PROBLEM STATEMENT :

A Pizza shop chain wants to automate dishes served with schemes or without scheme and delivered by the nearest shop in a chain. Use pervasive computing paradime to develop a web-application using Embedded Java/ Pythone/ Scala so that the order be delivered to the customer within 10 minutes. Use XML/JSON to store the data.

OBJECTIVE :

• To develop problem solving abilities for smart devices.
• To develop problem solving abilities for gamifications.
• To develop problem solving abilities of pervasiveness,embedded security and NLP.
• To study algorithmic examples in distributed, concurrent and parallel environments

THEORY :

Web Application :

In computing, a web application or web app is a client-server software application in which the client (or user interface) runs in a web browser. Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, online auctions, wikis and many other functions.

Web Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to an n-tiered approach by nature. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called *presentation*, *application* and *storage*, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, CGI, ColdFusion, Dart, JSP/Java, Node.js, PHP, Python or Ruby on Rails) is the middle tier (application logic), and a database is the third tier (storage).The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

JSP (JavaServer Pages **) And Servelet :**

JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications**.**Java Server Page (JSP) is a technology for controlling the content or appearance of Web pages through the use of servlets, small programs that are specified in the Web page and run on the Web server to modify the Web page before it is sent to the user who requested it.

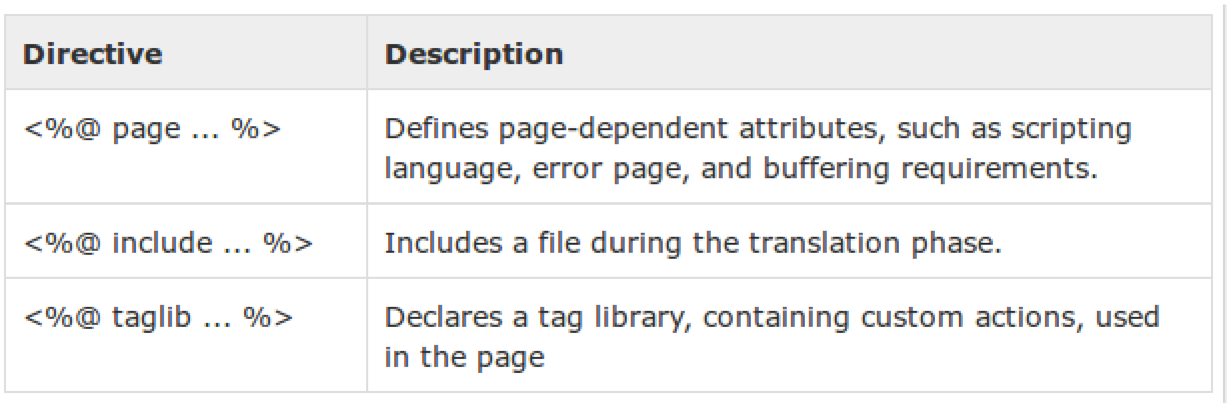
Generating dynamic content in Web applications is important when the content must reflect the most current and available data and personalized information. One of the main advantages of JavaServer Pages is the ability to generate dynamic content .

JSP DIRECTIVES :

JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing. A JSP directive affects the overall structure of the servlet class. It usually has the following form :

<%@ directive attribute="value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.



Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

page Directive:

The **page** directive is used to provide instructions to the container that pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.Following is the basic syntax of page directive:

<%@ page attribute="value" %>

include Directive:

The **include** directive is used to includes a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page.The general usage form of this directive is as follows:

<%@ include file="relative url" >

taglib Directive:

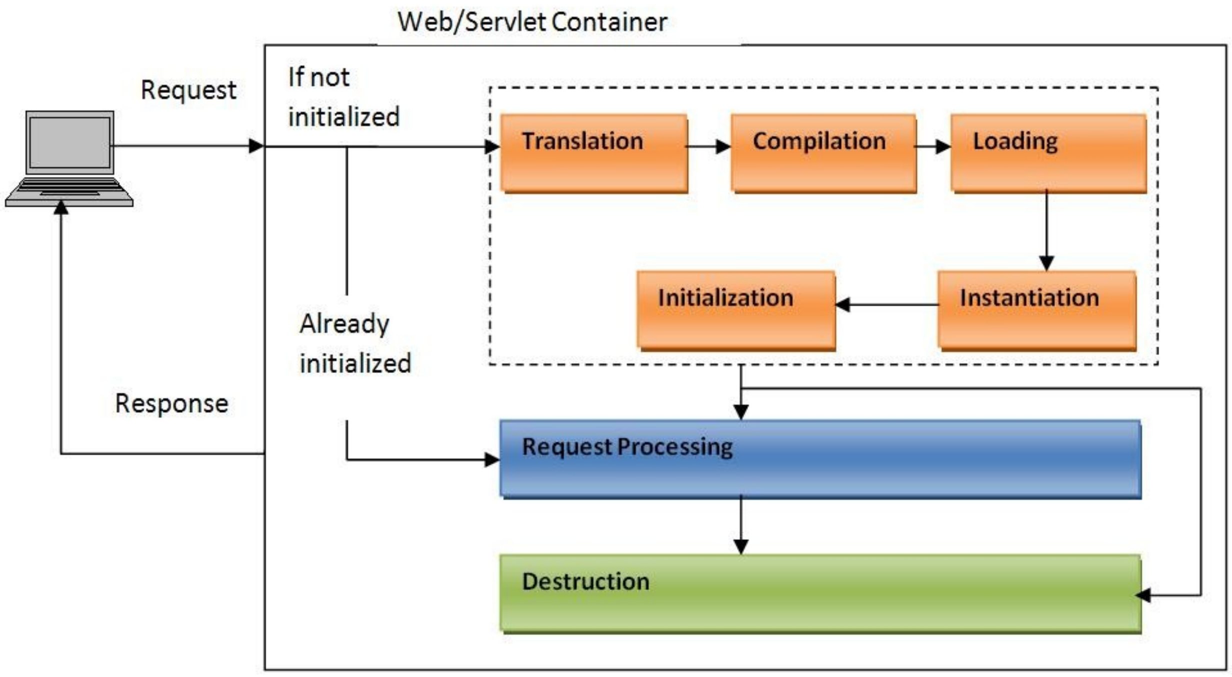
The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.The taglib directive follows the following syntax:

<%@ taglib uri="uri" prefix="prefixOfTag" >

JAVA SERVLET :

Servlets are programs that run on a Web or application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.Many client requests can be satisfied by prebuilt documents, and the server would handle these requests without invoking servlets. In many cases, however, a static result is not sufficient, and a page needs to be generated for each request.
A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods.

LIFE CYCLE OF A SERVLET :



1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet.The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.After the destroy() method is called, the servlet object is marked for garbage collection.

ALGORITHM :

1. Put all the pizza shop in the XML file .
2. Open Pizza order WebPage invoking the WebApp .
3. Enter city and corresponding area .
4. Check whether the pizza shop can deliver the order .
5. Depending upon the availability give appropriate response .

INPUT :

Enter area and city for ordering pizza .

EXPECTED OUTPUT :

Appropriate response whether pizza can be delivered within ten minutes or not .

MATHEMATICAL MODEL :

Let P be the solution perspective .

P ={ S , E , I , O, F }

S = { Initial state of the Webapp showing pizza order WebPage (UserInterface). }

I = Input of the system → { I1 , I2 }
where I1 = { Enter city }
I2 = { Enter area }

O = Output of the system → { O1 }
where O1 = { Appropriate response whether pizza can be delivered or not }

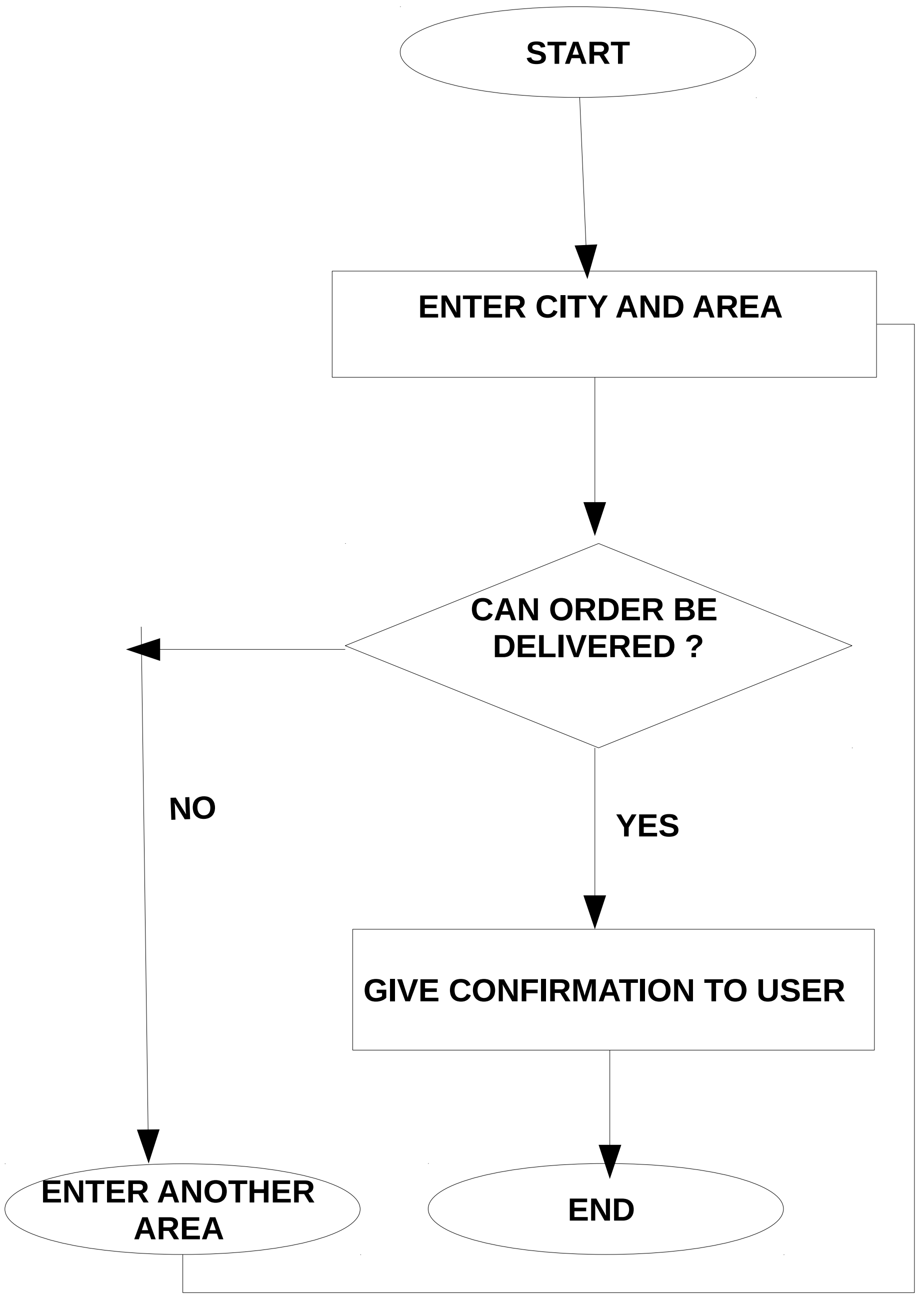
F = Functions used → { f1 , f2 , f3 }
where f1 = { Extracting pizza shop information from Xml File }
f2 = { Input area and city }
f3 = { Calculating whether pizza can be delivered within time limit }

E = End state of the system which tells the user whether his/her pizza order can be delivered to his area within ten minutes .

TEST CASES :

TEST CASE	INPUT	EXPECTED OUTPUT	OUTPUT ACHIEVED	REMARKS
1	CITY , AREA	ORDER DELIVERABLE	ORDER WILL BE DELIVERED WITHIN 10 MINUTES	Correct
2	CITY , AREA	ORDER CANNOT BE DELIVERED	ORDER CANNOT BE DELIVERED TO YPUR AREA WITHIN TEN MINUTES	Correct

FLOWCHART :



CONCLUSION :

Hence we have successfully implemented a JSP Servelet based WebApp which delivers pizza to the user from the nearest shop available .

OUTCOMES ACHIEVED :

COURSE OUTCOME	**ACHIEVED(√)**
Problem solving abilities for smart devices.	
Problem solving abilities for gamifications.	
Problem solving abilities of pervasiveness,embedded security and NLP.	
To solve problems for multicore or distributed,concurrent/Parallel environments	√