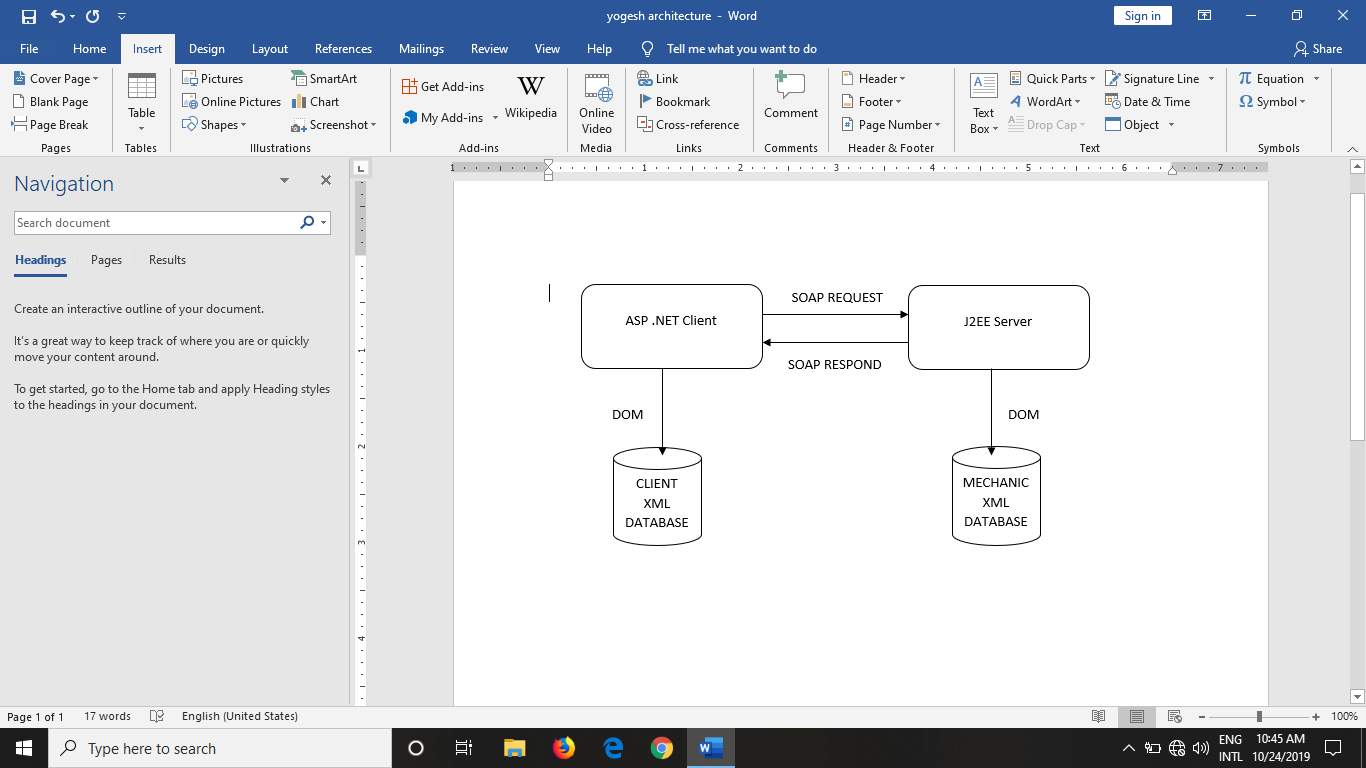
**EXP NO: 1 DATE:**

**LOCALIZED CAR MECHANIC FINDER USING JAX-WS WITH J2EE AND ASP.NET**

**ABSTRACT**

In our day to day life everything is made online and makes the people to use it to ease and save their time. This project focuses on the major problem which is faced by people that is to get a car repaired. It’s a herculean task but when it is in our hand it will make it easy. Our project abbreviated as LCMF, it aims to provide a localized car mechanical shop from your area where your car has stopped abruptly or you can abide the services from your home. Other than this the mechanic spare parts are also available to both users and mechanics. In this the online localized car-mechanical shop finder will be using jax-rs which uses REST architecture and the second part, the shopping module will be using jax-ws.

**SYSTEM ARCHITECTURE**



**TECHNOLOGIES USED**

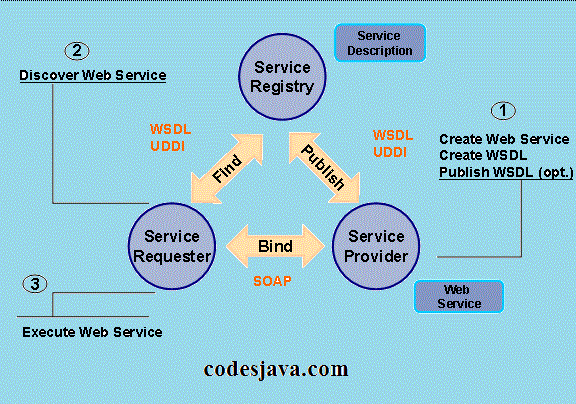
**JAVA WEBSERVICES**

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message. There are three aspects of web services.

They are

1. RPC style
2. Document style
3. Resteasy and jersey

**WEB SEVICE ARCHITECTURE DIAGRAM**



The Web Services architecture is built upon a three-role interaction:

**Service Provider**:

This is the service’s owner from the business perspective. From the architectural approach, this is the platform that is accessed in the service request. It is also the entity that creates the Web Service, being responsible to make its description in some standard format and publish its details in a central registry.

**Service Requestor**:

It is an application that invokes or initializes some interaction with the service. It could be a web browser or even a non-user interface program such as another Web Service. By using the service description it’s possible to discover and invoke Web Services.

.**Services Registry**:

It’s the place where service providers publishes their service descriptions. Service Requestors search the Registries, fetching binding and description information both during the development time (static bindings) or run time (dynamic bindings).

As mentioned early, there is the Service Description whose contents describes interface and implementation details, including data structures, operations and network binding information. Also it contains data to easy Service Requestors’ searching process.

The Service is the software deployed through the network by the Service Provider. Afterwards, there are common operations used in that architecture, such as:

**Bind**:

When a service must be accessed, this operation invokes and initializes interaction within its caller in runtime, using binding information provided by the service description to both locate and contact it.

**Publish**:

A service must be published in an Service Registry to be accessed. The Service Provider thus contacts the Service Registry to publish the service.

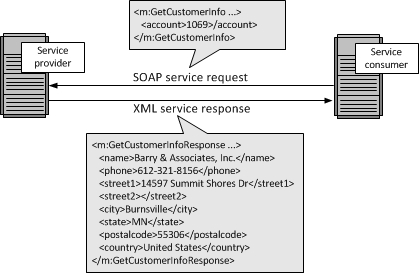
**Discover**:

A Service Requestor ﬁnds a description of the service or queries a Service Registry for the required service type. A Service Requestor can ﬁnd a service interface description in both run time or development time. Then, the necessary information regarding bindings and locales to invoke a service are found and contacted.

### **SOAP**

SOAP was originally part of the specification that included the Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). It is used now without WSDL and UDDI.

Instead of the discovery process described in the History of the Web Services Specification section below, SOAP messages are hard-coded or generated without the use of a repository. The interaction is illustrated in the figure below. More on [SOAP](https://www.service-architecture.com/articles/web-services/soap.html).



A SOAP package consists of four parts:

**SOAP Envelope**:

Deﬁnes a framework that contains the message content, who can process that message and the status of obligation to process it. It’s a SOAP message structure whose syntactic elements of the message are encapsulated.

**SOAP Codiﬁcation**:

Deﬁnes serialization mechanisms that could be used to exchange instances or data type’s deﬁned by an application.

**RPC SOAP**:

Speciﬁes how encapsulate remote procedure calls and responses inside the message, thus invoking remote procedures.

**Binding Framework and SOAP Transport**:

Deﬁnes a abstract framework to exchange SOAP envelopes between applications using a simple transport protocol. In the SOAP speciﬁcation, another concepts are also important;

**SOAP Client**:

It’s a program that creates a XML document containing the necessary information to invoke remotely a method within a distributed system (it could be a WEB or application server).

**SOAP Server**:

It’s responsible for executing a SOAP message and acts like an interpreter and distributor of documents.

**APPLICATIONS OF SOAP**

**JAX-WS**, a Java API for the XML-Based Web Services, provided a standard way to develop a Web Services in SOAP (Simple Object Access Protocol) notation.

In JAX-WS, Web Services are called/invoked through remote procedure calls. SOAP protocol is used to exchange of information between the client and the Web Service. Message exchange between the client and the server performed through XML based SOAP messages. Which is used in this project for the shopping module.

Java API for XML Web Services (JAX-WS) is a technology for building web services and clients that communicate using XML. JAX-WS allows developers to write message-oriented as well as Remote Procedure Call-oriented (RPC-oriented) web services.

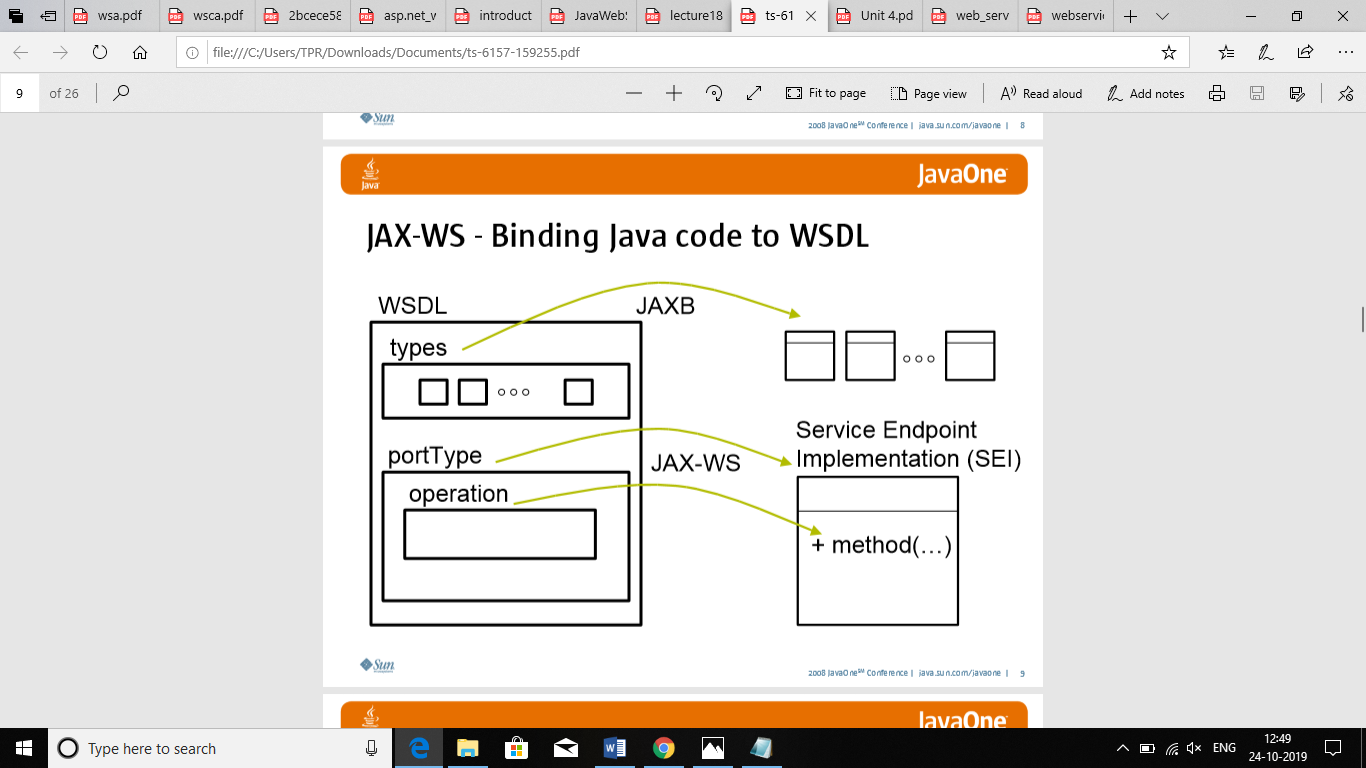
In JAX-WS, a web service operation invocation is represented by an XML-based protocol, such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

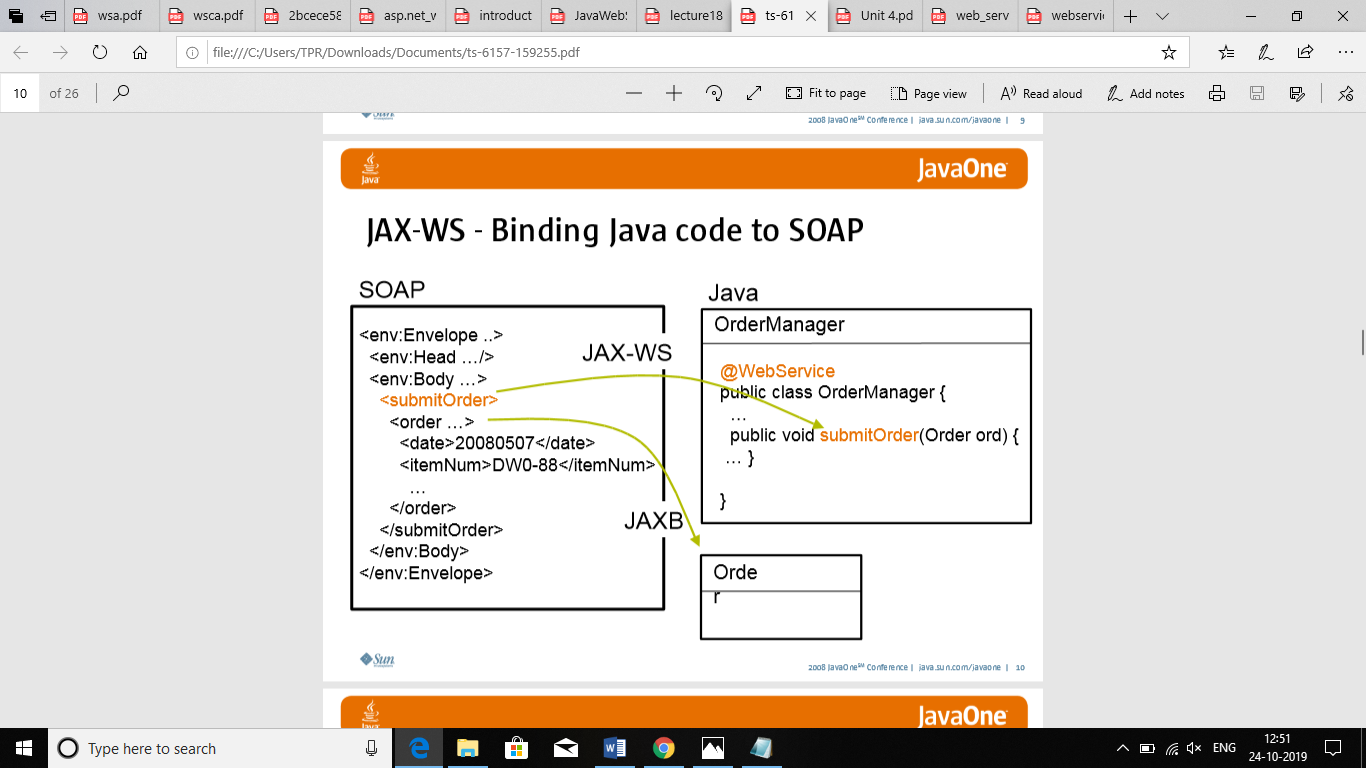
Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language.

The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. With JAX-WS, the developer does not generate or parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages.

With JAX-WS, clients and web services have a big advantage: the platform independence of the Java programming language. In addition, JAX-WS is not restrictive: A JAX-WS client can access a web service that is not running on the Java platform, and vice versa.

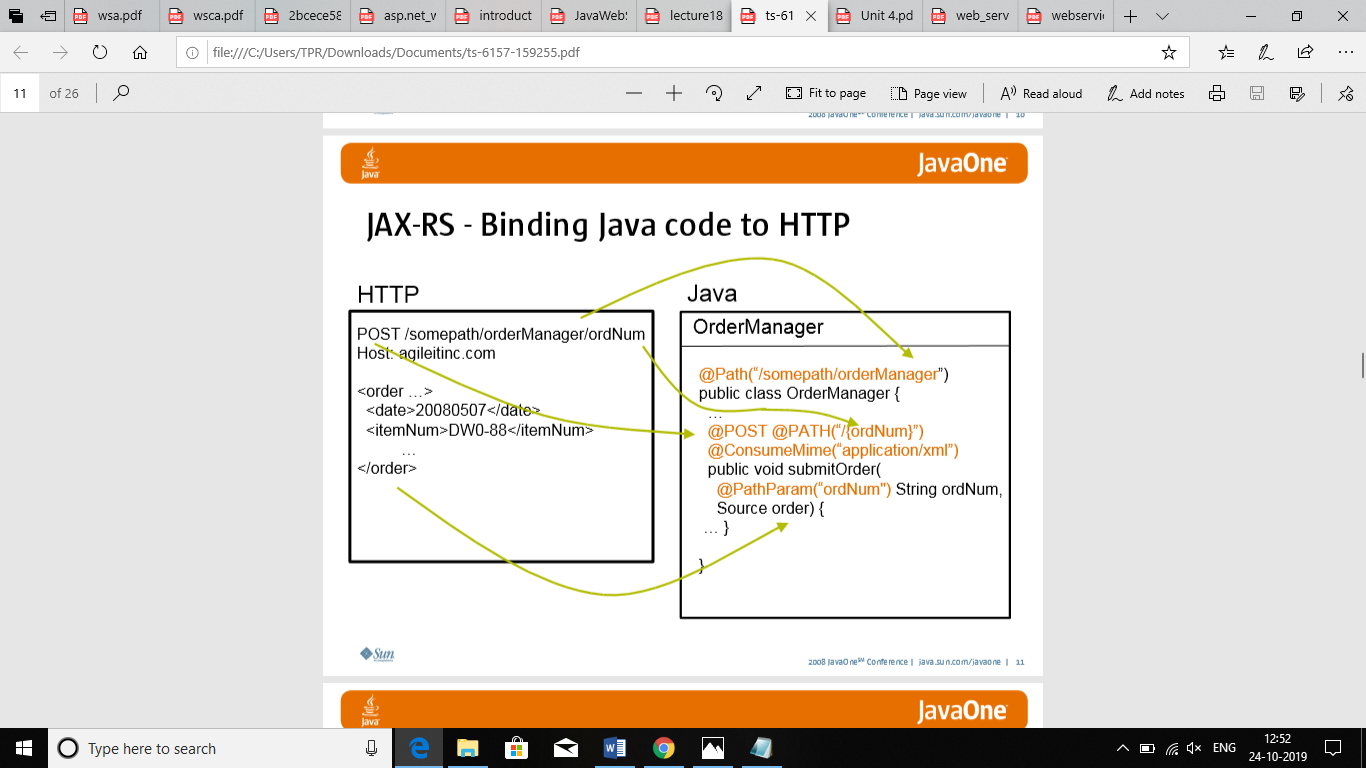
This flexibility is possible because JAX-WS uses technologies defined by the W3C: HTTP, SOAP, and WSDL. WSDL specifies an XML format for describing a service as a set of endpoints operating on messages.





**JAX-RS** is a Java API for RESTful Web Services Remote procedure call in this case is represented a HTTP- request and the necessary data is passed as parameters of the query.

Typically used for XML data exchange or JSON (JavaScript Object Notation) data exchange and this is used in the car mechanical shop finder and makes the client to book the mechanic based on their skills and the time and money of whole repair.



**APACHE TOMCAT SERVER**

Apache Tomcat (sometimes simply "Tomcat") is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and WebSocket technologies. Tomcat provides a "pure Java" HTTP web server environment in which Java code can run.

Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under the Apache License 2.0 license.

Tomcat 7.x implements the Servlet 3.0 and JSP 2.2 specifications. It requires Java version 1.6, although previous versions have run on Java 1.1 through 1.5. Versions 5 through 6 saw improvements in garbage collection, JSP parsing, performance and scalability. Native wrappers, known as "Tomcat Native", are available for Microsoft Windows and Unix for platform integration.

Tomcat 8.x implements the Servlet 3.1 and JSP 2.3 Specifications. Apache Tomcat 8.5.x is intended to replace 8.0.x and includes new features pulled forward from Tomcat 9.0.x. The minimum Java version and implemented specification versions remain unchanged.

Tomcat started off as a servlet reference implementation by James Duncan Davidson, a software architect at Sun Microsystems. He later helped make the project open source and played a key role in its donation by Sun Microsystems to the Apache Software Foundation. The Apache Ant software build automation tool was developed as a side-effect of the creation of Tomcat as an open source project.

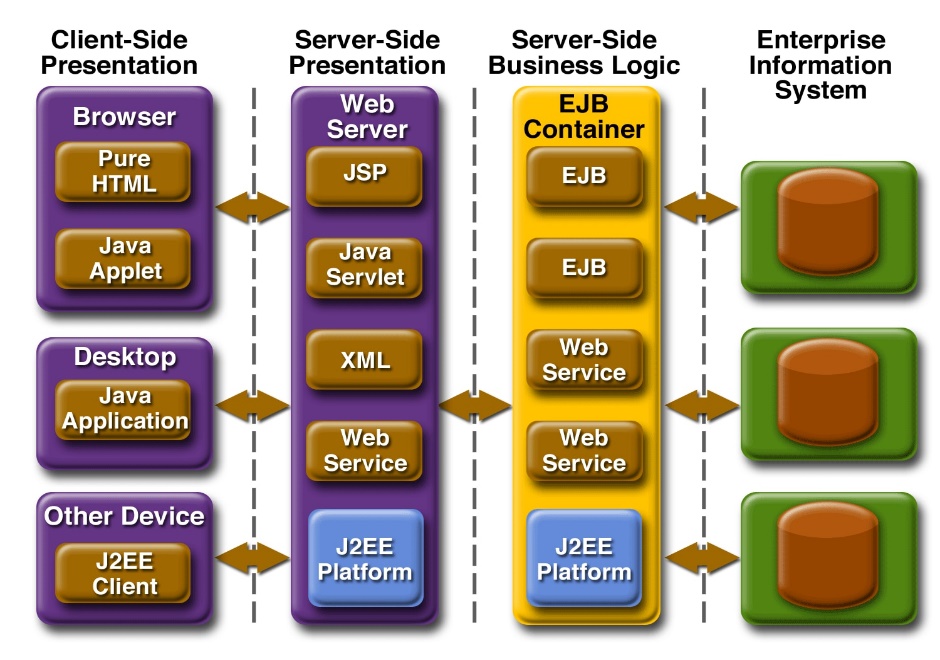
Davidson had initially hoped that the project would become open sourced and, since many open source projects had O'Reilly books associated with them featuring an animal on the cover, he wanted to name the project after an animal. He came up with Tomcat since he reasoned the animal represented something that could fend for itself. Although the tomcat was already in use for another O'Reilly title, his wish to see an animal cover eventually came true when O'Reilly published their Tomcat book with a snow leopard on the cover in 2003.

Apache software is built as part of a community process that involves both user and developer mailing lists. The developer list is where discussion on building and testing the next release takes place, while the user list is where users can discuss their problems with the developers and other users.

Some of the free Apache Tomcat resources and communities include Tomcatexpert.com (a SpringSource sponsored community for developers and operators who are running Apache Tomcat in large-scale production environments) and MuleSoft's Apache Tomcat Resource Center (which has instructional guides on installing, updating, configuring, monitoring, troubleshooting and securing various versions of Tomcat).

# **Java 2 Platform, Enterprise Edition (J2EE)**

**Enterprise Application Model**



|  |
| --- |
|  |

The Enterprise Java BluePrints for the J2EE platform describe the J2EE application model and best practices for using the J2EE platform. Building on the J2SE platform, the J2EE application model provides a simplified approach to developing highly scalable and highly available internet.  
  
Thanks to the J2EE application model, maybe the most interesting thing about J2EE applications is what they don't do. That is, various complexities inherent in enterprise applications -- transaction management, life-cycle management, resource pooling -- are built into the platform and provided automatically to the components it supports. Component and application developers are free to focus on specifics such as business logic and user interfaces.  
  
Another advantage of the J2EE platform is that the application model encapsulates the layers of functionality in specific types of components. Business logic is encapsulated in Enterprise JavaBeans (EJB) components.

Client interaction can be presented through plain HTML web pages, through web pages powered by applets, Java Servlets, or JavaServer Pages technology, or through stand-alone Java applications. Components communicate transparently using various standards.

Reusable J2EE components mean competitive choices for enterprise developers and IT organizations. The J2EE platform enables them to assemble applications from a combination of standard, commercially available components and their own custom components. From general business application components to vertical market solutions, a range of standardized J2EE.  
  
This means that an e-commerce site could be built using a combination of off-the-shelf EJB components for shopping cart behaviours, modified EJB components for specialized customer services, and completely customized layouts using JavaServer Pages technology that bring a uniqueness.  
  
This approach means faster development time, better quality, maintainability and portability, and Web services interoperability across a range of enterprise platforms. The bottom line benefits are increased programmer productivity, better strategic use of computing resources, and greater return on an organization's technology investments.

**Containers and Connectors: Hiding Complexity, Enhancing Portability**

|  |
| --- |
|  |

The J2EE application model divides enterprise applications into three fundamental parts: components, containers, and connectors.

Components are the key focus of application developers, while system vendors implement containers and connectors to conceal complexity and promote portability.  
  
Containers intercede between clients and components, providing services transparently to both, including transaction support and resource pooling. Container mediation allows many component behaviours to be specified at deployment time, rather than in program code.  
  
Connectors sit beneath the J2EE platform, defining a portable service API that communicates with existing enterprise vendor offerings. Connectors promote flexibility by enabling a variety of implementations of specific services. In particular, connectors implementing pluggable messaging contracts enable bidirectional communication between J2EE components and enterprise systems.

**Flexible User Interaction**

|  |
| --- |
|  |

The J2EE platform provides choices for graphical user interfaces across a company's intranet or on the World Wide Web. Clients can run on desktops, laptops, PDAs, cell phones, and other devices. Pure client-side user interfaces can use standard HTML and Java applets. Support for simple HTML means quicker prototypes, and support for a broader range of clients. Additionally, the J2EE platform supports automatic download of the Java Plug-in to add applet support where it's lacking. The J2EE platform also supports stand-alone Java application clients.  
  
For server-side generation of dynamic content, the J2EE platform supports two types of web component technologies: Java Servlets and JavaServer Pages (JSP). Java Servlets enable developers to easily implement server-side behaviours that take full advantage of the power of the rich Java API. JavaServer Pages technology combines the ubiquity of HTML with the power of server-side dynamic content generation. The JSP 2.0 specification supports static templates, simplified access to Java objects, and easy extensibility.

**Enterprise JavaBeans Component Model**

|  |
| --- |
|  |

Enterprise JavaBeans (EJB) technology enables a simplified approach to multitier application development, concealing application complexity and enabling the component developer.  
  
EJB technology gives developers the ability to model the full range of objects useful in the enterprise by defining several types of EJB components: session beans, entity beans, message-driven beans. Session beans represent behaviours associated with client sessions -- for example, a user purchase transaction on an e-commerce site.

Session beans can serve as Web service endpoints. Entity beans represent collections of data -- such as rows in a relational database -- and encapsulate operations on the data they represent. Entity beans are intended to be persistent, surviving as long as the data they're associated with remains viable.

Message-driven beans allow J2EE applications to process messages asynchronously. A message-driven bean normally acts as a JMS message listener, which is similar to an event listener except that it receives JMS messages instead of events. The messages may be sent by any J2EE component--an application client, another enterprise bean, or a Web component--or by a JMS application or system that does not use J2EE technology.

**Web Services Interoperability**

|  |
| --- |
|  |

The Java 2 Platform, Enterprise Edition version 1.4 is the most complete Web services platform ever. The platform features Web services support through the new JAX-RPC 1.1 API, which provides service endpoints based on servlets and enterprise beans.

JAX-RPC 1.1 provides interoperability with Web services based on the WSDL and SOAP protocols. The J2EE 1.4 platform also supports the Web Services for J2EE specification, which defines deployment requirements for Web services and utilizes the JAX-RPC programming model. In addition to numerous Web services APIs, the J2EE 1.4 platform also features support for the WS-I Basic Profile 1.0. This means that in addition to platform independence and complete Web services support, the J2EE 1.4 platform offers platform Web services interoperability.

**Expediting Development and Deployment**

|  |
| --- |
|  |

Based on these flexible component configurations, the J2EE application model means quicker development, easier customization and greater ability to deploy powerful enterprise applications. And, because it's based on the Java programming language, this model enables all J2EE applications to achieve all the benefits of Java technology: scalability, portability, and programming ease.

**DOM PARSER**



The DOM defines a standard for accessing and manipulating documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.

The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

Understanding the DOM is a must for anyone working with HTML or XML.

**The XML DOM**

All XML elements can be accessed through the XML DOM.

The XML DOM is:

* A standard object model for XML
* A standard programming interface for XML
* Platform- and language-independent
* A W3C standard

**XML DOM Properties**

These are some typical DOM properties:

* x.nodeName - the name of x
* x.nodeValue - the value of x
* x.parentNode - the parent node of x
* x.childNodes - the child nodes of x
* x.attributes - the attributes nodes of x

In the list above, x is a node object.

**XML DOM Methods**

* x.getElementsByTagName(*name*) - get all elements with a specified tag name
* x.appendChild(*node*) - insert a child node to x
* x.removeChild(*node*) - remove a child node from x

In the list above, x is a node object.

According to the XML DOM, everything in an XML document is a **node**:

* The entire document is a document node
* Every XML element is an element node
* The text in the XML elements are text nodes
* Every attribute is an attribute node
* Comments are comment nodes

**The XML DOM Node Tree**

The XML DOM views an XML document as a tree-structure. The tree structure is called a **node-tree.**

All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.

The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree.

**Node Parents, Children, and Siblings**

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters).

* In a node tree, the top node is called the root
* Every node, except the root, has exactly one parent node
* A node can have any number of children
* A leaf is a node with no children
* Siblings are nodes with the same parent

**Accessing Nodes**

You can access a node in three ways:

1. By using the getElementsByTagName() method

2. By looping through (traversing) the nodes tree.

3. By navigating the node tree, using the node relationships.

You should use a DOM parser when −

* You need to know a lot about the structure of a document.
* You need to move parts of an XML document around (you might want to sort certain elements, for example).
* You need to use the information in an XML document more than once.

When you parse an XML document with a DOM parser, you get back a tree structure that contains all of the elements of your document. The DOM provides a variety of functions you can use to examine the contents and structure of the document.

**Advantages**

The DOM is a common interface for manipulating document structures. One of its design goals is that Java code written for one DOM-compliant parser should run on any other DOM-compliant parser without having to do any modifications.

**DOM interfaces**

The DOM defines several Java interfaces. Here are the most common interfaces –

* **Node** − The base datatype of the DOM.
* **Element** − The vast majority of the objects you'll deal with are Elements.
* **Attr** − Represents an attribute of an element.
* **Text** − The actual content of an Element or Attr.
* **Document** − Represents the entire XML document. A Document object is often referred to as a DOM tree.

**Common DOM methods**

When you are working with DOM, there are several methods you'll use often –

* **Document.getDocumentElement()** − Returns the root element of the document.
* **Node.getFirstChild()** − Returns the first child of a given Node.
* **Node.getLastChild()** − Returns the last child of a given Node.
* **Node.getNextSibling()** − These methods return the next sibling of a given Node.
* **Node.getPreviousSibling()** − These methods return the previous sibling of a given Node.
* **Node.getAttribute(attrName)** − For a given Node, it returns the attribute with the requested name.

**Steps to Using JDOM**

Following are the steps used while parsing a document using JDOM Parser.

* Import XML-related packages.
* Create a SAXBuilder.
* Create a Document from a file or stream
* Extract the root element
* Examine attributes
* Examine sub-elements

**DOM PARSER TO STORE AN XML FORMAT**

The Document Object Model (DOM) is an official recommendation of the World Wide Web Consortium (W3C). It defines an interface that enables programs to access and update the style, structure, and contents of XML documents.

XML parsers that support DOM implement this interface. DOM is a common interface for manipulating document structures. One of its design goals is that Java code written for one DOM-compliant parser should run on any other DOM-compliant parser without having to do any modifications.

The DOM defines several Java interfaces. Here are the most common interfaces.

* **Node** -The base datatype of the DOM.
* **Element** -The vast majority of the objects you'll deal with are Elements.
* **Attr** - Represents an attribute of an element.
* **Text** -The actual content of an Element or Attr.
* **Document** -Represents the entire XML document. A Document object is often referred to as a DOM tree.

**RUNTIME ENVIRONMENTS**

The J2EE environment relies on a foundation Java runtime to process the core Java parts of any J2EE solution. In support of Web services, J2EE provides additional runtime layers that, in turn, supply additional Web services specific APIs (explained later). Most notable is the JAX-RPC runtime, which establishes fundamental services, including support for SOAP communication and WSDL processing.

Additionally, implementations of J2EE supply two types of component containers that provide hosting environments geared toward Web services-centric applications that are generally EJB or servlet-based.

**ASP.NET WEBSERVICES**

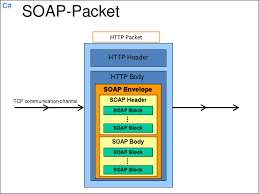
A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface. SOAP and XML created the solution for the problem that developers were facing before.

SOAP is a standard XML based protocol that communicated over HTTP. We can think of SOAP as message format for sending messaged between applications using XML. It is independent of technology, platform and is extensible too. We have SOAP and XML to give us connectivity between applications. Does it mean that I have to write XML and SOAP specific things myself to facilitate this communications? I could do that but that will be very time consuming and sometimes error prone too.

Where does Web Services come in picture? Well, Web services is the mechanism that ASP.NET framework provides to make it easy for us to write code to facilitate connectivity between applications. As ASP.NET developer, If I need an application that will be used by many other applications then I can simply decide to write a web service for it and ASP.NET framework will take care of doing the low level SOAP and XML work for us.

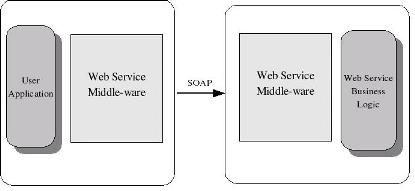
The other side of the coin is, if our ASP.NET application wants to use a web service, i.e., an application that is providing me SOAP based interface for communication. So what we are going to do now is write a small Web service to see how we can have our application communication-ready for other applications and secondly, we will try to consume a webservice to understand how we can use other applications from our application.

SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications. Exchanging data between applications is crucial in today's networked world. But data exchange between these heterogeneous applications would be complex. So will be the complexity of the code to accomplish this data exchange.

**SOAP PACKET**

One of the methods used to combat this complexity is to use XML (Extensible Markup Language) as the intermediate language for exchanging data between applications. Every programming language can understand the XML markup language. Hence, XML was used as the underlying medium for data exchange. But there are no standard specifications on use of XML across all programming languages for data exchange. That is where SOAP comes in.

* When developing Web services, you need to have some of language which can be used for web services to talk with client applications. SOAP is the perfect medium which was developed in order to achieve this purpose. This protocol is also recommended by the W3C consortium which is the governing body for all web standards.
* SOAP is a light-weight protocol that is used for data interchange between applications. Note the keyword **'light.**' Since SOAP is based on the XML language, which itself is a light weight data interchange language, hence SOAP as a protocol that also falls in the same category.
* SOAP is designed to be platform independent and is also designed to be operating system independent. So the SOAP protocol can work any programming language based applications on both Windows and[Linux](https://www.guru99.com/unix-linux-tutorial.html)platform.
* It works on the HTTP protocol –SOAP works on the HTTP protocol, which is the default protocol used by all web applications. Hence, there is no sort of customization which is required to run the web services built on the SOAP protocol to work on the World Wide Web.



**XML**

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions.

* **XML is extensible** − XML allows you to create your own self-descriptive tags, or language, that suits your application.
* **XML carries the data, does not present it** − XML allows you to store the data irrespective of how it will be presented.
* **XML is a public standard** − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

**XML TAGS**

XML tags form the foundation of XML. They define the scope of an element in XML. They can also be used to insert comments, declare settings required for parsing the environment, and to insert special instructions.

XML tags are broadly classified as follows:

## **Start Tag**

The beginning of every non-empty XML element is marked by a start-tag. Following is an example of start-tag.

<address>

**End Tag**

Every element that has a start tag should end with an end-tag. Following is an example of end-tag.

</address>

Note, that the end tags include a solidus ("/") before the name of an element.

## **Empty Tag**

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as empty. An empty element can be represented in two ways as follows.

A start-tag immediately followed by an end-tag as shown below

<hr></hr>

A complete empty-element tag is as shown below

<hr />

Empty-element tags may be used for any element which has no content.

**XML ELEMENTS**

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these. Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

## **Syntax**

Following is the syntax to write an XML element

<element-name attribute1 attribute2>

....content

</element-name>

where,

* **element-name** is the name of the element. The *name* its case in the start and end tags must match.
* **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as −

name = "value"

*name* is followed by an *=* sign and a string *value* inside double(" ") or single(' ') quotes.

**CODING:**

**MECHANIC REGISTERATION:**

**package** com.ncl;

**import** java.io.\*;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.List;

**import** javax.xml.bind.annotation.XmlAccessType;

**import** javax.xml.bind.annotation.XmlAccessorType;

**import** javax.xml.bind.annotation.XmlElement;

**import** javax.xml.bind.annotation.XmlRootElement;

**import** javax.xml.bind.annotation.XmlType;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.xml.bind.\*;

**import** javax.xml.transform.Result;

**import** java.io.FileWriter;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.parsers.ParserConfigurationException;

**import** javax.xml.transform.OutputKeys;

**import** javax.xml.transform.Transformer;

**import** javax.xml.transform.TransformerException;

**import** javax.xml.transform.TransformerFactory;

**import** javax.xml.transform.dom.DOMSource;

**import** javax.xml.transform.stream.StreamResult;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Element;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.NodeList;

**import** org.xml.sax.SAXException;

**import** java.util.Scanner;

/\*

\* Servlet implementation class simservelet

\*/

@WebServlet("/simservelet")

**public** **class** simservelet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Scanner input;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** simservelet() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter writer = response.getWriter();

String get\_firstname = request.getParameter("first\_name");

String get\_lastname = request.getParameter("last\_name");

String get\_birthday = request.getParameter("birthday");

String get\_gender = request.getParameter("gender");

String get\_email = request.getParameter("email");

String get\_phone = request.getParameter("phone");

String get\_password = request.getParameter("pasword");

String get\_address = request.getParameter("address");

writer.println("<h1> Information stored successfull </h1>");

writer.println("<h3> your informations: </h3>");

writer.println(get\_firstname);

writer.println(get\_lastname);

writer.println( get\_birthday);

writer.println(get\_gender);

writer.println(get\_email);

writer.println(get\_phone);

writer.println(get\_password);

writer.println(get\_address);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.*newInstance*();

DocumentBuilder dBuilder;

**try** {

dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.newDocument();

//add elements to Document

Element rootElement = doc.createElementNS("", "Customers");

//NodeList people = doc.getElementsByTagName("customer");

//append root element to document

doc.appendChild(rootElement);

rootElement.appendChild(

*createPersonElement*(doc, get\_firstname,get\_lastname ,get\_address));

//for output to file, console

TransformerFactory transformerFactory = TransformerFactory.*newInstance*();

Transformer transformer = transformerFactory.newTransformer();

//for pretty print

transformer.setOutputProperty(OutputKeys.***INDENT***, "yes");

DOMSource source = **new** DOMSource(doc);

StreamResult console = **new** StreamResult(System.***out***);

StreamResult file = **new** StreamResult(**new** PrintWriter(**new** BufferedWriter(**new** FileWriter("C:\\Users\\TPR\\eclipse-workspace\\usersmod\\src\\data\\ncl\\cuty.xml"))));

//write data

transformer.transform(source, console);

transformer.transform(source, file);

System.***out***.println("DONE");

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** Element createPersonElement(Document doc, String firstName,

String lastName, String address) {

//Element customer = doc.createElement("Customer");

Element el = doc.createElement("Customer");

el.appendChild(*createPersonDetailElement*(doc, "name", firstName));

el.appendChild(*createPersonDetailElement*(doc, "lname", lastName));

el.appendChild(*createPersonDetailElement*(doc, "add",address));

**return** el;

}

**private** **static** Element createPersonDetailElement(Document doc, String name,

String value) {

Element el = doc.createElement(name);

el.appendChild(doc.createTextNode(value));

**return** el;

}

/\*\*

\* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

doGet(request, response);

}

}

**(index.html) HTML PAGE:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<!-- Required meta tags-->

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1, shrink-to-fit=no"*>

<meta name=*"description"* content=*"Colorlib Templates"*>

<meta name=*"author"* content=*"Colorlib"*>

<meta name=*"keywords"* content=*"Colorlib Templates"*>

<!-- Title Page-->

<title>Customer register</title>

<!-- Icons font CSS-->

<link href=*"vendor/mdi-font/css/material-design-iconic-font.min.css"* rel=*"stylesheet"* media=*"all"*>

<link href=*"vendor/font-awesome-4.7/css/font-awesome.min.css"* rel=*"stylesheet"* media=*"all"*>

<!-- Font special for pages-->

<link href=*"https://fonts.googleapis.com/css?family=Poppins:100,100i,200,200i,300,300i,400,400i,500,500i,600,600i,700,700i,800,800i,900,900i"* rel=*"stylesheet"*>

<!-- Vendor CSS-->

<link href=*"vendor/select2/select2.min.css"* rel=*"stylesheet"* media=*"all"*>

<link href=*"vendor/datepicker/daterangepicker.css"* rel=*"stylesheet"* media=*"all"*>

<!-- Main CSS-->

<link href=*"css/main.css"* rel=*"stylesheet"* media=*"all"*>

</head>

<body>

<div class=*"page-wrapper bg-gra-02 p-t-130 p-b-100 font-poppins"*>

<div class=*"wrapper wrapper--w680"*>

<div class=*"card card-4"*>

<div class=*"card-body"*>

<h2 class=*"title"*>Registration Form</h2>

<form method=*"get"* action=*"simservelet"*>

<div class=*"row row-space"*>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>first name</label>

<input class=*"input--style-4"* type=*"text"* name=*"first\_name"*>

</div>

</div>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>last name</label>

<input class=*"input--style-4"* type=*"text"* name=*"last\_name"*>

</div>

</div>

</div>

<div class=*"row row-space"*>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>Birthday</label>

<div class=*"input-group-icon"*>

<input class=*"input--style-4 js-datepicker"* type=*"text"* name=*"birthday"*>

<i class=*"zmdi zmdi-calendar-note input-icon js-btn-calendar"*></i>

</div>

</div>

</div>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>Gender</label>

<div class=*"p-t-10"*>

<label class=*"radio-container m-r-45"*>Male

<input type=*"radio"* checked=*"checked"* name=*"gender"*>

<span class=*"checkmark"*></span>

</label>

<label class=*"radio-container"*>Female

<input type=*"radio"* name=*"gender"*>

<span class=*"checkmark"*></span>

</label>

</div>

</div>

</div>

</div>

<div class=*"row row-space"*>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>Email</label>

<input class=*"input--style-4"* type=*"email"* name=*"email"*>

</div>

</div>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>Phone Number</label>

<input class=*"input--style-4"* type=*"text"* name=*"phone"*>

</div>

</div>

</div>

<div class=*"row row-space"*>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>enter password</label>

<input class=*"input--style-4"* type=*"password"* placeholder=*"\*\*\*\*\*\*\*\*"* name=*"password"*>

</div>

</div>

</div>

<div class=*"row row-space"*>

<div class=*"col-2"*>

<div class=*"input-group"*>

<label class=*"label"*>enter address</label>

<input class=*"input--style-4"* type=*"text"* name=*"address"*>

</div>

</div>

</div>>

<div class=*"p-t-15"*>

<button class=*"btn btn--radius-2 btn--blue"* type=*"submit"*>Submit</button>

</div>

</form>

</div>

</div>

</div>

</div>

<!-- Jquery JS-->

<script src=*"vendor/jquery/jquery.min.js"*></script>

<!-- Vendor JS-->

<script src=*"vendor/select2/select2.min.js"*></script>

<script src=*"vendor/datepicker/moment.min.js"*></script>

<script src=*"vendor/datepicker/daterangepicker.js"*></script>

<!-- Main JS-->

<script src=*"js/global.js"*></script>

</body><!-- This templates was made by Colorlib (https://colorlib.com) -->

</html>

<!-- end document-->

**map.html:**

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>WELCOME</title>

</head>

<body>

<div >

<marquee behavior=*"alternate"* BGcolor=*"#ADD8E6"* direction=*"left"* class=><b>WELCOME To LOCALIZED MECHANIC FINDER </b> </marquee>

<iframe src=*"https://www.google.com/maps/embed?pb=!1m12!1m8!1m3!1d15615.086965894114!2d79.8184676978066!3d11.920967329635753!3m2!1i1024!2i768!4f13.1!2m1!1snear%20by%20mechanic%20shop%20for%20car!5e0!3m2!1sen!2sin!4v1571149020985!5m2!1sen!2sin"* width=*"800"* height=*"600"* frameborder=*"0"* style="border:*0*;" allowfullscreen=*""*></iframe>

</div>

</body>

</html>

**SOAP Response (werrr.java):**

**package** we;

**import** java.io.BufferedReader;

**import** java.io.DataOutputStream;

**import** java.io.InputStreamReader;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

**public** **class** werrr {

**public** **static** **void** main(String[] args) {

**try** {

String url = "http://localhost:49732/webbb.aspx";

URL obj = **new** URL(url);

HttpURLConnection con = (HttpURLConnection) obj.openConnection();

con.setRequestMethod("POST");

con.setRequestProperty("Content-Type","application/soap+xml; charset=utf-8");

String xml = "<?xml version=\"1.0\" encoding=\"utf-8\"?> " +

"<soap12:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" xmlns:soap12=\"http://www.w3.org/2003/05/soap-envelope\"> " +

" <soap12:Body><GetCountriesAvailable xmlns=\"http://localhost:49732/webbb.aspx\" />" +

" </soap12:Body> " +

"</soap12:Envelope>";

con.setDoOutput(**true**);

DataOutputStream wr = **new** DataOutputStream(con.getOutputStream());

wr.writeBytes(xml);

wr.flush();

wr.close();

String responseStatus = con.getResponseMessage();

System.***out***.println(responseStatus);

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(

con.getInputStream()));

String inputLine;

StringBuffer response = **new** StringBuffer();

**while** ((inputLine = in.readLine()) != **null**) {

response.append(inputLine);

}

in.close();

System.***out***.println("response:" + response.toString());

} **catch** (Exception e) {

System.***out***.println(e);

}

}

}

**ASP.NET**

**CUSTOMER SERVICE PAGE (webbb.aspx.cs):**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Xml;

using System.Xml.Linq;

using System.Text.RegularExpressions;

namespace webservice

{

public partial class webbb : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void DropDownList1\_SelectedIndexChanged(object sender, EventArgs e)

{

}

protected void Button1\_Click(object sender, EventArgs e)

{

XmlDocument xmldoc = new XmlDocument();

xmldoc.Load(Server.MapPath("car.xml"));

XmlElement parentelement = xmldoc.CreateElement("car");

XmlElement name = xmldoc.CreateElement("name");

name.InnerText = this.TextBox1.Text;

XmlElement address = xmldoc.CreateElement("address");

address.InnerText = this.TextBox2.Text;

XmlElement phone = xmldoc.CreateElement("phone");

phone.InnerText = this.TextBox3.Text;

XmlElement ser= xmldoc.CreateElement("ser");

ser.InnerText = this.DropDownList1.SelectedItem.Text;

// XmlNodeList xmlnodelist = xmldoc.GetElementsByTagName("");

parentelement.AppendChild(name);

parentelement.AppendChild(address);

parentelement.AppendChild(phone);

parentelement.AppendChild(ser);

xmldoc.DocumentElement.AppendChild(parentelement);

xmldoc.Save(Server.MapPath("car.xml"));

Response.Clear();

Response.Buffer = true;

Response.Charset = "";

Response.Cache.SetCacheability(HttpCacheability.NoCache);

Response.ContentType = "application/xml";

Response.WriteFile(Server.MapPath("~/car.xml"));

Response.Flush();

Response.End();

}

}

}

**CUSTOMER SERVICE DESIGN PAGE (Webbb.aspx):**

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="webbb.aspx.cs" Inherits="webservice.webbb" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div style="height: 434px">

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<asp:Label ID="Label6" runat="server" Text="Request service"></asp:Label>

<br />

<br />

<br />

<asp:Label ID="Label1" runat="server" Text="Name"></asp:Label>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<br />

<br />

<asp:Label ID="Label2" runat="server" Text="Address"></asp:Label>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<br />

<br />

<asp:Label ID="Label3" runat="server" Text="Mobile"></asp:Label>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

<br />

<br />

<br />

<asp:Label ID="Label5" runat="server" Text="type of service"></asp:Label>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<asp:DropDownList ID="DropDownList1" runat="server" OnSelectedIndexChanged="DropDownList1\_SelectedIndexChanged">

<asp:ListItem>choose</asp:ListItem>

<asp:ListItem>car wash</asp:ListItem>

<asp:ListItem>puncture</asp:ListItem>

<asp:ListItem>unknow</asp:ListItem>

<asp:ListItem></asp:ListItem>

</asp:DropDownList>

<br />

<br />

<br />

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<asp:Button ID="Button1" runat="server" OnClick="Button1\_Click" Text="search mech" />

</div>

</form>

</body>

</html>

**CUSTOMER BOOKING DATABASE (car.xml):**

<?xml version="1.0" encoding="utf-8"?>

<mech>

<car>

<name>keerthana</name>

<address>no.21,thiruma</address>

<phone>1234567890</phone>

<ser>car wash</ser>

</car>

<car>

<name>yogesh</name>

<address>no.21,thotta</address>

<phone>234567876</phone>

<ser>unknow</ser>

</car>

<car>

<name>brithiv</name>

<address>no.11,kamaraj</address>

<phone>3456789021</phone>

<ser>puncture</ser>

</car>

<car>

<name>harry</name>

<address>no.31,gandhi nagar,karaikal</address>

<phone>9894134621</phone>

<ser>car wash</ser>

</car>

<car>

<name>balzac</name>

<address>no.54,vallara nagar,puducherry</address>

<phone>98941345621</phone>

<ser>puncture</ser>

</car>

<car>

<name>sowmiya</name>

<address>no.56,kalapet</address>

<phone>2345678901</phone>

<ser>car wash</ser>

</car>

<car>

<name>lokesh</name>

<address>no.21,thiruma,puducherry</address>

<phone>1234567890</phone>

<ser>unknow</ser>

</car>

<car>

<name>yogesh</name>

<address>no.21,thotta</address>

<phone>1234567890</phone>

<ser>car wash</ser>

</car>

<car>

<name>yogesh</name>

<address>no.21,thotta</address>

<phone>1234567890</phone>

<ser>car wash</ser>

</car>

<car>

<name>keerthana</name>

<address>no.21,thiruma</address>

<phone>9994039721</phone>

<ser>car wash</ser>

</car>

</mech>

**MECHANIC REGISTER DATABASE:**

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<Customers>

<Customer>

<name>yogesh</name>

<lname>T</lname>

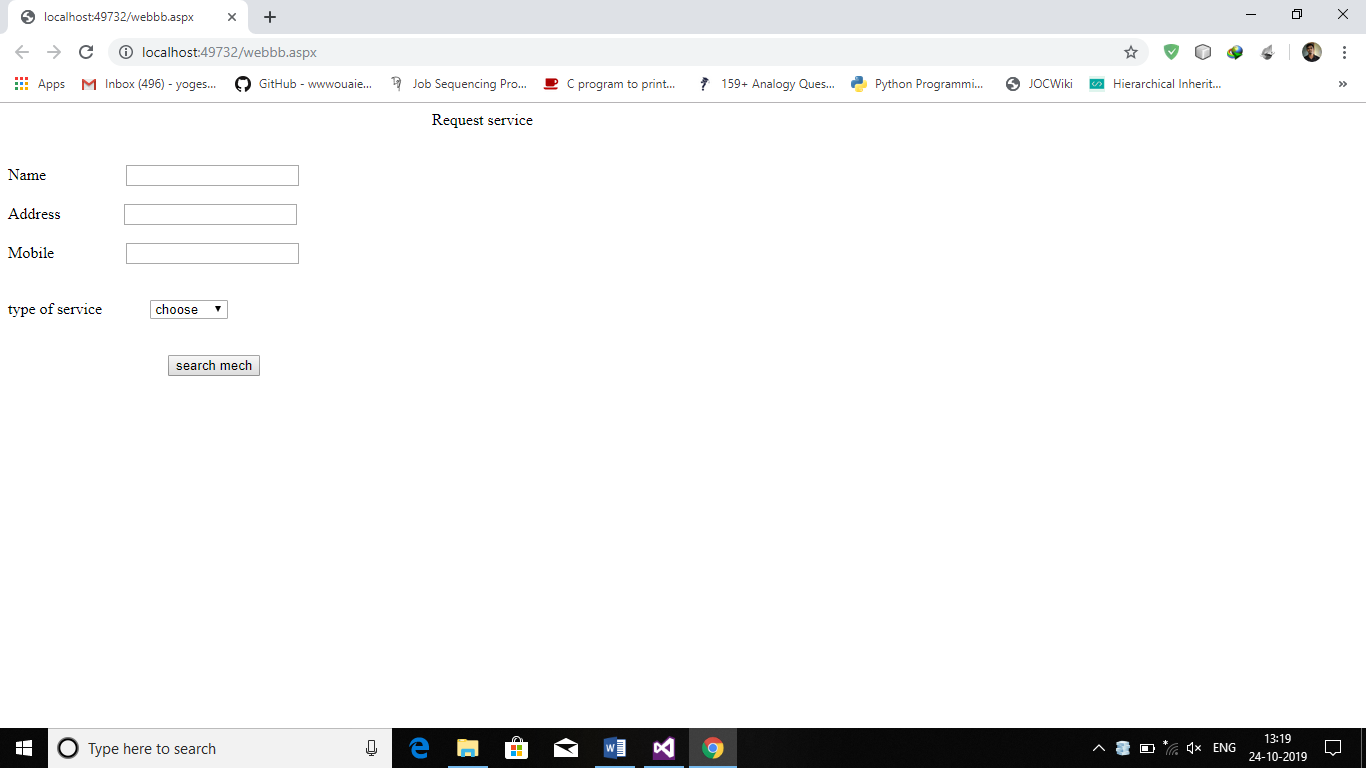
<add>no.21,kalapet</add>

</Customer>

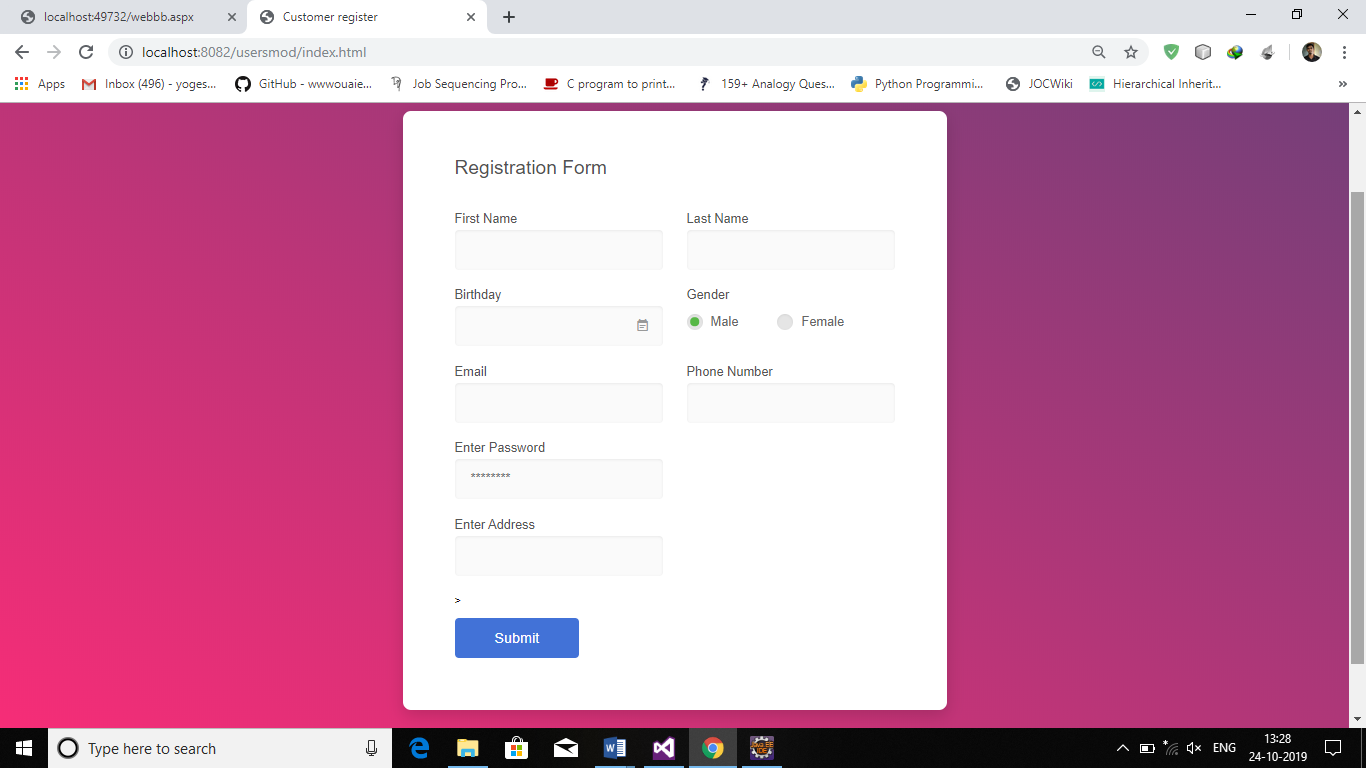
</Customers>

**INPUT SCREENSHOTS**

**SEARCH PAGE:**

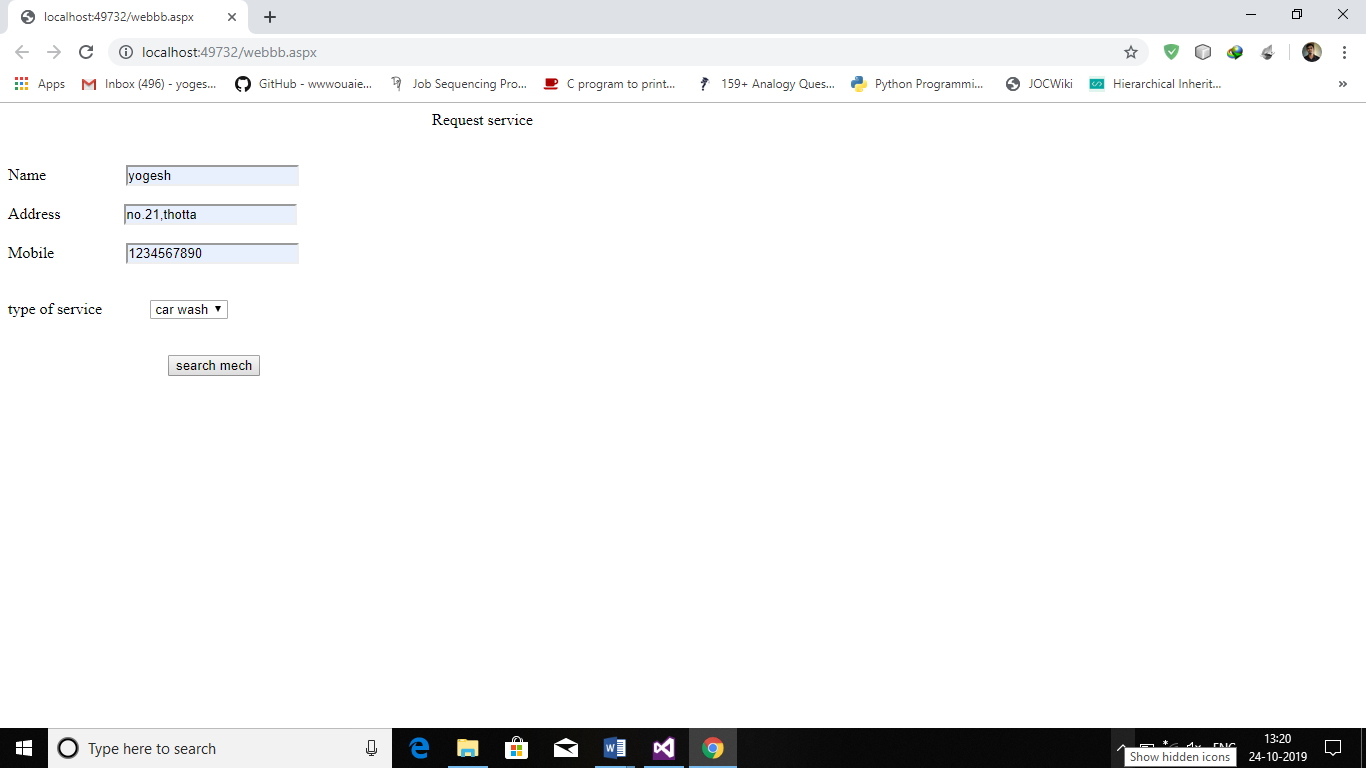


**MECHANIC REGISTERATION PAGE:**

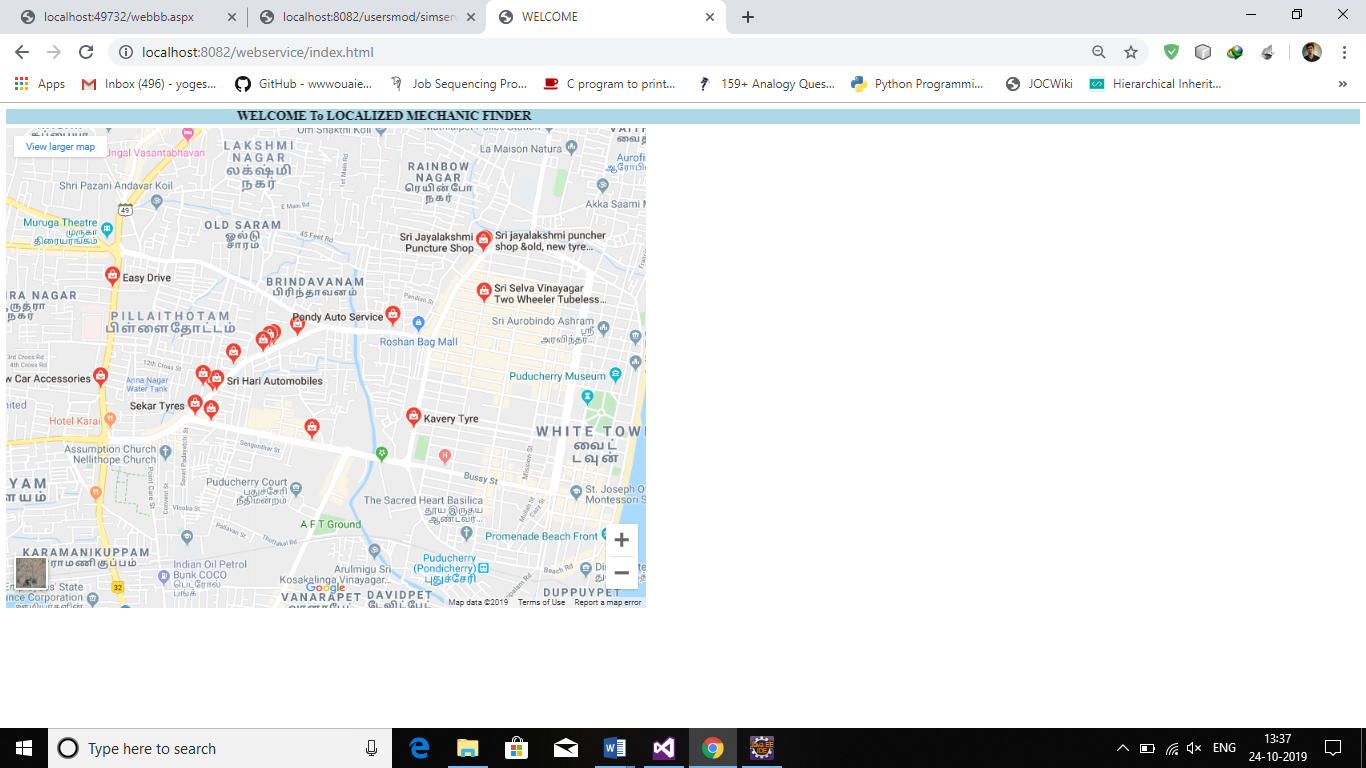


**OUTPUT SCREENSHOTS**

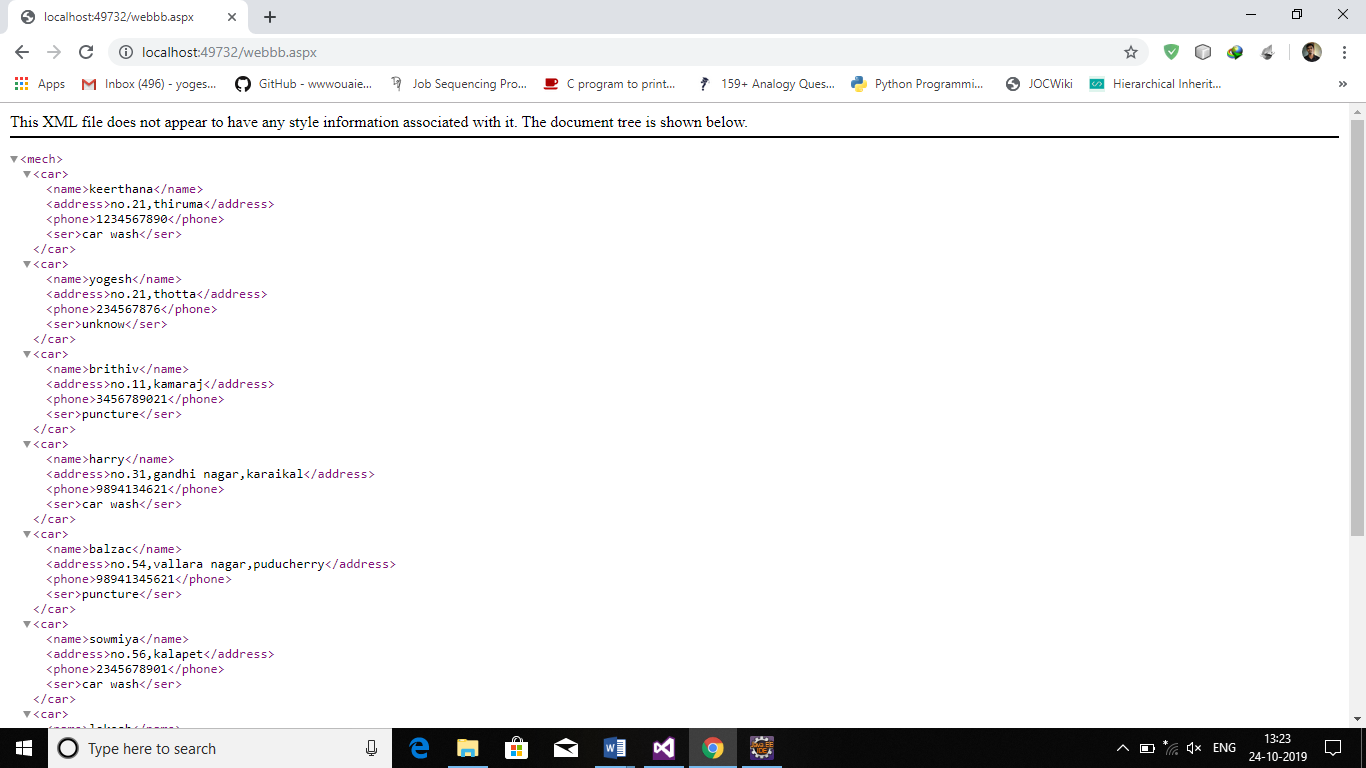
**SEARCH PAGE:**



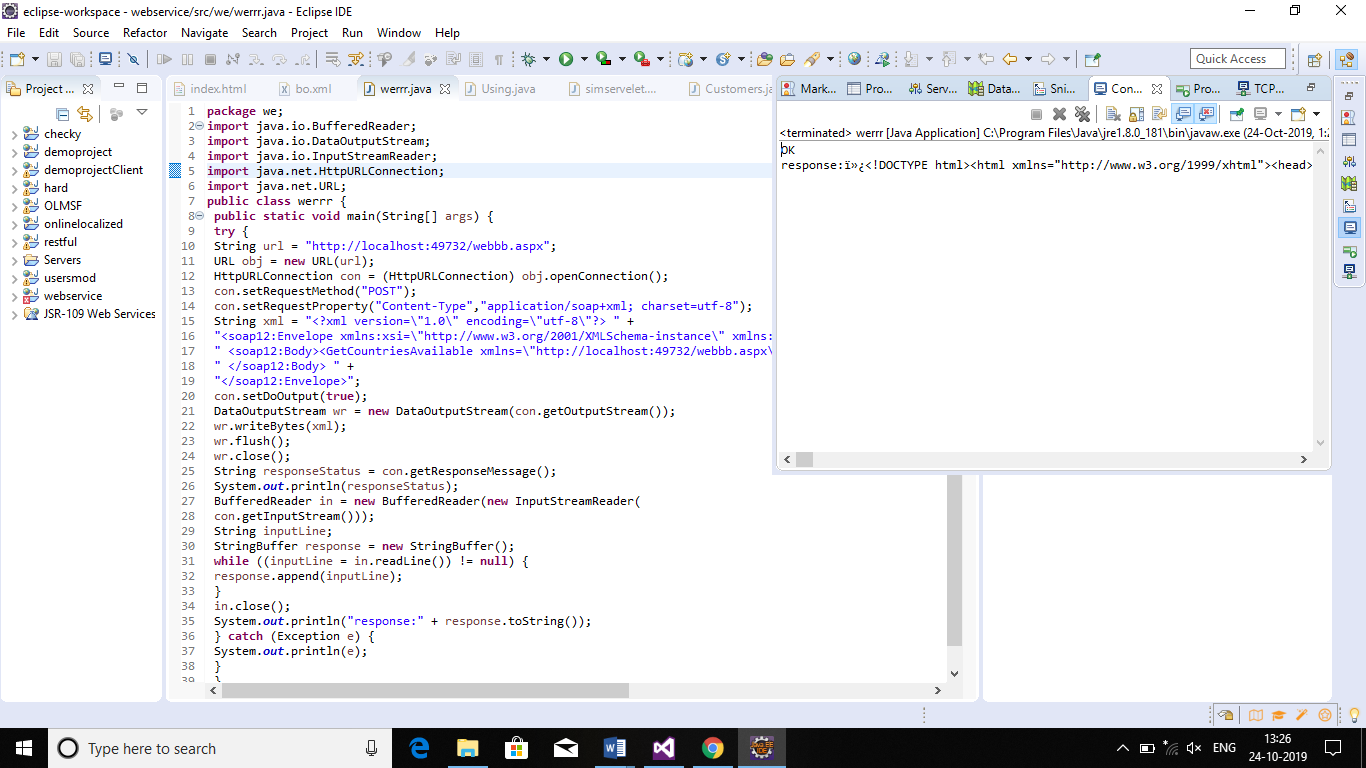
**LOCALIZED MECHANIC SHOP FINDER PAGE:**



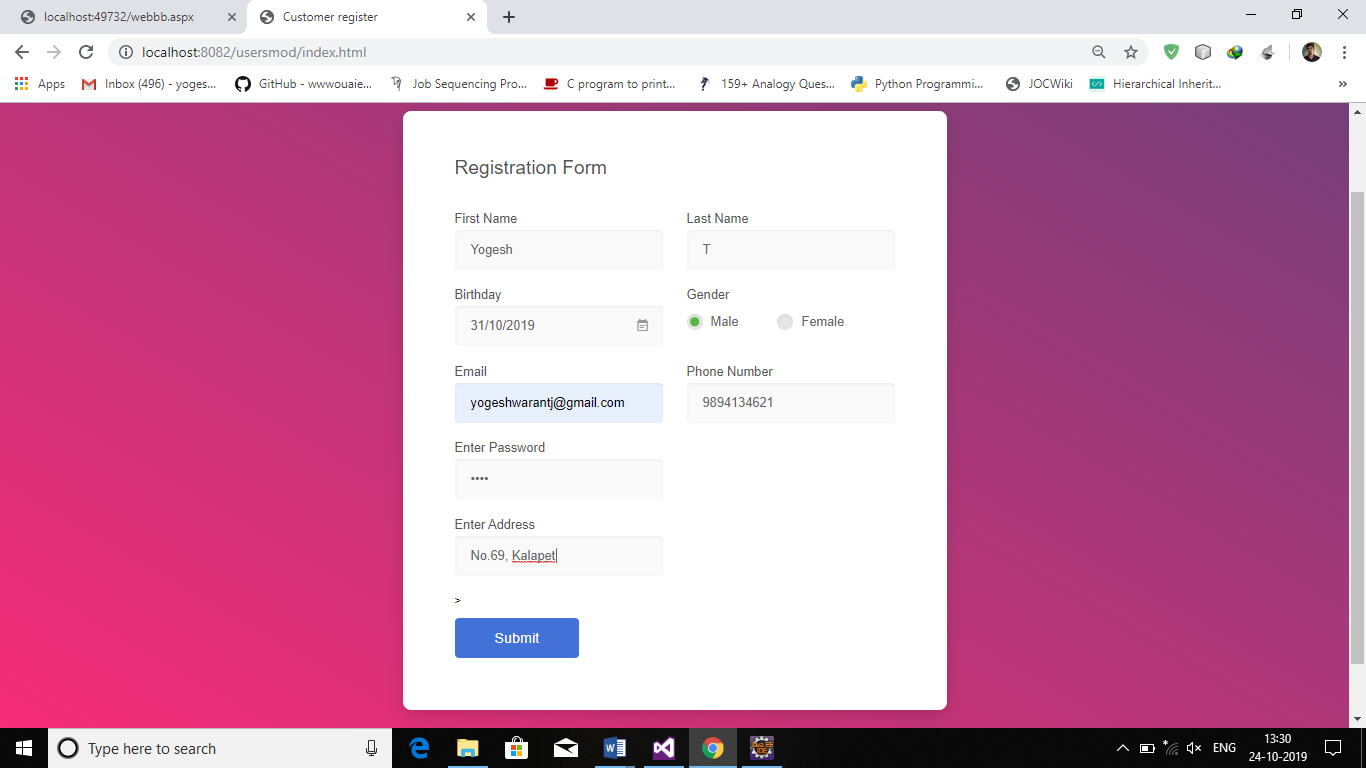
**SOAP REQUEST SENDER PAGE:**



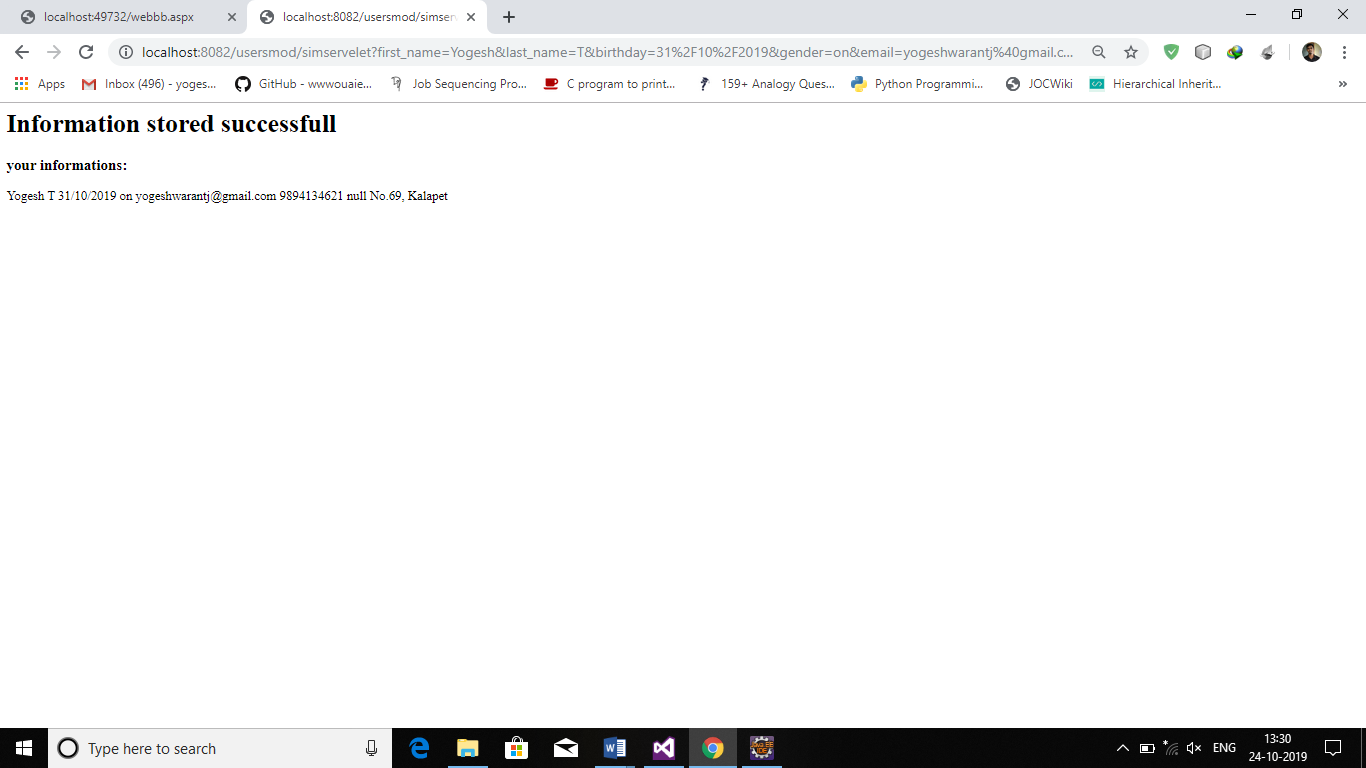
**SOAP RESPONSE PAGE:**



**MECHANIC REGISTERATION PAGE:**



**MECHANIC REGISTRATION STATUS PAGE:**



**CONCLUSION:**

The Localized car mechanic finder using jax ws with j2ee and asp.netis implemented in java and .net platforms. This project provides customers the localized service center for cars and the customer can post their types of repair and the mechanics who are willing to undertake the repairs can respond to the customer service which they have opted for. This project can be enhanced by adding shopping system using restful services.