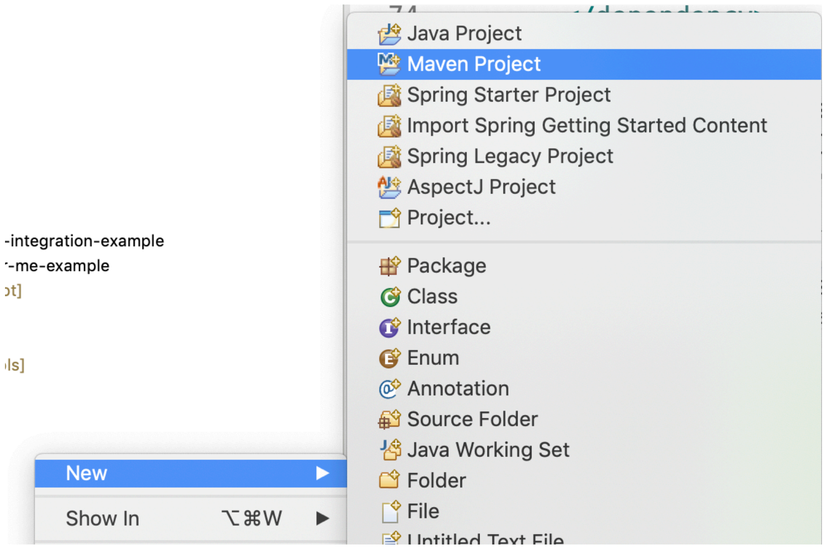
Spring MVC 5 + Spring Security 5 + Hibernate 5 with Custom Login Form Example

- How to create a custom login form in Spring MVC application with Spring Security.
- How to integrate the Hibernate with Spring security framework to load the user's authentication.
- How to use the UserDetailsService interface to load the user's authentication information from a database.

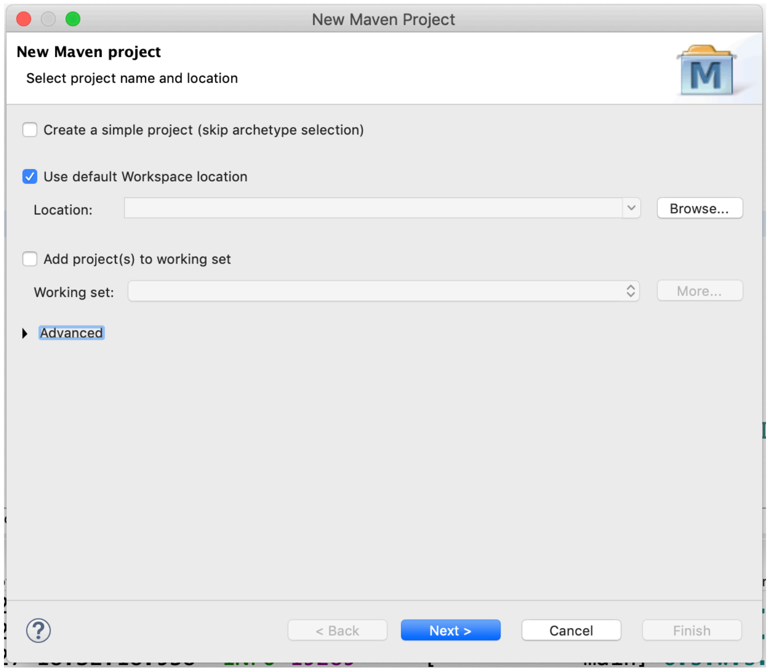
Tools and technologies used for this application are –

- Spring Security 5.0.0.RELEASE
- Spring MVC 5.0.2.RELEASE
- Spring ORM 5.0.2.RELEASE
- Hibernate 5.2.12.Final
- C3p0 0.9.5.2
- Servlet API 3.1.0
- Common Pool 2.1.1
- Java SE 8
- Maven 3.2
- Eclipse STS
- Tomcat 8.x / 9.x
- MySQL Server 8.x / Oracle 11+

1. Create a Maven Arch type project as shown below:



2. Click on Next


New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☑ Use default Workspace location

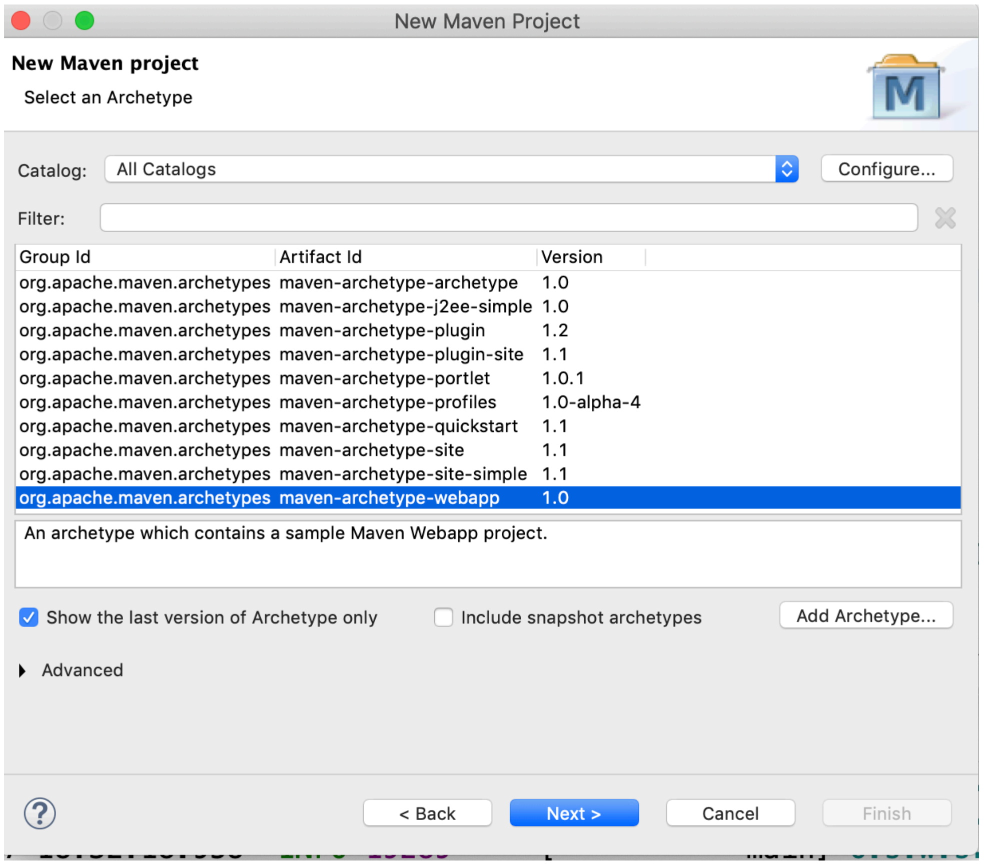
Location:

☐ Add project(s) to working set

Working set:

Advanced

3. Select archtype-webapp as shown, click on Next



Catalog: All Catalogs

Filter:

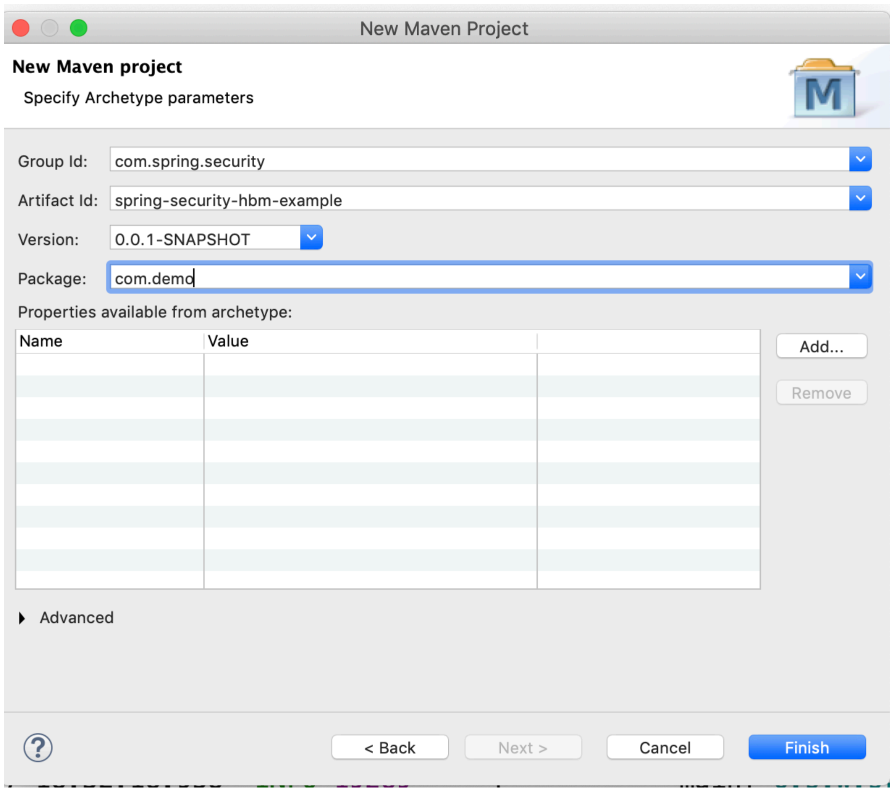
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven Webapp project.

[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes

4. Specify Group ID, Artifact ID & package name

& Click on Finish


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.spring.security

Artifact Id: spring-security-hbm-example

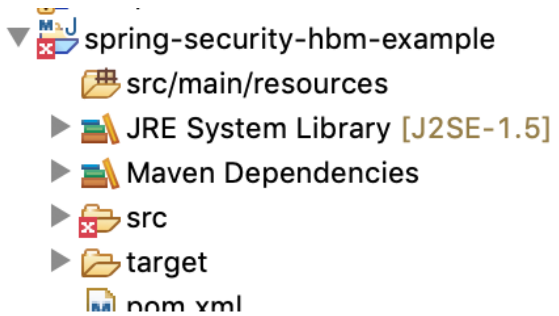
Version: 0.0.1-SNAPSHOT

Package: com.demo

Properties available from archetype:

Name	Value

5. If there is a following error as shown for JDK version, then you need to add <build> tag with JDK 8 in ***pom.xml***


spring-security-hbm-example

src/main/resources

JRE System Library [J2SE-1.5]

Maven Dependencies

src

target

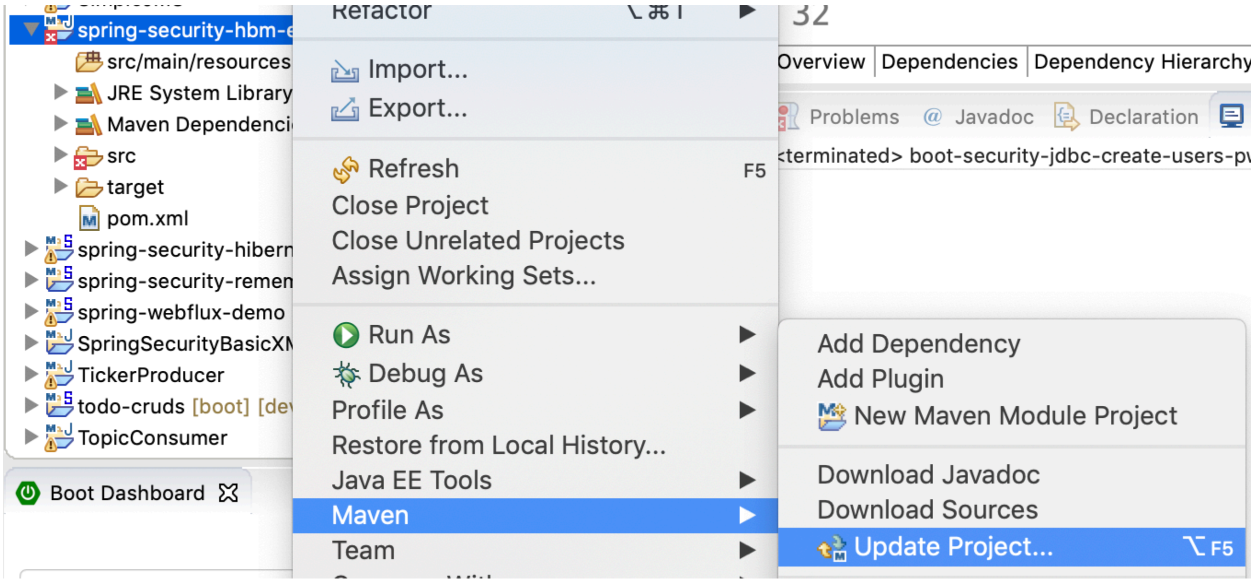
pom.xml

6. Add JDK version with <build> tag in ***pom.xml***:

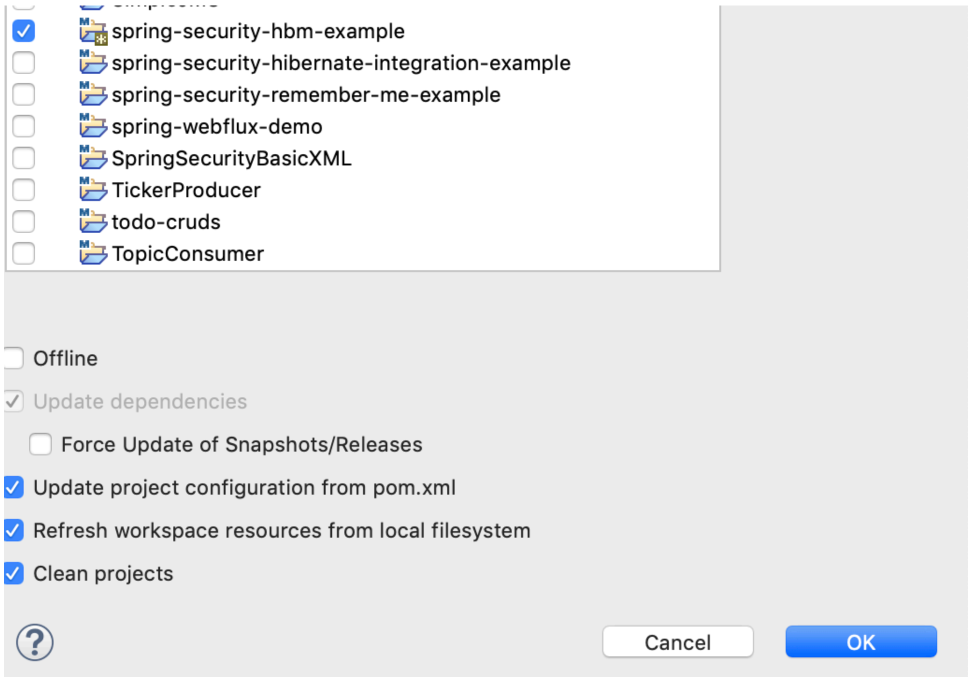
You can replace the existing <build> tag

```
<build>
     <plugins>
          <plugin>
               <groupId>org.apache.maven.plugins</groupId>
               <artifactId>maven-compiler-plugin</
artifactId>
               <version>3.2</version>
               <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
               </configuration>
          </plugin>
     </plugins>
</build>
```

7. Now you need to Update Project for Maven rebuild



8. A dialogue box appears, just click on **OK** button

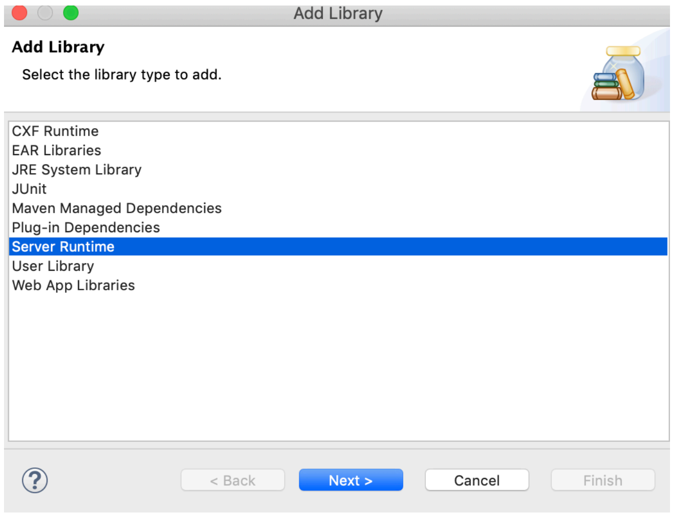


[✓] spring-security-hbm-example
[ ] spring-security-hibernate-integration-example
[ ] spring-security-remember-me-example
[ ] spring-webflux-demo
[ ] SpringSecurityBasicXML
[ ] TickerProducer
[ ] todo-cruds
[ ] TopicConsumer

[ ] Offline
[✓] Update dependencies
[ ] Force Update of Snapshots/Releases
[✓] Update project configuration from pom.xml
[✓] Refresh workspace resources from local filesystem
[✓] Clean projects

9. Error may still remain. So need to add Tomcat server now

```
Right Click –> Project Name
  –> Properties
     –> Java Build Path
          –> Libraries Tab –> Click on Add Library…
```



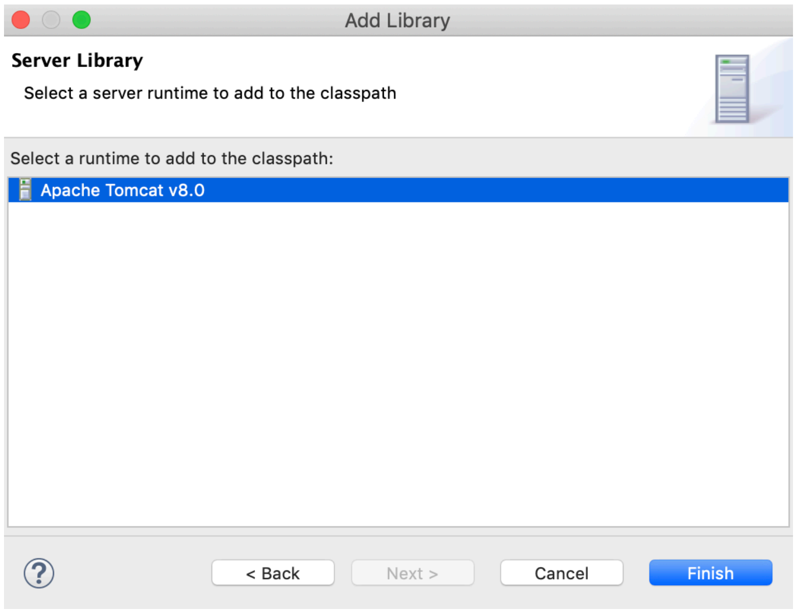
Add Library

Add Library

Select the library type to add.

CXF Runtime
EAR Libraries
JRE System Library
JUnit
Maven Managed Dependencies
Plug-in Dependencies
Server Runtime
User Library
Web App Libraries

10. Select your workspace Tomcat which is visible in this screen


Add Library

Server Library

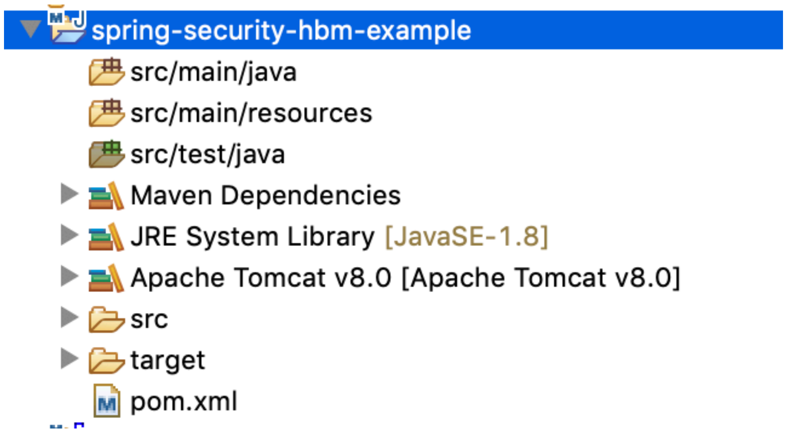
Select a server runtime to add to the classpath

Select a runtime to add to the classpath:

Apache Tomcat v8.0

11. Click on **Apply and Close** button

12. Error must be removed now and you should also see Tomcat as shown below


spring-security-hbm-example

src/main/java

src/main/resources

src/test/java

Maven Dependencies

JRE System Library [JavaSE-1.8]

Apache Tomcat v8.0 [Apache Tomcat v8.0]

src

target

pom.xml

13. Add the following dependencies in ***pom.xml***

Not to forget to add the following property
above <dependencies> tag

```
<properties>
          <failOnMissingWebXml>false</failOnMissingWebXml>
     </properties>
```

Now, you can have other dependencies as shown:

```
<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-orm</artifactId>
          <version>5.0.2.RELEASE</version>
     </dependency>

     <dependency>
          <groupId>org.hibernate</groupId>
          <artifactId>hibernate-core</artifactId>
          <version>5.2.12.Final</version>
     </dependency>
     <dependency>
          <groupId>com.mchange</groupId>
          <artifactId>c3p0</artifactId>
          <version>0.9.5.2</version>
     </dependency>

     <dependency>
          <groupId>mysql</groupId>
          <artifactId>mysql-connector-java</artifactId>
          <version>8.0.15</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>javax.xml.bind</groupId>
          <artifactId>jaxb-api</artifactId>
          <version>2.3.0</version>
     </dependency>
</dependencies>
```

14. Create a Test Case class in **src/test/java**

This is basically to get Encrypted password value for a given string.

```
package com.demo.test;

import org.junit.jupiter.api.Test;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class BCryptPasswordEncoderTest {

	@Test
	public void testPasswordEncode() {
		String encoded=new
BCryptPasswordEncoder().encode("admin@123");
		System.out.println(encoded);
	}

}
```

15. Database table creation

Use the following DDL statements to create tables for user’s login information in 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) references
users(username)
);
create unique index ix_auth_username on authorities (username,authority);

#Insert login information into database tables.

insert into users(username,password,enabled)
      values('admin','$2a$10$hbxecwitQQ.dDT4JOFzQAulNySFwEpaFLw38jda6Td.Y/
cOiRzDFu',true);
insert into authorities(username,authority)
      values('admin','ROLE_ADMIN');

# REMINDER

#Before inserting data into tables, you can use the following code to encrypt the
password.

#String encoded=new BCryptPasswordEncoder().encode("admin@123");
```

16. Entity classes

Create two **@Entity** classes, named as **User** and **Authorities**, to map with database tables as follows.

User.java

```java
package com.demo.model;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "USERS")
public class User {
  @Id
  @Column(name = "USERNAME")
  private String username;

  @Column(name = "PASSWORD", nullable = false)
  private String password;

  @Column(name = "ENABLED", nullable = false)
  private boolean enabled;

  @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
  private Set<Authorities> authorities = new HashSet<>();

  public String getUsername() {
    return username;
  }

  public void setUsername(String username) {
    this.username = username;
  }

  public String getPassword() {
    return password;
  }
public void setPassword(String password) {
    this.password = password;
  }

  public boolean isEnabled() {
    return enabled;
  }

  public void setEnabled(boolean enabled) {
    this.enabled = enabled;
  }

  public Set<Authorities> getAuthorities() {
    return authorities;
  }

  public void setAuthorities(Set<Authorities> authorities)
{
    this.authorities = authorities;
  }
}
```

Authorities.java

```java
package com.demo.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "AUTHORITIES")
public class Authorities {
  @Id
  @Column(name = "AUTHORITY")
  private String authority;

  @ManyToOne
  @JoinColumn(name = "USERNAME")
  private User user;

  public String getAuthority() {
    return authority;
  }

  public void setAuthority(String authority) {
    this.authority = authority;
  }

  public User getUser() {
    return user;
  }

  public void setUser(User user) {
    this.user = user;
  }

}
```

11. Hibernate configuration

First, create a properties file under src/main/resources folder and define the database connection, hibernate and C3P0 properties as follows.

db.properties

```
# MySQL connection properties
mysql.driver=com.mysql.cj.jdbc.Driver
mysql.jdbcUrl=jdbc:mysql://localhost:3306/spring_security?
serverTimezone=UTC
mysql.username=root
mysql.password=password

# Hibernate properties
hibernate.show_sql=true
hibernate.hbm2ddl.auto=update

#C3P0 properties
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.acquire_increment=1
hibernate.c3p0.timeout=1800
hibernate.c3p0.max_statements=150
```

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

In Spring based application, LocalSessionFactoryBean class is used to create a Hibernate SessionFactory.

AppConfig.java

```
package com.demo.config;

import java.util.Properties;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ComponentScans;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import
org.springframework.transaction.annotation.EnableTransactionManagement;

import com.demo.model.Authorities;
import com.demo.model.User;

import static org.hibernate.cfg.Environment.*;

@Configuration
@PropertySource("classpath:db.properties")
@EnableTransactionManagement
@ComponentScans(value = { @ComponentScan("com.demo.dao"),
    @ComponentScan("com.demo.service") })
public class AppConfig {

  @Autowired
  private Environment env;

  @Bean
  public LocalSessionFactoryBean getSessionFactory() {
    LocalSessionFactoryBean factoryBean = new
LocalSessionFactoryBean();

    Properties props = new Properties();

    // Setting JDBC properties
    props.put(DRIVER, env.getProperty("mysql.driver"));
    props.put(URL, env.getProperty("mysql.jdbcUrl"));
    props.put(USER, env.getProperty("mysql.username"));
    props.put(PASS, env.getProperty("mysql.password"));

    // Setting Hibernate properties
    props.put(SHOW_SQL, env.getProperty("hibernate.show_sql"));
    props.put(HBM2DDL_AUTO, env.getProperty("hibernate.hbm2ddl.auto"));
// Setting C3P0 properties
    props.put(C3P0_MIN_SIZE,
env.getProperty("hibernate.c3p0.min_size"));
    props.put(C3P0_MAX_SIZE,
env.getProperty("hibernate.c3p0.max_size"));
    props.put(C3P0_ACQUIRE_INCREMENT,
env.getProperty("hibernate.c3p0.acquire_increment"));
    props.put(C3P0_TIMEOUT,
env.getProperty("hibernate.c3p0.timeout"));
    props.put(C3P0_MAX_STATEMENTS,
env.getProperty("hibernate.c3p0.max_statements"));

    factoryBean.setHibernateProperties(props);
    factoryBean.setAnnotatedClasses(User.class,
Authorities.class);

    return factoryBean;
  }

  @Bean
  public HibernateTransactionManager getTransactionManager() {
    HibernateTransactionManager transactionManager = new
HibernateTransactionManager();

transactionManager.setSessionFactory(getSessionFactory().getObj
ect());
    return transactionManager;
  }
}
```

13. Repository classes

Create @Repository classes under com.demo.dao package as follows.

UserDetailsDao.java

```java
package com.demo.dao;

import com.demo.model.User;

public interface UserDetailsDao {
  User findUserByUsername(String username);
}
```

UserDetailsDaoImp.java

```java
package com.demo.dao;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.demo.model.User;

@Repository
public class UserDetailsDaoImp implements UserDetailsDao {

  @Autowired
  private SessionFactory sessionFactory;

  @Override
  public User findUserByUsername(String username) {
    return sessionFactory.getCurrentSession().get(User.class,
username);
  }
}
```

14. UserDetailsService or Service class

To create a custom user service, you need to implement the **UserDetailsService** interface and override the **loadUserByUsername()** method.

Create **@Service** class under **com.demo.service** package as follows.

UserDetailsServiceImp.java

```java
package com.demo.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.User.UserBuilder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.demo.dao.UserDetailsDao;
import com.demo.model.User;

@Service("userDetailsService")
public class UserDetailsServiceImp implements UserDetailsService {

  @Autowired
  private UserDetailsDao userDetailsDao;

  @Transactional(readOnly = true)
  @Override
  public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

    User user = userDetailsDao.findUserByUsername(username);
    UserBuilder builder = null;
    if (user != null) {

      builder =
org.springframework.security.core.userdetails.User.withUsername(username);
      builder.disabled(!user.isEnabled());
      builder.password(user.getPassword());
      String[] authorities = user.getAuthorities()
          .stream().map(a -> a.getAuthority()).toArray(String[]::new);

      builder.authorities(authorities);
    } else {
      throw new UsernameNotFoundException("User not found.");
    }
    return builder.build();
  }
}
```

15. Spring Security configuration

To configure Spring Security in Spring MVC application you need to create a **@Configuration** class by extending the **WebSecurityConfigurerAdapter** class and annotate it with **@EnableWebSecurity** as follows.

WebSecurityConfig.java

```
package com.demo.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.authentication.builders.Authenticatio
nManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigur
erAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private UserDetailsService userDetailsService;

  @Bean
  public BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  };

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().anyRequest().hasAnyRole("ADMIN", "USER")
    .and()
    .authorizeRequests().antMatchers("/login**").permitAll()
    .and()
    .formLogin().loginPage("/login").loginProcessingUrl("/
loginAction").failureUrl("/login?error=true").permitAll()
    .and()
    .logout().logoutSuccessUrl("/login").permitAll()
    .and()
    .csrf().disable();
  }
}
```

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

SecurityWebApplicationInitializer.java

```
package com.demo.config;

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

public class SecurityWebApplicationInitializer
  extends AbstractSecurityWebApplicationInitializer {

}
```

17. Spring MVC configuration

In this example, we are using the JSP views. So create a **@Configuration** class and override the **configureViewResolvers()** method to register the JSP view resolver.

Also, you can override the **addViewControllers()** method to map and render the default login page generated by Spring Security.

WebConfig.java

```java
package com.demo.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.demo.controller" })
public class WebConfig implements WebMvcConfigurer {
  @Override
  public void configureViewResolvers(ViewResolverRegistry registry) {
    registry.jsp().prefix("/WEB-INF/views/").suffix(".jsp");
  }

  @Override
  public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/login").setViewName("login");
  }
}
```

18. 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.xmlconfiguration.**

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

MvcWebApplicationInitializer.java

```
package com.demo.config;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatche
rServletInitializer;

public class MvcWebApplicationInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

  // Load database and spring security configuration
  @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[] { "/" };
  }

}
```

19. Controller class

Create a simple @Controller class under com.demo.controller package as follows.

UserContoller.java

```java
package com.demo.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 UserContoller {

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

20. JSP views

Create **login.jsp and index.jsp** files under
 src\main\webapp\WEB-INF\views folder and write the following
     code in it.

login.jsp

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
     pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://www.springframework.org/tags"
prefix="spring"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Login</title>
</head>
<body>

     <h1>Spring MVC 5 + Spring Security 5 + Hibernate 5 example</
h1>
     <h4>Login Form</h4>

     <form action='<spring:url value="/loginAction"/>'
method="post">
    <table>
      <tr>
        <td>Username</td>
        <td>

        <input type="text" name="username"></td>
      </tr>
      <tr>
        <td>Password</td>
        <td>

        <input type="password" name="password"></td>
      </tr>
      <tr>
        <td><button type="submit">Login</button></td>
      </tr>
    </table>
  </form>
  <br/>
</body>
</html>
```

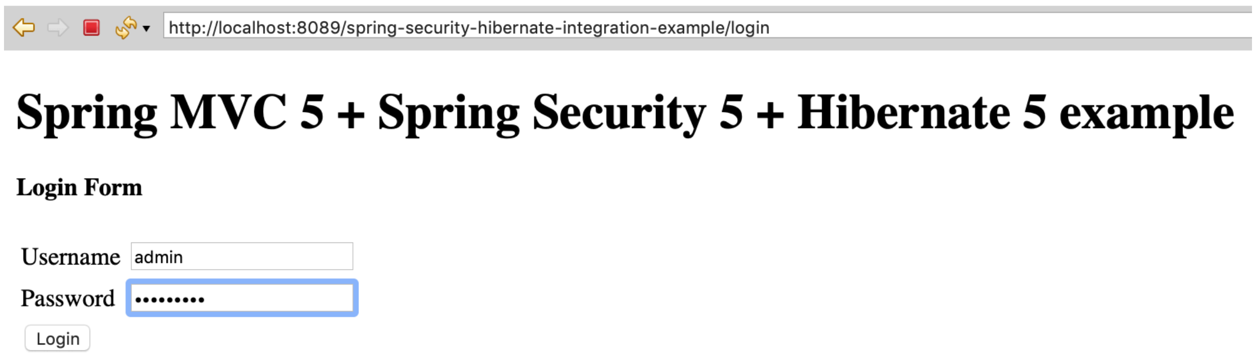
21. index.jsp

```
<%@ 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>Index</title>
</head>
<body>
     <h1>Spring MVC 5 + Spring Security 5 + Hibernate 5
example</h1>
     <h2>${message}</h2>

     <form action="logout" method="post">
          <input value="Logout" type="submit">
     </form>
</body>
</html>
```

22. OUTPUT

Visit
http://localhost:8089/spring-security-hibernate-integration-example

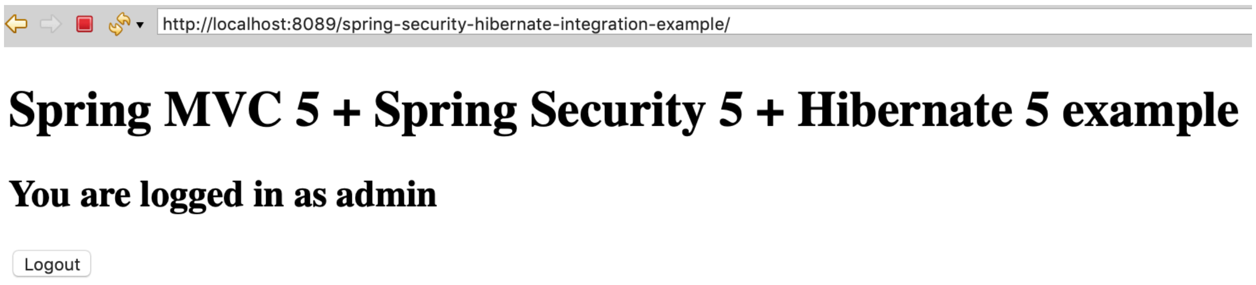

http://localhost:8089/spring-security-hibernate-integration-example/login

Spring MVC 5 + Spring Security 5 + Hibernate 5 example

Login Form

Username admin
Password •••••••••
Login

23. See the response as shown below:



http://localhost:8089/spring-security-hibernate-integration-example/

Spring MVC 5 + Spring Security 5 + Hibernate 5 example

You are logged in as admin

24. Click on Logout button