Web Servers

1. ***Apache Tomcat Server***

Apache Tomcat is a Java-capable HTTP server, which could execute special Java programs known as Java Servlet and Java Server Pages (JSP). Tomcat is an *open-source* project, under the "Apache Software Foundation". The mother site for Tomcat is http://tomcat.apache.org. Alternatively, you can find tomcat via the Apache mother site @ http://www.apache.org.

Tomcat was originally written by James Duncan Davison (then working in Sun), in 1998, based on an earlier Sun's server called Java Web Server (JWS). It began at version 3.0 after JSWDK 2.1 it replaced. Sun subsequently made Tomcat open-source and gave it to Apache.

The various Tomcat releases are:

Apache Tomcat version	Supported Java Versions	Servlet Spec
9.0	JDK 8 and later	4.0
8.5	7 and later	3.1
8.0	7 and later	3.1
7.0	6 and later	3.0
6.0	5 and later	2.5
5.5	1.4 and later	2.4
4.1	1.3 and later	2.3
3.3	1.1 and later	2.2

1. Tomcat 3.x (1999): RI for Servlet 2.2 and JSP 1.1.
2. Tomcat 4.x (2001): RI for Servlet 2.3 and JSP 1.2.
3. Tomcat 5.x (2002): RI for Servlet 2.4 and JSP 2.0.
4. Tomcat 6.x (2006): RI for Servlet 2.5 and JSP 2.1.
5. Tomcat 7.x (2010): RI for Servlet 3.0, JSP 2.2 and EL 2.2.
6. Tomcat 8.x (2013): RI for Servlet 3.1, JSP 2.3, EL 3.0 and Java WebSocket 1.0.

Tomcat is an HTTP application runs over TCP/IP. In other words, the Tomcat server runs on a specific TCP port in a specific IP address. The default TCP port number for HTTP protocol is 80, which is used for the *production* HTTP server. For *test* HTTP server, you can choose any unused port number between 1024 and 65535.

2. How to Install Tomcat 8 and Get Started with Java Servlet Programming

2.1 STEP 1: Download and Install Tomcat

For Windows

1. Gotohttp://tomcat.apache.org⇒ Downloads ⇒ Tomcat 8.0 ⇒ "8.0.{*xx*}" (where {*xx*} is the latest upgrade number) ⇒ Binary Distributions ⇒ Core ⇒ "**ZIP**" package (e.g., "apache-tomcat-8.0.{*xx*}.**zip**", about 8 MB).
2. Create your project directory, say "d:\myProject" or "c:\myProject". UNZIP the downloaded file into your project directory. Tomcat will be unzipped into directory "d:\myProject\apache-tomcat-8.0.{*xx*}".
3. For **ease of use**, we shall shorten and rename this directory to "d:\myProject\tomcat".

Take note of Your Tomcat Installed Directory. Hereafter, I shall refer to the Tomcat installed directory as <TOMCAT_HOME>.

Tomcat's Directories

Take a quick look at the Tomcat installed directory. It contains the following sub-directories:

- bin: contains the binaries; and startup script (startup.bat for Windows), shutdown script (shutdown.bat for Windows , and other binaries and scripts.
- conf: contains the system-wide configuration files, such as server.xml, web.xml, context.xml, and tomcat-users.xml.
- lib: contains the Tomcat's system-wide JAR files,servlepapifiles accessible by all webapps. You could also place external JAR file (such as MySQL JDBC Driver) here.
- logs: contains Tomcat's log files. You may need to check for error messages here.
- webapps: contains the webapps to be deployed. You can also place the WAR (Webapp Archive) file for deployment here.
- work: Tomcat's working directory used by JSP, for JSP-to-Servlet conversion.
- temp: Temporary files.

2.2 STEP 2: Create an Environment Variable JAVA_HOME

(For Windows)

You need to create an *environment variable* called "JAVA_HOME" and set it to your JDK installed directory.

1. First, find your JDK installed directory. The default is "c:\Program Files\Java\jdk1.8.0_{*xx*}", where {*xx*} is the upgrade number. Take note of your JDK installed directory.
2. To set the environment variable JAVA_HOME in Windows 7/8/10: Start "Control Panel" ⇒ System ⇒ Advanced system settings ⇒ Switch to "Advanced" tab ⇒ Environment Variables ⇒ System Variables ⇒ "New" ⇒ In "Variable Name", enter "JAVA_HOME" ⇒ In "Variable Value", enter your JDK installed directory as noted in Step 1.
3. To verify, **RE-START** a CMD shell (restart needed to refresh the environment) and issue:
4. **SET JAVA_HOME**

JAVA_HOME=c:\Program Files\Java\jdk1.8.0_{*xx*} <== Verify that this is YOUR JDK installed directory

2.3 STEP 3: Configure Tomcat Server

The Tomcat configuration files are located in the "conf" sub-directory of your Tomcat installed directory, e.g. "d:\myProject\tomcat\conf" (for Windows) .There are 4 configuration XML files:

1. server.xml
2. web.xml
3. context.xml
4. tomcat-users.xml

Make a BACKUP of the configuration files before you proceed.

2.4 STEP 4: Start Tomcat Server

The Tomcat's executable programs and scripts are kept in the "bin" sub-directory of the Tomcat installed directory, e.g., "d:\myProject\tomcat\bin" (for Windows).

Step 4(a) Start Server

For Windows

Launch a CMD shell. Set the current directory to "<TOMCAT_HOME>\bin", and run "startup.bat" as follows:

```
// Change the current directory to Tomcat's "bin"
// Assume that Tomcat is installed in "d:\myProject\tomcat"
d:                      // Change the current drive
cd \myProject\tomcat\bin  // Change Directory to YOUR Tomcat's "bin" directory

// Start Tomcat Server
startup
```

A *new* Tomcat console window appears.

```
xxx xx, xxxxx:xx:xx xx org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-9999"]
xxx xx, xxxxx:xx:xx xx org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
xxx xx, xxxxx:xx:xx xx org.apache.catalina.startup.Catalina start
INFO: Server startup in 2477 ms
```

Step 4(b) Start a Client to Access the Server

Start a browser (as HTTP client). Issue URL "http://localhost:9999" to access the Tomcat server's welcome page. For users on the other machines over the net, they have to use the server's IP address or DNS domain name or hostname in the format of "http://*serverHostnameOrIPAddress*:9999".

Try issuing URL http://localhost:9999/examples to view the servlet and JSP examples. Try running some of the servlet examples.

Step 4(c) Shutdown Server

For Windows

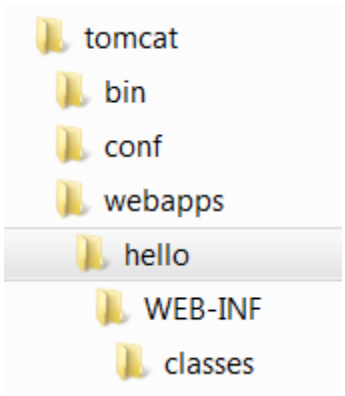
You can shutdown the tomcat server by either:

1. Press Ctrl-C on the Tomcat console; OR
2. Run "<TOMCAT_HOME>\bin**shutdown.bat**" script. Open a new "cmd" and issue:
3. // Change the current directory to Tomcat's "bin"
4. **d:** // Change the current drive
5. **cd \myProject\tomcat\bin** // Change Directory to YOUR Tomcat's "bin" directory
6.
7. // Shutdown the server
8. **shutdown**

WARNING: You MUST properly shutdown the Tomcat. DO NOT kill the cat by pushing the window's "CLOSE" button.

2.5 STEP 5: Develop and Deploy a WebApp

Step 5(a) Create the Directory Structure for your WebApp



First of all, choose a *name* for your webapp. Let's call it "hello". Goto Tomcat's "webapps" sub-directory. Create the following directory structure for you webapp "hello" (as illustrated):

1. Under Tomcat's "webapps", create your webapp*root* directory "hello" (i.e., "<TOMCAT_HOME>\webapps\hello").
2. Under "hello", create a sub-directory "WEB-INF" (case sensitive, a "dash" not an underscore) (i.e., "<TOMCAT_HOME>\webapps\hello\WEB-INF").
3. Under "WEB-INF", create a sub-sub-directory "classes" (case sensitive, plural) (i.e., "<TOMCAT_HOME>\webapps\hello\WEB-INF\classes").

You need to keep your web resources (e.g., HTMLs, CSSs, images, scripts, servlets, JSPs) in the proper directories:

- "hello": The is called the *context root* (or *document base directory*) of your webapp. You should keep all your HTML files and resources visible to the web users (e.g., HTMLs, CSSs, images, scripts, JSPs) under this *context root*.
- "hello/WEB-INF": This directory, although under the context root, is *not visible* to the web users. This is where you keep your application's web descriptor file "web.xml".
- "hello/WEB-INF/classes": This is where you keep all the Java classes such as servlet class-files.

You should RE-START your Tomcat server to pick up the hello webapp. Check the Tomcat's console to confirm that "hello" application has been properly depolyed:

.....
INFO: Deploying web application directory D:\myProject\tomcat\webapps**hello**
.....

You can issue the following URL to access the web application "hello":

http://localhost:9999/**hello**

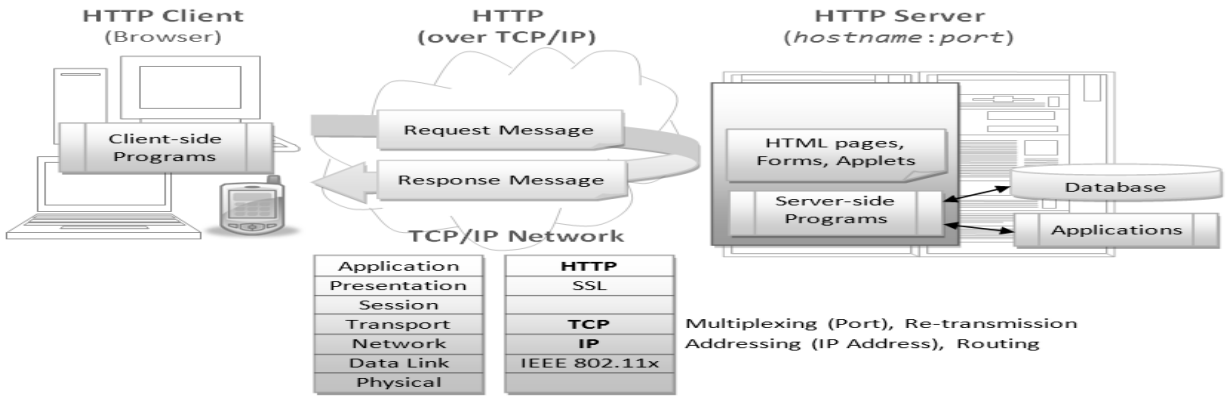
You should see the directory listing of the directory "<TOMCAT_HOME>\webapps\hello", which shall be empty (provided you have enabled directory listing in web.xml earlier).

3.Web Application and Its Architecture

A *web application* (or webapp), unlike standalone application, runs over the Internet. Examples of webapps are google, amazon, ebay, facebook and twitter.

A webapp is typically a *3-tier* (or *multi-tier*) *client-server database application* run over the Internet as illustrated in the diagram below. It comprises five components:

1. HTTP Server: E.g., Apache HTTP Server, Apache Tomcat Server, Microsoft Internet Information Server (IIS), nginx, Google Web Server (GWS), and others.
2. HTTP Client (or Web Browser): E.g., Internet Explorer (MSIE), FireFox, Chrome, Safari, and others.
3. Database: E.g., Open-source MySQL, Apache Derby, mSQL, SQLite, PostgreSQL, OpenOffice's Base; Commercial Oracle, IBM DB2, SAP SyBase, MS SQL Server, MS Access; and others.
4. Client-Side Programs: could be written in HTML Form, JavaScript, VBScript, Flash, and others.
5. Server-Side Programs: could be written in Java Servlet/JSP, ASP, PHP, Perl, Python, CGI, and others.



Web Application architecture

The typical use-case is:

1. A user, via a web browser (HTTP client), issues a URL request to an HTTP server to start a webapp.
2. A client-side program (such as an HTML form) is loaded into client's browser.
3. The user fills up the query criteria in the form.
4. The client-side program sends the query parameters to a server-side program.
5. The server-side program receives the query parameters, queries the database and returns the query result to the client.

6. The client-side program displays the query result on the browser.
7. The process repeats.

Challenges of Web Application

Usability

Web developers are often so involved in a project that they can't anticipate how real users will interact with the site. Because of this, tasks may be harder to accomplish than anticipated, and users may not be able to find the information they need.

Scalability

Scalability is neither performance nor it's about making good use of computing power and bandwidth. It's about load balancing between the servers, hence, when the load increases (i.e. more traffic on the page) additional servers can be added to balance it.

Performance

Generally, it is accepted that website speed has the major importance for a successful website. When your business is online every second counts. Slow web applications are a failure. As a result, customers abscond your website thus, damaging your revenue as well as reputation. It is said that think about performance first before developing the web application. Some of the performance issues are Poorly written code, Un-Optimized Databases, Unmanaged Growth of data, Traffic spikes, Poor load distribution, Default configuration, Troublesome third party services, etc.A content distribution network (CDN) is globally distributed network of proxy servers deployed in multiple data centers. It means instead of using a single web server for the website, use a network of servers.Some of the benefits of CDN are that the requests on the server will be routed to different servers balancing the traffic, the files are divided on different CDNs so there will be no queuing and wait for downloading different files like images, videos, text, etc.

Security

In the midst of design and user experience, web app security is often neglected. But security should be considered throughout the software development life cycle, especially when the application is dealing with the vital information such as payment details, contact information, and confidential data. There are many things to consider when it comes to web application security such as denial of service attacks, the safety of user data, database malfunctioning, unauthorized access to restricted parts of the website, etc. Some of the security threats are Cross-Site Scripting, Phishing, Cross-Site Request Forgery, Shell Injection, Session Hijacking, SQL Injection, Buffer Overflow, etc. The website should be carefully coded to be safe against these security concerns.

Alternative Explanation

Usability Challenge #1: Scalability

A contributor to Facebook's mini-feed debacle was the scale of their design. Facebook, making any change to their site, instantly affects eight million people. If even one percent has issues with the change, that's 80,000 affected users. (In 2010, that would affect 500,000 users.)

Being a social networking site compounded Facebook's issue. Users connect to other users, some users having dozens or even hundreds of connections. Those users with many connections instantly saw a very populated mini feed and realized their previously subtle interactions on their page were now broadcast to each connection.

Scale issues show up in a myriad of ways. Netflix, the online movie store, allows users to place movies they want to see into their queues. Managing a queue with 10 movies is quite simple. However, some users have hundreds of movies in their queue. Do they chunk the display of the queue into groups of 10 or 20? Do they display them all at once? How do they effectively give users control over the movies arrival order?

Many e-commerce sites give users the option of storing their shipping and billing information. What happens when users have multiple payment methods (such as a work credit card and a home credit card) or have multiple shipping addresses? For some gift sites, such as Proflowers.com, users could have many people they wish to send flowers to on a regular basis. That implies building sophisticated address book functionality into their order processing application.

Designers need to take both the scale of the user base and the scale of the data into account when thinking about how to design their web-based applications effectively.

Usability Challenge #2: Visual Design

Web apps live in this strange world, half application, half web site. Something as theoretically simple as making a command look like a command becomes difficult quickly. Do you make it a button? Do you make it a link?

Take the common practice of supplying an "**Advanced Search**" capability alongside the standard search. A typical implementation will have a text box (for entering the query), a "Search" function (for the standard search), and an "Advanced Search" function. Should the designers make both functions into buttons? Will that confuse the user? If they make "Advanced Search" a link, will users understand it's an alternative command (versus an explanation or some other site feature)?

Sometimes, in web application design, it feels like every pixel matters. This isn't just a question about the application's aesthetics. Visual design can have a huge impact on how the application communicates its use.

A user, doing their taxes or booking an airline reservation, doesn't want to think about the mechanics of interpreting the application's screen. They want to think about their deductions or their vacation destination.

Yet, if the visual design isn't clear and concise, the design takes the user's focus away from what they wish to think about and forces them to try to guess what the designers were trying to tell them.

At Lahey.org/appointments, patients can make new appointments or reschedule existing appointments with the Lahey Clinic's hundreds of doctors. The application helps keep costs down by reducing the calls coming into the clinic's offices. The app asks existing patients to enter their Lahey Clinic Number, an 8-digit number often starting with the letter "L" (such as "L1234567").

Yet, the appointment scheduling application doesn't want users to enter the "L". They only want the seven numeric digits. In informal testing, we found patients, not realizing they needed to enter only the numeric digits, would receive an obscure error message and become flustered, often resulting in having to call the clinic's offices for help.

The designers could choose one of several alternative visual treatments to solve this problem.
For example, they could start the field with an "L" or they could provide an example for users to follow. Many visual design issues, like this one, are simple to fix with a little creativity and experimentation.

Visual design problems affect an application's success in a variety of ways. In the mildest form, they slow users down and distract them from their task. In the worst cases, they confuse users to the point of giving up or needing assistance. If the application is in the organization's revenue stream or helps reduce costs, we've seen visual design issues can dramatically affect the bottom line.

Usability Challenge #3: Comprehension

Potential investors use the MSN Money Stock Research Wizard to help determine if a stock or mutual fund is the right investment for them. Because MSN Money tailored the wizard to new investors, the application contains detailed explanations, not only about the stock the investor inquired about, but about the questions the investor should be asking.

Even if the application functions properly, it will fail the user if they don't understand the information it's trying to tell them. The investor needs to both use and comprehend the wizard for it to succeed.

Web-based applications often help people by doing things outside their expertise. They turn to the application to help guide them through a decision making process they couldn't do on their own. Yet, if they make the wrong decision, it negatively affects their experience and their relationship with the organization.

Usability Challenge #4: Interactivity

One of the big differences between a web application and other types of web pages is the user is far more interactive. On a content-rich site, users mostly click links and occasionally search. Yet, in a web application, they enter data, sort it, rearrange it, and move back and forth through the screens.

Understanding how the user will manage their time becomes critical. Does the team put all the data entry on one long screen? Do they break it up across multiple screens? What is the logical order to enter the data?

Users don't always follow the "happy path." They enter data incorrectly. They decide they need to go back and change something they've already entered. They discover they need to learn more about what the application is asking of them and, thereby, need more detailed assistance.

Something as simple as always providing a mechanism to "undo" what's already been done can create interesting usability dynamics. Handling how the application deals with browser controls, such as the back button, can make the designer's life more challenging.

Usability Challenge #5: Change Management

The designers at Facebook learned the hard way that quick changes to the application, even if the team thinks it's an improvement, can have serious negative results if done incorrectly. We all know that users are resistant to change, yet designing how the change will happen is often overlooked, to serious detriment of the user experience.

While users are resistant to change, they are willing to do it when given enough support and structure. The problem with quick changes often happens when users frequently use an application and the old design conditioned them to things being a certain way. Even when the change is to their advantage, they often need warning and support to go from the old to the new.

We're now seeing teams start to design the change process along with designing the changes themselves. Paying attention to how users make the transition can increase a change's adoption and build long-term user loyalty.

Usability Challenge #6: Interoperability

Proving end-to-end functionality between communicating systems is always a challenging obstacle. Different users utilize different browsers and operating systems. To pull data, testing each one to confirm a clear information pathway is very important. Even if the browsers are similar, the web application may be rendered differently based on screen resolution and overall software configuration. This can present some serious issues for developers.

Usability Challenge #7: Security

In one of the most important tests, the developer must make sure that the continually evolving cyber threat can be countered and neutralized. Additionally, tests associated with data integrity before and after an attack are equally important when considering data breaches or lost information. Some of the challenges associated with security testing include dealing with unsecured communications, removing malicious files (if security firewalls have been breached), and the utilization (and integration) of different authentication procedures.

Usability Challenge #8:Performance

Slow applications are not successful. Developers understand that the speed of the app is defined by the need of the user, and with more users expecting more speed, the requirement of performance is non-negotiable. Testing large applications on minimal hardware, underestimating software requirements, and overextending application features are just a few of the issues associated with performance testing. Integration and interoperability issues can also have a direct effect on performance, and because of that, should be tested at the beginning.

Usability Challenge #9: Usability

Since web-based applications are dependent on different browsers, consistent usability is crucial. Additionally, since the app is the brand (or a component thereof), any inconsistency within the user experience may translate into a negative experience, affecting the brand and its potential growth. When testing usability, developers face issues with scalability and interactivity. Since every user is different, it is important for developers to utilize a representative group to test the application across different browsers, using different hardware.

Web development is accelerating at an exponential rate. Demand for better, customer centric interfaces are beginning to outpace the ability to supply them, and as more and more developers work to meet the demand for upcoming omni-channel interfaces, more obstacles appear on the horizon. Below are five challenges that developers will see in 2015. The ability for the developer/development team to meet each challenge head-on and execute accordingly may be the difference between a successful project, and a failed one.