Spring Security 5 - JDBC based authentication example

Posted on December 17, 2017

In JDBC based authentication user's authentication and authorization information are stored in database.

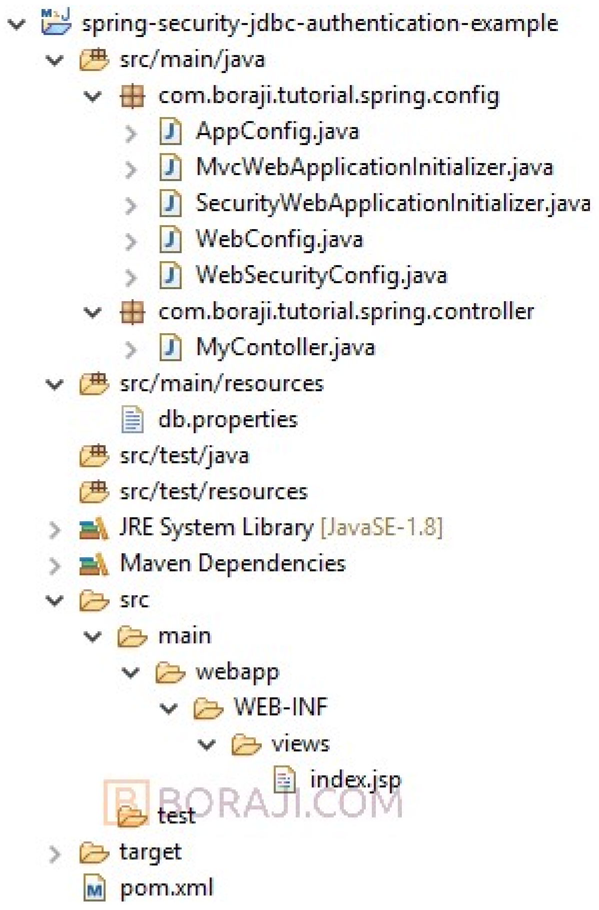
This post shows you how to secure a Spring MVC application with Spring Security -JDBC authentication.

Tools and technologies used for this application are -

- Spring Security 5.0.0.RELEASE
- Spring MVC 5.0.2.RELEASE
- Spring JDBC 5.0.2.RELEASE
- Servlet API 3.1.0
- Common Pool 2.1.1
- Java SE 1.8
- Maven 3.5.2
- Oxygen.1a Release (4.7.1a)
- Jetty Maven plugin 9.4.8
- MySQL Server 5.7

Project structure

Final project structure of our application will look like as follows.


```
spring-security-jdbc-authentication-example
  src/main/java
    com.boraji.tutorial.spring.config
      AppConfig.java
      MvcWebApplicationInitializer.java
      SecurityWebApplicationInitializer.java
      WebConfig.java
      WebSecurityConfig.java
    com.boraji.tutorial.spring.controller
      MyContoller.java
  src/main/resources
    db.properties
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
    main
      webapp
        WEB-INF
          views
            index.jsp
    test
  target
  pom.xml
```

Related - How to create a web project using maven build tool in eclipse IDE (https://www.boraji.com/how-to-create-a-web-project-using-maven-in-eclipse).

Jar dependencies

Open `pom.xml` file of your maven project and add the following dependencies in it.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0 (http://maven.apac
he.org/POM/4.0.0)" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce (http://www.w3.org/2001/XMLSchema-instance)"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 (http://mave
n.apache.org/POM/4.0.0) http://maven.apache.org/xsd/maven-4.0.0.xsd
(http://maven.apache.org/xsd/maven-4.0.0.xsd)">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.boraji.tutorial.springsecurity</groupId>
  <artifactId>spring-security-jdbc-authentication-example</artifactI
d>
  <version>0.0.1-SNAPSHOT</version>
  <name>Spring Security JDBC Authentication Example</name>
  <packaging>war</packaging>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <failOnMissingWebXml>false</failOnMissingWebXml>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-web</artifactId>
      <version>5.0.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-config</artifactId>
      <version>5.0.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.2.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.0.2.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp.jstl</groupId>
      <artifactId>javax.servlet.jsp.jstl-api</artifactId>
      <version>1.2.1</version>
    </dependency>
    <dependency>
      <groupId>taglibs</groupId>
      <artifactId>standard</artifactId>
      <version>1.1.2</version>
    </dependency>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-dbcp2</artifactId>
      <version>2.1.1</version>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>6.0.6</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
    <!-- Maven jetty plugin for testing war -->
      <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>9.4.8.v20171121</version>
      </plugin>
    </plugins>
  </build>
</project>
```

DataSource configuration

For JDBC based authentication, first you need to configure the datasource in your application. In this example, we will use the Apache Common DBCP (https://commons.apache.org/proper/commons-dbcp/) library for datasource configuration and MySQL server as a database.

First, create database schema for storing the user's authentication and authorization information. You can use the following DDL statements for MySQL database.

```
create table users(
        username varchar(50) not null primary key,
        password varchar(100) not null,
        enabled boolean not null
);
create table authorities (
        username varchar(50) not null,
        authority varchar(50) not null,
        constraint fk_authorities_users foreign key(username) refere
nces users(username)
);
create unique index ix_auth_username on authorities (username,author
ity);
```

Insert some data into `users` and `authorities` tables.

```
insert into users(username,password,enabled)
        values('admin','$2a$10$hbxecwitQQ.dDT4JOFzQAulNySFwEpaFLw38j
da6Td.Y/cOiRzDFu',true);
insert into authorities(username,authority)
        values('admin','ROLE_ADMIN');
```

Before inserting data into tables, you can encrypt the password using the

`BCryptPasswordEncoder`.

```
String encoded=new BCryptPasswordEncoder().encode("admin@123");
System.out.println(encoded);
```

Next, create a properties file under `src/main/resources` folder and define the database connection properties as follows.

db.properties

```
mysql.driver=com.mysql.cj.jdbc.Driver
mysql.jdbcUrl=jdbc:mysql://localhost:3306/BORAJI?useSSL=false
mysql.username=root
mysql.password=admin
```

Next, create a `@Configuration` class and define the `@Bean` method for `DataSource` as follows.

AppConfig.java

```
package com.boraji.tutorial.spring.config;

import javax.sql.DataSource;

import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;

@Configuration
@PropertySource("classpath:db.properties")
public class AppConfig {

  @Autowired
  private Environment env;

  @Bean
  public DataSource getDataSource() {
    BasicDataSource dataSource = new BasicDataSource();
    dataSource.setDriverClassName(env.getProperty("mysql.driver"));
    dataSource.setUrl(env.getProperty("mysql.jdbcUrl"));
    dataSource.setUsername(env.getProperty("mysql.username"));
    dataSource.setPassword(env.getProperty("mysql.password"));
    return dataSource;
  }
}
```

Spring Security configuration

To configure Spring Security in Spring MVC application you need to -

- Create a `springSecurityFilterChain` Servlet Filter for protecting and validating all URLs by create a `@Configuration` class.
- Register the `springSecurityFilterChain` filter with war.

Now, create a `@Configuration` class by extending the `WebSecurityConfigurerAdapter` class and annotate it with `@EnableWebSecurity` as follows.

WebSecurityConfig.java

```
package com.boraji.tutorial.spring.config;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.config.annotation.authenticatio
n.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.H
ttpSecurity;
import org.springframework.security.config.annotation.web.configurat
ion.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurat
ion.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEnco
der;

@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter
{

  @Autowired
  private DataSource dataSource;

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throw
s Exception {

    auth.jdbcAuthentication().dataSource(dataSource)
        .usersByUsernameQuery("select username, password, enabled"
            + " from users where username=?")
        .authoritiesByUsernameQuery("select username, authority "
            + "from authorities where username=?")
        .passwordEncoder(new BCryptPasswordEncoder());
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {

    http.authorizeRequests().anyRequest().hasAnyRole("ADMIN", "USE
R")
    .and()
    .httpBasic(); // Authenticate users with HTTP basic authenticati
on
  }
}
```

Next, create `SecurityWebApplicationInitializer` class by extending the `AbstractSecurityWebApplicationInitializer` to register the `springSecurityFilterChain` filter.

SecurityWebApplicationInitializer.java

```
package com.boraji.tutorial.spring.config;

import org.springframework.security.web.context.AbstractSecurityWebA
pplicationInitializer;

public class SecurityWebApplicationInitializer
  extends AbstractSecurityWebApplicationInitializer {

}
```

Spring MVC configuration

To enable the Spring MVC in your application, you need to annotate your `@Configuration` class with `@EnableWebMvc` annotation.

In this example, we are using the JSP views. So create a `@Configuration` class and register the JSP view resolver as follows.

WebConfig.java

```
package com.boraji.tutorial.spring.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMv
c;
import org.springframework.web.servlet.config.annotation.ViewResolve
rRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfi
gurer;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages= {"com.boraji.tutorial.spring.controlle
r"})
public class WebConfig implements WebMvcConfigurer {
        @Override
        public void configureViewResolvers(ViewResolverRegistry regi
stry) {
                registry.jsp().prefix("/WEB-INF/views/").suffix(".js
p");
        }
}
```

Servlet container Initialization and configuration

In Spring MVC, The `DispatcherServlet` needs to be declared and mapped for processing all requests either using java or `web.xml` configuration.

In a Servlet 3.0+ environment, you can use `AbstractAnnotationConfigDispatcherServletInitializer` class to register and initialize the `DispatcherServlet` programmatically as follows.

MvcWebApplicationInitializer.java

```
package com.boraji.tutorial.spring.config;

import org.springframework.web.servlet.support.AbstractAnnotationCon
figDispatcherServletInitializer;

public class MvcWebApplicationInitializer
      extends AbstractAnnotationConfigDispatcherServletInitializer {

  // Load database and spring security configurations
  @Override
  protected Class<?>[] getRootConfigClasses() {
    return new Class[] { AppConfig.class, WebSecurityConfig.class };
  }

  // Load spring web configuration
  @Override
  protected Class<?>[] getServletConfigClasses() {
    return new Class[] { WebConfig.class };
  }

  @Override
  protected String[] getServletMappings() {
    return new String[] { "/" };
  }

}
```

Controller class

Create a simple `@Controller` class
under `com.boraji.tutorial.spring.controller` package as follows.

MyContoller.java

```
package com.boraji.tutorial.spring.controller;

import java.security.Principal;

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

@Controller
public class MyContoller {

  @GetMapping("/")
  public String index(Model model, Principal principal) {
    model.addAttribute("message", "You are logged in as " + principa
l.getName());
    return "index";
  }
}
```

JSP views

Create an `index.jsp` file under `src\main\webapp\WEB-INF\views` folder and write the following code in it.

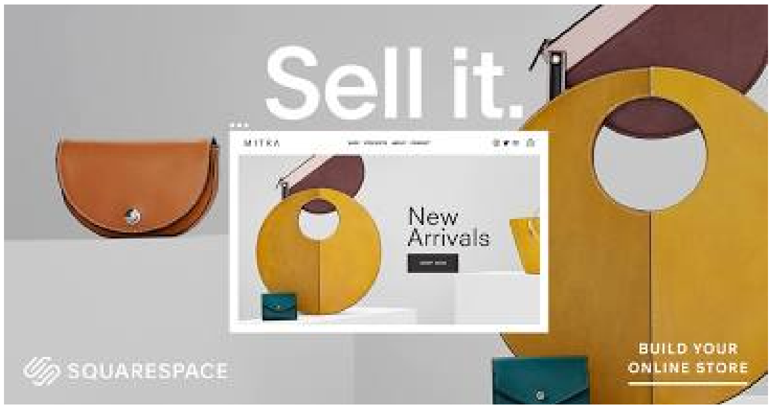
index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859
-1">
<title>Spring Security JDBC Authentication Example</title>
</head>
<body>
<h1>Spring Security JDBC Authentication Example</h1>
<h2>${message}</h2>
</body>
</html>
```

Run application

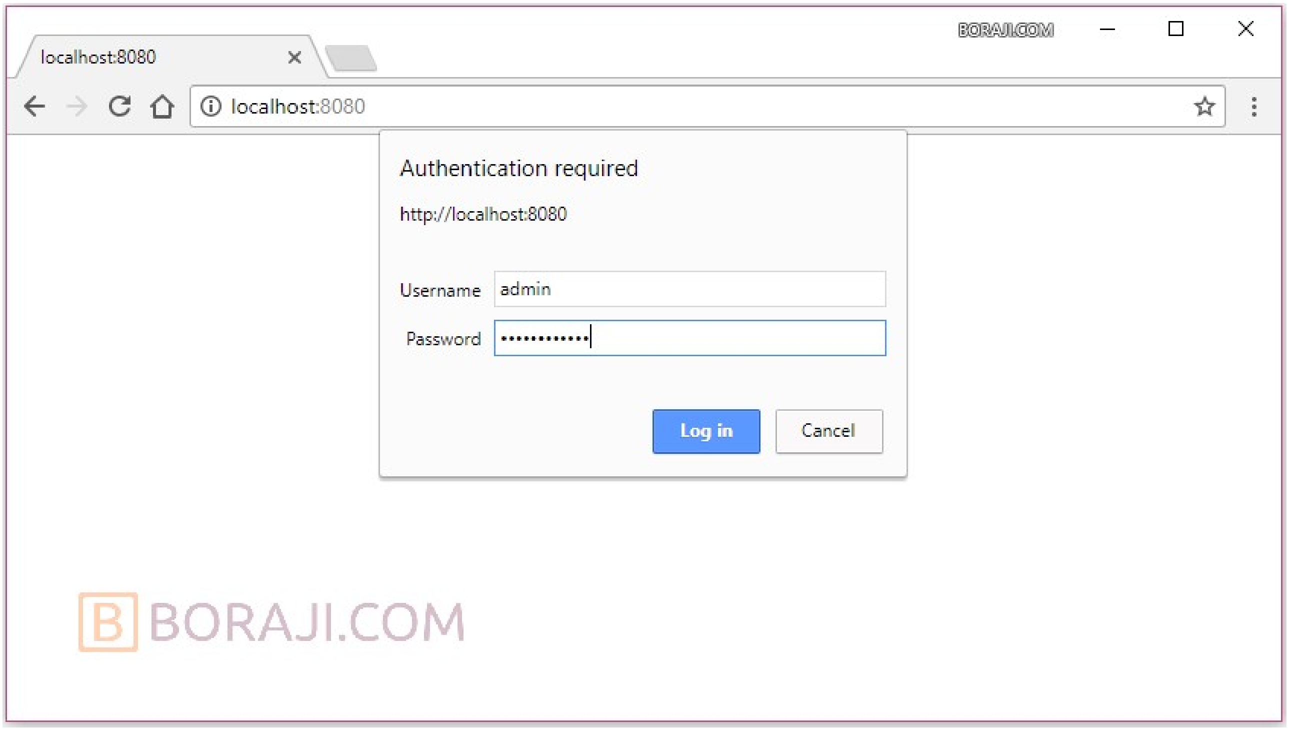
Use the following maven command to run your application.

`mvn jetty:run` (This command deploy the webapp from its sources, instead of build war).



Enter the **http://localhost:8080/** URL in browser's address bar to test our application.

On entering the URL, you will see the dialog box asking for username and password as follows.



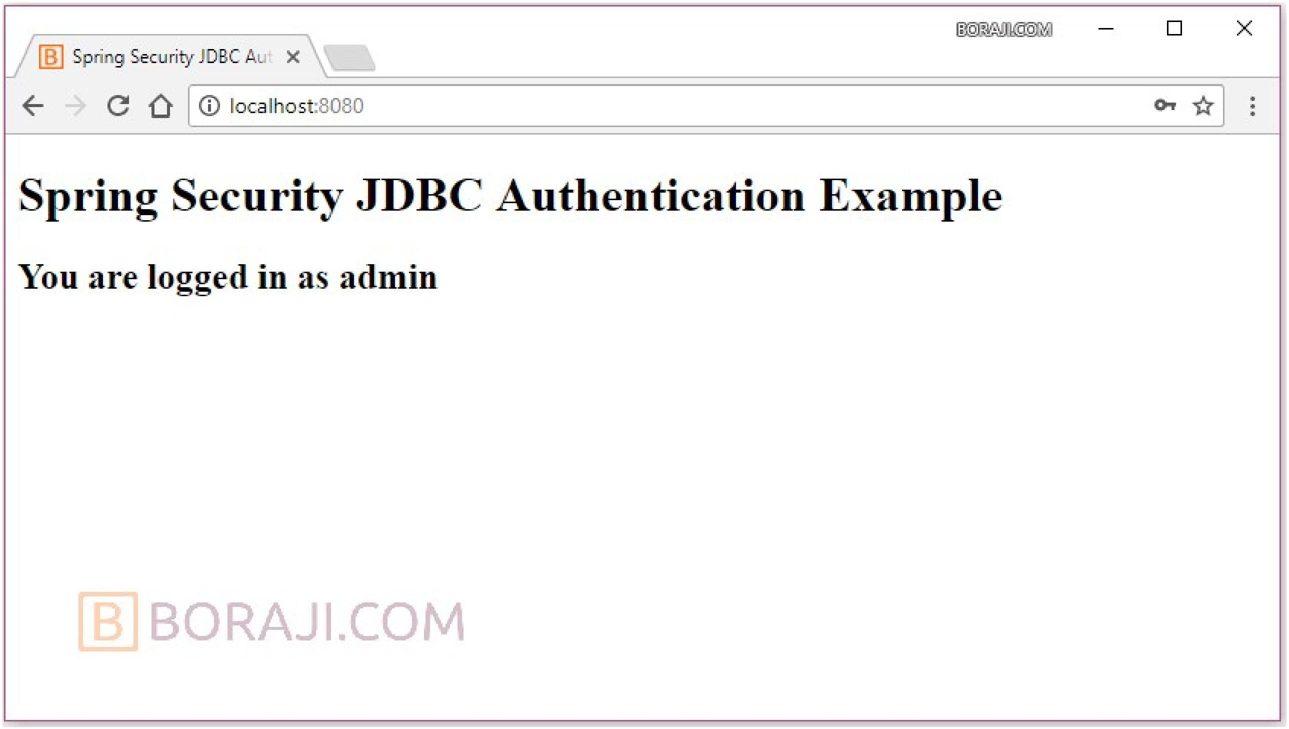
Authentication required

http://localhost:8080

Username admin

Password ••••••••••••

On successful login, you will see the index page as follows.



Spring Security JDBC Authentication Example

You are logged in as admin

Spring Security (/category/spring-security) **Spring MVC (/category/spring-mvc)**

Download Sources:

24.95 KB

spring-security-jdbc-authentication-example.zip (https://boraji.com/sites/default/files/2017-12/spring-security-jdbc-authentication-example.zip)

Related Posts

Spring MVC 5 + Spring Security 5 + Hibernate 5 example (/spring-mvc-5-spring-security-5-hibernate-5-example)

Spring Security 4 - Hello World example (/spring-security-4-hello-world-example)

Spring Security 4 - HTTP basic authentication example (/spring-security-4-http-basic-authentication-example)

Spring Security 4 - Custom login from example (/spring-security-4-custom-login-from-example)

Spring Security 5 - JDBC based authentication example (/spring-security-5-jdbc-based-authentication-example)

Spring Security 5 - Custom UserDetailsService example (/spring-security-5-custom-userdetailsservice-example)

Spring Security 5 - Remember-Me authentication example (/spring-security-5-remember-me-authentication-example)

Spring MVC 4 - Form validation example using Hibernate Validator (/spring-4-mvc-form-validation-example-using-hibernate-validator)

Spring MVC 4 - @RestController example (/spring-mvc-4-restcontroller-example)

Spring MVC 4 - JQuery Ajax form submit example (/spring-4-mvc-jquery-ajax-form-submit-example)

0 Comments BORAJI.COM Login

Recommend Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd

Search for...

Navigation

Spring Frameworks ▾ ()

Hibernate ORM (/category/hibernate-orm)

Log4j 2 (/category/log4j-2)

JFreeChart (/category/jfreechart)

Core Java ▾ ()

Misc ▾ ()

Facebook (https://www.facebook.com/imssbora/)

Recent Post

- Hibernate 5 - Named query example (/hibernate-5-named-query-example)
- Hibernate 5 - Native SQL query example (/hibernate-5-native-sql-query-example)
- Hibernate 5 - Batch processing example (/hibernate-5-batch-processing-example)
- Jackson API - Collection serialization and deserialization example (/jackson-api-collection-serialization-and-deserialization-example)
- Jackson API - Converting POJOs to JSON example (/jackson-api-converting-pojos-to-json-example)
- Jackson API - Streaming parser and generator example (/jackson-api-streaming-parser-and-generator-example)
- Spring MVC 5 - Handling WebSocket message example (/spring-mvc-5-handling-websocket-message-example)
- Spring MVC 5 - Internationalization example (/spring-mvc-5-internationalization-example)
- Spring MVC 5 - Theme Resolver example (/spring-mvc-5-theme-resolver-example)
- Spring MVC 5 - Static resources handling example (/spring-mvc-5-static-resources-handling-example)

Reference links

- Java 9 API Specification (https://docs.oracle.com/javase/9/docs/api/overview-summary.html)
- Java 8 API Specification (https://docs.oracle.com/javase/8/docs/api/)
- Java 7 API Specification (https://docs.oracle.com/javase/7/docs/api/)
- Java Tutorial (https://docs.oracle.com/javase/tutorial/)
- Spring Framework API (https://docs.spring.io/spring/docs/current/javadoc-api/)
- Spring Framework Reference Documentation (https://docs.spring.io/spring/docs/current/spring-framework-reference/)
- Spring Boot API (https://docs.spring.io/spring-boot/docs/current/api/)
- Spring Boot Reference Documentation (https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/)
- Spring Security API (https://docs.spring.io/spring-security/site/docs/current/api/)
- Spring Security Reference Documentation (https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/)
- Hibernate JavaDoc (http://docs.jboss.org/hibernate/orm/current/javadocs/)
- Hibernate User Guide (http://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User

About me

Sunil Singh Bora, founder of BORAJI.COM (https://www.boraji.com), loves Java programming and open source technologies. The vision of this website is to teach java programming and java web technologies with short and simple examples. If you liked tutorials and examples on this website, please follow me on

(http://www.facebook.com/imssbora)

(http://twitter.com/intent/follow?source=followbutton&variant=1.0&screen_name=imssbora)