1. **Folder structure of web application:**

Web Application Directory Structure: I put MyWebProject into webapp folder in apache Tomcat.

MyWebProject/

WEB-INF/

web.xml

weblogic.xml

lib/

MyLib.jar

classes/

MyPackage/

MyServlet.class

index.html

index.jsp

In **WEB-INF** folder we put **web.xml** file that is heart of web application because all entry related to servlets and servlets related mapping we do in this web.xml file.

This is also a disadvantage of web application because if we have more servlets entry in single file web.xml then we would be difficult to read code and manage servlets. If any thing we miss web.xml(i.e any tag entry or do wrong entry ) then application will not work and we can not find easily mistake in our web application. That’s why we use MVC pattern for enterprise application. In this we can manage our code easily. And in web.xml just we put only single entry of servlet that is DispatcherServlet. We have to put DispatcherServlet entry because web container understand only servlet nothing else.

“*Spring MVC says I give you one servlet that put into web.xml but servlet entry is must”. In Spring MVC we use one servlet entry into web.xml and also create another mapping file to map urls with servlets. It reduce complexity*.

Dispatcher Servlet is called front controller.

**Point**-1:- *if we are creating multiple servlets we have to map(configure) some where(mapping file) then Spring MVC says I have a way to configure those servlets but that is Spring Specific.*

*One servlet entry (Dispatcher Servlet ) have done into web.xml and another entry have done into another mapping file. Which servlets entry we have done into another file Spring does not understand that. So Spring MVC says I have standard format for those servlets ( in spring MVC servlets means controllers). All servlets mapped into mapping file through URL pattern and put DS (Dispatcher Servlet) into web.xml. before MVC pattern if we have 100 servlets into web application we have to map all servlets into web.xml file but now in Spring MVC just single entry in web.xml file and remaining entries into another mapping file.*

*Dispatcher Servlet will handles all the requests and forward to particular controller with help of Handler mapping.*

## Required Configuration

You need to map requests that you want the *DispatcherServlet* to handle, by using a URL mapping in the **web.xml** file. The following is an example to show declaration and mapping for **HelloWeb** *DispatcherServlet* example:

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

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

<servlet>

<servlet-name>HelloWeb</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

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

</servlet>

<servlet-mapping>

<servlet-name>HelloWeb</servlet-name>

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

</servlet-mapping>

</web-app>

The **web.xml** file will be kept *WebContent/WEB-INF* directory of your web application. OK, upon initialization of **HelloWeb** *DispatcherServlet*, the framework will try to load the application context from a file named **[servlet-name]-servlet.xml** located in the application's *WebContent/WEB-INF* directory. In this case our file will be **HelloWeb-servlet.xml**.

Next, <servlet-mapping> tag indicates what URLs will be handled by the which DispatcherServlet. Here all the HTTP requests will be handled by the **HelloWeb** DispatcherServlet.

If you do not want to go with default filename as *[servlet-name]-servlet.xml* and default location as *WebContent/WEB-INF*, you can customize this file name and location by adding the servlet listener *ContextLoaderListener* in your web.xml file as follows:

<web-app...>

<!-------- *DispatcherServlet* definition goes here----->

....

<context-param>

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

<param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

1. **Apache and tomcat both are different Apache is a web server and Tomcat is web container.**

Web container is component of web server if web server get request for static page then it handles itself and provide response to client. But web server get request for servlets then web server forward that request to web container that handles servlets request.

When web server gets a request for servlet then web server it forward to web container(servlet container) then web container look into **web.xml** file for url mapping for that servlet and call that servlet. Servlet has business login( also pick data from database by using JDBC connection ) and return response.

A **Web** [**server**](http://whatis.techtarget.com/definition/server) is a program that uses [HTTP](http://searchwindevelopment.techtarget.com/definition/HTTP) (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients. Dedicated computers and appliances may be referred to as Web servers as well.

Web servers are [computers](http://www.webopedia.com/TERM/C/computer.html) that deliver (serves up) [Web pages](http://www.webopedia.com/TERM/W/web_page.html). Every Web server has an [IP address](http://www.webopedia.com/TERM/I/IP_address.html) and possibly a [domain name](http://www.webopedia.com/TERM/D/domain_name.html). For example, if you enter the [URL](http://www.webopedia.com/TERM/U/URL.html) http://www.webopedia.com/index.html in your [browser](http://www.webopedia.com/TERM/B/browser.html), this sends a request to the Web server whose domain name is webopedia.com. The server then fetches the page named index.html and sends it to your browser.

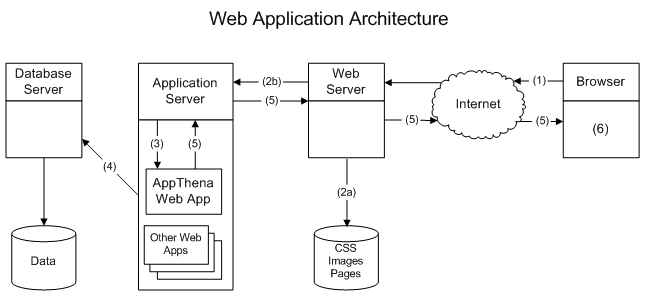
**Web container** (also known as a Servlet container) is the component of a [web server](https://en.wikipedia.org/wiki/Web_server) that interacts with [Java](https://en.wikipedia.org/wiki/Java_%28programming_language%29) [servlets](https://en.wikipedia.org/wiki/Servlet). A web container is responsible for managing the lifecycle of servlets, mapping a [URL](https://en.wikipedia.org/wiki/URL) to a particular servlet and ensuring that the URL requester has the correct access rights.[[1]](https://en.wikipedia.org/wiki/Web_container#cite_note-1)

A web container handles requests [servlets](https://en.wikipedia.org/wiki/Java_servlet), [JavaServer Pages](https://en.wikipedia.org/wiki/JavaServer_Pages) (JSP) files, and other types of files that include server-side code. The Web container creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks.

A **Web application runs within a Web container of a Web server.** The Web container provides the runtime environment through components that provide naming context and life cycle management. Some Web servers may also provide additional

1. **Architecture and Flow of web application:**

It shows you how the system handles page requests from a web browser and which components are involved. AppThena is web application:-



This diagram shows how AppThena fits within a standard enterprise web application architecture. The flow of control in the architecture is as follows:

1. The browser sends a request for a resource of some kind to the web server.
2. The web server decides what to do with the request.
   1. Static resources such as images, CSS and static web pages are read from disk and returned directly to the browser.
   2. Requests for dynamic resources such as an AppThena Edit screen are forwarded to an application server.
3. The application server passes the request to the correct web application. e.g. AppThena.
4. The web application constructs a response using data from the database server when necessary.
5. The response is passed back up the chain to the browser.
6. The browser displays the response.

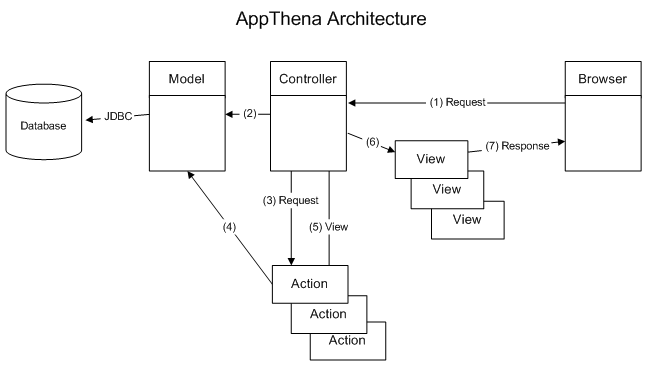
The web, application and database servers are independent pieces of software which can be installed on the same computer or on different computers. This gives systems administrators a lot of flexibility when they deploy these systems. For example, a single database server can support the web application as well as other systems.

Note that the web server and the application server are often rolled together into one program. For example, Apache Tomcat™ can host web applications and serve static content as well.

## **How AppThena Handles Page Requests**

This section shows how AppThena handles standard page requests such as a user clicking on an "Action" button in the default AppThena web application. Other scenarios, such as JSPs that request data directly from AppThena's [Model](http://www.appthena.com/developer/javadoc/com/glasspaw/appthena/model/Model.html) will be described later.

Note that the web and application servers are hidden in this illustration. It is assumed that they just pass requests and responses between the browser and AppThena without interfering in any way.



The flow of control for a typical page request is:

1. The [Controller](http://www.appthena.com/developer/javadoc/com/glasspaw/appthena/controller/Controller.html) receives a page request.
2. The Controller loads the [Model](http://www.appthena.com/developer/javadoc/com/glasspaw/appthena/model/Model.html) and starts a database transaction.
3. The Controller works out which [Action](http://www.appthena.com/developer/javadoc/com/glasspaw/appthena/controller/Action.html) should handle the request and passes it the request details.
4. The Action does its work, reading and updating the database via the Model.
5. The Action tells the Controller which object to use as the page object and which View should be used to render it.
6. The Controller passes the page object to the View which is usually a JSP.
7. The View creates an HTML document which is returned to the browser.