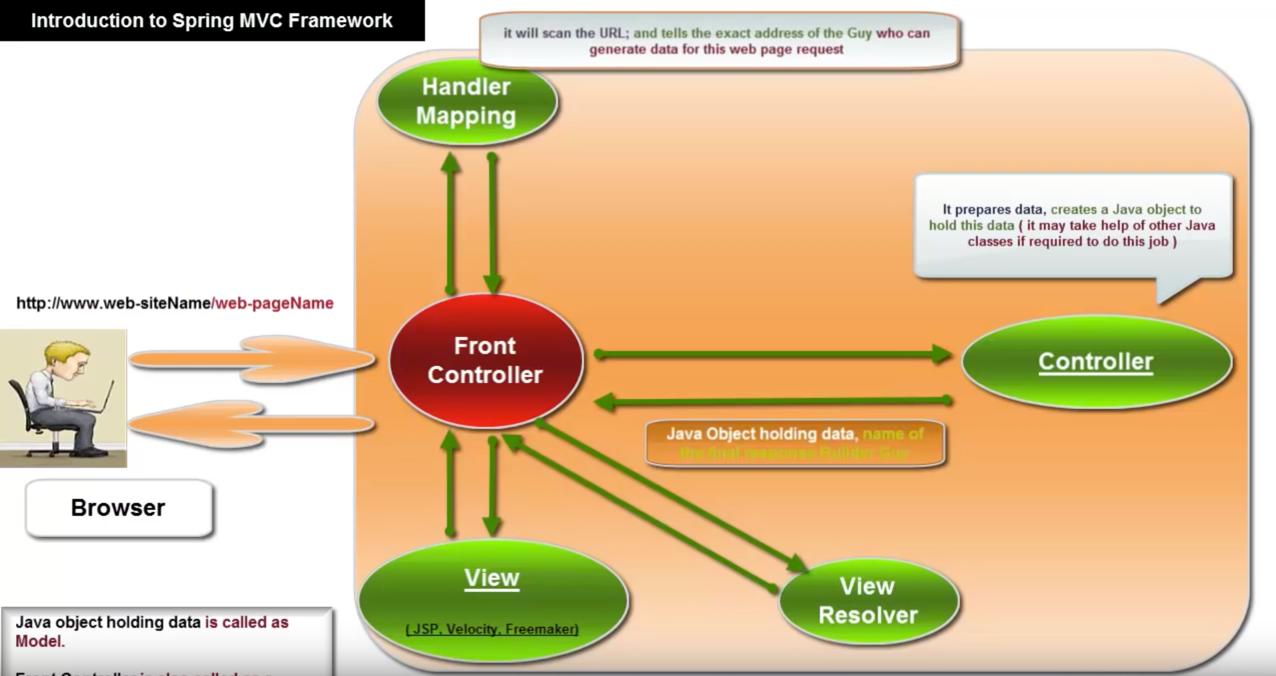
Question 3

3. A web page has a simple form with one text field and a submit button. Entering a value in the text field and clicking Submit causes the text field to be stored in a relational database and a result message to be displayed to the user. Describe in as much detail as you can what happens from the time the Submit button is pressed until a result is displayed.

· Assume a Spring/Hibernate based web application is running on the server.

I have put together an example that has the following directory structure. This application was created using the Spring Source Toolkit (STS) 3.7.2. I created the application using the File->New->Spring Legacy Project->Spring MVC Project. The project will be attached as a zip file in the email. Please look for it.

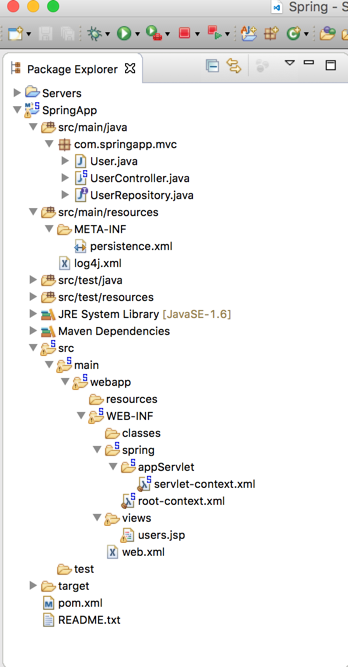
The overall process flow of an MVC is pictured below:



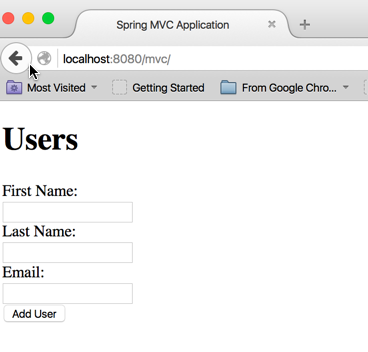
This creates a project based on the MVC design pattern. There is no definitive definition of what makes up an MVC, but it does have some basic principles to follow. The [V]iew has the responsibility of presentation to the user or the UI. The [C]ontroller has the responsibility of the logic flow of the application, typically the business logic. The [M]odel contains the data.

This example is taken from <http://scottsdigitalcommunity.blogspot.com/2013/05/developing-spring-mvc-project-using.html> and modified to work within STS as described above.

The structure of the application looks like the picture below:



The web page for the final application looks like the picture below:



I realize that this has 3 text fields, but it is the same behavior.

The form that produces the web page is the users.jsp file:

<!doctype html>

<%@taglib uri=*"http://www.springframework.org/tags"* prefix=*"spring"*%>

<%@taglib uri=*"http://www.springframework.org/tags/form"* prefix=*"form"*%>

<%@taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"*%>

<html>

<head>

<meta charset=*"utf-8"*>

<title>Spring MVC Application</title>

<meta content=*"IE=edge,chrome=1"* http-equiv=*"X-UA-Compatible"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<link href=*"http://twitter.github.io/bootstrap/assets/css/bootstrap.css"*

rel=*"stylesheet"*>

<link

href=*"http://twitter.github.io/bootstrap/assets/css/bootstrap-responsive.css"*

rel=*"stylesheet"*>

</head>

<body>

<div class=*"container"*>

<div class=*"row"*>

<div class=*"span8 offset2"*>

<h1>Users</h1>

<form:form method=*"post"* action=*"add"* commandName=*"user"*

class=*"form-horizontal"*>

<div class=*"control-group"*>

<form:label cssClass=*"control-label"* path=*"firstName"*>First Name:</form:label>

<div class=*"controls"*>

<form:input path=*"firstName"* />

</div>

</div>

<div class=*"control-group"*>

<form:label cssClass=*"control-label"* path=*"lastName"*>Last Name:</form:label>

<div class=*"controls"*>

<form:input path=*"lastName"* />

</div>

</div>

<div class=*"control-group"*>

<form:label cssClass=*"control-label"* path=*"email"*>Email:</form:label>

<div class=*"controls"*>

<form:input path=*"email"* />

</div>

</div>

<div class=*"control-group"*>

<div class=*"controls"*>

<input type=*"submit"* value=*"Add User"* class=*"btn"* />

</form:form>

</div>

</div>

<c:if test=*"*${!**empty** users}*"*>

<h3>Users</h3>

<table class=*"table table-bordered table-striped"*>

<thead>

<tr>

<th>Name</th>

<th>Email</th>

<th>&nbsp;</th>

</tr>

</thead>

<tbody>

<c:forEach items=*"*${users}*"* var=*"user"*>

<tr>

<td>${user.lastName}, ${user.firstName}</td>

<td>${user.email}</td>

<td>

<form action=*"delete/*${user.id}*"* method=*"post"*>

<input type=*"submit"* class=*"btn btn-danger btn-mini"*

value=*"Delete"* />

</form>

</td>

</tr>

</c:forEach>

</tbody>

</table>

</c:if>

</div>

</div>

</div>

</body>

</html>

You will note that the file uses the jstl tag library and the bootstrap stylesheet.

The page consists of a form denoted by the <form:form method=*"post"* action=*"add"* commandName=*"user"* and the a table at the end of the page for handling the response to each submit operation by the user.

The users.jsp file represents the “view” portion of the MVC architecture.

The web.xml page consists of the following:

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

<web-app version=*"2.5"* xmlns=*"http://java.sun.com/xml/ns/javaee"*

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

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/root-context.xml</param-value>

</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- Processes application requests -->

<servlet>

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>appServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

The highlighted section in red is where spring tells the front controller where to dispatch the user request.

All spring MVC applications use servlets as their underlying technology. In order for spring to figure out where to route the request, the front controller uses the web.xml file to tell it what to use as the front controller. In this case, the front controller is the “org.springframework.web.servlet.DispatcherServlet”, which is initialized by the contexts of the /WEB-INF/spring/appServlet/servlet-context.xml. The DispatcherServlet is the front controller in the picture above.

The servlet-context.xml file contains the following:

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

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

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

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

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd*

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

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

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

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

<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

<!-- Enables the Spring MVC @Controller programming model -->

<annotation-driven />

<!-- Handles HTTP GET requests for /resources/\*\* by efficiently serving up static resources in the ${webappRoot}/resources directory -->

<resources mapping=*"/resources/\*\*"* location=*"/resources/"* />

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->

<beans:bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<beans:property name=*"prefix"* value=*"/WEB-INF/views/"* />

<beans:property name=*"suffix"* value=*".jsp"* />

</beans:bean>

<context:component-scan base-package=*"com.springapp.mvc"* />

<!-- Adding to default

Added properties to beans:beans

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

xmlns:jpa="http://www.springframework.org/schema/data/jpa"

Added to beans:beans property xsi:schemaLocation

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

http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa.xsd

-->

<jpa:repositories base-package=*"com.springapp.mvc"*/>

<beans:bean id=*"entityManagerFactory"* class=*"org.springframework.orm.jpa.LocalEntityManagerFactoryBean"*>

<beans:property name=*"persistenceUnitName"* value=*"defaultPersistenceUnit"*/>

</beans:bean>

<beans:bean id=*"transactionManager"* class=*"org.springframework.orm.jpa.JpaTransactionManager"*>

<beans:property name=*"entityManagerFactory"* ref=*"entityManagerFactory"* />

</beans:bean>

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

<!-- Done adding to default -->

</beans:beans>

The red highlighted text tells spring which view resolver to use. In this case it is “org.springframework.web.servlet.view.InternalResourceViewResolver”. This is the View Resolver indicated in the picture above.

The bottom half of the file sets up the transaction management for persisting data to the back end.

The [C]ontroller, as indicated in the picture above is defined in the UserController.java file as follows:

package com.springapp.mvc;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.validation.BindingResult;

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;

@Controller

public class UserController {

@Autowired

private UserRepository userRepository;

@RequestMapping(value = "/", method = RequestMethod.GET)

public String listUsers(ModelMap model) {

model.addAttribute("user", new User());

model.addAttribute("users", userRepository.findAll());

return "users";

}

@RequestMapping(value = "/add", method = RequestMethod.POST)

public String addUser(@ModelAttribute("user") User user,

BindingResult result) {

userRepository.save(user);

return "redirect:/";

}

@RequestMapping("/delete/{userId}")

public String deleteUser(@PathVariable("userId") Long userId) {

userRepository.delete(userRepository.findOne(userId));

return "redirect:/";

}

}

The model consists of two files: User.java and UserRepository.java as follows:

User.java

**package** com.springapp.mvc;

**import** javax.persistence.\*;

@Entity(name = "account")

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

**private** Long id;

@Basic

**private** String firstName;

@Basic

**private** String lastName;

@Basic

**private** String email;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String name) {

**this**.firstName = name;

}

**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;

}

}

UserRepository.java

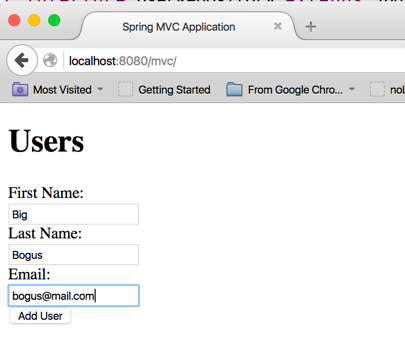
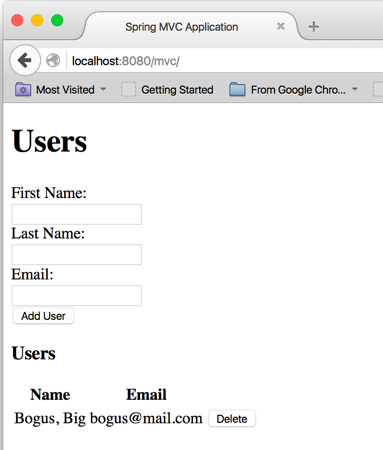
**package** com.springapp.mvc;

**import** org.springframework.data.jpa.repository.JpaRepository;

**public** **interface** UserRepository **extends** JpaRepository<User, Long> {

}

In summary, the process flow of the app is as follows:

1. The application starts up and UserController() gets invoked by the front end controller and is routed to the “/” which is mapped to the listUsers() method as indicated by the @RequestMapping annotation. The method creates a new User() with nothing in it and adds it to the “model” to mixin to the users.jsp page. It also does a query for all the users in the current database using the userRepository.findAll() method and adds this list of the “users” attribute in the model to be mixed in with the users.jsp page. The UserController.listUsers() then returns “users” which goes back to the front end controller which then routes to the users.jsp page. The users.jsp is a servlet that the dispatcher/front controller knows about. The user is then presented with the page in the picture above.
2. Now the user enters the data in the text fields as indicated here:
3. 
4. Next the user clicks on the “Add User”. The form has a method of “post”, and action of “add”, and a commandName of “user”. This pattern gets resolved by the front controller to be dispatched to the UserController to the addUser() method. This method has a @ModelAttribute of “user” that represents the model User from the form in the users.jsp page. The UserController.addUser() method calls the userRepository.save(user) method to persist the new user to the backend database. The UserController.addUser() method then returns “redirect:/” to the front controller which then routes another request to the UserController.listUsers() method which then routes the request to the “users” which is the users.jsp page. The following page is displayed:
5. 
6. You will note that the “Users” html table is displayed. This is because the UserController.listUsers() method populated the model for the “users” attribute using the userRepository.findAll() method. The findAll() method is part of the “org.springframework.data.jpa.repository.JpaRepository” class in spring. Since the UserRepository is inherited from the JpaRepository, spring injects this functionality into the application.