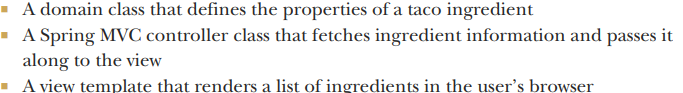
2. Developing web applications

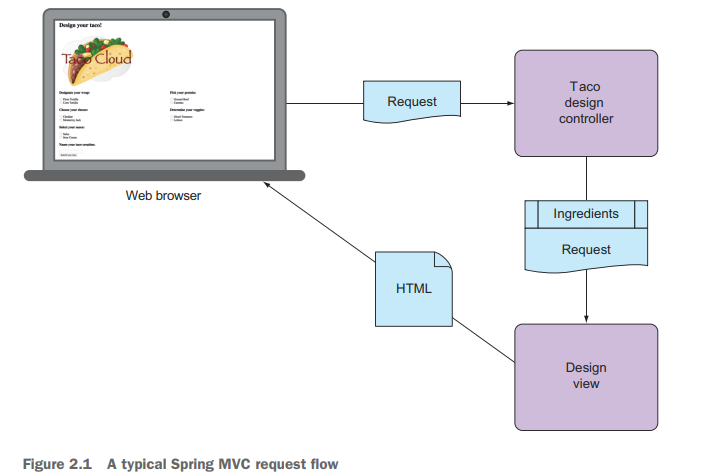
-First impression the user will get comes from the UI. In many application, UI is a web application presented in a web browser.

# 2.1 Displaying information

-Taco Cloud: order tacos online, design custom tacos.

-You are going to create 3 components of taco creation:

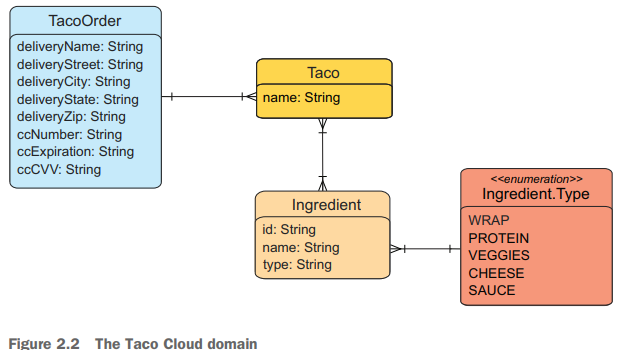




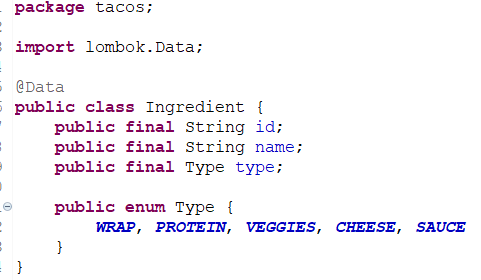
## 2.1.1 Establishing the domain

-Application domains: the ideas and concepts that influence the understanding of application

+Taco Cloud domain: taco designs, the ingredients, customers, taco orders. These entities showed:



-Ingredient domain class:

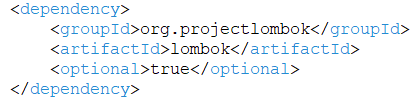


+The library **Lombok**: **@Data** tell Lombok to generate all **missing methods** and a constructor accept all final properties as arguments.

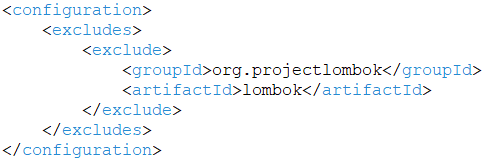
-Lombok isn’t a Spring library but so useful. You need to add it yourself.

+In STS: right click pom.xml->Add Starters

+Add manually:

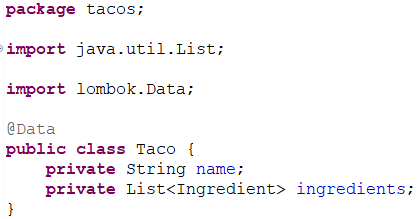


And exclude it from Spring Boot Maven plugin in <build>



-Lombok dependency provides Lombok annotation at development time and automatic method generation at compile time.

-Taco domain class:



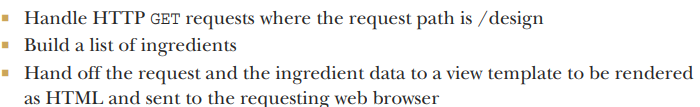
-TacoOrder class:



## 2.1.2 Creating a controller class

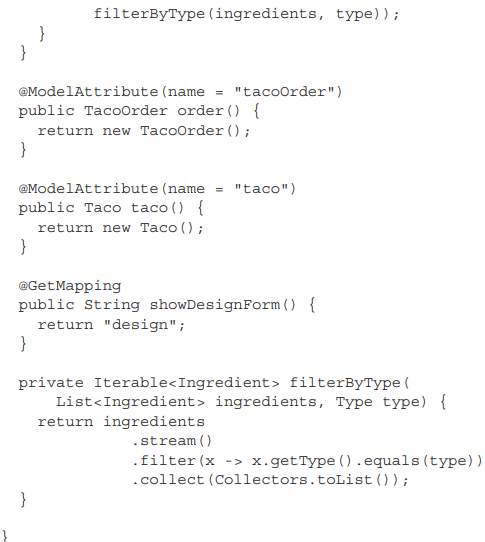
-**Controllers** are the major players in Spring MVC framework. Its job is to handle HTTP requests and either hand off a request to a view to render HTML(browser-displayed) or write data directly to the body of a response (RESTful)

-You need a controller that will do:



-Design Controller Class





**+@Slf4j**: Lombok annotation, at compile time: generate an SLF4J Logger static property in class (<https://www.slf4j.org/>). It is the same with this lines:



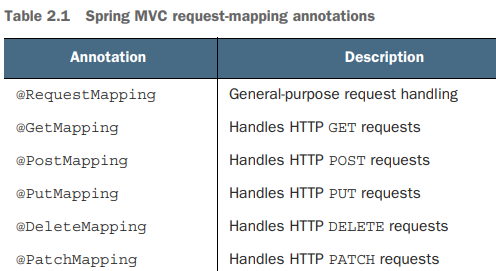
+**@RequestMapping():** specifies what kinds of requests the controller will handler

**+@SessionAttributes(“a”)**: The A object that is put into the model should be maintained in session.

-Handling a GET Request:

**+@GetMapping**+showDesignForm(): when HTTP GET request is received for /design, SpringMVC will call showDesignForm() to handle the request.

+List of all request-mapping annotation in Spring MVC



+showDesignForm(): populates the given Model with an empty Taco object + return logical name of the view.

+**@ModelAttribute**+addIngredientsToModel(): invoked when a request is handled and will construct a list of Ingredient objects to be put into the model, then filter the list by ingredient type filterByType(). Next, the list is added as an attribute to Model object that passed into showDesignForm().

+**Model** is an object that ferries data between a controller and view to render data. Data placed in Model attribute is copied into the servlet request attributes, where the view can find them and use to render

## 2.1.3 Designing the view

-View libraries are designed to be decoupled from web framework. So they are unable to work with data that the controller places in Model. But they can work with servlet request attributes. Therefore, before Spring hands the request over to view, it copies the model data into request attributes that view templating have ready access to.

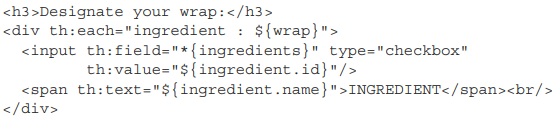
-**Thymeleaf** templates are HTML with some additional element attributes that guide in rendering request data.

-Example: render a request attribute whose key is “message” into <p> tag by Thymeleaf:



+The body of <p> element will be replaced with the value of the servlet request attribute whose key is “message”. th:text attribute is Thymeleaf namespace attribute that performs the replacement. ${} operator tells it to use the value of request attribute.

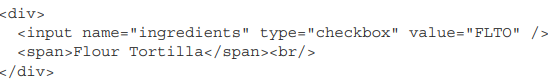
-**th:each** iterates over a collection of elements, rendering the HTML once for each item in collection:



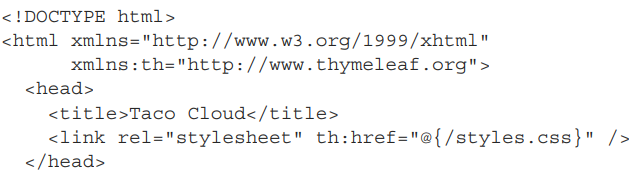
+Each iteration the item in collection found in wrap request attribute, the ingredient item is bound to Thymeleaf variable named ingredient.

+<input> and <span>: provide a label for the check box. <th:value>: set the rendered <input> element’s value attribute. th:field sets the <input> element’s name attribute and used to remember whether the check box is checked.

+When rendered with actual model data, one iteration might look:

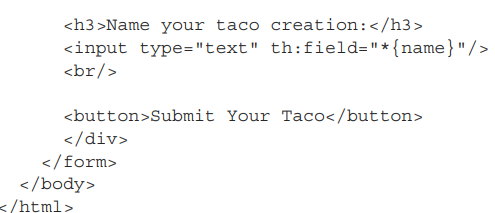


-Design.html



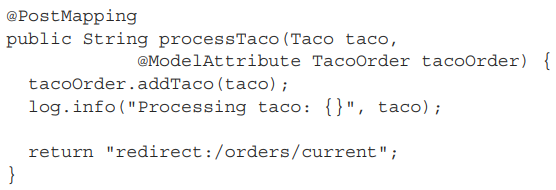






# 2.2 Processing form submission

-The <form> doesn’t declare an action attribute. So when the form is submitted, the browser will gather all data in form and send to server in an HTTP POST request to the same path for which a GET request displayed the form.

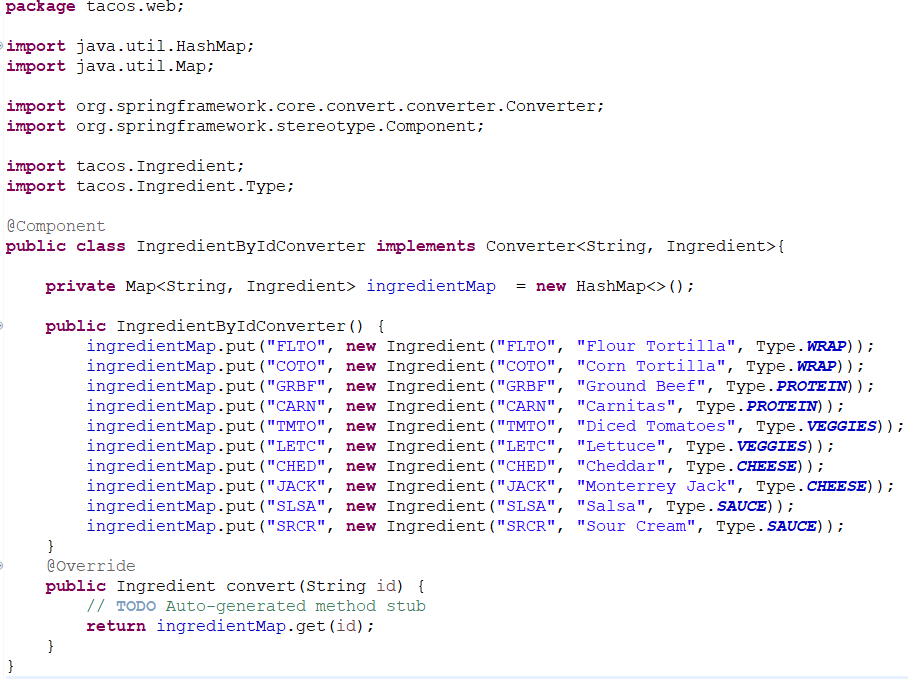


**+@PostMapping** coordinates with **@RequestMapping** to indicate the method should handle POST request for the path.

**+@ModelAttribute** applied to **TacoOrder** parameter indicates it should use the TacoOrder object that was placed into the model via the **order()**

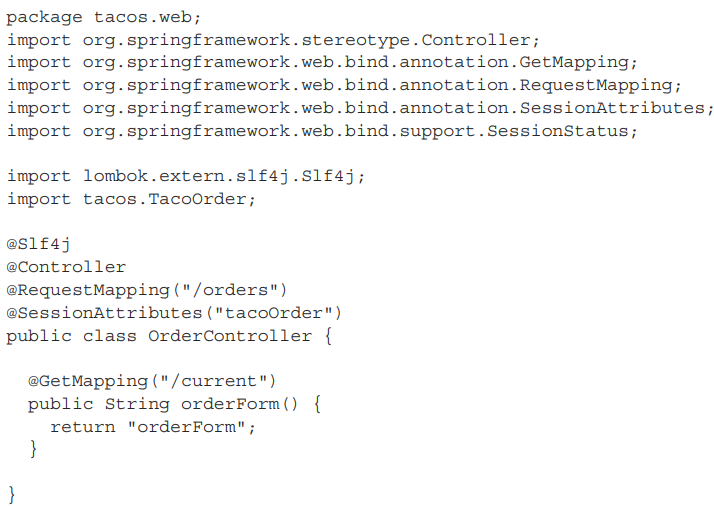
+**redirect**: indicate that this is a redirect view. Specifically, after processTaco() completes, the browser should be redirected to /orders/current

-**Converter:** class implements Spring’s Converter interface and **convert()** to take one value and convert it to another. To **convert** **String -> Ingredient**:



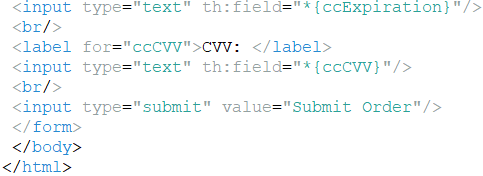
**+@Component**: make it discoverable as a bean in container. This bean is be used when the conversion of request parameters to bound properties is needed.

-Controller handle **/orders/current**



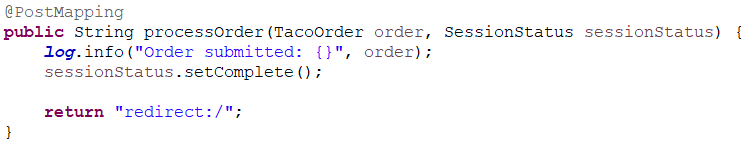
-orderForm.html





+Notice that <form> tag specifies a form action HTTP POST request to /orders.

-Handle the form action:



+The method is given a TacoOrder object whose properties are bound to form fields. The order object carries order information.

+**setComplete():** the session is cleanup up and ready for a new order the next time the user creates a taco.

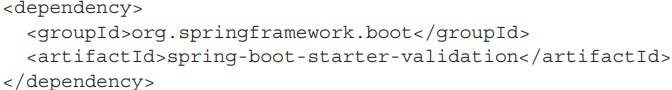
# 2.3 Validating form input

-One way to perform form validation is if/then blocks, but that would be difficult to read and debug.

-Spring supports **JavaBean Validation API (JSR303**: <https://jcp.org/en/jsr/detail?id=303>**)**

**+**The Validation API offers **annotations** that can be placed on properties of domain objects to declare validation rules.

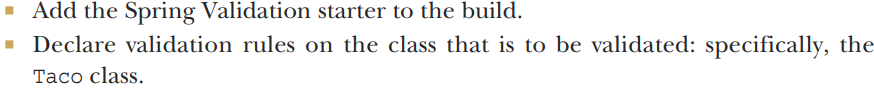
+Maven pom.xml:

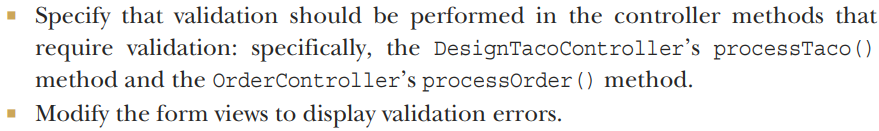


+Gradle:



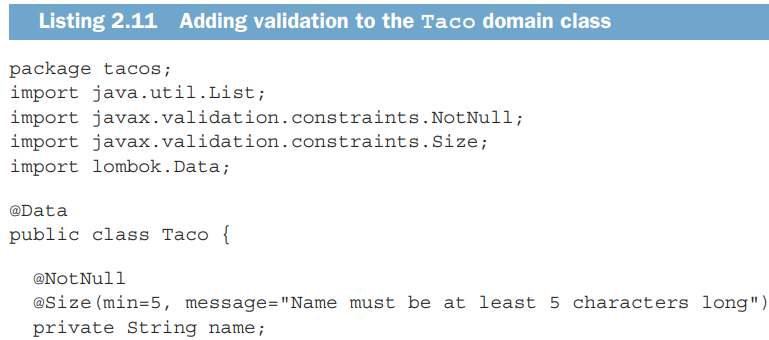
-Add validation in Spring MVC:

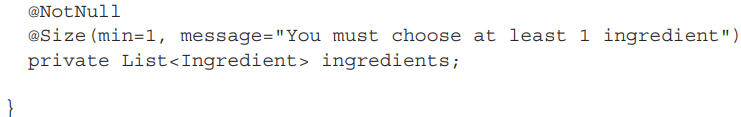




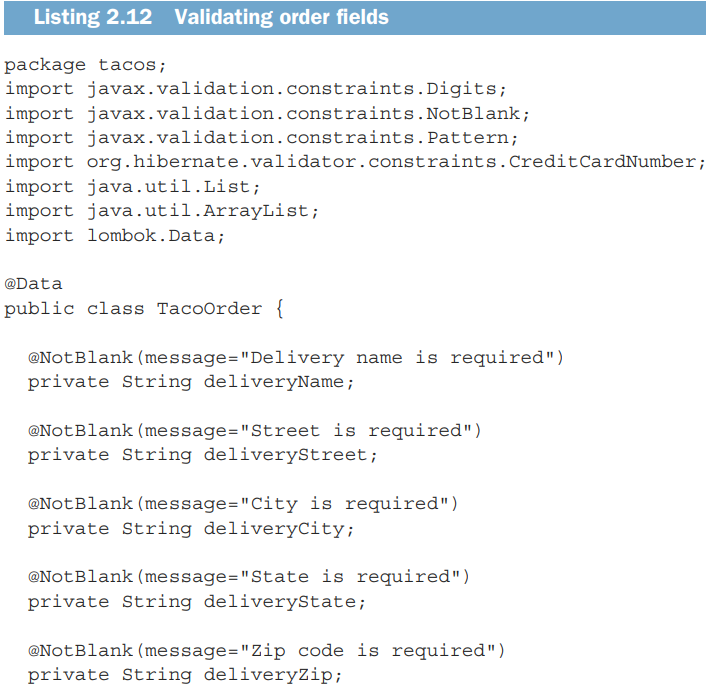
## 2.3.1 Declaring validation rules

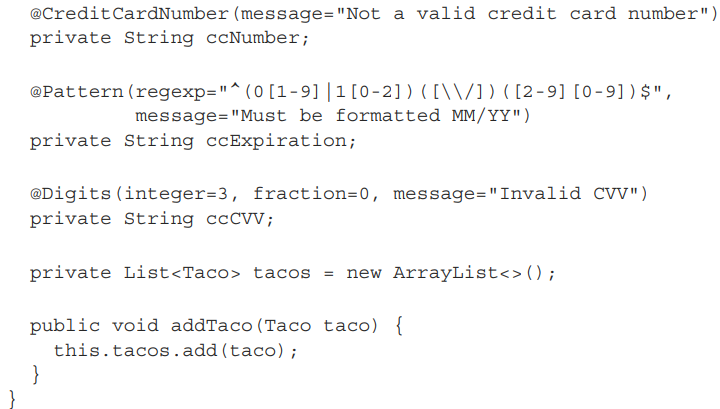
-Add validation to domain class:





-Validate class TacoOrder:

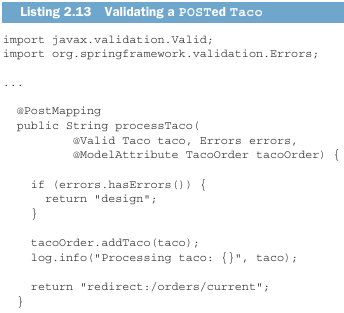




## 2.3.2 Performing validation at form binding

-After you have declared validation in Taco and TacoOrder, we revisit each of the controllers, specify that validation should be performed when then forms are posted to their handler methods.

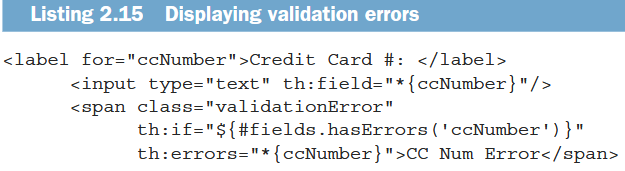
+To validate a submitted Taco, add **@Valid** to Taco argument in Taco():



## 2.3.3 Displaying validation errors

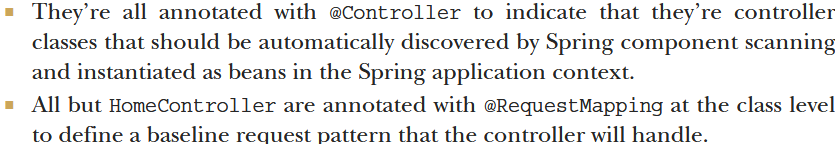
-Thymeleaf offers access to Errors object via **fields** and **th:errors**

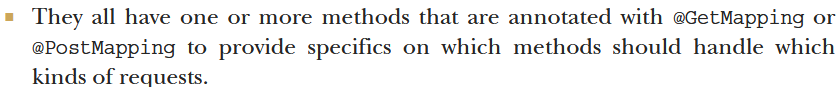
+Example: display validation errors on credit card number field, use a <span>



# 2.4 Working with view controllers

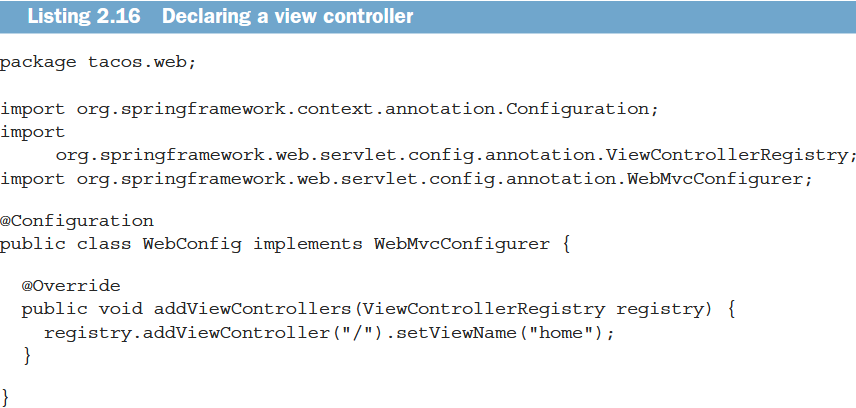
-All controllers follow programming model:



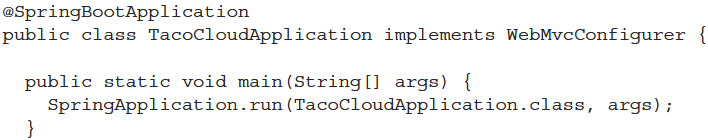


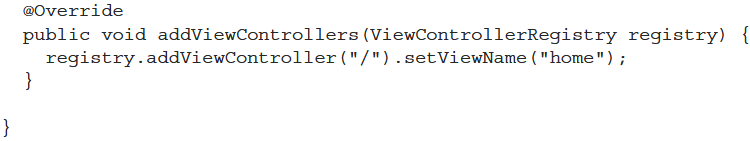
-If a controller is simple enough that it doesn’t populate model or process input (HomeController), there is another way to define controller.

-Example: a controller that just request to a view.



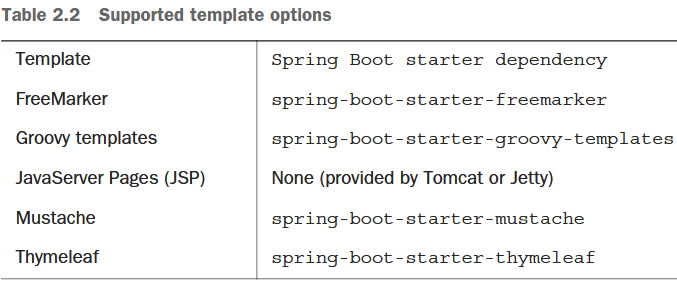
-Any configuration class can implement **WebMvcConfigurer** and override **addViewController()**. Example: add view controller declaration to TacoCloudApplication





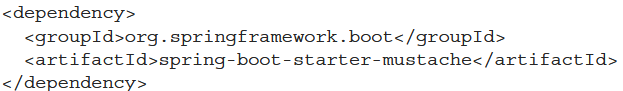
# 2.5 Choosing a view template library

-The choice of view template library is a matter of personal taste. Spring is flexible and supports many common templating options.

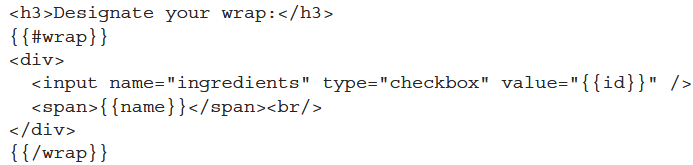


-Select the view template library-> add it as a dependency in your build->start writing template in /template directory. SB detects the chosen template library and automatically configures the components.

-Use Mustache:



+A snippet from **-** template that renders one of ingredient groups in taco design form:



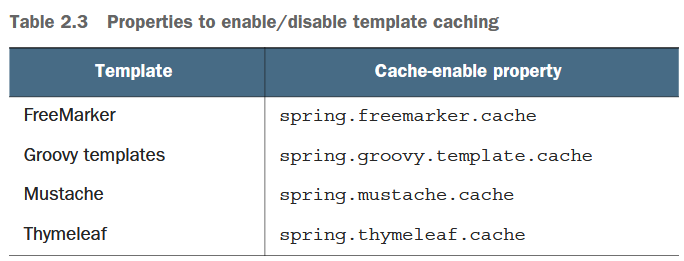
-The servlet container (Tomcat by default) implements **JSP** specification->doesn’t require any denpendency

-Build your app as WAR file and deploy it in a traditional servlet container: use JSP

-Build JAR file: Thymeleaf, FreeMaker…

## 2.5.1 Catching templates

-Caching: the templates have to be compiled only one. If you disable it, the templates have to be compiled repeatedly for every request.



-Disable caching in **application.properties**



# Summary

