



Spring MVC 3.x

Krishantha Dinesh
[kdinesh@virtusa.com]

@2013 Nov

2000 West Park Drive
Westborough MA 01581 USA
Phone: 508 389 7300 Fax: 508 366 9901

The entire contents of this document are subject to copyright with all rights reserved. All copyrightable text and graphics, the selection, arrangement and presentation of all information and the overall design of the document are the sole and exclusive property of Virtusa.

Copyright © 2012 Virtusa Corporation. All rights reserved

Objectives

- Not to teach entire spring mvc framework
- Get a idea and concept about spring MVC

Prerequisites

- `Java.basicknowledge >(60/100);`
- `Mobilephone.soundmode=soundmode.completesilent && soundmode.completesilent != soundmode.vibrate`

What are the similar frameworks

- Struts

used to be the standard, Struts 1. It was eventually very outdated. Struts 2 was a significant change all for the better than the original Struts but different enough that a lot of people has lost their loyalty for Struts,

- Tapestry

still is a very decent framework but is governed by one person and someone at the WIM or whatever project he was working on. Tapestry was one of the first heavily object-oriented or POJO-based web frameworks.

- Wicket

is a lightweight framework that sells itself on being easier to configure than some of the other frameworks that are out there.

- GWT

Google Web Toolkit is a very rich user experience framework with a steeper learning curve than a lot of the other frameworks that are out there.

- JSF

Actually is the only true standard framework for the web out there is, J2EE based as well and it's a very rich user experience. It comes with libraries like ice faces and richfaces and other third-party components for integrating into JSF.

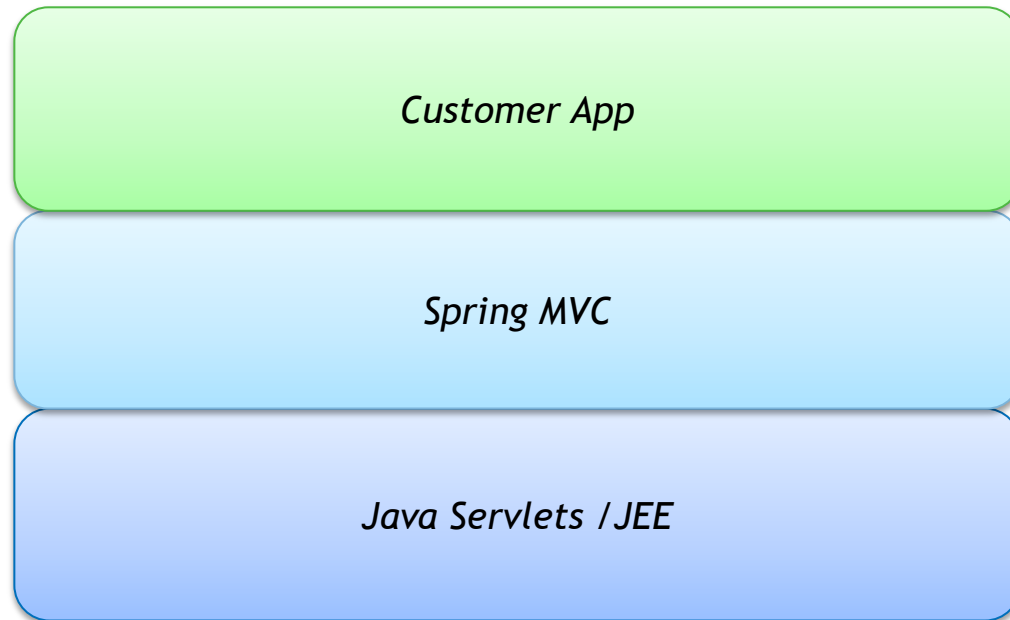
- Spring MVC.

Spring MVC is a very unobtrusive framework. It is lightweight in a sense that there's very little overhead in running it but it's a very capable, heavy duty framework. It pulls a lot of the best practices from all of these frameworks we just mentioned and combines it into Spring MVC. Plus, it just makes sense that Spring MVC integrates very nicely with Spring and all the other capabilities that Spring has available for us to use.

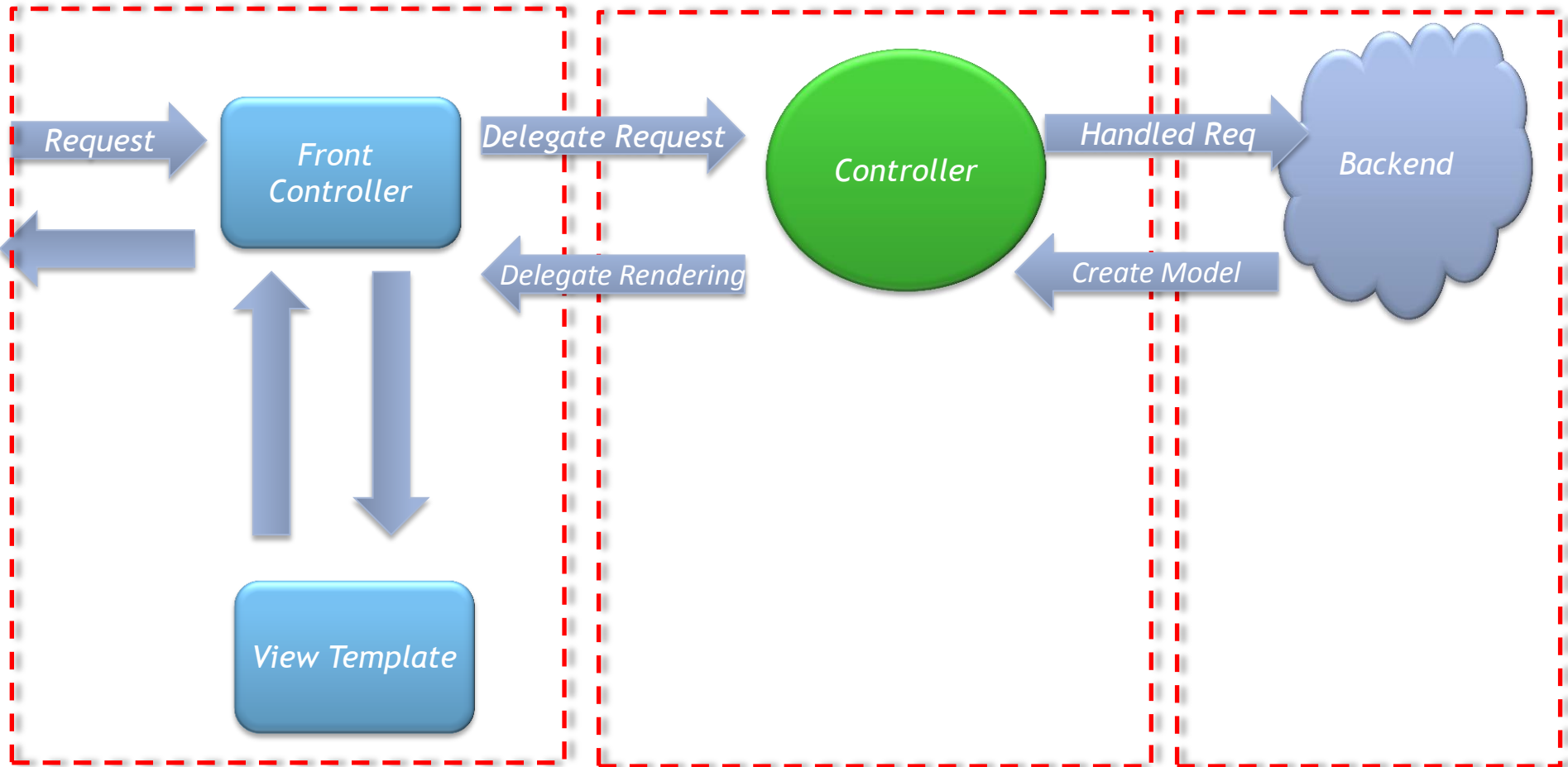
What is spring

- Its not limited to specific group
- It may be a RESTfull application , jsp , freemarker, velocity or etc
- Web framework developed based on principles of spring
- POJO based, unit testable, interface driven
- Very light weight
- Based on dispatcher servlet/front controller pattern
- Build from short comings from struts 1
- Support for Themes, Locales , REST services and annotation based configuration

Architecture



Request Response life cycle



Engineers talks

- Dispatcher servlets – entry point for the servlet [front controller]
- Controller – router , command pattern object that handle the request
- Request mapping – URL and request type that method tied to
- View resolver – to locate view
- Servlet config – Configuration file per dispatcher servlet

Requirement [prerequisites]

- Java 7 or 5 later
- Maven 3.0 +
- Spring STS or eclipse
- Apache tomcat server

What we going to build

- Sales tracker application with following features
 1. Allow enter the sales value
 2. Show total sales
 3. Set sales target for day
 4. Notify when goal reach

Download spring without Maven

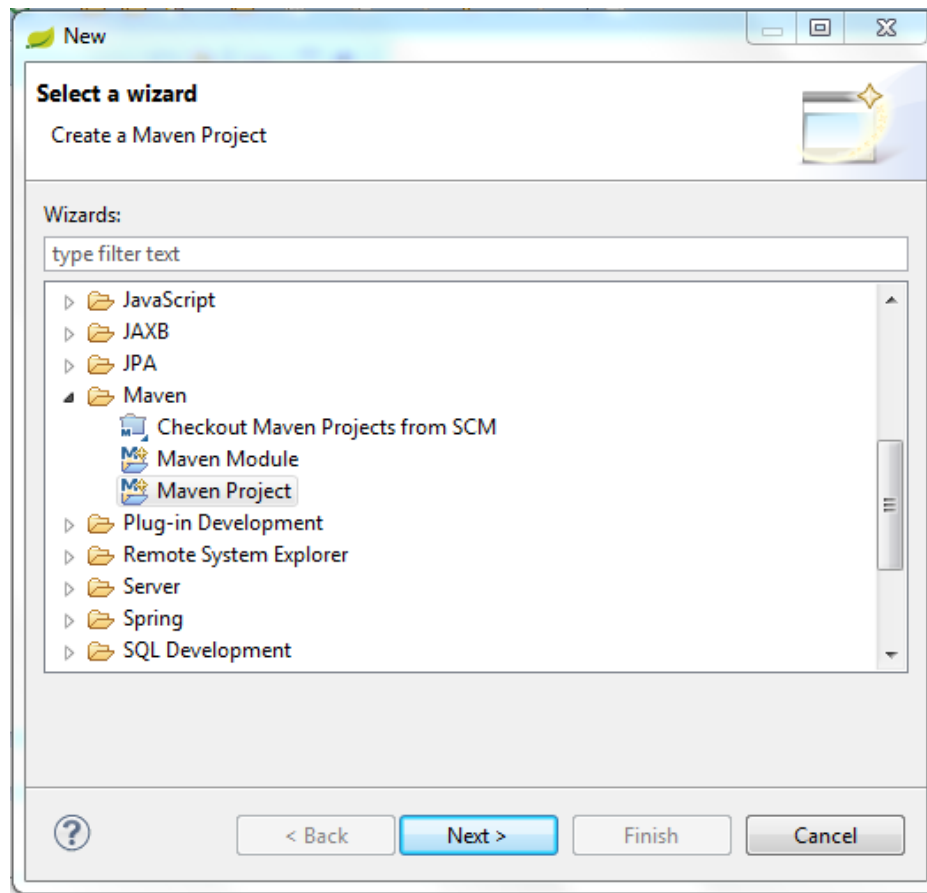
- Download spring framework. This presentation use the version 3.2.4 which is current version.
- <http://repo.springsource.org/libs-release-local/org/springframework/spring/3.2.4.RELEASE/spring-framework-3.2.4.RELEASE-dist.zip>
- GTO link
- http://ct-corpsearchdb/technology/Lists/Software%20Master%20List/DispForm.aspx?ID=73&Source=http%3A%2F%2Fct-corpsearchdb%2Ftechnology%2FLists%2FSoftware%2520Master%2520List%2FAll.aspx%3FPaged%3DTRUE%26p_Title%3DMicro%2520Cloud%2520Foundry%26p_ID%3D701%26View%3D%257bD03C0240%252dBE81%252d41F7%252d9260%252d2E09B38E89F6%257d%26PageFirstRow%3D201

Download with maven

- Only following 3 dependency needed
- Spring-webmvc
- Servlet-api
- jstl

Demo

- Open spring sts and close all existing projects
- File→New→other→Maven project and click next



Click Next

New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location:

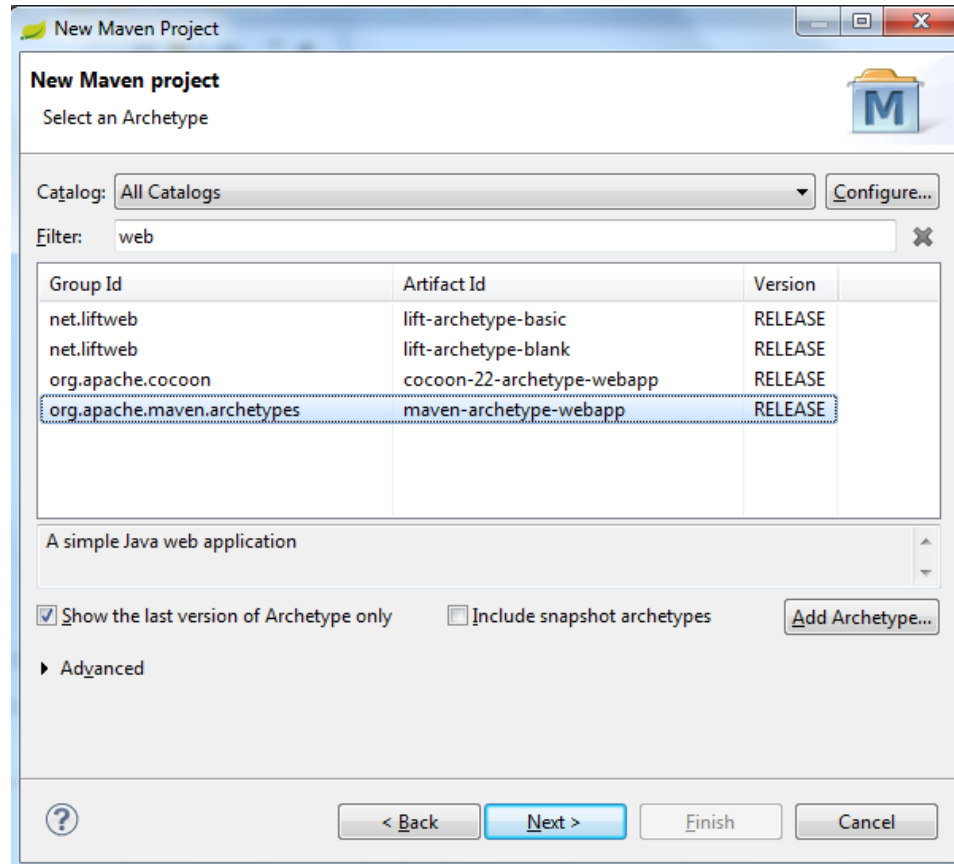
☐ Add project(s) to working set

Working set:

► Advanced

archetype

- Select maven-archetype-webapp



Fill the required detail

- Groupid – company
- Artifactid – application name

New Maven Project
Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

Properties available from archetype:













Name	Value

▶ Advanced

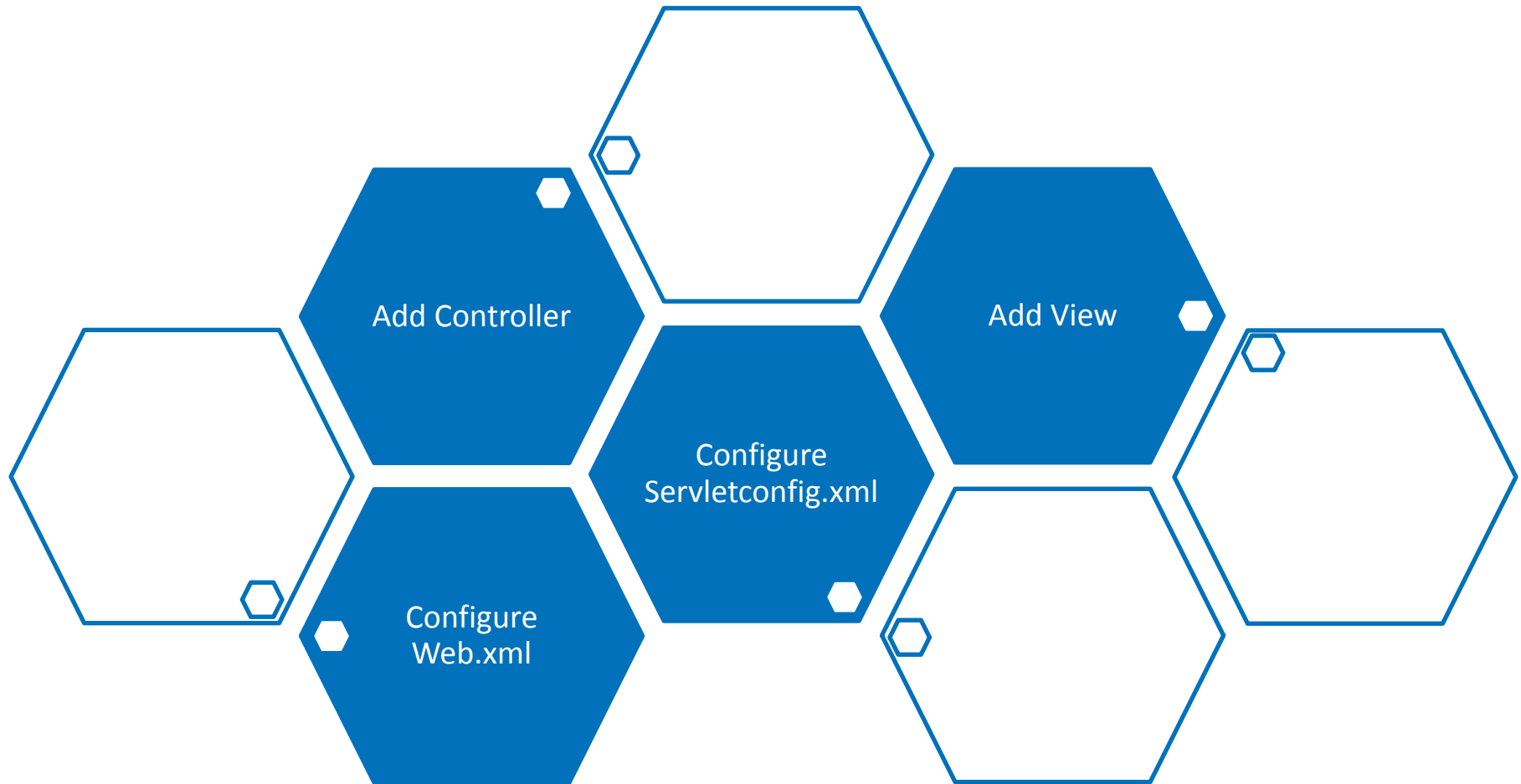
Add required dependencies

```
⊖ <dependencies>
⊖   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>3.2.0.RELEASE</version>
    </dependency>
⊖   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
⊖   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
⊖ <build>
      <finalName>SalesTracker</finalName>
    </build>
```

📁 Maven Dependencies

- ▷  junit-3.8.1.jar - C:\Users\kdinesh\.m2\repository\junit\junit\3.8.1
- ▷  spring-webmvc-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-webmvc\3.2.0.RELEASE
- ▷  spring-context-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-context\3.2.0.RELEASE
- ▷  spring-aop-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-aop\3.2.0.RELEASE
- ▷  spring-core-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-core\3.2.0.RELEASE
- ▷  commons-logging-1.1.1.jar - C:\Users\kdinesh\.m2\repository\commons-logging\commons-logging\1.1.1
- ▷  spring-web-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-web\3.2.0.RELEASE
- ▷  aopalliance-1.0.jar - C:\Users\kdinesh\.m2\repository\org.aopalliance\aopalliance\1.0
- ▷  spring-expression-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-expression\3.2.0.RELEASE
- ▷  spring-beans-3.2.0.RELEASE.jar - C:\Users\kdinesh\.m2\repository\org.springframework\spring-beans\3.2.0.RELEASE
- ▷  servlet-api-2.5.jar - C:\Users\kdinesh\.m2\repository\javax.servlet\javax.servlet-api\2.5
- ▷  jstl-1.2.jar - C:\Users\kdinesh\.m2\repository\javax.servlet.jsp\javax.servlet.jsp-api\1.2

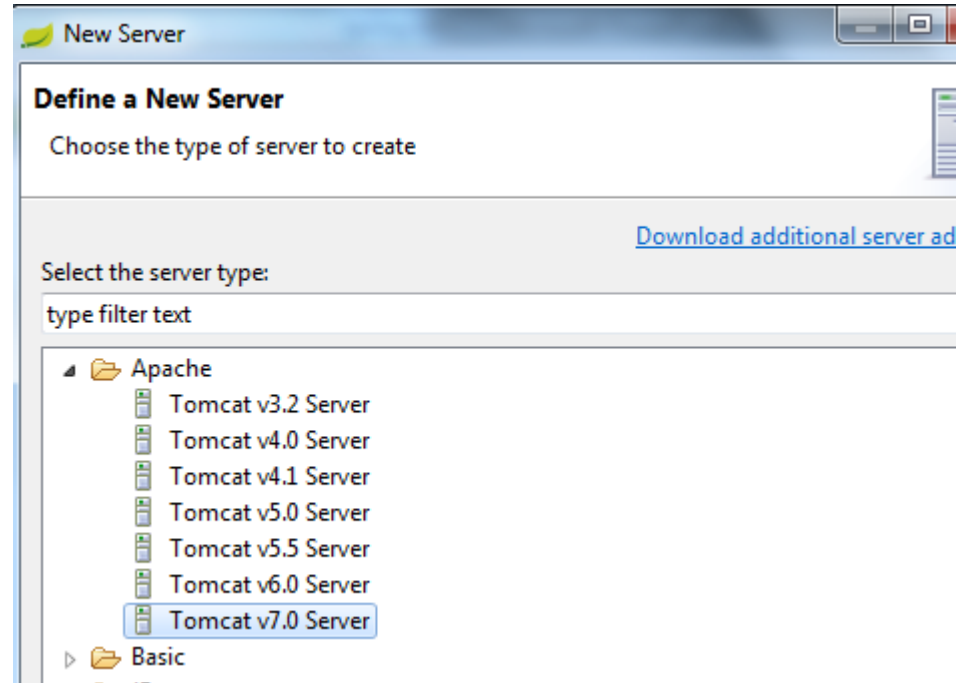
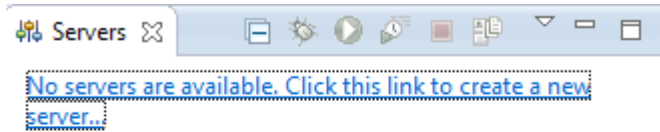
What's need to be configured



Where it going to host

- We are going to use tomcat
- Tomcat is just a web container. It does not have app server functionality
- Download form <http://tomcat.apache.org>
- For this training we are using 7.x which is current

Setup server on IDE



New Server

Tomcat Server

Specify the installation directory

Name:
Apache Tomcat v7.0

Tomcat installation directory:
C:\Krishantha\developments\java\tools\apache-tomcat-7.0.42

JRE:
jre7

apache-tomcat-7.0.12 Do

Add and Remove...

Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available:

Configured:

► SalesTracker

Add >

< Remove

Add All >>

<< Remove All

☒ If server is started, publish changes immediately

Standards

- Put all view technology pages under WEB-INF Directory
 - Doing this we can hide filename from visitors
 - Control the direct navigation. Even user knows about file name they can't access directly
- Internal resource view resolver can resolve the view
- addSales.jsp will come like
 - <http://localhost:8080/salesTracker/addSales>
- Controller names with @Controller annotation
- Name according to your business domain
- Set path using @RequestMapping annotation

Change the web.xml for 2.5 standards

- Default web.xml file use old style configuration. Delete first part and change it to schema based configurations
- ```
<?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">
```

A screenshot of a code editor window titled '\*web.xml'. The editor shows the XML configuration for a web application. The first line is the XML declaration: `<?xml version="1.0" encoding="UTF-8"?>`. The second line is the `<web-app>` tag with `version="2.5"` and two namespace declarations: `xmlns="http://java.sun.com/xml/ns/javaee"` and `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`. The `xsi:schemaLocation` attribute is set to two URIs: `http://java.sun.com/xml/ns/javaee` and `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd`. The third line is `<display-name>Archetype Created Web Application</display-name>`. The tag is closed with `</web-app>`.

```
<?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">

 <display-name>Archetype Created Web Application</display-name>
</web-app>
```

# Lets begin...

---

- Now we going to first create the dispatcher servlet for the application. Open the web.xml file and edit as follows
- Mentioning about the servlet configuration is optional however it is highly recommended to do it.
- Because its tell where it should look for configuration file and how it named
- After that define servlet mapping. It is nothing but route traffic
- Name of the servlet mapping section should be exactly match with servlet section

# Web.xml look like this

```
<servlet>
 <servlet-name>salesTrackerDispatcherServlet</servlet-name> <!-- name can be anything -->
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
 <init-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/config/servletConfig.xml</param-value>
 </init-param>
</servlet>

<servlet-mapping>
 <servlet-name>salesTrackerDispatcherServlet</servlet-name>
 <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

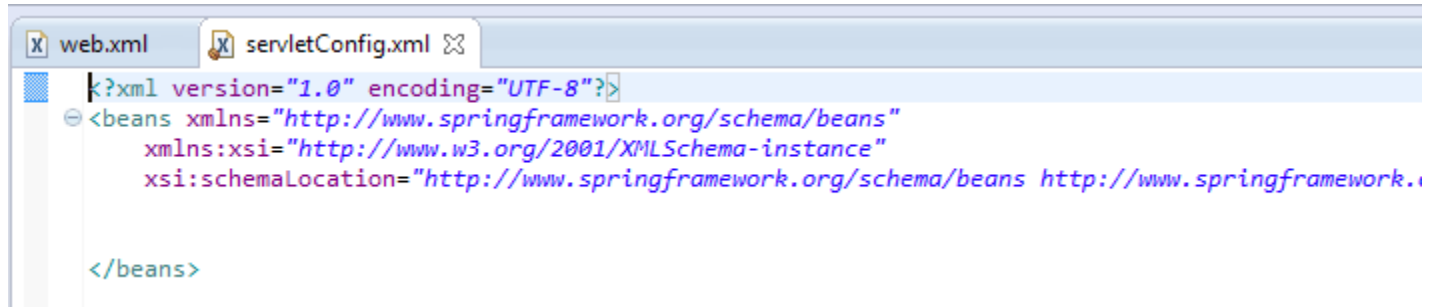
# servletConfig.xml

---

- This is namespace basis
- For use you need to define the namespace
- There are certain pre-defined namespaces comes with IDE

# servletConfig.xml 2

- As we mentioned in web.xml file create the folder structure
- Now add file → new → spring bean configuration file to config directory
- Since we use STS it will add xml header automatically

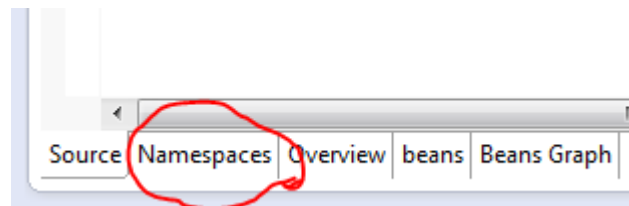


The screenshot shows an IDE window with two tabs: 'web.xml' and 'servletConfig.xml'. The 'servletConfig.xml' tab is active, displaying the following XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd">

</beans>
```

- Click on namespace tab and add MVC and CONTEXT namespaces



## Configure Namespaces

Select XSD namespaces to use in the configuration file

☐ aop - http://www.springframework.org/schema/aop

☒ beans - http://www.springframework.org/schema/beans

☐ c - http://www.springframework.org/schema/c

☐ cache - http://www.springframework.org/schema/cache

☒ context - http://www.springframework.org/schema/context

☐ jee - http://www.springframework.org/schema/jee

☐ lang - http://www.springframework.org/schema/lang

☒ mvc - http://www.springframework.org/schema/mvc

☐ p - http://www.springframework.org/schema/p

☐ task - http://www.springframework.org/schema/task

☐ util - http://www.springframework.org/schema/util

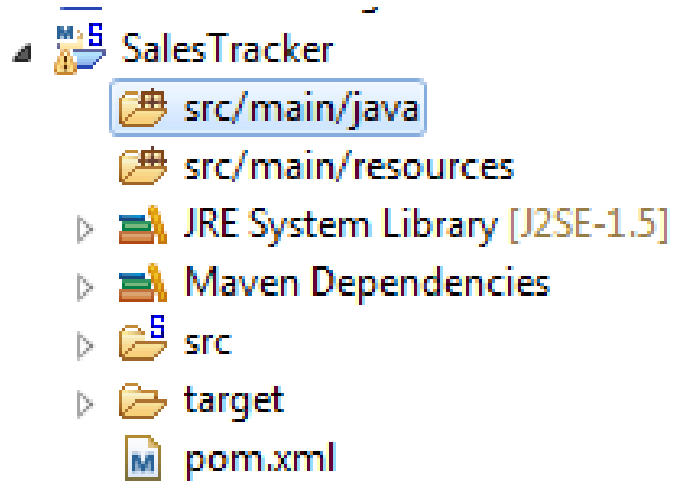
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/beans http://www.s
http://www.springframework.org/schema/context http://www
</beans>
```

# Configure as annotation driven

- Add following lines to config file
- `<mvc:annotation-driven/>`
  - To specify this project is annotation driven
- `<context:component-scan base-package="com.virtusa"/>`
  - To specify where it should look for component

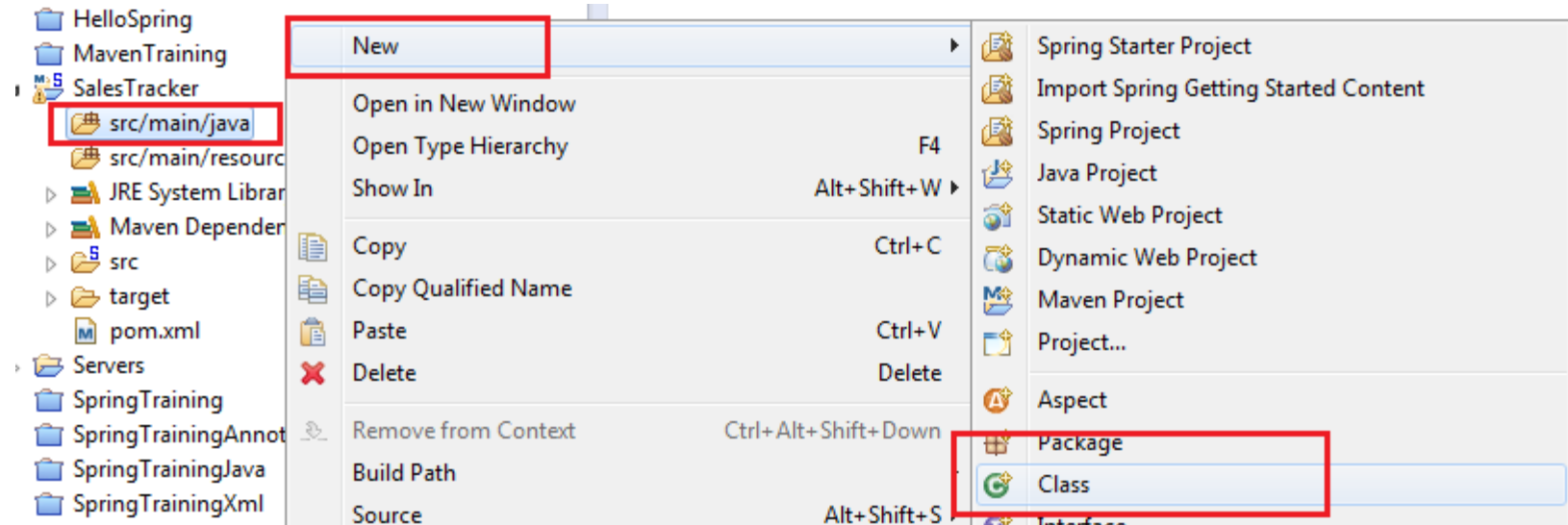
# Create java code

- First create folder structure
- Right click on project and add new folder src/main/java
- Right click on java folder and add to build path if it is not added





# Add new controller class



New Java Class

### Java Class

Source folder: SalesTracker/src/main/java [Browse...](#)

Package: com.virtusa.controller [Browse...](#)

☐ Enclosing type: [Browse...](#)

---

Name: SalesController

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object [Browse...](#)

Interfaces: [Add...](#)  
[Remove](#)

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

[?](#) [Finish](#) [Cancel](#)

# Convert POJO to controller

- Add @Controller annotation
- Add a method to return a greeting message
- @RequestMapping annotation to bind a url to this method
- Add a string to model render on view

```
@Controller
public class SalesController {

 @RequestMapping(value = "/greeting")
 // to bind url to method
 public String sayGreeting(Model model) {
 model.addAttribute("greetingMsg", "Hello Spring MVC.. you looked handy :) ");
 return "welcome"; //jsp page name
 }
}
```

# Create JSP page

- Create new directory under SalesTracker/src/main/webapp/WEB-INF for jsp as jsps
- File → new → jsp
- Make sure to same the file name what we returned from method
- Edit the jsp to print the value comes for model [key]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
 ${greetingMsg}
</body>
</html>
```

```
package com.virtusa.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class SalesController {

 @RequestMapping(value="/greeting") //to handle the request
 public String sayGreeting(Model model){
 model.addAttribute("greetingMsg", "welcome");
 return "welcome";
 }
}
```

# Configure which jsp to pick [routing]

- Need to add a bean to sevletConfig file
- There are two ways to do it. We can use either way
- Long way

```
<bean
class="org.springframework.web.servlet.view.InternalResourceV
iewResolver">
<property name="prefix" value="WEB-INF/jsp/" />
<property name="suffix" value=".jsp"></property>
</bean>
```

# Short way

- Add namespace “P” by navigating to namespace tab
- And now you can add bean like this
- `<bean`  
`class="org.springframework.web.servlet.view.InternalResource`  
`eViewResolver" p:prefix="WEB-INF/jsp/" p:suffix=".jsp"/>`

```
<mvc:annotation-driven/>

<context:component-scan base-package="com.virtusa"/>

<!-- LONG WAY -->
<!--<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="WEB-INF/jsp/" />
<property name="suffix" value=".jsp"></property>
</bean>-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" p:prefix="WEB-INF/jsp/" p:suffix=".jsp"/>
```

# Its time to RUN

---

- Now all required configurations are made.
- Restart tomcat server make sure no errors on startup
- Enter following URL on browser
- <http://localhost:8080/SalesTracker/greeting.html>
- You will see the page

# Anatomy

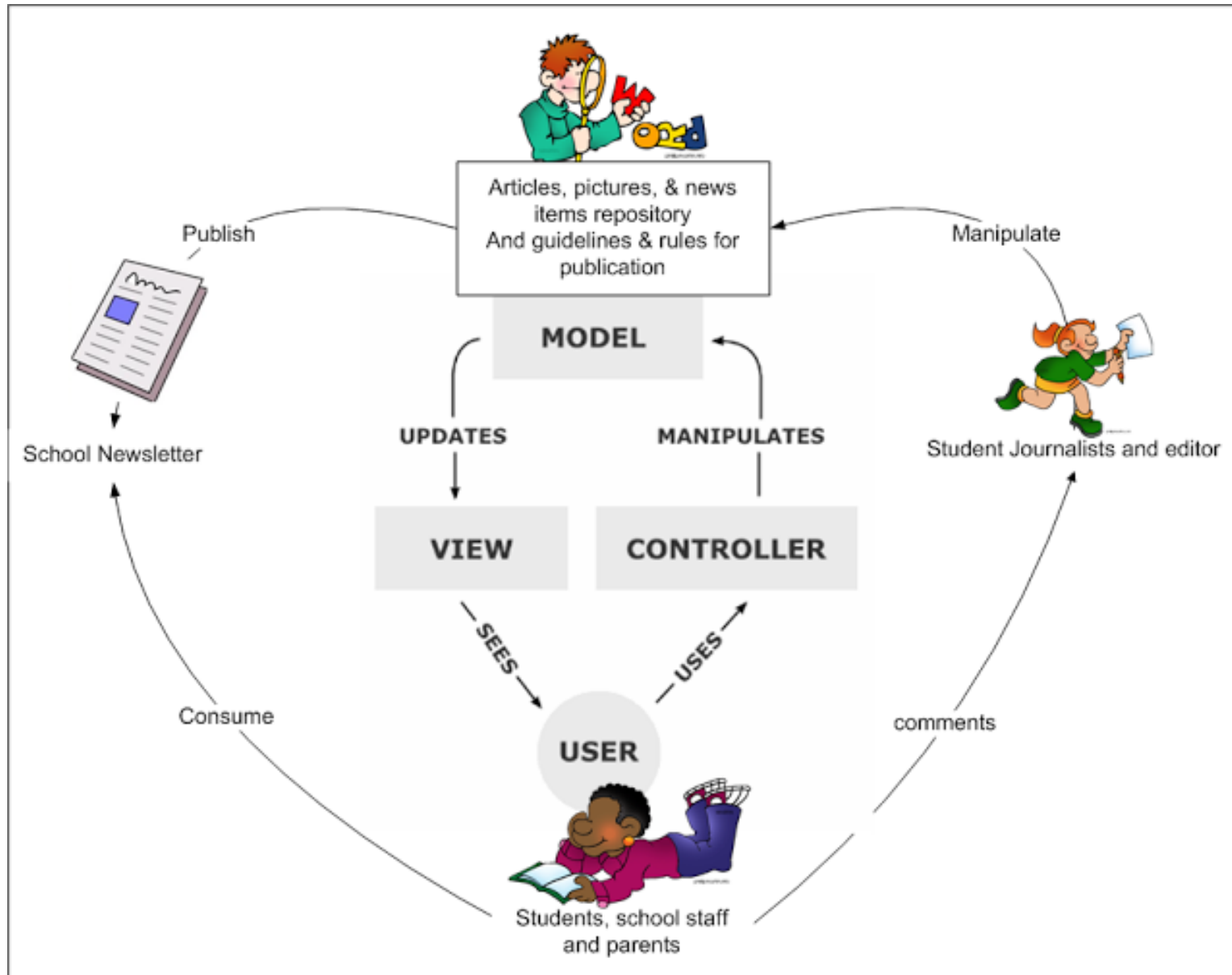
---

- Web.xml has dispatcher servlet mapped to servletConfig.xml
- Added internalResourceViewResolver for resolve jsp pages
- Added java controller
- Added @RequestMapping annotation to tight method with URL
- Added a return jsp name. this return value should same as jsp page

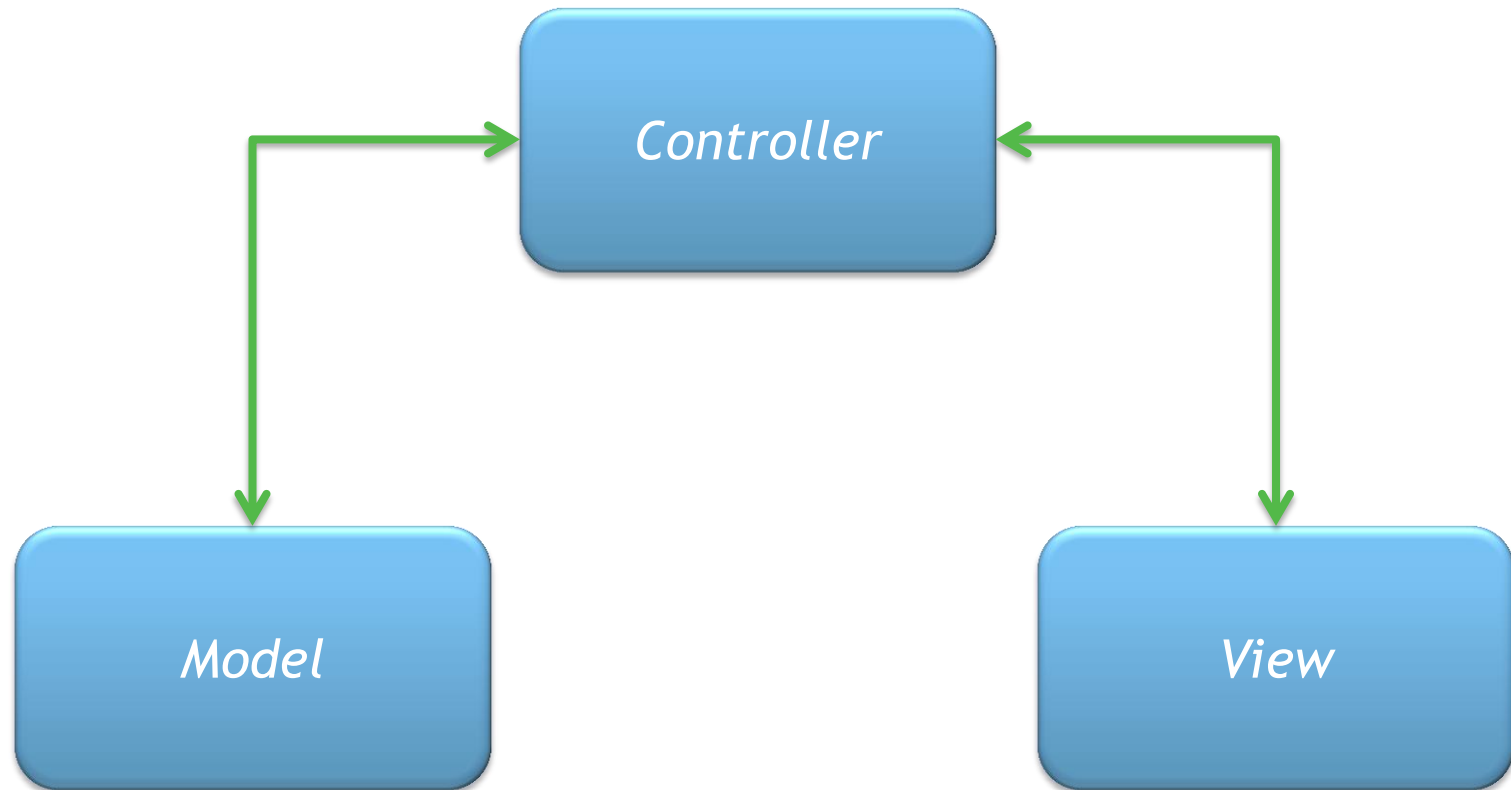


# Architecture

# MVC

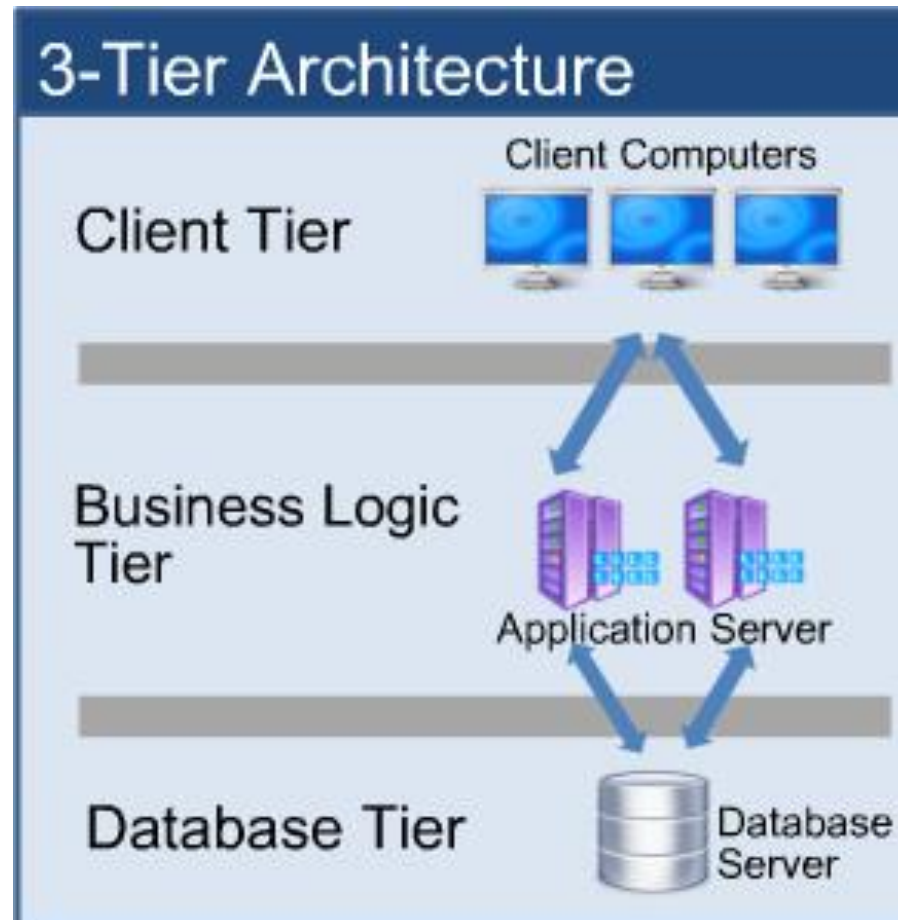


# MVC in web



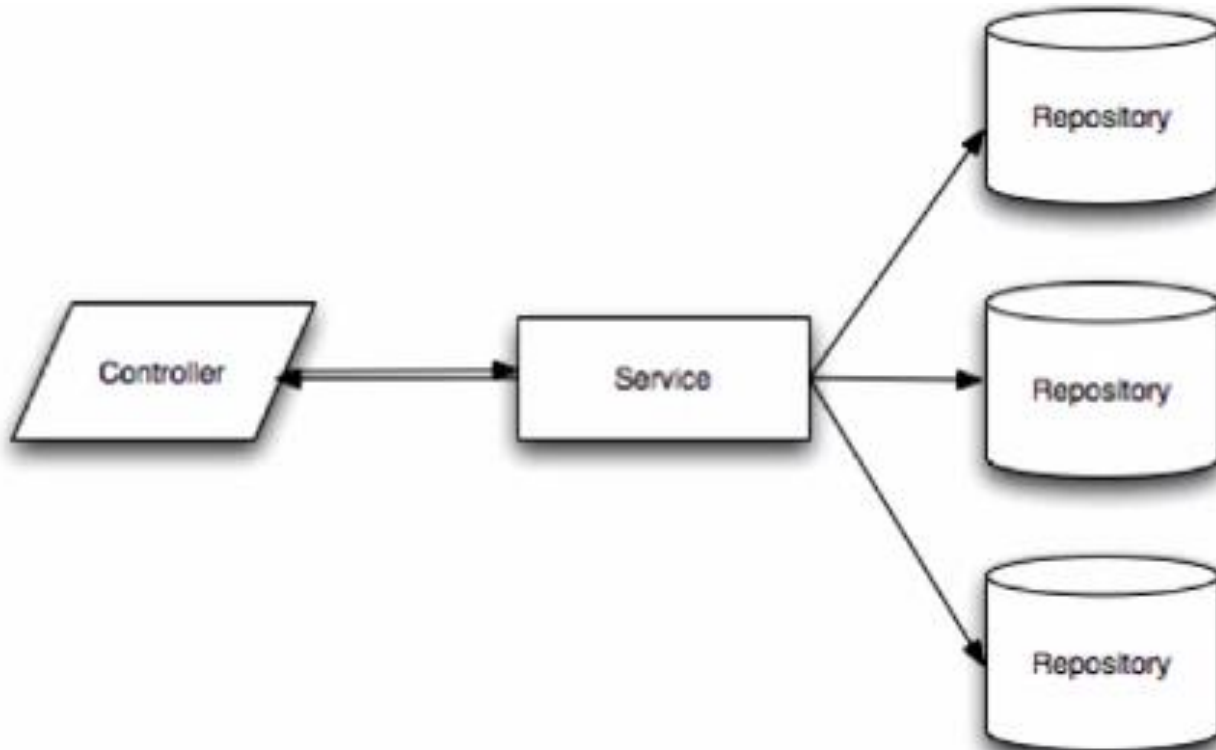
# Tiered architecture

- Main benefit is separation of concerns
- Re usable layers
- Maintenance



# Spring MVC 3 component

- Controller to route traffic
- Service for business logic store also where start our tractions
- Repository tier is almost like DAO



# Controller

---

- Annotated with @Controller
- Suppose to handle incoming request and build the response
- Business logic should NOT put here
- Grab information from request/response and hand over to the business logic
- Handle exception and routed in required way

# Service

---

- Annotated with `@service`
- Service tier define action (verbs) to the system
- Business logic should contained here
- State management should handle here
- Transaction should begins here
- May have same method which repository has but in different focus

# Repository

---

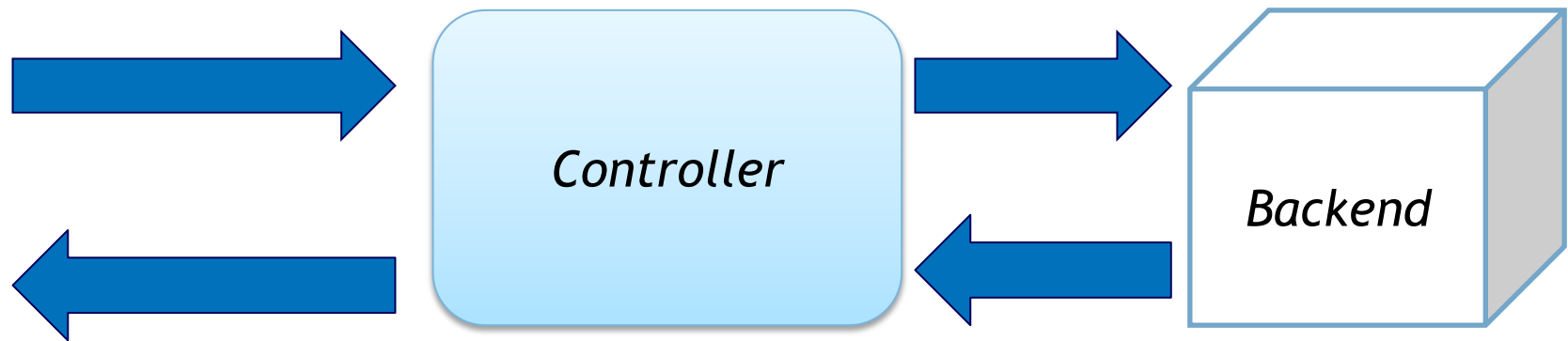
- Annotated with @Repository
- Define data (nouns) to the system
- Focus to interaction with database and CRUD operations
- One to one mapping with object
  - Customer → customerRepository
  - Company → companyRepository



# Controllers

# What is Controller

- It is the center of the concept as well as part of the spring mvc framework
- Decide what to do based on the action



# Controller responsibilities

---

- Transform user input in to model
- Provide access to business logic
- Handle exception and route those in correct way and tier
- Determine view based on logic
- Its is really simple class which has no implements , interfaces etc
- Need @Controller annotation and @RequestMapping. This tells spring which method is going to handle the request
  - Spring do have concrete classes you can extends and it maps url to class based on name. however it is old fashion and NOT depreciated yet.

# Lets change the application in meaningful way

- Add new method to controller and create new jsp file

```
package com.virtusa.controller;
```

```
⊕ import org.springframework.stereotype.Controller;␣
```

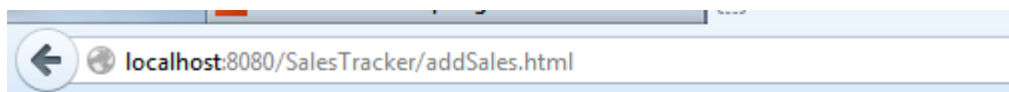
```
@Controller
```

```
public class SalesController {
```

```
⊖ @RequestMapping(value = "/greeting")
 // to bind url to method
 public String sayGreeting(Model model) {
 model.addAttribute("greetingMsg", "Hello Spring MVC.. you looked handy :) ");
 return "welcome"; //jsp page name
 }
```

```
⊖ @RequestMapping(value="/addSales")
 public String addSales(){
 return "addSalesValue";
 }
}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add Sales Value</title>
</head>
<body>
<h1>Add Sales Values</h1>
</body>
</html>
```



## Add Sales Values

# Working with parameter

---

- @ModelAttribute is used to whenever wants to send data to controller or retrieve data from controller
- All these doing using simple POJO
- Used with HTTP GET for get data back
- Use HTTP POST when to send to controller
- Can be validated with binding results

# Customize jsp

- Add following tag library

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

- Add new form

```
<body>
 <h1>Add Sales Values</h1>

 <form:form>
 <table>
 <tr>
 <td>Enter sales Total</td>
 <td><form:input path="sales" /></td>
 </tr>
 <tr>
 <td colspan="2"><input type="submit" value="Save sales" />
 </tr>
 </table>
 </form:form>
</body>
```

# Execute and read the exception

- In simply what it said is it cannot find server side component to serve for value of the form

Stacktrace:

```
org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:568)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:465)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:390)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:334)
javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
org.springframework.web.servlet.view.InternalResourceView.renderMergedOutputModel(InternalResourceView.java:238)
org.springframework.web.servlet.view.AbstractView.render(AbstractView.java:264)
org.springframework.web.servlet.DispatcherServlet.render(DispatcherServlet.java:1208)
org.springframework.web.servlet.DispatcherServlet.processDispatchResult(DispatcherServlet.java:992)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:939)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:856)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:915)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:811)
javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:796)
javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
```



# Fix the problem

- Create new package as com.virtusa.model
- Create new class as sales

```
package com.virtusa.model;

public class Sales {
 private int sales;

 public int getSales() {
 return sales;
 }

 public void setSales(int sales) {
 this.sales = sales;
 }
}
```

# Edit controller to attach the model

```
package com.virtusa.controller;

+ import org.springframework.stereotype.Controller;

@Controller
public class SalesController {

- @RequestMapping(value = "/greeting")
 // to bind url to method
 public String sayGreeting(Model model) {
 model.addAttribute("greetingMsg", "Hello Spring MVC.. you looked handy :) ");
 return "welcome"; // jsp page name
 }

- @RequestMapping(value = "/addSales")
 public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 return "addSalesValue";
 }
}
```

# Edit the jsp file to bind the controller

```
<body>
 <h1>Add Sales Values</h1>

 <form:form commandName="salesValues">
 <table>
 <tr>
 <td>Enter sales Total</td>
 <td><form:input path="sales" /></td>
 </tr>
 <tr>
 <td colspan="2"><input type="submit" value="Save sales" />
 </tr>
 </table>
 </form:form>
</body>
```

# Secret 1

- Has to have model call by command name

```
org.springframework.stereotype.Controller;

class SalesController {

 @RequestMapping(value = "/greeting")
 @ResponseBody
 public String sayGreeting(Model model) {
 model.addAttribute("greetingMsg", "Hello Spring MVC.. you look good");
 return "welcome"; // jsp page name
 }

 @RequestMapping(value = "/addSales")
 public String addSales(@ModelAttribute("salesValues") Sales sales) {
 System.out.println("Sales is " + sales.getSales());
 return "addSalesValue";
 }
}
```

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"%>
<title>Add Sales Value</title>
</head>
<body>
<h1>Add Sales Values</h1>
<form:form commandName="salesValues">
<table>
<tr>
<td>Enter sales Total</td>
<td><form:input path="sales" /></td>
</tr>
<tr>
<td colspan="2"><input type="submit" value="Save sales" />
</td>
</tr>
</table>
</form:form>
</body>
</html>
```

localhost:8080/SalesTracker/addSales.html

## Add Sales Values

Enter sales Total

# Secret 2

- Has to have a method by path name

```
package com.virtusa.model;
```

```
public class Sales {
 private int sales;
```

```
 public int getSales() {
 return sales;
 }
```

```
 public void setSales(int sales) {
 this.sales = sales;
 }
```

```
}
```

```
Read in: English || <a href="?lang=i
```

```
<h1>Add Sales Values</h1>
```

```
<form:form commandName="salesValues">
```

```
 <table>
```

```
 <tr>
```

```
 <td><spring:message code="sales.enter"/> </td>
```

```
 <td><form:input path="Sales" /></td>
```

```
 </tr>
```

```
 <tr>
```

```
 <td colspan="2"><input type="submit" value="Submit" /></td>
```

```
 </tr>
```

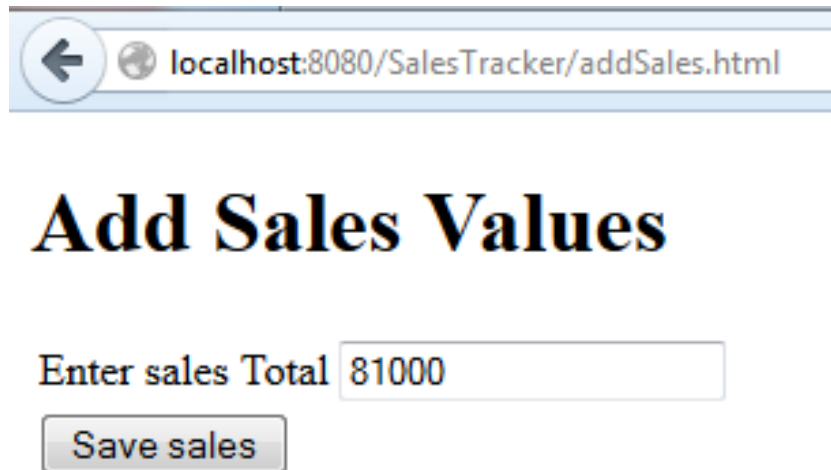
```
 </table>
```

```
</form:form>
```

```
</body>
```

# It is working.

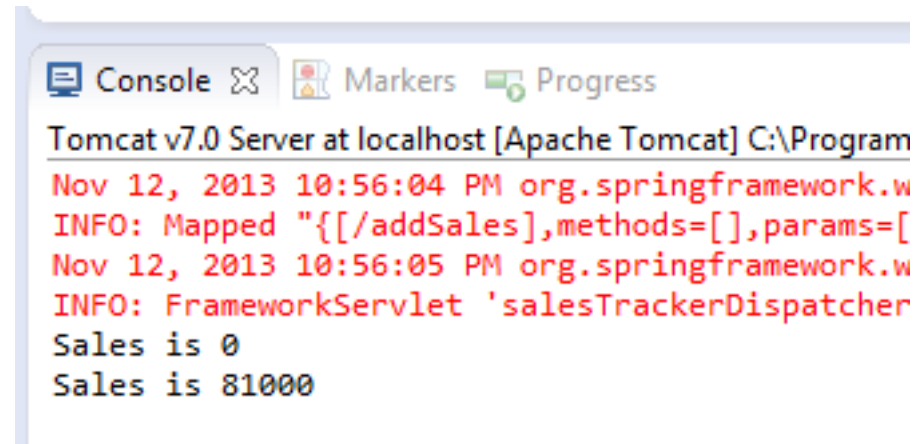
- Just run the jsp page and enter value and submit
- Go to IDE and see console window



localhost:8080/SalesTracker/addSales.html

## Add Sales Values

Enter sales Total



```
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program
Nov 12, 2013 10:56:04 PM org.springframework.w
INFO: Mapped "{[/addSales],methods=[],params=[
Nov 12, 2013 10:56:05 PM org.springframework.w
INFO: FrameworkServlet 'salesTrackerDispatcher
Sales is 0
Sales is 81000
```

# Views

# About view

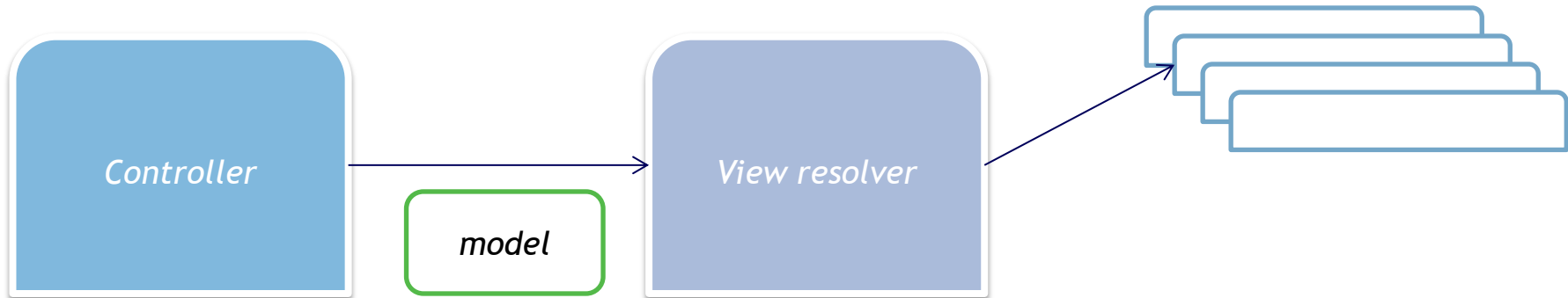
---

- View is the V part of mvc and also what we see
- As a convention we place all jsp in jsps directory inside WEB-INF
- It will help to prevent deep link and book marking as user must navigate through the our page navigation system means to secure our application
- Should not carry any business logic here as this is dedicated for presentation part
- We use internalResourceViewResolver to pick the correct view. It can find view by looking the string view return



# View - advance

- Controller and pass model to view resolver with the data for view if necessary.
- View can render those data for presentations



# View resolvers

---

- There are many view resolvers ship with spring and you can have your own. It just implement `ViewResolver` interface
  - `BeanNameViewResolver`
  - `ContentNegotiatingViewResolver`
  - `JasperReportViewResolver`
  - `TilesViewResolver`
  - `UrlBasedViewResolver`
  - `ResourceBundleViewResolver`

# Chaining

- There are 2 types of chaining
- Chaining helpful to navigation specially on redirections

```
@RequestMapping(value = "/greeting")
// to bind url to method
public String sayGreeting(Model model) {
 model.addAttribute("greetingMsg", "Hello Spring MVC.. you looked handy :) ");
 return "welcome"; // jsp page name
}
```

```
@RequestMapping(value = "/addSales")
public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 return "forward:addBonusSales";
}
```

```
@RequestMapping(value = "/addBonusSales")
public String addBonusSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Bonus Sales is " + sales.getSales());
 return "addSalesValue";
}
```

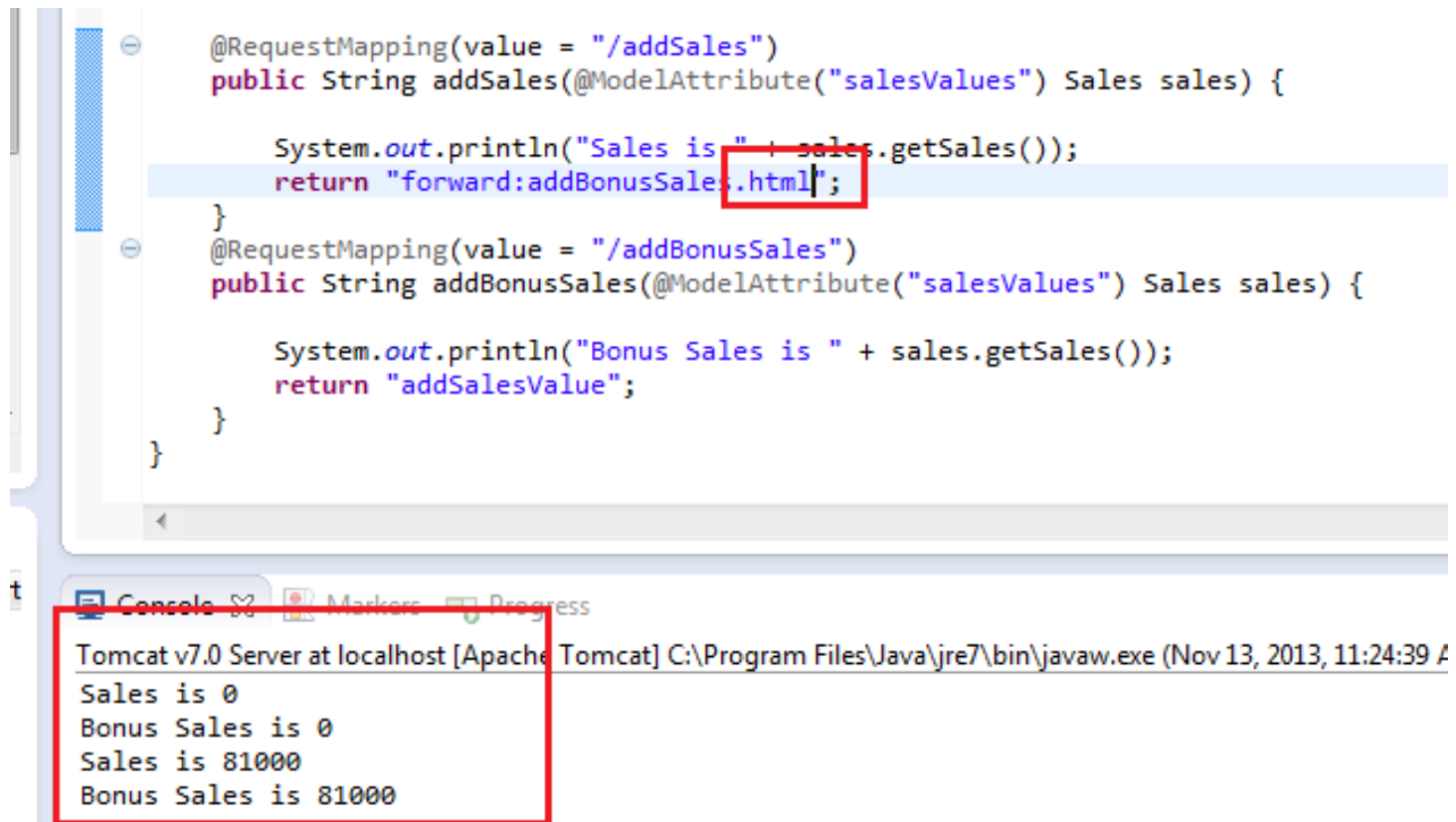
# Execute

---

- Now try to browse the page and see what you can see
- **Why**



- Because when forward chain requested it go out from container and comes back. Since we mentioned to process only \*.html files in web.xml it gives error.
- Fix the code like this



The screenshot shows an IDE with two panels. The top panel displays Java code for two Spring MVC controllers. The bottom panel shows the console output of a Tomcat server.

```
@RequestMapping(value = "/addSales")
public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 return "forward:addBonusSales.html";
}

@RequestMapping(value = "/addBonusSales")
public String addBonusSales(@ModelAttribute("salesValues") Sales sales) {

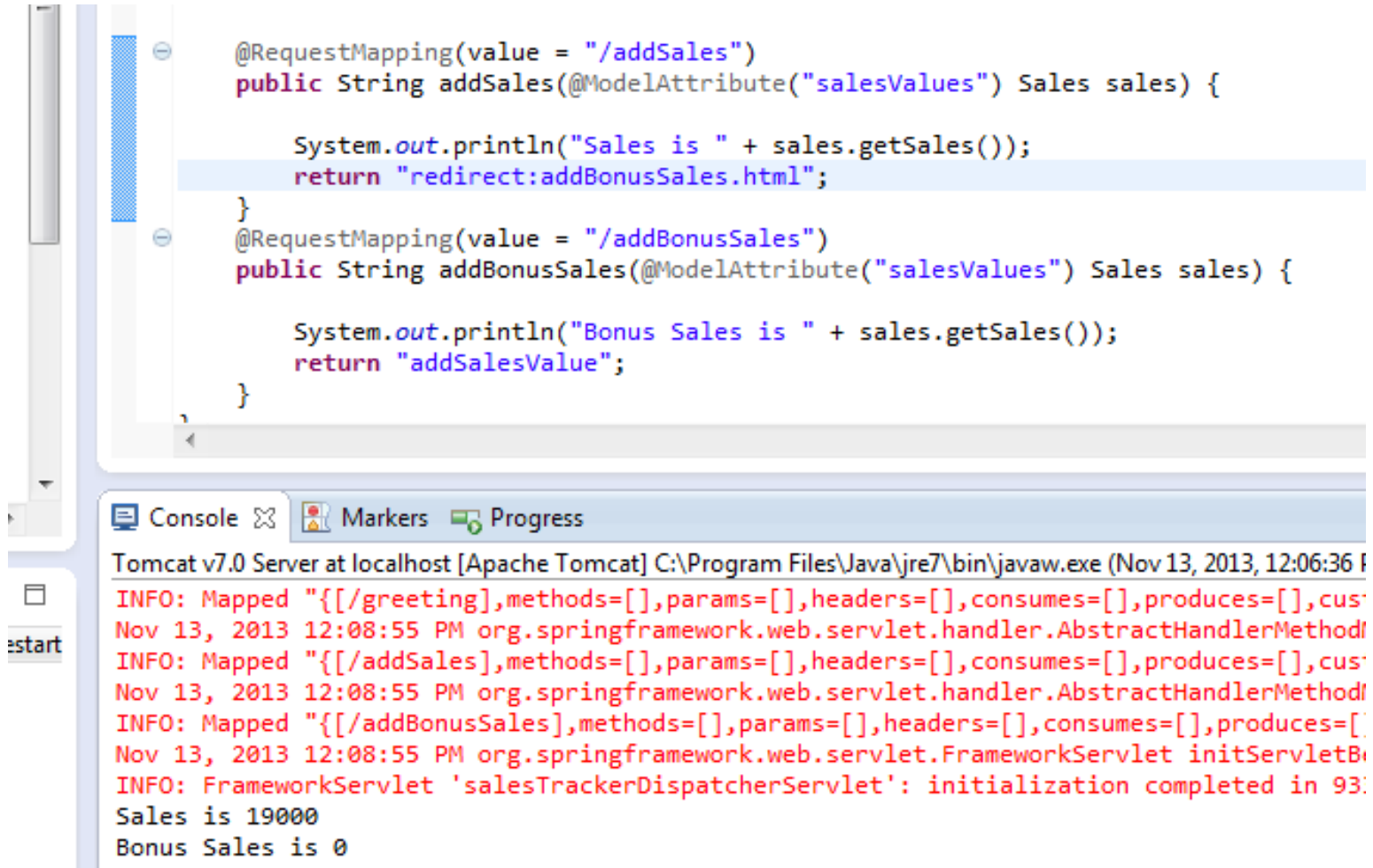
 System.out.println("Bonus Sales is " + sales.getSales());
 return "addSalesValue";
}
```

Console Output:

```
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 13, 2013, 11:24:39 A
Sales is 0
Bonus Sales is 0
Sales is 81000
Bonus Sales is 81000
```

# Chaining - redirect

- Change forward to redirect and see how its work



The screenshot shows an IDE with two Java classes and a console window. The first class, `addSales`, has a `@RequestMapping` for `"/addSales"` and returns a redirect to `addBonusSales.html`. The second class, `addBonusSales`, has a `@RequestMapping` for `"/addBonusSales"` and returns `addSalesValue`. The console window shows the output of the application, including the mapping of the two controllers and the execution of the `addSales` method, which results in a redirect to `addBonusSales.html`.

```
@RequestMapping(value = "/addSales")
public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 return "redirect:addBonusSales.html";
}

@RequestMapping(value = "/addBonusSales")
public String addBonusSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Bonus Sales is " + sales.getSales());
 return "addSalesValue";
}
```

Console

Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 13, 2013, 12:06:36 PM)

INFO: Mapped "{[/greeting],methods=[],params=[],headers=[],consumes=[],produces=[],contentType=application/json}" to handler method org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter@1a2b3c4d:org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter\$1.handle

Nov 13, 2013 12:08:55 PM org.springframework.web.servlet.handler.AbstractHandlerMethodAdapter:111 INFO: Mapped "{[/addSales],methods=[],params=[],headers=[],consumes=[],produces=[],contentType=application/json}" to handler method org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter@1a2b3c4d:org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter\$1.handle

Nov 13, 2013 12:08:55 PM org.springframework.web.servlet.handler.AbstractHandlerMethodAdapter:111 INFO: Mapped "{[/addBonusSales],methods=[],params=[],headers=[],consumes=[],produces=[],contentType=application/json}" to handler method org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter@1a2b3c4d:org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter\$1.handle

Nov 13, 2013 12:08:55 PM org.springframework.web.servlet.FrameworkServlet:144 INFO: FrameworkServlet 'salesTrackerDispatcherServlet': initialization completed in 93ms

Sales is 19000

Bonus Sales is 0

- Because when you use redirect it close the existing request and create new request
- But you can see you cant test is as on first run it redirected to bonuspage.
- Make following change to overcum this.

```
@RequestMapping(value = "/addSales")
public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 if (sales.getSales() > 0)
 return "redirect:addBonusSales.html";
 else
 return "addSalesValue";
}
```

# Resolve static views

- So far we know how we can resolve dynamic views
- Reset project to the status where we was
- Add `mvc:resources` to `servletConfig.xml`
- Update `web.xml` to accept pdf
- Create new folder and add files for it
- Try to browse the file via URL
- You can see its bypass standard controller



```
<servlet-mapping>
 <servlet-name>salesTrackerDispatcherServlet</servlet-name>
 <url-pattern>*.html</url-pattern>
</servlet-mapping>

<servlet-mapping>
 <servlet-name>salesTrackerDispatcherServlet</servlet-name>
 <url-pattern>/pdfs/**</url-pattern>
</servlet-mapping>
```

localhost:8080/SalesTracker/pdfs/Krishantha-profile.pdf

Page: 1 of 5

Automatic Zoom

virtusa

## Krishantha Dinesh

Senior Consultant - Technology

A technically competent and industry practiced experienced IT professional with expertise

virtusa®

# Tag library

# What are the tags In spring

---

- There are two type of tag libraries available in spring

## 1. Spring.tlg

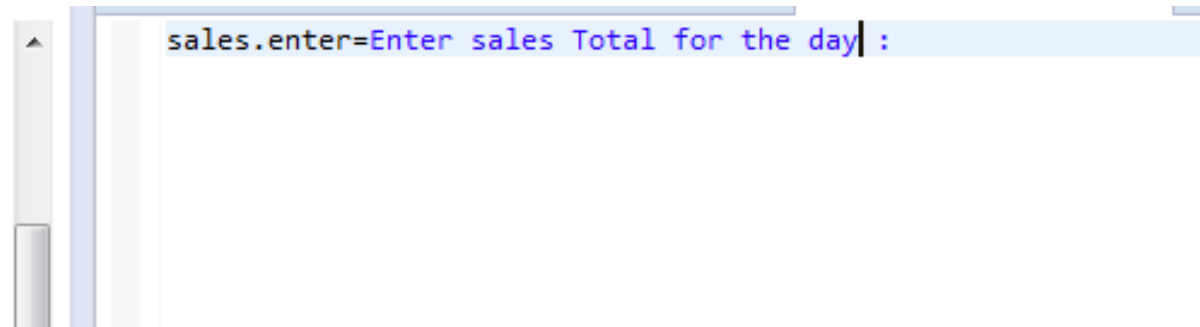
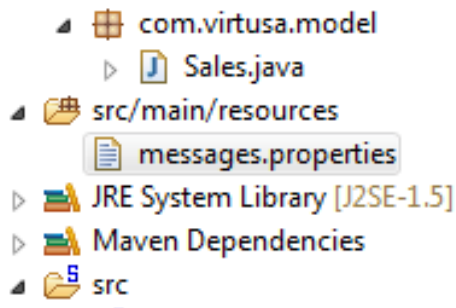
1. Validating error
2. Output internationalize messages
3. Setting themes

## 2. Spring-form.tld

1. Processing form data
2. Came from HTML form tag

# Message tag for get message from external files

- Instead of hard code message in a application we can get captions from text files.
- We can create property file and use tag libraries for read those file
- Create property file in resource section of the project. Maven will automatically add it in to classpath
- Add enter sales caption for that file



# Change jsp file

- Edit jsp file and add tag library and add spring tag over text

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add Sales Value</title>
</head>
<body>
 <h1>Add Sales Values</h1>

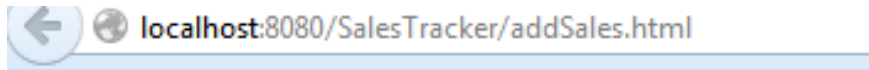
 <form:form commandName="salesValues">
 <table>
 <tr>
 <td><spring:message code="sales.enter"/> </td>
 <td><form:input path="sales" /></td>
 </tr>
 <tr>
 <td colspan="2"><input type="submit" value="Save sales" />
 </tr>
 </table>
 </form:form>

</body>
</html>
```

# Create a bean for process

- Open servletConfig.xml and add following bean

```
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMess
ageSource" p:basename="messages" />
```



## Add Sales Values

Enter sales Total for the day :

# Interceptors

---

- It is part of request life cycle and usually use to intercept the data which comes from jsp on the way to controller
- It has ability to intercept data on the way to controller or way back from controller. [pre-handle and post-handle]
- It has override methods to change values
- we can use this for multi language application

# Modify the page to support for multiple language

```
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <title>Add Sales Value</title>
</head>
<body>

 Read in: English || Japanese

 <h1>Add Sales Values</h1>

 <form:form commandName="salesValues">
 <table>
 <tr>
 <td><spring:message code="sales.enter"/> </td>
 <td><form:input path="sales" /></td>
 </tr>
 </table>
 </form:form>
</body>
```

Read in: [English](#) || [Japanese](#)

## Add Sales Values

Enter sales Total for the day :



# Create message file

- Add new property file for the resource folder with the \_jp example messages\_jp.properties for japan
- Translate your text and paste their
- And create new bean now (see next page)

```
<bean id="LocaleResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver"
p:defaultLocale="en"/>
```

```
<!-- register interceptor -->
```

```
<mvc:interceptors> <bean
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
p:paramName="lang"/>
```

- </mvc:interceptors>

# Change locale

- com.virtusa.model
  - Sales.java
- src/main/resources
  - messages\_it.properties
  - messages.properties
- JRE System Library [J2SE-1.5]
- Maven Dependencies

sales.enter=Inserisci le vendite totali per il giorno :

```
<bean id="localeResolver"
 class="org.springframework.web.servlet.i18n.SessionLocaleResolver"
 p:defaultLocale="en" />
<!-- register interceptor -->
<mvc:interceptors>
 <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
 p:paramName="lang" />
</mvc:interceptors>
```

Read in: [English](#) || [Italian](#)

## Add Sales Values

Inserisci le vendite totali per il giorno :

# Extend the application

---

- We are going to extend our application
- Create other page to enter sales target
- R/C → new → jsp [addSalesTarget]

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add Sales Target</title>
</head>
<body>
 <form:form commandName="addTarget">
 <table>
 <tr>
 <td>Add Sales Target</td>
 <td><form:input path="salesTarget" /></td>
 </tr>
 <tr>
 <td colspan="2"><input type="submit" value="Save Target" />
 </tr>
 </table>
 </form:form>
</body>
</html>

```

# Add Target object to model

```
package com.virtusa.model;

public class Sales {
 private int salesVal;

 public int getSales() {
 return salesVal;
 }

 public void setSales(int sales) {
 this.salesVal = sales;
 }
}
```

# Controller

- This controller design in different way
- We get Model object to in to send the data to jsp
- Also we limit method only to GET

```
@Controller
public class SalesTargetController {

 // new way to do. we can use this mode to send data to jsp
 @RequestMapping(value = "/addTarget", method = RequestMethod.GET)
 public String addSalesTarget(Model model) {
 model.addAttribute("target", new SalesTarget());
 return "addSalesTarget";
 }
}
```

# Now try to load and submit the page

---



# Where it went wrong

- Since we added .GET it will not accept the post requests
- Remove the GET part and see if it worked

```
⊕ import org.springframework.stereotype.Controller;

@Controller
public class SalesTargetController {

 // new way to do. we can use this mode to send data to jsp
 ⊖ @RequestMapping(value = "/addTarget")
 public String addSalesTarget(Model model) {
 model.addAttribute("addTarget", new SalesTarget());
 return "addSalesTarget";
 }
}
```



# Save data

---

- Since we use new SalesTarget() at every time it kills the data.
- Lets try to save the data to session!!!!
- It is easy with spring mvc. Add @SessionAttribute("<what you want to save")
- We are going to save model to session
- Also add new method for post request
- Redirect to addSales page once done

```

@Controller
@SessionAttributes("addTarget")
public class SalesTargetController {

 // new way to do. we can use this mode to send data to jsp
 @RequestMapping(value = "/addTarget", method = RequestMethod.GET)
 public String addSalesTarget(Model model) {
 SalesTarget salesTarget = new SalesTarget();
 salesTarget.setSalesTargetValue(1000);
 model.addAttribute("addTarget", salesTarget);
 return "addSalesTarget";
 }

 @RequestMapping(value = "/addTarget", method = RequestMethod.POST)
 public String updateSalesTarget(
 @ModelAttribute("addTarget") SalesTarget salesTarget) {
 System.out.println("Target updated : "
 + salesTarget.getSalesTargetValue());
 return "redirect:addSales.html";
 }
}

```

# Now run and see

```
public String addSalesTarget(Model model) {
 SalesTarget salesTarget = new SalesTarget();
 salesTarget.setSaleTargetValue(81000);
 model.addAttribute("addTarget", salesTarget);
 return "addSalesTarget";
}
```

Add Sales Target

Read in: [English](#) || [Italian](#)

Add Sales Target

## Add Sales Values

Enter sales Total for the day :

```
Tomcat v7.0 Server at localhost [Apache Tor
Target updated :91000
Sales is 0
```

# Make sure its come from session

- Change the jsp as follows and test

```
SalesTargetController.java addSalesValue.jsp SalesTarg

<h1>Add Sales Values</h1>

<form:form commandName="salesValues">
 <table>
 <tr>
 <td><spring:message code="sales.ente
 <td><form:input path="Sales" /></td>
 </tr>
 <tr>
 <td colspan="2"><input type="submit"
 </tr>
 </table>
</form:form>
From Session

<h1>
Target set as : ${addTarget.saleTargetValue }
</h1>
</body>
</html>
```

## Add Sales Values

Enter sales Total for the day :

From Session

**Target set as : 81000**

# Validations

- Validations we do perform for any application development
- Spring mvc has a capability to the validation through more systematic way
- Validation are two types
  1. Business logic validation
    1. Validation should not bring to controller. Need to handle those in service tier
  2. Restriction validation
    1. Name not null , age >20 etc

# Errors with tag

---

- We have tag libraries to handle the errors
- On that there are specific libraries to display errors
- Those can change colors , fonts etc based on error
- We can use validator interface and validationUtil helper class by the way this is old fashion
- Now we use JSR-303 standards specification for validation. This is java standards
- It use hibernate validator and It is annotation based

# Lets validate to define minimum target

- In our application there is not minimum target defined. Lets define it using JSR-303
- Add following dependancy for pom file

```
<dependency>
```

```
<groupId>org.hibernate</groupId>
```

```
<artifactId>hibernate-validator</artifactId>
```

```
<version>4.2.0.Final</version>
```

```
</dependency>
```



# Add validation to Target

```
package com.virtusa.model;

import org.hibernate.validator.constraints.Range;

public class SalesTarget {
 @Range(min = 1000, max = 100000)
 private int saleTargetValue;

 public int getSaleTargetValue() {
 return saleTargetValue;
 }

 public void setSaleTargetValue(int saleTargetValue) {
 this.saleTargetValue = saleTargetValue;
 }
}
```

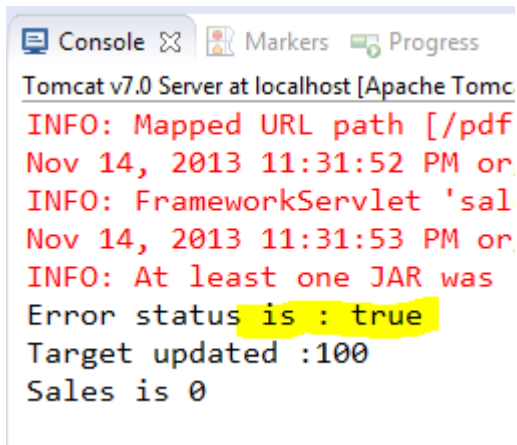
# Controller to validate the validator

- Now need to configure controller to say not to accept if not validated

```
@RequestMapping(value = "/addTarget", method = RequestMethod.POST)
public String updateSalesTarget(
 @Valid @ModelAttribute("addTarget") SalesTarget salesTarget, BindingResult bindingResult) {

 System.out.println("Error status is : " + bindingResult.hasErrors());

 System.out.println("Target updated : " + salesTarget.getSaleTargetValue());
 return "redirect:addSales.html";
}
```



Console Markers Progress

Tomcat v7.0 Server at localhost [Apache Tomc

INFO: Mapped URL path [/pdf

Nov 14, 2013 11:31:52 PM on

INFO: FrameworkServlet 'sal

Nov 14, 2013 11:31:53 PM on

INFO: At least one JAR was

Error status is : true

Target updated :100

Sales is 0

## Add Sales Values

Enter sales Total for the day :

From Session

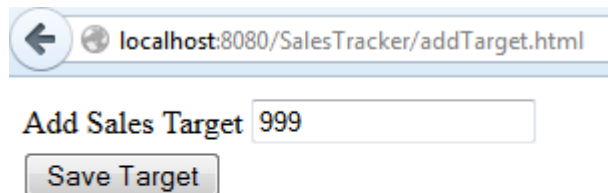
**Target set as : 100**

# Error not handled!!!!

- Even it has errors its loaded because its not doing anything
- Now handle if error exist

```
@RequestMapping(value = "/addTarget", method = RequestMethod.POST)
public String updateSalesTarget(
 @Valid @ModelAttribute("addTarget") SalesTarget salesTarget,
 BindingResult bindingResult) {
 if (bindingResult.hasErrors()){
 System.out.println("Ended with error");
 return "addSalesTarget";
 }

 System.out.println("Target updated : "
 + salesTarget.getSaleTargetValue());
 return "redirect:addSales.html";
}
```



← localhost:8080/SalesTracker/addTarget.html

Add Sales Target

```
Servlet: See http://m
Nov 14, 2013 11:37:
INFO: Mapped URL pa
Nov 14, 2013 11:37:
INFO: FrameworkServ
Ended with error
```

# No error messages ☹️

- We need to add meaning full messages to user to see where it went wrong
- Change the jsp to post error

```
<title>Add Sales Target</title>
<style>
.error {
 color: #ff0000;
}

.errorborder {
 color: #000000;
 background-color: #ffEEEE;
 border: 3px solid #ff0000;
 padding: 8px;
 margin: 16px;
}
</style>
</head>
```

```
<body>
 <form:form commandName="addTarget">
 <form:errors path="*" cssClass="errorborder" element="div" />
 <table>
 <tr>
 <td>Add Sales Target</td>
 <td><form:input path="saleTargetValue" /></td>
 <td><form:errors path="saleTargetValue" cssClass="error" /></td>
 </tr>
 <tr>
 <td colspan="3"><input type="submit" value="Save Target" />
 </tr>
 </table>
 </form:form>
</body>
..
-
```

# Message is not mean full !!!!

← localhost:8080/SalesTracker/addTarget.html

must be between 1000 and 100000

Add Sales Target 999 must be between 1000 and 100000

Save Target

# Customize error message

- Change the message.properties file and add new message

```
Range.addTarget.saleTargetValue=Sales Target should be in 1000 and 10000
```

```
public class SalesTarget {
 @Range(min = 1000, max = 100000)
 private int saleTargetValue;
```

```
 public int getSaleTargetValue() {
 return saleTargetValue;
 }
```

```
 public void setSaleTargetValue(int saleTargetValue) {
 this.saleTargetValue = saleTargetValue;
 }
```

```
@RequestMapping(value = "/addTarget", method = RequestMethod.
public String updateSalesTarget(
 @Valid @ModelAttribute("addTarget") SalesTarget s
 BindingResult bindingResult) {
```

```
 if (bindingResult.hasErrors()){
 System.out.println("Ended with error");
 }
```

Sales Target should be in 1000 and 10000

Add Sales Target

Sales Target should be in 1000 and 10000

Save Target



# Highlight the text color when has error

```
<form:form commandName="addTarget">
 <form:errors path="*" cssClass="errorborder" element="div" />
 <table>
 <tr>
 <td>Add Sales Target</td>
 <td><form:input path="saleTargetValue" cssErrorClass="error" /></td>
 <td><form:errors path="saleTargetValue" cssClass="error" /></td>
 </tr>
 <tr>
 <td colspan="3"><input type="submit" value="Save Target" />
 </td>
 </table>
```

Sales Target should be in 1000 and 10000

Add Sales Target  Sales Target should be in 1000 and 10000

# Ajax

# REST ???

---

- Spring MVC able to produce JSON services
- More easy consumable across the application
- Using deferent view resolver we can produce JSON
- REST services mean what we can do with domain object and can be interact with services (action)
- REST design almost like CRUD function replace POST,GET,PUT,DELETE respectively

# ContentNegotiatingViewResolver

---

- It use to handle multiple content type based on request
- Accept XML,JSON,HTML headers
- Can be combine with other view resolvers
- Need to have additional jar
- Add following dependency

```
<dependency>
 <groupId>org.codehaus.jackson</groupId>
 <artifactId>jackson-mapper-asl</artifactId>
 <version>1.4.1</version>
</dependency>
<dependency>
 <groupId>com.thoughtworks.xstream</groupId>
 <artifactId>xstream</artifactId>
 <version>1.3.1</version>
</dependency>
<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-oxm</artifactId>
 <version>3.2.0.RELEASE</version>
</dependency>
```

# Add BIG bean

```
<bean
 class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
 <property name="order" value="1" />
 <property name="contentNegotiationManager">
 <bean class="org.springframework.web.accept.ContentNegotiationManager">
 <constructor-arg>
 <bean
 class="org.springframework.web.accept.PathExtensionContentNegotiationStrategy">
 <constructor-arg>
 <map>
 <entry key="json" value="application/json"></entry>
 <entry key="xml" value="application/xml"></entry>
 </map>
 </constructor-arg>
 </bean>
 </constructor-arg>
 </bean>
 </property>

 <property name="defaultViews">
 <list>
 <bean
 class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />
 <bean class="org.springframework.web.servlet.view.xml.MarshallingView">
 <constructor-arg>
 <bean class="org.springframework.oxm.xstream.XStreamMarshaller">
 <property name="autodetectAnnotations" value="true" />
 </bean>
 </constructor-arg>
 </bean>
 </list>
 </property>
 </bean>
```

# Return value

- Now beans set up. We need to return values
- Add new class to model

```
package com.virtusa.model;

public class SalesType {

 private String salesTypes;

 public String getSalesTypes() {
 return salesTypes;
 }

 public void setSalesTypes(String salesTypes) {
 this.salesTypes = salesTypes;
 }

}
```

# Change SalesController

```
@RequestMapping(value = "/addSales")
public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 return "addSalesValue";
}

@RequestMapping(value="/salesTypes",method=RequestMethod.GET)
public @ResponseBody List <SalesType> getAllSalesTypes(){
 List<SalesType> salesTypes = new ArrayList<SalesType>();

 SalesType direct = new SalesType();
 direct.setSalesTypes("Direct");
 salesTypes.add(direct);

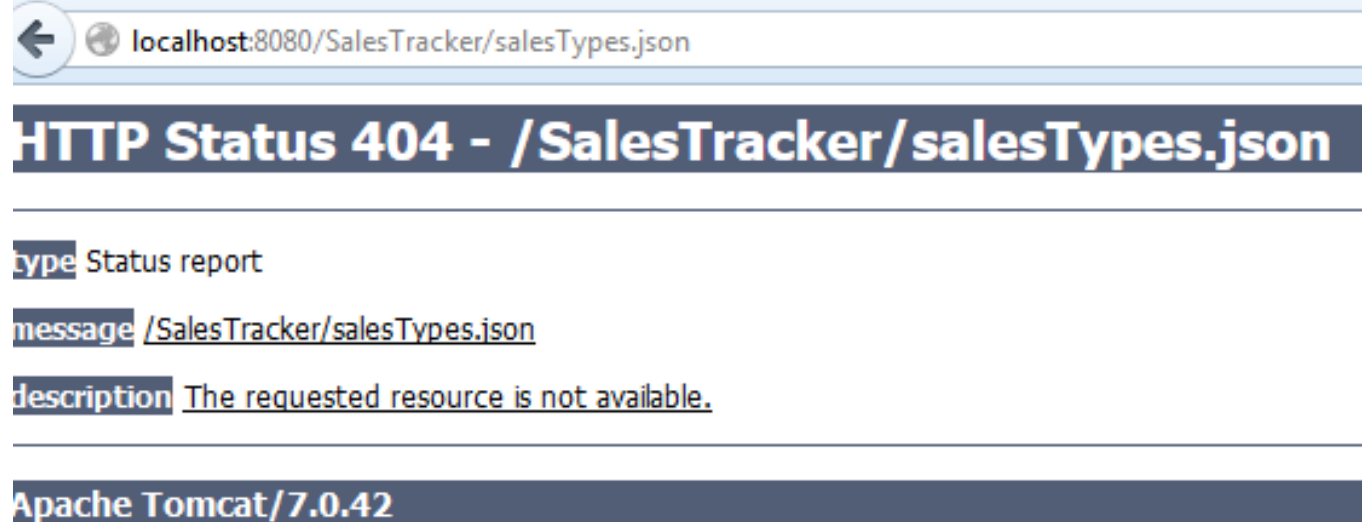
 SalesType web = new SalesType();
 web.setSalesTypes("Web");
 salesTypes.add(web);

 SalesType shop = new SalesType();
 shop.setSalesTypes("Direct");
 salesTypes.add(shop);

 return salesTypes;
}
```



# Try to execute



# Change web.xml and see

---

```
<servlet-mapping>
 <servlet-name>salesTrackerDispatcherServlet</servlet-name>
 <url-pattern>*.json</url-pattern>
</servlet-mapping>
```

# Merge with JQuery

---

- Download jquery and add to project
- Browse following link and save to computer
  - <http://code.jquery.com/jquery-1.10.2.js>
- Now drag it to web app directory
- Now change addSales.jsp

```
<form:form commandName="salesValues">
 <table>
 <tr>
 <td><spring:message code="sales.enter" /></td>
 <td><form:input path="Sales" /></td>
 <td><form:select id="saletypes" path="salesTypes"></form:select>
 </td>
 </tr>
 <tr>
 <td colspan="3"><input type="submit" value="Save Sales Values" />
 </td>
 </tr>
</form>
```

```

<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add Sales Value</title>

<script type="text/javascript" src="jquery-1.10.2.js"></script>

<script type="text/javascript">
$(document).ready(
 function(){
 $.getJSON('http://localhost:8080/SalesTracker/salesTypes.json',{
 ajax:'true'
 }, function(data){
 var html='<option value="">--select type-- </option>';
 var len=data.length;
 for (var i=0;i<len;i++){
 html += '<option value="'+data[i].salesTypes +' ">'
 +data[i].salesTypes+ '</option>';
 }
 html += '</option>';
 $('#saletypes').html(html);
 });
 });
</script>

</head>

```

# Change sales.java

```
package com.virtusa.model;

public class Sales {
 private int sales;
 private int salesTypes;

 public int getSalesTypes() {
 return salesTypes;
 }

 public void setSalesTypes(int salesTypes) {
 this.salesTypes = salesTypes;
 }

 public int getSales() {
 return sales;
 }

 public void setSales(int sales) {
 this.sales = sales;
 }
}
```

# Best Practices

- Where is service ?
  - Service layer is missing. Theoretically all business logics should reside service layer
- Create service package
- Add service class
- Annotate with @service

```
package com.virtusa.service;

import org.springframework.stereotype.Service;

@Service("SalesService")
public class SalesServiceImpl {

}
```

# Now move business logic to here

- Remove business logic from salesController and move here

```
package com.virtusa.service;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Service;

import com.virtusa.model.SalesType;

@Service("SalesService")
public class SalesServiceImpl {
 public List<SalesType> getAllServiceList() {
 List<SalesType> salesTypes = new ArrayList<SalesType>();
 SalesType direct = new SalesType();
 direct.setSalesTypes("Direct");
 salesTypes.add(direct);
 SalesType web = new SalesType();
 web.setSalesTypes("Web");
 salesTypes.add(web);
 SalesType shop = new SalesType();
 shop.setSalesTypes("Direct");
 salesTypes.add(shop);
 return salesTypes;
 }
}
```



# Code to interface

- Now extract interface from service (R/C on Impl → refactor → extract interface)
- Then auto wire the controller

```
@Controller
public class SalesController {

 @Autowired
 private SalesService salesService;

 @RequestMapping(value = "/greeting")
 // to bind url to method
 public String sayGreeting(Model model) {
 model.addAttribute("greetingMsg",
 "Hello Spring MVC.. you looked handy :) ");
 return "welcome"; // jsp page name
 }

 @RequestMapping(value = "/addSales")
 public String addSales(@ModelAttribute("salesValues") Sales sales) {

 System.out.println("Sales is " + sales.getSales());
 return "addSalesValue";
 }

 @RequestMapping(value = "/salesTypes", method = RequestMethod.GET)
 public @ResponseBody
 List<SalesType> getAllSalesTypes() {

 return salesService.getAllServiceList();
 }
}
```

---

**Have Fun Life with SpRiNg 😊**