alterações no código! Qualquer texto que desejar!

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  **<title>Queremos saber teu nome! :)</title>**  </head>  <body>  **<h1>Qual o seu nome?</h1>** |

1. Após, vamos inserir vamos vincular as tags com as propriedades dos arquivos de do framework + os formulários do HTML. Coloque o código abaixo das tags **<h1> e </h1>.**

|  |
| --- |
| **<spring:nestedPath path="name">**  **<form action="" method="post">**  **Name:**  **<spring:bind path="value">**  **<input type="text" name="${status.expression}" value="${status.value}">**  **</spring:bind>**  **<input type="submit" value="OK">**  **</form>**  **</spring:nestedPath>** |

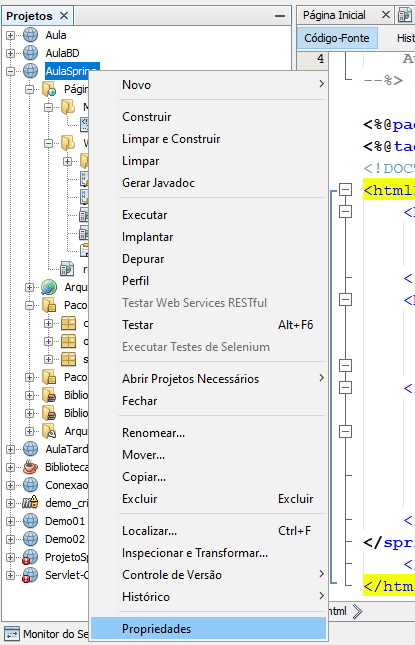
**Ficará desta forma:**



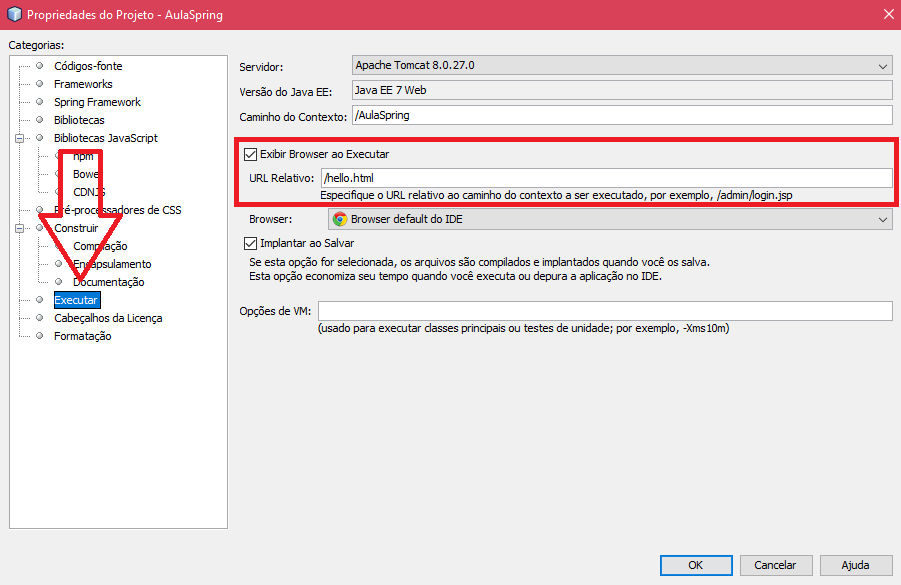
**spring:bind** permite que você vincule uma propriedade de bean. A marca de vinculação fornece um status de e valor de vinculação, que você usa como o nome e o valor do campo de entrada. **Dessa forma, quando o formulário é enviado, o Spring saberá como extrair o valor enviado. Aqui, nossa classe de comando (controller.Name) possui uma propriedade value, portanto, você define path como value.**

**spring:nestedPath** permite que você anexe um caminho especificado a um bean. Sendo assim, quando usado com spring:bind como mostrado acima, o caminho para o bean se torna: name.value. Como você lembra, o nome do comando de HelloController é name. **Portanto, esse caminho se refere à propriedade value de um bean chamado name no escopo da página.**

1. **Agora iremos alterar o ponto de entrada relativo do aplicativo.** Atualmente, o ponto de entrada do projeto ainda é **index.html** que, redireciona para **WEB-INF/jsp/index.jsp**.
2. Você pode especificar um ponto de entrada para o projeto quando ele é implantado e executado. Na janela Projetos, clique com o botão direito do mouse no nó do projeto e escolha Propriedades.



1. A caixa de diálogo Propriedades do projeto é exibida. Em Categorias, selecione Executar. No campo URL relativa, digite **/hello.html,** em seguida, clique em OK.



**Ops!! E agora??!! você deve estar pensando onde o mapeamento de hello.html para HelloController se encontra.** **De onde surgiu isso???** Não foi adicionado um mapeamento para o **bean urlMapping**, como é o caso de **index.html**, a página de boas-vindas do projeto de esqueleto. Pois então, isso é possível com um pouco da mágica do **Spring fornecida pela definição do bean seguinte em dispatcher-servlet.xml:**

|  |
| --- |
| <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/> |

**Agora é só executar o projeto!**

|  |  |
| --- | --- |
|  |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_