



WebSystique

learn together

Spring 4 MVC+Hibernate Many-to-many JSP Example with annotation

🕒 August 1, 2015 👤 websystiqueadmin

This post demonstrates Hibernate Many-to-many example, with join table in Spring MVC CRUD Web application. We will discuss managing Many-to-Many relationship both in views and back-end. We will perform Create, Update, Delete & Query all using application Web interface.

This posts makes use of

Spring `org.springframework.core.convert.converter.Converter` interface, which helps us with mapping Id's of items to actual entities in database.

WebSyst

G+ Follow

+ 82

Recent Posts

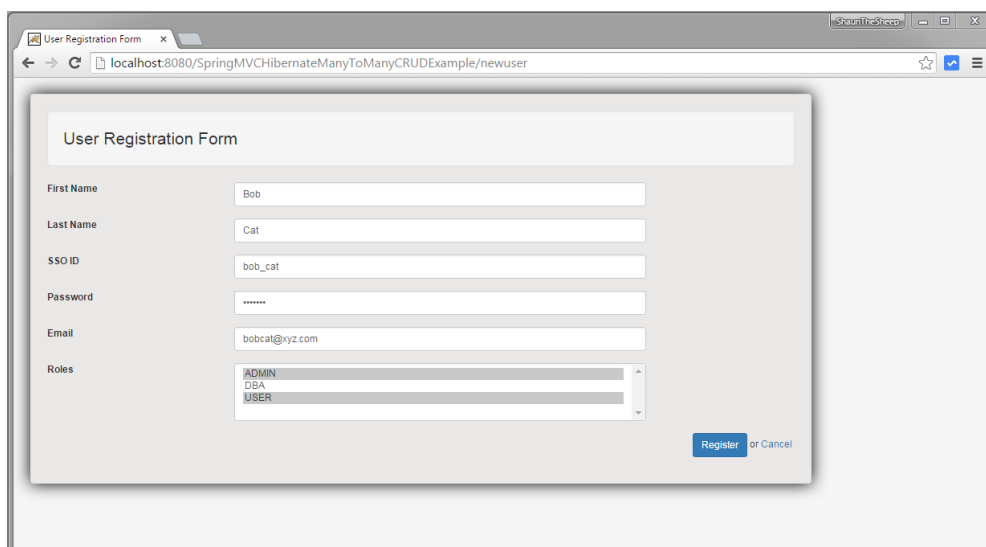
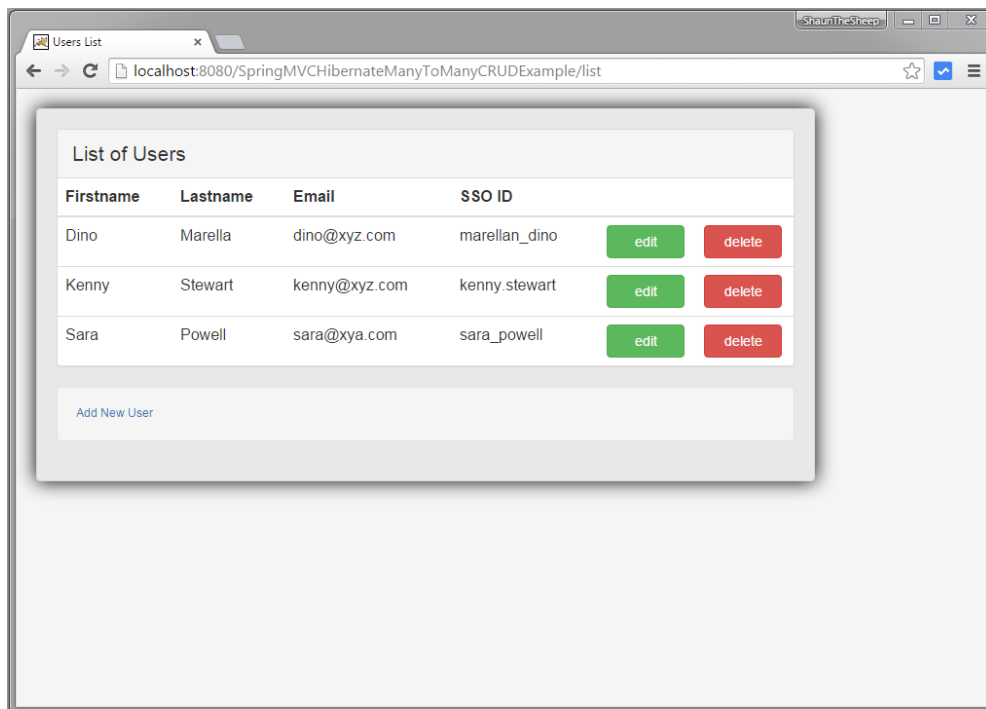
Spring 4 MVC+AngularJS
CRUD Application using
ngResource

Angularjs Server
Communication using
ngResource-CRUD
Application

AngularJS Custom-Directives
controllers, require option
guide

AngularJS Custom-Directives
transclude, ngTransclude
guide

AngularJS Custom-Directives
replace option guide



Complete example with detailed explanation is presented below.

Other interesting posts you may like

- [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+MySQL+Maven integration example](#)
- [Spring MVC 4+Hibernate 4+MySQL+Maven integration + Testing example using annotations](#)

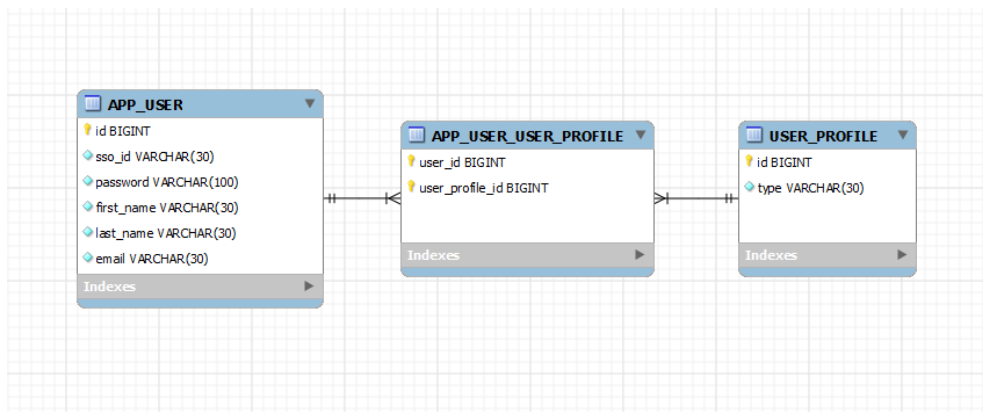
- [Spring MVC4 FileUpload-Download Hibernate+MySQL Example](#)
- [Spring MVC 4 Form Validation and Resource Handling](#)
- [TestNG Mockito Integration Example Stubbing Void Methods](#)
- [Maven surefire plugin and TestNG Example](#)

Following technologies being used:

- Spring 4.1.7.RELEASE
- Hibernate Core 4.3.10.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 JUNO Service Release 2

Let's begin.

Step 1. Create schema for Many-To-Many association with Join table



APP_USER : Contains Users. A User can have several profiles[USER,ADMIN,DBA].

USER_PROFILE : Contains User Profiles. A Profile can be linked to several users.

APP_USER_USER_PROFILE : It's a Join table linking APP_USER &

USER_PROFILE in Many-To-Many relationship.

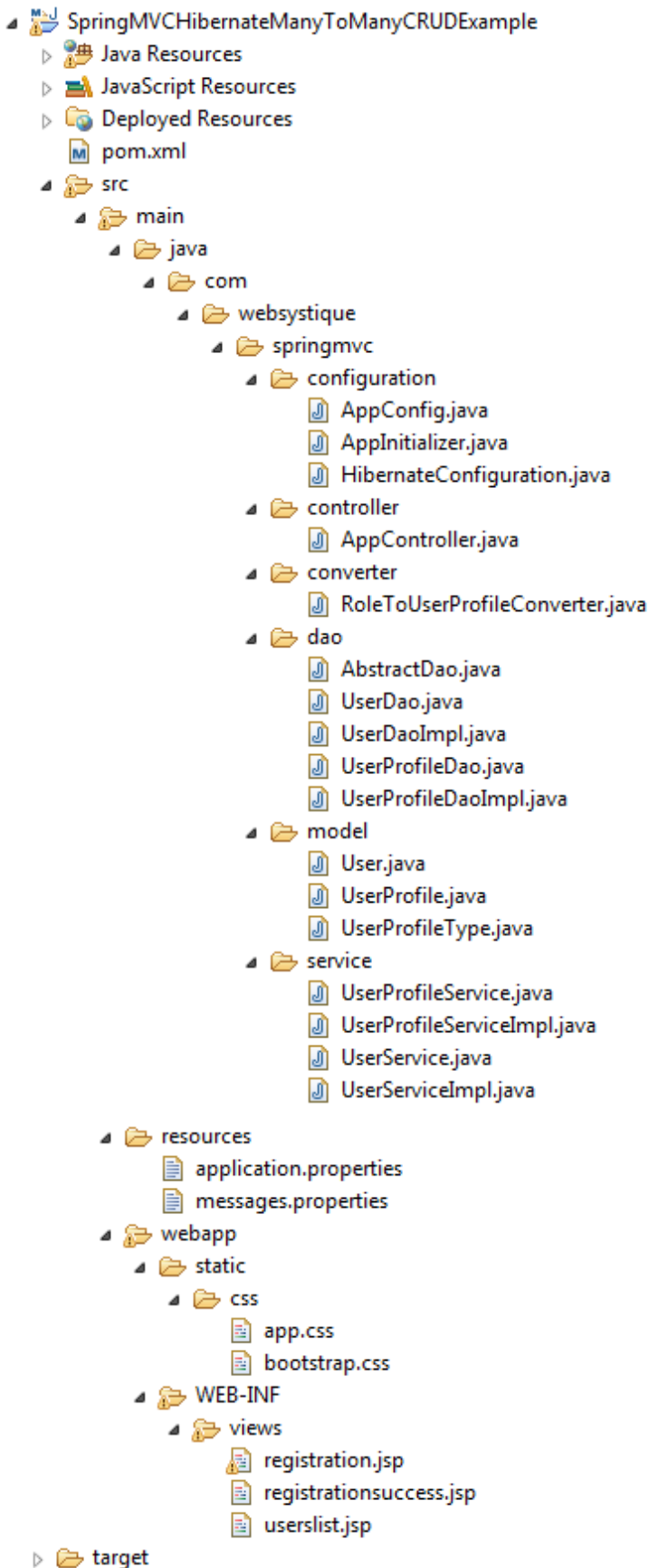
For demonstration purpose, We will discuss Many-to-Many unidirectional [User to UserProfile] setup in this example.

```
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)  
);  
  
create table USER_PROFILE(  
    id BIGINT NOT NULL AUTO_INCREMENT,  
    type VARCHAR(30) NOT NULL,  
    PRIMARY KEY (id),  
    UNIQUE (type)  
);  
  
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 (id),  
    CONSTRAINT FK_USER_PROFILE FOREIGN KEY (user_profile_id) REFERENCES USER_PROFILE (id)  
);  
  
/* 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');  
  
commit;
```

For any help with MySQL & database setup , please visit [MySQL installation on Local PC](#).

Step 2: Create the directory structure

Following will be the final project structure:



Step 3: Update pom.xml to include required dependencies

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0" xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.websystique.springmvc</groupId>
```

```

<artifactId>SpringMVCHibernateManyToManyCRUDExample</artifactId>
<packaging>war</packaging>
<version>1.0.0</version>
<name>SpringMVCHibernateManyToManyCRUDExample</name>

<properties>
  <springframework.version>4.1.7.RELEASE</springframework.ver
  <hibernate.version>4.3.10.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>

  <!-- 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>

  <!-- 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>
<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-war-plugin</artifactId>
        <version>2.4</version>
        <configuration>
          <warSourceDirectory>src/main/webapp</warSou
          <warName>SpringMVCHibernateManyToManyCRUDEx
          <failOnMissingWebXml>>false</failOnMissingWe
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
  <finalName>SpringMVCHibernateManyToManyCRUDExample</finalNa
</build>
</project>

```

Step 4. Prepare Model classes

```

package com.websystique.springmvc.model;

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 {

    @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
@ManyToOne(fetch = FetchType.LAZY)
@JoinTable(name = "APP_USER_USER_PROFILE",
    joinColumns = { @JoinColumn(name = "USER_ID") },
    inverseJoinColumns = { @JoinColumn(name = "USER_PROFI
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;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", ssoId=" + ssoId + ", password="
            + ", firstName=" + firstName + ", lastName=" + lastName
            + ", email=" + email + "]\n";
    }
}

```

Look at how userProfile property is annotated with **ManyToMany**.

`@ManyToMany` indicates that there is a **Many-to-Many relationship** between User and UserProfile. A User can have several profiles [USER, ADMIN, DBA] while a profile can belong to several users. `@JoinTable` indicates that there is a link table which joins two tables using foreign key constraints to their primary keys. This annotation is mainly used on the owning side of the relationship. `joinColumns` refers to the column name of owning side(ID of USER), and `inverseJoinColumns` refers to the column of inverse side of relationship(ID of USER_PROFILE). Primary key of this joined table is combination of USER_ID & USER_PROFILE_ID.

Lazy Loading:

Pay special attention to `fetch = FetchType.LAZY`. Here we are informing hibernate to lazy load the userProfile collection. It's also the default behavior. With this setup, a query to load the collection will be fired only when it is first accessed. It's a good way to avoid fetching all connected object graph which is an expensive operation. When you are in transaction/active session, and will try to access collection, hibernate will fire separate select to fetch them.

But if you are outside active session (session closed/no transaction :you are in JSP e.g.), and tried to access the collection, you will meet your nemesis :

org.hibernate.LazyInitializationException – could not initialize proxy – no Session. To avoid it, you need to initialize the collection on demand by calling `Hibernate.initialize(user.getUserProfiles());` within an active session [you know the DAO method you were in, before coming all the way to view, you may call this initialize in that method.]

Also note that we are not using any cascade. It is because a userprofile is not dependent of user, and can live independently.

One important remark : In case of *Many* association, always override hashCode and equals method which are looked by hibernate when holding entities into collections.

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

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 {

    @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 + "]";
}
}

```

Since we are showing unidirectional relationship(User to UserProfile), no need to refer User in UserProfile.

```

package com.websystique.springmvc.model;

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

    String userProfileType;

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

    public String getUserProfileType(){
        return userProfileType;
    }
}

```

Step 5. Create DAO layer

```

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 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.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.  
(())).getActualTypeArguments()[1];  
    }  
  
    @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 delete(T entity) {  
        getSession().delete(entity);  
    }  
  
    protected Criteria createEntityCriteria(){  
        return getSession().createCriteria(persistentClass);  
    }  
  
}
```

```
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.springframework.stereotype.Repository;

import com.websystique.springmvc.model.User;

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

    public User findById(int id) {
        User user = getByKey(id);
        if(user!=null){
            Hibernate.initialize(user.getUserProfiles());
        }
        return user;
    }

    public User findBySSO(String sso) {
        System.out.println("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("ssoId"));
        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
        /*
        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 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, UserProfile> {

    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 6. Create Service layer

```
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 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.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();
    }
}
```

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

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

    @Autowired
    private UserDao dao;

    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) {
    dao.save(user);
}

/*
 * Since the method is running with Transaction, No need to call
 * Just fetch the entity from db and update it with proper values.
 * 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());
        entity.setPassword(user.getPassword());
        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() ==
}
}

```

Step 7. Create Hibernate Configuration

```

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.EnableTransactionManagement

@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 LocalSessionFactoryBean();
    sessionFactory.setDataSource(dataSource());
    sessionFactory.setPackagesToScan(new String[] { "com.websys" });
    sessionFactory.setHibernateProperties(hibernateProperties());
    return sessionFactory;
}

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

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

@Bean
@Autowired
public HibernateTransactionManager transactionManager(SessionFactory sessionFactory) {
    HibernateTransactionManager txManager = new HibernateTransactionManager(sessionFactory);
    txManager.setSessionFactory(sessionFactory);
    return txManager;
}
}

```

Above Hibernate configuration uses below mentioned

`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 8. Create Controller

```

package com.websystique.springmvc.controller;

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

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
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;

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

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

    /**
     * This method will provide the medium to add a new user.
     */
    @RequestMapping(value = { "/newuser" }, method = RequestMethod.POST)
    public String newUser(ModelMap model) {
        User user = new User();
        model.addAttribute("user", user);
        model.addAttribute("edit", false);
        return "registration";
    }

    /**
     * This method will be called on form submission, handling POST
     * 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
         * and applying it on field [sso] of Model class [User].
         *
         * Below mentioned piece of code [if block] is to demonstrate
         * validation
         * framework as well while still using internationalized messages
         */
    }
}

```

```

        if(!userService.isUserSSOUnique(user.getId(), user.getSsoId())) {
            FieldError ssoError =new FieldError("user","ssoId",message);
            String[] {user.getSsoId()}, Locale.getDefault()););
            result.addError(ssoError);
            return "registration";
        }

        userService.saveUser(user);

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

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

    /**
     * This method will be called on form submission, handling POST
     * updating user in database. It also validates the user input
     */
    @RequestMapping(value = { "/edit-user-{ssoId}" }, method = RequestMethod.POST)
    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
        if(!userService.isUserSSOUnique(user.getId(), user.getSsoId())) {
            FieldError ssoError =new FieldError("user","ssoId",message);
            String[] {user.getSsoId()}, Locale.getDefault()););
            result.addError(ssoError);
            return "registration";
        }*/

        userService.updateUser(user);

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

    /**
     * This method will delete an user by it's SSOID value.
     */
    @RequestMapping(value = { "/delete-user-{ssoId}" }, method = RequestMethod.GET)
    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();
    }

```

```
}

```

Messages are defined in below mentioned `messages.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

```

Step 9. Create Converter

This is the heart of this post. It will take care of mapping the individual userProfile id's to actual UserProfile Entities in

database.

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

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
 */
@Component
public class RoleToUserProfileConverter implements Converter<Object> {

    @Autowired
    UserProfileService userProfileService;

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

```

Step 10. Create Spring Configuration

```

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.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
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 InternalResourceViewResolver();
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        registry.viewResolver(viewResolver);
    }

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

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

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

    /**Optional. It's only required when handling '.' in @PathVariable
     * It's a known bug in Spring [https://jira.spring.io/browse/SPR-11344]
     */
}

```

```

    * This is a workaround for this issue.
    */
    @Override
    public void configurePathMatch(PathMatchConfigurer matcher) {
        matcher.setUseRegisteredSuffixPatternMatch(true);
    }
}

```

First interesting thing is registration converter we created in previous step with Spring configuration using **addFormatters**. Next is the method **configurePathMatch** which provides a workaround (although other workaround exists) for a known bug in spring, which is still found in Spring 4.1.7.RELEASE.

Above Converter setup in **XML configuration** will be:

```

<mvc:annotation-driven conversion-service="conversionService"/>
<bean id="conversionService" class="org.springframework.format.support.FormattingConversionService"
    <property name="converters">
        <list>
            <bean id="roleToUserProfile" class="com.websystique.springmvc.converter.RoleToUserProfileConverter"/>
        </list>
    </property>
</bean>

```

Add initializer class:

```

package com.websystique.springmvc.configuration;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

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

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

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

```

Step 11: Add Views/JSP's

Note that we are using Bootstrap for styling in JSP.

userslist.jsp

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

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
    <title>Users List</title>
    <link href="<c:url value='/static/css/bootstrap.css' />" rel="stylesheet" />
    <link href="<c:url value='/static/css/app.css' />" rel="stylesheet" />
</head>

<body>
    <div class="generic-container">
        <div class="panel panel-default">
            <!-- Default panel contents -->
            <div class="panel-heading"><span class="lead">List of Users</span></div>
            <table class="table table-hover">
                <thead>
                    <tr>
                        <th>Firstname</th>
                        <th>Lastname</th>
                        <th>Email</th>
                        <th>SSO ID</th>
                        <th width="100"></th>
                        <th width="100"></th>
                    </tr>
                </thead>
                <tbody>
                    <c:forEach items="${users}" var="user">
                        <tr>
                            <td>${user.firstName}</td>
                            <td>${user.lastName}</td>
                            <td>${user.email}</td>
                            <td>${user.ssoId}</td>
                            <td><a href="<c:url value='/edit-user-${user.ssoId}' />">
                                custom-width">edit</a></td>
                            <td><a href="<c:url value='/delete-user-${user.ssoId}' />">
                                custom-width">delete</a></td>
                        </tr>
                    </c:forEach>
                </tbody>
            </table>
        </div>
        <div class="well">
            <a href="<c:url value='/newuser' />">Add New User</a>
        </div>
    </div>
</body>
</html>
```

registration.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>
<%@ 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-8859-1">
<title>User Registration Form</title>
<link href="<c:url value='/static/css/bootstrap.css' />" rel="stylesheet">
<link href="<c:url value='/static/css/app.css' />" rel="stylesheet">
</head>

<body>

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

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

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

  <div class="row">
    <div class="form-group col-md-12">
      <label class="col-md-3 control-label" for="ssoId">Social Security ID</label>
      <div class="col-md-7">
        <c:choose>
          <c:when test="${edit}">
            <form:input type="text" path="ssoId" id="ssoId" class="input-sm" disabled="true"/>
          </c:when>
          <c:otherwise>
            <form:input type="text" path="ssoId" id="ssoId" class="input-sm" />
          </c:otherwise>
        </c:choose>
        <div class="has-error">
          <form:errors path="ssoId" class="help-inline">
        </div>
      </div>
    </div>
  </div>

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

  <div class="row">

```



```

<div class="form-group col-md-12">
  <label class="col-md-3 control-label" 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-label" for="userProf
    <div class="col-md-7">
      <form:select path="userProfiles" items="${roles
itemLabel="type" class="form-control input-sm" />
      <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="
href="<c:url value='/list' />">Cancel</a>
      </c:when>
      <c:otherwise>
        <input type="submit" value="Register" class
href="<c:url value='/list' />">Cancel</a>
      </c:otherwise>
    </c:choose>
  </div>
</div>
</form:form>
</div>
</body>
</html>

```

registrationsuccess.jsp

```

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

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO
  <title>Registration Confirmation Page</title>
  <link href="<c:url value='/static/css/bootstrap.css' />" rel="s
  <link href="<c:url value='/static/css/app.css' />" rel="stylesh
</head>
<body>
<div class="generic-container">
  <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>
```

And a tiny custom stylesheet file:

app.css

```
body, #mainWrapper {
    height: 100%;
    background-color:rgb(245, 245, 245);
}

body, .form-control{
    font-size:12px!important;
}

.floatRight{
    float:right;
    margin-right: 18px;
}

.has-error{
    color:red;
}

.generic-container {
    position:fixed;
    width:80%;
    margin-left: 20px;
    margin-top: 20px;
    margin-bottom: 20px;
    padding: 20px;
    background-color: #EAE7E7;
    border: 1px solid #ddd;
    border-radius: 4px;
    box-shadow: 0 0 30px black;
}

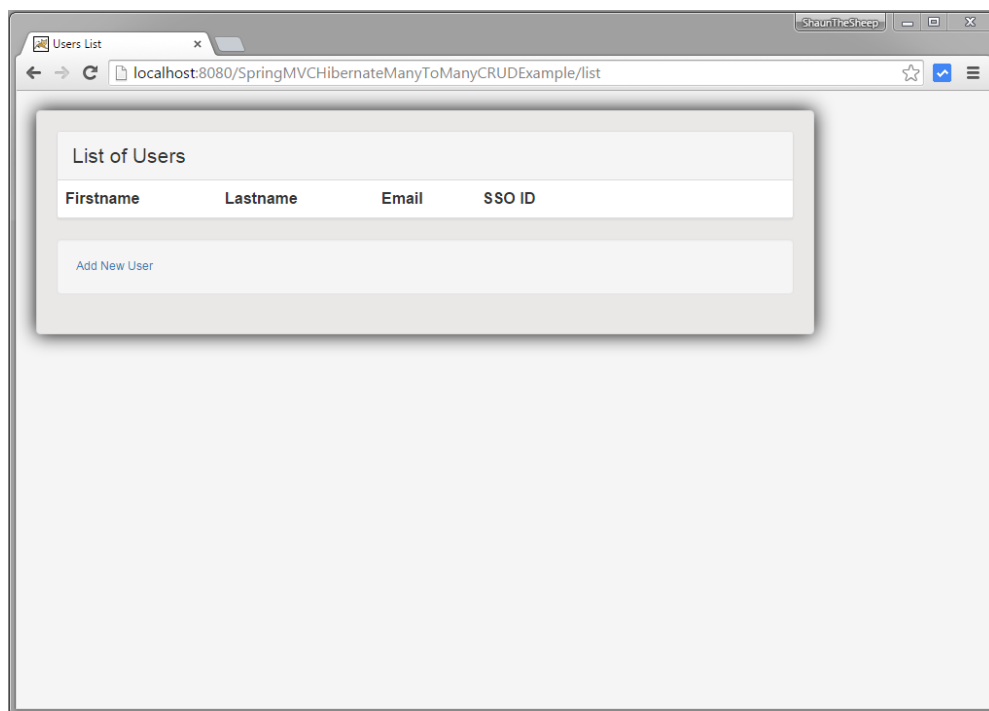
.custom-width {
    width: 80px !important;
}
```

Step 12: 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.

Open browser and browse at

<http://localhost:8080/SpringMVCHibernateManyToManyCRUDEExample/>



Click on 'Add New User'

User Registration Form

localhost:8080/SpringMVCHibernateManyToManyCRUDExample/newuser

First Name

Last Name

SSO ID

Password

Email

Roles

ADMIN
DBA
USER

[Register](#) or [Cancel](#)

Submit without filling anything.

User Registration Form

First Name

First name can not be blank.

Last Name

Last name can not be blank.

SSO ID

SSO ID can not be blank.

Password

Password can not be blank.

Email

Email can not be blank.

Roles
DBA
USER

At least one profile must be selected.

or

Fill in details

User Registration Form

First Name

First name can not be blank.

Last Name

Last name can not be blank.

SSO ID

SSO ID can not be blank.

Password

Password can not be blank.

Email

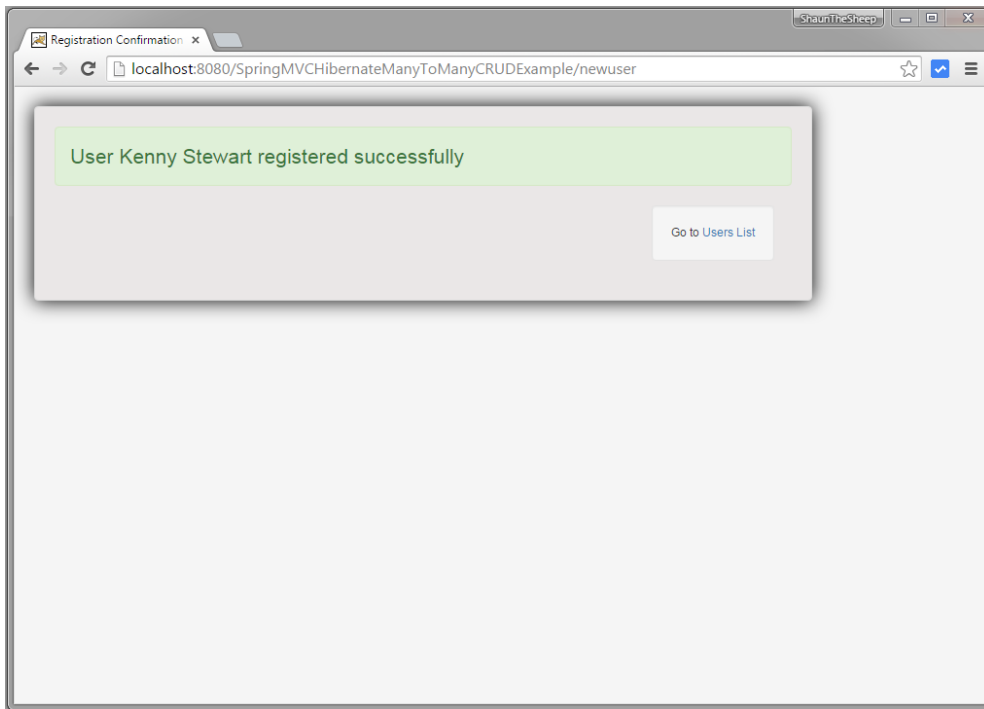
Email can not be blank.

Roles
DBA
USER

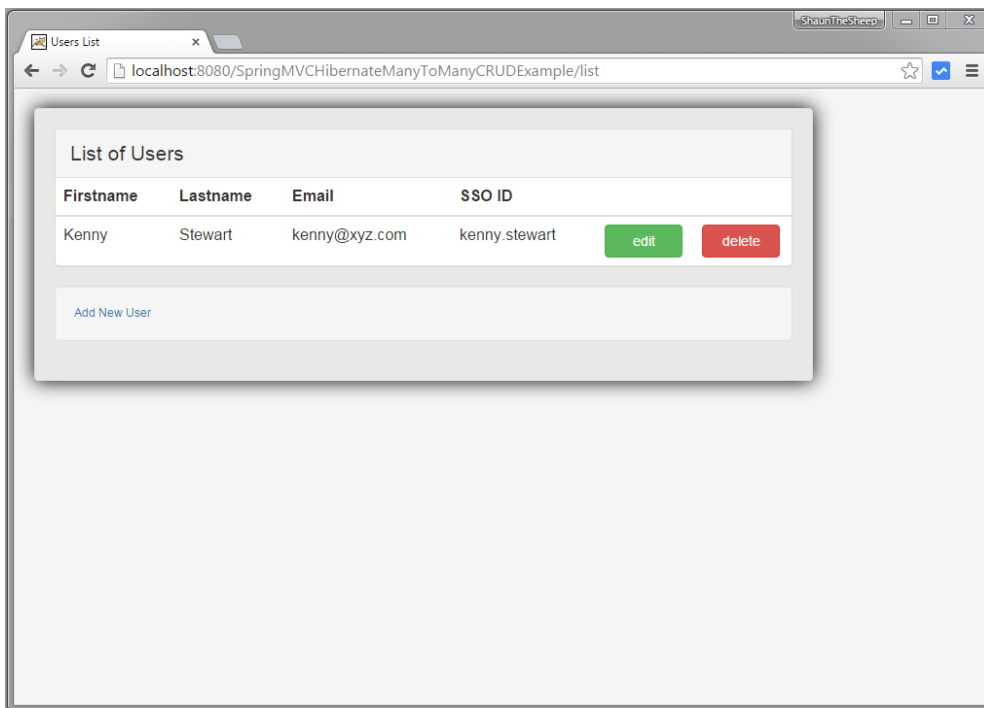
At least one profile must be selected.

or

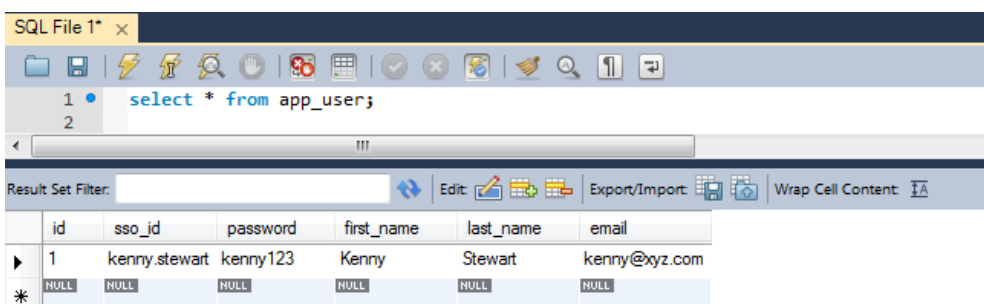
Submit.

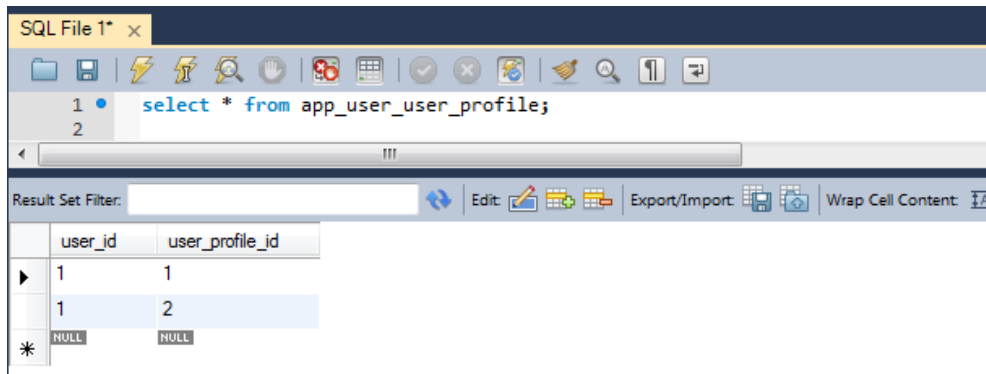


Click on 'Users List' link.

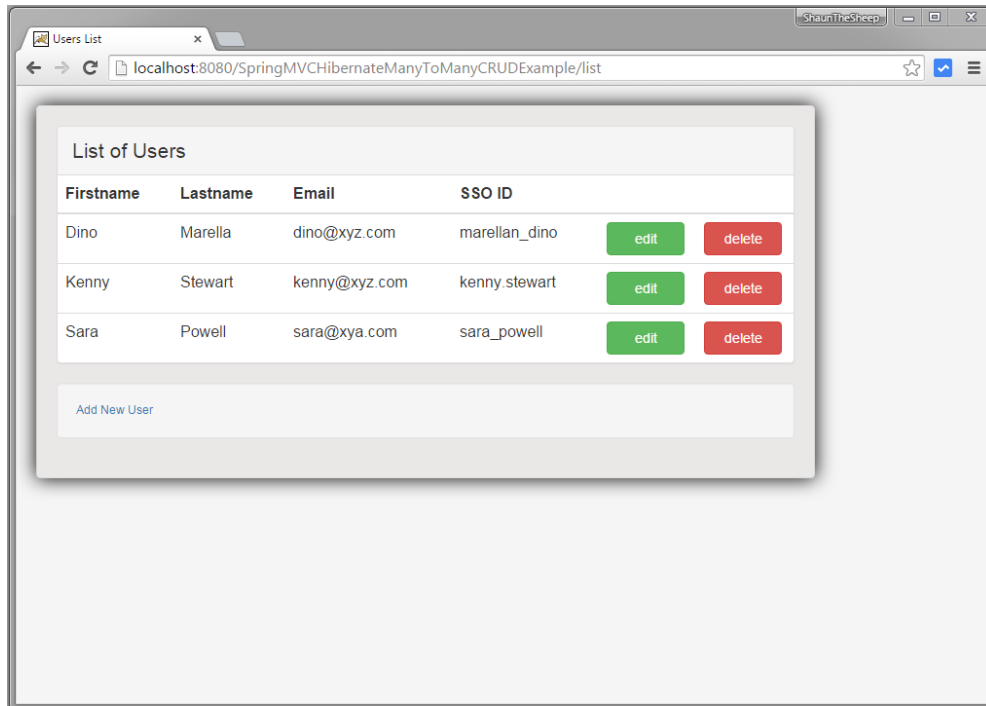


Check the database at this moment.

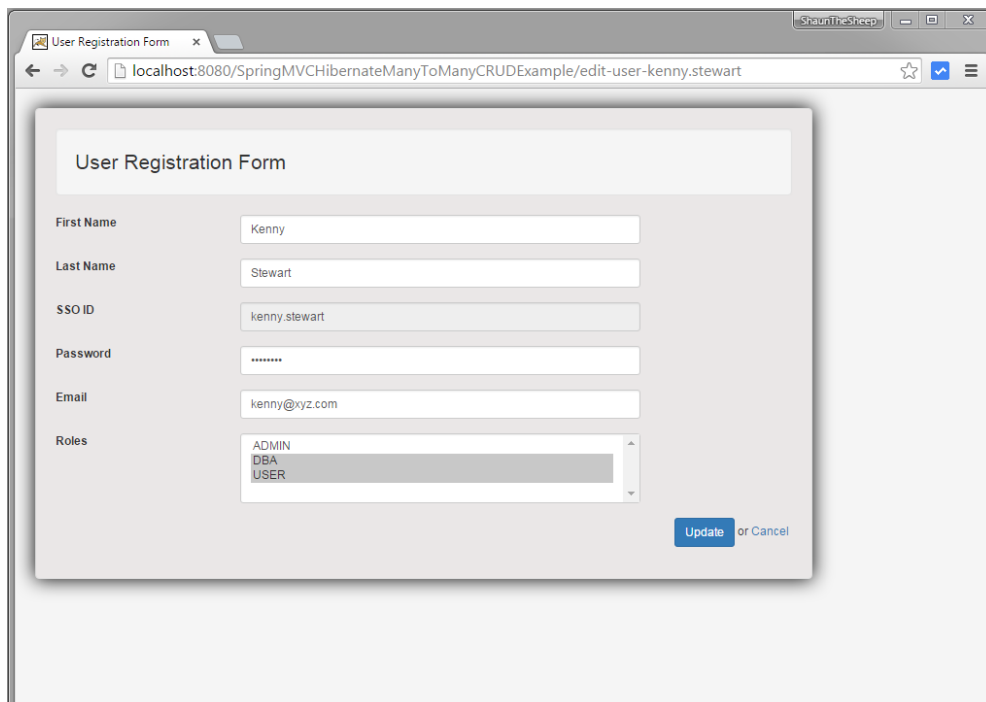




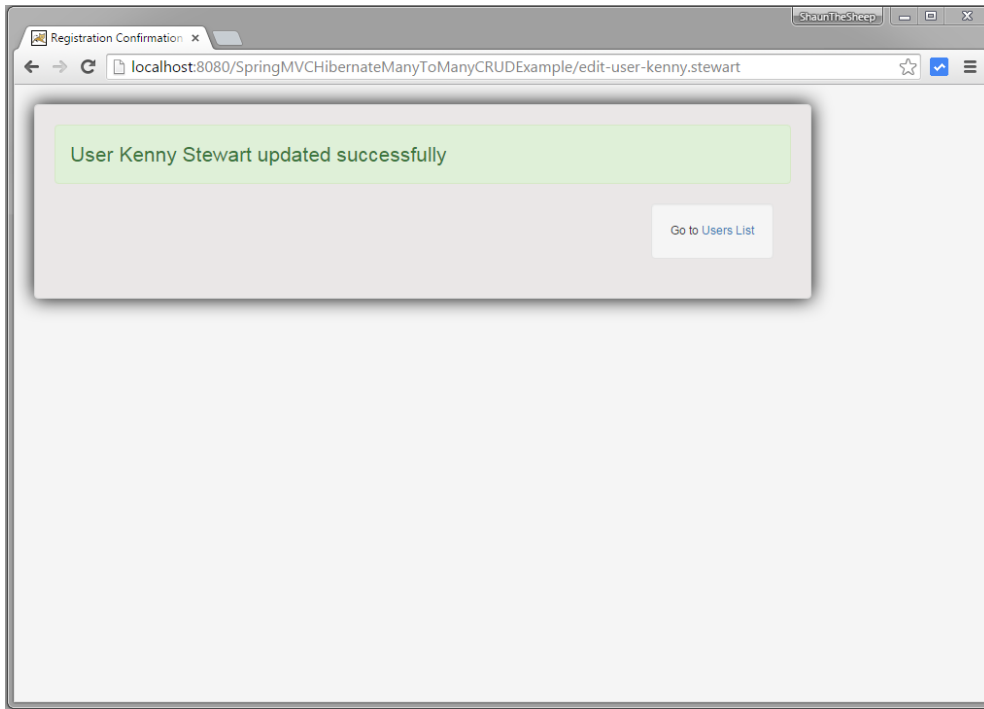
Add more users.



Click Edit button for User Kenny. Change Roles.



Submit.



Verify the database.

SQL File 1* x

```
1 • select * from app_user;  
2
```

Result Set Filter: [] Edit: [] Export/Import: [] Wrap Cell Content: []

	id	sso_id	password	first_name	last_name	email
▶	1	kenny.stewart	kenny123	Kenny	Stewart	kenny@xyz.com
	2	sara_powell	sara123	Sara	Powell	sara@xya.com
	3	marellan_dino	dino123	Dino	Marella	dino@xyz.com
*	NULL	NULL	NULL	NULL	NULL	NULL

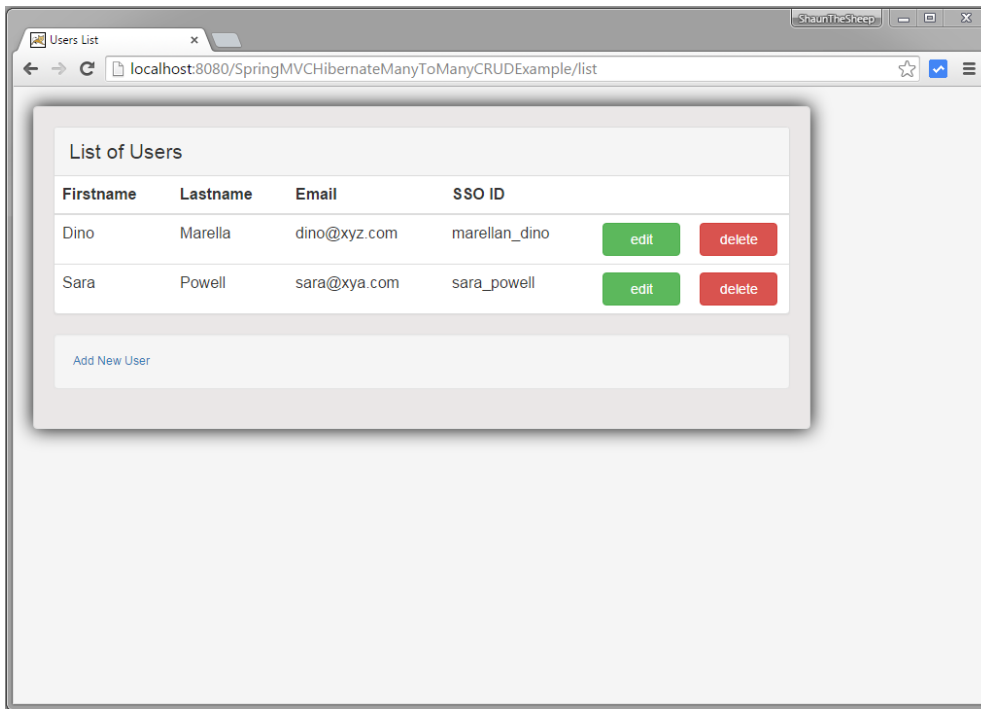
SQL File 1* x

```
1 • select * from app_user_user_profile order by user_id;  
2
```

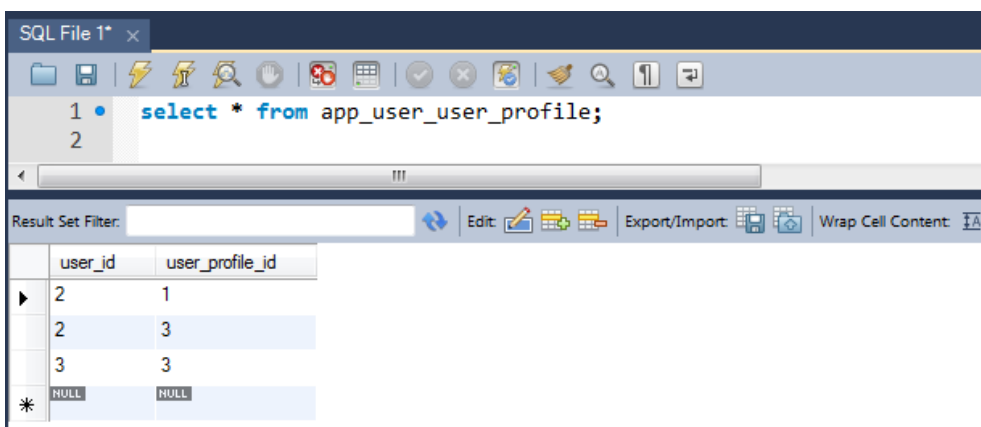
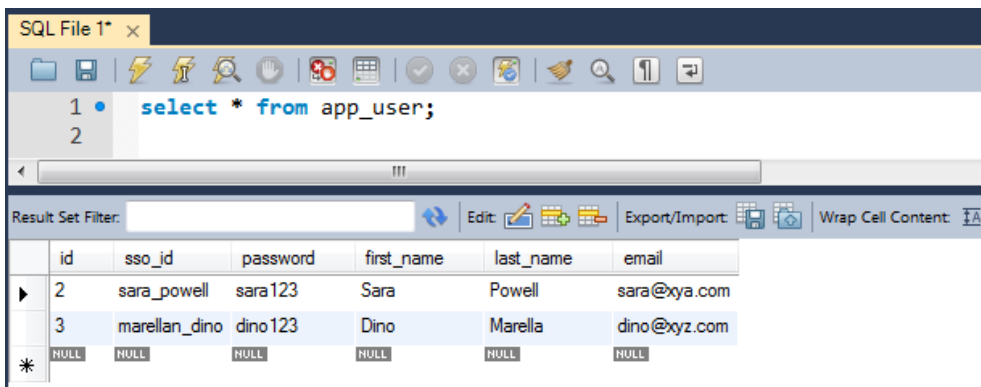
Result Set Filter: [] Edit: [] Export/Import: [] Wrap Cell Content: []

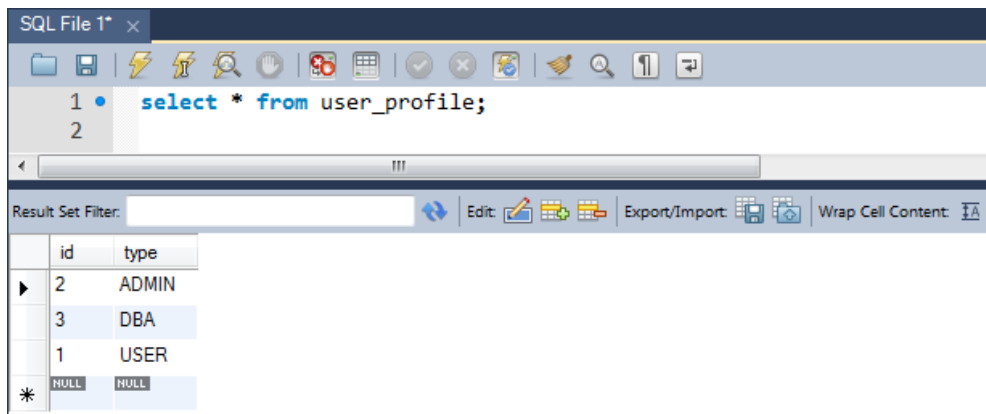
	user_id	user_profile_id
▶	1	1
	1	3
	2	1
	2	3
	3	3
*	NULL	NULL

Now go back to list and click on DELETE for user kenny. It should be history.



Finally check the database at this moment :





The screenshot shows a SQL IDE window titled 'SQL File 1*'. The query editor contains the following SQL statement:

```
1 • select * from user_profile;  
2
```

Below the query editor, the 'Result Set Filter' is empty. The results are displayed in a table with columns 'id' and 'type'.

id	type
2	ADMIN
3	DBA
1	USER
*	HULL

That's it.

[Download Source Code](#)

[Download Now!](#)

References

- [Spring framework](#)
- [Hibernate Validator](#)



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 Hibernate Integration Annotation+XML Example](#)

2. [Spring MVC 4 FileUpload-Download Hibernate+MySQL Example](#)
3. [Spring Security 4 Hibernate Password Encoder Bcrypt Example](#)
4. [Spring Security 4 Remember Me Example with Hibernate](#)

 [springmvc.](#)  [permalink.](#)

[← Spring Security 4 Role Based Login Example](#)

[Spring MVC @RequestBody @ResponseBody Example →](#)

43 Comments

[websystique](#)

 [Login](#) ▾

 [Recommend](#)

 [Share](#)

[Sort by Best](#) ▾

Join the discussion...



Devarshi Shah • 5 days ago

can i see ur web.xml file?

^ | ▾ • [Reply](#) • [Share](#) ›



Devarshi Shah → [Devarshi Shah](#) • 5 days ago

the code you have posted in not running.

^ | ▾ • [Reply](#) • [Share](#) ›



websystique Mod → [Devarshi Shah](#) • 5 days ago

Hi Devarshi,

It's Servlet 3.x app, so you don't need web.xml.
Now, about the code, it is not running in your system probably because your eclipse+tomcat setup is not configured properly. Have a look at [Setup Tomcat With Eclipse](#) to configure it properly. Let me know if you still face any issue.

^ | ▾ • [Reply](#) • [Share](#) ›



Devarshi Shah → [websystique](#) • 3 days ago

hey my tomcat is working properly. there is some issue with bean in appconfig. if you dont mind can i send you my entire project?

^ | ▾ • [Reply](#) • [Share](#) ›



Greg H • 9 days ago

Thanks for the tutorial!

^ | ▾ • [Reply](#) • [Share](#) ›



websystique Mod → Greg H • 9 days ago

Hi Greg,

Probably the reason is that your Jetty plugin is not configured properly.

Here is the [Official Plugin Doc](#). Plugin itself can be found [here](#). Note that Jetty 9.3.X requires Java 8 to work, due to [this bug](#).

Add following to your pom.xml, plugins section (assuming you use java 1.7):

```
<plugin>
<groupId>org.eclipse.jetty</groupId>
<artifactId>jetty-maven-plugin</artifactId>
<version>9.2.14.v20151106</version>
<configuration>
<scanIntervalSeconds>10</scanIntervalSeconds>
<webapp>
<contextPath>/SpringMVCHibernateManyToManyCRUDExample</contextPath>
</webapp>
</configuration>
</plugin>
```

This should unblock you. Let me know if you still face any issue.

1 ^ | v • Reply • Share ›



Greg H → websystique • 8 days ago

I actually just saw that bug and made the fix right before you commented. Thanks for the help! I'm trying to build upon your example now. When I add a new object trying to make a new many-to-many relationship, I receive the following error:

```
org.hibernate.QueryException: could not resolve
property: type of:
com.websystique.springmvc.model.Institution
at
org.hibernate.persister.entity.AbstractPropertyMapping
at
org.hibernate.persister.entity.AbstractPropertyMapping
at
org.hibernate.persister.entity.AbstractEntityPersister
at
org.hibernate.persister.entity.BasicEntityPropertyMapping
at
```

[see more](#)

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**websystique** Mod [→](#) Greg H • 8 days ago

Hi Greg,

Do you have a type property defined in your Institution model class, something like shown below?

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

If not, could you please show your com.websystique.springmvc.model.Institution class & InstitutionDaoImpl.findAll() method?

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Greg H** [→](#) websystique • 8 days ago

I don't have a type property defined. Here is the institution class:

```
@Entity
@Table(name="INSTITUTION")
public class Institution {
@Id
@GeneratedValue(strategy=GenerationType
private Integer id;

@Column(name = "NAME")
private String name;

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

public String getName() {
return name;
}
```

[see more](#)[^](#) | [v](#) • [Reply](#) • [Share](#) ›**websystique** Mod [→](#) Greg H • 8 days ago

If you only have id property available, then in Dao findAll method, you should be using 'id' instead of 'type'.

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Greg H** [→](#) websystique • 7 days ago

I apologize, I pasted the wrong bit of code. My edit shows the correct class.

[^](#) | [v](#) • [Reply](#) • [Share](#) ›

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Greg H** [→](#) Greg H • 7 days ago

Ok, I see what you meant by "type" there in the sort order criteria of the DaoImpl. I changed that to name and things seem to be working now. Thanks again for the help and keep up with the amazin tutorials!

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Jithin Mathew** • 12 days ago

First of all, this is really a great tutorial for beginners like me. Really loved it and build just like what you taught here.

Just to know, if you have any JUnit testing code for this same project listed above! It would be really great if you could share the same if you have one, so that it will help a lot with beginners like me!

Thanks in advance!

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**websystique** Mod [→](#) Jithin Mathew • 11 days ago

Hi Jithin, Take a look at [this post](#). These two posts are almost same, so you will get idea about how to write tests for them. If you get stuck somewhere, don't hesitate to shout.

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Ahmed Deen** • 23 days ago

Role's values are not passing to controller, can u pls help me

User Registration Form

First Name	Deen
Last Name	Khan
SSN ID	123
Password	***
Email	deenkhan@gmail.com
Roles	ADMIN DBA USER

Invalid format

[Register](#) or [Cancel](#)

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**websystique** Mod [→](#) Ahmed Deen • 23 days ago

Hi Ahmed, Did you configure the converter [RoleToUserProfileConverter] as shown in this post? BTW, did you make any changes in project after download, because i did not see this error before?

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Ahmed Deen** [→](#) websystique • 23 days ago



Thanks Websystique, Its now working fine,

Actually i configured converter
[RoleToUserProfileConverter] package name
wrongly.

Very very thanks for your quick response!

^ | v • Reply • Share ›



Mario Cuollo Conforti • a month ago

Hello websystique, your tutorials are great! I'm learning a lot from them. I have a question for you. If I want to list all the users of a certain type (i would like a "USER" list, an "ADMIN" list and so on. I'm trying to modify the userService.findAllUsers(); to pass to it a "type" parameter to use a filter for the query, but with no success. I will pass this "type" via GET and taking it via @RequestParam in the controller.

But, i don't know how to use your infrastructure to achieve it.

I was trying in UserDaoImpl something like this, but with no effect because the table name is not a User parameter...

```
public List<user> findAllUsers(int typeId) {

Criteria criteria = createEntityCriteria();
criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);

@SuppressWarnings("unchecked")
List<user> users = (List<user>)
criteria.createAlias("users_user_profile",
"uup").createAlias("user_profile", "up").add(Restrictions.eq("up.id",
typeId)).list();

return users;

}
```

^ | v • Reply • Share ›



websystique Mod ➔ Mario Cuollo Conforti • a month ago

Hi Mario,

This should unblock you:

```
public List<user> findAllUsersOfSpecificType(String type) {

Criteria criteria =
createEntityCriteria().addOrder(Order.asc("firstName"));
if(StringUtils.isNotBlank(type)){
criteria.createAlias("userProfiles", "profiles");//inner join
criteria.add(Restrictions.eq("profiles.name",
type.toUpperCase()));
}
```

```
criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);
// avoid duplicates.
List<user> users = (List<user>) criteria.list();
return users;
}
```

Let me know if it does not help you.

1 ^ | v • Reply • Share ›



Mario Cuollo Conforti → websystique
• a month ago

Thanks, it's perfect!!! The only thing, if you want to publish it, is that

"criteria.add(Restrictions.eq("profiles.name", type.toUpperCase()));" should be

"criteria.add(Restrictions.eq("profiles.type", type.toUpperCase()));"

I can reach the list in this way: /users?type=admin or /users?type=user

Then the controller is

```
@RequestMapping(value = "/users",
method=RequestMethod.GET)
public String listUserByType(ModelMap model,
@RequestParam String type){
List<user> users =
userService.findAllUsersOfSpecificType(type);
model.addAttribute("users", users);
return path+"/users";
}
```

1 ^ | v • Reply • Share ›



websystique Mod → Mario Cuollo Conforti
• a month ago

Glad it was helpful.

^ | v • Reply • Share ›



Stepan Cleverjava • 2 months ago

The more I read the more I understand how good your posts are. Thanks for them. I would like to ask you about this post.

As far as I understood, in the following chain: public String newUser(...) --> registration.jsp --> public String saveUser(...) you bind twice: 1) An empty user created in the newUser(...) method with registration.jsp (that is why the fields are empty) 2) Nonempty object created in registration.jsp (the fields are mandatory) with the one passing to saveUser(...) method. Correct me if I'm wrong.

My question is why do you need double binding? Instead you could

only exploit binding registration.jsp --> public String
saveUser(@ModelAttribute...). In that case instead of e.g.
<form:input...> you could use plain <input...>

Thanks in advance for you response!

^ | v • Reply • Share ›



websystique Mod → Stepan Cleverjava • 2 months ago

Hi Stepan,

Sorry for late reply. I kept them separate to cleanly apply
validation. You can of course handle it as you mentioned but
then you may need to put some conditional check to handle
validations. Hope it helps.

^ | v • Reply • Share ›



Stepan Cleverjava → websystique • 2 months ago

Thanks a lot!

While waiting for your response I've got another
question. It looks like that @ModelAttribute and
@SessionAttributes play the roles of old request and
session attributes. Then a natural question arises –
is there some new analogue of context attribute?
Again thank you for your current and further replies!

^ | v • Reply • Share ›



websystique Mod → Stepan Cleverjava
• 2 months ago

Hi Stepan,

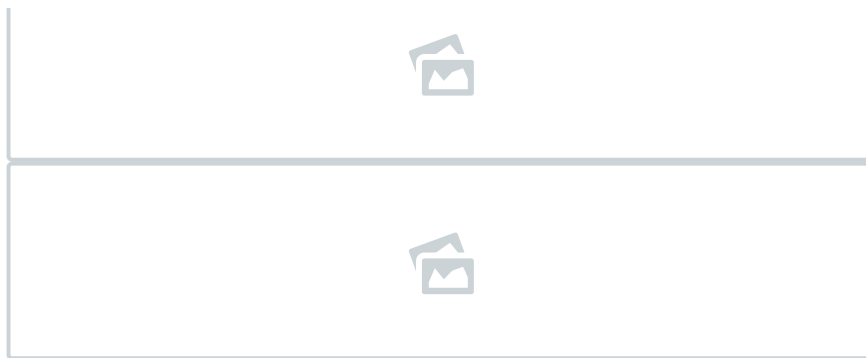
Not that i am aware about. But you can get
the behavior you want by implementing
ServletContextAware in your bean class and
then eventually using setAttribute and
getAttribute on ServletContext. An example
can be found [here](#).

^ | v • Reply • Share ›



Aravind Karthick • 3 months ago

I am getting the following error. Kindly help.



^ | v • Reply • Share ›



vamshi ➔ Aravind Karthick • 2 months ago

Hi,

Try this (from Stackoverflow)

Another way is to edit the project facet configuration file itself: `org.eclipse.wst.common.project.facet.core.xml`

Change the dynamic web module version in this line to 3.1 -
`<installed facet="jst.web" version="2.5"/>`

And then:

Right-click on the project (in the Project Explorer panel).

Select Maven » Update Project (or press Alt+F5)

You'll find this file in the `.settings` directory within the Eclipse project.

^ | v • Reply • Share ›

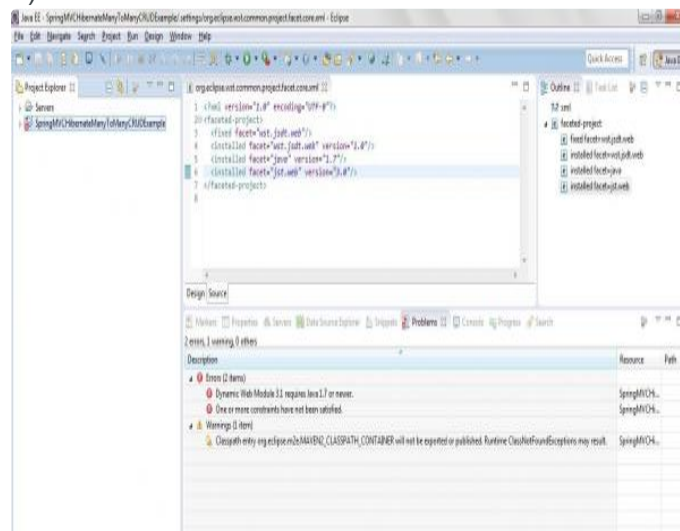


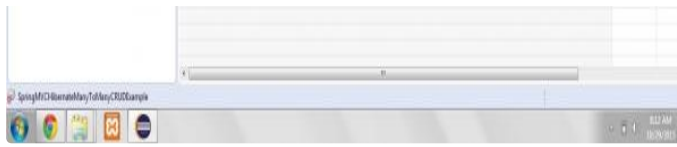
Aravind Karthick ➔ vamshi • 2 months ago

I tried this one many times (Attached the screen shot already).. But still i'm facing the issue.

I am using the following tools

- 1) Eclipse Mars
- 2) JDK 1.7
- 3) Tomcat 7.X





4) Maven 3.3

5) My Sql - Xampp

I find no means to identify the root cause of this issue.

^ | v • Reply • Share ›



websystique Mod → Aravind Karthick
• 2 months ago

Hi Aravind, why don't you simply remove the error (select the top level node named "errors" in Problems window, delete it). It's Eclipse idiosyncrasy, project itself should work fine. Additionally, you may want to look at [Setting tomcat with eclipse](#) in case you are facing any issue with Eclipse setup. Let me know if you have any other issues.

^ | v • Reply • Share ›



guojunjun • 3 months ago

Hi websystique,

how do to make it bidirectional ?

User knows UserProfile by:

```
```private Set<userprofile> userProfiles = new
HashSet<userprofile>();```
```

if I add this to UserProfile side it would bidirectional:

```
```private Set<user> users = new HashSet<user>();```
```

I've tried ```fetch = FetchType.LAZY, FetchType.EAGER, cascade = {CascadeType.ALL} ...```

but cannot make it work, any suggestion for this?

Best greetings

^ | v • Reply • Share ›



websystique Mod → guojunjun • 3 months ago

Hi Guojunjun,

I would suggest you to first go through [Hibernate Many-To-Many-Bidirectional Example](#) post. It should clear your doubts. Do let me know if you still face other issues.

^ | v • Reply • Share ›



guojunjun → websystique • 3 months ago

Thank you very much for your reply

@websystique, I do checked the example you posted, the database works as it should, but when it comes to request (with servlet, spring implemented) the No Session problem comes again if I use 'fetch = FetchType.LAZY', (and I also tried 'fetch = FetchType.EAGER' then I have 'com.fasterxml.jackson.databind.JsonMappingException: Premature EOF (through reference chain' problem, but I do want to use FetchType.LAZY, this is the one could handle more request, with my understanding)

How do I fix this?

Best greetings

^ | v • Reply • Share ›



websystique Mod → guojunjun
• 3 months ago

Hi Guojunjun,

Sorry for late reply. What about creating a separate method in UserProfileDaoImpl where you can initialize the users collection, which you can then use (via service) in ApplicationController, something like

```
//UserProfileDaoImpl
public UserProfile
findByTypeInitialized(String type) {
    Criteria crit = createEntityCriteria();
    crit.add(Restrictions.eq("type", type));
    UserProfile userProfile = (UserProfile)
    crit.uniqueResult();
    if(userProfile!=null){
        Hibernate.initialize(userProfile.getUsers());
    }
}
```

[see more](#)

^ | v • Reply • Share ›



guojunjun → websystique • 3 months ago

Hi websystique, thank you very much for your solution, I thought something similar, but of course not as good as yours:)

I got this error with the solution you suggested:

`object references an unsaved transient instance - save the transient instance before

flushing`

So I added 'cascade = {CascadeType.ALL}' to first make it bidirectional, then use '@JsonIgnore' at child side, to stop the infinite loop problem. After this I added your solution to get it bidirectional again :)

Thank you very much again for all your helps :)

Best greetings

Junjun Guo

^ | v • Reply • Share ›



Marcos Lozina • 3 months ago

Hello again, great tutorial, I did not find anything like it in the entire web. You are java certified? one question: where you define "welcome-file" originally defined in the web.xml ??

^ | v • Reply • Share ›



websystique Mod ➔ Marcos Lozina • 3 months ago

Hi Marcos,

Glad you liked it. For the welcome page, I have defined it in default controller method('/'). Look at listUser method in ApplicationController class.

Copyright © 2014-2015 WebSystique.com. All rights reserved.