Why should we care about **Server Side Validations** ?

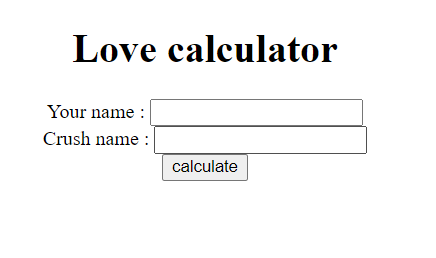
Even though, we have Client-Side-Validations which is more efficient in terms of speed.

Client-Side-Validations respond quickly if user is breaking any validation.

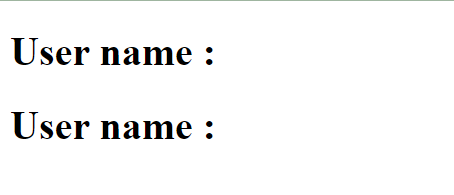
But we should validate data at Client Side as well as Server Side.

So the question is what are those problems with Client Side Validations which make us to validate data at Server Side also ?

We have a running project **lovecalculator**



Suppose we don’t fill any data in any of field then what happens is :



It allows us to submit data and shows nothing in front of Username and Crushname.

So now we want to validate data at client side. To do that we have make some changes in our **homepage.jsp** by adding some JS code.

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Home page</**title**>

<**script** type=*"text/javascript"*>

**function** submitHandler(){

let yourname = document.querySelector("#yourname").value;

let crushname = document.querySelector("#crushname").value;

**if**(yourname.length<1 || crushname.length<1) {

alert("Field cannot be empty !")

**return** **false**;

}

**return** **true**;

}

</**script**>

</**head**>

<**body**>

<**div** align=*'center'*>

<**h1**>Love calculator</**h1**>

<**form** action=*"/lovecalculator/myhome/processhomepage"* onsubmit="return submitHandler()">

<**div**>

<**label** for=*'yourname'*>Your name : </**label**>

<**input** type=*'text'* name=*'yourName'* id=*'yourname'* />

</**div**>

<**div**>

<**label** for=*'crushname'*>Crush name : </**label**>

<**input** type=*'text'* name=*'crushName'* id=*'crushname'* />

</**div**>

<**div**>

<**input** type=*'submit'* value=*'calculate'* />

</**div**>

</**form**>

</**div**>

</**body**>

</**html**>

After adding the above code into homepage.jsp you will see that your webpage is validating perfectly.

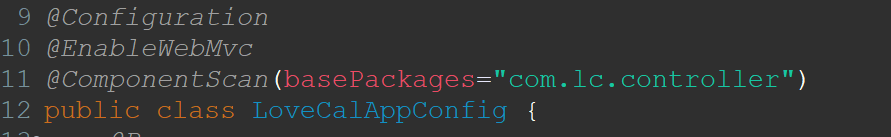
Now assume, a web designer is using your website. What he can do to bypass the validation.

1. In our JS code we are accessing the HTML input element using it’s id. What if he/she removed that id.
2. What if he/she disabled the JavaScript ( can be easily disabled from inside chrome settings ).

These are two way by which anyone can bypass the Client Side Validations and nobody can stop invalid data insert into your database.

Please, remove the Client Side Validations that we just applied so that we can do the same validation at Server Side.

Before move ahead, please check your configuration file i.e. LoveCalAppConfig.java if it is annotating with **@EnableWebMvc** annotation. If not, do it right away. Without this annotation, our bean validation is not going to work.



**Introducing Bean Validation API**

Bean validation API is actually JSRs and there are different versions available of it like JSR 303, JSR 349, JSR 389

What do you mean by JSRs ?

It is stand for **Java Specification Requests.**

So there is a community called **JCP (Java Community Process)** if anyone is the member of this community so he/she can share their idea for improvement in Java. To share the idea he/she have to file a JSR and it will reviewed by the concern team and if the idea/specification is really make sense then that specification will be release with the next update.

Bean validation API is a kind of JSR. It’s first version was JSR 303 and latest is JSR 380. As a conclusion, we can say that **bean validation api 2.0** can also be termed as **JSR 380.**

**bean validation api 1.0 -> JSR 303**

**bean validation api 1.1 -> JSR 349**

So there many number of JSRs available, you can check that out by following the <https://jcp.org/>

As we discussed, JSRs are only the Specification and the implementation is provided by some other companies. Hibernate Validator is most popular out of other companies implementation. We also are going to use this.

So just two dependencies showing below :

<!-- https://mvnrepository.com/artifact/javax.validation/validation-api -->

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>2.0.1.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->

<dependency>

<groupId>org.hibernate.validator</groupId>

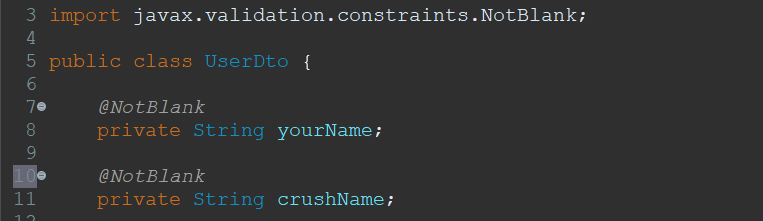
<artifactId>hibernate-validator</artifactId>

<version>8.0.0.Final</version>

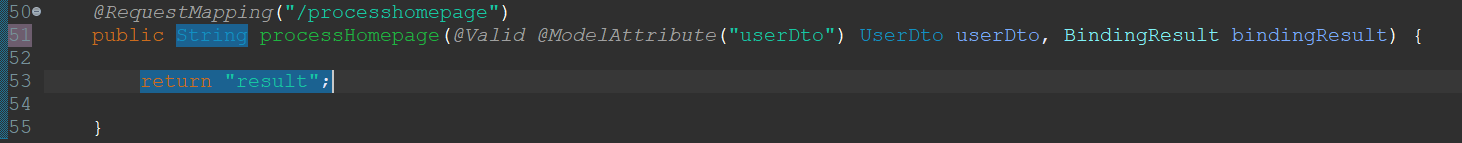
</dependency>

Now theory over let’s do some practical !

As a first step, you need to use @NotBlank annotation over the top of your DTO class fields.



Now see the prototype of processHomepage() method



We are using @Valid annotation before @ModelAttribute, the position of these two cannot be swap.

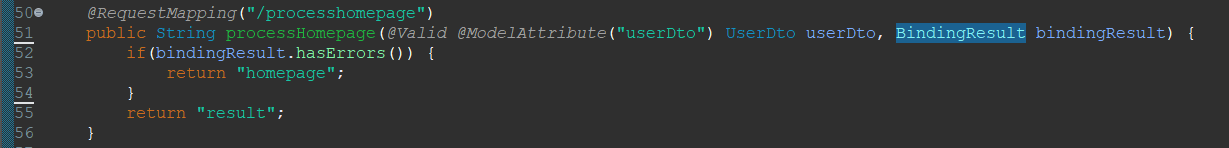
Whatever data is coming from the client spring will create **UserDto** object by using this data and then inject this object into given parameter.

The @Valid annotation is actually forcing spring to scan **UserDto** object for validation we have applied inside **UserDto**.

After scan, spring will generate a object of **BindingResult** which helps us to detect whether the validation fail or pass.

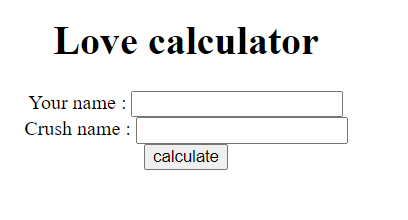
**Let’s implement how we are going to exploit BindingResult object to valid data.**

Look at the below picture !



**BindingResult** interface has a method called **hasErrors()** which returns true if any validation is breaking.

Now suppose, in the below form



User don’t provide Crush Name and hit on calculate button. What will happen then ?

So, as user hits on calculate button. A request will be fire for /processhomepage endpoint.

Then spring try fill the data i.e. Crush Name and Your Name inside **UserDto**. While filling, spring will see that Crush Name is a blank string and we have already annotate the crushName field of UserDto class by @NotBlank annotation. This will lead to break validation.

According to that, spring will create appropriate **UserDto** and **BindingResult** object and then pass both of them as a parameter to **processHomepage()** method**.**

Now, as we can guess that there is one error in our **UserDto**. So, how can we detect that error.

As we have already mentioned above, **BindingResult** has a method called hasErrors().

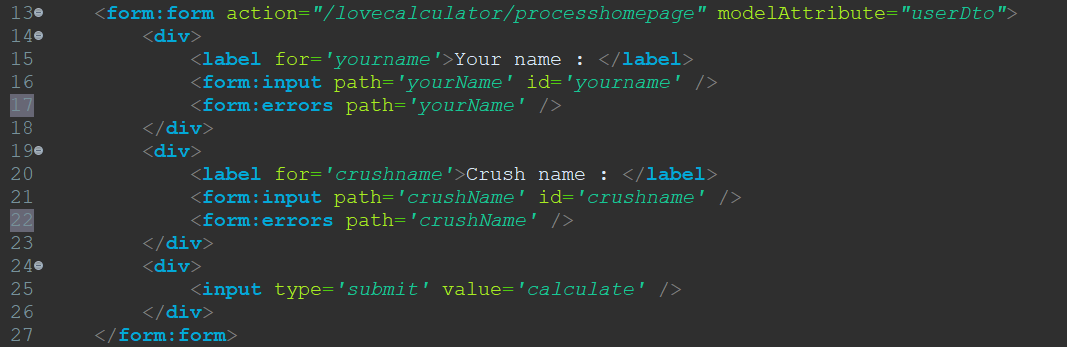
In this case, this method is going to return us true. Then the **if** condition of **processHomepage()** method will be true and if body will execute.

Inside **if** block we are returning the user on the same page. Likewise, the program is running in our favor. But in context of user, we are not acknowledge user that he/she is making some error. To do so, we should print some error method right beside of text fields.

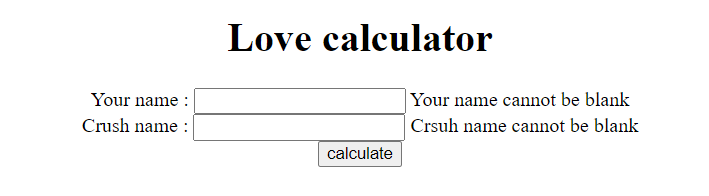
**How we do that ?**

It is very simple. Form API is going to help us !

Form API provides a tag **<form:errors />.** We’ll use that tag to display error messages.



After added this tag into your JSP you page should display error messages also.



If you are wondering that from where these error messages are coming. So, these are those messages which was set while adding @NotBlank validation into the **UserDto** class.

Below picture will give you idea !

