Overview

@author R.L. Martinez, Ph.D.

Complexity is one of the primary challenges that has troubled software developers from the inception of the discipline. Software systems designed to perform multifaceted tasks are by nature complicated. Furthermore, software complexity continues to increase as we ask systems to accomplish more. With that in mind, designers are regularly pursuing techniques and methods to help reduce, or at least control, complexity growth. One such approach is known as MVC or, Model-View-Controller.

Model-View-Controller

MVC has its origins at Xerox's PARC (Palo Alto Research Center) during the 1970-80's. While there, Trygve Reenskaug, a visiting researcher from Norway, along with others such as Adele Goldberg, developed the MVC pattern for conceptualizing user interface based programs. PARC was an active site at that time for other important developments such as the graphical pointing device (mouse) and Object Oriented Programming (OOP). The first published work dealing with MVC at length was a 1988 article in The Journal of Object Technology.

In software development terminology, MVC is known as an architectural pattern. A pattern is a reproducible and reusable construct which appears often enough in computer science as to be recognized. Instead of solving the same problem in a variety of sub-optimal ways, a pattern attempts to overlay a design with a common architectural solution. Developers not only become familiar with its application, but can also more easily understand a solution that uses a familiar pattern. There are a number of popular web application frameworks that offer MVC architectures which include: ASP.NET, Ruby on Rails, Zend Framework, Spring-MVC, etc.

The key concepts driving MVC are easily understood. The pattern's simplicity is a compellingly favorable attribute for those considering its use. It should be noted that there are a number of interpretations of MVC which have evolved over the years. The pattern has been transformed to fulfill the needs of various applications.

For instance, while MVC was originally developed for the desktop environment, many web architectures currently utilize it. The descriptions and depictions herein

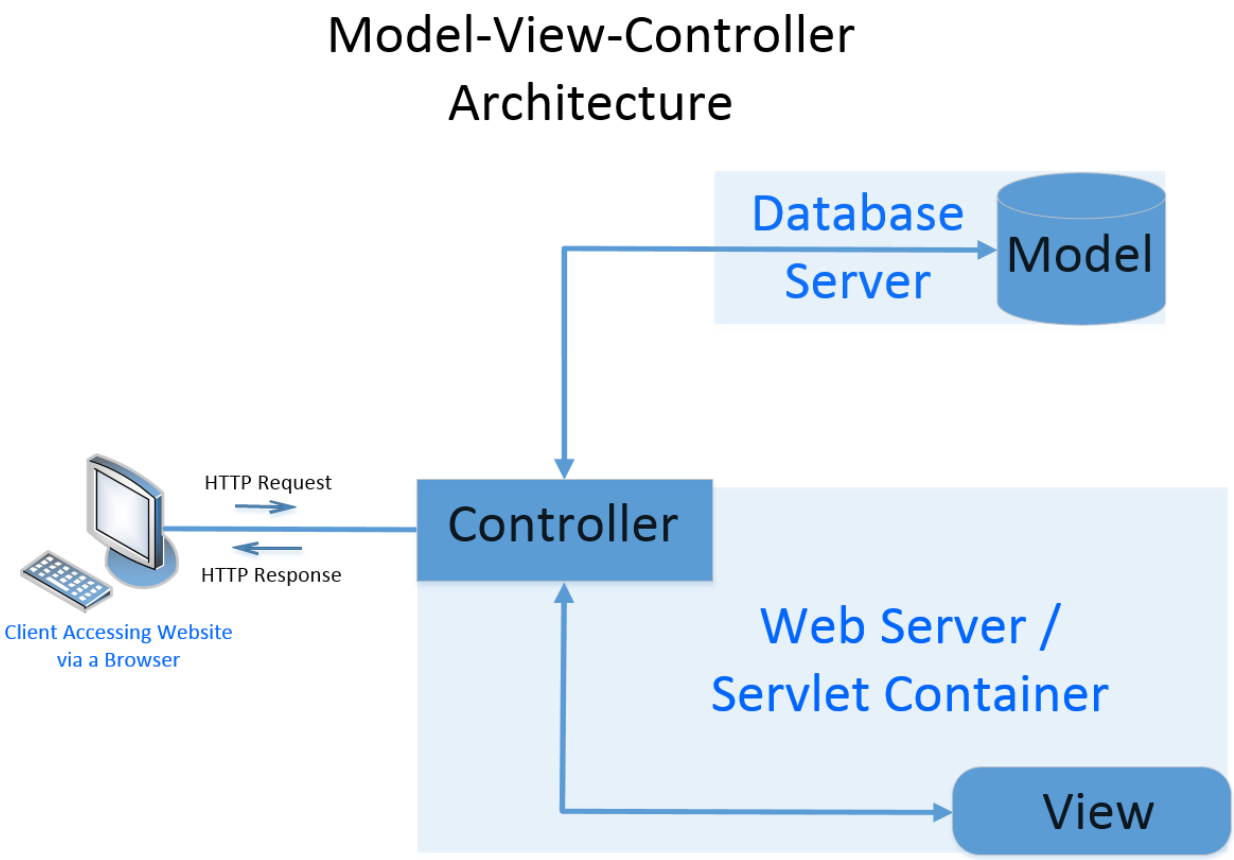
are example approaches. The pattern can be customized to fulfill the needs of the system being designed.

General MVC

Observe a few things about the pattern in the diagram. Notice that requests from a client are trafficked through the controller and on to model and/or view. An alternative view of the architecture may depict a direct connection between the model and the view. If present, that connection could represent stored procedures, functions, or triggers set in the database. Given time and breadth constraints, stored procedures, functions, and triggers are not covered in this course. More variations on the MVC theme are introduced below.

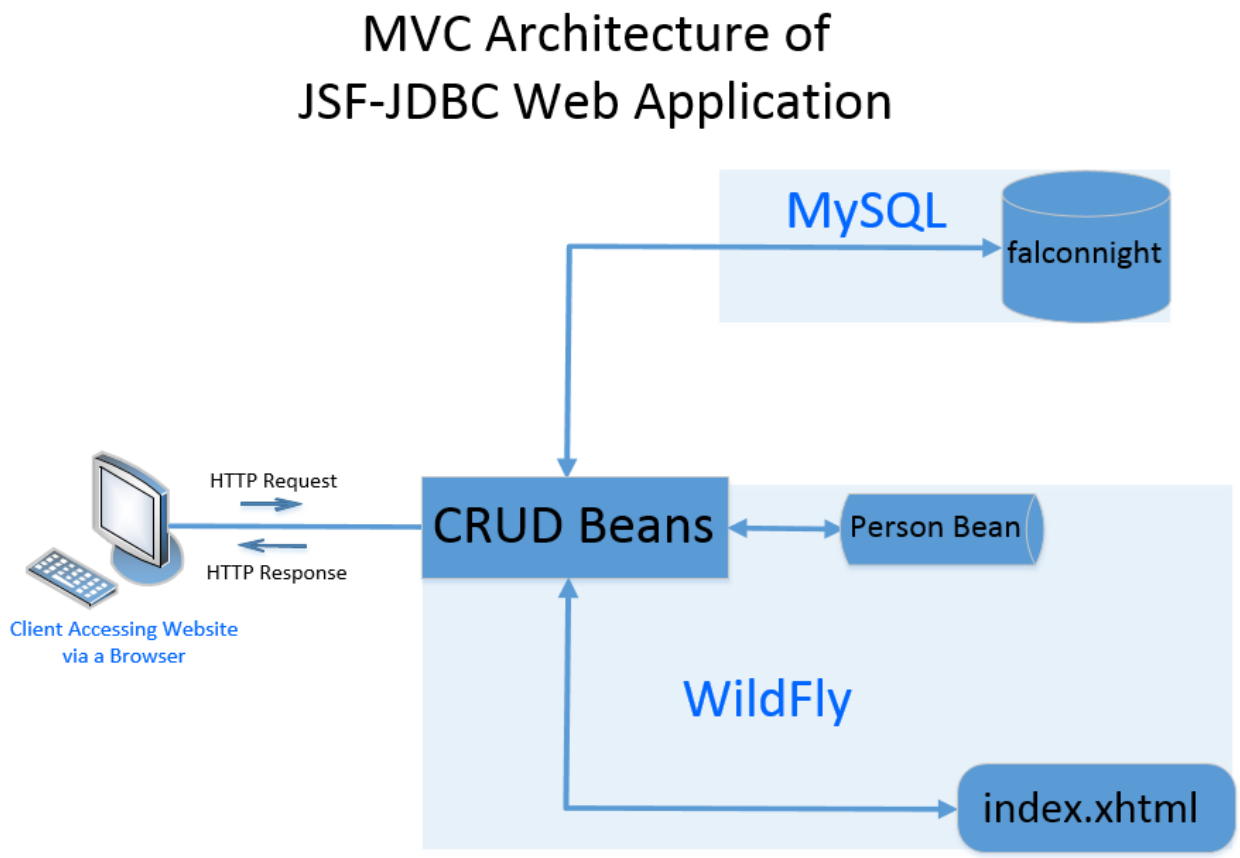
In the first diagram shown, the controller acts as the conduit and facilitator between the other components. For instance, if the browser submits an HTTP request for data from the database, the controller interfaces with the model (database), obtains a dataset, sends that data to the view which supplies the output formatted as an HTTP response.

Notice that the model typically runs on a database server. The controller and view are hosted on a web server or servlet container. In some scenarios, data can be stored on and accessed from the same server as the controller and view. This in-memory storage technique is used in both the JSF-JDBC and Spring-MVC examples in the course. In addition to volatile, in-memory storage, larger solutions almost always include persistent storage in databases.



JSF-JDBC MVC

In the previous tutorial, the JSF-JDBC application used an MVC architecture to access the database, process the view, and render output back to the browser. The diagram shows the components of the application that we coded and deployed. Recall that there was a controller bean for each of the CRUD operations. The Person bean was used to temporarily store data entered by the user or data selected from the database. Therefore, the Person bean is a form of model in MVC. This is another example of an alternative implementation of MVC architecture.



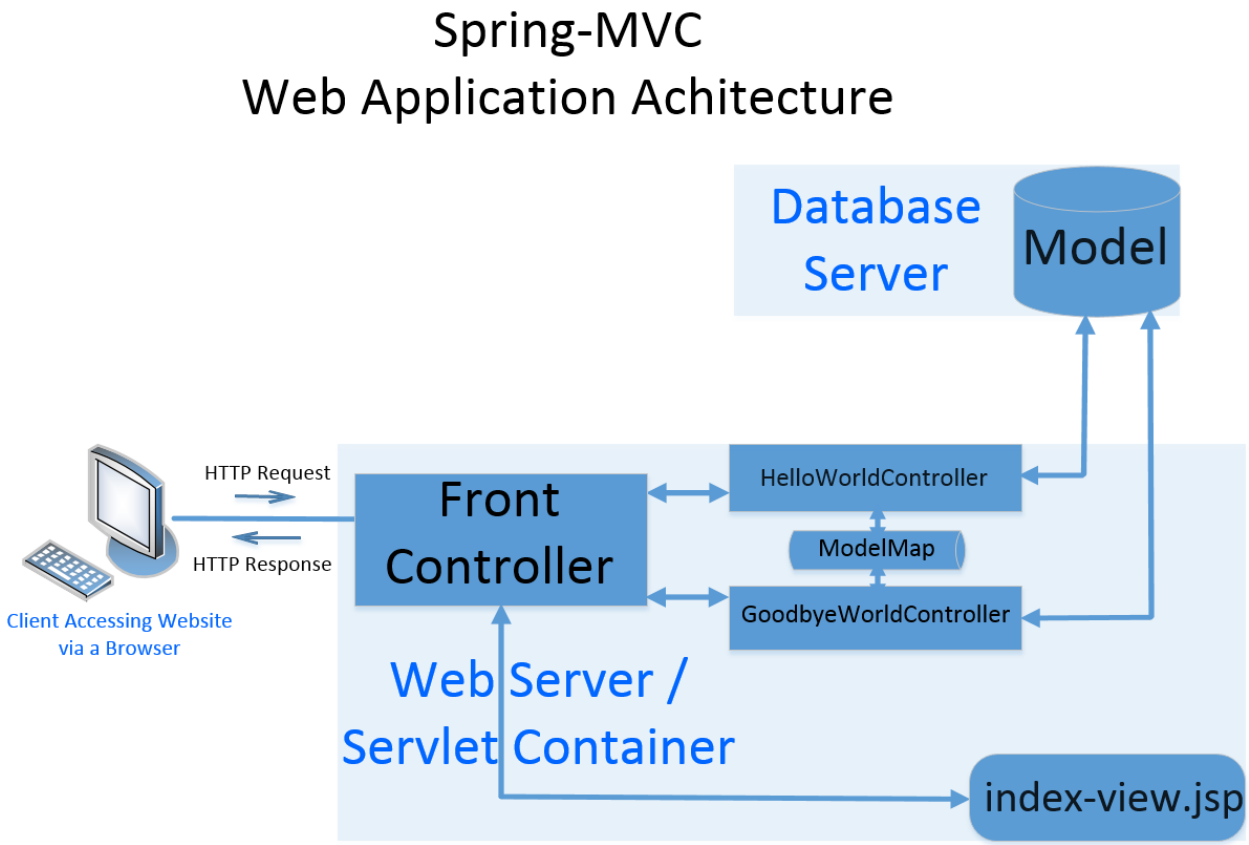
Spring-MVC

Java EE is a multifaceted and extremely powerful platform for developing enterprise and web applications. Large, multi-tiered software systems can be constructed with a set of modular components. However, as with any highly capable and intricate solution, Java EE comes at the price of complexity. Many improvements in the platform have been implemented with each new release. Most of the advances were designed not only to increase platform capabilities but to also reduce difficulty for the developer.

J2EE, an early predecessor to the current version of the enterprise platform, was particularly well known for complicated approaches. In response to the sub-optimal methods employed by J2EE, a number of frameworks were developed and introduced by members of the Java community.

As mentioned in previous tutorials, a framework provides an abstraction of its underlying target technology with the purpose of simplifying use of that

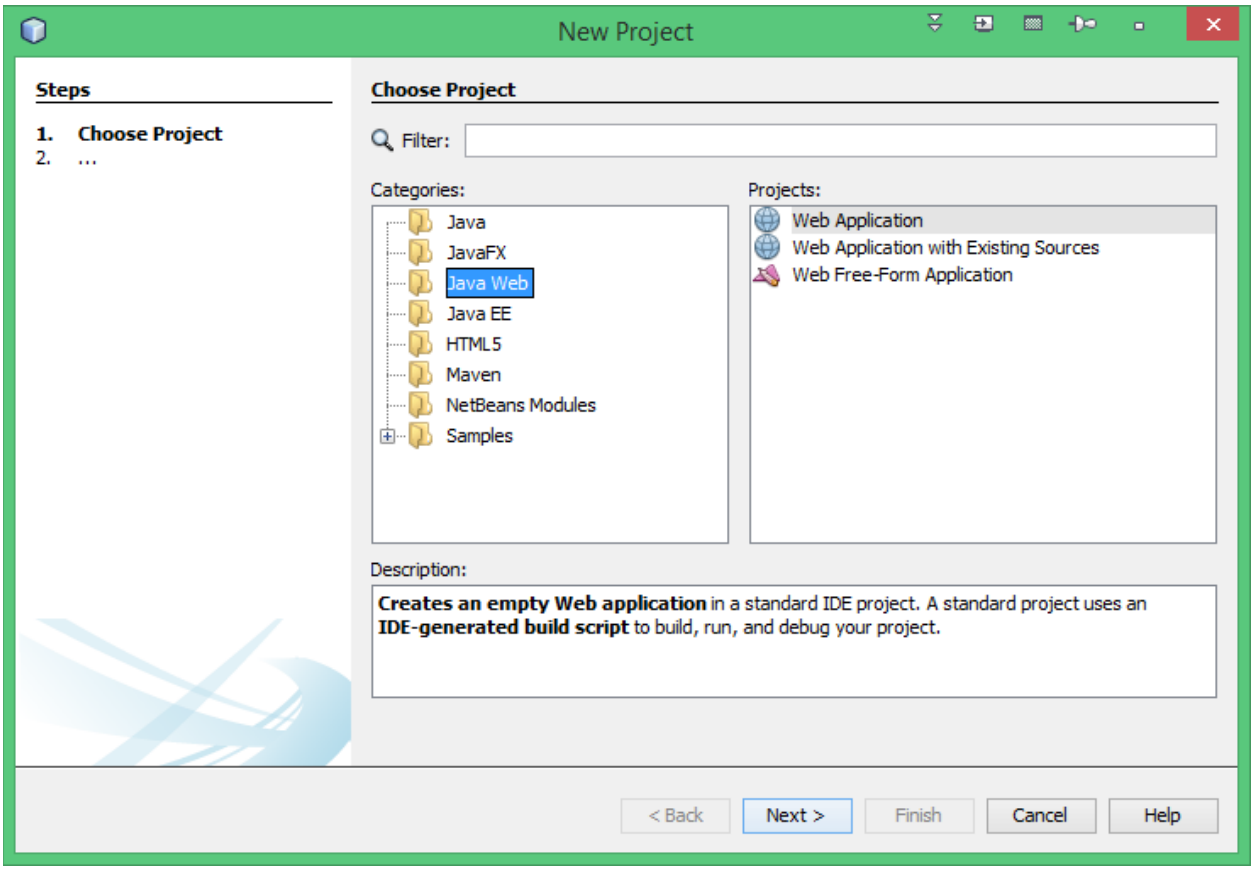
technology. The Spring framework, maintained by Pivotal Software as of 2014, is designed to make coding with Java's enterprise software easier and more efficient. Most, if not all, of the complexity issues associated with J2EE have since been addressed in the more recent versions of enterprise edition. A diagram of Spring-MVC is shown.



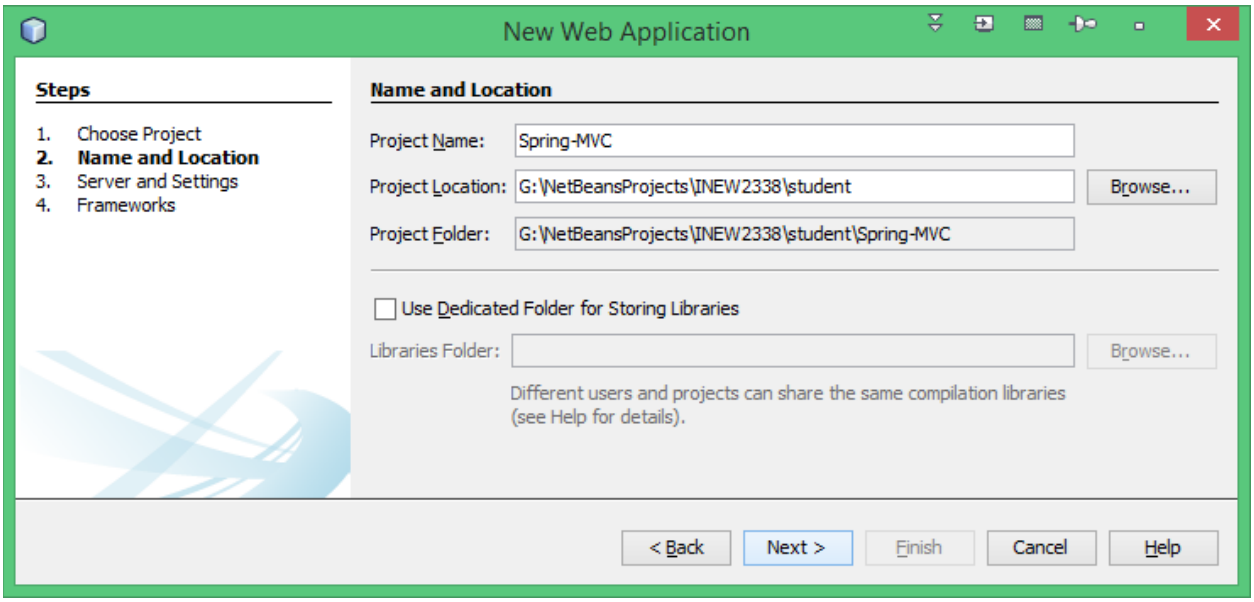
Many of the enhancements to Java EE have resulted from the positive contributions of framework developers. For instance, JSF 2.2 Faces Flow (an EE enhancement), was developed by Oracle for Java EE in response to the framework community's release of flow solutions such as ADF Task Flow, Apache MyFaces, and Spring Web Flow. Spring-MVC is a popular community framework solution that provides built-in controller flexibility.

Developing the Spring-MVC Web Application

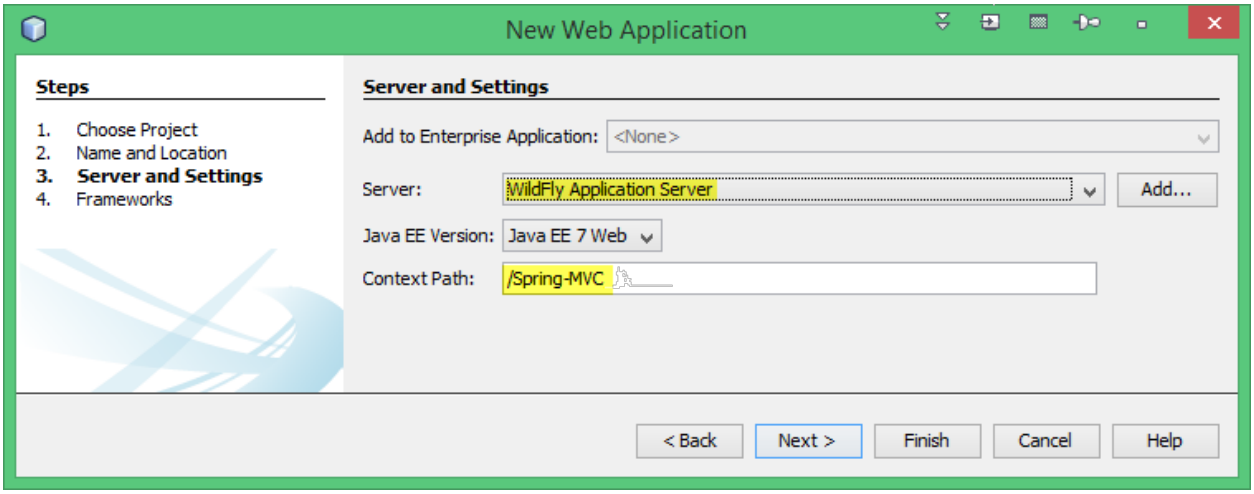
In NetBeans, create a new project and select Java Web | Web Application.



Name the project Spring-MVC and make the appropriate location and folder settings.



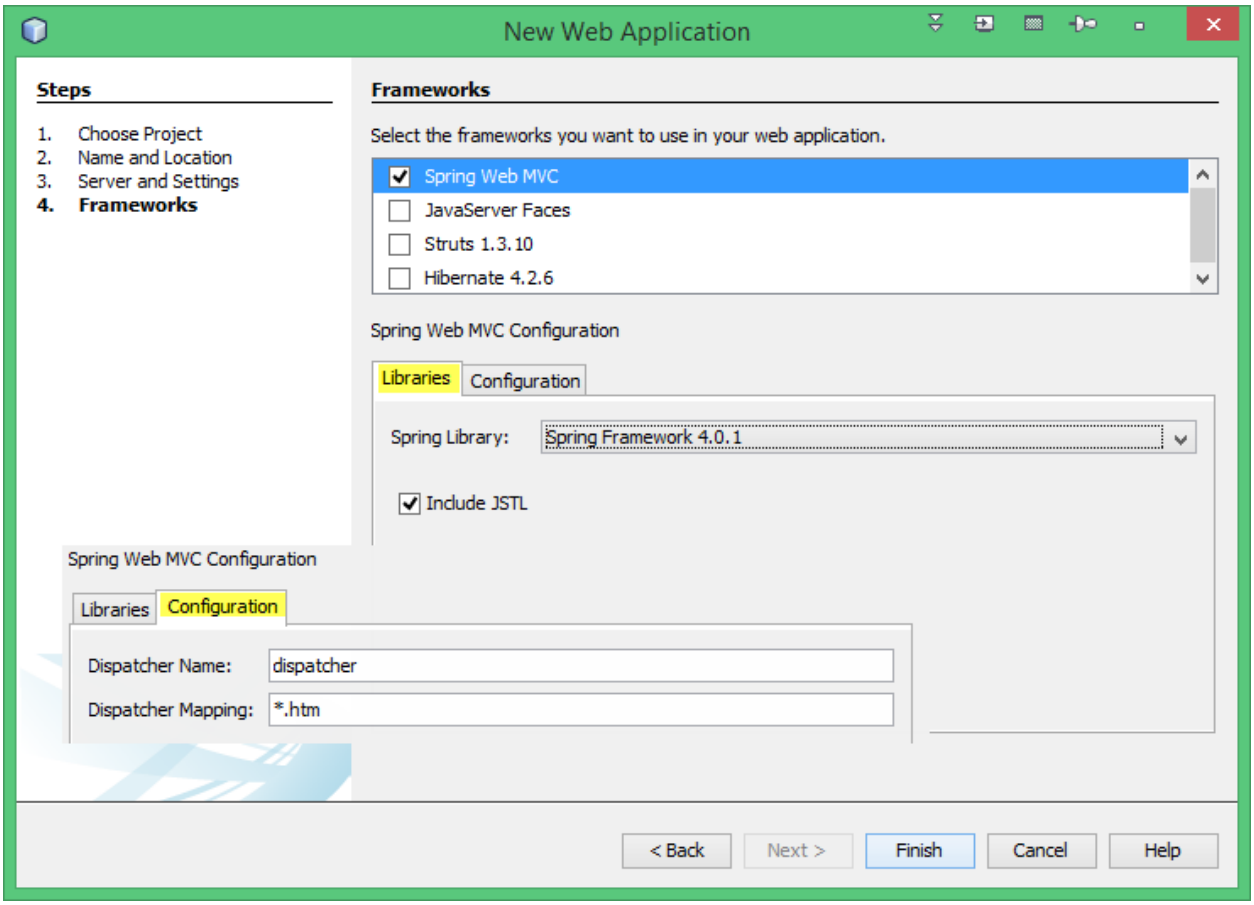
Choose WildFly Server 4 and Java EE 7. Select Next (not Finish).



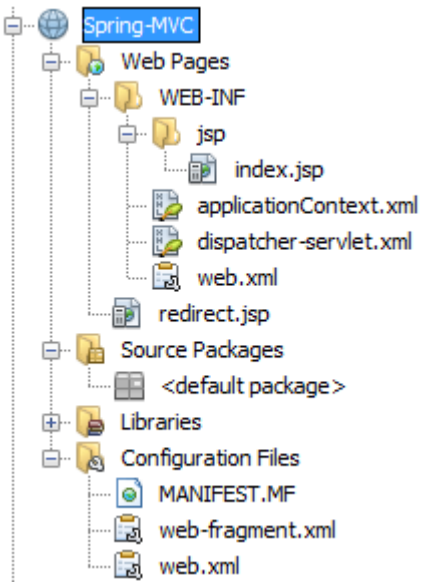
Select the Spring Web MVC framework and the latest version of Spring 4 under the Libraries tab. Retain defaults under the Configuration tab. NetBeans may present a warning message saying that Spring cannot be found while it is attempts to locate the library files.

Note: Spring Web MVC should appear in the frameworks list for the NetBeans EE installations. However, if you do not see Spring Web MVC in the list, navigate to Tools | Plugins | Available Plugins and select Spring to install the required libraries.

Select Finish.



View the project structure in the Project window.



We need to make a number of changes and additions to the project. Start by deleting redirect.jsp since it will not be used. Rename index.jsp to index-view.jsp and code the following.

```
<%-- index-view.jsp @author R.L. Martinez, Ph.D. --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link href="<c:url value="/resources/css/spring-mvc.css" />" rel="stylesheet">
    <title>Spring MVC</title>
  </head>

  <body>
    <div id='container'>
      <h1>Spring Model-View-Controller Example</h1>
      <h2>Currently in the "View"</h2>
      <h1>Message 1 : ${message1}</h1>
      <h1>Message 2 : ${message2}</h1>
      <div id="nav">
        <a id="aleft" href="http://localhost:8080/Spring-MVC/url-hello">Say Hello</a>
        <a id="aright" href="http://localhost:8080/Spring-MVC/url-goodbye">Say Goodbye</a>
      </div>
    </div>
  </body>
</html>
```

For applicationContext.xml, the comments can be removed and the format improved as follows. However, these changes are not required.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- applicationContext.xml -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
          http://www.springframework.org/schema/aop
          http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
          http://www.springframework.org/schema/tx
          http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
</beans>
```

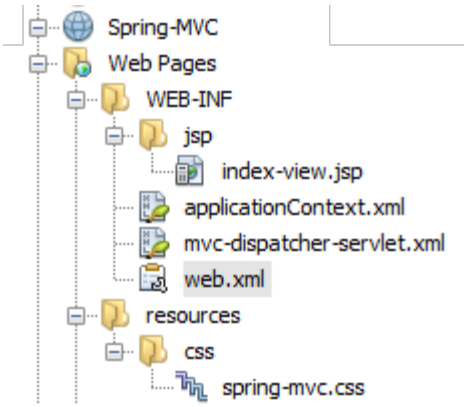
Change the name of dispatcher-servlet.xml to mvc-dispatcher-servlet.xml and code the following.

```
<!-- mvc-dispatcher-servlet.xml @author R.L. Martinez, Ph.D. -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       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">

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

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

  <mvc:resources mapping="/resources/**" location="/resources/" />
  <mvc:annotation-driven />

</beans>
```

Make the following changes to web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- web.xml -->
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

  <display-name>Spring MVC Application</display-name>

  <servlet>
    <servlet-name>mvc-dispatcher</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>mvc-dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
  </context-param>

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

  <session-config>
    <session-timeout>
      330
    </session-timeout>
  </session-config>

</web-app>
```

Add the resources and css folders as shown.



Add the file spring-mvc.css to the css folder and code it as shown.

```
/* spring-mvc.css @author R.L. Martinez, Ph.D.*/
body{
  background-color: khaki;
}
a{
  font-size: 2em;
}
a:link{
  background-color: gainsboro;
}
a:hover{
  background-color: aliceblue;
}
h1{
  text-align:center;
  color: blue;
}
h2{
  font-style: italic;
}
#container{
  display: table;
  margin: 20px auto;
  text-align:center;
}
#nav{
  width: 400px;
  margin: 30px auto;
}
#aleft{
  float: left;
}
#aright{
  float: right;
}
```

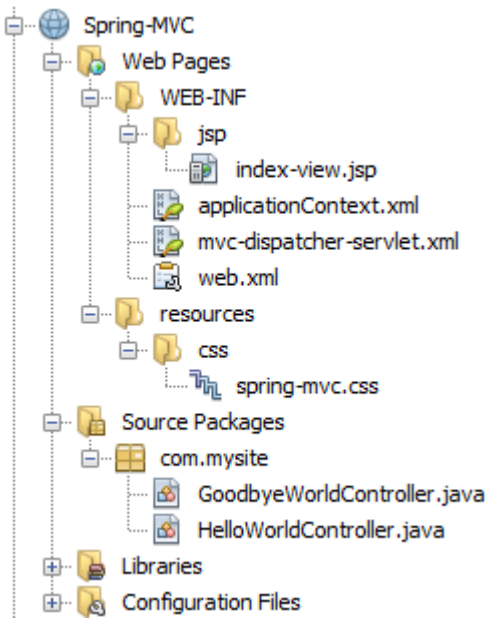
Add the HelloWorldController.java (HWC) class file to the com.mysite package by R-clicking Spring-MVC | New | Java Class. Add it to the com.mysite package. See previous tutorials for help with adding Java class files if necessary. Code the file as shown.

```
1  /* HelloWorldController.java @author R.L. Martinez, Ph.D. */
2  package com.mysite;
3
4  import org.springframework.stereotype.Controller;
5  import org.springframework.ui.ModelMap;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8
9  @Controller
10 @RequestMapping("/url-hello")
11 public class HelloWorldController {
12
13   @RequestMapping(method = RequestMethod.GET)
14   public String sayHello(ModelMap model) {
15     model.addAttribute("message1", "Hello - Stored in model via HelloWorldController.");
16     model.addAttribute("message2", "World - Stored in model via HelloWorldController.");
17     return "index-view";
18   }
19 }
```

Add the GoodbyeWorldController.java (GWC) class file to the com.mysite package by R-clicking Spring-MVC | New | Java Class. Add it to the com.mysite package. Notice the fundamental (and minor) differences between HWC and GWC are lines 10 and 14 which are discussed below.

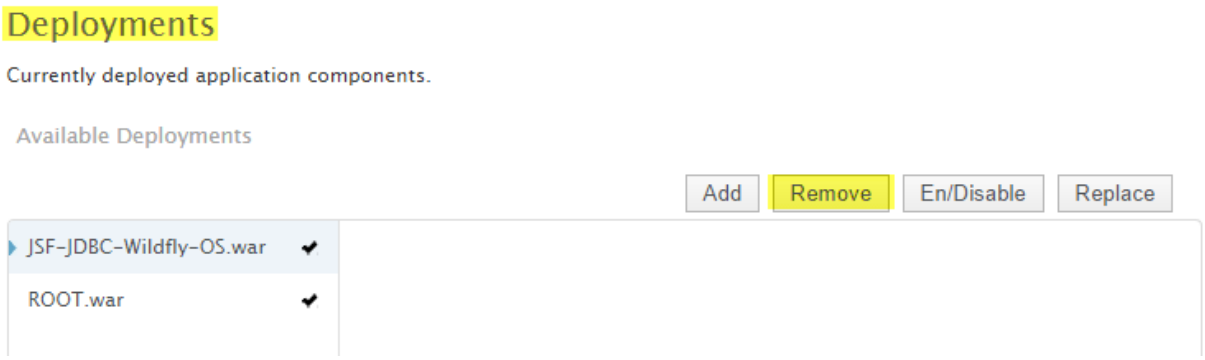
```
1  /* GoodbyeWorldController.java @author R.L. Martinez, Ph.D. */
2  package com.mysite;
3
4  import org.springframework.stereotype.Controller;
5  import org.springframework.ui.ModelMap;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8
9  @Controller
10 @RequestMapping("/url-goodbye")
11 public class GoodbyeWorldController {
12
13   @RequestMapping(method = RequestMethod.GET)
14   public String sayGoodbye(ModelMap model) {
15     model.addAttribute("message1", "Goodbye - Stored in model via GoodbyeWorldController.");
16     model.addAttribute("message2", "World - Stored in model via GoodbyeWorldController.");
17     return "index-view";
18   }
19 }
```

Your project structure should now look like the following.



Start WildFly by clicking the start button in the output window or R-clicking WildFly in the Services window and selecting Start. Monitor the WildFly startup progress in the WildFly output window. After WildFly has FULLY started, R-click Spring-MVC in the Project window and select Build to build the project.

Note: it is usually a good idea to Clean and Build a project. However, that action may return an error if attempting before any builds because NetBeans can find nothing to clean. Also, there are times when it can be helpful to undeploy (Remove) the application from within WildFly. That can be accomplished by opening WildFly Admin at localhost:9990 and selecting Runtime | Manage Deployments. Recall that WildFly Admin can also be accessed in NetBeans via Services Window | Servers | R-click WildFly | View Admin Console.



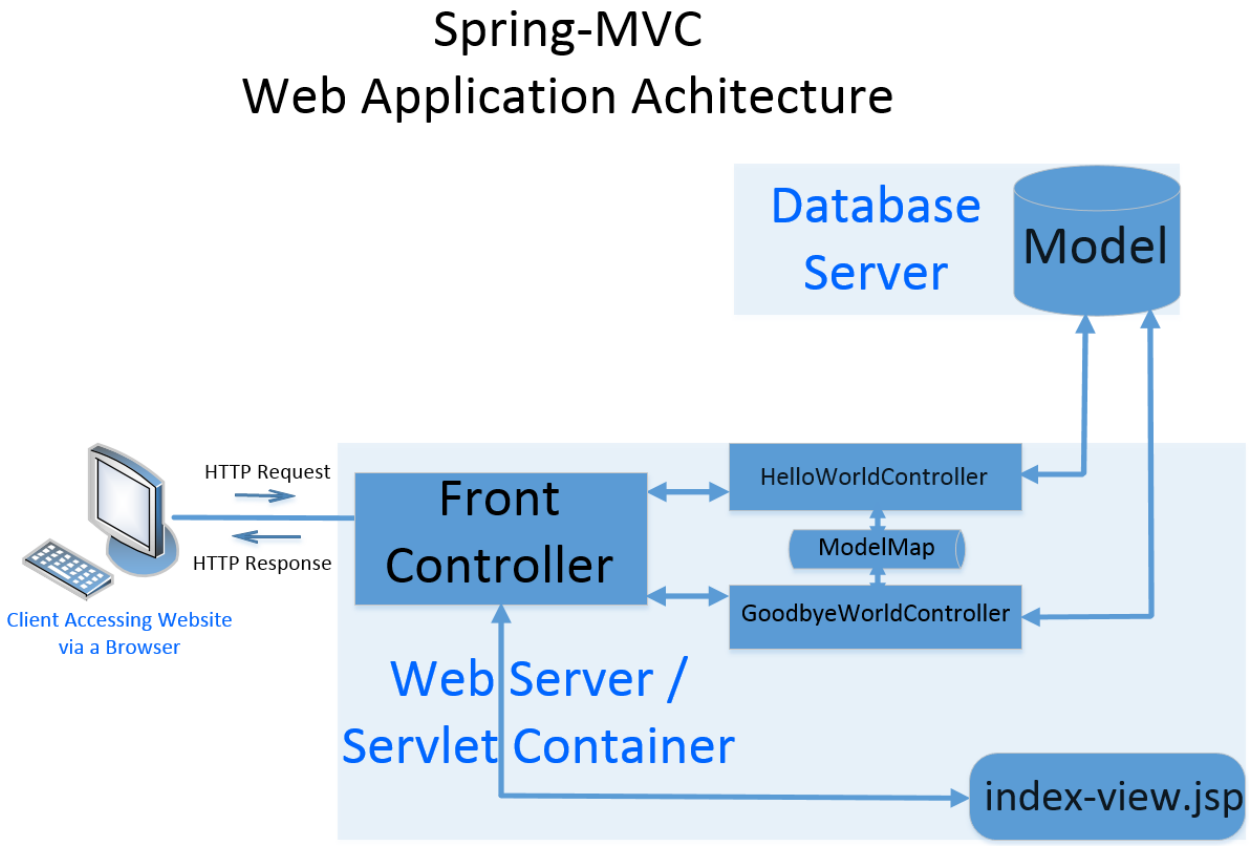
After a successful build, R-click the project in NetBeans and select Deploy. After a successful deployment, we are aready to run the application. Running the application is a little different from previous tutorials. Instead of selecting the project or file and selecting R-click | Run, we need to supply the project URL including the @RequestMapping specified in the controller. Notice that url-hello is the request mapping on line 10 of HWC. Therefore, url-hello is appended to the application name to access the application in the address box as shown.

This is the output of index-view.jsp when accessed via HWC.

This is the output of index-view.jsp when accessed via GWC.

Code Review

In addtion to the customary controller used by all MVC solutions, Spring MVC also uses an additional component known as a front controller. Notice the front controller as shown in the diagram. It can service more than one controller which we use in our example. Note: the database server and persistent storage are shown for completeness but are not implemented in the code example. The front controller routes requests to either the HelloWorldController.java (HWC) or GoodbyeWorldController.java (GWC). The particular route directed by the front controller is specified by the @RequestMapping settings in each controller. Line 10 of both HWC and GWC designate the controller mapping used to access each controller.



The front controller in our example is provided by Spring's DispatcherServlet which is specified on lines 10-21 of web.xml. On those lines the name of the dispatcher is indicated (lines 11 and 19) and the url-pattern is indentified. Since a forward slash "/" is used for the url-pattern, this means that all requests coming in will be routed to the dispatcher (front controller). If we wanted to limit the routing to only files ending with the .req extention, we could use "*.req" for the url-pattern.

Notice that line 25 in web.xml specifies the location of the dispatcher context configuration as mvc-dispatcher-servlet.xml. On line 14 of that file the context is listed as the package com.mysite which is the location of our controllers HWC and GWC. The bean on lines 16-24 in mvc-dispatcher-servlet.xml is a Spring InternalResourceViewResolver component. It specifies a view resolution prefix and suffix that will be added to a view returned by a controller.

Let's look at a view resolution example. On line 17 of HWC and GWC a view by the name "index-view' is returned. That name maps to the file index-view.jsp by lines 18-23 of mvc-dispatcher-servlet.xml. Adding the prefix and suffix as specified produces /WEB-INF/jsp/index-view.jsp.

To reiterate, the front controller is provided by Spring and is the component DispatcherServlet per line 13 of web.xml. All HTTP requests and responses are handled by the front controller determined by the forward slash "/" on line 20 of web.xml. The controllers HWC and GWC exist in the context listed on line 14 of mvc-dispatcher-servlet.xml which is com.mysite.

To complete the discussion of mvc-dispatcher-servlet.xml, the lines 9-10 and 26-27 are required so views managed by DispatcherServlet can recognize resources under the resources directory. Notice that line 8 of index-view.xml accesses the spring-mvc.css stored in the resources directory tree.

On line 9 of HWC the @Controller annotation specifies that the bean is a controller. As mentioned earlier, on line 10 the @RequestMapping is used to determine the URL appendix that will be used to access the controller. The name is arbitrary. The mapping "url-hello" is used to access HWC and "url-goodbye" is used to access GWC.

Line 13 of the controllers specifies that the controller methods sayHello and sayGoodbye will respond to GET requests types. If the method is not specified, all HTTP request types (GET, PUT, POST, etc.) will be processed.

The sayHello and sayGoodbye methods accept a ModelMap argument when called by the front controller. A Spring ModelMap data type is used to store model data when working with user interfaces. Lines 15 and 16 invoke the addAttribute method of the model to add the name of the attribute followed by its value. The controllers delegate response rendering to the front controller. The controllers also pass model data to the front controller which makes it available to the view. The view name returned to the front controller is specified on line 17 as index-view.

After receiving a delegate response from either HWC or GWC, the front controller routes the traffic to the index-view per the return statement on line 17 of both controllers. Lines 16 and 17 of index-view.jsp access the message attributes of the

model and output the messages that were set in the respective controller. Notice in the architectural diagram that the Model Map is in-memory storage managed by the servlet container.

That concludes our discussion of MVC and specific MVC implementations. To enhance understanding of the Spring-MVC example, I recommend that you review the material and practice modifications with it until the architecture and code become clear. Considerable time and effort investments will likely be required unless you have a significant amount of software development experience.