**LESSON 1:**

1. **1-Tier Arch.** Has the UI, Business Logic(BL) and the Data Access(DAO) on a single machine(Standalone). **ADV:** Easy to design **DISADV:** Least Scalable
2. **2-Tier Arch.** Has 2 types: 1) Fat Client and Thin Server-The UI and the BL is on the client and only the DAO is on the server. **DISADV:** Time Consuming and less Processing power.

2) Thin Client and Fat Server-The UI is on the client and the BL and DAO is on the server. **DISADV:** Overload on the server which reduces the processing time.

1. **3-Tier Arch.** Has all the UI on the web client(Browser), the BL is present on the Application Server and the database servers handles data functions.  **DISADV:** Reduces Scalability and flexibility.
2. **N-Tier Arch.** Has all the UI on the web client(Browser), the BL is present on the Server that decides which servlet to be called based on the request.

**LESSON 2:**

1. A **Web Component** is a software entity that provides a response to the web request. **Ex:** Servlet and JSP

**Java Container** provides a runtime environment to the web components. They are of 2 types: **Web Containers:** Used to test,deploy and host components **Ex:** Apache’s TOMCAT

**Application Containers:** To provide a server support **Ex:** RedHat’s Wilfly

1. **Hibernate(ORM)** ie Object Relational Mapping is used to map the object oriented models to the traditional relational databases.
2. **Java Connector Architecture(JCA)** is a std. arch. For connecting the JEE platform to heterogeneous enterprise Information System.
3. **Standalone Directories: i)Configuration-** Configuration files for the standalone server. **ii) Data-** Persistent data written by the server which is required by it during runtime. **iii)Deployment-** End user’s data that need to be deployed in the server during runtime. **iv) lib\ext-** Location for the installed libraries and jar files. **v) Log-** Server’s log file. **vi) tmp-** location for the temporary files written by the server. **vii) tmp/auth-** Location to exchange authentication tokens to confirm that the client is to local.
4. **Standalone.xml-**Web Profile certified Config with required technologies, **Standalone-ha.xml-**Web Profile certified Config with High Availability, **Standalone-full.xml-** Full Profile certified Config with All required technologies, **Standalone-full-ha.xml-**FullProfile certified Config with High Availability, **domain.xml-**Java Enterprise Edition 7 full and web profiles available with or without high availability. **NOTE:** Important to note is that the domain and standalone modes determine how the servers are managed not what capabilities they provide.

**LESSON 3:**

1. **Servlets** extend the functionality of a web server and generates dynamic response for the request using HTTP Request and Response Paradigm.
2. Servlets are **Platform and Server independent.**
3. Servlets can do the following: i)Process the information provided by the client, ii)Provide authentication and security, iii) interact with db, file, other app or n/w iv) Manage state info. And can attach design elements on pages returned by the server v) Dynamically build/ generate and return the response file.
4. Servlets can read implicit data sent by the browser in the header, as well as the explicit data sent by the client.
5. **Advantages of servlet:** Compiled, Crash Resistant, Cross Platform and Server, Durable, Dynamically Loaded, Multithreaded, Protocol independent, Secure and Exyensible.
6. HTTP request contains request Method, URL, Header and body(Optional).
7. HTTP response contains response code, textual info w.r.t code, Header and body.
8. The Servlet Interface has 3 lifecycle methods init(),service(),destroy() which are implemented by the GenericServlet Class..
9. getServletConfig() is handled by the GenericServlet
10. The service() needs to be thread safe as multiple threads may call the service method simultaneously.
11. The init() is invoked only once(when the servlet is loaded in the WC), service() is called for every process, destroy() is called to remove the servlet or when the server is shut down.
12. The Servlet container is responsible for servlet’s creating, execution and destruction.
13. **GenericServlet** Class implements the methods of Servlet class, ServletConfig interface and log() of ServletContext Interface.
14. ServletContext defines methods that a servlet uses to communicate with the Container.
15. ServletConfig defines methods to pass info to the servlet during initialization.
16. All of its initialization parameters can be set in deployment descriptor, or else they can be passed via annotations.
17. **Elements of a Web Application**: i) A WAR file is really just a JAR file with a .war extension. We can use the jar command to create it. **Ex:** **jar cvf myWebApp.war \*** ii)Servlet iii)JSP iv)Utility Class v)Static Documets vi) Applets, beans, classes vii) Descriptive meta info that binds all.
18. A web application is rooted at a specific path within a web server called **context root.**
19. **WEB-INF** directory contains all things related to the application that are **not in the context root** of the application.

**Note:** The WEB-INF node is not part of the public document tree of the application. No file contained in the WEB-INF directory may be served directly to a client.

1. **web.xml** It is a deployment descriptor. The Java EE server reads the **deployment descriptor**(web.xml) at run time and acts upon the component accordingly.
2. Configuration and deployment information exist in **web.xml:**  1)ServletContext Init Parameters 2)Session Configuration 3)Servlet / JSP Definitions 4)Servlet / JSP Mappings 5)Mime Type Mappings 6)Welcome File list 7)Error Pages 8)Security
3. **classes directory** It is for servlet and utility classes. The classes in this directory are used by the application class loader to load classes from.
4. **Annotations:** Meta Data for the code which doesn’t contain any BL. They specify a std. way of defining a metadata in the code.
5. @WebServlet(name=”any name”(Optional),urlPattern={“/anyURL”} (Mandatory), loadOnStartup=1(Optional))//loadOnStartup ke liye init() ya constructor me jo karwana ho likna.
6. @WebListener is used to get the notifications if any changes made to the ServletContext. ServletContextListener is used to set the context Parameters.
7. A Listener is a class that implements servletContextListener Interface and implements the life cycle methods of ServleContext: contextInitialized(), contextDestroyed()

**Note:** We are using the annotation @WebListener above the Listener class name that would instruct the container to create the ServletContextListener and invoked it when a ServletContext object is created by Container

1. **To obtain server details:** req. getServerName(), req. getServerPort(),req. getAttribute(“name”), context.getServerInfo()
2. **To Obtain Client details:** req.getRemoteAddr(), req.getRemoteHost()
3. **To Obtain User details:** req.getRemoteUser();//Returns login if authenticated user else null.

**LESSON 4:**

1. **Get :** Appended in the URL, Not Secured, Limitation of data that could be sent.
2. **Post:** Appended in the request body, Secured, Unlimited data can be sent.
3. The container handovers the HTTPRequest and response objects to the service() of the HTTPServlet class by downcasting the ServletRequest to the HTTPServletRequest , and ServletResponse to the HTTPServletResponse.
4. When a browser makes a request it also sends HTTP Information in the form of request headers **Ex: Accept:** MIME type accepted by the client, **Accept-Language:** Lang. that client can receive, **Accept-Encoding:** encoding format that client can use, **User-Agent:** It gives information about client software etc.
5. **Header Access Methods: getHeader(String name):** It returns the value of the header as a String, **getDateHeader(String name):** It returns the value of the specified request header as a long value that represents a Date object, **getIntHeader(String name):** It returns the value of the specified request header as an int, **getHeaderNames( ):** It returns an enumeration of all the header names this request contains.

**LESSON 5:**

1. A session starts with either i) The first connection of the user, ii) Log-in of the user. The session ends with i) Last Connection ii) Log-out iii) Session Time-out due to inactivity.
2. A session is both the application and individual user specific.
3. Session Tracking is maintaining the client’s info on server across multiple client req dusring a session.
4. **Process:** The Client makes 1st request->The Server sends Session Id along with the response ->Next time when client requests the same server, the browser sends the session ID along with it to be easily identified by the server.
5. **Session Tracking:** i) **Hidden Fields-** Info. Sent using hidden fields. **DISADV:**Works only for a sequence of dynamically generated forms. ii) **URL Rewriting-** When the user clicks a URL it dynamically gets modified by either adding an extra path info., Adding parameter,custom or server specific URL. EX:Original URL : <http://server:port/servlet/Rewritten> Extra path information : <http://server:port/servlet/Rewritten/123> Added parameter : <http://server:port/servlet/Rewritten?sessionid=123> Added parameter change : <http://server:port/servlet/Rewritten;$sessionid$123> **DISADV:** The info. On the URL can be limited.

iii) **Cookies:** It isa small text file containing piece of client infowritten by the server on the browser. A Web Server can have 20 cookies, 300 cookies total, and the size of each cookie is 4kb. **EX:**Cookie(String Name,String Values) -> response.addCookie(Cookie cookie) -> request.getCookies().

And **optional attributes** such as a comment, path and domain qualifiers, a maximum age, and a version number.

**NOTE:** Do not use whitespace or any of these characters: [ ] ( ) = , " / ? @ : ;

Cookie[] cookies = req.getCookies();

if (cookies != null) {

for (int i = 0; i < cookies.length; i++) {

if (cookies[i].getName().equals("sessionid")) {

String name =cookies[i].getName();

String value =cookies[i].getValue();

}

}

}