**Simple Form Controller**

**The main aim of this is to submit data from front end using html controls and display the same data in another page.**

Simple form controller is a way of designing the controller where a single URL can handle both **GET** and **POST** request. That means we have two methods i.e. 1 method to handle GET request and another method to handle the POST request.

This simple form controller allows us to do form handling mechanism. Form handling mechanism means the data entered by user should not be lost if validation error occurs. This process we will discuss later more.

The user input data is bounded to a commandObject. Actually when user sends a get request to the controller then the commandObject’s dataMembers will be initialized with default values or with some values provided by the developer. To understand this look the below code.

@RequestMapping(method=RequestMethod.*GET*,value="register")

**public** String showRegisterPage(Model model)

{

Student s=**new** Student("Rajesh","kumarsethi.rajesh@gmail.com");

model.addAttribute("studentCommandKey", s);

**return** "registerView";

}

Above code is kept as part of **com.spring.controller.DemoController**. This method accepts a GET request with incoming URL as /register.

**public** **class** Student {

**private** String name;

**private** String email;

// getters and setters

}

Here Student is a commandClass. This class has 2 attributes such as name and email. These two fields are initialized with “Rajesh” and “kumarsethi.rajesh@gmail.com”. To initialize with default values we have to write **Student s= new Student ()** instead of Student s=**new** Student ("Rajesh","kumarsethi.rajesh@gmail.com"); . Here is the code for our commandObject class (Student.java).

Now we add the commandObject to model with a key by writing the below code.

model.addAttribute ("studentCommandKey", s);

In frontend [means in jsp page] we will have a form that contains text fields. And these values will be populated to those respective fields.

The form designed in front end uses spring form tags. The input field names must be same as the attribute names of command class.

<form: form action=*"register"* commandName=*"studentCommandKey"*>

Name

<form: input path=*"name"* />

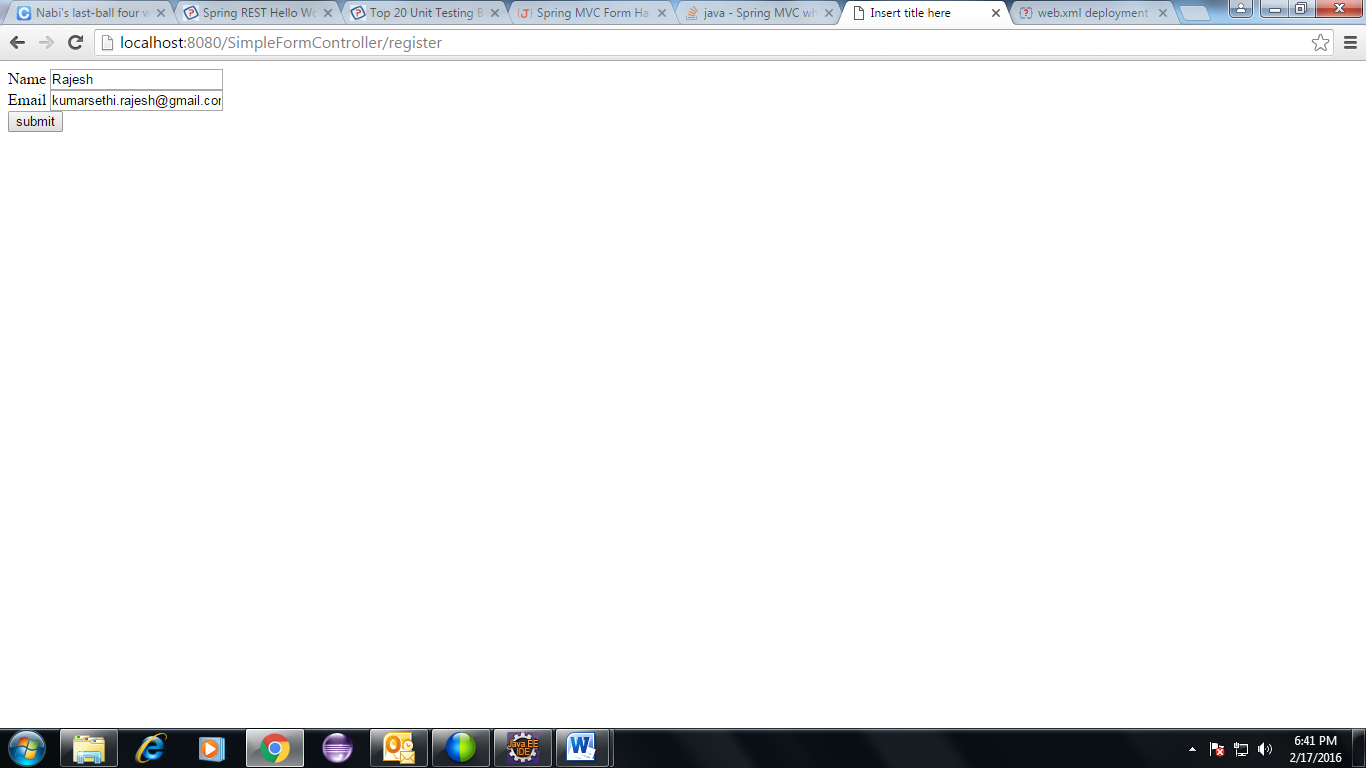
<br /> Email

<form: input path=*"email” /*>

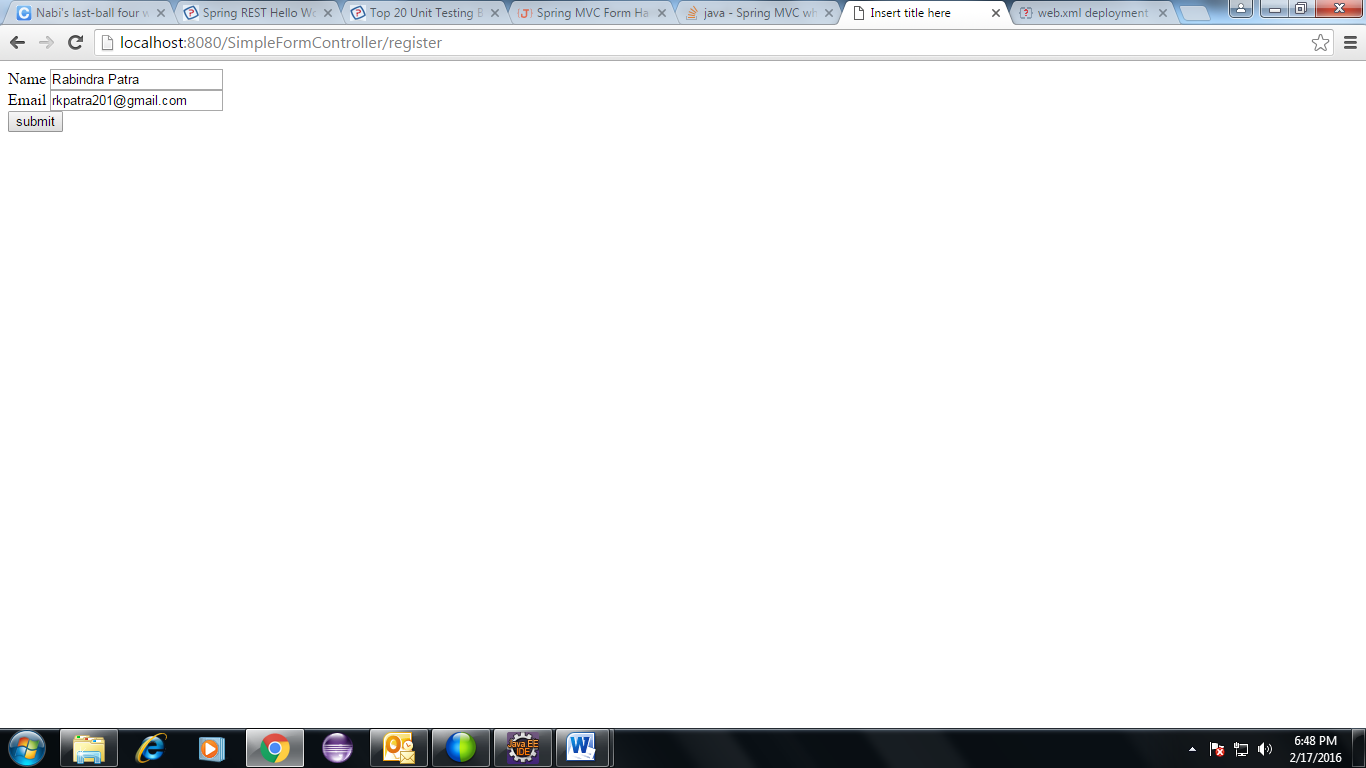
<br /> <input type=*"submit"* value=*"submit"* />

</form: form>

Above code is kept as part of **registerView.jsp.** In the above code you need to notice that commandName is same as the key with which we have added the commandObject to the model. So when you hit the URL as shown in below image you will get the similar result. It means the input fields of the form have been populated with some values.



Now you can remove the texts from the textboxes and can add some value as you want like I did in the next snapshot where I have added my name and email instead of yours. Then press the submit button.



Now you are submitting post request to the controller with URL as /register. This request is accepted by the below method.

@RequestMapping (method=RequestMethod.*POST*, value="register")

**public** String processRegisterPage (@ModelAttribute ("studentCommandKey") Student s,Model model)

{

String name=s.getName();

String email=s.getEmail();

String message="You have submitted:<br/> Name: "+name+"<br/> Email: "+email;

model.addAttribute ("messageKey", message);

**return** "success";

}

Above code is kept as part of **com.spring.controller.DemoController.** This code accepts a POST request with the URL /register.

We have commandName as part of front end jsp page is studentCommandKey. When we send post request the form input field data is converted into an object and bounded against the key studentCommandKey. We retrieve input data using the same key with the help of spring provided annotation @ModelAttribute. And populate into the **Student** object as part of the **method parameter**. Now we can extract the input values in our controller by calling the getters of the commandObject.

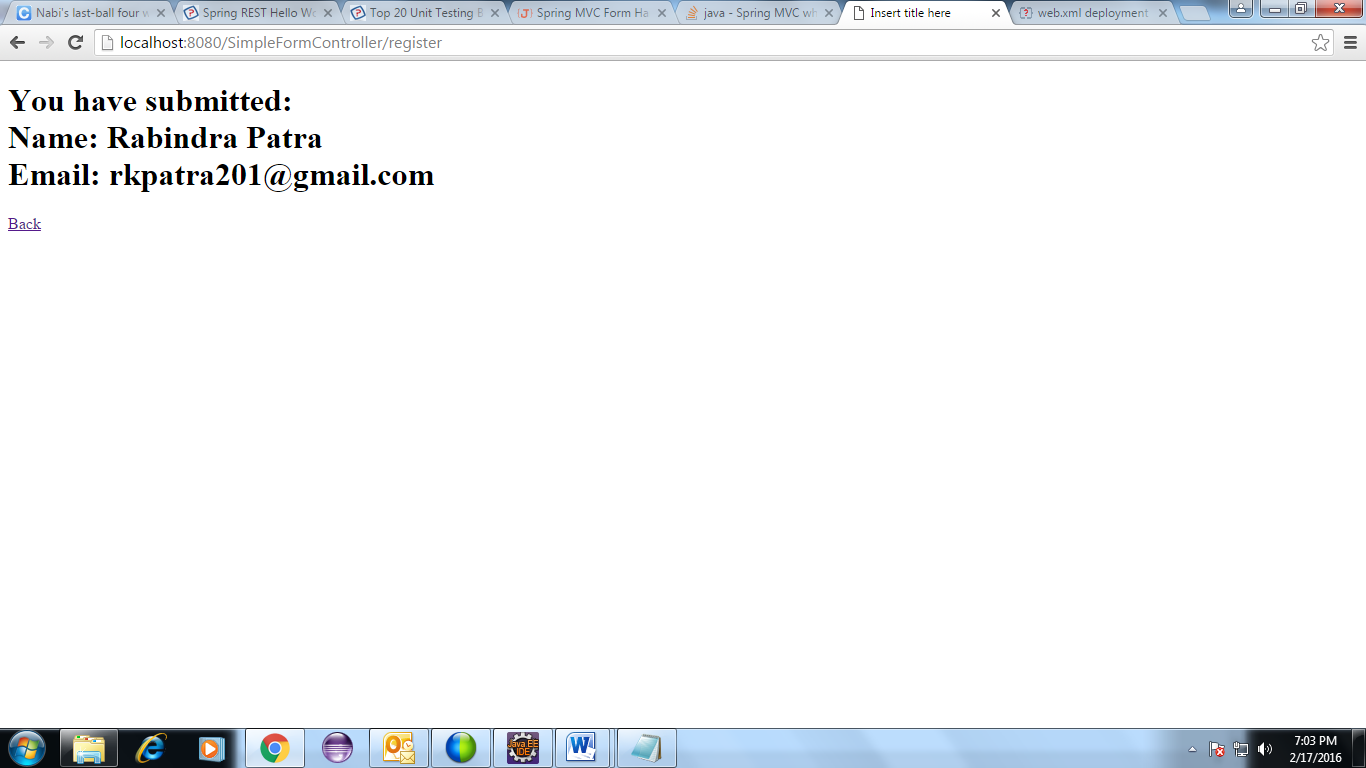
And then we created a message and added all input request values there. We added the message as part of the model with key as **messageKey**. So in the success.jsp we will retrieve the message using the key **messageKey**. Below code is used in success.jsp.

${messageKey}

The above is standard jstl syntax to display data. We have added the below jstl taglib in **success.jsp** page.

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

And below is the snapshot of success.jsp.



To run the program right click on your project then run on server.