WebSystique
learn together

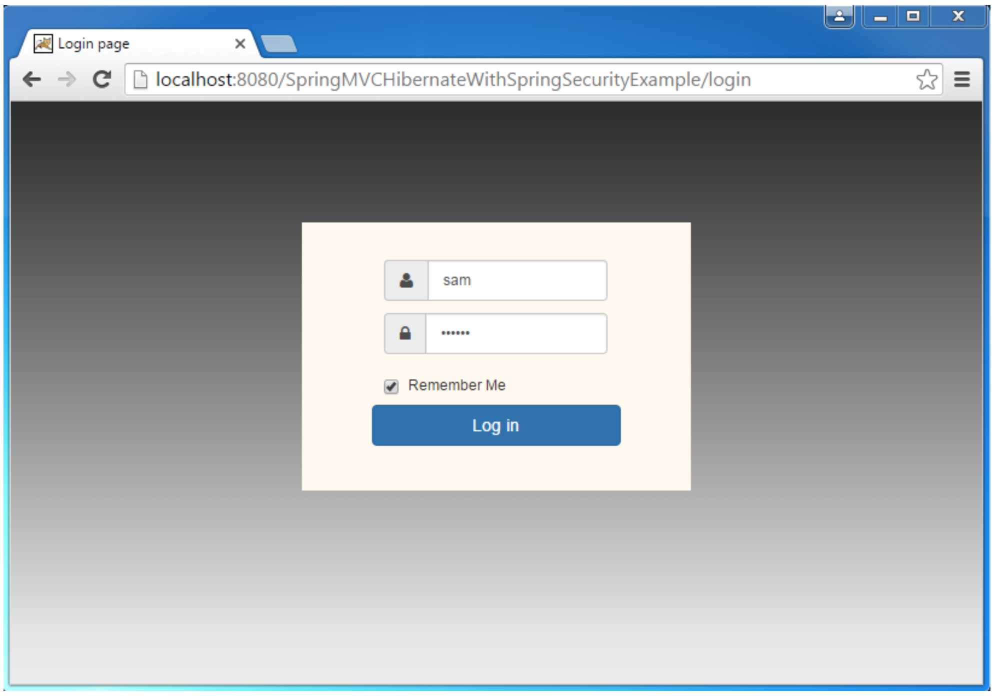
Spring MVC 4 + Spring Security 4 + Hibernate Example

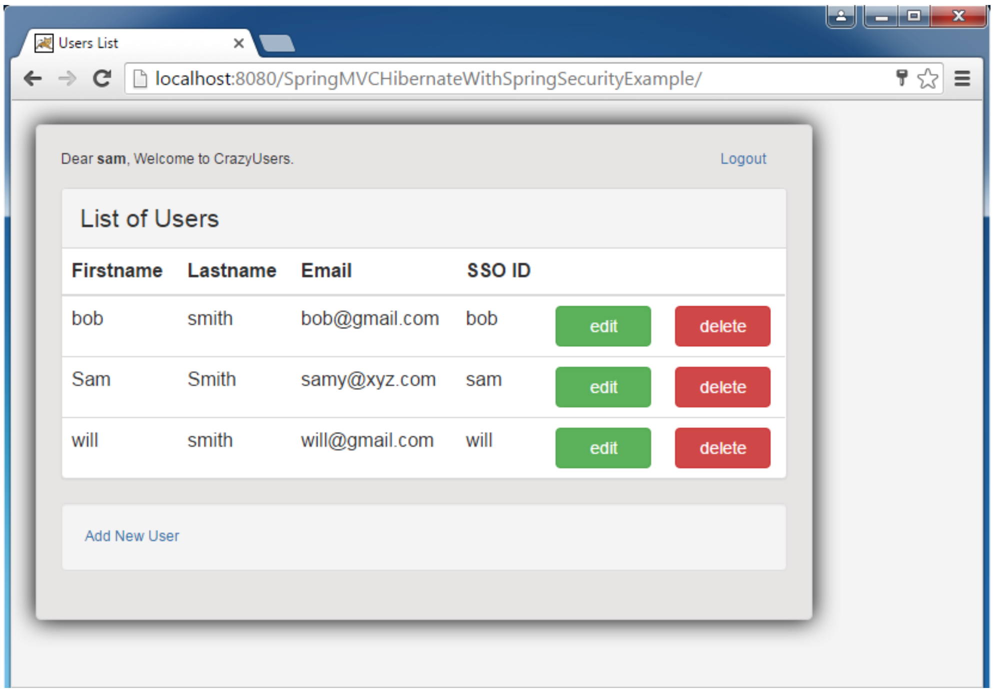
Created on: May 8, 2016 | **Last updated on:** June 17, 2017 websystiqueadmin

In this post, we will build a full-blown `Spring MVC` application secured using `Spring Security`, integrating with `MySQL` database using `Hibernate`, handling **Many-to-Many** relationship on view, storing passwords in **encrypted** format using `BCrypt`, and providing `RememberMe` functionality using custom `PersistentTokenRepository` implementation with Hibernate `HibernateTokenRepositoryImpl`, retrieving the records from database and updating or deleting them within `transaction`, all using annotation configuration. Let's get started.

This project can be served as a template for your own Spring MVC projects integrating Spring Security.

Websystique
987 likes
Like Page


Login page
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/login
sam
••••••
Remember Me
Log in


Users List

localhost:8080/SpringMVCHibernateWithSpringSecurityExample/

Dear **sam**, Welcome to CrazyUsers. Logout

List of Users

Firstname	Lastname	Email	SSO ID		
bob	smith	bob@gmail.com	bob	edit	delete
Sam	Smith	samy@xyz.com	sam	edit	delete
will	smith	will@gmail.com	will	edit	delete

Add New User

Note:

This post demonstrates a complete application with complete code. In order to manage the size of the post, i have skipped the textual descriptions of some basic stuff. In case you are interested in those details, this ,this & this post will help you.

Summary:

Recent Posts

Spring Boot Rest API Example

Spring Boot WAR deployment example

Spring Boot Introduction + hello world example

Secure Spring REST API using OAuth2

The project shows a simple user-management application. One can create a new user, edit or delete an existing user, and list all the users. User can be associated with one or more UserProfile, showing many-to-many relationship. URL's of the applications are secured using Spring Security. That means, based on the roles of logged in user, access to certain URL's will be granted or prohibited. On the view layer, user will see only the content he/she is allowed to based on the roles assigned to him/her, thanks to Spring Security tags for view layer.

Other interesting posts you may like

- Spring Boot+AngularJS+Spring Data+Hibernate+MySQL CRUD App
- Secure Spring REST API using OAuth2
- Spring Boot REST API Tutorial
- Spring Boot WAR deployment example
- Spring Boot Introduction + Hello World Example
- AngularJS+Spring Security using Basic Authentication
- Secure Spring REST API using Basic Authentication
- Spring 4 Email Template Library Example
- Spring 4 Caching Annotations Tutorial
- Spring 4 Cache Tutorial with EhCache
- Spring 4 MVC+JPA2+Hibernate Many-to-many Example
- Spring 4 Email With Attachment Tutorial
- Spring 4 Email Integration Tutorial
- Spring MVC 4+JMS+ActiveMQ Integration Example
- Spring 4+JMS+ActiveMQ @JmsLister @EnableJms Example
- Spring 4+JMS+ActiveMQ Integration Example
- Spring MVC 4+Apache Tiles 3 Integration Example
- Spring MVC 4+AngularJS Example
- Spring MVC 4+AngularJS Server communication example : CRUD application using ngResource $resource service
- Spring MVC 4+AngularJS Routing with UI-Router Example
- Spring MVC 4+Hibernate 4 Many-to-many JSP Example
- Spring MVC 4+Hibernate 4+MySQL+Maven integration + Testing example using annotations
- Spring Security 4 Hibernate Integration Annotation+XML Example
- Spring MVC4 FileUpload-Download Hibernate+MySQL Example

- Spring MVC 4 Form Validation and Resource Handling
- Spring Batch- MultiResourceItemReader & HibernateItemWriter example

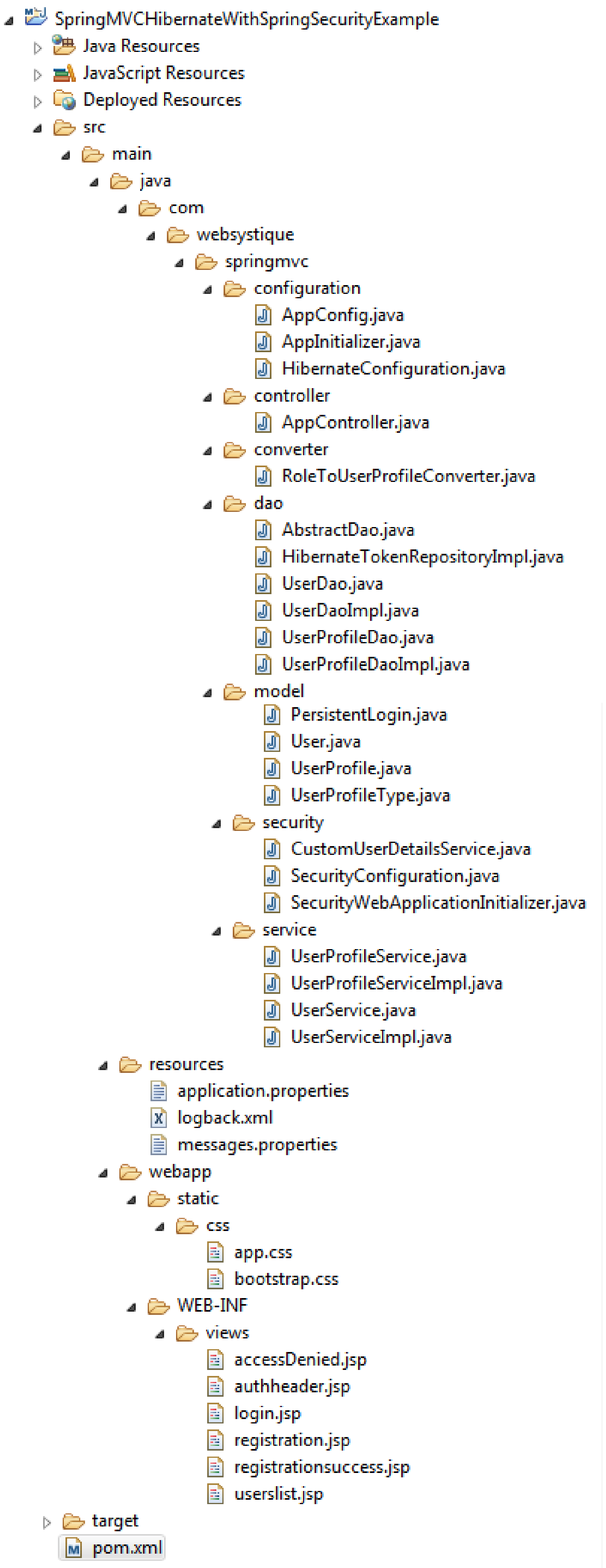
Following technologies being used:

- Spring 4.2.5.RELEASE
- Spring Security 4.0.4.RELEASE
- Hibernate Core 4.3.11.Final
- validation-api 1.1.0.Final
- hibernate-validator 5.1.3.Final
- MySQL Server 5.6
- Maven 3
- JDK 1.7
- Tomcat 8.0.21
- Eclipse MARS.1 Release 4.5.1
- logback 1.1.7

Let’s begin.

Step 1: Create the directory structure

Following will be the final project structure:



```
SpringMVCHibernateWithSpringSecurityExample
  Java Resources
  JavaScript Resources
  Deployed Resources
  src
    main
      java
        com
          websystique
            springmvc
              configuration
                AppConfig.java
                AppInitializer.java
                HibernateConfiguration.java
              controller
                AppController.java
              converter
                RoleToUserProfileConverter.java
              dao
                AbstractDao.java
                HibernateTokenRepositoryImpl.java
                UserDao.java
                UserDaoImpl.java
                UserProfileDao.java
                UserProfileDaoImpl.java
              model
                PersistentLogin.java
                User.java
                UserProfile.java
                UserProfileType.java
              security
                CustomUserDetailsService.java
                SecurityConfiguration.java
                SecurityWebApplicationInitializer.java
              service
                UserProfileService.java
                UserProfileServiceImpl.java
                UserService.java
                UserServiceImpl.java
      resources
        application.properties
        logback.xml
        messages.properties
      webapp
        static
          css
            app.css
            bootstrap.css
        WEB-INF
          views
            accessDenied.jsp
            authheader.jsp
            login.jsp
            registration.jsp
            registrationsuccess.jsp
            userslist.jsp
  target
  pom.xml
```

Let’s now add the content mentioned in above structure explaining each in detail.

Step 2: Update pom.xml to include required dependencies

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://m
    xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.websystique.springmvc</groupId>
    <artifactId>SpringMVCHibernateManyToManyCRUDExample</artifactId>
    <packaging>war</packaging>
    <version>1.0.0</version>
    <name>SpringMVCHibernateWithSpringSecurityExample</name>

    <properties>
        <springframework.version>4.2.5.RELEASE</springframework.version
        <springsecurity.version>4.0.4.RELEASE</springsecurity.version>
        <hibernate.version>4.3.11.Final</hibernate.version>
        <mysql.connector.version>5.1.31</mysql.connector.version>
    </properties>

    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${springframework.version}</version>
        </dependency>

        <!-- Spring Security -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>


        <!-- Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <!-- jsr303 validation -->
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>1.1.0.Final</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>5.1.3.Final</version>
        </dependency>

        <!-- MySQL -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.connector.version}</version>
        </dependency>

        <!-- SLF4J/Logback -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.1.7</version>
        </dependency>

        <!-- Servlet+JSP+JSTL -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.7</source>
                        <target>1.7</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>2.4</version>
                    <configuration>
                        <warSourceDirectory>src/main/webapp</warSourceD
                        <warName>SpringMVCHibernateWithSpringSecurityEx
                        <failOnMissingWebXml>false</failOnMissingWebXml
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
        <finalName>SpringMVCHibernateWithSpringSecurityExample</finalNa
    </build>
</project>
```

Step 3: Configure Security

The first and foremost step to add spring security in our application is to create Spring Security Java Configuration. This configuration creates a Servlet Filter known as the `springSecurityFilterChain` which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within our application

```
package com.websystique.springmvc.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationTrustR
import org.springframework.security.authentication.AuthenticationTrustR
import org.springframework.security.authentication.dao.DaoAuthenticatio
import org.springframework.security.config.annotation.authentication.bu
import org.springframework.security.config.annotation.web.builders.Http
import org.springframework.security.config.annotation.web.configuration
import org.springframework.security.config.annotation.web.configuration
import org.springframework.security.core.userdetails.UserDetailsService
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.rememberme.Persi
import org.springframework.security.web.authentication.rememberme.Persi

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter

    @Autowired
    @Qualifier("customUserDetailsService")
    UserDetailsService userDetailsService;

    @Autowired
    PersistentTokenRepository tokenRepository;

    @Autowired
    public void configureGlobalSecurity(AuthenticationManagerBuilder au
        auth.userDetailsService(userDetailsService);
        auth.authenticationProvider(authenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().antMatchers("/", "/list")
                .access("hasRole('USER') or hasRole('ADMIN') or hasRole
                .antMatchers("/newuser/**", "/delete-user-*").access("h
                .access("hasRole('ADMIN') or hasRole('DBA')").and().for
                .loginProcessingUrl("/login").usernameParameter("ssoId"
                .rememberMe().rememberMeParameter("remember-me").tokenR
                .tokenValiditySeconds(86400).and().csrf().and().excepti
    }

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

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthe
        authenticationProvider.setUserDetailsService(userDetailsService
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }

    @Bean
    public PersistentTokenBasedRememberMeServices getPersistentTokenBas
        PersistentTokenBasedRememberMeServices tokenBasedservice = new
                "remember-me", userDetailsService, tokenRepository);
        return tokenBasedservice;
    }

    @Bean
    public AuthenticationTrustResolver getAuthenticationTrustResolver()
        return new AuthenticationTrustResolverImpl();
    }
}
```

As shown above, the access to URLs is governed as follows:

- '/' & '/list' : Accessible to everyone
- '/newuser' & '/delete-user-*' : Accessible only to Admin
- '/edit-user-*' : Accessible to Admin & DBA

Since we are storing the credentials in database, configuring `DaoAuthenticationProvider` with `UserDetailsService` would come handy. Additionally, in order to encrypt the password in database, we have chosen `BCryptPasswordEncoder`. Moreover, since we will also provide RememberMe functionality, keeping track of token-data in database, we configured a `PersistentTokenRepository` implementation.

Spring Security comes with two implementation of PersistentTokenRepository : JdbcTokenRepositoryImpl and InMemoryTokenRepositoryImpl. We could have opted for JdbcTokenRepositoryImpl [this post demonstrates the RememberMe with JdbcTokenRepositoryImpl], but since we are using Hibernate in our application, why not create a custom implementation using Hibernate instead of using JDBC? Shown below is an attempt for the same.

```
package com.websystique.springmvc.dao;

import java.util.Date;

import org.hibernate.Criteria;
import org.hibernate.criterion.Restrictions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.web.authentication.rememberme.Persi
import org.springframework.security.web.authentication.rememberme.Persi
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.websystique.springmvc.dao.AbstractDao;
import com.websystique.springmvc.model.PersistentLogin;

@Repository("tokenRepositoryDao")
@Transactional
public class HibernateTokenRepositoryImpl extends AbstractDao<String, P
        implements PersistentTokenRepository {

    static final Logger logger = LoggerFactory.getLogger(HibernateToken

    @Override
    public void createNewToken(PersistentRememberMeToken token) {
        logger.info("Creating Token for user : {}", token.getUsername()
        PersistentLogin persistentLogin = new PersistentLogin();
        persistentLogin.setUsername(token.getUsername());
        persistentLogin.setSeries(token.getSeries());
        persistentLogin.setToken(token.getTokenValue());
        persistentLogin.setLast_used(token.getDate());
        persist(persistentLogin);

    }

    @Override
    public PersistentRememberMeToken getTokenForSeries(String seriesId)
        logger.info("Fetch Token if any for seriesId : {}", seriesId);
        try {
            Criteria crit = createEntityCriteria();
            crit.add(Restrictions.eq("series", seriesId));
            PersistentLogin persistentLogin = (PersistentLogin) crit.un

            return new PersistentRememberMeToken(persistentLogin.getUse
                    persistentLogin.getToken(), persistentLogin.getLast
        } catch (Exception e) {
            logger.info("Token not found...");
            return null;
        }
    }

    @Override
    public void removeUserTokens(String username) {
        logger.info("Removing Token if any for user : {}", username);
        Criteria crit = createEntityCriteria();
        crit.add(Restrictions.eq("username", username));
        PersistentLogin persistentLogin = (PersistentLogin) crit.unique
        if (persistentLogin != null) {
            logger.info("rememberMe was selected");
            delete(persistentLogin);
        }

    }

    @Override
    public void updateToken(String seriesId, String tokenValue, Date la
        logger.info("Updating Token for seriesId : {}", seriesId);
        PersistentLogin persistentLogin = getByKey(seriesId);
        persistentLogin.setToken(tokenValue);
        persistentLogin.setLast_used(lastUsed);
        update(persistentLogin);
    }

}
```

Above implementation uses an Entity [PersistentLogin] mapped to persistent_logins table, shown below is the entity itself.

```
package com.websystique.springmvc.model;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name="PERSISTENT_LOGINS")
public class PersistentLogin implements Serializable{

    @Id
    private String series;

    @Column(name="USERNAME", unique=true, nullable=false)
    private String username;

    @Column(name="TOKEN", unique=true, nullable=false)
    private String token;

    @Temporal(TemporalType.TIMESTAMP)
    private Date last_used;

    public String getSeries() {
        return series;
    }

    public void setSeries(String series) {
        this.series = series;
    }

    public String getUsername() {
        return username;
    }

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

    public String getToken() {
        return token;
    }

    public void setToken(String token) {
        this.token = token;
    }

    public Date getLast_used() {
        return last_used;
    }

    public void setLast_used(Date last_used) {
        this.last_used = last_used;
    }


}
```

The UserDetailsService implementation, used in Security configuration is shown below:

```
package com.websystique.springmvc.security;

import java.util.ArrayList;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthori
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService
import org.springframework.security.core.userdetails.UsernameNotFoundEx
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.websystique.springmvc.model.User;
import com.websystique.springmvc.model.UserProfile;
import com.websystique.springmvc.service.UserService;


@Service("customUserDetailsService")
public class CustomUserDetailsService implements UserDetailsService{

    static final Logger logger = LoggerFactory.getLogger(CustomUserDeta

    @Autowired
    private UserService userService;

    @Transactional(readOnly=true)
    public UserDetails loadUserByUsername(String ssoId)
            throws UsernameNotFoundException {
        User user = userService.findBySSO(ssoId);
        logger.info("User : {}", user);
        if(user==null){
            logger.info("User not found");
            throw new UsernameNotFoundException("Username not found");
        }
            return new org.springframework.security.core.userdetails.Us
                 true, true, true, true, getGrantedAuthorities(user));
    }
    private List<GrantedAuthority> getGrantedAuthorities(User user){
        List<GrantedAuthority> authorities = new ArrayList<GrantedAutho

        for(UserProfile userProfile : user.getUserProfiles()){
            logger.info("UserProfile : {}", userProfile);
            authorities.add(new SimpleGrantedAuthority("ROLE_"+userProf
        }
        logger.info("authorities : {}", authorities);
        return authorities;
    }

}
```

Finally, register the springSecurityFilter with application war using below mentioned initializer class.

```
package com.websystique.springmvc.security;

import org.springframework.security.web.context.AbstractSecurityWebAppl

public class SecurityWebApplicationInitializer extends AbstractSecurity

}
```

That's all with Spring Security Configuration. Now let's begin with Spring MVC part, discussing Hibernate configuration, necessary DAO, models & services along the way.

Step 4: Configure Hibernate

```
package com.websystique.springmvc.configuration;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate4.HibernateTransactionManager;
import org.springframework.orm.hibernate4.LocalSessionFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionMana

@Configuration
@EnableTransactionManagement
@ComponentScan({ "com.websystique.springmvc.configuration" })
@PropertySource(value = { "classpath:application.properties" })
public class HibernateConfiguration {

    @Autowired
    private Environment environment;

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactor
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan(new String[] { "com.websystiqu
        sessionFactory.setHibernateProperties(hibernateProperties());
```

```
            return sessionFactory;
         }

        @Bean
        public DataSource dataSource() {
            DriverManagerDataSource dataSource = new DriverManagerDataSourc
            dataSource.setDriverClassName(environment.getRequiredProperty("
            dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));
            dataSource.setUsername(environment.getRequiredProperty("jdbc.us
            dataSource.setPassword(environment.getRequiredProperty("jdbc.pa
            return dataSource;
        }

        private Properties hibernateProperties() {
            Properties properties = new Properties();
            properties.put("hibernate.dialect", environment.getRequiredProp
            properties.put("hibernate.show_sql", environment.getRequiredPro
            properties.put("hibernate.format_sql", environment.getRequiredP
            return properties;
        }

        @Bean
        @Autowired
        public HibernateTransactionManager transactionManager(SessionFactor
           HibernateTransactionManager txManager = new HibernateTransaction
           txManager.setSessionFactory(s);
           return txManager;
        }
    }
```

Below is the properties file used in this post.
`/src/main/resources/application.properties`

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/websystique
jdbc.username = myuser
jdbc.password = mypassword
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = true
```

Step 5: Configure Spring MVC

```
package com.websystique.springmvc.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.format.FormatterRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.PathMatchConfi
import org.springframework.web.servlet.config.annotation.ResourceHandle
import org.springframework.web.servlet.config.annotation.ViewResolverRe
import org.springframework.web.servlet.config.annotation.WebMvcConfigur
import org.springframework.web.servlet.view.InternalResourceViewResolve
import org.springframework.web.servlet.view.JstlView;

import com.websystique.springmvc.converter.RoleToUserProfileConverter;


@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.websystique.springmvc")
public class AppConfig extends WebMvcConfigurerAdapter{
    @Autowired
    RoleToUserProfileConverter roleToUserProfileConverter;


    /**
     * Configure ViewResolvers to deliver preferred views.
     */
    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {

        InternalResourceViewResolver viewResolver = new InternalResourc
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        registry.viewResolver(viewResolver);
    }

    /**
     * Configure ResourceHandlers to serve static resources like CSS/ J
     */
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/**").addResourceLocations(
    }

    /**
     * Configure Converter to be used.
     * In our example, we need a converter to convert string values[Rol
     */
    @Override
    public void addFormatters(FormatterRegistry registry) {
        registry.addConverter(roleToUserProfileConverter);
    }


    /**
     * Configure MessageSource to lookup any validation/error message i
     */
    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource messageSource = new ResourceBundleM
        messageSource.setBasename("messages");
        return messageSource;
    }

    /**Optional. It's only required when handling '.' in @PathVariables
     * It's a known bug in Spring [https://jira.spring.io/browse/SPR-61
     * This is a workaround for this issue.
     */
    @Override
    public void configurePathMatch(PathMatchConfigurer matcher) {
        matcher.setUseRegisteredSuffixPatternMatch(true);
    }
}
```

The main highlight of this configuration is RoleToUserProfileConverter. It will take care of mapping the individual userProfile id's on view to actual UserProfile Entities in database.

```
package com.websystique.springmvc.converter;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

import com.websystique.springmvc.model.UserProfile;
import com.websystique.springmvc.service.UserProfileService;

/**
```

```
  * A converter class used in views to map id's to actual userProfile ob
  */
@Component
public class RoleToUserProfileConverter implements Converter<Object, Us

    static final Logger logger = LoggerFactory.getLogger(RoleToUserProf

    @Autowired
    UserProfileService userProfileService;

    /**
     * Gets UserProfile by Id
     * @see org.springframework.core.convert.converter.Converter#conver
     */
    public UserProfile convert(Object element) {
        Integer id = Integer.parseInt((String)element);
        UserProfile profile= userProfileService.findById(id);
        logger.info("Profile : {}",profile);
        return profile;
    }

}
```

Since we are using JSR validators in our application to validate user input, we have configured the messages to be shown to user in case of validation failures. shown below is message.properties file:

```
NotEmpty.user.firstName=First name can not be blank.
NotEmpty.user.lastName=Last name can not be blank.
NotEmpty.user.email=Email can not be blank.
NotEmpty.user.password=Password can not be blank.
NotEmpty.user.ssoId=SSO ID can not be blank.
NotEmpty.user.userProfiles=At least one profile must be selected.
non.unique.ssoId=SSO ID {0} already exist. Please fill in different val
```

Finally, the Spring Intializer class is shown below:

```
package com.websystique.springmvc.configuration;

import org.springframework.web.servlet.support.AbstractAnnotationConfig

public class AppInitializer extends AbstractAnnotationConfigDispatcherS

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { AppConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

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

}
```

Step 6: Create Spring Controller

```
package com.websystique.springmvc.controller;

import java.util.List;
import java.util.Locale;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.security.authentication.AuthenticationTrustR
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.rememberme.Persi
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.SessionAttributes;

import com.websystique.springmvc.model.User;
import com.websystique.springmvc.model.UserProfile;
import com.websystique.springmvc.service.UserProfileService;
import com.websystique.springmvc.service.UserService;



@Controller
@RequestMapping("/")
@SessionAttributes("roles")
public class AppController {

    @Autowired
    UserService userService;

    @Autowired
    UserProfileService userProfileService;

    @Autowired
    MessageSource messageSource;

    @Autowired
    PersistentTokenBasedRememberMeServices persistentTokenBasedRemember

    @Autowired
    AuthenticationTrustResolver authenticationTrustResolver;


    /**
     * This method will list all existing users.
     */
    @RequestMapping(value = { "/", "/list" }, method = RequestMethod.GE
    public String listUsers(ModelMap model) {

        List<User> users = userService.findAllUsers();
        model.addAttribute("users", users);
        model.addAttribute("loggedinuser", getPrincipal());
        return "userslist";
    }

    /**
     * This method will provide the medium to add a new user.
     */
    @RequestMapping(value = { "/newuser" }, method = RequestMethod.GET)
    public String newUser(ModelMap model) {
        User user = new User();
        model.addAttribute("user", user);
        model.addAttribute("edit", false);
        model.addAttribute("loggedinuser", getPrincipal());
        return "registration";
    }
    /**
     * This method will be called on form submission, handling POST req
     * saving user in database. It also validates the user input
     */
    @RequestMapping(value = { "/newuser" }, method = RequestMethod.POST
    public String saveUser(@Valid User user, BindingResult result,
            ModelMap model) {

        if (result.hasErrors()) {
            return "registration";
        }

        /*
         * Preferred way to achieve uniqueness of field [sso] should be
         * and applying it on field [sso] of Model class [User].
         *
         * Below mentioned peace of code [if block] is to demonstrate t
         * framework as well while still using internationalized messag
         *
         */
        if(!userService.isUserSSOUnique(user.getId(), user.getSsoId()))
            FieldError ssoError =new FieldError("user","ssoId",messageS
            result.addError(ssoError);
            return "registration";
        }

        userService.saveUser(user);

        model.addAttribute("success", "User " + user.getFirstName() + "
        model.addAttribute("loggedinuser", getPrincipal());
        //return "success";
        return "registrationsuccess";
    }


    /**
     * This method will provide the medium to update an existing user.
     */
    @RequestMapping(value = { "/edit-user-{ssoId}" }, method = RequestM
    public String editUser(@PathVariable String ssoId, ModelMap model)
        User user = userService.findBySSO(ssoId);
        model.addAttribute("user", user);
        model.addAttribute("edit", true);
        model.addAttribute("loggedinuser", getPrincipal());
        return "registration";
    }

    /**
     * This method will be called on form submission, handling POST req
     * updating user in database. It also validates the user input
     */
    @RequestMapping(value = { "/edit-user-{ssoId}" }, method = RequestM
    public String updateUser(@Valid User user, BindingResult result,
            ModelMap model, @PathVariable String ssoId) {

        if (result.hasErrors()) {
            return "registration";
        }

        /*//Uncomment below 'if block' if you WANT TO ALLOW UPDATING SS
        if(!userService.isUserSSOUnique(user.getId(), user.getSsoId()))
            FieldError ssoError =new FieldError("user","ssoId",messageS
            result.addError(ssoError);
            return "registration";
        }*/


        userService.updateUser(user);

        model.addAttribute("success", "User " + user.getFirstName() + "
        model.addAttribute("loggedinuser", getPrincipal());
        return "registrationsuccess";
    }


    /**
     * This method will delete an user by it's SSOID value.
     */
    @RequestMapping(value = { "/delete-user-{ssoId}" }, method = Reques
    public String deleteUser(@PathVariable String ssoId) {
        userService.deleteUserBySSO(ssoId);
        return "redirect:/list";
    }


    /**
     * This method will provide UserProfile list to views
     */
    @ModelAttribute("roles")
    public List<UserProfile> initializeProfiles() {
        return userProfileService.findAll();
    }

    /**
     * This method handles Access-Denied redirect.
     */
    @RequestMapping(value = "/Access_Denied", method = RequestMethod.GE
    public String accessDeniedPage(ModelMap model) {
        model.addAttribute("loggedinuser", getPrincipal());
        return "accessDenied";
    }

    /**
     * This method handles login GET requests.
     * If users is already logged-in and tries to goto login page again
     */
    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String loginPage() {
        if (isCurrentAuthenticationAnonymous()) {
            return "login";
        } else {
            return "redirect:/list";
        }
    }

    /**
     * This method handles logout requests.
     * Toggle the handlers if you are RememberMe functionality is usele
     */
    @RequestMapping(value="/logout", method = RequestMethod.GET)
    public String logoutPage (HttpServletRequest request, HttpServletRe
        Authentication auth = SecurityContextHolder.getContext().getAut
        if (auth != null){
            //new SecurityContextLogoutHandler().logout(request, respon
            persistentTokenBasedRememberMeServices.logout(request, resp
            SecurityContextHolder.getContext().setAuthentication(null);
        }
        return "redirect:/login?logout";
    }

    /**
     * This method returns the principal[user-name] of logged-in user.
     */
    private String getPrincipal(){
        String userName = null;
        Object principal = SecurityContextHolder.getContext().getAuthen

        if (principal instanceof UserDetails) {
            userName = ((UserDetails)principal).getUsername();
        } else {
            userName = principal.toString();
        }
        return userName;
    }

    /**
     * This method returns true if users is already authenticated [logg
     */
    private boolean isCurrentAuthenticationAnonymous() {
        final Authentication authentication = SecurityContextHolder.get
        return authenticationTrustResolver.isAnonymous(authentication);
    }


}
```

This is a trivial Spring MVC controller. Comments on Each method provide the explanations.

Step 7: Create Models

```
package com.websystique.springmvc.model;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

import org.hibernate.validator.constraints.NotEmpty;

@Entity
@Table(name="APP_USER")
public class User implements Serializable{

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    @NotEmpty
    @Column(name="SSO_ID", unique=true, nullable=false)
    private String ssoId;

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

    @NotEmpty
    @Column(name="FIRST_NAME", nullable=false)
    private String firstName;

    @NotEmpty
    @Column(name="LAST_NAME", nullable=false)
    private String lastName;

    @NotEmpty
    @Column(name="EMAIL", nullable=false)
    private String email;

    @NotEmpty
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "APP_USER_USER_PROFILE",
             joinColumns = { @JoinColumn(name = "USER_ID") },
             inverseJoinColumns = { @JoinColumn(name = "USER_PROFILE_ID
    private Set<UserProfile> userProfiles = new HashSet<UserProfile>();

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    public String getSsoId() {
        return ssoId;
    }

    public void setSsoId(String ssoId) {
        this.ssoId = ssoId;
    }

    public String getPassword() {
        return password;
    }

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

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Set<UserProfile> getUserProfiles() {
        return userProfiles;
    }

    public void setUserProfiles(Set<UserProfile> userProfiles) {
        this.userProfiles = userProfiles;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        result = prime * result + ((ssoId == null) ? 0 : ssoId.hashCode
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (!(obj instanceof User))
            return false;
        User other = (User) obj;
        if (id == null) {
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        if (ssoId == null) {
            if (other.ssoId != null)
                return false;
        } else if (!ssoId.equals(other.ssoId))
            return false;
        return true;
    }

    /*
```

```
     * DO-NOT-INCLUDE passwords in toString function.
     * It is done here just for convenience purpose.
     */
    @Override
    public String toString() {
        return "User [id=" + id + ", ssoId=" + ssoId + ", password=" +
                + ", firstName=" + firstName + ", lastName=" + lastName
                + ", email=" + email + "]";
    }



}
```

```
package com.websystique.springmvc.model;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="USER_PROFILE")
public class UserProfile implements Serializable{

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    @Column(name="TYPE", length=15, unique=true, nullable=false)
    private String type = UserProfileType.USER.getUserProfileType();

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        result = prime * result + ((type == null) ? 0 : type.hashCode()
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (!(obj instanceof UserProfile))
            return false;
        UserProfile other = (UserProfile) obj;
        if (id == null) {
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        if (type == null) {
            if (other.type != null)
                return false;
        } else if (!type.equals(other.type))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "UserProfile [id=" + id + ", type=" + type + "]";
    }



}
```

```
package com.websystique.springmvc.model;

import java.io.Serializable;

public enum UserProfileType implements Serializable{
    USER("USER"),
    DBA("DBA"),
    ADMIN("ADMIN");

    String userProfileType;

    private UserProfileType(String userProfileType){
        this.userProfileType = userProfileType;
    }

    public String getUserProfileType(){
        return userProfileType;
    }

}
```

Step 7: Create DAOs

```
package com.websystique.springmvc.dao;

import java.io.Serializable;

import java.lang.reflect.ParameterizedType;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;

public abstract class AbstractDao<PK extends Serializable, T> {

    private final Class<T> persistentClass;

    @SuppressWarnings("unchecked")
    public AbstractDao(){
        this.persistentClass =(Class<T>) ((ParameterizedType) this.getC
    }

    @Autowired
    private SessionFactory sessionFactory;

    protected Session getSession(){
        return sessionFactory.getCurrentSession();
    }

    @SuppressWarnings("unchecked")
    public T getByKey(PK key) {
        return (T) getSession().get(persistentClass, key);
    }

    public void persist(T entity) {
        getSession().persist(entity);
    }

    public void update(T entity) {
        getSession().update(entity);
    }

    public void delete(T entity) {
        getSession().delete(entity);
    }

    protected Criteria createEntityCriteria(){
        return getSession().createCriteria(persistentClass);
    }

}
```

```
package com.websystique.springmvc.dao;

import java.util.List;

import com.websystique.springmvc.model.User;


public interface UserDao {

    User findById(int id);

    User findBySSO(String sso);

    void save(User user);

    void deleteBySSO(String sso);

    List<User> findAllUsers();

}
```

```
package com.websystique.springmvc.dao;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Hibernate;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Repository;

import com.websystique.springmvc.model.User;



@Repository("userDao")
public class UserDaoImpl extends AbstractDao<Integer, User> implements

    static final Logger logger = LoggerFactory.getLogger(UserDaoImpl.cl

    public User findById(int id) {
        User user = getByKey(id);
        if(user!=null){
            Hibernate.initialize(user.getUserProfiles());
        }
        return user;
    }
    public User findBySSO(String sso) {
        logger.info("SSO : {}", sso);
        Criteria crit = createEntityCriteria();
        crit.add(Restrictions.eq("ssoId", sso));
        User user = (User)crit.uniqueResult();
        if(user!=null){
            Hibernate.initialize(user.getUserProfiles());
        }
        return user;
    }

    @SuppressWarnings("unchecked")
    public List<User> findAllUsers() {
        Criteria criteria = createEntityCriteria().addOrder(Order.asc("
        criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);//
        List<User> users = (List<User>) criteria.list();

        // No need to fetch userProfiles since we are not showing them
        // Uncomment below lines for eagerly fetching of userProfiles i
        /*
        for(User user : users){
            Hibernate.initialize(user.getUserProfiles());
        }*/
        return users;
    }

    public void save(User user) {
        persist(user);
    }

    public void deleteBySSO(String sso) {
        Criteria crit = createEntityCriteria();
        crit.add(Restrictions.eq("ssoId", sso));
        User user = (User)crit.uniqueResult();
        delete(user);
    }

}
```

```
package com.websystique.springmvc.dao;

import java.util.List;

import com.websystique.springmvc.model.UserProfile;


public interface UserProfileDao {

    List<UserProfile> findAll();

    UserProfile findByType(String type);

    UserProfile findById(int id);
}
```

```
package com.websystique.springmvc.dao;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.stereotype.Repository;

import com.websystique.springmvc.model.UserProfile;



@Repository("userProfileDao")
public class UserProfileDaoImpl extends AbstractDao<Integer, UserProfil
```

```
    public UserProfile findById(int id) {
        return getByKey(id);
    }

    public UserProfile findByType(String type) {
        Criteria crit = createEntityCriteria();
        crit.add(Restrictions.eq("type", type));
        return (UserProfile) crit.uniqueResult();
    }

    @SuppressWarnings("unchecked")
    public List<UserProfile> findAll(){
        Criteria crit = createEntityCriteria();
        crit.addOrder(Order.asc("type"));
        return (List<UserProfile>)crit.list();
    }

}
```

Step 8: Create Services

```
package com.websystique.springmvc.service;

import java.util.List;

import com.websystique.springmvc.model.User;


public interface UserService {

    User findById(int id);

    User findBySSO(String sso);

    void saveUser(User user);

    void updateUser(User user);

    void deleteUserBySSO(String sso);

    List<User> findAllUsers();

    boolean isUserSSOUnique(Integer id, String sso);

}
```

```
package com.websystique.springmvc.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.websystique.springmvc.dao.UserDao;
import com.websystique.springmvc.model.User;


@Service("userService")
@Transactional
public class UserServiceImpl implements UserService{

    @Autowired
    private UserDao dao;

    @Autowired
    private PasswordEncoder passwordEncoder;
    public User findById(int id) {
        return dao.findById(id);
    }

    public User findBySSO(String sso) {
        User user = dao.findBySSO(sso);
        return user;
    }

    public void saveUser(User user) {
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        dao.save(user);
    }

    /*
     * Since the method is running with Transaction, No need to call hi
     * Just fetch the entity from db and update it with proper values w
     * It will be updated in db once transaction ends.
     */
    public void updateUser(User user) {
        User entity = dao.findById(user.getId());
        if(entity!=null){
            entity.setSsoId(user.getSsoId());
            if(!user.getPassword().equals(entity.getPassword())){
                entity.setPassword(passwordEncoder.encode(user.getPassw
            }
            entity.setFirstName(user.getFirstName());
            entity.setLastName(user.getLastName());
            entity.setEmail(user.getEmail());
            entity.setUserProfiles(user.getUserProfiles());
        }
    }


    public void deleteUserBySSO(String sso) {
        dao.deleteBySSO(sso);
    }

    public List<User> findAllUsers() {
        return dao.findAllUsers();
    }

    public boolean isUserSSOUnique(Integer id, String sso) {
        User user = findBySSO(sso);
        return ( user == null || ((id != null) && (user.getId() == id))
    }

}
```

```
package com.websystique.springmvc.service;

import java.util.List;

import com.websystique.springmvc.model.UserProfile;


public interface UserProfileService {

    UserProfile findById(int id);

    UserProfile findByType(String type);

    List<UserProfile> findAll();

}
```

```
package com.websystique.springmvc.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.websystique.springmvc.dao.UserProfileDao;
import com.websystique.springmvc.model.UserProfile;


@Service("userProfileService")
@Transactional
public class UserProfileServiceImpl implements UserProfileService{

    @Autowired
    UserProfileDao dao;

    public UserProfile findById(int id) {
        return dao.findById(id);
    }

    public UserProfile findByType(String type){
        return dao.findByType(type);
    }

    public List<UserProfile> findAll() {
        return dao.findAll();
    }
}
```

Step 9: Create Views

Start with login page,asking username & password, and optionally 'RememberMe' flag.

`WEB-INF/views/login.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pa
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO
        <title>Login page</title>
        <link href="<c:url value='/static/css/bootstrap.css' />"  rel="
        <link href="<c:url value='/static/css/app.css' />" rel="stylesh
        <link rel="stylesheet" type="text/css" href="//cdnjs.cloudflare
    </head>

    <body>
        <div id="mainWrapper">
            <div class="login-container">
                <div class="login-card">
                    <div class="login-form">
                        <c:url var="loginUrl" value="/login" />
                        <form action="${loginUrl}" method="post" class=
                            <c:if test="${param.error != null}">
                                <div class="alert alert-danger">
                                    <p>Invalid username and password.</
                                </div>
                            </c:if>
                            <c:if test="${param.logout != null}">
                                <div class="alert alert-success">
                                    <p>You have been logged out success
                                </div>
                            </c:if>
                            <div class="input-group input-sm">
                                <label class="input-group-addon" for="u
                                <input type="text" class="form-control"
                            </div>
                            <div class="input-group input-sm">
                                <label class="input-group-addon" for="p
                                <input type="password" class="form-cont
                            </div>
                            <div class="input-group input-sm">
                              <div class="checkbox">
                                <label><input type="checkbox" id="remem
                              </div>
                            </div>
                            <input type="hidden" name="${_csrf.paramete

                            <div class="form-actions">
                                <input type="submit"
                                    class="btn btn-block btn-primary bt
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>

    </body>
</html>
```

Once the user is logged-in successfully, he will be presented with list page, showing all existing users. Pay special attentions to Spring Security tags usage below. Add, Edit & Delete links/buttons are shown based on roles only, so a user with ‘User’ role will not even be able to see them. You may ask: but what about directly typing the url in browser-bar? Well, we have already secured the URL’s in Spring Security configuration, so no-worries.

`WEB-INF/views/userslist.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pa
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/ta

<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-885
    <title>Users List</title>
    <link href="<c:url value='/static/css/bootstrap.css' />" rel="style
    <link href="<c:url value='/static/css/app.css' />" rel="stylesheet"
</head>

<body>
    <div class="generic-container">
        <%@include file="authheader.jsp" %>
        <div class="panel panel-default">
              <!-- Default panel contents -->
            <div class="panel-heading"><span class="lead">List of Users
            <table class="table table-hover">
                <thead>
                    <tr>
                        <th>Firstname</th>
                        <th>Lastname</th>
                        <th>Email</th>
                        <th>SSO ID</th>
                        <sec:authorize access="hasRole('ADMIN') or hasR
                            <th width="100"></th>
                        </sec:authorize>
                        <sec:authorize access="hasRole('ADMIN')">
                            <th width="100"></th>
                        </sec:authorize>

                    </tr>
                </thead>
                <tbody>
                <c:forEach items="${users}" var="user">
                    <tr>
                        <td>${user.firstName}</td>
                        <td>${user.lastName}</td>
                        <td>${user.email}</td>
                        <td>${user.ssoId}</td>
                        <sec:authorize access="hasRole('ADMIN') or hasR
                            <td><a href="<c:url value='/edit-user-${use
                        </sec:authorize>
                        <sec:authorize access="hasRole('ADMIN')">
                            <td><a href="<c:url value='/delete-user-${u
                        </sec:authorize>
                    </tr>
                </c:forEach>
                </tbody>
            </table>
        </div>
        <sec:authorize access="hasRole('ADMIN')">
            <div class="well">
                <a href="<c:url value='/newuser' />">Add New User</a>
            </div>
        </sec:authorize>
    </div>
</body>
</html>
```

Above page also includes a jsp containing welcome-messagealong with Logout link as shown below:

`WEB-INF/views/authheader.jsp`

```
<div class="authbar">
    <span>Dear <strong>${loggedinuser}</strong>, Welcome to CrazyUsers.
</div>
```

A user with 'Admin' role can add a new user. Shown below is the registration page for the same.
`WEB-INF/views/registration.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pa
<%@ page isELIgnored="false" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-885
    <title>User Registration Form</title>
    <link href="<c:url value='/static/css/bootstrap.css' />" rel="style
    <link href="<c:url value='/static/css/app.css' />" rel="stylesheet"
</head>

<body>
    <div class="generic-container">
        <%@include file="authheader.jsp" %>

        <div class="well lead">User Registration Form</div>
        <form:form method="POST" modelAttribute="user" class="form-hori
            <form:input type="hidden" path="id" id="id"/>

            <div class="row">
                <div class="form-group col-md-12">
                    <label class="col-md-3 control-lable" for="firstNam
                    <div class="col-md-7">
                        <form:input type="text" path="firstName" id="fi
                        <div class="has-error">
                            <form:errors path="firstName" class="help-i
                        </div>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="form-group col-md-12">
                    <label class="col-md-3 control-lable" for="lastName
                    <div class="col-md-7">
                        <form:input type="text" path="lastName" id="las
                        <div class="has-error">
                            <form:errors path="lastName" class="help-in
                        </div>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="form-group col-md-12">
                    <label class="col-md-3 control-lable" for="ssoId">S
                    <div class="col-md-7">
                        <c:choose>
                            <c:when test="${edit}">
                                <form:input type="text" path="ssoId" id
                            </c:when>
                            <c:otherwise>
                                <form:input type="text" path="ssoId" id
                                <div class="has-error">
                                    <form:errors path="ssoId" class="he
                                </div>
                            </c:otherwise>
                        </c:choose>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="form-group col-md-12">
                    <label class="col-md-3 control-lable" for="password
                    <div class="col-md-7">
                        <form:input type="password" path="password" id=
                        <div class="has-error">
                            <form:errors path="password" class="help-in
                        </div>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="form-group col-md-12">
                    <label class="col-md-3 control-lable" for="email">E
                    <div class="col-md-7">
                        <form:input type="text" path="email" id="email"
                        <div class="has-error">
                            <form:errors path="email" class="help-inlin
                        </div>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="form-group col-md-12">
                    <label class="col-md-3 control-lable" for="userProf
                    <div class="col-md-7">
                        <form:select path="userProfiles" items="${roles
                        <div class="has-error">
                            <form:errors path="userProfiles" class="hel
                        </div>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="form-actions floatRight">
                    <c:choose>
                        <c:when test="${edit}">
                            <input type="submit" value="Update" class="
                        </c:when>
                        <c:otherwise>
                            <input type="submit" value="Register" class
                        </c:otherwise>
                    </c:choose>
                </div>
            </div>
        </form:form>
    </div>
</body>
</html>
```

`WEB-INF/views/registrationsuccess.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pa
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>


<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-885
    <title>Registration Confirmation Page</title>
    <link href="<c:url value='/static/css/bootstrap.css' />" rel="style
    <link href="<c:url value='/static/css/app.css' />" rel="stylesheet"
</head>
<body>
    <div class="generic-container">
        <%@include file="authheader.jsp" %>

        <div class="alert alert-success lead">
            ${success}
        </div>

        <span class="well floatRight">
            Go to <a href="<c:url value='/list' />">Users List</a>
        </span>
    </div>
</body>

</html>
```

AccessDenied page will be shown if the users is not allowed to go to certain url’s.

`WEB-INF/views/accessDenied.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pa
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-885
    <title>AccessDenied page</title>
</head>
<body>
    <div class="generic-container">
        <div class="authbar">
            <span>Dear <strong>${loggedinuser}</strong>, You are not au
        </div>
    </div>
</body>
</html>
```

Step 10: Create and populate schema in database

```
/*All User's gets stored in APP_USER table*/
create table APP_USER (
   id BIGINT NOT NULL AUTO_INCREMENT,
   sso_id VARCHAR(30) NOT NULL,
   password VARCHAR(100) NOT NULL,
   first_name VARCHAR(30) NOT NULL,
   last_name  VARCHAR(30) NOT NULL,
   email VARCHAR(30) NOT NULL,
   PRIMARY KEY (id),
   UNIQUE (sso_id)
);

/* USER_PROFILE table contains all possible roles */
create table USER_PROFILE(
   id BIGINT NOT NULL AUTO_INCREMENT,
   type VARCHAR(30) NOT NULL,
   PRIMARY KEY (id),
   UNIQUE (type)
);

/* JOIN TABLE for MANY-TO-MANY relationship*/
CREATE TABLE APP_USER_USER_PROFILE (
    user_id BIGINT NOT NULL,
    user_profile_id BIGINT NOT NULL,
    PRIMARY KEY (user_id, user_profile_id),
    CONSTRAINT FK_APP_USER FOREIGN KEY (user_id) REFERENCES APP_USER (i
    CONSTRAINT FK_USER_PROFILE FOREIGN KEY (user_profile_id) REFERENCES
);

/* Populate USER_PROFILE Table */
INSERT INTO USER_PROFILE(type)
VALUES ('USER');

INSERT INTO USER_PROFILE(type)
VALUES ('ADMIN');

INSERT INTO USER_PROFILE(type)
VALUES ('DBA');


/* Populate one Admin User which will further create other users for th
INSERT INTO APP_USER(sso_id, password, first_name, last_name, email)
VALUES ('sam','$2a$10$4eqIF5s/ewJwHK1p8lqlFOEm2QIA0S8g6./Lok.pQxqcxaBZY


/* Populate JOIN Table */
INSERT INTO APP_USER_USER_PROFILE (user_id, user_profile_id)
  SELECT user.id, profile.id FROM app_user user, user_profile profile
  where user.sso_id='sam' and profile.type='ADMIN';

/* Create persistent_logins Table used to store rememberme related stuf
CREATE TABLE persistent_logins (
    username VARCHAR(64) NOT NULL,
    series VARCHAR(64) NOT NULL,
    token VARCHAR(64) NOT NULL,
    last_used TIMESTAMP NOT NULL,
    PRIMARY KEY (series)
);
```

Note that we have inserted one user manually(we do need one Admin user to actually login and create further users for application). This is a real-world scenario. Notice the password which is encrypted form of password ‘abc125’. It’s generated using below mentioned utility class [it could even have been a script] which is used only and only to generate a password for one initial Admin user. It can well be removed from application.

```
package com.websystique.springsecurity.util;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
```

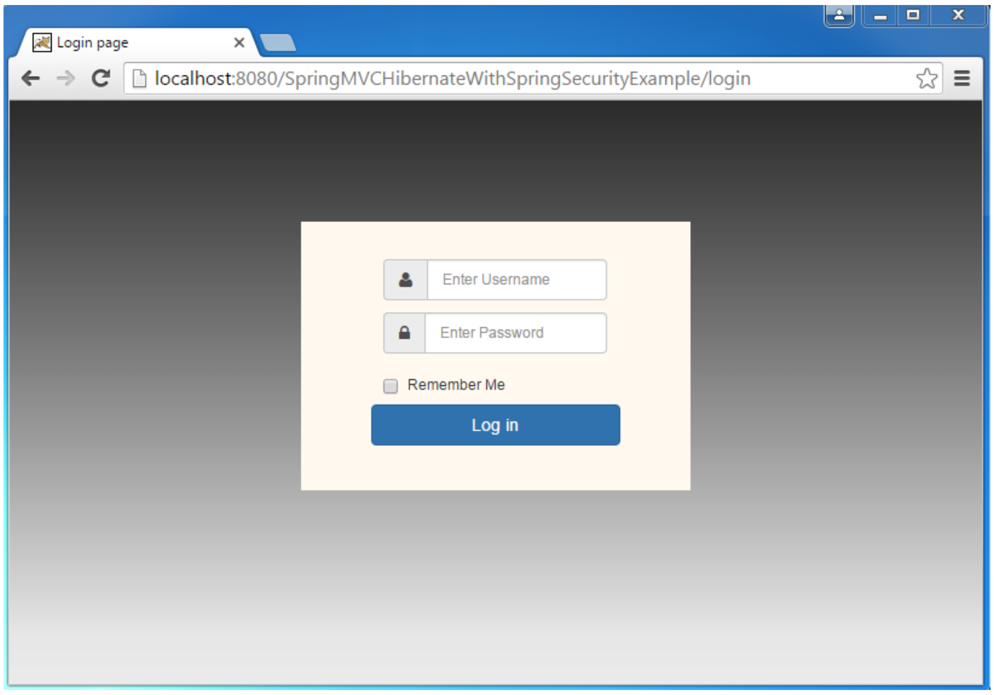
```
public class QuickPasswordEncodingGenerator {

    /**
     * @param args
     */
    public static void main(String[] args) {
            String password = "abc125";
            BCryptPasswordEncoder passwordEncoder = new BCryptPasswordE
            System.out.println(passwordEncoder.encode(password));
    }

}
```

Step 11: Build, deploy and Run Application

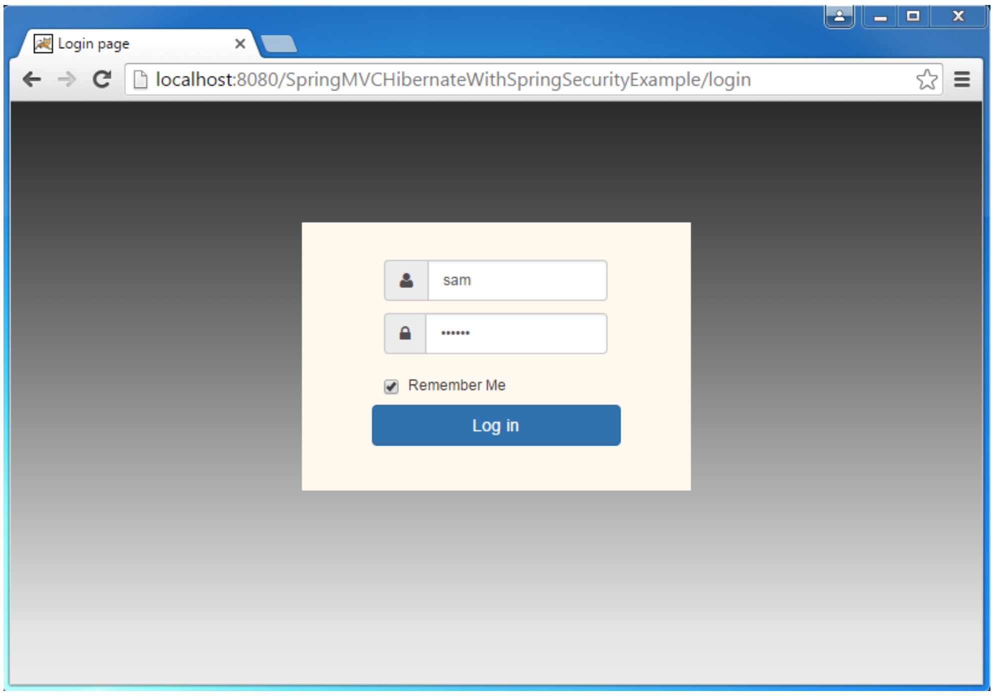
Now build the war (either by eclipse as was mentioned in previous tutorials) or via maven command line(`mvn clean install`). Deploy the war to a Servlet 3.0 container . Since here i am using Tomcat, i will simply put this war file into `tomcat webapps folder` and click on `start.bat` inside tomcat/bin directory.

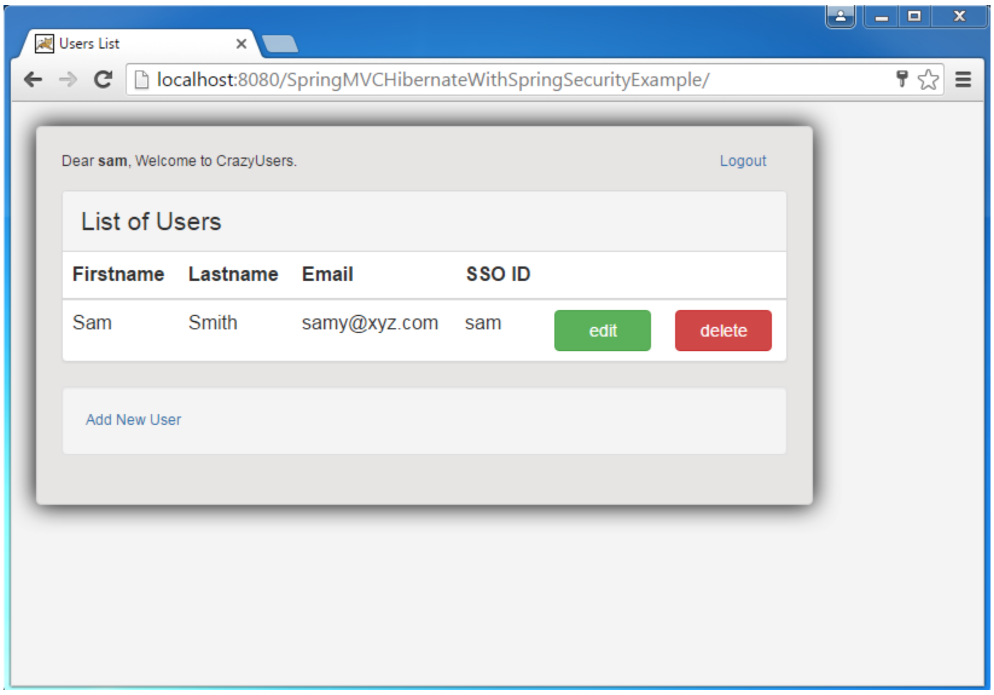
If you prefer to deploy from within Eclipse using tomcat: For those of us, who prefer to deploy and run from within eclipse, and might be facing difficulties setting Eclipse with tomcat, the detailed step-by-step solution can be found at : How to setup tomcat with Eclipse.

Open browser and browse at http://localhost:8080/SpringMVCHibernateWithSpringSecurityExample/


Login page
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/login
Enter Username
Enter Password
Remember Me
Log in

Login with User Sam & password abc125, check RememberMe as well.


Login page
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/login
sam
••••••
Remember Me
Log in


Users List

localhost:8080/SpringMVCHibernateWithSpringSecurityExample/

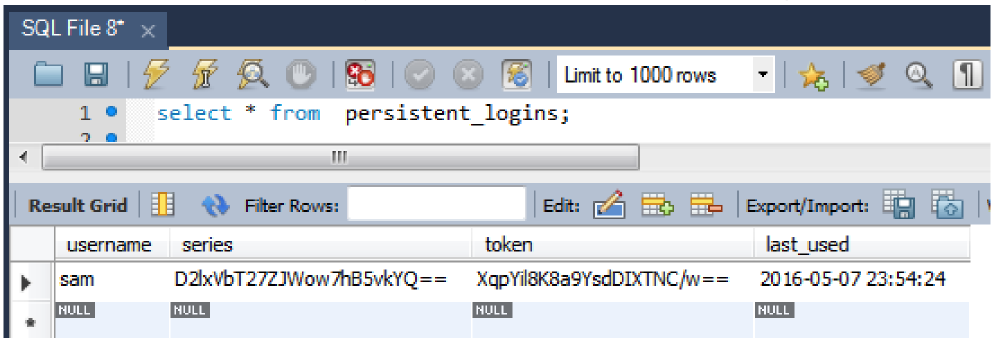
Dear **sam**, Welcome to CrazyUsers. Logout

List of Users

Firstname	Lastname	Email	SSO ID		
Sam	Smith	samy@xyz.com	sam	edit	delete

Add New User

Check database now.An entry should be made in persistent_logins table.


SQL File 8*

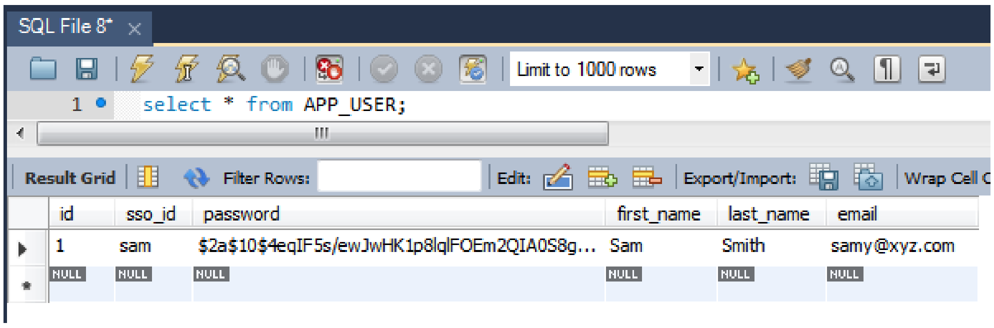
Limit to 1000 rows

```
select * from persistent_logins;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

username	series	token	last_used
sam	D2lxVbT27ZJWow7hB5vkYQ==	XqpYil8K8a9YsdDIXTNC/w==	2016-05-07 23:54:24
NULL	NULL	NULL	NULL

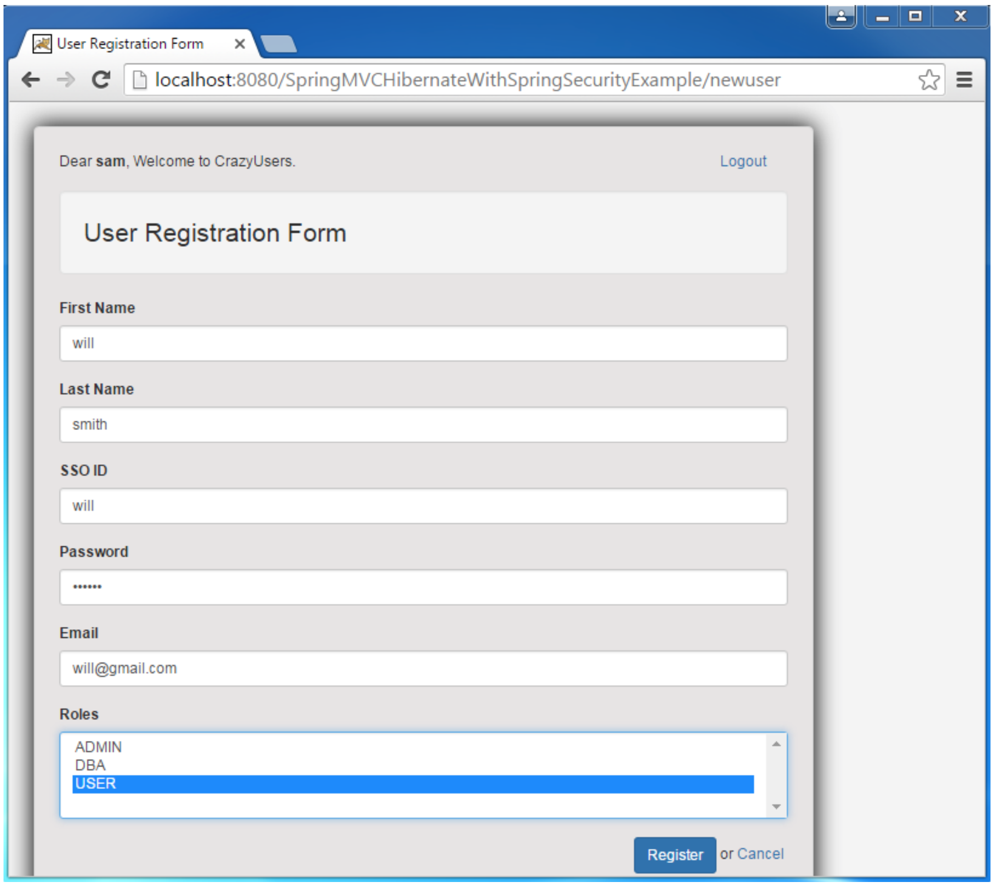
Nothing changes for APP_USER table though.



```
select * from APP_USER;
```

id	sso_id	password	first_name	last_name	email
1	sam	$2a$10$4eqIF5s/ewJwHK1p8lqlFOEm2QIA0S8g...	Sam	Smith	samy@xyz.com
NULL	NULL	NULL	NULL	NULL	NULL

Now click on ‘Add new user’ link. Add a user with ‘USER’ role.



localhost:8080/SpringMVCHibernateWithSpringSecurityExample/newuser

Dear **sam**, Welcome to CrazyUsers. Logout

User Registration Form

First Name: will

Last Name: smith

SSO ID: will

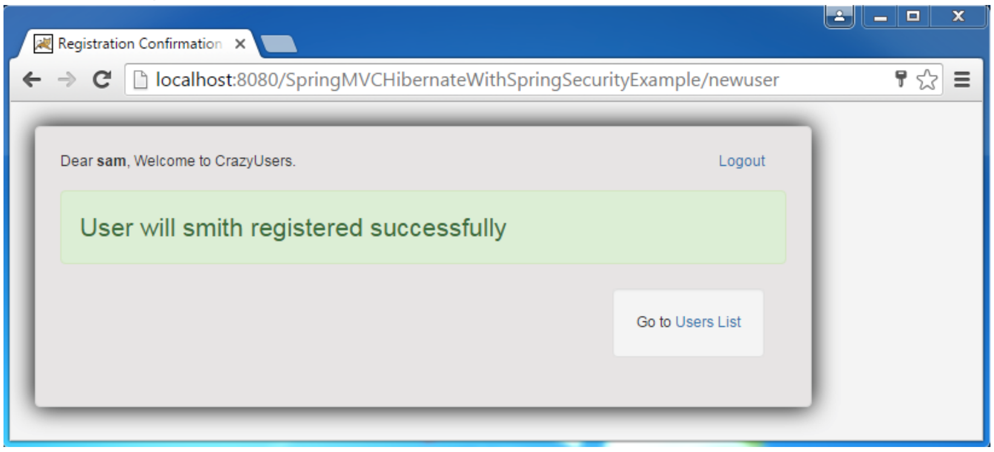
Password: ••••••

Email: will@gmail.com

Roles: ADMIN, DBA, USER (selected)

Register or Cancel

Click on Register, user should be added.



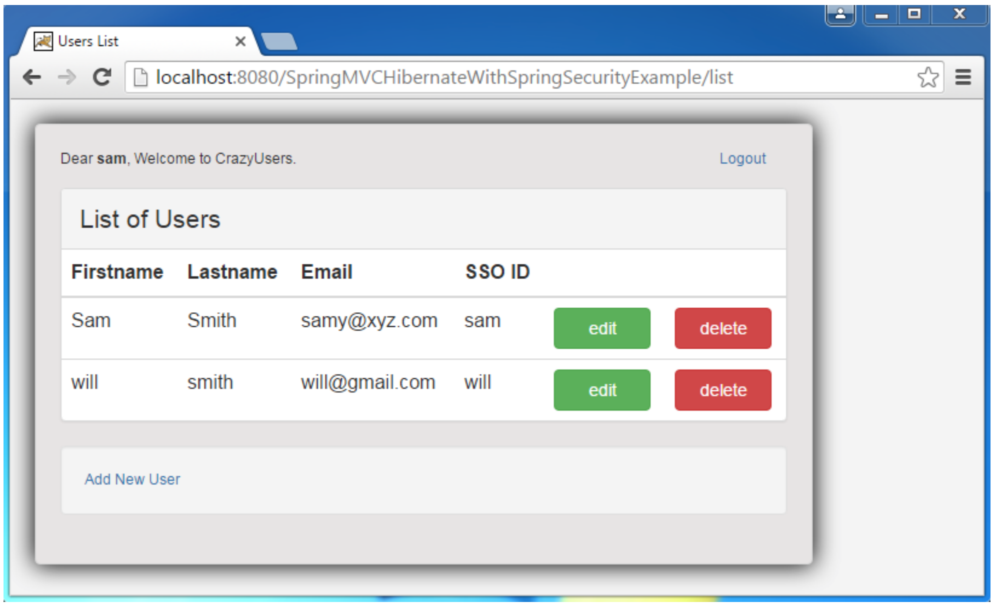
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/newuser

Dear **sam**, Welcome to CrazyUsers. Logout

User will smith registered successfully

Go to Users List

Click on ‘Users List’ link. You should see the newly added user.


Users List

localhost:8080/SpringMVCHibernateWithSpringSecurityExample/list

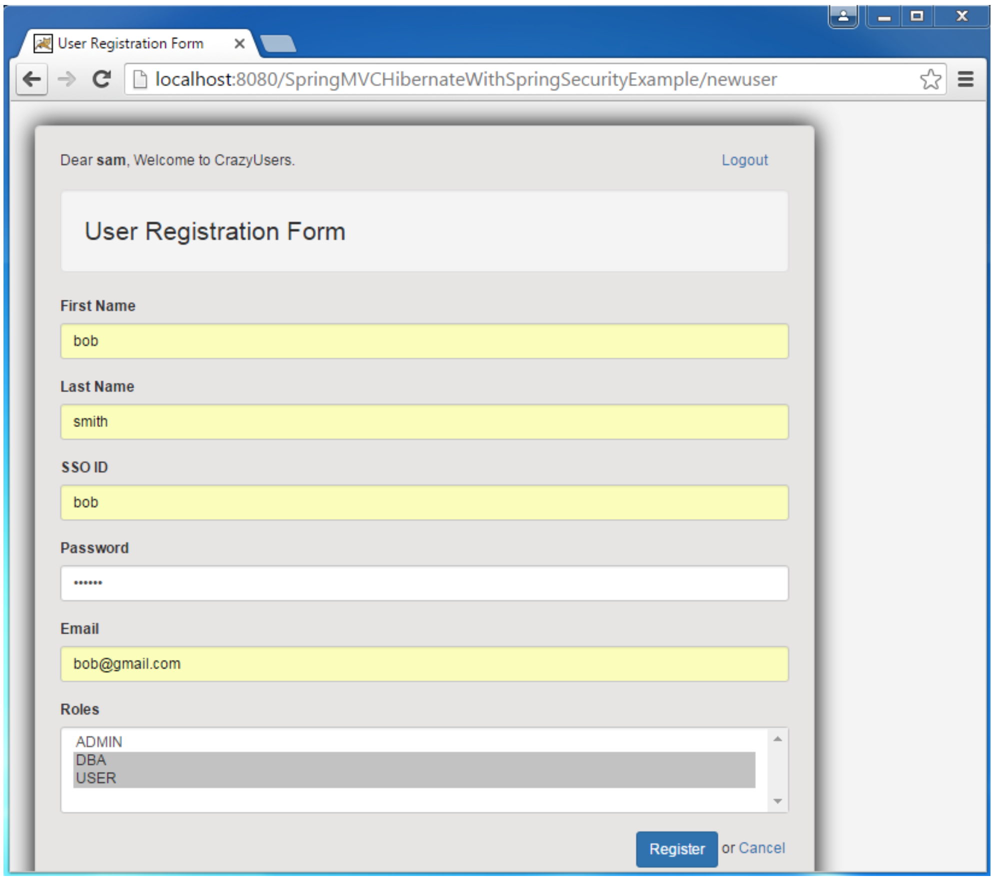
Dear **sam**, Welcome to CrazyUsers. Logout

List of Users

Firstname	Lastname	Email	SSO ID		
Sam	Smith	samy@xyz.com	sam	edit	delete
will	smith	will@gmail.com	will	edit	delete

Add New User

Add another user with DBA & USER role.


User Registration Form

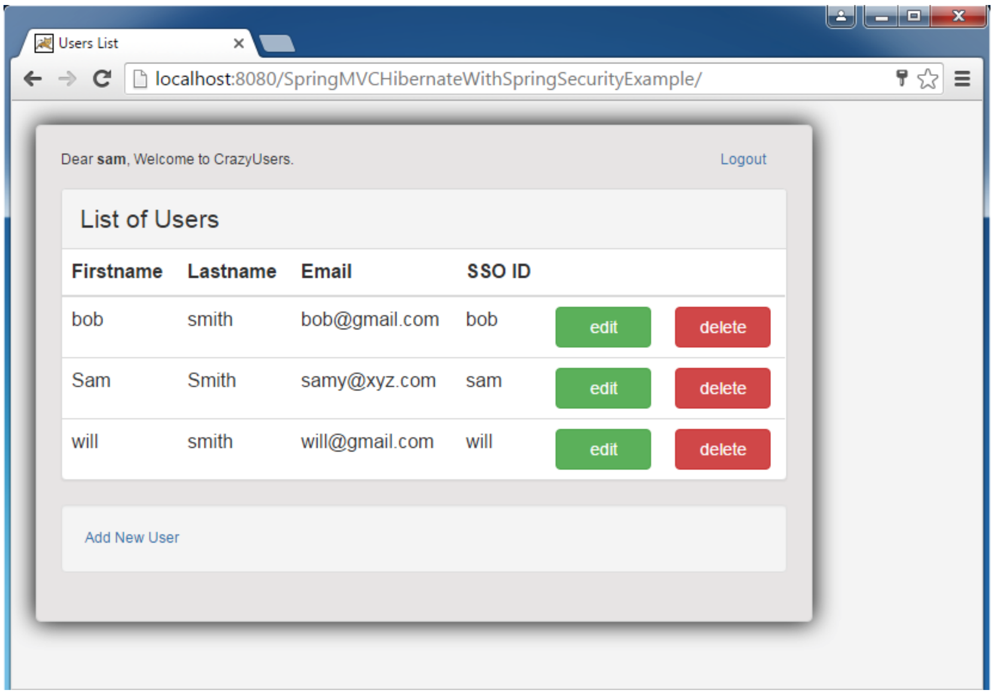
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/newuser

Dear **sam**, Welcome to CrazyUsers. Logout

User Registration Form

First Name	bob
Last Name	smith
SSO ID	bob
Password	••••••
Email	bob@gmail.com
Roles	ADMIN, DBA, USER

Register. Now check the list again.

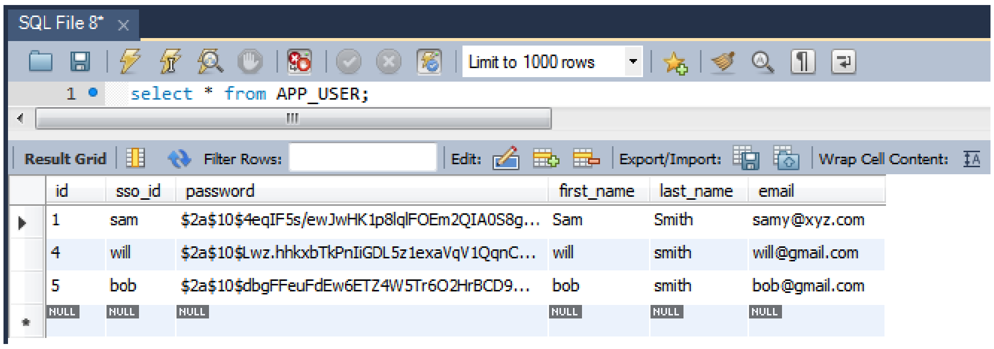


Dear **sam**, Welcome to CrazyUsers.

List of Users

Firstname	Lastname	Email	SSO ID		
bob	smith	bob@gmail.com	bob	edit	delete
Sam	Smith	samy@xyz.com	sam	edit	delete
will	smith	will@gmail.com	will	edit	delete

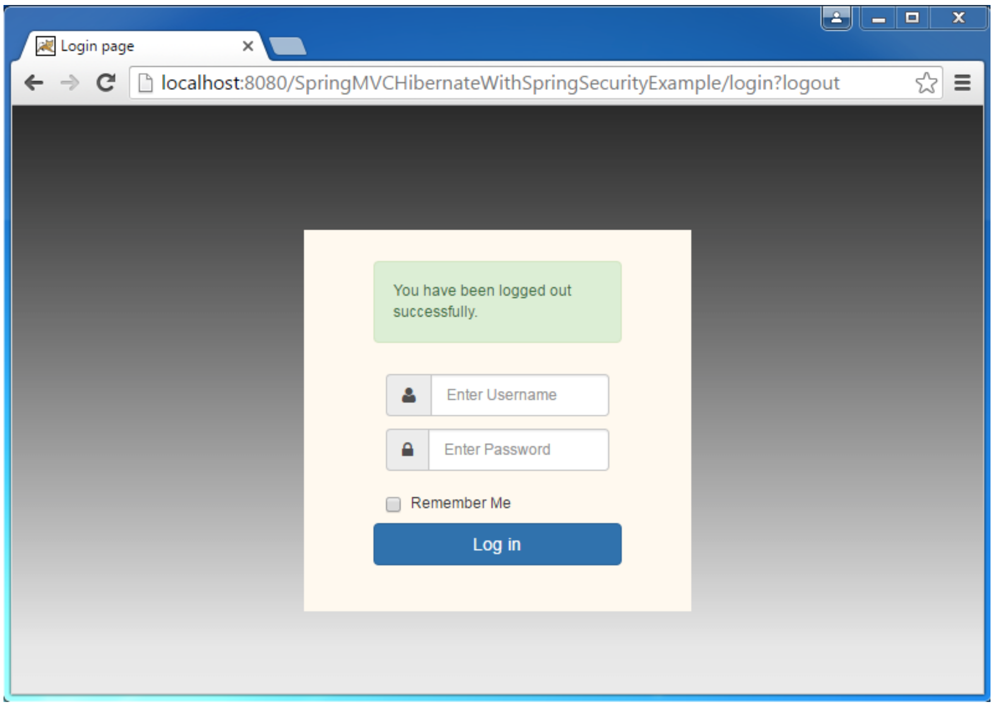
Verify APP_USER table.



```
select * from APP_USER;
```

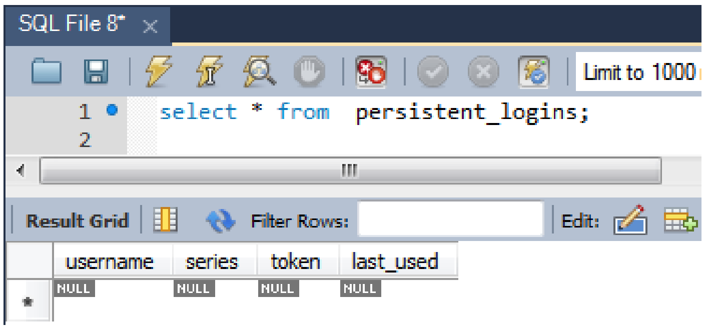
id	sso_id	password	first_name	last_name	email
1	sam	$2a$10$4eqIF5s/ewJwHK1p8lqlFOEm2QIA0S8g...	Sam	Smith	samy@xyz.com
4	will	$2a$10$Lwz.hhkxbTkPnIiGDL5z1exaVqV1QqnC...	will	smith	will@gmail.com
5	bob	$2a$10$dbgFFeuFdEw6ETZ4W5Tr6O2HrBCD9...	bob	smith	bob@gmail.com
NULL	NULL	NULL	NULL	NULL	NULL

Now logout.


Login page
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/login?logout
Enter Username
Enter Password
Remember Me
Log in

You have been logged out successfully.

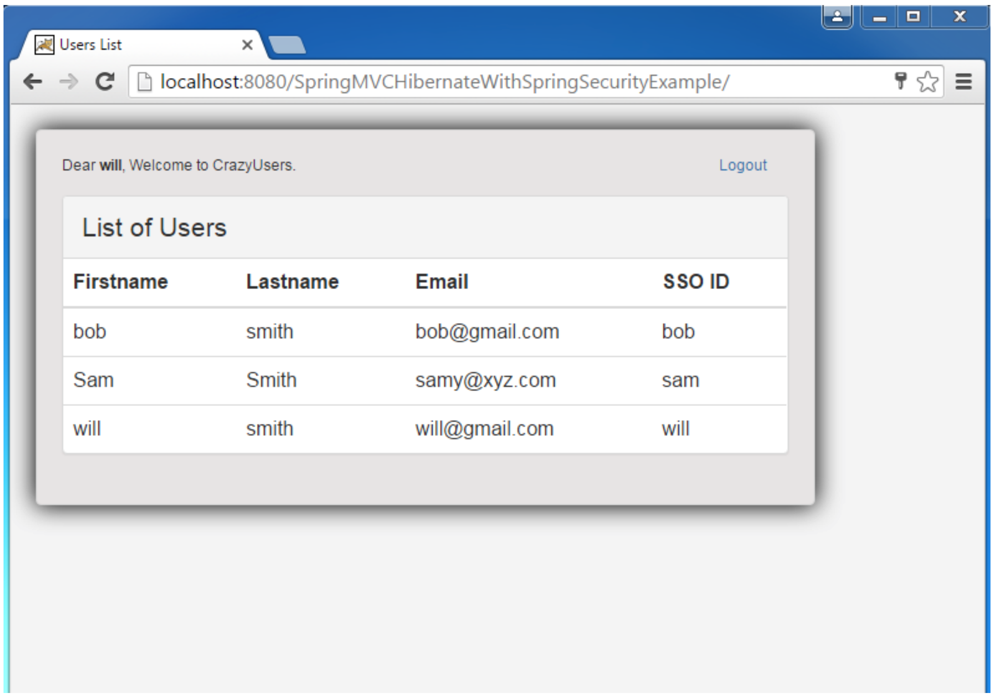
Check persistent_logins table, entry should be removed.



```
select * from persistent_logins;
```

username	series	token	last_used
NULL	NULL	NULL	NULL

Login with user ‘will’ which has ‘User’ role. No Add/Edit/Delete links are available to this user.



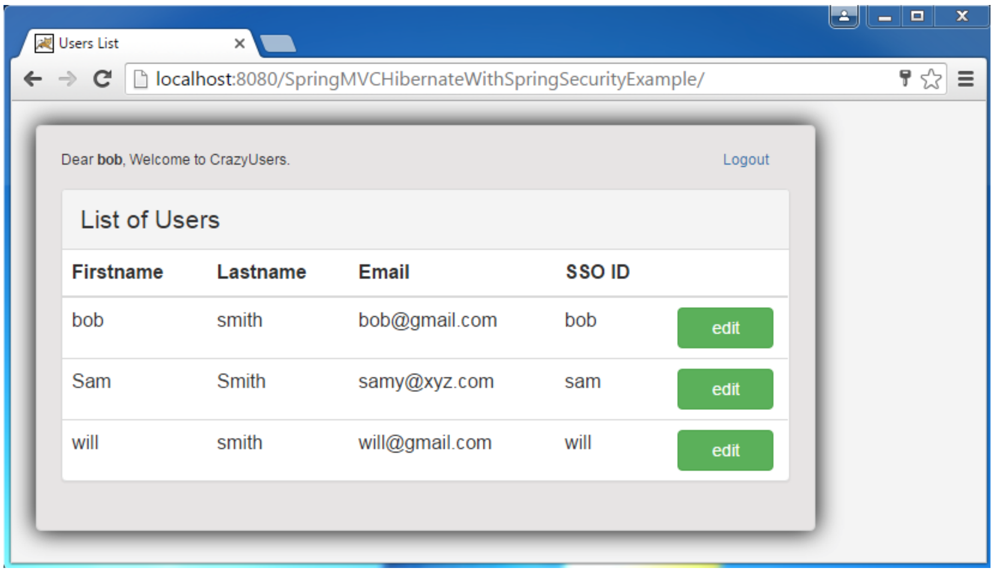
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/

Dear **will**, Welcome to CrazyUsers. Logout

List of Users

Firstname	Lastname	Email	SSO ID
bob	smith	bob@gmail.com	bob
Sam	Smith	samy@xyz.com	sam
will	smith	will@gmail.com	will

Now logout and login with ‘bob’. No Add/Delete links are available to this user.



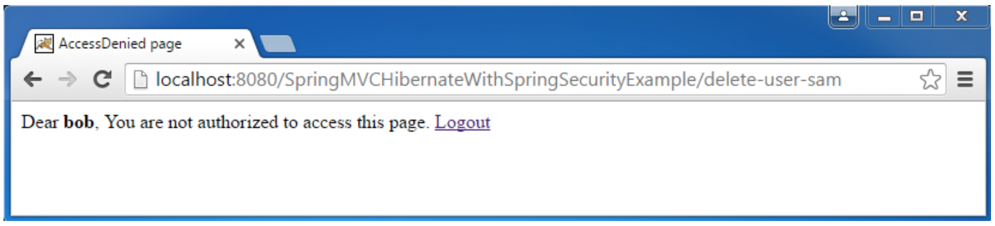
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/

Dear **bob**, Welcome to CrazyUsers. Logout

List of Users

Firstname	Lastname	Email	SSO ID	
bob	smith	bob@gmail.com	bob	edit
Sam	Smith	samy@xyz.com	sam	edit
will	smith	will@gmail.com	will	edit

Now try to manually type the delete URL in browser-bar and enter.You should see AccessDenied page.


AccessDenied page
localhost:8080/SpringMVCHibernateWithSpringSecurityExample/delete-user-sam

Dear **bob**, You are not authorized to access this page. Logout

That's it. As we saw, it's rather simple to integrate Spring Security with Spring MVC. Feel free to Comment, and suggest improvements.

Download Source Code

Download Now!

References

- Improved Persistent Login Cookie Best Practice
- Spring Security 4 Project Page
- Spring Security 4 Reference Manual
- Spring 4 Reference Manual

websystiqueadmin

If you like tutorials on this site, why not take a step further and connect me on Facebook , Google Plus & Twitter as well? I would love to hear your thoughts on these articles, it will help me improve further our learning process.

If you appreciate the effort I have put in this learning site, help me improve the visibility of this site towards global audience by sharing and linking this site from within and beyond your network. You & your friends can always link my site from your site on www.websystique.com, and share the learning.

After all, we are here to learn together, aren't we?

Related Posts:

1. **Spring Security 4 Method security using @PreAuthorize,@PostAuthorize, @Secured, EL**
2. **Spring 4 MVC+Hibernate Many-to-many JSP Example with annotation**
3. **Spring 4 MVC+JPA2+Hibernate Many-to-many-Example**
4. **Spring Security 4 Hibernate Role Based Login Example**

springmvc. permalink.

← Spring 4 MVC+AngularJS CRUD Application using ngResource

Spring 4 MVC+Apache Tiles 3 Example →

Sponsored

MAREMAGNUM

¡PARTICIPA!

Report ad

185 Comments **websystique** 1 **Login**

Recommend 9 Share Sort by Best

Join the discussion…

Siavonen • 8 months ago

I've tried to run this with some edits that I made in it and it's giving me this error message, what is possibly wrong with it and how can I fix it?

These are the errors that I think that are the main cause

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean found for dependency [fi.paino.painohallinta.converter.RoleToUserProfileConverter]: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=t

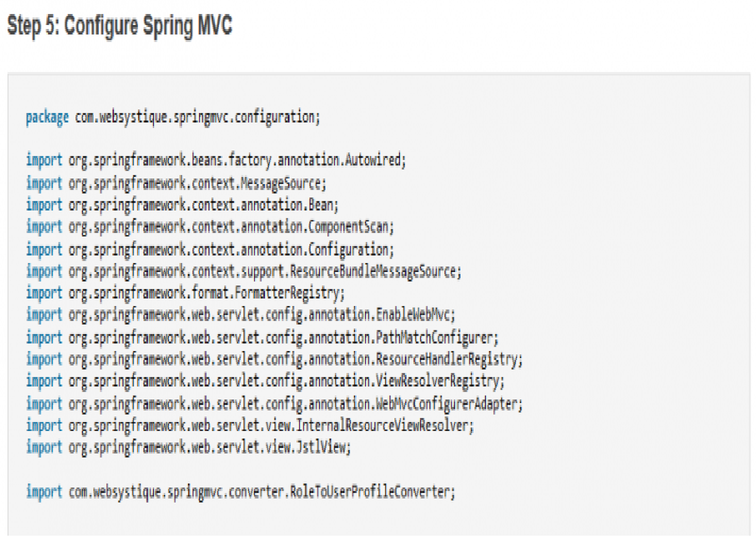
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'appConfig': Unsatisfied dependency expressed through field 'roleToUserProfileConverter'; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean found for dependency [fi.paino.painohallinta.converter.RoleToUserProfileConverter]: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations:

{@org.springframework.beans.factory.annotation.Autowired(required=t

1 ^ | ˅ • Reply • Share ›

Munarso ➦ Siavonen • 6 months ago

Make sure you have imported correct springframework.



Step 5: Configure Spring MVC

```
package com.websystique.springmvc.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.format.FormatterRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

import com.websystique.springmvc.converter.RoleToUserProfileConverter;
```

^ | ˅ • Reply • Share ›

websystique Mod ➦ Siavonen • 8 months ago

Hi, Please make sure that your package of your converter RoleToUserProfileConverter package is covered by @ComponentScan path.

^ | ˅ • Reply • Share ›

Siavonen ➦ websystique • 8 months ago

can you email me your skype or some other type of messenger that you use? I have no idea how to fix it my self. I've tried to make it work in so many different ways that I'm all out of ideas. :D

^ | ˅ • Reply • Share ›

Ergün Kargün • 17 days ago

hey thanks for tutorial, why registration form has not action attibute?

^ | ˅ • Reply • Share ›

Raj Pawar • 23 days ago

Hi,
When i download the source code and run it i am getting 404 error in login.
Can you please tell me what's the problem.
Thanks

^ | ˅ • Reply • Share ›

Comments continue after advertisement

Raj Pawar • 23 days ago

Hi,
Can anyone tell me why i getting 404 error when i run the application.

• Reply • Share ›

Alex • a month ago

@NotEmpty is already mean not null.
Why not use Spring Data JPA?

• Reply • Share ›

Kevin • a month ago

thanks a million

• Reply • Share ›

MiguelS • a month ago

Hi! Thanks for this tutorial, it's amazing! I've working on it the last 2 months and I have made some modifications... I Added a class named Card, and I'm trying to relate it with the User class. I have created a table called app_user_card with two primary keys: id_user and id_card, and 2 more columns: quantity and price. The foreign keys are id in user and card tables. I don't know how to do that in Hibernate. I tried to do it like the table app_user_user_profile, but it didn't work. I have always a 404 error and this this log in the console "Error creating bean with name 'appConfig': Injection of autowired dependencies failed;" over and over again... Does anybody know how can I fix It? I have been loking in stackOverflow the last 2 days, but the solutions they gave doesn't work with me... Thanks for your time!

• Reply • Share ›

楊文齊 • 2 months ago

Hi, I rewrite the security configuration class from this tutorial in order to provide two entry points with security, one for web user, the other one for mobile connection. Login page works fine, and it redirect well when user not login yet. But the response after summit authentication to login process url is 404 page not found. How do I do to fix it? Here is the security confiruation code below:

package com.websystique.springmvc.s...;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import
org.springframework.security.authentication.AuthenticationTrustResolv
import
org.springframework.security.authentication.AuthenticationTrustResolv

see more

Reply • Share ›

mohammad iliyas • 2 months ago

i am facing problem while converting annotation to xml configuration on FormatterRegistry .So please tell me how to solve this.thank you.

Reply • Share ›

Comments continue after advertisement

Millonaria de Madrid expone cómo gana 563 €/hora desde casa

¡No vas a creer cómo lo hace!

Learn More

Sponsored by **Daily News**

Report ad

Prabhat Singh • 2 months ago

thanks a lot, it's best example for spring mvc and spring security tutorial....

Reply • Share ›

Vaibhav Kadu • 2 months ago

can anyone help me to know, How login functionality works here. I'm confused.

Reply • Share ›

Vaibhav Kadu • 2 months ago

can anyone help me to know, How login functionality works here. I'm confused bcz it does nit find the records explicitly like using Find or select query..

Reply • Share ›

Tushar Girase • 2 months ago

I am getting following exception
I tried to add a property in application.properties file as "spring.jpa.hibernate.ddl-auto=create"

from the last exception its clear that there isnt any table but hibernate should create one but above property also not resolved the issue please reply back.

exception
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.hibernate.exception.SQLGrammarException: could not extract ResultSet

root cause
org.hibernate.exception.SQLGrammarException: could not extract

ResultSet

root cause
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'websystique.user_profile' doesn't exist

^ | ⌄ • Reply • Share ›

Tushar Girase ➦ Tushar Girase • 2 months ago

The problem has been resolved. I haven't added the property in the HibernateConfiguration it was only in properties file. I added to HibernateConfiguration.java and changed the property key respectively.

properties.put("hibernate.hbm2ddl.auto", environment.getProperty("hibernate.hbm2ddl.auto"));

Thanks Admin for wonderful tutorial. Keep posting.

^ | ⌄ • Reply • Share ›

CommonSense • 2 months ago

I appreciate the work you put into these.

I had a problem with a validation exception but it was my lib problems pre-loaded in tomcat.

Thanks

^ | ⌄ • Reply • Share ›

Karthikeyan Balasubramaniyan • 2 months ago

Hi Websystique,

Thanks for the tutorial.
my environment:
<properties>
<springframework.version>4.3.3.RELEASE</springframework.version
<springsecurity.version>4.0.4.RELEASE</springsecurity.version>
<hibernate.version>5.1.5.Final</hibernate.version>
<postgresql.version>42.0.0.jre7</postgresql.version>
</properties>
I downloaded your code and try to run it locally. I am not able to login since isCurrentAuthenticationAnonymous() returns true even for a first time user. Could you please advice..

Thanks a lot...

^ | ˅ • Reply • Share ›

Comments continue after advertisement

Rishi Kumar • 3 months ago

```
[http-nio-8081-exec-7] DEBUG org.hibernate.loader.Loader - Result set row: 0
16:42:41.089 [http-nio-8081-exec-7] DEBUG org.hibernate.loader.Loader - Result row: EntityKey[com.bunnty.model.User#1]
16:42:41.089 [http-nio-8081-exec-7] DEBUG org.hibernate.engine.internal.TwoPhaseLoad - Resolving associations for [com.bunnty.model.User#1]
16:42:41.090 [http-nio-8081-exec-7] DEBUG org.hibernate.engine.internal.TwoPhaseLoad - Done materializing entity [com.bunnty.model.User#1]
16:42:41.093 [http-nio-8081-exec-7] DEBUG org.hibernate.transform.DistinctResultTransformer - Transformed: 1 rows to: 1 distinct results
16:42:41.093 [http-nio-8081-exec-7] DEBUG org.hibernate.engine.transaction.spi.AbstractTransactionImpl - committing
16:42:41.094 [http-nio-8081-exec-7] DEBUG
```

see more

^ | ˅ • Reply • Share ›

Zulfi Malik • 3 months ago

Hi Websystique ,
First of all thank so you much. This is one of the best example I found related to Spring MVC security.

I have one question, I have AngularJs based application running separately (node js, 3000 localhost). I tried to make an api call to the spring mvc controller , but some how I am unable to login with. I am getting session null and isCurrentAuthenticationAnonymous() = = false every time. The code for http request in service is as;

```
var deferred = $q.defer();
var config = {
ignoreAuthModule: 'ignoreAuthModule',
headers: {'Content-Type': 'application/x-www-form-urlencoded'}
};
var url =
```

'http://localhost:8080/SpringMVCHibernateWithSpringSecurityExample
;

see more

Reply • Share ›

Luis Alberto Castrillo Velilla • 3 months ago

This is a form of organized code to perform security settings, much more understandable:

@Override
protected void configure(HttpSecurity http) throws Exception {
http.formLogin()
.loginPage("/login")
.permitAll()
.failureUrl("/login?login_error=true")
.loginProcessingUrl("/authenticate").usernameParameter("j_username'

http.logout()
.logoutUrl("/j_spring_security_logout")
.logoutSuccessUrl("/login")
.invalidateHttpSession(true)
.permitAll();

http.csrf().disable();

http.exceptionHandling().accessDeniedPage("/denied");

http.authorizeRequests().antMatchers("/home**").authenticated();
http.authorizeRequests().antMatchers("/vista/*/table.htm",
"/vista/*/report/*").access("hasAnyRole('ROLE_Administrator','ROLE_E
http.authorizeRequests().antMatchers("/rest/*/delete.htm",
"/rest/*/update.htm").access("hasRole('ROLE_Administrator')");
}

Regards

Reply • Share ›

Natalia Dranchuk • 3 months ago

Hi!
First of all thanks a lot for this amazing tutorial, it helped me at the beginning of discovering spring)
I'm trying to implement a service based on this tutorial using angular 2, and I faced some problems with login. Could you please explain this procedure more detailed? What is the correct way to build a login, accepting json? and what is name="${_csrf.parameterName}" value="${_csrf.token}" in login.jsp and how can I manage it using angular?
Hope for your help)

Reply • Share ›

Zulfi Malik ➦ Natalia Dranchuk • 3 months ago

I am having same difficulties too. Unable to login via my angularJs application which is running separately. Were you able to find a way to login with Angular ? if so, then can you please assist me in this regard?

Reply • Share ›

Swapnil More • 3 months ago

Hello, I am new to spring security concept and have gone through your example and i had created db still it does authenticate my credentials. so please can you help me regarding it.

Reply • Share ›

Alaa_bourouissi • 3 months ago

Hello , I added another class Car.java in relation with User.java , so a user can have many cars , in the User.java :
@OneToMany(mappedBy="user",cascade=CascadeType.ALL)
private Set<car> cars = new HashSet<car>();

in the Car.java :

@NotNull
@ManyToOne(optional=false)
@JoinColumn(name="user_fk")
private User user;

Also I created StringToUser.java class (like RoleToUserProfileConverter class)

In the carController I just do like the AppController (same things) :

@RequestMapping(value={"/newCar"},method=RequestMethod.GET)
public String newCar(ModelMap model){

see more

Reply • Share ›

throne • 4 months ago

Hello websystique...you are doing a great job here. i have learnt alot...and even more. i tried implementing the hibernate code in this project but get a stackoverflow error each time i try running the update method. what should i do?

Reply • Share ›

websystique Mod → throne • 3 months ago

Hello Throne, what exactly is the error you are getting? What version of Hibernate, spring, spriung-security are you using? You can always paste the full exception here.

Reply • Share ›

Pablo • 4 months ago

Hi,

I need to get all users that has type as 'DBA' or 'USER', for example.

How can I get this list?

Reply • Share ›

websystique Mod ➦ Pablo • 3 months ago

Hi Pablo, this is more of a query to get specific users.Write a new method in your DAO and expose that using service layer which would be accessed on GUI/elsewhere, it should be straightforward.

^ | ˅ • Reply • Share ›

Rahil Baig • 4 months ago

hi sir,

I have followed the tutorial to integrate in my project. I am able to login and the user gets validated also.Its running perfect with one small issue

1) I have produced this scenario by logging same user from two different browsers. Login was successful. however i faced the following exception When I Tried to logout. org.hibernate.NonUniqueResultException: query did not return a unique result: 2

For above problem one of the person has given the solution,i have mentioned below

We could make username key unique by combination of deviceId+username but i just overrides the persistent repository Service class and use series as key to identify the user. That solved my problem. Thanks for your feedback.

Questions of mine?

1)Here,how to get the device id, and which one needs to override?(if you share that piece of code it will be great help)

2) As your giving Add New Users for the admin role ,but i want to make that available for login page,am tring to make the login application with new user registration,where user can login if he is not registered then he can registered with new user name and password...Thanks in advance

^ | ˅ • Reply • Share ›

websystique Mod ➦ Rahil Baig • 3 months ago

Hello Rahil, first of all, username itself should be unique for login. Instead of simple name as 'sam', an email id can be used which is guranteed to be unique, it also saves you from using other tricks like deviceId to enforce uniqueness. Now, for handling the multiple login at same time, you would need to tweak HibernateTokenRepositoryImpl and PersistentLogin, so that while login, you should first check if there is already an entry in persistent login table, if yes, you should inform [via some message notification] to the user that there is already a user logged it with this id/username. At this moment, you might want to give him more flexibility asking if he wants to logout from other devices. If yes, you can use that trigger to invalidate the existing session on other device, and let the user create a new session.

For your other question, you can make seperate login+register pages, and redirect to appropriate page accordingly. It's more of a management and should be straightforward to implement,

• Reply • Share ›

nnson1610 • 4 months ago

Hi websystique, thank you a lot for this example, but when I run it, I get an error. Please help me!

see more

• Reply • Share ›

websystique Mod ➦ nnson1610 • 3 months ago

Hi , you have a classnotfound error for 'CorsFilter'. Which version of Spring are you using? This class is available from 4.2 onwards.

• Reply • Share ›

Alaa_bourouissi • 4 months ago

please , where can I find the call to the messages.properties file ?? How we use this file ??

• Reply • Share ›

websystique Mod ➦ Alaa_bourouissi • 3 months ago

Hello, There is no such call to messages.properties, instead It's all standard Spring configuration which enable us to use properties file to declare validation or error messages in seperate files as part of internationalization.

Look at AppConfig:

```
/**
* Configure MessageSource to lookup any validation/error message in internationalized property files
*/
@Bean
public MessageSource messageSource() {
```

ResourceBundleMessageSource messageSource = new
ResourceBundleMessageSource();
messageSource.setBasename("messages");
return messageSource;
}

Now, for any error/validation msg lookup, Spring will consult messages.properties file on classpath.

Reply • Share ›

Nitesh Sharma • 4 months ago

@Column(name="TYPE", length=15, unique=true, nullable=false)
private String type = UserProfileType.USER.getUserProfileType();
what is the use of UserProfileType enum while i am using DB

Reply • Share ›

websystique Mod Nitesh Sharma • 3 months ago

Enum usage in this example has nothing to do with DB but a practice to declare a set of items which are conceptually similar. You can get rid of them and use straightaway strings.

Reply • Share ›

Nitesh Sharma • 4 months ago

@Column(name="TYPE", length=15, unique=true, nullable=false)
private String type = UserProfileType.USER.getUserProfileType();
what is the use of UserProfileType enum ?

Reply • Share ›

Raichand Ray • 4 months ago

Hi,
I am using JSF not Spring MVC.Please publish a similar article with JSF.
Thanks
Raichand

Reply • Share ›

Alaa_bourouissi • 4 months ago

Hello , the application works fine , butit's not possible to login with Username:sam and Password:abc125 , the message : Dear sam, You are not authorized to access this page (Access Denied) , I added the row of sam in the database

Reply • Share ›

Samuel T • 5 months ago