J2EE

Day 7

J2EE Topics Covered

Spring Framework + Spring Webflow
+ Spring Security + JPA (Hibernate)
+ JSF 2.0 (PrimeFaces) + Apache Maven 2
+ Apache Tomcat + MySQL + Eclipse

Bugs in Spring Security

Change in pom.xml

```
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>3.1.3.RELEASE</version>
    </dependency>
    <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>3.1.3.RELEASE</version>
    </dependency>
```

Make sure they match

```
    <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>3.1.3.RELEASE</version>
    </dependency>
```

Step1

Create an Entity

Under src/main/java
Create the following packages

com.example.j2eedemo.entities

Don’t create com.example.j2eedemo.domain

Step2

Inside com.example.j2eedemo.domain,
Create UserEntity.java

```
public class UserEntity
{

        private String firstName;

        private String lastName;

        private String userName;

        private String password;


}
```

Step3

Create public getters and settters for these

This is called a java bean

private fields and public accessor methods

Step4

We need to bind it to our view

In main-flow.xml :

Just before the first view state, create var tag so that it looks like this :

```
    <var name="com.example.j2eedemo.
domain" class="UserEntity"/>
```

Step5

Add the model to which the view is bound to

Add model="user"

to

```
<view-state id="signup" view="signup.xhtml"
model="user">
```

Step6

We use expression language to model the field in our view

${user.firstName}

Similarly for other fields, make sure you reference the name exactly like in the model

Don’t need the value attribute in pass2 so just remove it

Step7

Compile, and deploy

Make sure persistence.xml looks like this

```
<persistence-unit name="default" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <class>com.example.j2eedemo.domain.UserEntity</class>
<!--    <properties> -->
<!--            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" /> -->
<!--            <property name="hibernate.hbm2ddl.auto" value="create-drop" /> -->
<!--            <property name="hibernate.show_sql" value="true" /> -->
<!--    </properties> -->
   </persistence-unit>
```

Advanced Topics Step1

In case you have created a project without using the Maven Project Wizard, you can right-click->Configure>Convert to Maven Project

Advanced Topics Step2

Right-click->SpringTools->Add Spring nature to project

Persistence Step1

We need to add the Spring Data Project library.
We will be using the JPA flavor

With pom.xml open in Dependencies tab -> Click on Add and search for org. springframework

Add spring-data-jpa

Step2

We need a JPA implementation, so we can grab the hibernate-entitymanager which does all the work of persisting to the database

With pom.xml open in Dependencies tab -> Click on Add and search for org.hibernate

Add hibernate-entitymanager

Step3

Next, we need to add the spring-test framework

With pom.xml open in Dependencies tab -> Click Add -> search for org.springframework -> Add spring-test

Step4

Next, we need to add the H2 database, which is an embedded java database

With pom.xml open in Dependencies tab -> Click Add -> search for com.h2database

Add h2

Step5

If you look at the Dependency Hierarchy in pom, you will see that maven has resolved all the dependencies of the libraries we’ve included.

Step6

Right-click on datasource-config.xml and open with Spring Config Editor

Then switch to the namespaces tab and select the jdbc checkbox

By adding the jdbc namespace to the config, we get some assistance with the configuration

Also, add the jpa namespace for later

Step7

Add the following line above entityManagerFactory in datasource-config.xml

```
    <!-- Database -->
    <jdbc:embedded-database id="datasource"
type="H2"></jdbc:embedded-database>
```

Step8

Make sure the following config is present in datasource-config.xml

```
      <!-- EntityManager -->
   <bean id="entityManagerFactory"
        class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="persistenceUnitName" value="default"></property>
   </bean>
   <!-- TransactionManager -->
   <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="dataSource" ref="dataSource" />
        <property name="entityManagerFactory" ref="entityManagerFactory" />
   </bean>
<tx:annotation-driven transaction-manager="transactionManager" />
```

Step9

Add the following below TransactionManager

<!-- Jpa Respositories →

<jpa:repositories base-package="com.
example.j2eedemo"></jpa:repositories>

Step10

Change your persistence.xml to look as follows :

The first property specifies what database we’re using.

The second property tells Hibernate to create our tables when the application starts.

The third property tells Hibernate to show the SQL it uses for debugging and it prints in the console.

```
     <persistence-unit name="test" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <properties>
           <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
           <property name="hibernate.hbm2ddl.auto" value="create-drop" />
           <property name="hibernate.show_sql" value="true" />
      </properties>
  </persistence-unit>
```

Step11

Right-click on the com.example.package and create a new sub-package called entities

Right-click on entities and create an entity called Post

Create the following properties :

postId, title, postDate and generate getters and setters for them.

Step12

Annotate these fields with JPA annotations.

```
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="POST")
public class Post
{
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name="POST_ID")
        Integer postId;
        @Column(name="TITLE")
        String title;
        @Column(name="POST_DATE")
        Date postDate;
```

Step12

Create your repository so that it is an interface that extends JpaRepository

The interface must have some Generic type parameters that we must specify.

The first being the type of object.
The second being the type of the object’s id.

Step13

Create Post Repository in a package called com.example.j2eedemo.repositories so that it looks like this :

```
public interface PostRepository extends
JpaRepository<Post, Integer> {


}
```

Step14

Create a folder called src/test/java

In src/test/java, create PostRepositoryTest by selecting New ->JUnit Test Case and put it under the com.example.j2eedemo package

It will ask if we want to add JUnit 4 to the classpath, so we say yes

Step15

Copy the following code into PostRepositoryTest

```
package com.example.j2eedemo;

import static org.junit.Assert.*;

import java.util.Date;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.example.j2eedemo.entities.Post;
import com.example.j2eedemo.repositories.PostRepository;
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath:WEB-INF/applicationContext.xml")
```

Step16

Copy the following code into PostRepositoryTest

```
public class PostRepositoryTest {


        @Autowired
        PostRepository repository;
 @Test
        public void test() {
   Post post = new Post();
   post.setPostDate(new Date());
   post.setTitle("First Post");
   repository.save(post);



   Post dbpost = repository.findOne(post.getPostId());
   assertNotNull(dbpost);
   System.out.println(dbpost.getTitle());

   fail("Not yet implemented");
        }


}
```

Step17

Change entityManagerFactory in datasource-config.xml to look as follows :

```
<bean id="entityManagerFactory"
        class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="jpaVendorAdapter">
                <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                        <property name="showSql" value="true" />
                        <property name="generateDdl" value="true" />
                        <property name="databasePlatform" value="org.hibernate.dialect.H2Dialect" />
                </bean>
        </property>
        <property name="dataSource" ref="datasource" />
        <property name="persistenceUnitName" value="default"></property>
  </bean>
```

Step18

Make sure that the name of the persistenceUnityName property under entityManagerFactory in datasource-config.xml is the same as the name of the persistence-unit in persistence.xml

Step19

Right-click on src/main/java

Right click on src/main/java -> Click on New->Package

Name it com.example.j2eedemo

Appendix : References

To configure your project as a maven project, you may refer to the following link :

http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

Appendix : References

Hibernate : www.hibernate.org

```
<!--      <persistence-unit name="default" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
         <properties>
                 <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
        <property name="hibernate.hbm2ddl.auto" value="create-drop" />
        <property name="hibernate.show_sql" value="true" />
         </properties>
    <class>com.example.j2eeapp.domain.UserEntity</class>
   </persistence-unit>  -->
```