

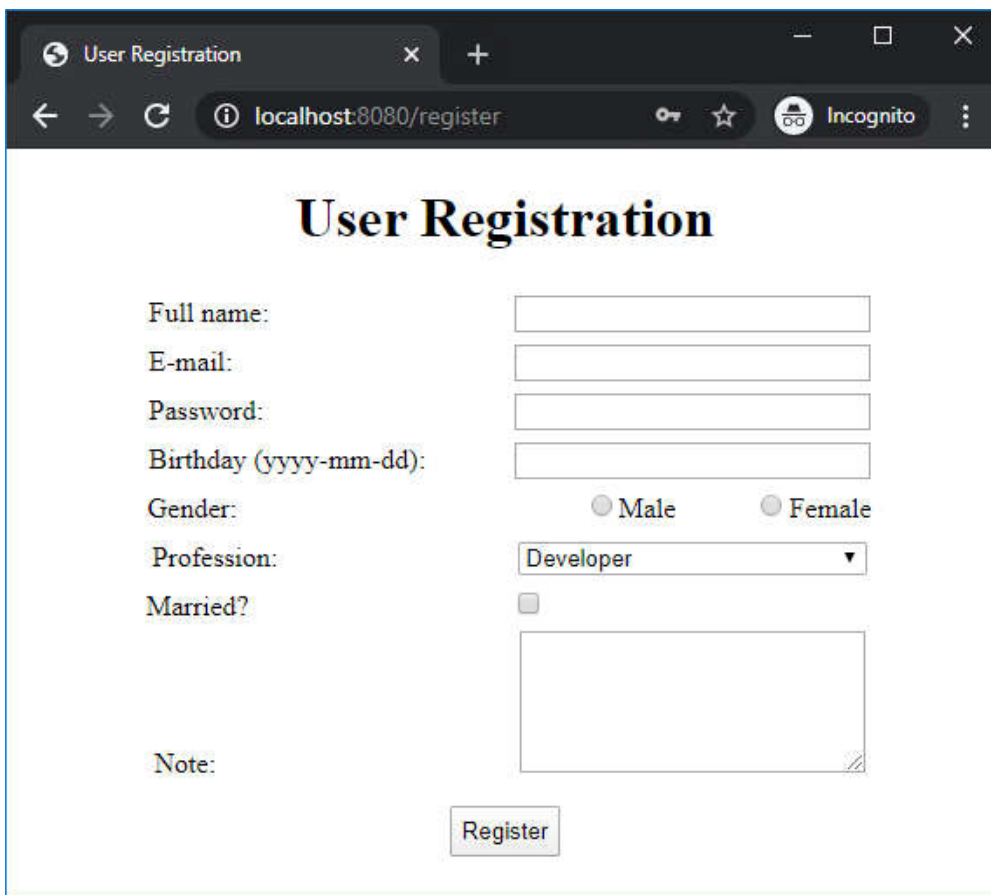
# Spring Boot Thymeleaf Form Handling Tutorial

Written by [Nam Ha Minh](#)

Last Updated on 04 March 2020 | [Print](#) [Email](#)

In this Spring Boot tutorial, you will learn to code a web form that captures information from user and handle form submission – read form data. We will use Spring Boot as it simplifies Java web development and Thymeleaf as it keeps HTML templates still look like HTML when mixed with code (Thymeleaf is well-integrated with Spring MVC).

Visually, I will walk you through the process of developing a Spring Boot application that presents the following form to the user:



The screenshot shows a web browser window titled "User Registration" with the address bar displaying "localhost:8080/register". The form is titled "User Registration" and contains the following fields:

- Full name:
- E-mail:
- Password:
- Birthday (yyyy-mm-dd):
- Gender: ☐ Male ☐ Female
- Profession:
- Married?: ☐
- Note:

A "Register" button is located at the bottom of the form.

You will learn to use Thymeleaf to code this form with almost all of standard HTML fields like input textbox, password field, radio button, select/dropdown list, check box and textarea – and also handle submission of this form with Spring MVC.

To follow this tutorial, I recommend you to use [Spring Tool Suite IDE](#) as it is dedicated for Spring projects development.

## 1. Create Spring Boot project with Thymeleaf and DevTools

In Spring Tool Suite, create a new Spring Starter project with Java 1.8 and Maven. Use the default packaging type which is jar. Choose the starters web, thymeleaf and devtools. The dependencies look like this in the Maven's build file:

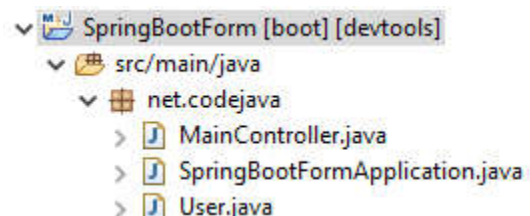
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

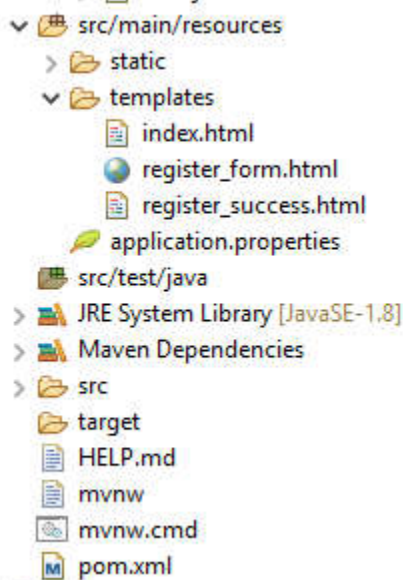
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
</dependencies>
```

As you can see, the `spring-boot-starter-web` dependency enables Java web application based on Spring MVC and running on embedded Tomcat server. The `spring-boot-starter-thymeleaf` is for integration between Spring MVC and Thymeleaf template engine.

And the `spring-boot-devtools` dependency enables [automatic restart](#) and [live reload](#) features so whenever you make changes to the project, Spring Boot will automatically restart the application and refresh the browser – making your development experience more convenient.

The project's directory structure looks like this:



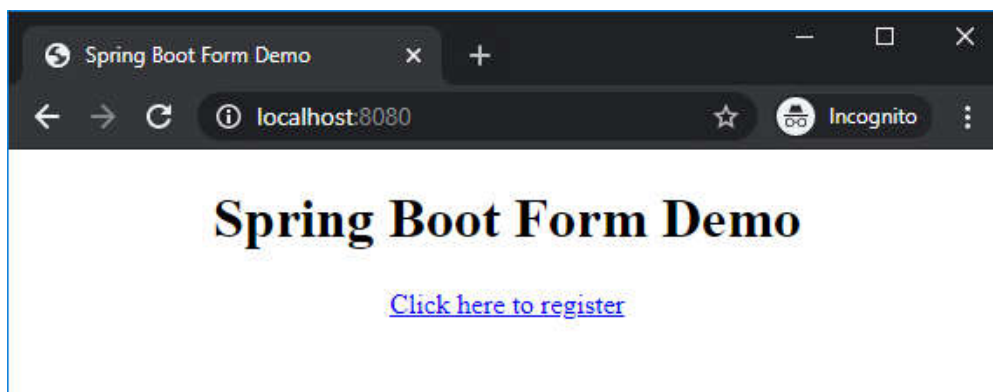


## 2. Code the Homepage

We put Thymeleaf template files under the `src/main/resources/templates` directory. For the homepage of the application, create the `index.html` file with the following code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Spring Boot Form Demo</title>
</head>
<body>
  <div align="center">
    <h1>Spring Boot Form Demo</h1>
    <a href="/register">Click here to register</a>
  </div>
</body>
</html>
```

Now you can start your Spring Boot application and access the homepage in browser. It will look like this:



When the user clicks on the hyperlink, we will display the registration form.

### 3. Code Java Model Class

To represent the information in the user registration form, create the `User` class with the following fields:

```
package net.codejava;

import java.sql.Date;

public class User {
    private String name;
    private String email;
    private String password;
    private String gender;
    private String note;
    private boolean married;
    private Date birthday;
    private String profession;

    // getters
    // setters

    // override toString()
}
```

This is a simple domain model class. Note that the getter and setter methods are not shown for brevity. Also you should override the `toString()` method to return all details of the user.

### 4. Code Spring MVC Controller Class

Next, create a Spring MVC controller class with the first handler method that handles the requests to show the form:

```
package net.codejava;

import java.util.*;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
```

```
public class MainController {

    @GetMapping("/register")
    public String showForm(Model model) {
        User user = new User();
        model.addAttribute("user", user);

        List<String> listProfession = Arrays.asList("Developer", "Teste
r", "Architect");
        model.addAttribute("listProfession", listProfession);

        return "register_form";
    }
}
```

As you can see, the `showForm()` method handles the request `/register` when the user clicks the hyperlink in the homepage. It creates and puts two objects to the model:

- A `User` object to represent the information in the form.
- a `List` of profession that will be used to display a select box/dropdown list in the form.

Then it returns view name `register_form` which will be resolved to a Thymeleaf template file.

## 5. Code Thymeleaf Form

Next, create the `register_form.html` file under `src/main/resources/templates` directory. To use Thymeleaf syntax, you should declare this XML namespace at the beginning of the document:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
```

Use the HTML form tag mixed with Thymeleaf attributes as below:

```
<form action="#" th:action="@{/register}" method="post" th:object="${user}">
```

As you can see, Thymeleaf uses `th:xxx` attributes inside HTML tags so it doesn't break the structure of the page – unlike JSP and JSTL. So a Thymeleaf template file still looks like HTML in browser – very convenient for both designer and programmer.

In Thymeleaf, hyperlink is wrapped inside `@{ }` and access a model object inside `${ }`. Note that in this form tag, the `th:object` attribute points to the name of the

model object sent from Spring MVC controller.

Then for each form field, use the following code:

```
<label>Full name:</label>
<input type="text" th:field="*{name}" /><br/>
```

The `th:field` attribute points to the field name of the object in the model. Field name is wrapped inside `*{}`.

Below is the whole code of the user registration form:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>User Registration</title>
</head>
<body>
<div align="center">
    <h1>User Registration</h1>
    <form action="#" th:action="@{/register}" method="post" th:object="${user}">

        <label>Full name:</label>
        <input type="text" th:field="*{name}" /><br/>

        <label>E-mail:</label>
        <input type="text" th:field="*{email}" /><br/>

        <label>Password:</label>
        <input type="password" th:field="*{password}" /><br/>

        <label>Birthday (yyyy-mm-dd):</label>
        <input type="text" th:field="*{birthday}" /><br/>

        <label>Gender:</label>
        <input type="radio" th:field="*{gender}" value="Male" />Male
        <input type="radio" th:field="*{gender}" value="Female" />Female
    <br/>

        <label>Profession:</label>
        <select th:field="*{profession}">
            <option th:each="p : ${listProfession}" th:value="${p}"
th:text="${p}" />
        </select>
        <br/>

        <label>Married?</label>
        <input type="checkbox" th:field="*{married}" /><br/>

        <label>Note:</label>
        <textarea rows="5" cols="25" th:field="*{note}"></textarea>
```

```
        <br/>

        <button type="submit">Register</button>
    </form>
</div>
</body>
</html>
```

Note that the code for the HTML select/option tags to display a dropdown list, with values taken from a model object sent from the controller:

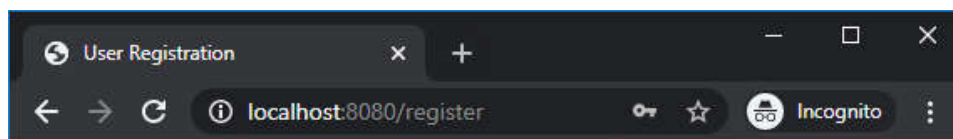
```
<label>Profession:</label>
<select th:field="**{profession}">
    <option th:each="p : ${listProfession}" th:value="${p}" th:text="${p}"
/>
</select>
```

Here, we use the `th:each` attribute to iterate over elements in a collection; `th:value` attribute to set value and `th:text` to set text for the HTML option tag.

And to format the form, put the following CSS code inside the head section of the page:

```
<style type="text/css">
    label {
        display: inline-block;
        width: 200px;
        margin: 5px;
        text-align: left;
    }
    input[type=text], input[type=password], select {
        display: inline-block;
        width: 200px;
    }
    input[type=radio] {
        margin-left: 45px;
    }
    input[type=checkbox] {
        margin-right: 190px;
    }
    button {
        padding: 5px;
        margin: 10px;
    }
</style>
```

When running, the form looks neat like this:





## User Registration

Full name:

E-mail:

Password:

Birthday (yyyy-mm-dd):

Gender: ☐ Male ☐ Female

Profession:

Married? ☐

Note:

## 6. Handle Form Submission

When the user clicks on Register button on the form, an HTTP POST request will be sent to the relative URL `/register`, so we need to code the handler method in the controller class as follows:

```
@PostMapping("/register")
public String submitForm(@ModelAttribute("user") User user) {
    System.out.println(user);
    return "register_success";
}
```

You see, handling form submission with Spring MVC is very simple and easy – just use the `@ModelAttribute` annotation – then it will read form data and set the values to fields of the model object. So in the method we can print the `user` object. No need to write code to get form data.

And this handler method returns a view name `register_success` which resolves to a Thymeleaf template file as described in the next section.

## 7. Code Thymeleaf Result Page

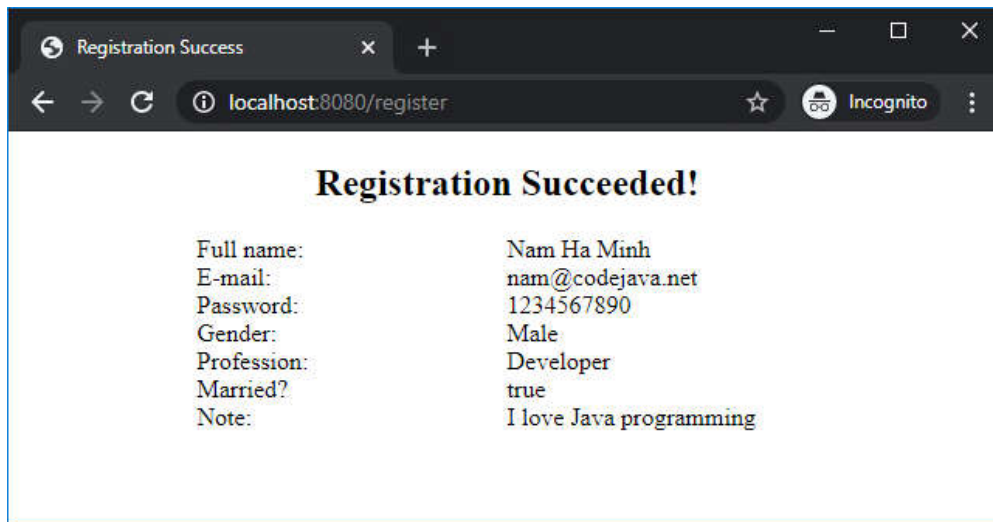
Create the `register_success.html` file with the following code:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
```



```
<meta charset="ISO-8859-1">
<title>Registration Success</title>
<style type="text/css">
    span {
        display: inline-block;
        width: 200px;
        text-align: left;
    }
</style>
</head>
<body>
<div align="center">
    <h2>Registration Succeeded!</h2>
    <span>Full name:</span><span th:text="${user.name}"></span><br/>
    <span>E-mail:</span><span th:text="${user.email}"></span><br/>
    <span>Password:</span><span th:text="${user.password}"></span><br/>
    <span>Gender:</span><span th:text="${user.gender}"></span><br/>
    <span>Profession:</span><span th:text="${user.profession}"></span><br/>
    <span>Married?</span><span th:text="${user.married}"></span><br/>
    <span>Note:</span><span th:text="${user.note}"></span><br/>
</div>
</body>
</html>
```

In this page, we use Thymeleaf attribute `th:text` to set text for the HTML span element. The user object is still available in the model so we can access it. When running, it displays the details of the user – which is the information submitted by the user in the form:



That's the tutorial for form handling with Spring Boot and Thymeleaf. You can download the sample project in the attachments section below.

For form validation, read my [Spring Boot Form Validation Tutorial](#).

For visual howtos, I recommend you to watch the video version of this tutorial below: