H2 Database connectivity using Hibernate and Spring MVC

1. Configuration of H2 database
   1. The executable jar file inside the bin folder of the setup has to be run,it'll open the brower .
   2. In it we have to select h2(server) in place of h2(embedded).
   3. Then in place of test as database name , i've entered Sports.
   4. Userid was sa, and password was specified as sports.
   5. Then connect,it opens the database .
2. Adding Dependencies for implementing Spring Hibernate ORM with H2 Database
   1. spring-orm with spring jdbc

https://mvnrepository.com/artifact/org.springframework/spring-orm/5.0.0.M5

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.3.6.RELEASE</version>

</dependency>

* 1. Driver h2

https://mvnrepository.com/artifact/com.h2database/h2

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.191</version>

</dependency>

* 1. hibernate-core--version 5 and above

<dependency>

   <groupId>org.hibernate</groupId>

   <artifactId>hibernate-core</artifactId>

   <version>5.2.5.Final</version>

  </dependency>

* 1. commons-bdcp2(database connection pooling)

https://commons.apache.org/proper/commons-dbcp/

<dependency>

    <groupId>org.apache.commons</groupId>

    <artifactId>commons-dbcp2</artifactId>

    <version>2.1.1</version>

</dependency>

1. Adding JPA code in model classes
   1. Open existing Product.java file and add the following as below:

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

@Entity

**public** **class** Product {

@Id

@GeneratedValue(strategy=GenerationType.***IDENTITY***)//for autonumber

**int** productId;

String productName;

String productDescription;

**int** productPrice;

**int** productQuantity;

String imageUrl;

1. Create applicationContext.xml file inside WEB-INF folder.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx-4.0.xsd"* >

</beans>

1. Load applicationContext.xml in project inside web.xml helping contextLoaderListener class

<!-- Loading applicationContext.xml -->

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

1. Entire configuration required to be placed inside the applicationContext.xml from dispatcher-servlet.xml file. Dispatcher-servlet only contain view resolver configuration code.
2. Add 3 beans inside the applicationContext.xml file.

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd"*>

<!-- Load all components -->

<context:component-scan base-package=*"com.health.HealthMedicineQuest"* />

<!-- DataSource -->

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp2.BasicDataSource"*>

<property name=*"driverClassName"* value=*"org.h2.Driver"* />

<property name=*"url"* value=*" jdbc:h2:tcp://localhost/~/Health2"* />

<property name=*"username"* value=*"sa2"* />

<property name=*"password"* value=*""* />

</bean>

<!-- SessionFactory -->

<bean id=*"sessionFactory"*

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"packagesToScan"*>

<list>

<value>com.health.HealthMedicineQuest.model" </value>

</list>

</property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.H2Dialect</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.format\_sql"*>true</prop>

</props>

</property>

</bean>

<!-- Transaction Management -->

<tx:annotation-driven transaction-manager=*"txManager"* />

<bean id=*"txManager"*

class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"* />

</bean>

</beans>

1. Change hibernate specific code into ProductDAOImpl file.

**import** org.hibernate.SessionFactory;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Repository;

**import** org.springframework.transaction.annotation.Propagation;

**import** org.springframework.transaction.annotation.Transactional;

**import** com.health.HealthMedicineQuest.model.Product;

@Repository("productDAO")

@Transactional(propagation=Propagation.***SUPPORTS***, readOnly=**false**)

**public** **class** ProductDAOImpl **implements** IProductDAO {

@Autowired

SessionFactory sessionFactory;

**public** List<Product> getAllProducts() {

**return** sessionFactory.openSession().createQuery("from Product", Product.**class**).getResultList();

}

}

1. Run the project, which will create table inside the h2 Database. Insert any record and open the product detail page to view the output.
2. Retrieving single product by clicking view product button:
   1. Open IProductDAO interface and uncomment getProduct().
   2. Open ProductDAOImpl class add following code.

**public** Product getProduct(**int** id){

List<Product> prod =**new** ArrayList<Product>();

prod=getAllProducts();

**return** prod.get(id);

}

* 1. Open ProductController class and add following code:

@RequestMapping(value={"/product"})

**public** ModelAndView product(@PathVariable("id")**int** id){

ModelAndView model=**new** ModelAndView("product");

model.addObject("list", productDAO.getProduct(id));

**return** model;

}

* 1. Open ProductDetail.jsp page and add following code after table tag for creating single product detail as modal page:

[%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %](mailto:%25@taglib%20prefix=%22c%22%20uri=%22http://java.sun.com/jsp/jstl/core%22%20%25)

<c:forEach var=*"p"* items=*"*${list}*"*>

<div class=*"modal fade"* id=*"*${p.productId}*"* tabindex=*"-1"* role=*"dialog"* aria-labelledby=*"myModalLabel"*>

<div class=*"modal-dialog"* role=*"document"*>

<div class=*"modal-content"*>

<div class=*"modal-header"* >

<button type=*"button"* class=*"close"* data-dismiss=*"modal"* aria-label=*"Close"*><span aria-hidden=*"true"*>&times;</span></button>

<h2 class=*"modal-title"* id=*"myModalLabel"*>Product Details</h2>

</div>

<div class=*"modal-body"*>

<div class=*"col-sm-12"* id=*"letv"*>

<div class=*"thumbnail"*>

<img src=*"*${images}*/*${p.imageUrl}*.jpg"* alt=*""* >

<div class=*"caption"*>

<h2>${p.productName}</h2>

<h2> ${p.productDescription}</h2>

<p>${p.productPrice} </p>

<p><a href=*"#"* class=*"btn btn-info btn-xs"* role=*"button"*>close</a></p>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</c:forEach>