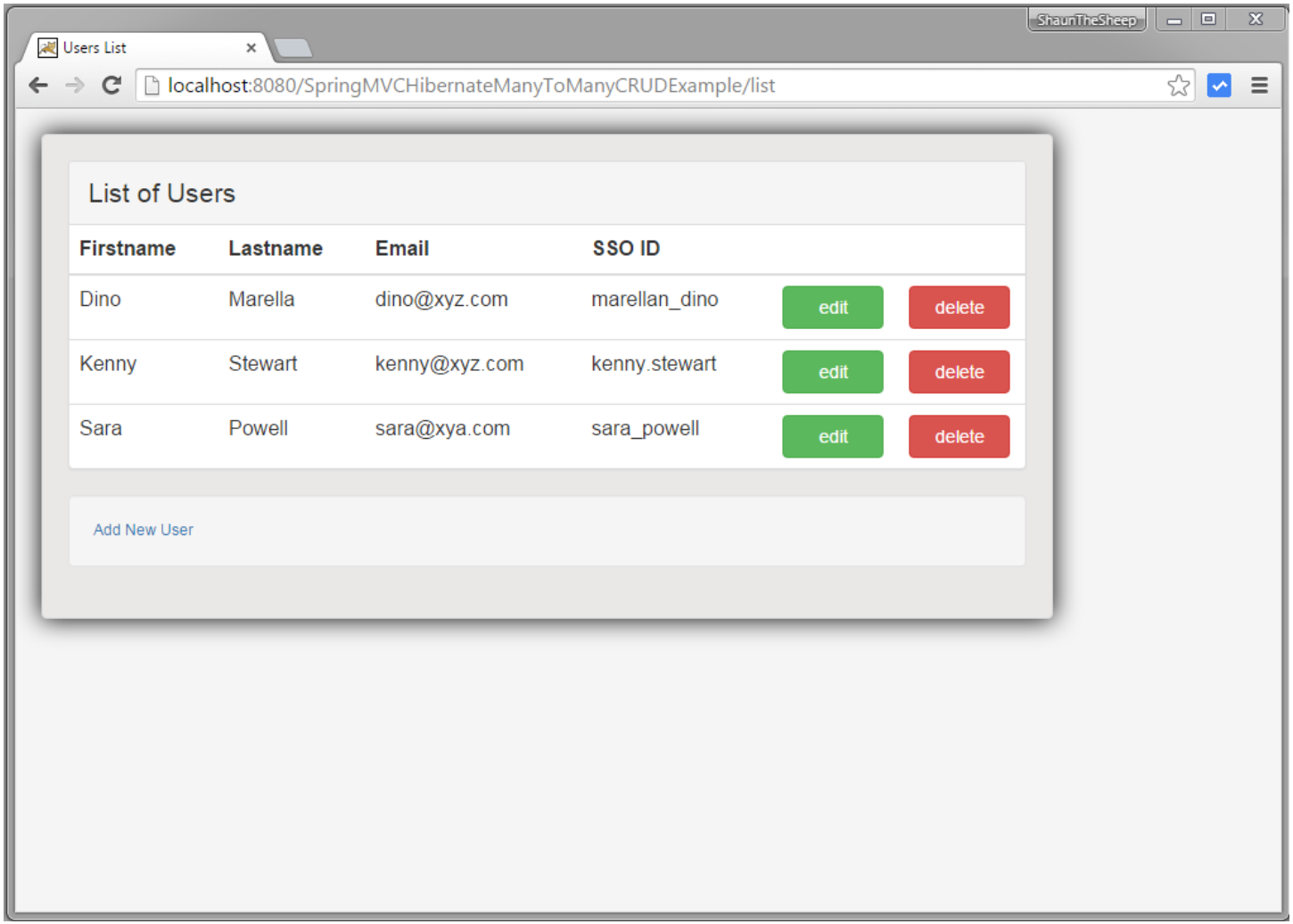
WebSystique
learn together

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

Created on: August 1, 2015 | **Last updated on:** September 30, 2017 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. Let's get going.

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.



Recent Posts

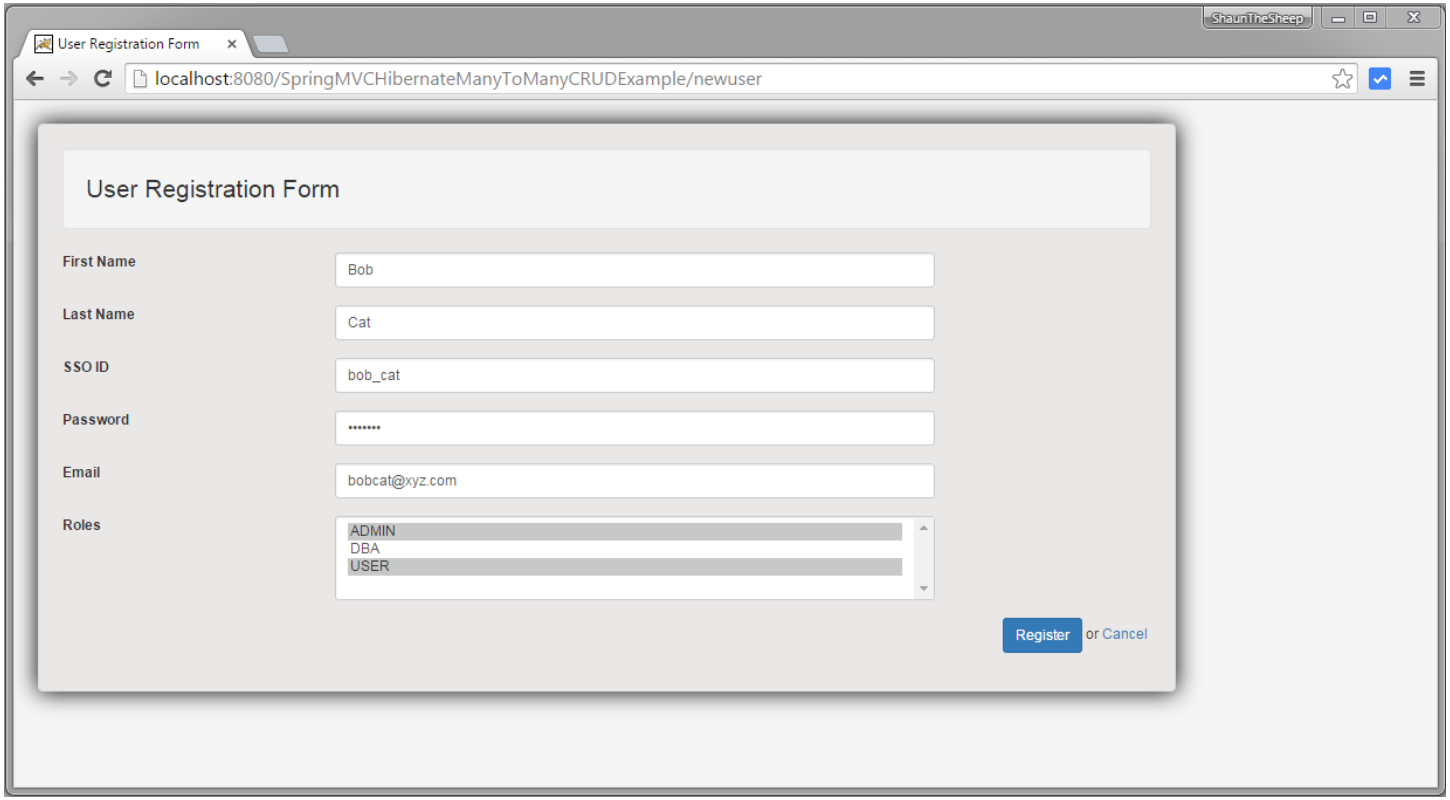
Spring Boot + AngularJS + Spring Data + JPA CRUD App Example

Spring Boot Rest API Example

Spring Boot WAR deployment example

Spring Boot Introduction + hello world example

Secure Spring REST API using OAuth2



Complete example with detailed explanation is presented below.

Other interesting posts you may like

- Spring Boot+AngularJS+Spring Data+Hibernate+MySQL CRUD App
- Spring Boot REST API Tutorial

- Spring Boot WAR deployment example
- Spring Boot Introduction + Hello World Example
- Secure Spring REST API using OAuth2
- Secure Spring REST API using Basic Authentication
- AngularJS+Spring Security using Basic Authentication
- Spring 4 MVC+JPA2+Hibernate Many-to-many Example
- Spring 4 Caching Annotations Tutorial
- Spring 4 Cache Tutorial with EhCache
- Spring 4 Email Template Library 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 Integration Example
- Spring MVC 4+Spring Security 4 + Hibernate 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+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

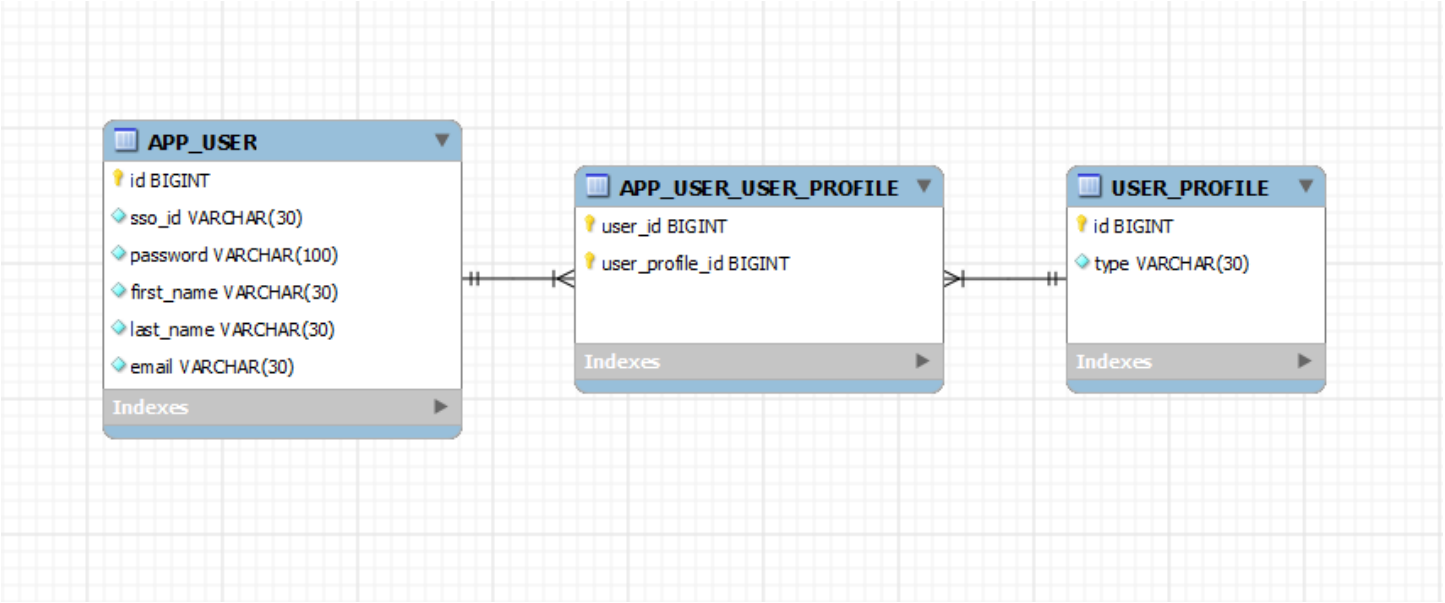
- TestNG Mockito Integration Example Stubbing Void Methods
- Maven surefire plugin and TestNG Example

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
id BIGINT
sso_id VARCHAR(30)
password VARCHAR(100)
first_name VARCHAR(30)
last_name VARCHAR(30)
email VARCHAR(30)
Indexes
APP_USER_USER_PROFILE
user_id BIGINT
user_profile_id BIGINT
Indexes
USER_PROFILE
id BIGINT
type VARCHAR(30)
Indexes

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.

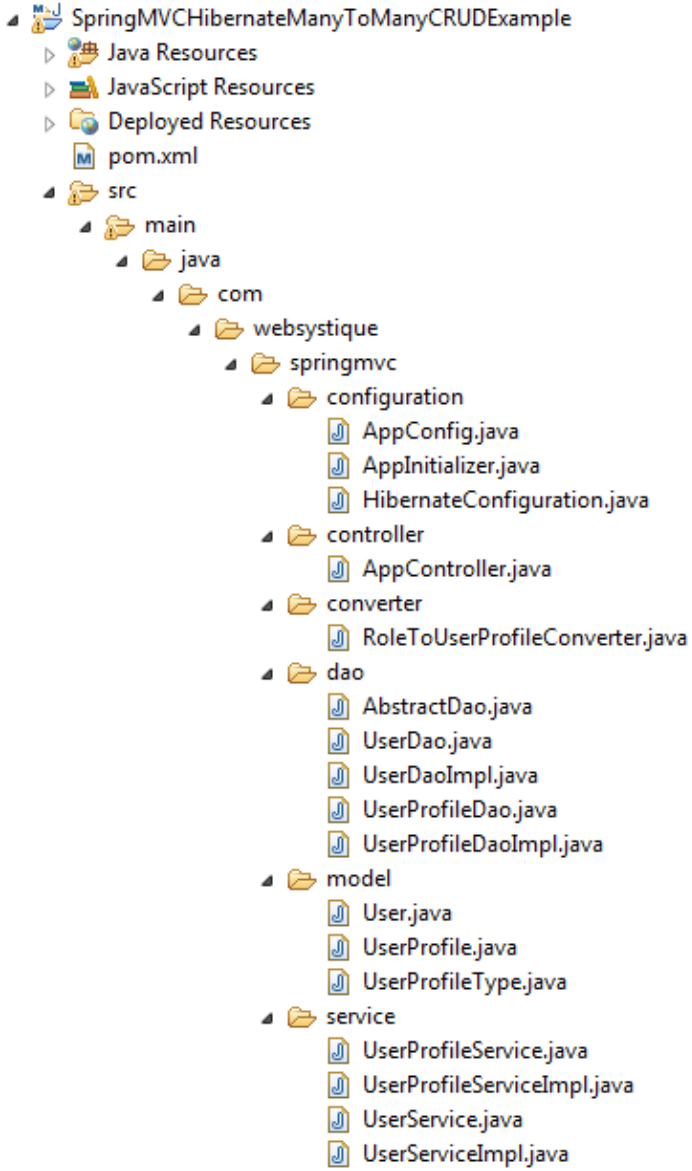
```
create table APP_USER (
   id BIGINT NOT NULL AUTO_INCREMENT,
   sso_id VARCHAR(30) NOT NULL,
   password VARCHAR(100) NOT NULL,
   first_name VARCHAR(30) NOT NULL,
   last_name  VARCHAR(30) NOT NULL,
   email VARCHAR(30) NOT NULL,
   PRIMARY KEY (id),
   UNIQUE (sso_id)
);


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');
```

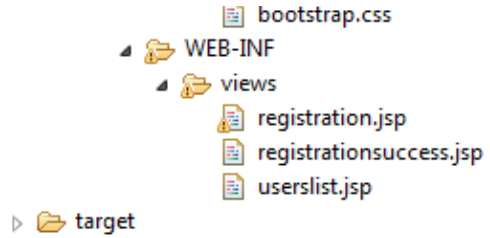
```
INSERT INTO USER_PROFILE(type)
VALUES ('DBA');

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:


SpringMVCHibernateManyToManyCRUDExample
Java Resources
JavaScript Resources
Deployed Resources
pom.xml
src
main
java
com
websystique
springmvc
configuration
AppConfig.java
AppInitializer.java
HibernateConfiguration.java
controller
AppController.java
converter
RoleToUserProfileConverter.java
dao
AbstractDao.java
UserDao.java
UserDaoImpl.java
UserProfileDao.java
UserProfileDaoImpl.java
model
User.java
UserProfile.java
UserProfileType.java
service
UserProfileService.java
UserProfileServiceImpl.java
UserService.java
UserServiceImpl.java
resources
application.properties
messages.properties
webapp
static
css
app.css



Step 3: Update pom.xml to include required dependencies

```
<project xsi:schemaLocation="<a class="vglnk" href="http://maven.apache.org/POM/4.0.0" rel='
    xmlns="<a class="vglnk" href="http://maven.apache.org/POM/4.0.0" rel="nofollow"><span>ht

    <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.version>
        <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>
            <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</warSourceDirectory>
                        <warName>SpringMVCHibernateManyToManyCRUDExample</warName>
                        <failOnMissingWebXml>false</failOnMissingWebXml>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
        <finalName>SpringMVCHibernateManyToManyCRUDExample</finalName>
    </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
    @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;
    }

    @Override
    public String toString() {
```

```
        return "User [id=" + id + ", ssoId=" + ssoId + ", password=" + password
                + ", firstName=" + firstName + ", lastName=" + lastName
                + ", email=" + email + "]";
    }

}
```

Look at how userProfiles 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.getClass().getGenericSupe

()).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("firstName"));
        criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);//To avoid duplicates.
        List<User> users = (List<User>) criteria.list();

        // No need to fetch userProfiles since we are not showing them on list page. Let the
        // Uncomment below lines for eagerly fetching of userProfiles if you want.
        /*
        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>implements UserProf

    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 hibernate update explic
     * Just fetch the entity from db and update it with proper values within transaction.
     * 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() == id)));
    }

}
```

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.websystique.springmvc.model" })
        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.diale
        properties.put("hibernate.show_sql", environment.getRequiredProperty("hibernate.show
        properties.put("hibernate.format_sql", environment.getRequiredProperty("hibernate.fo
        return properties;
    }

    @Bean
    @Autowired
    public HibernateTransactionManager transactionManager(SessionFactory s) {
       HibernateTransactionManager txManager = new HibernateTransactionManager();
       txManager.setSessionFactory(s);
```

```
        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.GET)
    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 request for
     * 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 implementing custom
         * and applying it on field [sso] of Model class [User].
         *
         * Below mentioned peace of code [if block] is to demonstrate that you can fill cust

validation
         * framework as well while still using internationalized messages.
         *
         */
        if(!userService.isUserSSOUnique(user.getId(), user.getSsoId())){
            FieldError ssoError =new FieldError("user","ssoId",messageSource.getMessage("nor

String[]{user.getSsoId()}, Locale.getDefault()));
            result.addError(ssoError);
            return "registration";
        }

        userService.saveUser(user);

        model.addAttribute("success", "User " + user.getFirstName() + " "+ user.getLastName(

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 request for
     * 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 SSO_ID in UI which is a
        if(!userService.isUserSSOUnique(user.getId(), user.getSsoId())){
            FieldError ssoError =new FieldError("user","ssoId",messageSource.getMessage("nor

String[]{user.getSsoId()}, Locale.getDefault()));
            result.addError(ssoError);
            return "registration";
        }*/

        userService.updateUser(user);

        model.addAttribute("success", "User " + user.getFirstName() + " "+ user.getLastName(
        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 value.
```

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 objects.
 */
@Component
public class RoleToUserProfileConverter implements Converter<Object, UserProfile>{

    @Autowired
    UserProfileService userProfileService;

    /**
     * Gets UserProfile by Id
     * @see org.springframework.core.convert.converter.Converter#convert(java.lang.Object)
     */
```

```
    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.WebMvcConfigurerAdapter;
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 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[Roles] to UserProfiles i
     */
    @Override
    public void addFormatters(FormatterRegistry registry) {
        registry.addConverter(roleToUserProfileConverter);
    }


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

    /**Optional. It's only required when handling '.' in @PathVariables which otherwise ignc

@PathVaidables argument.
     * It's a known bug in Spring [<a class="vglnk" href="https://jira.spring.io/browse/SPR-
     * 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.FormattingConversionS

    <property name="converters">
        <list>
            <bean id="roleToUserProfile" class="com.websystique.springmvc.converter.RoleToUs
        </list>
    </property>
</bean>
```

Add initializer class:

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

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInit

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" pageEncoding="ISO-8859-
<%@ taglib prefix="c" uri="<a class="vglnk" href="http://java.sun.com/jsp/jstl/core" rel="nc
<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>
    <link href="<c:url value='/static/css/app.css' />" rel="stylesheet"></link>
</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}' />" class="btn

custom-width">edit</a></td>
                        <td><a href="<c:url value='/delete-user-${user.ssoId}' />" class="bt

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" pageEncoding="ISO-8859-
<%@ taglib prefix="form" uri="<a class="vglnk" href="http://www.springframework.org/tags/for
<%@ taglib prefix="c" uri="<a class="vglnk" href="http://java.sun.com/jsp/jstl/core" rel="nc

<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>
    <link href="<c:url value='/static/css/app.css' />" rel="stylesheet"></link>
</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-lable" for="firstName">First Name</label>
                <div class="col-md-7">
                    <form:input type="text" path="firstName" id="firstName" class="form-cont
                    <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-lable" for="lastName">Last Name</label>
                <div class="col-md-7">
                    <form:input type="text" path="lastName" id="lastName" class="form-contro
                    <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-lable" for="ssoId">SSO ID</label>
                <div class="col-md-7">
                    <c:choose>
                        <c:when test="${edit}">
                            <form:input type="text" path="ssoId" id="ssoId" class="form-cont

input-sm" disabled="true"/>
                        </c:when>
                        <c:otherwise>
                            <form:input type="text" path="ssoId" id="ssoId" class="form-cont

input-sm" />
                            <div class="has-error">
                                <form:errors path="ssoId" class="help-inline"/>
                            </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">Password</label>
                <div class="col-md-7">
                    <form:input type="password" path="password" id="password" class="form-cc

/>
                    <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-lable" for="email">Email</label>
                <div class="col-md-7">
                    <form:input type="text" path="email" id="email" class="form-control inpu
                    <div class="has-error">
                        <form:errors path="email" class="help-inline"/>
                    </div>
                </div>
            </div>
        </div>

        <div class="row">
            <div class="form-group col-md-12">
                <label class="col-md-3 control-lable" for="userProfiles">Roles</label>
                <div class="col-md-7">
                    <form:select path="userProfiles" items="${roles}" multiple="true" itemVa

itemLabel="type" class="form-control input-sm" />
                    <div class="has-error">
                        <form:errors path="userProfiles" class="help-inline"/>
                    </div>
                </div>
            </div>
        </div>

        <div class="row">
            <div class="form-actions floatRight">
                <c:choose>
                    <c:when test="${edit}">
                        <input type="submit" value="Update" class="btn btn-primary btn-sm"/>
```

```
href="<c:url value='/list' />">Cancel</a>
                    </c:when>
                    <c:otherwise>
                        <input type="submit" value="Register" class="btn btn-primary btn-sm'

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" pageEncoding="ISO-8859-
<%@ taglib prefix="c" uri="<a class="vglnk" href="http://java.sun.com/jsp/jstl/core" rel="nc


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

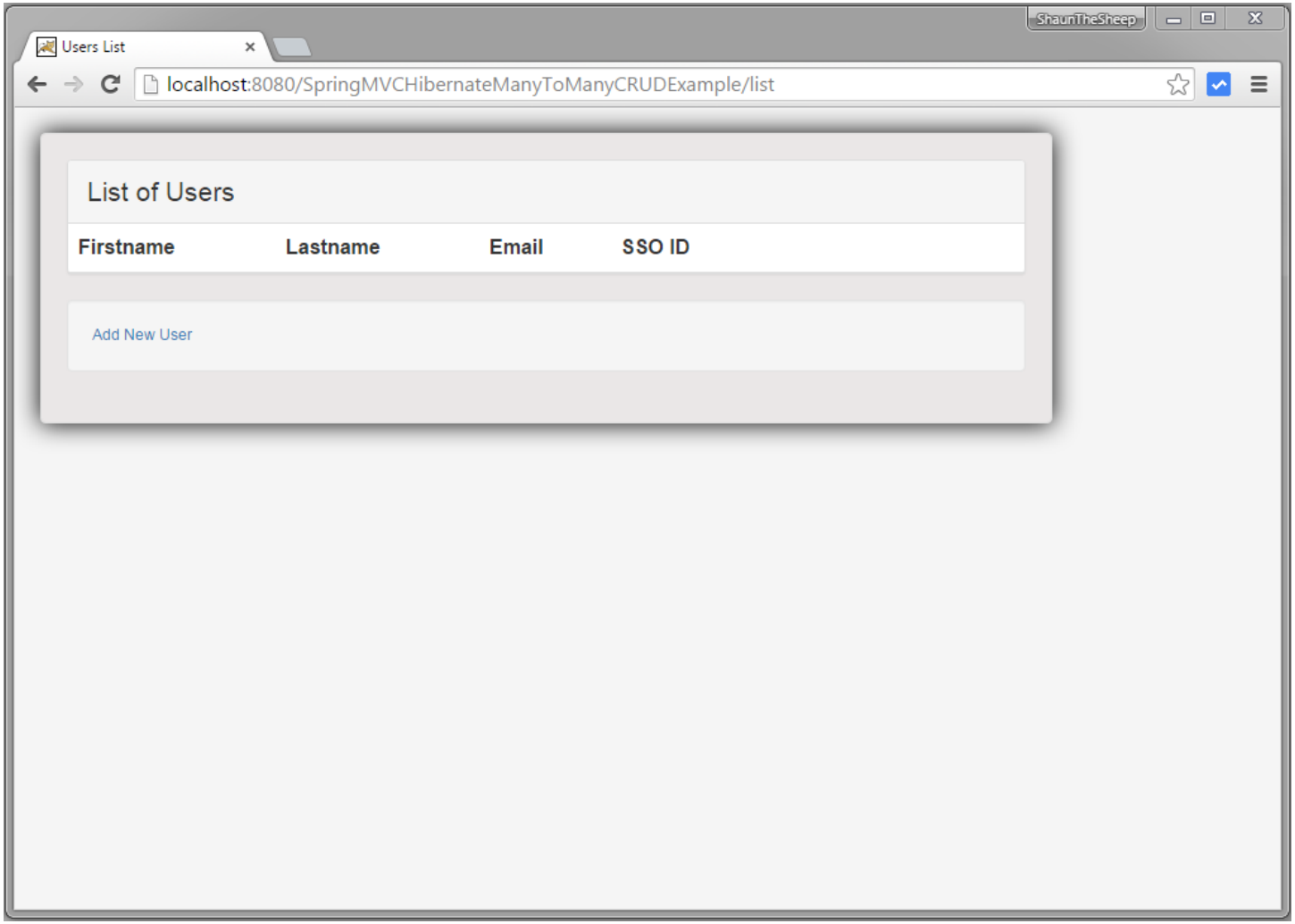
app.css

```
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;
}
```

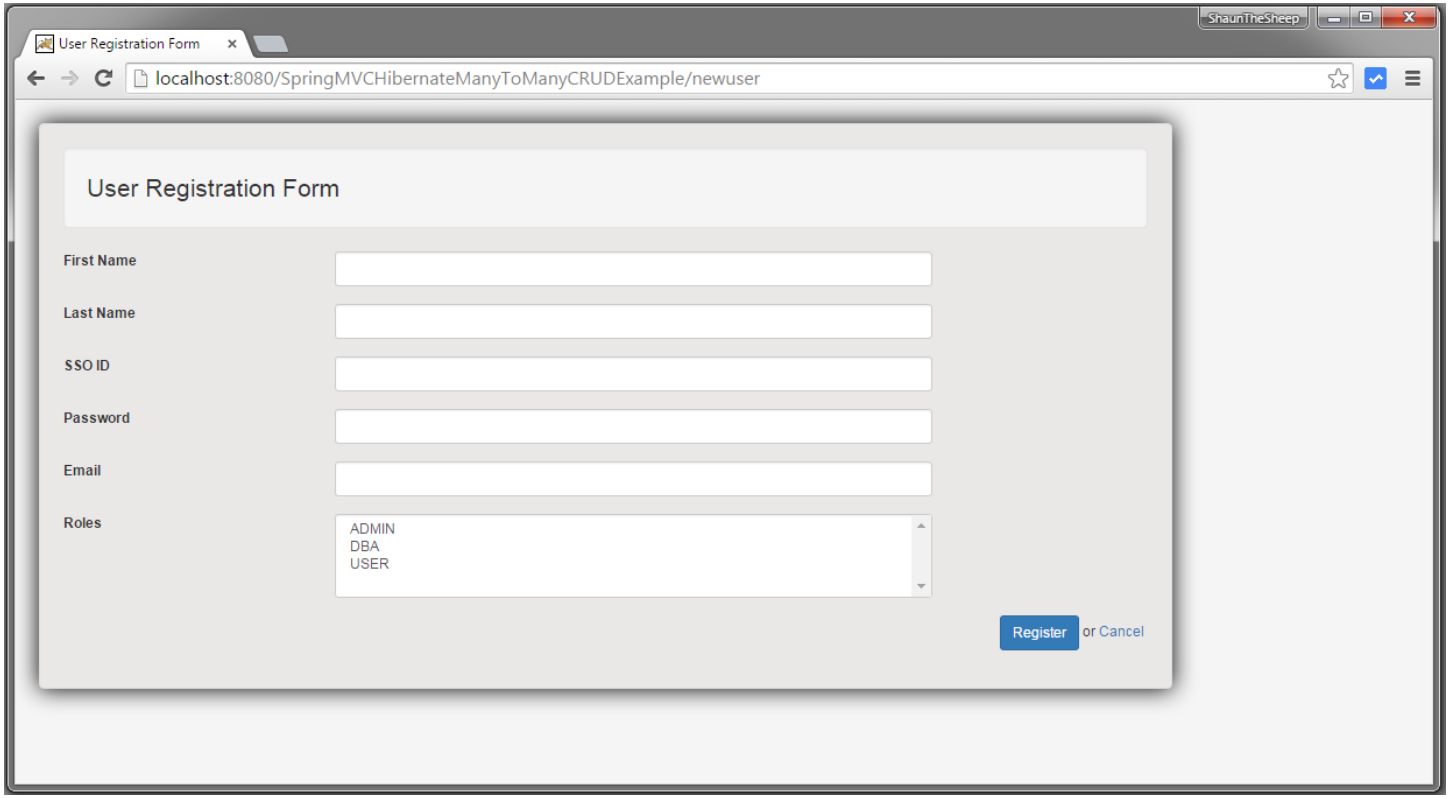
Step 12: Build, deploy and Run Application

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.

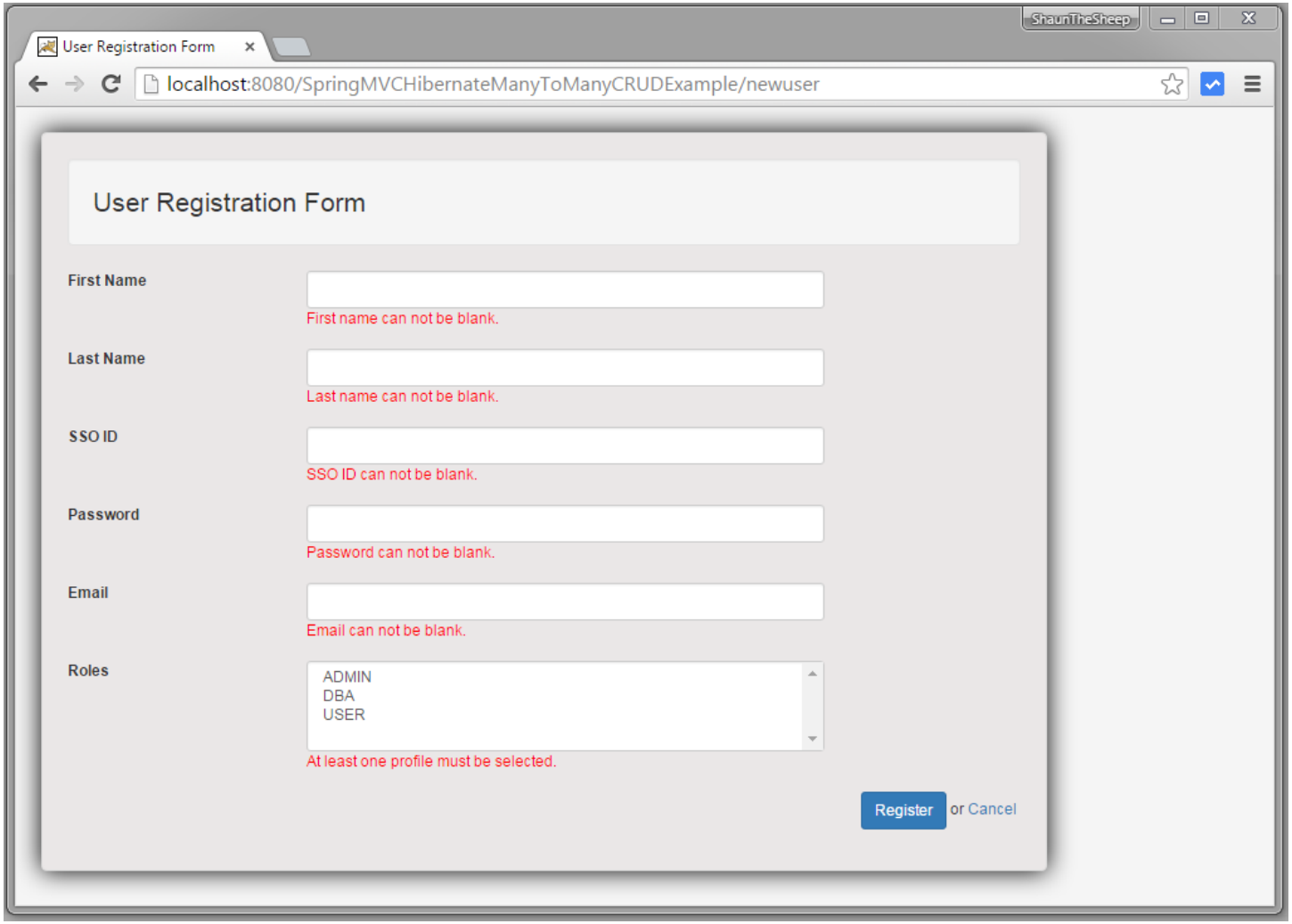
Open browser and browse at http://localhost:8080/SpringMVCHibernateManyToManyCRUDExample/



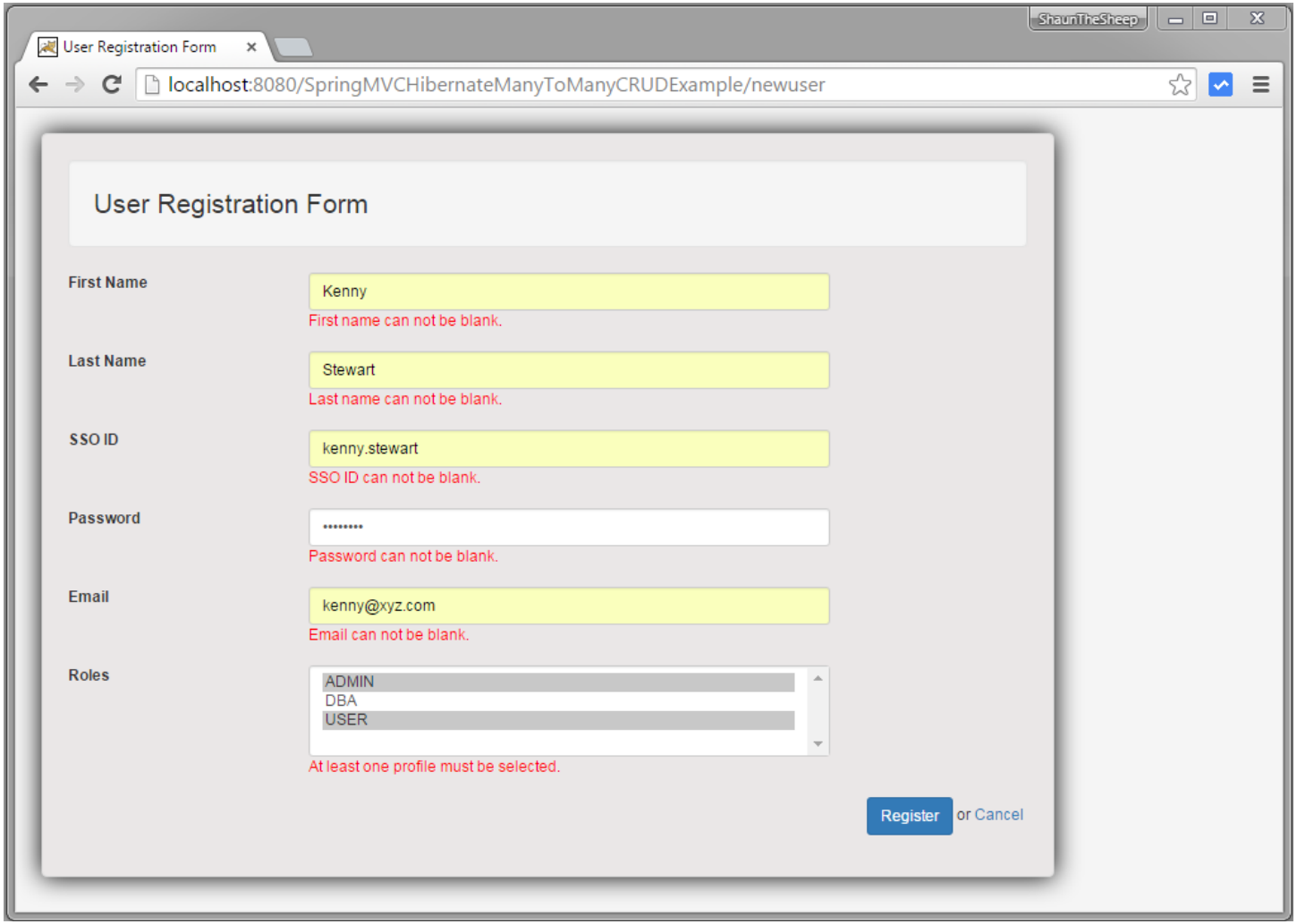
Click on ‘Add New User’



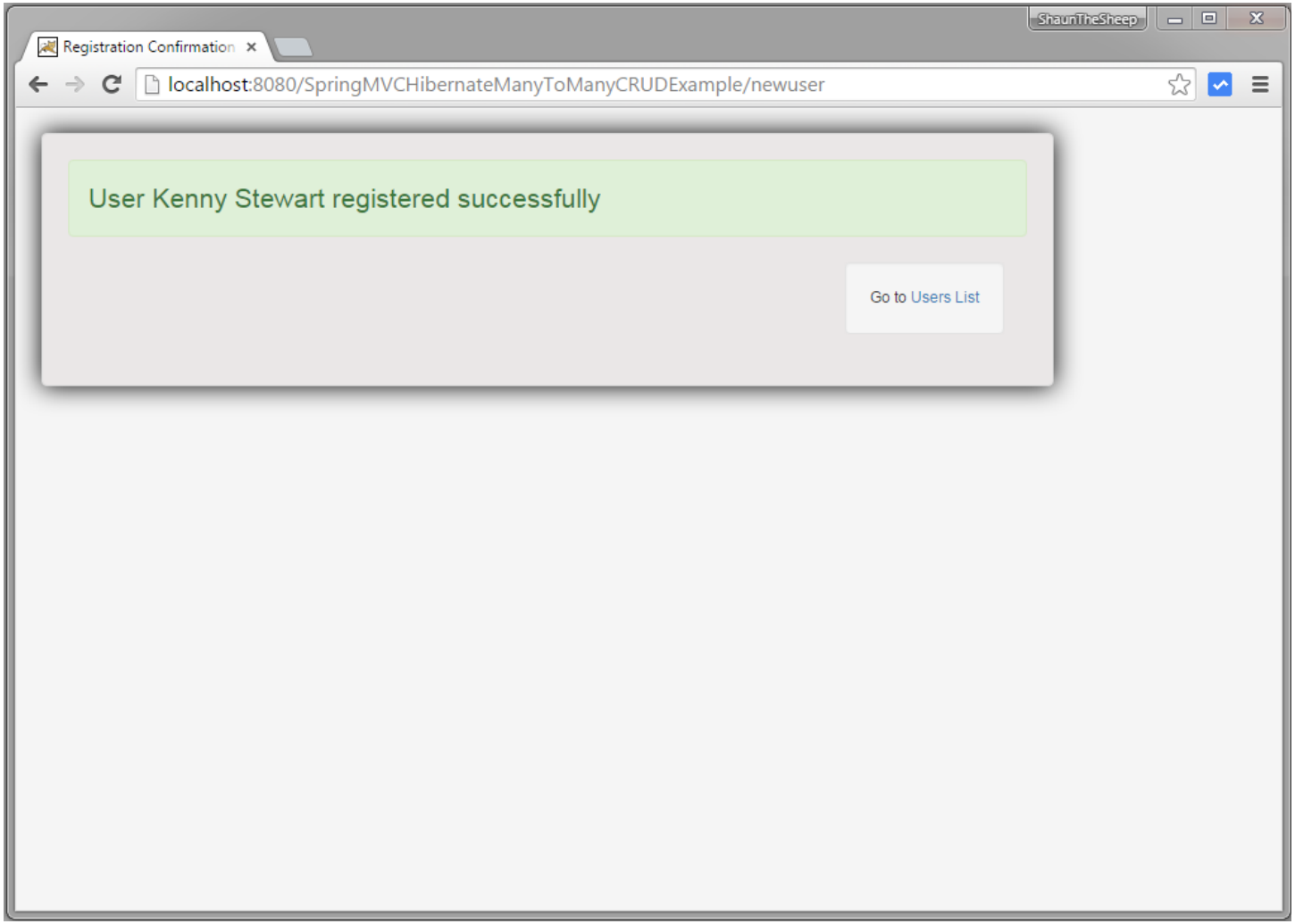
Submit without filling anything.



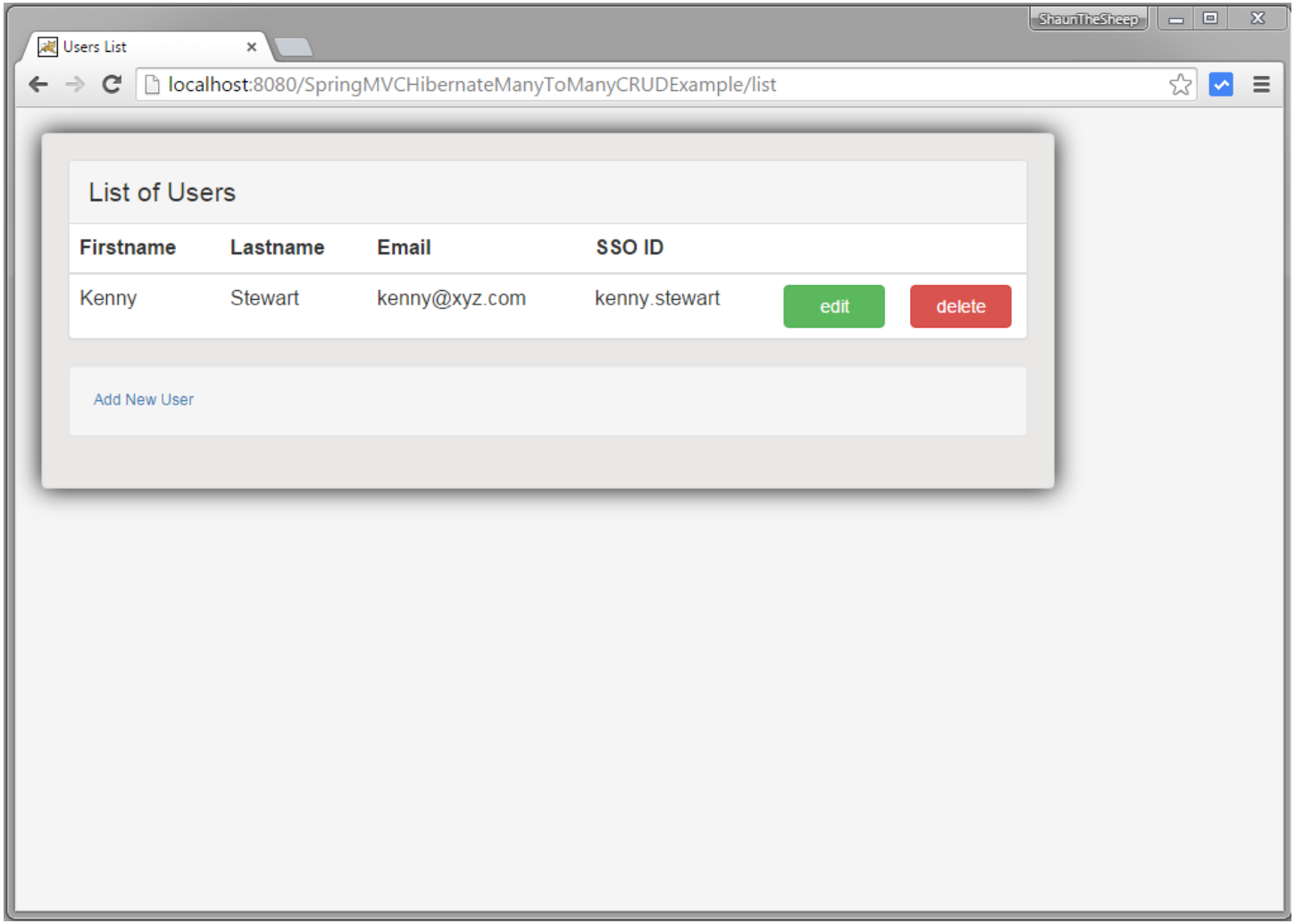
Fill in details


User Registration Form
localhost:8080/SpringMVCHibernateManyToManyCRUDExample/newuser
User Registration Form
First Name
Kenny
First name can not be blank.
Last Name
Stewart
Last name can not be blank.
SSO ID
kenny.stewart
SSO ID can not be blank.
Password
Password can not be blank.
Email
kenny@xyz.com
Email can not be blank.
Roles
ADMIN
DBA
USER
At least one profile must be selected.
Register
or Cancel

Submit.



Click on ‘Users List’ link.



Check the database at this moment.

SQL File 1*

```
select * from app_user;
```

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
NULL	NULL	NULL	NULL	NULL	NULL

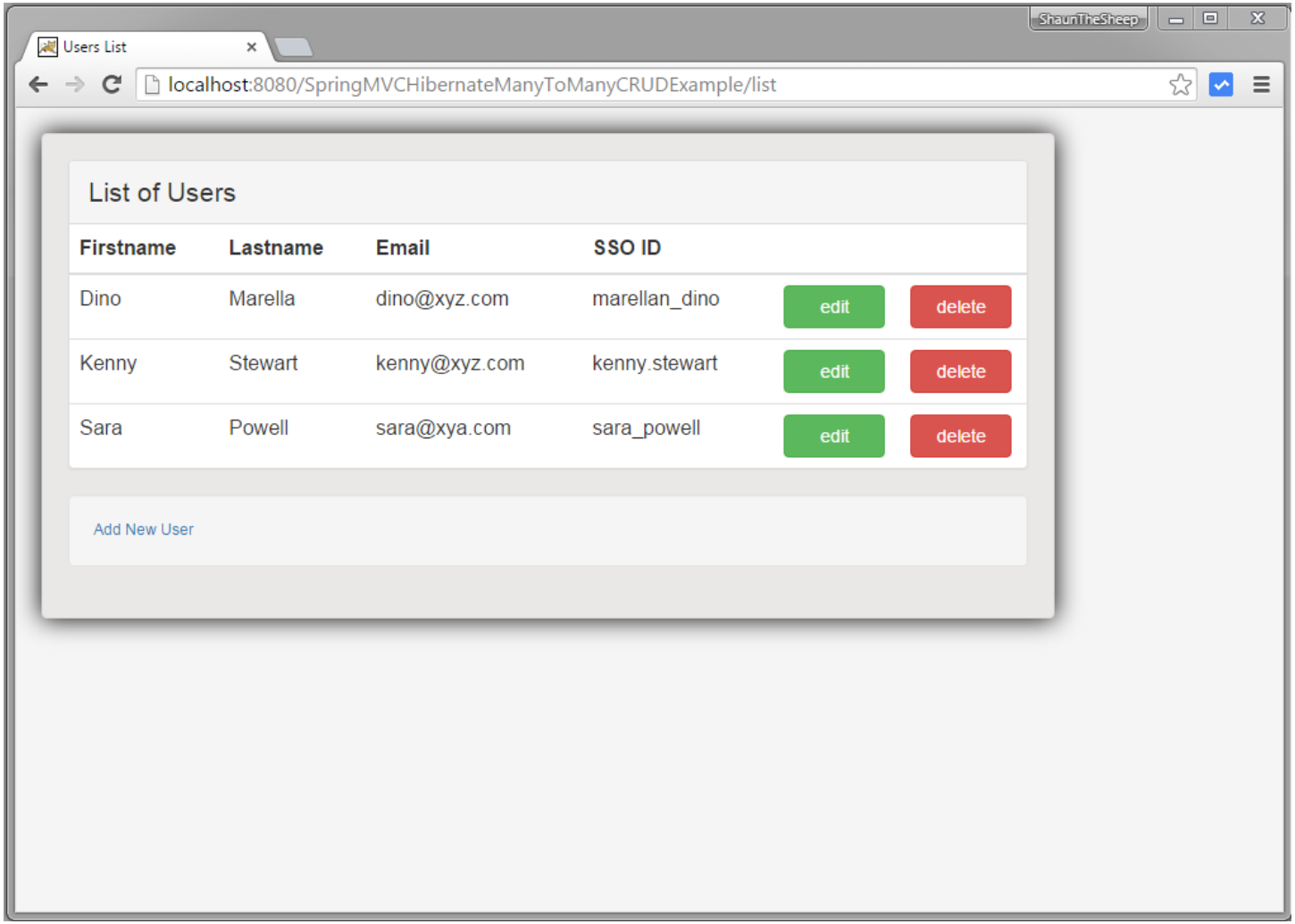
SQL File 1*

```
select * from app_user_user_profile;
```

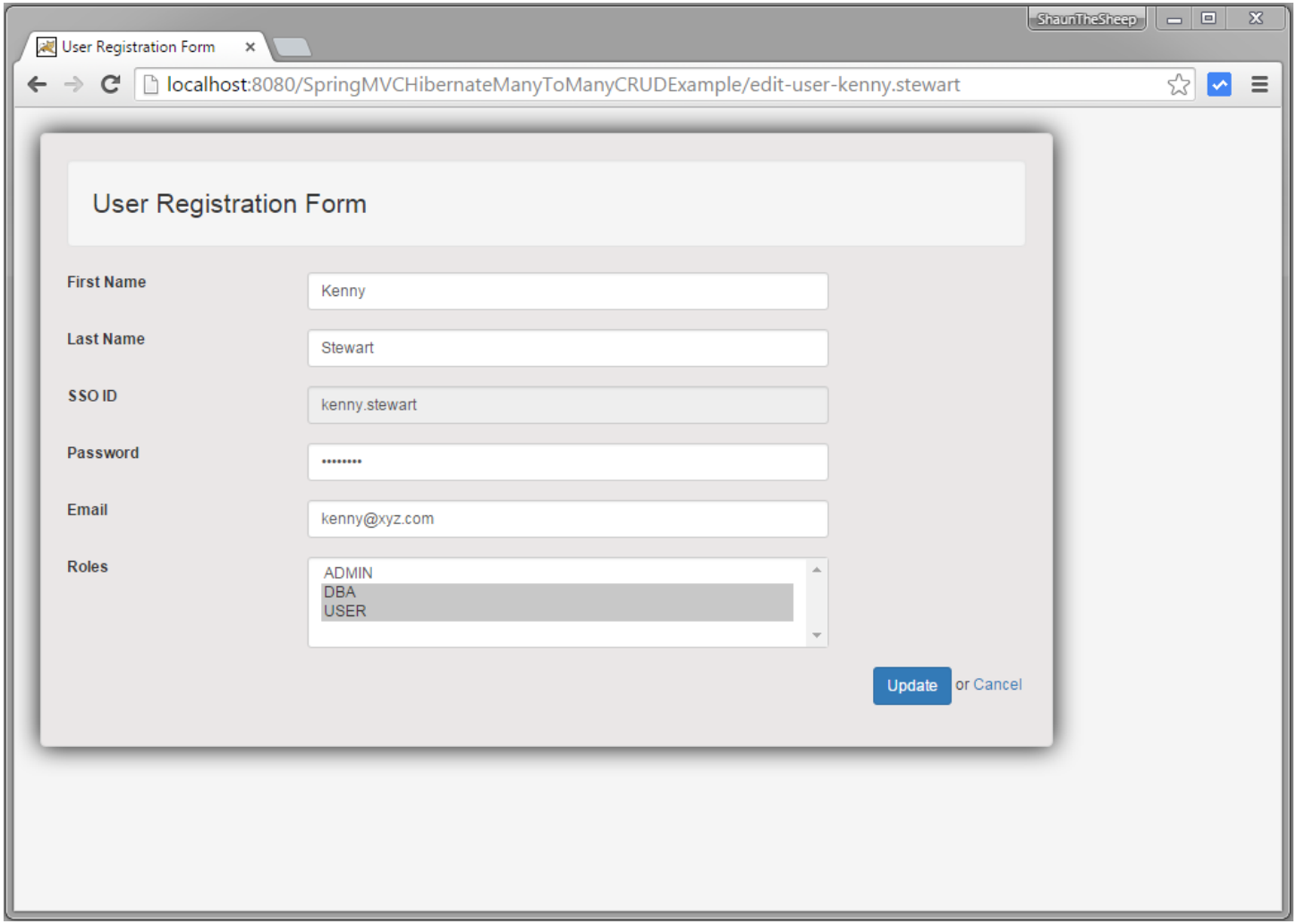
Result Set Filter: Edit: Export/Import: Wrap Cell Content:

user_id	user_profile_id
1	1
1	2
NULL	NULL

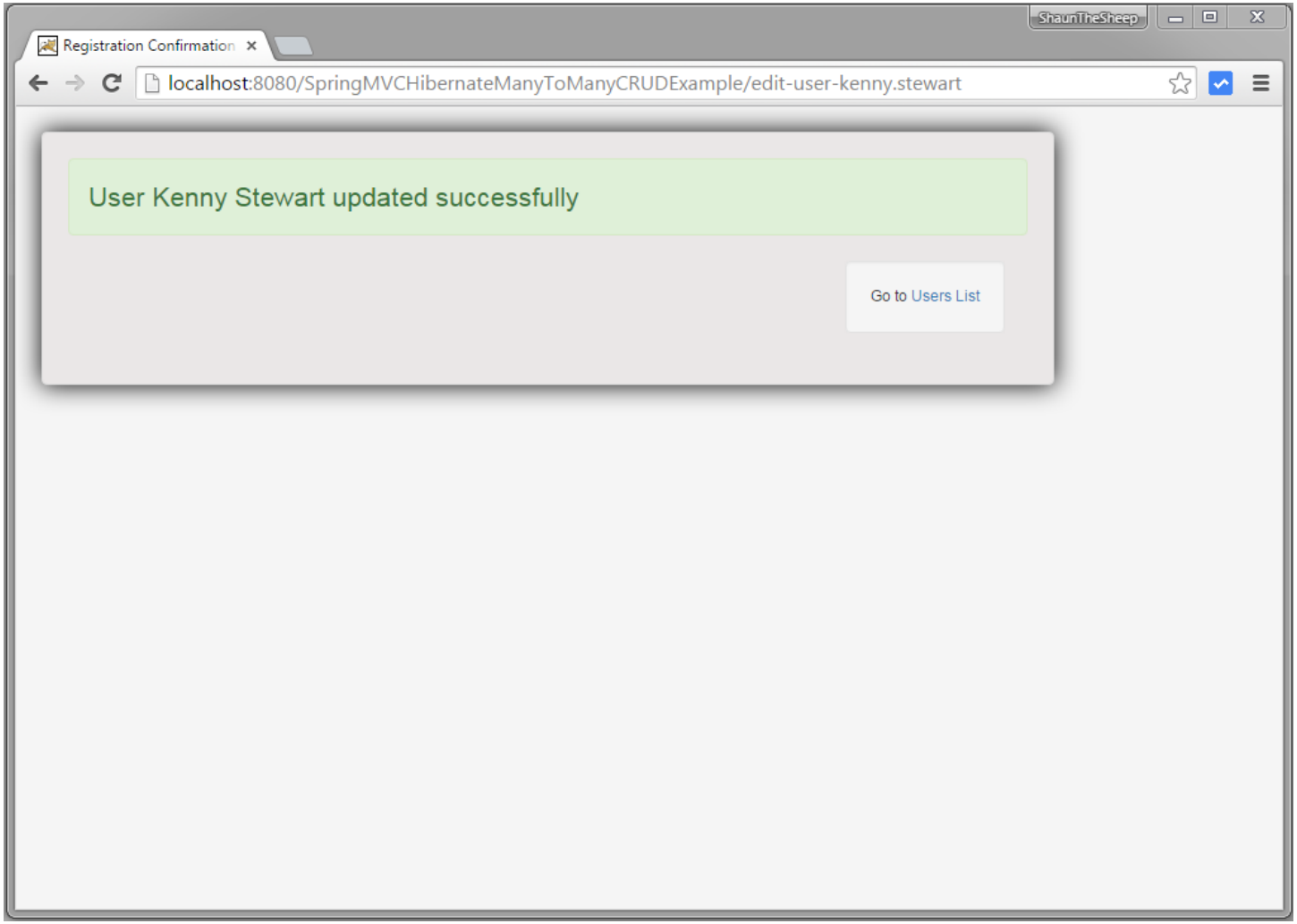
Add more users.



Click Edit button for User Kenny. Change Roles.


User Registration Form
localhost:8080/SpringMVCHibernateManyToManyCRUDExample/edit-user-kenny.stewart
User Registration Form
First Name
Kenny
Last Name
Stewart
SSO ID
kenny.stewart
Password
Email
kenny@xyz.com
Roles
ADMIN
DBA
USER
Update
or Cancel

Submit.



Verify the database.

SQL File 1*

```
select * from app_user;
```

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

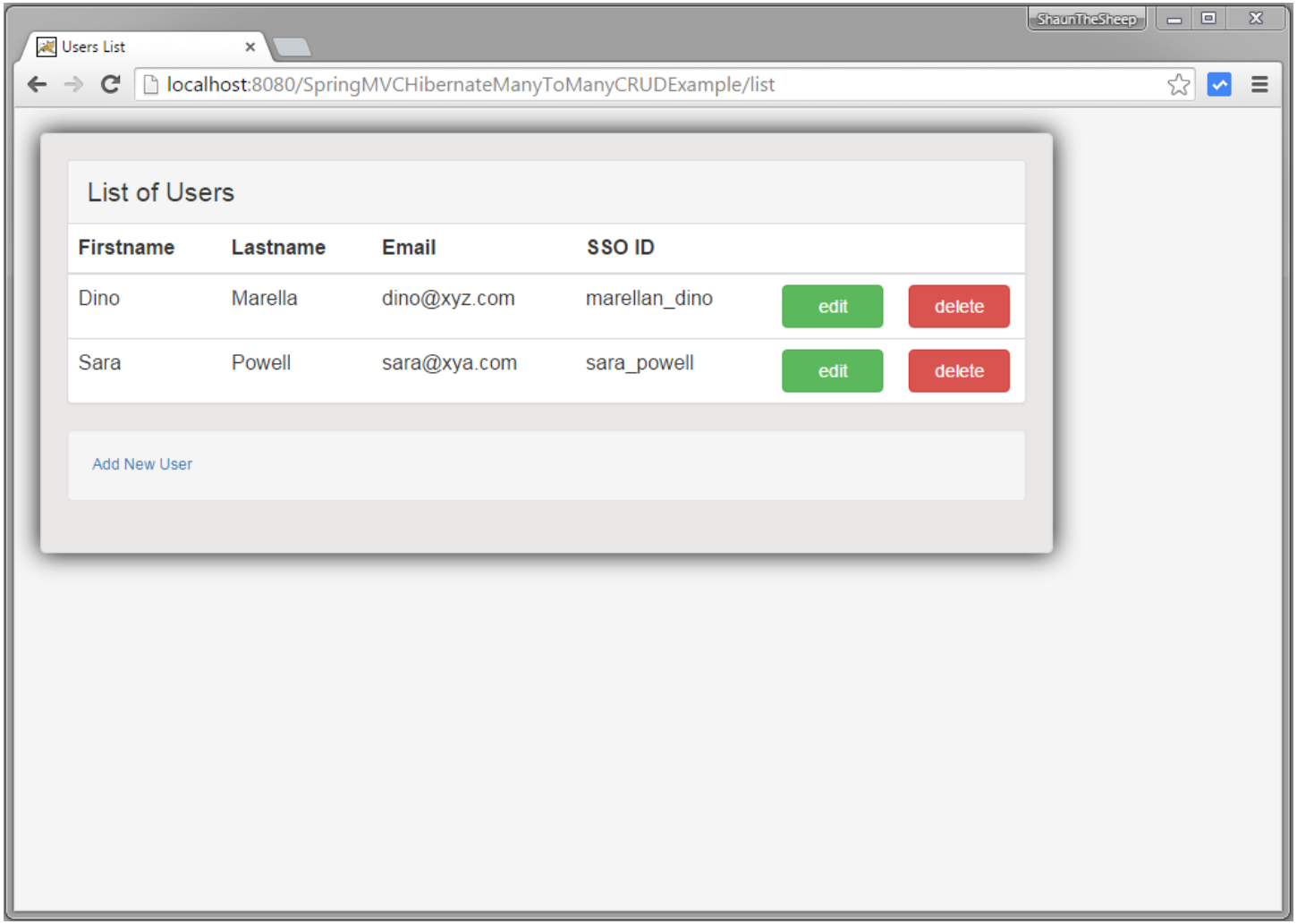
SQL File 1*

```
select * from app_user_user_profile order by user_id;
```

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 :

SQL File 1*

```
select * from app_user;
```

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

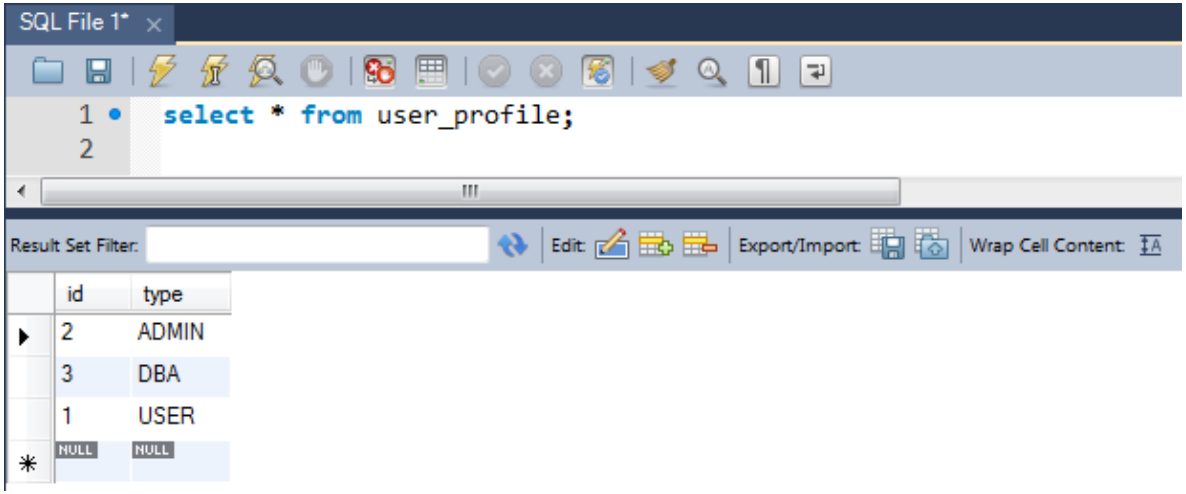
id	sso_id	password	first_name	last_name	email
2	sara_powell	sara123	Sara	Powell	sara@xya.com
3	marellan_dino	dino123	Dino	Marella	dino@xyz.com
NULL	NULL	NULL	NULL	NULL	NULL

SQL File 1*

```
select * from app_user_user_profile;
```

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

user_id	user_profile_id
2	1
2	3
3	3
NULL	NULL



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 improve further our learning process.

Related Posts:

1. **Spring MVC 4 + Spring Security 4 + Hibernate Example**
2. **Spring Security 4 Hibernate Integration Annotation+XML Example**
3. **Spring 4 MVC HelloWorld Tutorial – Annotation/JavaConfig Example**
4. **Spring 4 MVC + JMS + ActiveMQ annotation based Example**

springmvc. permalink.

← Spring Security 4 Role Based Login Example

Spring MVC @RequestBody @ResponseBody Example →

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Jayanta Pramanik • 3 years ago

Hi Admin,

I'm not able to run successfully this application, everytime when I click for Add User this is showing HTTP Error 400 !

Please guide me to resolve this !

Thanks

Jayanta P