**SpringHibernateSecurity changes**

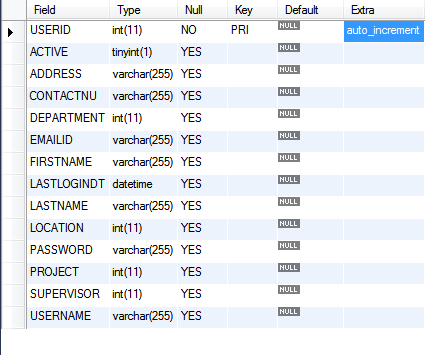
Please find the changes which are included in Springhibernatesecurity project:

1. **Database changes:**

Our application relies on a custom database schema to obtain its users.   
The database contains three tables: user, user\_role and role tables.

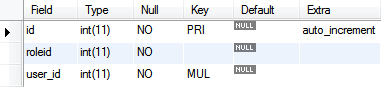
**User table:**

The *user* table contains personal information of each user. Notice the password values are hashed using Md5.



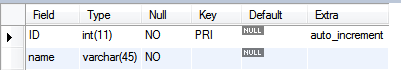
**user\_role table:**

The user\_role table contains roleid and user\_id mapping.



**Roles table:**

The *roles* table contains roleid and its rolename.



# Layers:

## Domain Layer:

## This layer contains three domain classes, *User, UserRole* and *Role*. They represent our database tables, *user, user\_role*  and *roles* respectively.

## Controller Layer:

## This layer contains two controllers, *AccessController* and *RoleController*.

## *AccessController* is responsible for handling access related requests, mainly login and logout requests.

## *RoleController* is responsible for handling role related requests.

## Service Layer:

This layer contains a  *CustomUserDetailsService and RoleService*.

* *CustomUserDetailsService:* Its main purpose is to retrieve user information from our custom database and translate that user information into a format that Spring Security understands.

logic here is:

1. We must implement *UserDetailsService* because we have a custom database
2. *loadUserByUsername* method must return an object that implements the [*UserDetails*](http://static.springsource.org/spring-security/site/docs/3.1.x/apidocs/org/springframework/security/core/userdetails/UserDetails.html)interface
3. We must map our domain *com.spring.hibernate.model.User* to

*org.springframework.security.core.userdetails.User*

1. We must map numerical roles as *SimpleGrantedAuthority* objects
2. We are responsible for interpreting what each numerical role value represents
3. *ROLE\_USER* and *ROLE\_ADMIN* are arbitrary values we assigned to numerical values

Operations included in this service layer:

* public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
* com.spring.hibernate.model.User domainUser = userDAO.getUser(username);

- here you will get the user informations with username entered in login page.

-Later we are mapping with spring security core user.

- And also we get the roles associated with the user by quering user\_role table with the user entered getAuthorities(domainUser.getUserRole().getRoleid()));

**public** Collection<? **extends** GrantedAuthority> getAuthorities(Integer role) {

List<GrantedAuthority> authList = *getGrantedAuthorities*(getRoles(role));

*logger*.info("inside CustomUserDetailsService:getAuthorities"+authList.size());

**return** authList;

}

-This method will return the list of roles assigned to the user logged in by getting the roles with roleid assigned to the logged in user .

* *RoleService: This includes Role CRUD operations as mentioned below*

* **public** List<Role> findRoleByName(String name);
* **public** **void** createRole(Role role);
* **public** List<Role> getRoleView();
* **public** **void** deleteRole(String name);

**DAO Layer:**

* Created the RoleDAO

# Configuration

* mvc-dispatcher-security.xml :

changes:

* 1. <security:http pattern=*"/resources"* security=*"none"* />

This means the path */resources* should be ignored by Spring Security; therefore it will not be secured. Mainly because these are CSS, and JavaScript files that don't need to be secured.

* 1. <security:http auto-config=*"true"* use-expressions=*"true"*>

<security:intercept-url pattern=*"/login"*

access=*"permitAll"* />

<security:intercept-url pattern=*"/logout"*

access=*"permitAll"* />

<security:intercept-url pattern=*"/denied"*

access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/"*

access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/user"*

access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/role"*

access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/admin"*

access=*"hasRole('ROLE\_ADMIN')"* />

Here we declare URL patterns to be protected. By using SPEL *hasRole* and *permitAll*

* 1. <security:form-login login-page=*"/login"*

authentication-failure-url=*"/login/failure"* default-target-url=*"/"* />

<security:access-denied-handler

error-page=*"/denied"* />

<security:logout invalidate-session=*"true"*

logout-success-url=*"/logout/success"* logout-url=*"/logout"* />

</security:http>

Hope this change is self explanatory.

* 1. <security:authentication-manager>

<security:authentication-provider

user-service-ref=*"customUserDetailsService"*>

<security:password-encoder hash=*"md5"* />

</security:authentication-provider>

</security:authentication-manager>

<bean id=*"customUserDetailsService"* class=*"com.spring.hibernate.service.CustomUserDetailsService"* />

* *authentication-manager:* registers an *AuthenticationManager* that provides authentication services.
* *authentication-provider:* this is a shorthand for configuring a *DaoAuthenticationProvider* which loads user information from a *UserDetailsService*.
* *user-service-ref:* this allows us to declare a custom *UserDetailsService*
* *password-encoder:* this allows us to declare MD5 password encoders.
* mvc-dispatcher-servlet.xml:

changes:

<bean id=*"userDao"* class=*"com.spring.hibernate.dao.UserDAOImpl"*></bean>

<bean id=*"roleDao"* class=*"com.spring.hibernate.dao.RoleDAOImpl"*></bean>

<bean id=*"userService"* class=*"com.spring.hibernate.service.UserServiceImpl"*></bean>

<bean id=*"roleService"* class=*"com.spring.hibernate.service.RoleServiceImpl"*></bean>

<bean id=*"adminService"* class=*"com.spring.hibernate.service.AdminServiceImpl"*></bean>

* hibernate.cfg.xml

<hibernate-configuration>

<session-factory>

<mapping class=*"com.spring.hibernate.model.User"* />

<mapping class=*"com.spring.hibernate.model.Role"* />

<mapping class=*"com.spring.hibernate.model.UserRole"* />

</session-factory>

</hibernate-configuration>

* web.xml:

Below change was included to ignore routing pattern for css files.

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.css</url-pattern>

</servlet-mapping>

**JSP FILES:**

Included below jsp files for role, admin, user and access under jsp folder. And included validate plugin in roles page to validate the rolename entered for role creation, search and delete pages.

