#Video1 (Why MVC...)

**Servlet** - In the java code the HTML code is embeded. is one ot the first technology providing dynamically generated content to the user. The problem was that the servlet class extending HttpServlet had to contain all the logic and provide HTML code needed to generete HTML file. That was a bit messy.

**JSP** - In the hmtl code the java code is embeded. That was an alternative to the Servlets. One of the problew was to write the whole sql sequences connecting to the DB (<sql:setDataSource var="" driver="" url="" user="" password="".>, <sql:query dataSource="" var="">SELECT \* FROM Employees</sql:query>).

None of above provided clear separation between business logic and HTML(HTML, Java, CSS, JavaScript, JQuery in one file!). **Problem to write, edit and maintain.**

**MVC is the solution**. It separates:

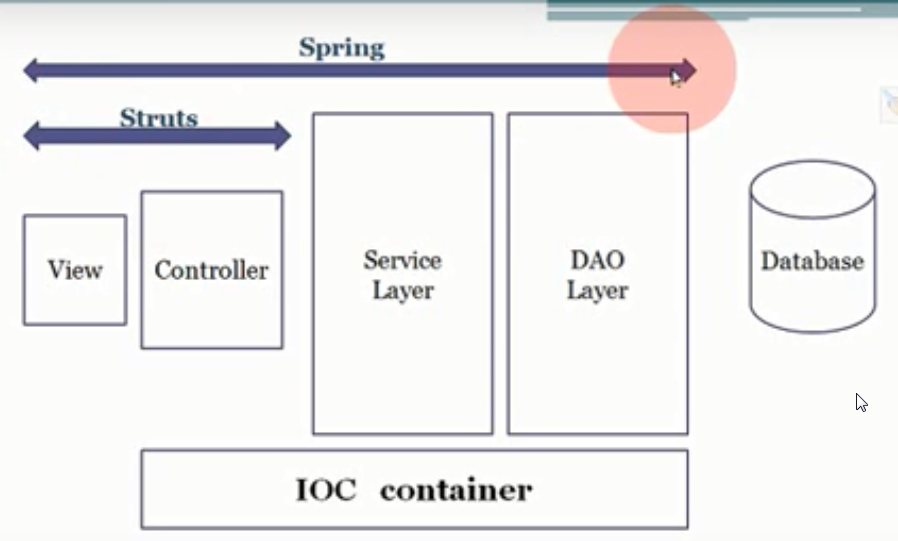
* **Model** - data to be displayed in web page: retrieving data from persistent storage
* **View** - Web page in HTML (simple JSP that contains only HTML and tag library) : HTML + CSS - presents model in user friendy interface (UI,gui)
* **Controller** - contains logic that solves domain problem

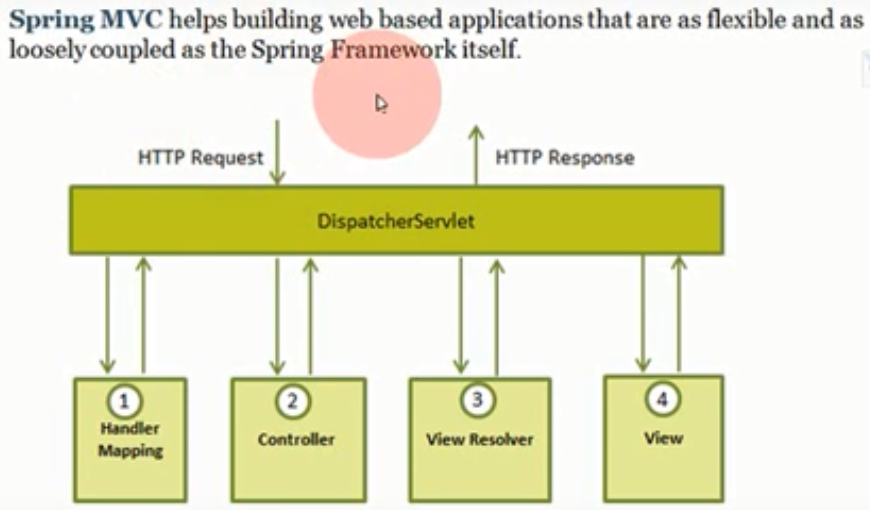
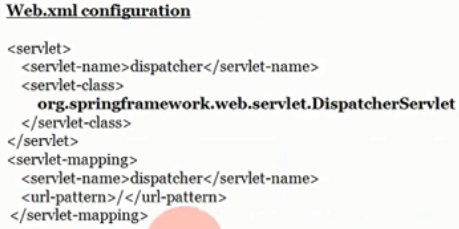
The Flow

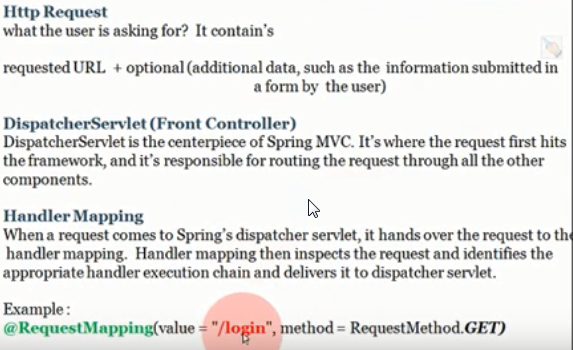
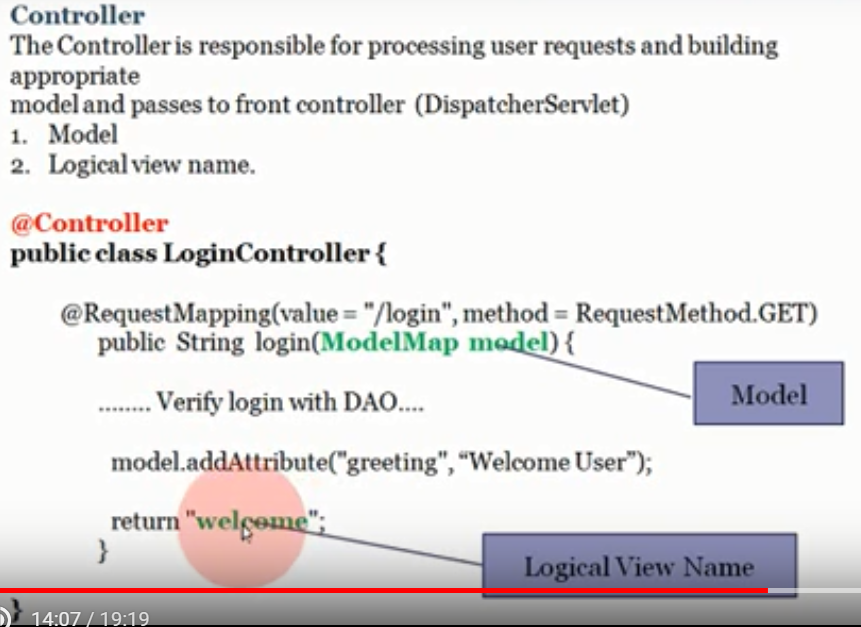
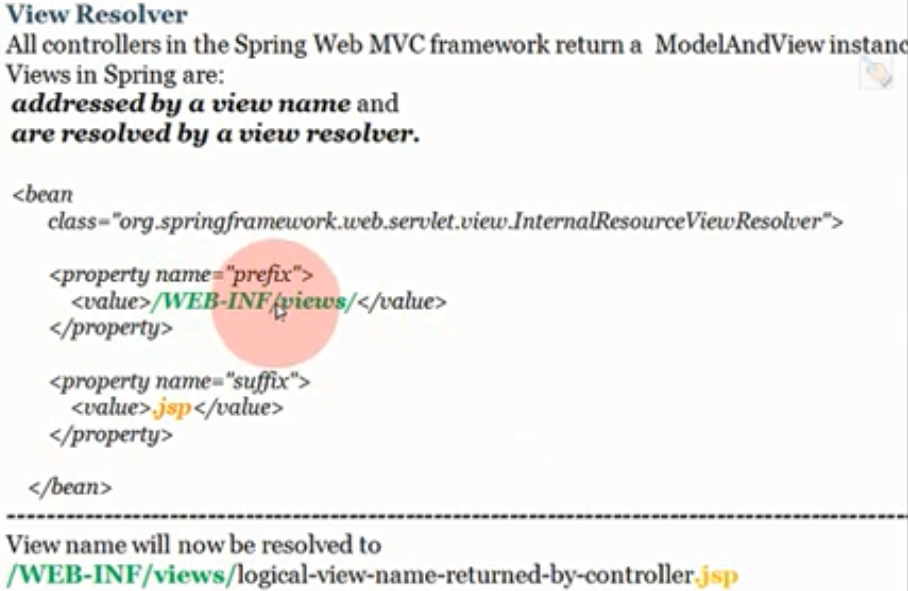
1. User request
2. Controller: prepares the model (call to model)
3. Controller: perform the logic
4. Controller: selects the view
5. Controller: sends the view to the model
6. View: renders the HTML page
7. Page is send to the client

#Video2 (Spring MVC Introduction & architecture)

**IOC Container** – Inversion of control container



**Dispatcher Servlet=Front Controller** – the center piece of the SpringMVC – it routs the information to different component. It is configured in the **Web.xml**. Below is the architecture of the spring framework.  
 

1. **Handler mapping** – selecting a controller that will handle the client request and passing it to the controller. Each request must match to unique **mapping** of a controller. 
2. **Controller** – contains add/delete/update/find. It processes the request, prepares a **model** and passes it to the front controller as well and the logical **view** name.  
   
3. **View resolver** – is a bean in the spring context configuration file. It adds prefix and suffix to build the path to the **view** (that is an instance of ModelAndView.class) and returns it to the front controller.
4. The view and the motel together can generate a HTML that is returned to the client

#video3 (Creating Environment for Spring MVC Using MAVEN)

Creating HelloWorldXml (xml in the name to indicate one of xml or java configuration)

1. Create Dynamic Web Project
   1. Configure the server (choose the server, point to the server installation directory,)
   2. Change source folder on build path to match maven structure. Remove “src” and add:
      1. “src/main/java”
      2. “src/main/resources” for static resources
      3. “src/main/webapp” for views
      4. “src/test/java”
   3. Change the content directory to “src/main/webapp” check generate web.xml.
2. Convert to maven project
   1. **GroupId** - is a base package name, where all the classes will be saved
   2. **ArtifactId** – is the project name
   3. **Version** – in the version of the project

Add dependencies to pom.xml  
 <properties>

<springframework.version>4.2.5.RELEASE</springframework.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${springframework.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>javax.servlet.jsp-api</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

#video 4 (Spring MVC Hello World Application)

1. Configure **web.xml** be defining a servlet with definition of **(1)DispatcherServlet**, indication to **(2)spring context configuration** (file location, it contains **view resolver** definition) in parameter and **(3)servlet mapping** for all strings after slash “/”.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*  xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*  id=*"WebApp\_ID"* version=*"3.1"*>  <display-name>HelloWorldXml</display-name>  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/spring-servlet.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

1. Create a Controller in a package for conrollers **org.greysalmon.controller.HelloWorldController.java**
   1. @Controller stereotype annotation
   2. First public **handler** method that returns view name as a string and model, that is the data required by the view.
   3. Second handler method with different mapping

|  |
| --- |
| package org.greysalmon.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  //stereotype annotation  @Controller  public class HelloWorldController {    @RequestMapping(value="/", method=RequestMethod.GET)  public String sayHello(ModelMap model){    // it is a map, so it takes string as a key and the value, that might be any object  model.addAttribute("message", "Welcome form spring MVC");    // this is the logical view name, that will later go through the view resolver  return "welcome";  }    // providing another mapping for unique identifying the handler method  @RequestMapping(value="/helloAgain", method=RequestMethod.GET)  public String sayHelloAgain(ModelMap model){    // it is a map, so it takes string as a key and the value, that might be any object  model.addAttribute("message", "Welcome Again form spring MVC");    // this is the logical view name, that will later go through the view resolver  return "welcome";  }  } |

1. View Resolver definition in the context configuration **Spring-servlet.xml** file
   1. Has the name defined in web.xml file
   2. Contains beans required by the IOC container (view resolver)
   3. View Resolver is a bean itself defined in it and it will be loaded into IOC Container
   4. Define annotation driven configuration
   5. Define the location scanned in order to find components(beans)

|  |
| --- |
| <?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:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.0.xsd"*>  <!-- the schema might be found in the cods in the jar/docs/spring-framework-reference/html/ -->  <!-- define location where to scan for components -->  <context:component-scan base-package=*"org.greysalmon.controller"*/>    <!-- to define that the components are annotation driven -->  <mvc:annotation-driven/>  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"*>  <value>/WEB-INF/views/</value>  </property>  <property name=*"suffix"*>  <value>.jsp</value>  </property>  </bean>  </beans> |

1. **view.jsp** HTML containing some java
   1. charset=ISO-8859-1
   2. reference to the data from model provided by the controller by ${message}

|  |
| --- |
| <%@ 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>Insert title here</title>  </head>  <body>  <h1>Hello from spring framework</h1>    <h3>${message}</h3>  </body>  </html> |

1. deploy the app on a server
   1. Download and install tomcat
   2. Choose jre location
   3. Wondows+R ->services.msc->tomcat.properties.startupType:manual
   4. run as -> run on server
   5. go to <http://localhost:8080/HelloWorldXml/> or <http://localhost:8080/HelloWorldXml/helloAgain>