**What this project does??**

**This is Full Spring MVC and Hibernate application with Java based configuration(i.e. by using annotations).**

Note that this is fifth project in the series of Spring MVC which explains functioning of controller layer and service layer and DAO layer and how database operation takes place in Spring Hibernate project. **It carries out CRUD (i.e. Create, Read, Update and Delete) operations on database tables by using Hibernate.** Before moving further please go through projects 'SpringMVCApplicationFlow' which is first project in Spring MVC Series. It does not contains any web related operations, it explains functionality of 'spring-servlet.xml' in Spring MVC context, ' SpringMVCHelloWorld' which explains how controller layer functions in Spring MVC, 'SpringMVCArtihmeticOperationsDemo' which explains functioning of controller and service layer in Spring Framework, and 'SpringMVCHibernateWithXMLFull' , it explains full Spring+Hibernate database table CRUD operation **with full XML configuration** which includes all three layers of Spring MVC viz. Controller, Service and DAO.

Thus

' SpringMVCApplicationFlow': explains functionality of 'spring-servlet.xml' in Spring MVC context

' SpringMVCHelloWorld': explains how controller layer functions in Spring MVC

'SprinArithematicOperation': explains how controller layer and service layer functions in Spring MVC

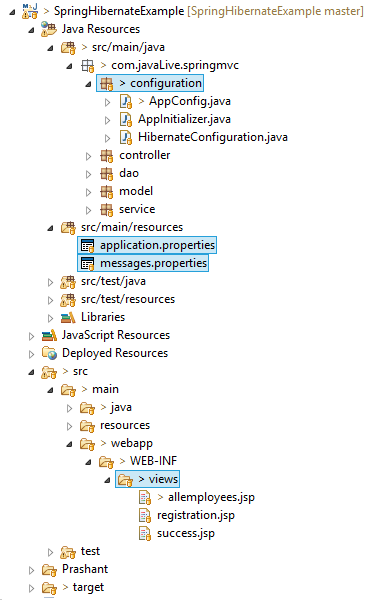
'SpringMVCHibernateWithXMLFull' : explains full Spring+Hibernate database table CRUD operation **with full XML configuration** which includes all three layers of Spring MVC viz. Controller, Service and DAO.

' SpringHibernateExample' : explains full Spring+Hibernate database table CRUD operation **with full Java configuration i.e. annotation based** which includes all three layers of Spring MVC viz. Controller, Service and DAO.

This project explains functioning of Spring MVC and Hibernate collaboration. How controller acts and handle request, makes call to service layer which contains business logic, collects the result of business logic from service layer and render the specified jsp accordingly.

This is web based maven project.

**Note that this is pure Java based spring MVC project(making use of annotations). Note that we do not have web config file web.xml and spring configuration file spring-servlet.xml. Instead we have Java files replacing these file which uses annotations. These files are stored in package com.javaLive.springmvc/configuration Please refer below screen shot for more detail:**

****

**In above project please go through files, packages and folders as follows:**

Most important part of this project is package com.javaLive.springmvc.configuraion.

This package contains classes which replaces .xml files (web.xml and spring-servlet.xml) those are in previous project viz. ' SpringMVCHibernateWithXMLFull'. Now we will study the package contents along with the difference with its .xml counterpart.

#### 1. Configure Initializer class

com.javaLive.springmvc.configuraion.AppInitializer.java:

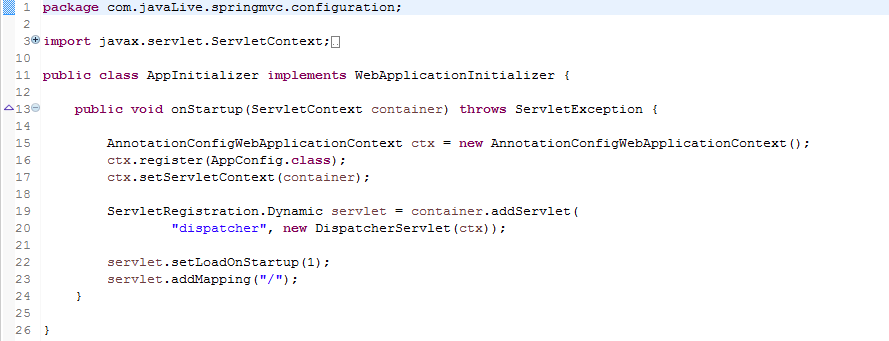
**This file is Java version of web.xml file which provides same functionality as that of web.xml by using annotations.**

**Screen shots of both these file are given below.**

web.xml file from project ' SpringMVCHibernateWithXMLFull' :

****

AppInitializer.java from project ' SpringHibernateExample'

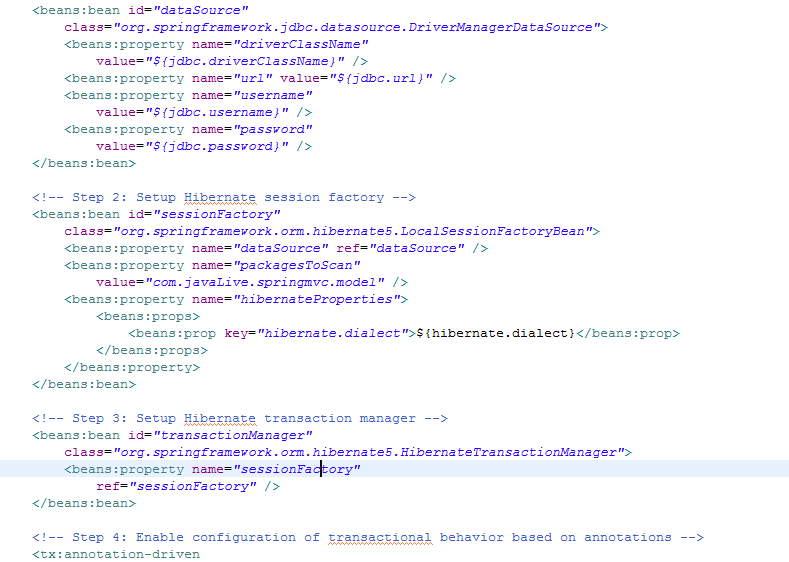


\*\*\*The content above resembles the content of web.xml as we are using the front-controller DispatherServler, assigning the mapping (url-pattern in xml) and instead of providing the path to spring configuration file(spring-servlet.xml) , here we are registering the Configuration Class AppConfig.java which is Java counterpart of spring-servlet.xml.

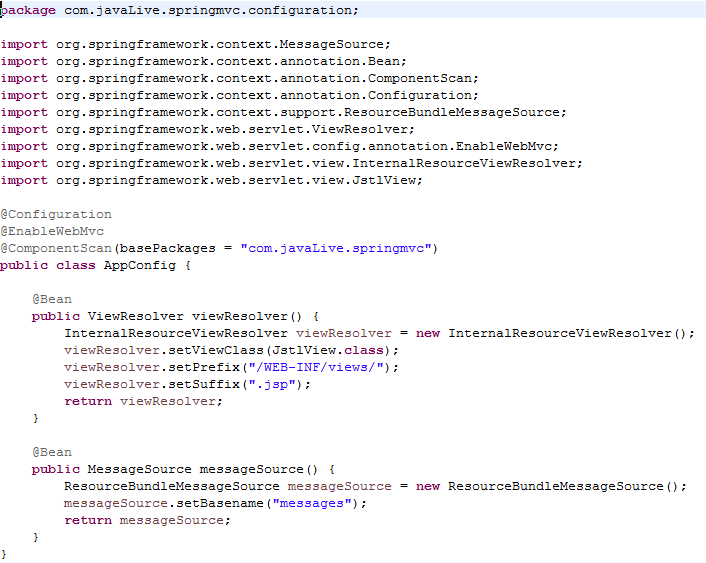
#### 2. Configure Spring MVC

com.javaLive.springmvc.configuraion.AppConfig.java:

This file is Java counterpart of spring-servlet.xml file as that of project 'SpringMVCHibernateWithXMLFull'. Below are the screen shots of both of them:

File spring-servlet from project ' SpringMVCHibernateWithXMLFull' 

AppConfig.java file from project ' SpringHibernateExample'

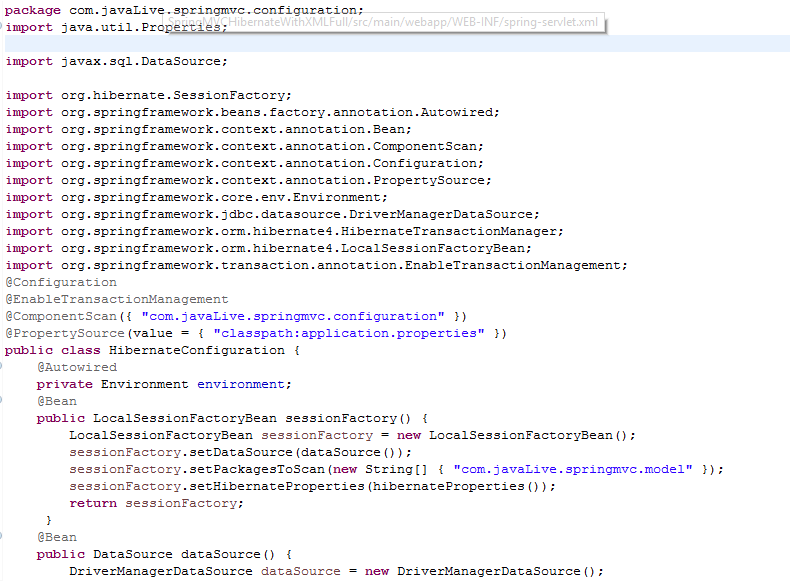


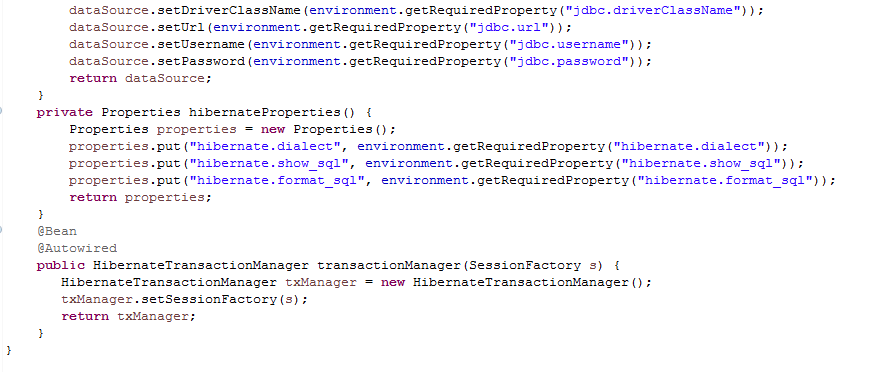
@Configuration marks this class as configuration class as mentioned above & ComponentScan referes to package locations to find the associated beans.  
@EnableWebMvc is equivalent to mvc:annotation-driven in XML.  
Method viewResolver configures a view resolver to identify the real view.

#### 3.  Configure Hibernate

Note that here we have different configuration file for Hibernate viz. com.javaLive.springmvc.configuration. HibernateConfiguration.java unlike spring-servlet.xml file in project ' SpringMVCHibernateWithXMLFull'.

Below is the screen shot for this file:

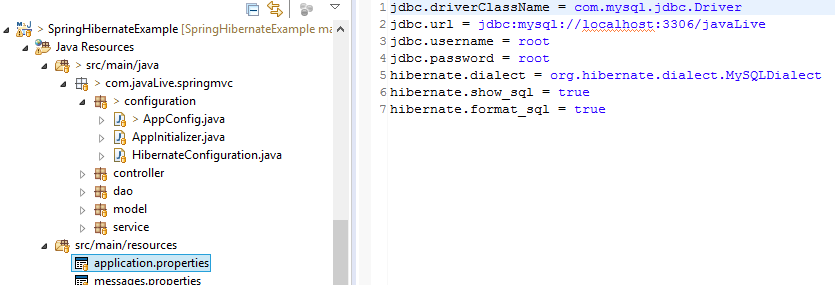




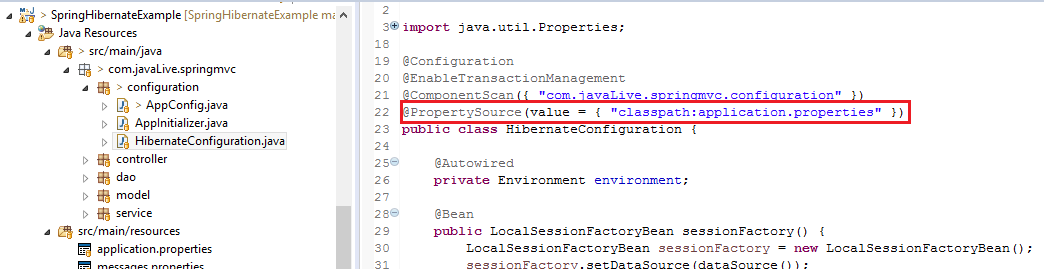
@Configuration indicates that this class contains one or more bean methods annotated with @Bean producing beans manageable by spring container. In our case, this class represent hibernate configuration.  
@ComponentScan is equivalent to context:component-scan base-package="..." in xml, providing with where to look for spring managed beans/classes.  
@EnableTransactionManagement is equivalent to Spring’s tx:\* XML namespace, enabling Spring’s annotation-driven transaction management capability.  
@PropertySource is used to declare a set of properties(defined in a properties file in application classpath) in Spring run-time Environment, providing flexibility to have different values in different application environments.

Method sessionFactory() is creating a LocalSessionFactoryBean, which exactly mirrors the XML based configuration : We need a dataSource and hibernate properties (same as hibernate.properties). Thanks to @PropertySource, we can externalize the real values in a .properties file, and use Spring’s Environment to fetch the value corresponding to an item. Once the SessionFactory is created, it will be injected into Bean method transactionManager which may eventually provide transaction support for the sessions created by this sessionFactory.

application.properties which contains database related properties as below:

****

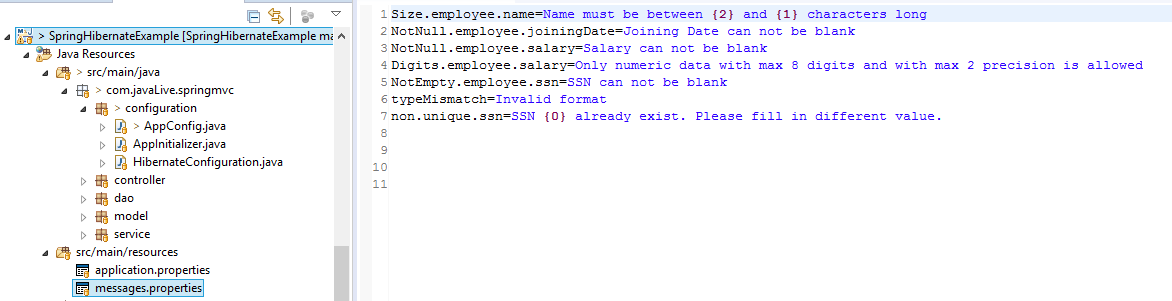
We have mentioned path of above file in spring-ervlet.xml file as below:



**4. Validation Mechanism :**

In this post, we are working with form submission, validating user input (via JSR303 annotations). In case of validation failure, default error messages are shown.To override those default by your own custom [internationalized] messages from an external Message bundle [.properties file], we need to configure a ResourceBundleMessageSource. Method messageSource is there for same purpose. Notice the parameter provided (messages) to basename method. Spring will search for a file named messages.properties in application class path. Let’s add that file:

/src/main/resources/messages.properties



Notice that above message follows a specific pattern

{ValidationAnnotationClass}.{modelObject}.{fieldName}

Additionally, based on specific annotation (e.g. @Size) you can also pass the arguments to these messages using {0},{1},..{i} indexes.

**Other packages of the project are**

Packages com.javaLive.springmvc.controller which contains controllers.

Package com.javaLive.springmvc.service: which contains service classes.

Package com.javaLive.springmvc.dao: which contains DAO classes.

Package com.javaLive.springmvc.model: which contains Entity classes.

Folder WEB-INF/views which contains various JSP files.

**Steps to create project:-**

1. This is web based maven project. Create webapp maven project. Please refer the file **'CreateWebBasedMavenProjectInEclipse.docx'** in this project to know more about creation of web based maven project.

2. Add require dependencies for spring as shown in pom.xml file.

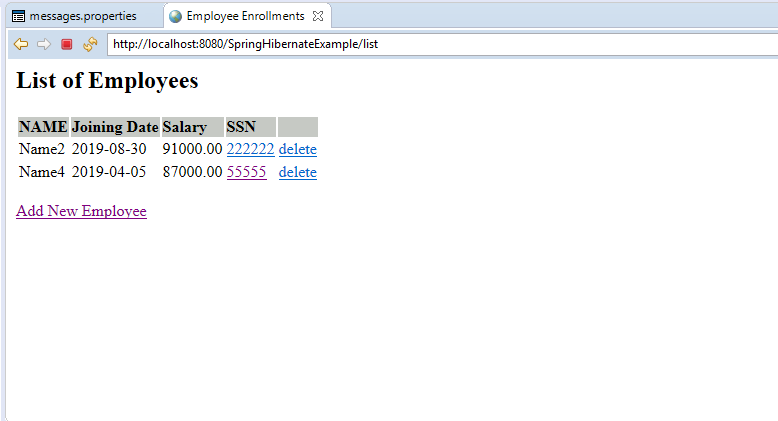
3. Create required packages and add the files.

**Functioning of the project : As explained earlier, this project deals with CRUD operation on a database table with Spring and Hibernate.**

**Please note that refer file '** **SpringMVCHibernateWithXMLFullProjectGuide.docx' in project '** **SpringMVCHibernateWithXMLFull ' to know more about how actually these operations takes place behind the scene. Here we have provide only overview of the functioning. As main reason behind this project is to know the configuration level difference between XML based Spring Hibernate application and Java(annotation) based Spring Hibernate application. Rest of the functioning of the this app is almost same as that of 'SpringMVCHibernateWithXMLFull' app. This project has only one more extra functionality viz. form validation.**

**Launching the app:**

**Step1 : Right click on project ->Run As -> Run on Server**



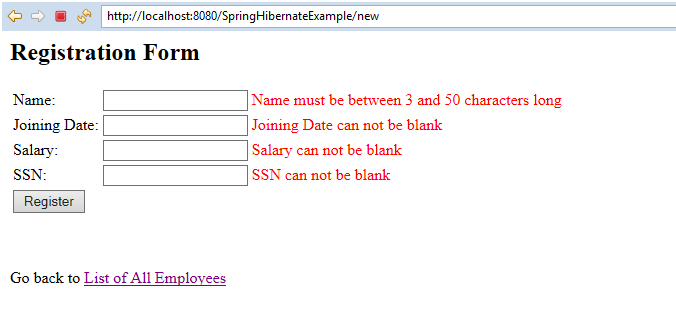
As we can see it fetches and shows the Employee related data to JSP page.

Thus performing **R**ead operation of the database.

**Step 2 : Now when we click link 'Add New Employee' it leads to rendering following page:**



If you click 'Register' button without entering any values, you will get following errors

 This is result of Spring form validation mechanism. It is combined effect of following files:

1. src/main/resources/messages.properties: this file contents text messages.

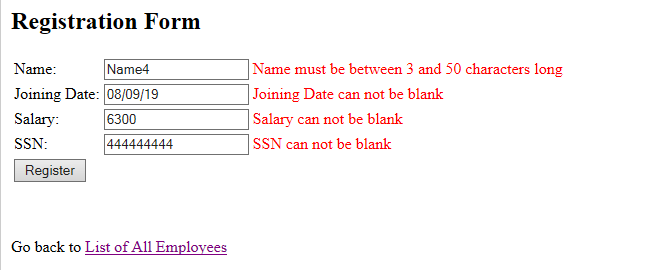
2. Bean MessageSourcein spring configuration file viz. ' com.javaLive.springmvc.configuration.AppConfig.java':

In Spring, you can use ResourceBundleMessageSource to resolve text messages from properties file, base on the selected locales.

3. Entity file ' com.javaLive.springmvc.model.Employee.java':

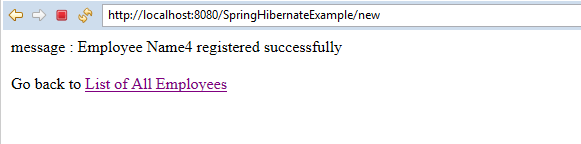
This file contents class variables with ' javax.validation.constraints.\* ' and

' org.hibernate.validator.constraints.\* ' which triggers validation process.

Now fill the required values and click 'Register' button 

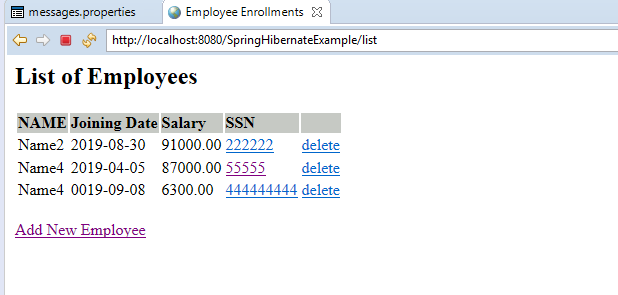
It displays success message for given added record as follows:

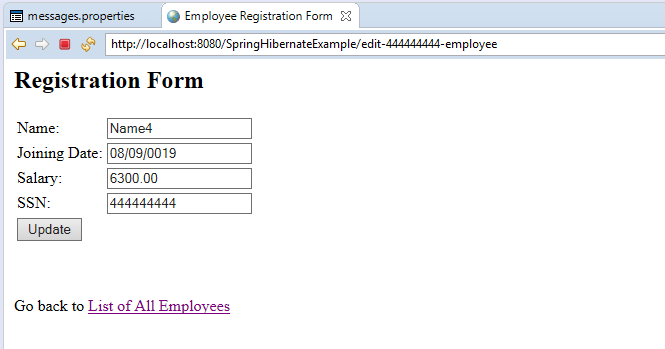
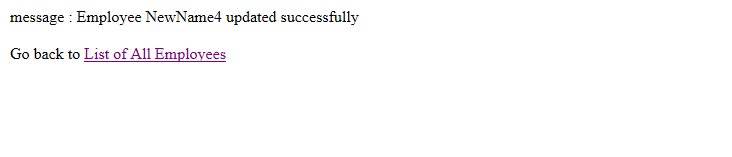
In this way it carries out ' **C**reate ' operation for database table record.



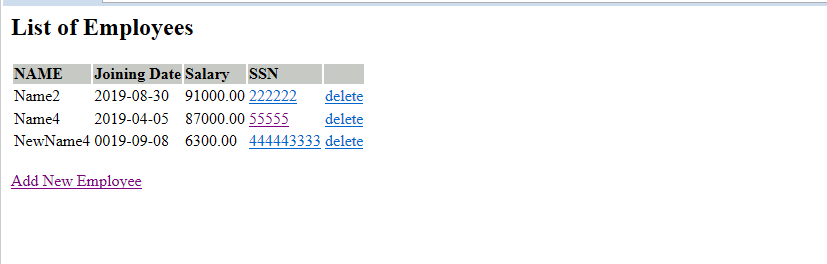
Now go back to Employee list page by clicking 'List of All Employees' link

**Step 3: Now click last record in SSN column. This leads to Update operation for given record.**



Make some changes in the record and click 'Update' button. This leads to displaying of following success message.

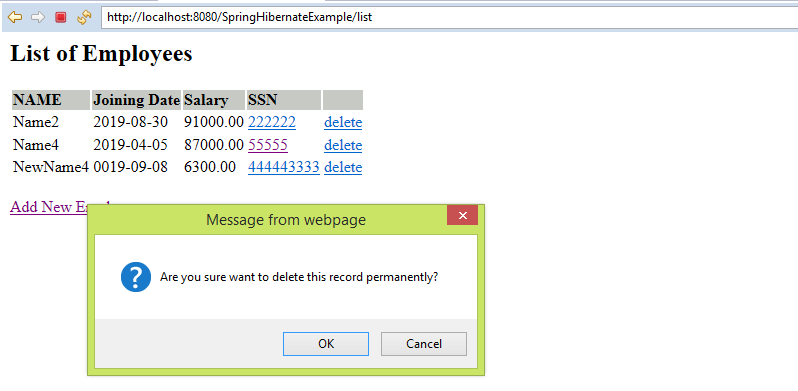
In this way it carries out ' **U**pdate ' operation for database table record.



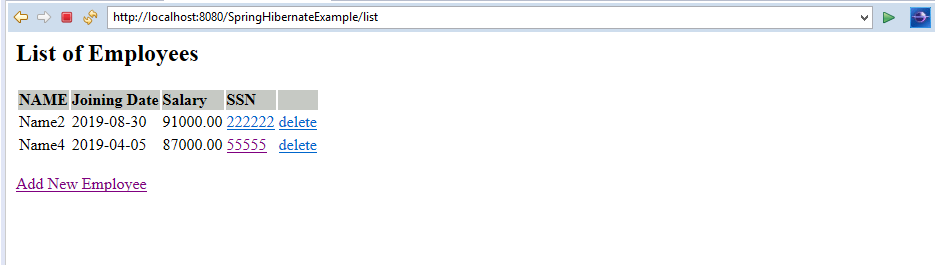
**Step 4 : Delete operation for given record.**

Select our newly created record in following list and click respective delete link.

Confirmation dialogue box pop up. Click 'OK'.

 Note the selected record get deleted successfully.

In this way it carries out ' **D**elete ' operation for database table record.



Now next project is extension of this project only with additional Testing code for various layers of Spring MVC Hibernate application.