**1. Writing Comparator using Lambda Expression**

Since Comparator and Comparable are also SAM interfaces e.g. they contain just one abstract method like compare() and compareTo(), you can easily implement them using a lambda expression.

if you want to write a Comparator to sort Books by their author, you can write like in the following example:

Comparator<Book> byAuthor = (b1, b2) -> b1.getAuthor().compareTo(b2.getAuthor());

In Java 8, you can write Comparator using [lambda expression](https://javarevisited.blogspot.com/2014/02/10-example-of-lambda-expressions-in-java8.html#axzz5b2nmYJFN) in Just one line. Now, you can sort the list of books by using this comparator either by using Collections.sort() method or newly added List.sort() method, which sorts the list in place, as shown below:

listOfBooks.sort(byAuthor);

System.out.println("list of books after sorting: " + listOfBooks);

output:

list of books after sorting: [

Book [title=Java Concurrency in Practice, author=Brian Goetz, price=42],

Book [title=Java SE 8 **for** Really Impatient, author=Cay S. Horstmann, price=34],

Book [title=Core Java, author=Cay S. Horstmann, price=32],

Book [title=Effective Java, author=Joshua Bloch, price=32],

Book [title=Java Puzzlers, author=Joshua Bloch, price=22]

]

You can see that Brian Goetz book top the list because "B" comes first in lexicographic order, followed by Cay. S. Horstmann book and finally Josh Bloch's books.

**2. Writing Comparator using Method Reference**

When you are simply accessing a property of an object in lambda expression then you can replace the reference to lambdas by method reference. This result in more cleaner and readable code than you ever have seen in Java.

# Spring Boot + Spring MVC + Spring Security + MySQL with Validations:

<https://medium.com/@gustavo.ponce.ch/spring-boot-spring-mvc-spring-security-mysql-a5d8545d837d>

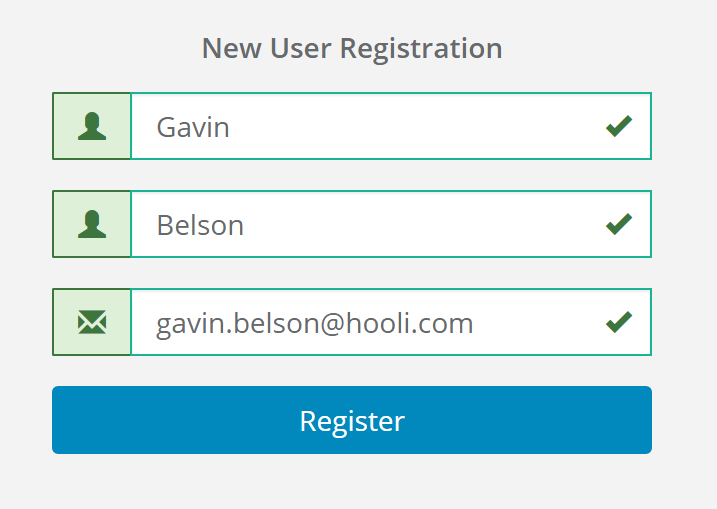
For GIthub code:

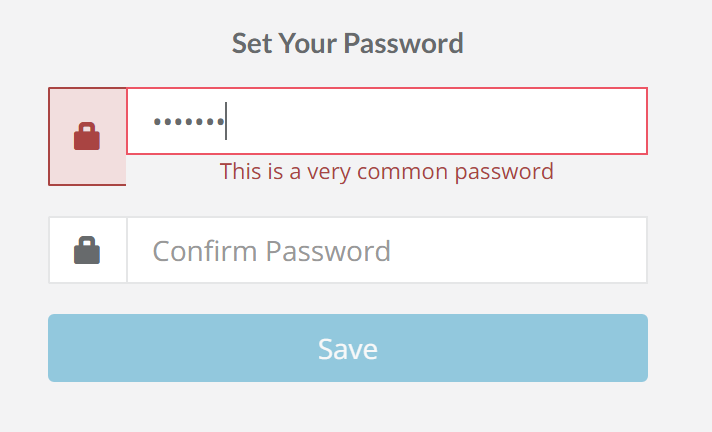
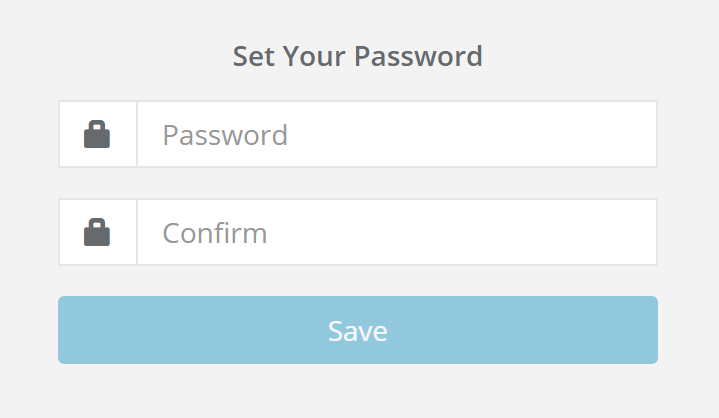
<https://github.com/gustavoponce7/spring-login?source=post_page--------------------------->

# [User Account Registration features with Spring Boot](https://www.codebyamir.com/blog/user-account-registration-with-spring-boot):

Reset Password and Email sent features:

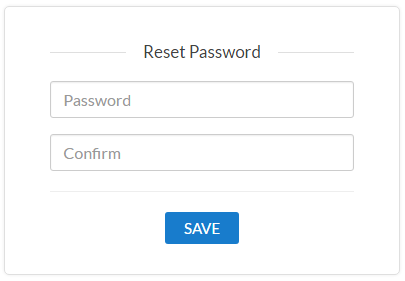
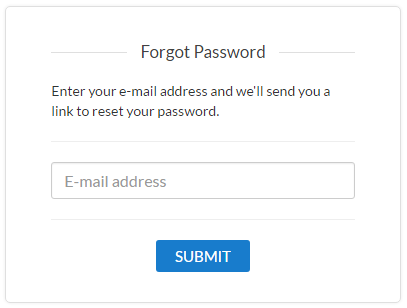
<https://www.codebyamir.com/blog/user-account-registration-with-spring-boot>





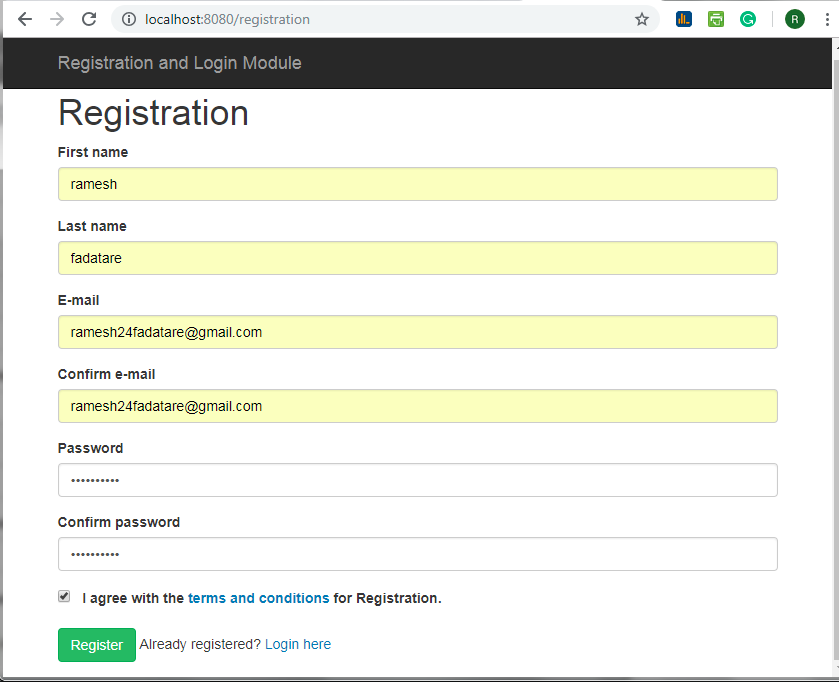
# [Forgot Password feature with Java and Spring Boot](https://www.codebyamir.com/blog/forgot-password-feature-with-java-and-spring-boot):

<https://www.codebyamir.com/blog/forgot-password-feature-with-java-and-spring-boot>

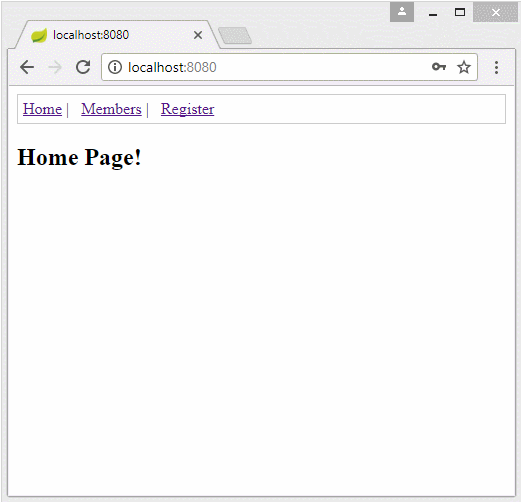


Registration form with Spring Boot+Spring Security+ Hibernate+Thymeleaf:

<https://www.javaguides.net/2018/10/user-registration-module-using-springboot-springmvc-springsecurity-hibernate5-thymeleaf-mysql.html>



Develop the Application like screenshot:



E-Mail sent:

<https://o7planning.org/en/11145/spring-email-tutorial>

generate and verify the recaptcha using google recaptcha:

<https://dzone.com/articles/using-google-recaptcha-with-spring-boot-applicatio>

Using Angular:

<https://www.devglan.com/angular/spring-boot-angular-captcha>

Spring Boot 2 RestAPI Controller:

* In Spring, a controller class, which is capable of serving REST API requests, is called rest controller. It should be annotated with **@RestController** annotation.
* The resource uris are specified in **@RequestMapping** annotations. It can be applied at class level and method level both. Complete URI for an API is resolved after adding class level path and method level path.
* We should always write **produces** and **consumes** attributes to specify the mediatype attributes for the API. Never reply on assumptions.