University Of Leicester

Computer Science Department

MSc. Web Applications and Services

HiPerVison Projects on Shelton Vision's Textile Inspection Systems: Backend web-based interface

Malavika Reghunathan Nair

08 December 2014

A report submitted in fulfilment of the requirements for Course CO7501- Individual Project, Department of Computer Science, University of Leicester.

Supervised by: Dr. Reiko Heckel

Summary:

This project involves working on the Shelton WebSPECTOR surface inspection system which is currently used in various industries to inspect materials for defects. The system uses a combination of vision hardware (lights, cameras and electronics) and software. There are two primary software platforms. The first is the front-end (written in C#) where the operator interface, defect analysis and system co-ordination is done. The second is the back-end (written in C++) where the image processing and defect detection is carried out.

The outline aim of the project is to explore the possibilities of using existing web technologies and services to provide a web-based interface to allow engineers to interrogate the back-end remotely to troubleshoot any faults in identifying defects and adjust parameters.

List of Acronyms:

HTTP	Hyper Text Transfer Protocol
XML	eXtensible Markup Language
REST	REpresentational State Transfer
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
CGI	Common Gateway Script
MIME	Multi-Purpose Internet Mail Extensions
AJAX	Asynchronous JavaScript + XML
IDE	Integrated development environment
JAX-RS	Java API for RESTful Services
PHP	Hypertext Preprocessor
JSON	JavaScript Object Notation
SV	Shelton Machines Ltd

Table of Contents:

List of Figures:

List of Tables:

Chapter 1

1. Introduction

1.1 Overall aim of the project

'Shelton Vision' (SV) a company that works on textile inspection systems currently has a desktop based application for their engineers to access and write information from and to a product database. The database holds information viz. material images, process graphs and other displays.

The company's current vision is to have a back-end web based interface to allow its engineers to interrogate the backend remotely to troubleshoot any faults in identifying defects and adjust parameters.

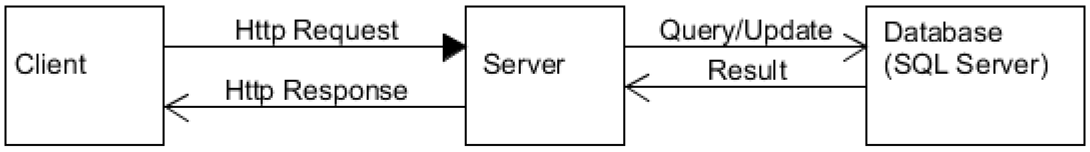


Figure 1: High-Level Architecture

The aim of this project is to do a feasibility study of current web technologies available in the market and then decide on the best technology to implement a web application to replace the existing desktop based application. Upon discussion with SV, for study purposes only a few core functionalities of the application are taken into consideration at this stage.

A comparative analysis will be done on the implemented solutions.

1.2 Objectives

The project assesses the feasibility of implementing a Web-based interface for the engineers to access the information linked to a product database. The information the engineer sees is a mixture of material images, process graphs and other displays. The existing system works under the Microsoft Windows XP operating system and a MS SQL database. It has been developed over 15 years with a mixture of Visual Basic 6, C and assembler libraries, SQL procedures, and WiT(visual programming package).

The project is supervised in cooperation with SV, who provides the requirements and technical support.

The **main objective** of this project is to:

- Review available technologies
- Create a web service API for client-service interactions
- Create a client side prototype exploiting the API

After discussing with SV, the **scenarios** to be implemented have been decided as follows:

i. Live streaming of images in its original size at client side. Images are of size 4MB stored at a location both client and server can access. Image paths are stored in the database.
ii. Update parameters in the database on a button push at client side. On successful update, display a message at client side which is send from server.
iii. Display graph at client side from a list of vector data. The vector data being stored in the database.

1.3 Schedule

A Gantt chart was prepared to manage the allocation time of the project tasks. The schedule starts from June 1st 2014 and each column represents end of week. There were regular meetings with the supervisor and SV to discuss on the progress and for technical support. It helped to decide on how to go about the solution. Sufficient time was allocated to do the background research of available technologies and to set up the environment.

Below is the embedded Gantt chart file.

Gantt Chart.xlsm

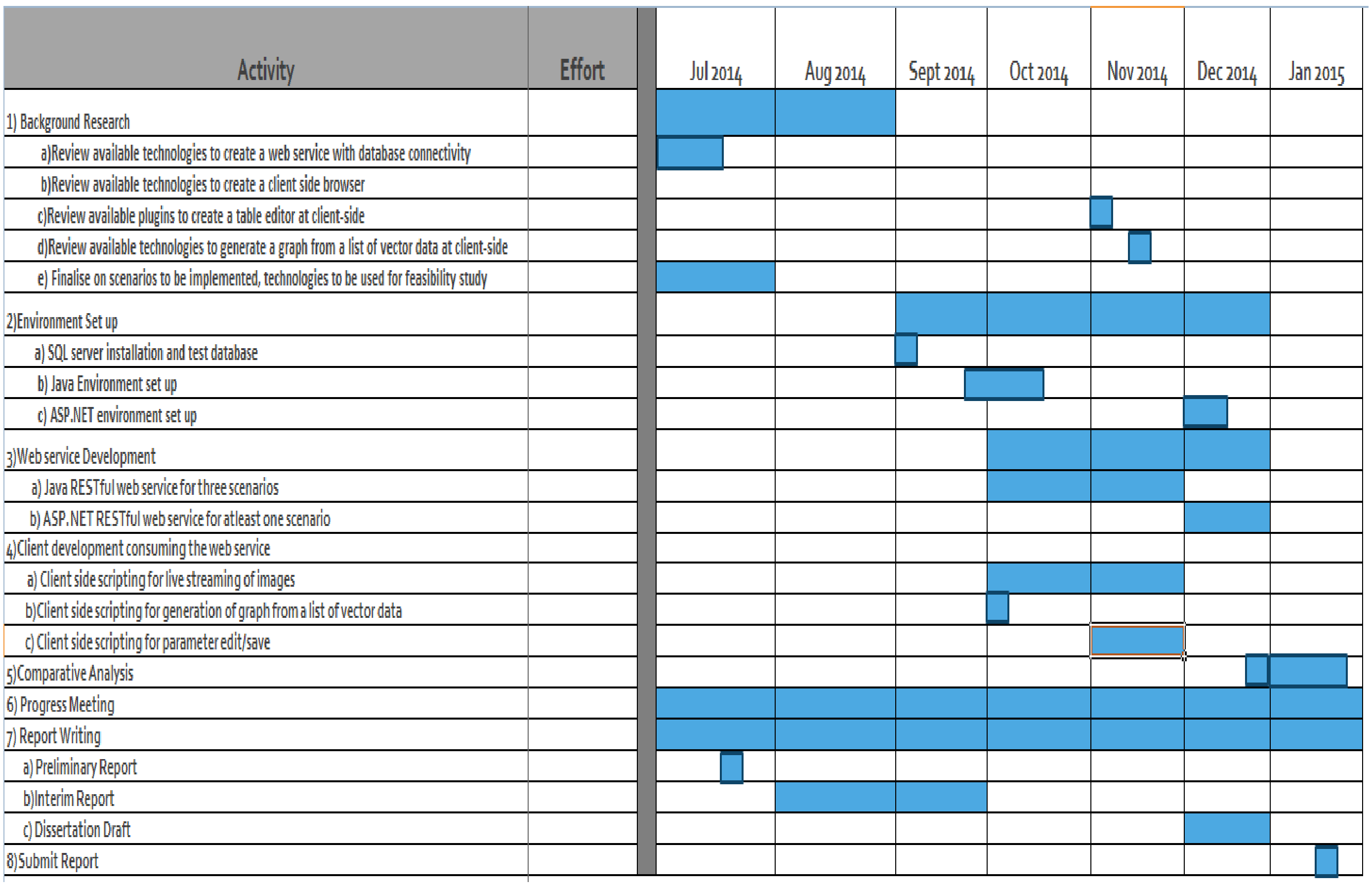


Figure 1.3a: Gantt chart excerpt

1.4 Report Structure

The following is the structure of the report.

- Chapter 2 describes the background research on client-server architecture, basics of web technology, specifications used for web services, technologies available for client-side scripting and a brief description of a framework that can be used to build web services.

- Chapter 3 describes the system architecture and describes the various layers of a 3-Tier architecture

- Chapter 4 describes the setting up of environment essential to perform a client-server task. Basically, the IDE, database, Operating system, Server etc and the obstacles faced during the progress

- Chapter 5 describes the implementation of a RESTful web service API using JAX-RS Jersey framework

- Chapter 6 describes the implementation of RESTful web service API using ASP.NET
- Chapter 7 describes how a JAX-RS web service is being consumed by a jQuery client using Ajax call
- Chapter 8 describes how a ASP .NET web service is being consumed by a jQuery client using Ajax call
- Chapter 9 describes the project management methods employed in this project
- Chapter 10 does a comparative analysis of the web application implemented using Java and .NET platforms.

1.5 Reflective Analysis

According to the initial plan, a web service API which is able to communicate to C++ applications was a requirement. But it was later decided to replace C++ application with a database as the C++ application is still not developed.

Chapter 2

2. Literature Review

2.1 Introduction

The web is a distributed, dynamic and large information repository [3]. Communication over the web or internet can be broken down to two interested parties: *Clients and Servers*. The machine providing services are servers. Clients are the machine used to connect to those services [10]. *Services* are self-contained modules-deployed over standard middleware platforms- that can be described, published, located, orchestrated and programmed using technologies over a network [4].

A web browser is the web client which acts on behalf of the user. The browser contacts the web server and sends a request for the information and receives the information and displays it on the user's computer [10]. Fig 2.1a shows how a basic web technology works.

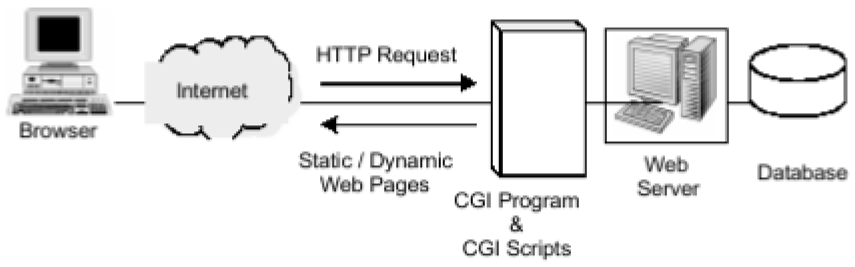


Figure 2: Web Technology [10]

The figure 2.1b illustrates the steps for a web page to access the database. A web browser cannot directly access a database. Most of the cases, browsers are a program running on the web server which acts as an intermediary to the database [10].

When the user hits a URL or clicks a submit button on the web page, the browser sends the request to the web server, which passes it to the Common Gateway Script (CGI). The CGI loads a library which sends the SQL commands to the SQL database server. The database server then executes the query and sends the result to the CGI script. The CGI script generates an HTML document and writes it to the web server. The web server sends the HTML page back to the remote user [10].

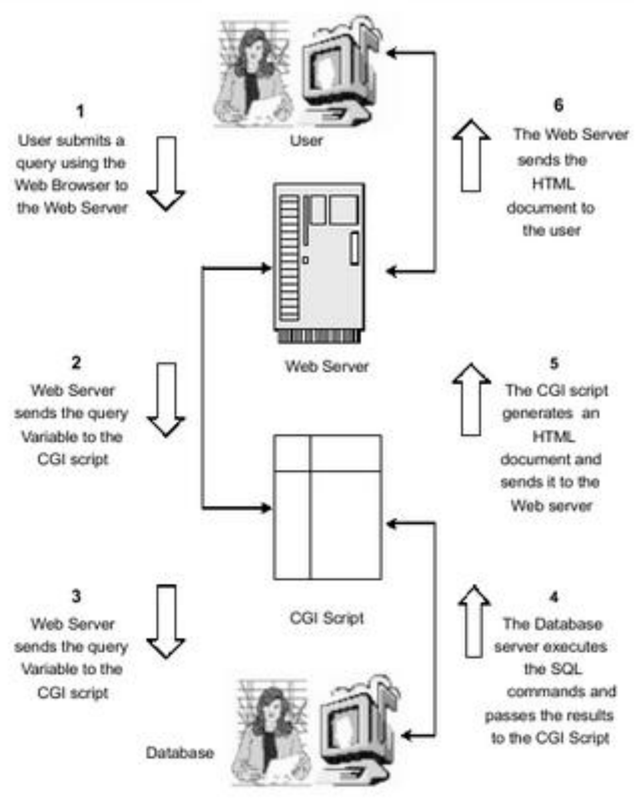

User
1
User submits a query using the Web Browser to the Web Server
6
The Web Server sends the HTML document to the user
Web Server
2
Web Server sends the query Variable to the CGI script
5
The CGI script generates an HTML document and sends it to the Web server
CGI Script
3
Web Server sends the query Variable to the CGI script
4
The Database server executes the SQL commands and passes the results to the CGI Script
Database

Figure 3: Communication between a user and web-based database [10]

A web service is a mode of communication between two machines over a network. The main goal of web service is to exchange information among applications in a standard way [9]. Two most widely used approaches for web service development are SOAP and REST (Representational State Transfer).REST has been accepted widely as a simpler alternative to SOAP and WSDL based web services [8].

2.1.1 Web service API

A web service is a piece of software, or a system that provides access to its services via an address on the World Wide Web. This address is called URI, or URL. Web service sends the information in a transferrable format that the other application or client can understand or parse [20].

Most often-used types of web service:

- SOAP

- XML-RPC
- JSON-RPC
- REST

Web service can send the information to the client in any of the transferrable format, the most common being XML and JSON. The method of converting the data to a particular format is called 'data serialisation'.

A web service can be categorised as 'RESTful' if it conforms to the constraints or the set of rules insisting by a REST architecture. RESTful APIs do not require XML-based web service protocols (SOAP & WSDL).

The main benefit of having an API-centric web application is that it can be used anywhere and it helps to build functionalities which can be used by any device, be it a browser, mobile phone, tablet or even desktop.

2.2 SOAP, REST and JSON

SOAP, REST and JSON are three specifications of web services.

SOAP defines a communication protocol for web services. WSDL enables service providers to describe their applications. UDDI offers a registry service that allows advertisement and discovery of web services [3]. XML is used to define Simple Object Access Protocol (SOAP).

REST (Representational Stat Transfer) defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages [8].REST is an architecture style for defining networked applications.

JSON (JavaScript Object Notation) is an open standard format that uses a non-strict subset of JavaScript. Information is exchanged using data objects in the form of attribute-value pair. The MIME type for JSON text data is "application/json".

JSON is much simpler than XML and is a better data exchange format.

JSON Sample:

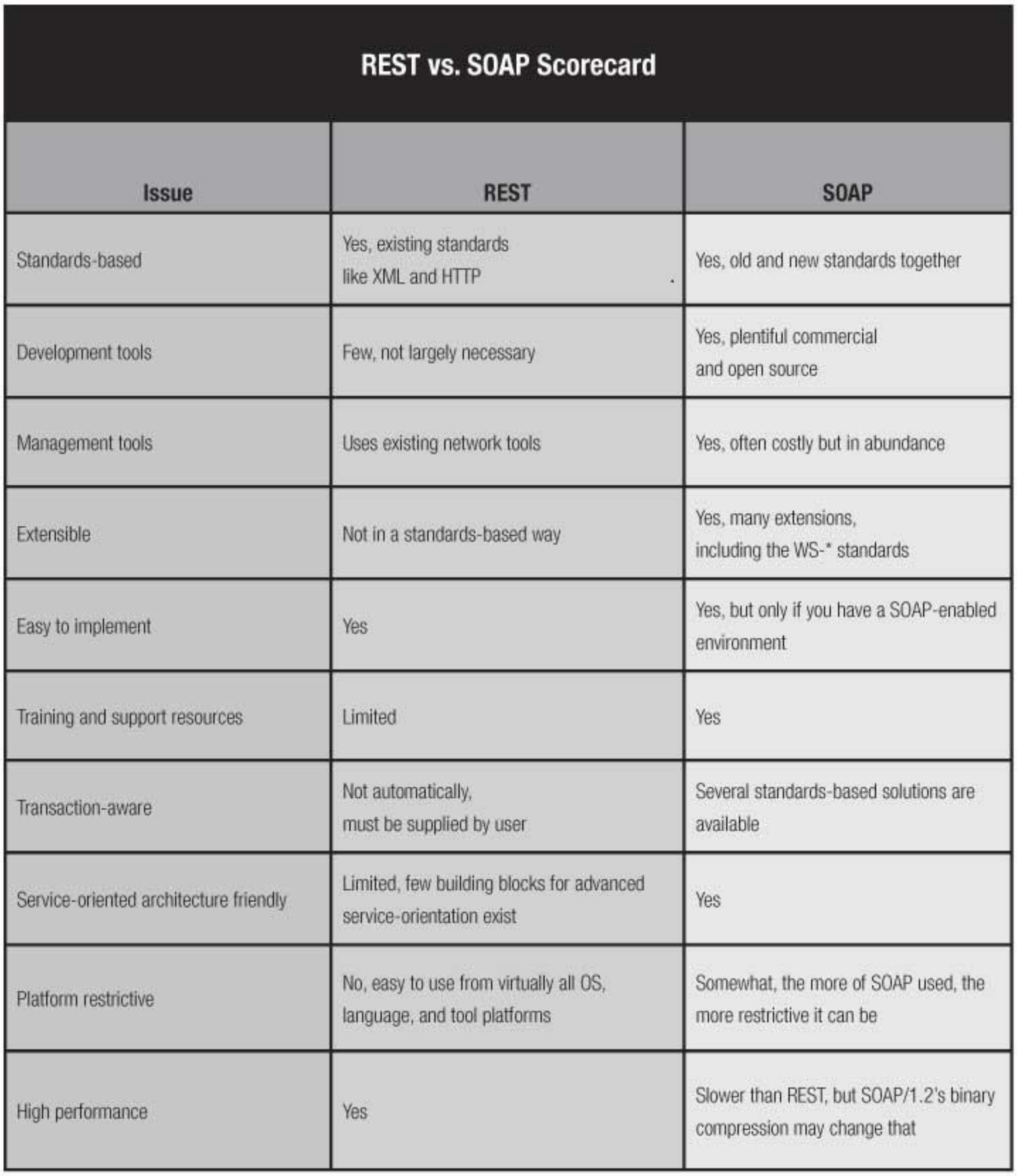
```
{
   "id": 1,
   "name": "Dave",
   "city": "London"
```

```
  “gender” : { “type” : “male”
                        },
  “phone number” :{ “type” : “work”,
                                 “number” : “000 007 131”
                               }
}
```

We could use a combination of web service specifications in order to obtain a better performance.

2.2.1 Advantages of REST over SOAP

Web services performance is an important factor. SOAP communications causes network traffic, higher latency and processing delays. To overcome this limitations the RESTful architecture is used. REST is a lightweight, easy and better alternative for the SOAP [9]. Table 1 illustrates a comparison of SOAP and REST.



REST vs. SOAP Scorecard

Issue	REST	SOAP
Standards-based	Yes, existing standards like XML and HTTP	Yes, old and new standards together
Development tools	Few, not largely necessary	Yes, plentiful commercial and open source
Management tools	Uses existing network tools	Yes, often costly but in abundance
Extensible	Not in a standards-based way	Yes, many extensions, including the WS-* standards
Easy to implement	Yes	Yes, but only if you have a SOAP-enabled environment
Training and support resources	Limited	Yes
Transaction-aware	Not automatically, must be supplied by user	Several standards-based solutions are available
Service-oriented architecture friendly	Limited, few building blocks for advanced service-orientation exist	Yes
Platform restrictive	No, easy to use from virtually all OS, language, and tool platforms	Somewhat, the more of SOAP used, the more restrictive it can be
High performance	Yes	Slower than REST, but SOAP/1.2's binary compression may change that

Table 1: REST Vs SOAP [11]

2.2.2 RESTful Web Services

RESTful web services have the advantage of being simple, lightweight and fast [22]. Basically, web services are viewed as resources and can be identified by their URLs or URIs. A Uniform Resource Identifier, or URI, in a RESTful web service is a hyperlink to a resource, and it's the only means for clients and servers to exchange representations. These representations are negotiated between clients and servers through the communication protocol at runtime— through HTTP [21].

Resources may be accessed in various formats such as HTML, plain text, XML, PDF, JPEG, and JSON among others [22]. They are commonly represented using JSON rather than XML because it is more compact than XML and it can be used with almost all programming languages including JavaScript [5]. JAX-RS uses annotations to simplify RESTful web service development. By adding annotations, we can define resources and can define the operations or actions to be performed on those resources.

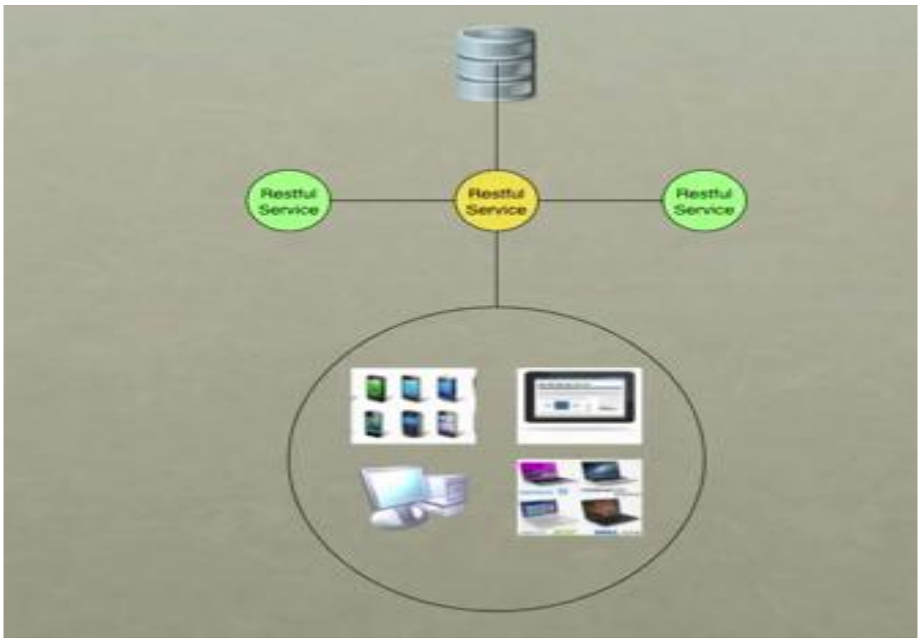


Figure 4: RESTful Web Service

With the delineated roles for resources and representations, we can now map our CRUD actions to the HTTP methods POST, GET, PUT , and DELETE as follows [21]:

Data action	HTTP protocol equivalent
CREATE	POST
RETRIEVE	GET
UPDATE	PUT

DELETE	DELETE

Following constraints define a RESTful system:

- It must be a client-server system
- It has to be stateless— there should be no need for the service to keep users' sessions; in other words, each request should be independent of others
- It has to support a caching system— the network infrastructure should support cache at different levels
- It has to be uniformly accessible— each resource must have a unique address and a valid point of access
- It has to be layered— it must support scalability
- It should provide code on demand— although this is an optional constraint, applications can be extendable at runtime by allowing the downloading of code on demand, for example, Java Applets [21].

REST services can be accessed in different programming languages – C#, Java, Python, PHP, Ruby etc.

According to Shelton Vision, in the next steps of this project, this web service should be able to communicate with a C++ application in the backend. Their existing system works under the Microsoft windows XP operating system.

Given the time scale of the project, two native frameworks are selected to develop the web service API – **Java and C#.**

2.3 JAVA API for Restful web service

Java API for RESTful Web Services (JAX-RS) is defined in JSR 311 (https://jcp.org/en/jsr/detail?id=311) [22].

2.3.1 JAX-RS with Jersey

"Jersey RESTful Web Services framework is an open source, production quality JAX-RS reference implementation for building RESTful Web Services in Java" [6].Jersey produces and consumes RESTful web services. In Jersey framework, the following methods are used which are defined in the HTTP/1.1 protocol.

- @GET is used to retrieve data or perform a query on a resource. The data returned from the web service is a representation of the requested resource [7]. This is the most-used, read only , public access method

- @POST is used to create a new resource. The web service may respond with data or status indicating success or failure [7].
- @PUT is used to update existing resources or data [7]. But it can also be used for inserting or adding data.
- @DELETE is used to remove a resource or data [7].
- @HEAD used to return meta-data of the resource

Sample code:

```
@Path("/manage/display")

public class Rest_Service {

    @Path("/graph/”)
    @GET // HTTP verb required to access this method
    @Produces(MediaType.Application_JSON)
    public Response returnVectorData () throws Exception {

            // code goes here
        }
}
```

For the sample code above, let’s assume the location of this service is at http://restjava.com. A request to retrieve the list of vector data uses the GET method with the URI http://restjava.com/manage/display/graph/. The return type of the service is ‘JSON’ .So the service will respond with the details in JSON representation.

2.4 ASP.NET Web API for a REST Service

In progress.......

2.5 Client-Side scripting techniques

There are various client side scripting techniques available today of which jQuery, is a very popular one with a good community. Some of the popular sites using jQuery include Google (code search), Twitter, Dell Inc., CBS News, Slashdot and others.

jQuery is a free, open-source and cross-platform JavaScript library used to simplify the client-side scripting of HTML. It has become very popular today and mostly used to develop dynamic web pages. Using jQuery library eliminates cross-browser incompatibilities.

The library provides a general-purpose abstraction layer for common web scripting [15] by taking a lot of common tasks and wrapping them into methods that can be invoked by a single line of code. In addition the framework comes with various plug-ins that are constantly being developed to add new features [15].

The use of JQuery has several advantages over several other JavaScript libraries. Some of them are listed below:

- Ease of use: This is one of the primary advantages of using JQuery. As mentioned above, the level of abstraction that the framework provides means that a task may be performed more easily with lesser lines of code than when using most alternatives.
- Library Size: The large library that JQuery provide allows performing more functions in comparison to other JavaScripts. In addition a compressed version of the library is only around 90 k, which is very small.
- Documentation and Tutorials: JQuery's dedicated website provides ample information, tutorial and examples to demonstrate the use of the library. In addition it has got a large developer community [17].
- Ajax support: The jQuery library has a full suite of Ajax capabilities that can access by making use of the provided APIs. Actions can be performed on pages without requiring the entire page to be reloaded. [16]

However with these come certain disadvantages as well. They are listed below:

- Limited Functionality: Since jQuery is a framework that provides an abstraction over JavaScript, there may be inevitable cases where the raw JavaScript might have to be used depending on the customization required for example on a webpage.
- jQuery javaScript file: The jQuery file is required to run jQuery commands. Though the size of the file is relatively small, it is still an overhead on the client computer and as well as the web server in certain cases [17].

It can be summarized that the advantages of using the jQuery library clearly out-weighs its disadvantages and hence is clearly a potential candidate for use in this project.

Sending Data with Ajax

The jQuery function 'jQuery.ajax()' performs an asynchronous HTTP (Ajax) request [16]. Underneath all the hype and trappings, Ajax is just a means of loading data from the server to the web browser or client without a visible page refresh [15].

Example:

HTML defining the content area of a page [16]:

```
<div id="dictionary">

</div>
```

Now, suppose we want to insert another HTML document into this without a page refresh, it can be accomplished by jQuery as follows [16]:

```
$(document).ready(function(){

   $('#letter-a a').click(function(event)  {

      event.preventDefault();

      $('#dictionary').load('a.html');

});

});
```

The .load () method does all the heavy lifting for us. The target location for the HTML snippet is specified by using a normal jQuery selector, and then pass the URL of the file to be loaded as a parameter to the method. Now, when the first link is clicked, the file is loaded and placed inside <div id="dictionary">. The browser will render the new HTML as soon as it is inserted. As soon as the new HTML snippet is inserted, any CSS rules for the main document will get applied its new elements as well [16].

Highcharts

Scenario ii described in chapter 1 can be achieved by using a plugin to display the graph from a list of vector data. There are great jQuery chart plugins available in the net.

Highcharts is a charting library written in pure JavaScript, offering an easy way of adding interactive charts to a web site or web application[19]. Highcharts currently supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange, bubble, box plot, error bars, funnel, waterfall and polar chart types [19].

DataTables

Scenarios iii described in chapter 1 can be achieved by using a plugin to control an HTML table. DataTables is one of such plugin which is very popular in web development world.

DataTables is a plug-in for the jQuery Javascript library. It is a highly flexible tool, based upon the foundations of progressive enhancement, and will add advanced interaction controls to any HTML table [23].

2.6 Conclusion

Based on the background research, it was decided to create a REST service in a Java environment as well as in ASP.NET environment with client-side in JQuery and do a comparative analysis.

Chapter 3

3. System Architecture

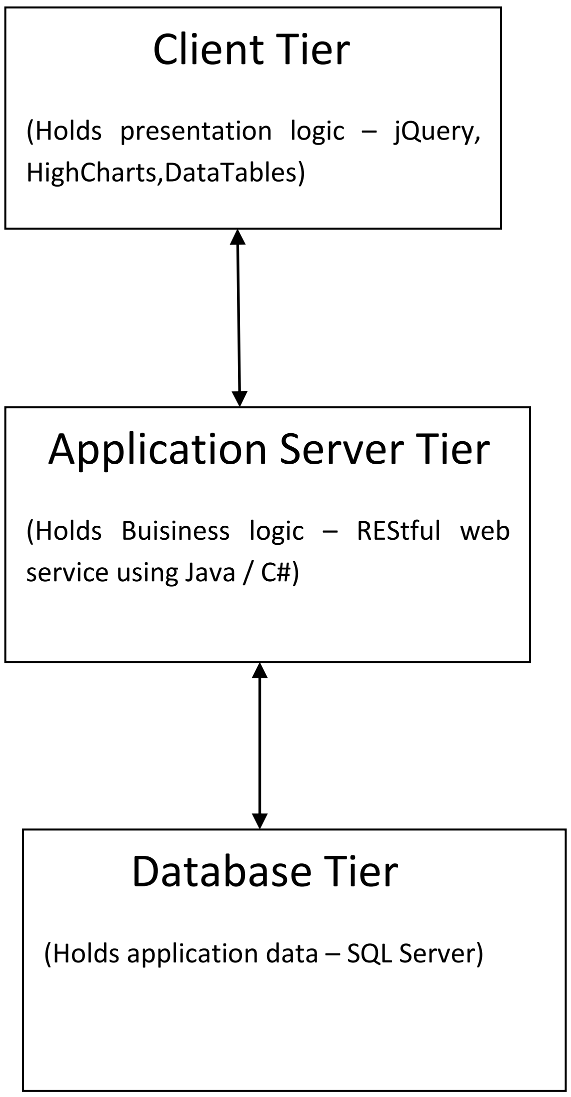

Client Tier
(Holds presentation logic – jQuery, HighCharts,DataTables)
Application Server Tier
(Holds Buisiness logic – RESTful web service using Java / C#)
Database Tier
(Holds application data – SQL Server)

Figure 3a: 3-Tier Deployment of the application

Chapter 4

4. Environment Set Up

4.1 System and Hardware

The system and its hardware details used for this project are as below:

Operating System: Windows 7 Home Premium
Hardware: 4.00GB RAM, 32-bit (x86)

4.2 Setting up the Database

The relevant files and configuration details required for setting up the SQL Server 2005 and SQL management Studio installation instructions recommended by SV are explained in the subsections.

4.2.1 SQL Server 2005 Installation Instructions

<u>Steps:</u>

1) Install the dot net framework 2.0 file dotnetfx.exe
2) Run the SQLEXPR32.EXE install file
4) For the name field put Developer, company Shelton Machines Ltd.
5) Untick 'hide advanced configuration options' and click next
6) For the feature selection, click on the plus next to Database services, and add the replication option. This is done by left clicking and selecting 'will be installed on local hard drive'
7) Add the connectivitiy components under the client components option. Then click next.
8) The named instance should be SQLExpress
9) On the next screen put a tick in the SQL server and SQL browser start service s and click next
10) IN the authentication mode screen, click on 'mixed mode' and set the password
11) Click next through collation settings
12) For Configuration options put a tick in 'add user to SQL server administrator role' and click next
13) Click next through error and Usage report settings
14) Once installed click on Start>Microsoft SQL server 2005>Configuration tools > SQL server surface area configuration.
15) Click on Surface area configuration for services and connections (near the bottom of the screen)

16) Click on remote connections and select 'Local and remote connections' and 'Using both TCP/IP and named connections'
17) Click apply and ok.
18) Select database Engine > service, then click on Stop and then Start,
19) Click OK and exit the surface area configuration program.
20) Start the windows firewall.(from control panel)
21) Click on exceptions, and Add Program...
22) Click on Browse and select “c:\programfiles\MicrosoftSQL server\MSSQL.1\MSSQL\Binn\sqlservr.exe
23) click Open and then OK
24) Click on add program again and browse to “c:\programfiles\MicrosoftSQL server\90\shared\sqlbrowser.exe”
25) Click open and then ok

4.2.2 SQL Management Studio Installation Instructions

Steps:

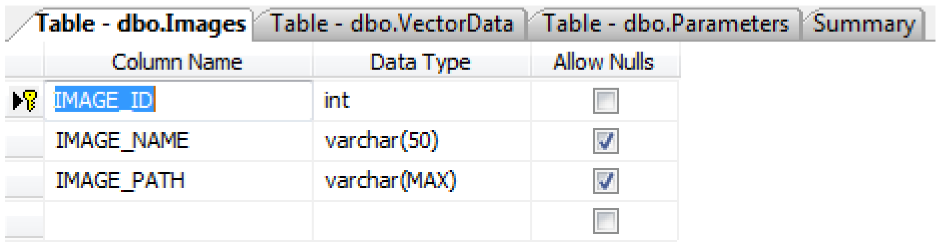
1) Run the .exe install file SQLServer2005_SSMSEE.msi
2) Run through the installation steps and when installation is complete connect to the desired database.

Server Name:	MALU-PC\SQLEXPRESS
Authentication:	SQL Server Authentication
Login:	SA
Password:	**********

4.2.3 Setting up a test database

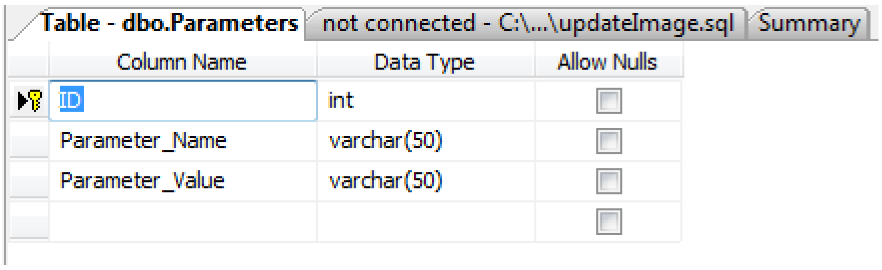
A test database ‘TestDB’ was created using SQL server 2005 with three test tables ‘Images’ , ‘VectorData’ and ‘Parameters’ whose specifications are shown in figure5a, 5b and 5c.

For testing purpose of the project, the images are of size 4MB and are stored in the C drive of the machine.



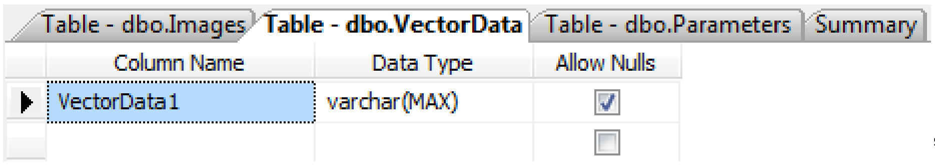
Column Name	Data Type	Allow Nulls
IMAGE_ID	int	☐
IMAGE_NAME	varchar(50)	✓
IMAGE_PATH	varchar(MAX)	✓
		☐

Figure 4.2.3a : Specification for table ‘Images’



Column Name	Data Type	Allow Nulls
ID	int	☐
Parameter_Name	varchar(50)	☐
Parameter_Value	varchar(50)	☐
		☐

Figure 4.2.3b: Specification for table ‘Parameters’



Column Name	Data Type	Allow Nulls
VectorData1	varchar(MAX)	✓
		☐

,

Figure 4.2.3c: Specification for table ‘VectorData’

4.3 Setting up Java Environment

Using Eclipse IDE for development of Java RESTful service simplifies the task significantly and offers several tools. With a new Dynamic Web Project created in the eclipse IDE, it is possible to create both service and client in the same package through a web design view. Still the coding needs to be done manually. Any Java EE container can be used as server but her **Oracle WebLogic** has been chosen as it is very easy to use once installed and configured. Also it has a very good GUI.

For this project, a web application is developed in **Eclipse 3.7**, **Oracle Enterprise Pack for Eclipse (OEPE) 12 c**, and deploy the application to the Java application Server ,**Oracle WebLogic Server 12 c**.

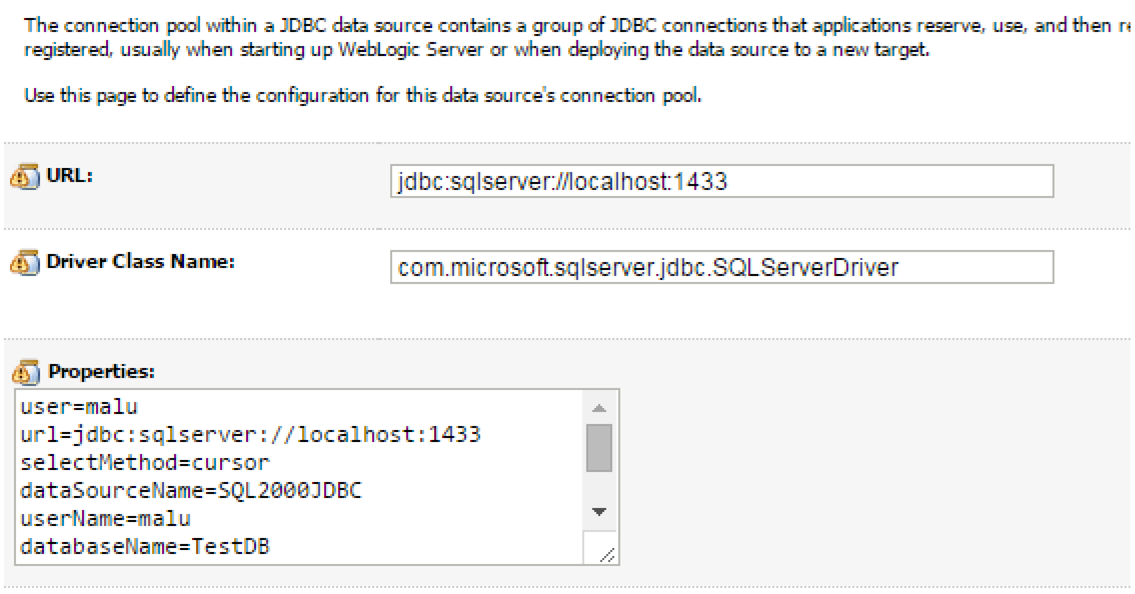
“Oracle WebLogic Server 12c is the industry's best application server for building and deploying enterprise Java EE applications with support for new features for lowering cost of operations, improving performance, enhancing scalability and supporting the Oracle Applications portfolio “[12].

Downloaded the Free Oracle WebLogic Server 12c (12.1.1),
‘wls1213_dev.zip’ from http://www.oracle.com/technetwork/middleware/fusion-middleware/downloads/index.html.
Installed Oracle WebLogic 12c to a directory C:/Oracle and created a WebLogic Server domain as base_domain. Also need to install JDK 7.0 (atleast JDK 6.0 is required)

Downloaded the Jersey 1.8 ZIP bundle ‘jersey-archive-1.18.zip’ from https://jersey.java.net/download.html and placed it to the directory C:/Oracle.It contains the Jersey JARs and core dependencies.

Download the Microsoft JDBC driver 4.0 for SQL server ‘sqljdbc4.jar’ from
http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774,
a Type 4 JDBC driver that provides database connectivity through the standard JDBC application program interfaces (APIs) available in Java Platform. Placed this jar file to the directory ‘C:\Oracle\Middleware\wlserver_12.1\server\lib’ and add the following to the CLASSPATH declaration in the bin/ commEnv batch file:
C:\Oracle\Middleware\wlserver_12.1\server\lib\sqljdbc4.jar

In order to connect the server with the database, a data source needs to be created. MS SQL data source with the name ‘SqlDataSource’ is created from the oracle WebLogic Administration console and a successful connection is established to the database. Figure 4.3a shows the properties of the created JDBC datasource and figure 4.3b shows the created datasource in the admin console.


The connection pool within a JDBC data source contains a group of JDBC connections that applications reserve, use, and then r
registered, usually when starting up WebLogic Server or when deploying the data source to a new target.

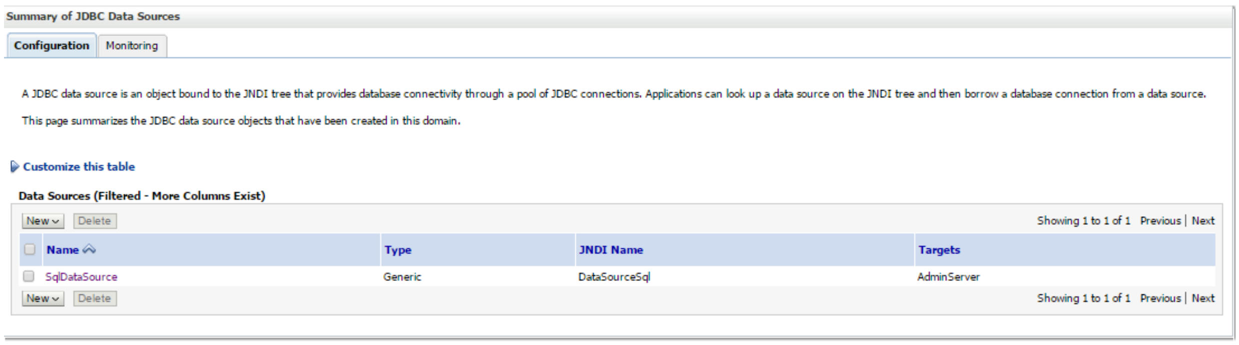
Use this page to define the configuration for this data source's connection pool.

URL: jdbc:sqlserver://localhost:1433

Driver Class Name: com.microsoft.sqlserver.jdbc.SQLServerDriver

Properties:
```
user=malu
url=jdbc:sqlserver://localhost:1433
selectMethod=cursor
dataSourceName=SQL2000JDBC
userName=malu
databaseName=TestDB
```

Figure 4.3a JDBC DataSource properties


Summary of JDBC Data Sources

A JDBC data source is an object bound to the JNDI tree that provides database connectivity through a pool of JDBC connections. Applications can look up a data source on the JNDI tree and then borrow a database connection from a data source.

This page summarizes the JDBC data source objects that have been created in this domain.

Data Sources (Filtered - More Columns Exist)

Showing 1 to 1 of 1 Previous | Next

Name	Type	JNDI Name	Targets
SqlDataSource	Generic	DataSourceSql	AdminServer

Showing 1 to 1 of 1 Previous | Next

Figure 4.3b Summary of JDBC datasource created

4.4 Setting up ASP.NET environment

[in progress....]

4.5 Obstacles faced during set up

Problem: Installation of MSSQL 2005 in Windows 8 (x64) Operating System unsuccessful.

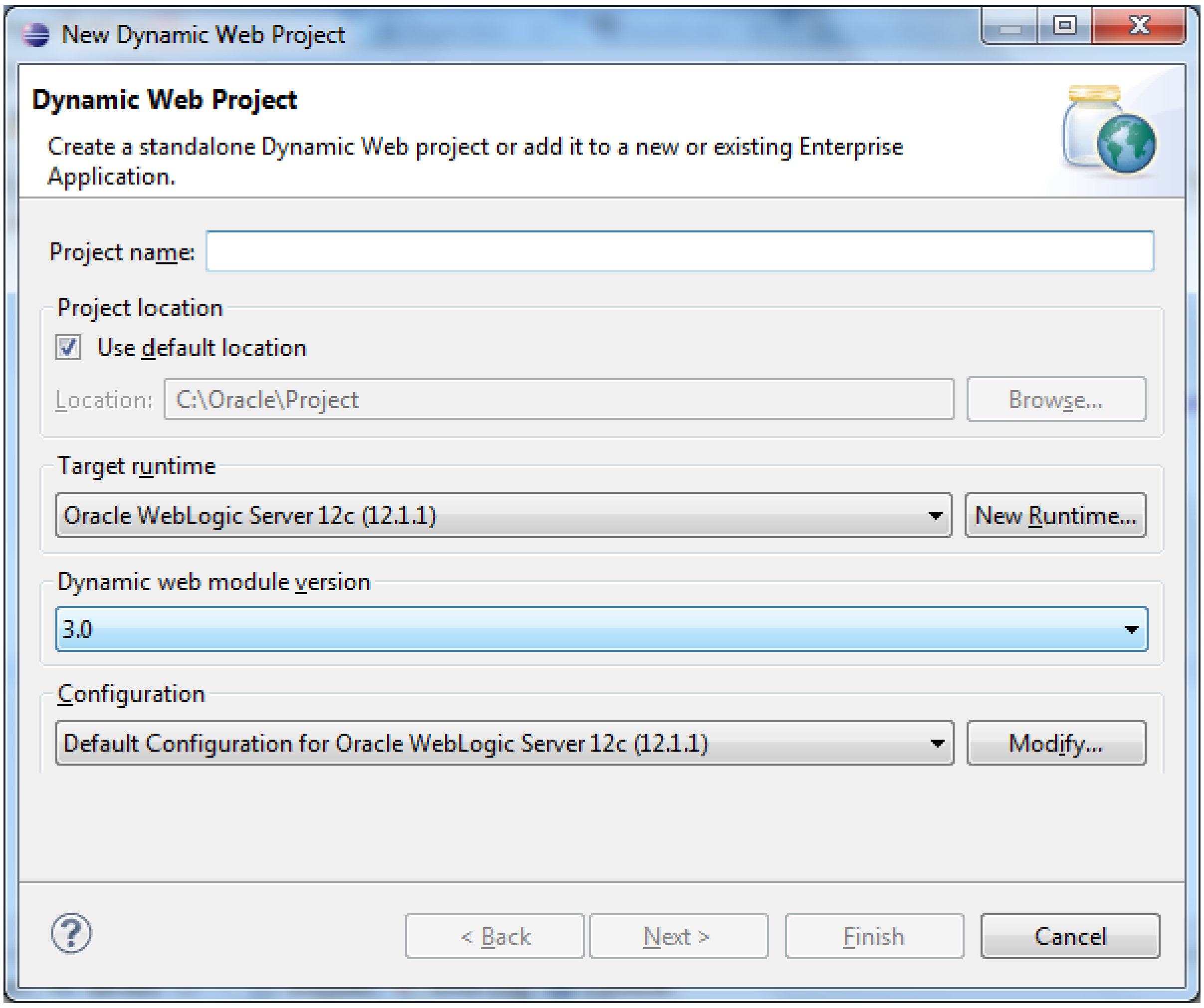
Solution: After discussing with SV, ordered Windows 7 Home Premium OS online as it was not physically available to purchase. Following the installation of Windows 7, MS SQL 2005 was successfully installed.

Chapter 5

5. Implementation of REST Service using Java

This section describes the steps followed to create the JAX-RS web service using the environment set up in section 4.3 of chapter 4.

Selected **File|New|Other** and in the **New** wizard, selected **Web|Dynamic Web Project** . In the **New Dynamic Web Project** wizard ((Figure 5a), specified project name as **‘com.sheltonmachines.service’** and a new target runtime is configured for **Oracle Weblogic Server 12c (12.1.1).** A new dynamic web project gets created with necessary libraries.


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☑ Use default location

Location: C:\Oracle\Project

Target runtime: Oracle WebLogic Server 12c (12.1.1)

Dynamic web module version: 3.0

Configuration: Default Configuration for Oracle WebLogic Server 12c (12.1.1)

Figure 5a. New Dynamic web project wizard

Copied the Jersey JARs and sqljdbc4.jar explained in section 4.3 of chapter 4 to the web content library of the project at the below location :

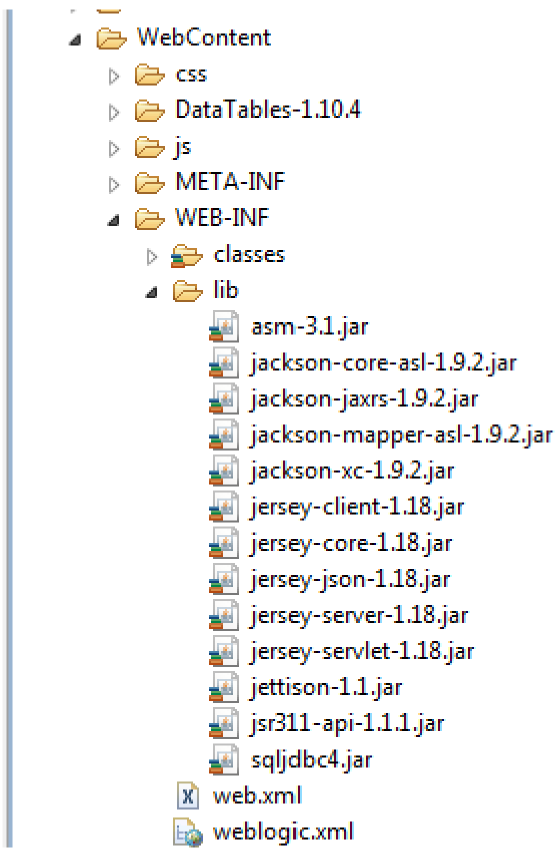
C:\Oracle\Project\com.sheltonmachines.service\WebContent\WEB-INF\lib

The project gets created and the JAR files get added to the project as shown in Figure 5b.JAX-RS Servlet and the Servlet mapping get configured in web.xml, as shown in Figure 5c.

A RESTful web service resource is created using a root resource class. A root resource class is a POJO, annotated with the @PATH annotation and consisting of at least one method annotated with the @PATH annotation or request method designator such as @GET , @PUT , @POST , or @DELETE . Selected **File | New | Other** and in the New wizard select **Java | Class**. In New Java Class specified the Source folder (com.sheltonmachines.service/src), specified a new Package name for better organization and maintainability (com.sheltonmachines.service.manage), and specify a class name (manage_display) in the Name field as shown in the figure 5d. Figure 5e shows a code snippet of the resource class(manage_display). As seen in line 41, the HTTP verb “POST” is used to invoke the service method.

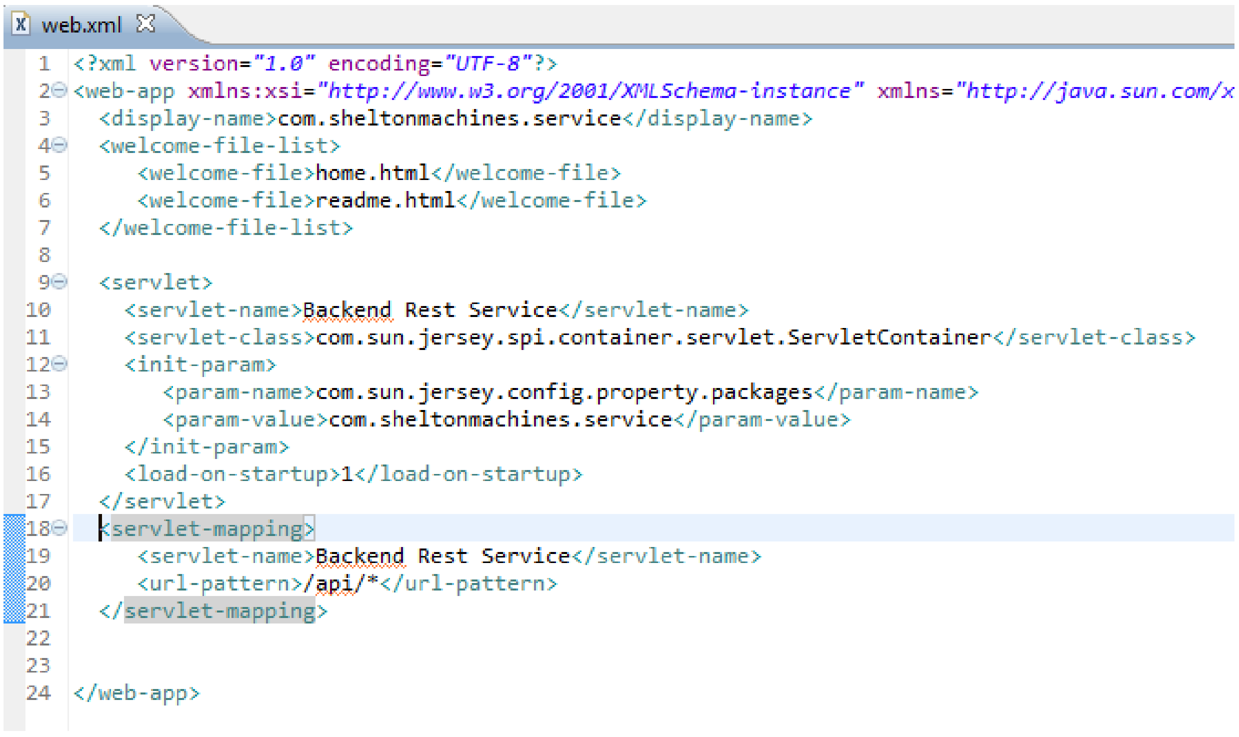
To separate the RESTful service from SQL Logic and JSON object mapping, the code handling database queries are done in a separate package (com.sheltonmachines.service.dao) and the code to convert the data into JSON format are done in a separate package(com.sheltonmachines.service.util).

The advantage of this is that if the REST service needs changing, there is no risk of breaking the SQL connection and logging packages. The structure of the packages is shown in figure 5f.



```
WebContent
  css
  DataTables-1.10.4
  js
  META-INF
  WEB-INF
    classes
    lib
      asm-3.1.jar
      jackson-core-asl-1.9.2.jar
      jackson-jaxrs-1.9.2.jar
      jackson-mapper-asl-1.9.2.jar
      jackson-xc-1.9.2.jar
      jersey-client-1.18.jar
      jersey-core-1.18.jar
      jersey-json-1.18.jar
      jersey-server-1.18.jar
      jersey-servlet-1.18.jar
      jettison-1.1.jar
      jsr311-api-1.1.1.jar
      sqljdbc4.jar
    web.xml
    weblogic.xml
```

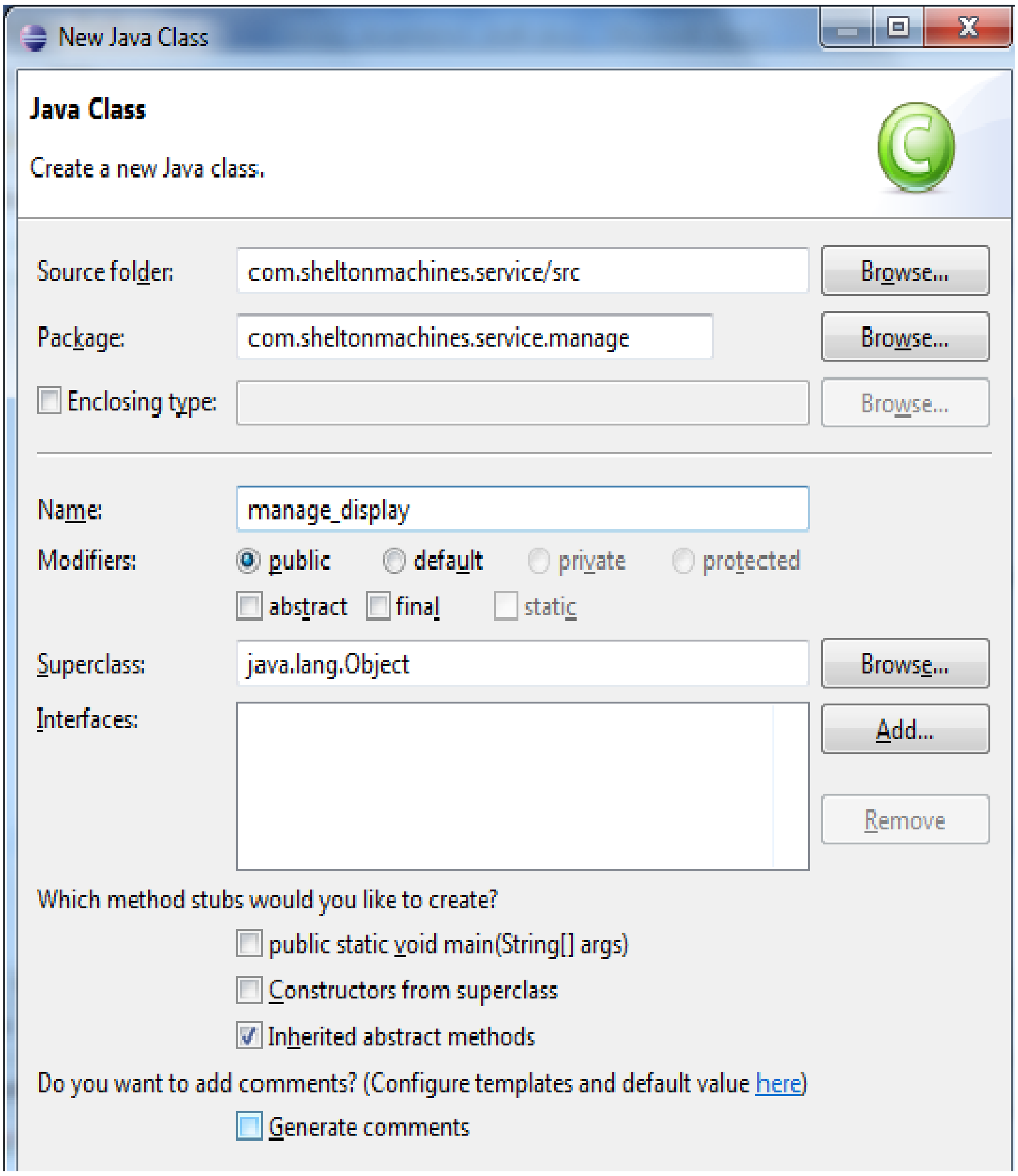
Figure5b. After adding the Jersey JARs and sqljdbc4.jar to the web content library



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/x
  <display-name>com.sheltonmachines.service</display-name>
  <welcome-file-list>
     <welcome-file>home.html</welcome-file>
     <welcome-file>readme.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>Backend Rest Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
       <param-name>com.sun.jersey.config.property.packages</param-name>
       <param-value>com.sheltonmachines.service</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
     <servlet-name>Backend Rest Service</servlet-name>
     <url-pattern>/api/*</url-pattern>
  </servlet-mapping>


</web-app>
```

Figure 5c. Web.xml file



New Java Class

Java Class

Create a new Java class.

Source folder:	com.sheltonmachines.service/src	Browse...
Package:	com.sheltonmachines.service.manage	Browse...
☐ Enclosing type:		Browse...
Name:	manage_display	
Modifiers:	● public ○ default ○ private ○ protected	
	☐ abstract ☐ final ☐ static	
Superclass:	java.lang.Object	Browse...
Interfaces:		Add...
		Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

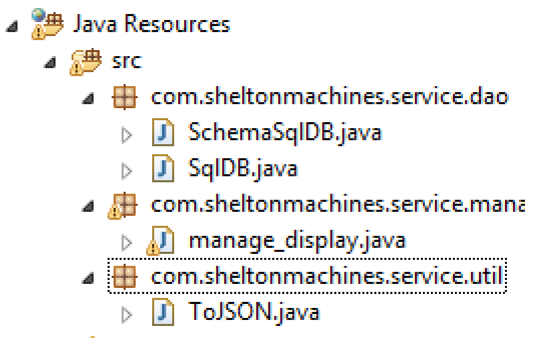
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

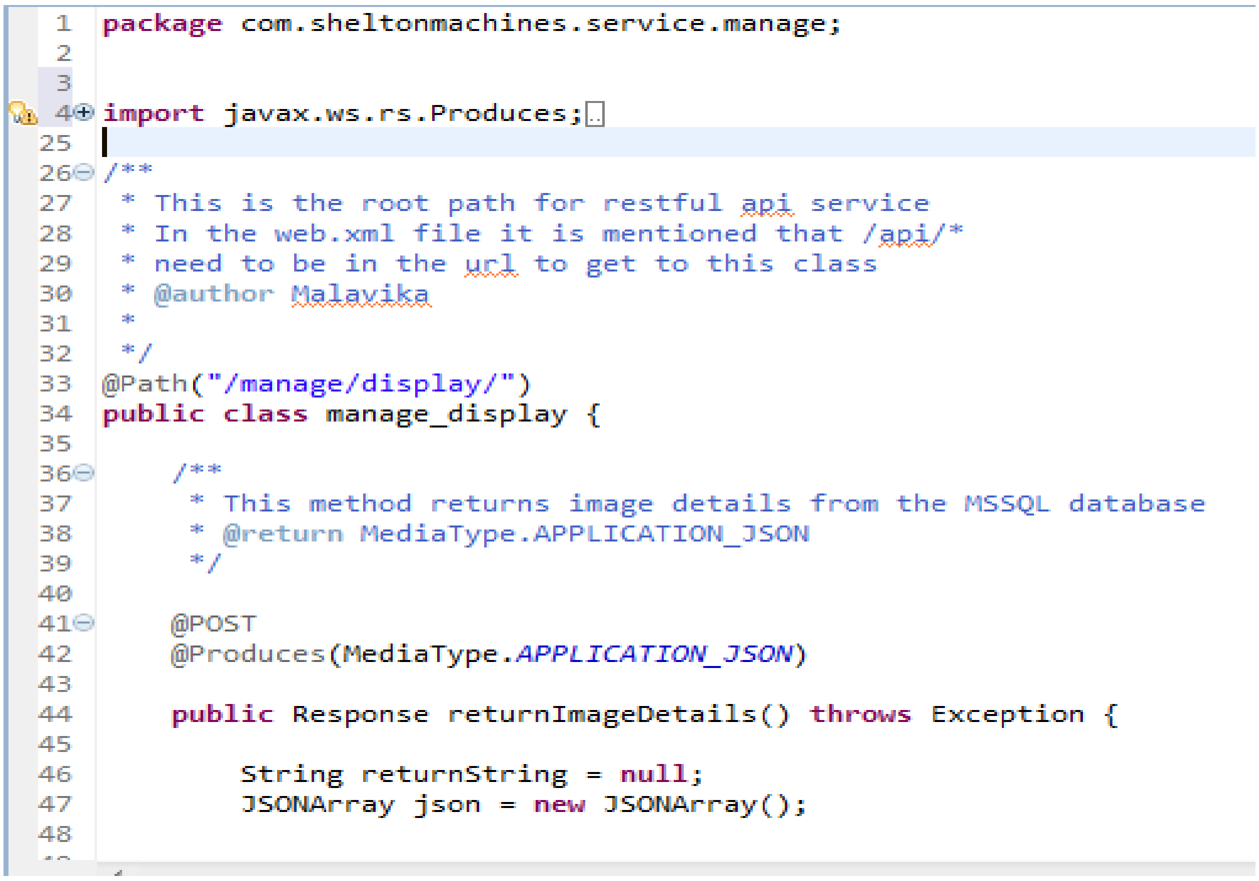
☐ Generate comments

Figure 5d. Creating root resource class.



```
Java Resources
  src
    com.sheltonmachines.service.dao
      SchemaSqlDB.java
      SqlDB.java
    com.sheltonmachines.service.mana
      manage_display.java
    com.sheltonmachines.service.util
      ToJSON.java
```

Figure 5f: REST Service Package structure



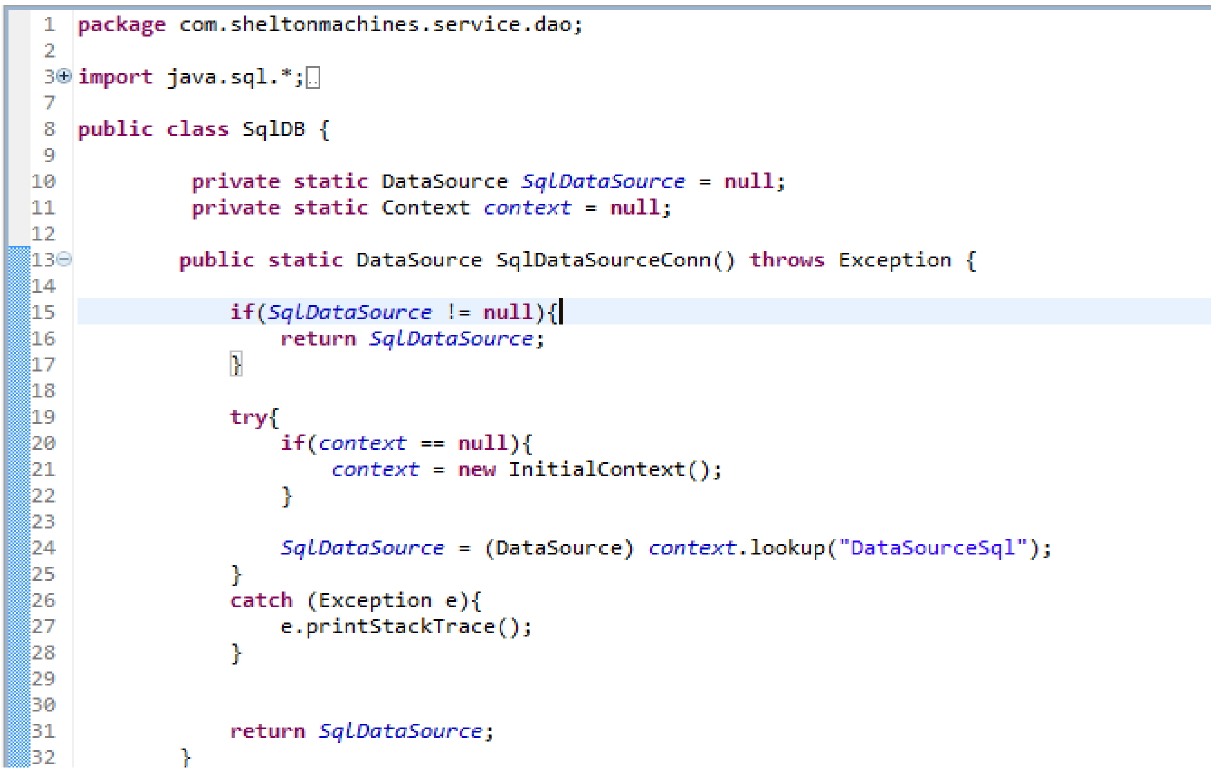
```java
package com.sheltonmachines.service.manage;


import javax.ws.rs.Produces;

/**
 * This is the root path for restful api service
 * In the web.xml file it is mentioned that /api/*
 * need to be in the url to get to this class
 * @author Malavika
 *
 */
@Path("/manage/display/")
public class manage_display {

    /**
     * This method returns image details from the MSSQL database
     * @return MediaType.APPLICATION_JSON
     */

    @POST
    @Produces(MediaType.APPLICATION_JSON)

    public Response returnImageDetails() throws Exception {

        String returnString = null;
        JSONArray json = new JSONArray();

```

Figure 5e: Code snippet of resource class

The method retrieves the SQLDataSource of WebLogic which was created as explained in section 3.3 and returns a connection. The code snippet is showed in Figure 7. The method response is the image path to the client as a JSON object. Code snippet of the method doing the SQL queries is shown in Figure 8.

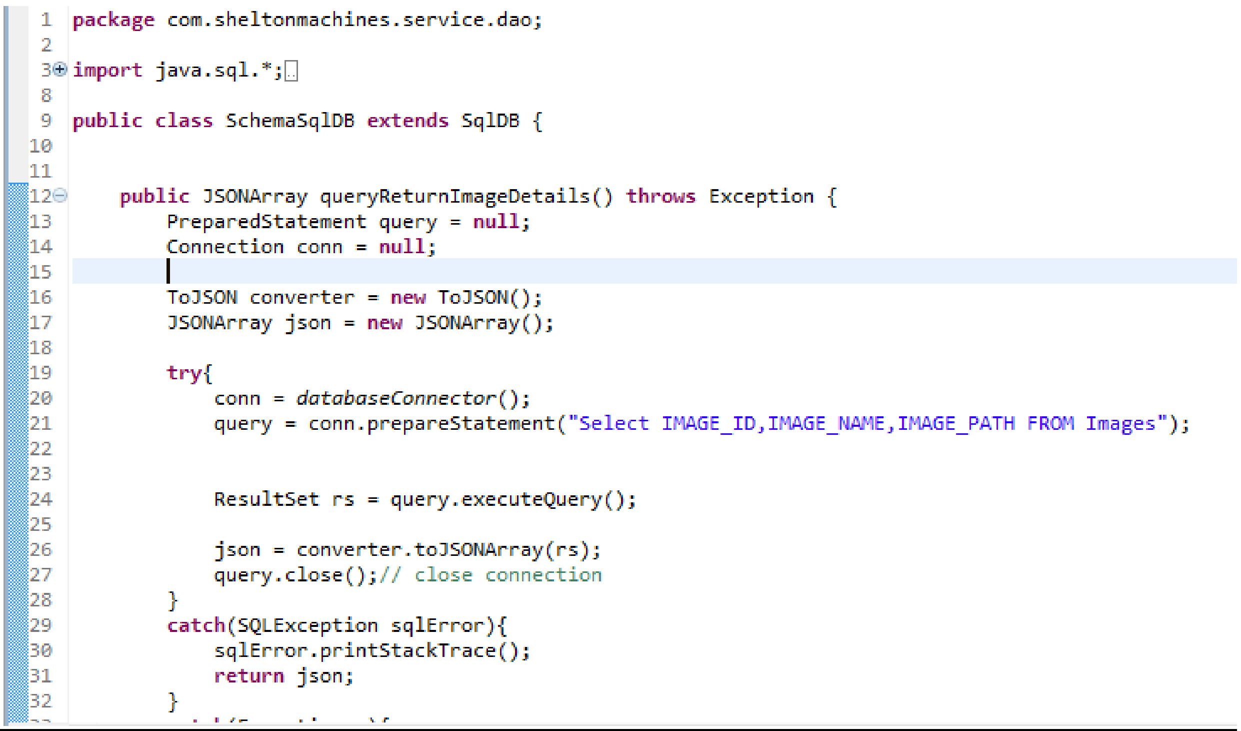
The service is then published to the WebLogic server. The service can be tested by entering the endpoint URL in the browser (for GET methods) or by using a client application. It can also be tested by using REST console apps available for each browser.

The result set of the database query is converted to JSON object using a separate class as mentioned before. The code snippet for the same is shown in figure 10.



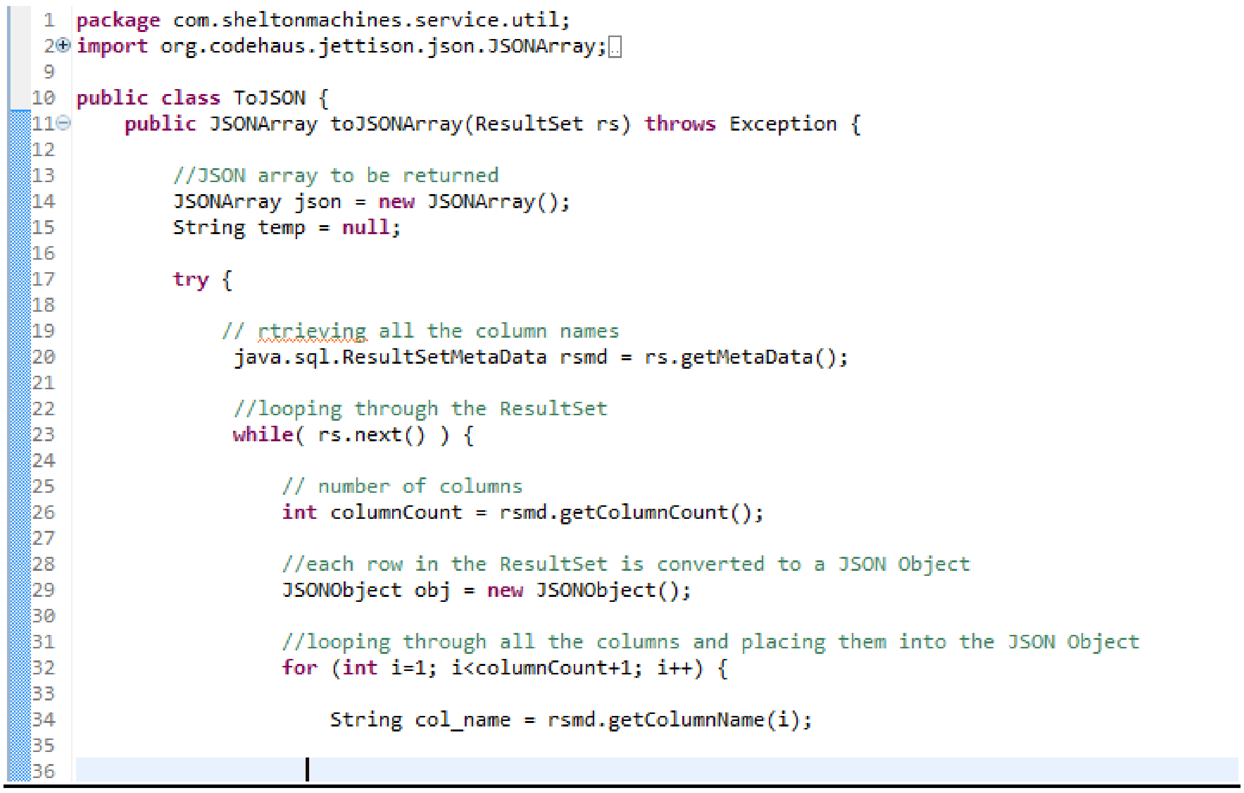
```java
package com.sheltonmachines.service.dao;

import java.sql.*;

public class SqlDB {

         private static DataSource SqlDataSource = null;
         private static Context context = null;

        public static DataSource SqlDataSourceConn() throws Exception {

            if(SqlDataSource != null){
                return SqlDataSource;
            }

            try{
                if(context == null){
                    context = new InitialContext();
                }

                SqlDataSource = (DataSource) context.lookup("DataSourceSql");
            }
            catch (Exception e){
                e.printStackTrace();
            }


            return SqlDataSource;
        }
```

Figure 7: Code snippet of the Java method to retrieve the DataSource created in WebLogic



```java
package com.sheltonmachines.service.dao;

import java.sql.*;

public class SchemaSqlDB extends SqlDB {


    public JSONArray queryReturnImageDetails() throws Exception {
        PreparedStatement query = null;
        Connection conn = null;

        ToJSON converter = new ToJSON();
        JSONArray json = new JSONArray();

        try{
            conn = databaseConnector();
            query = conn.prepareStatement("Select IMAGE_ID,IMAGE_NAME,IMAGE_PATH FROM Images");


            ResultSet rs = query.executeQuery();

            json = converter.toJSONArray(rs);
            query.close();// close connection
        }
        catch(SQLException sqlError){
            sqlError.printStackTrace();
            return json;
        }
```

Figure 8: Code snippet of the method which performs the SQL query to the database



```
package com.sheltonmachines.service.util;
import org.codehaus.jettison.json.JSONArray;

public class ToJSON {
    public JSONArray toJSONArray(ResultSet rs) throws Exception {

        //JSON array to be returned
        JSONArray json = new JSONArray();
        String temp = null;

        try {

            // rtrieving all the column names
             java.sql.ResultSetMetaData rsmd = rs.getMetaData();

             //looping through the ResultSet
             while( rs.next() ) {

                 // number of columns
                 int columnCount = rsmd.getColumnCount();

                 //each row in the ResultSet is converted to a JSON Object
                 JSONObject obj = new JSONObject();

                 //looping through all the columns and placing them into the JSON Object
                 for (int i=1; i<columnCount+1; i++) {

                     String col_name = rsmd.getColumnName(i);

```

Figure 10: Code snippet for converting the database ResultSet to JSON Object

Chapter 6

6. Implementation of REST Service using ASP.NET

[in progress]

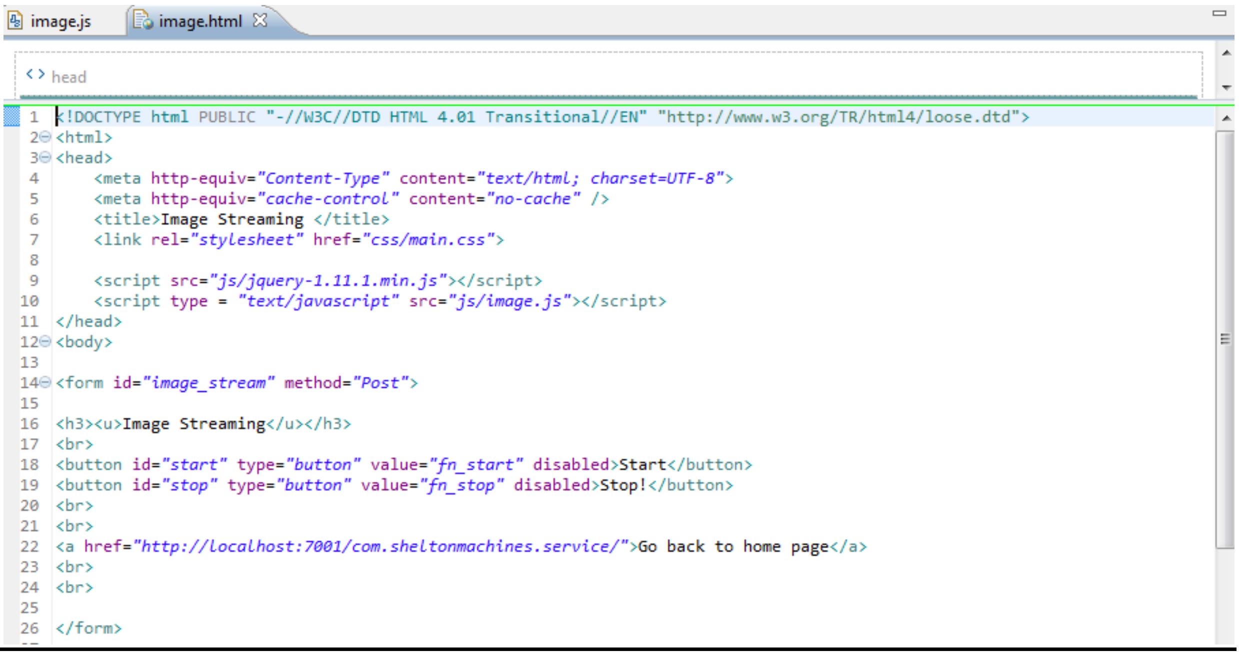
Chapter 7

7. Consuming Java RESTful Web Service by JQuery Client

The client written in JQuery will consume the Jersey-based RESTful web service created in chapter 4.

Scenario i) Live streaming of images in its original size at client side. Images are of size 4MB stored at a location both client and server can access. Image paths are stored in the database.

The RESTful web service implemented in chapter 4 returns the image details as a JSON Object. Here the client makes Ajax calls using JQuery library. The code snippet is shown in figure 12. Specify the REST service URL in the Ajax call and the request is send to the server. The server then responds with the image details. On success of the Ajax call, the images are displayed in the browser.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta http-equiv="cache-control" content="no-cache" />
    <title>Image Streaming </title>
    <link rel="stylesheet" href="css/main.css">

    <script src="js/jquery-1.11.1.min.js"></script>
    <script type = "text/javascript" src="js/image.js"></script>
</head>
<body>

<form id="image_stream" method="Post">

<h3><u>Image Streaming</u></h3>
<br>
<button id="start" type="button" value="fn_start" disabled>Start</button>
<button id="stop" type="button" value="fn_stop" disabled>Stop!</button>
<br>
<br>
<a href="http://localhost:7001/com.sheltonmachines.service/">Go back to home page</a>
<br>
<br>

</form>
```

Figure 11: Code snippet for image.html

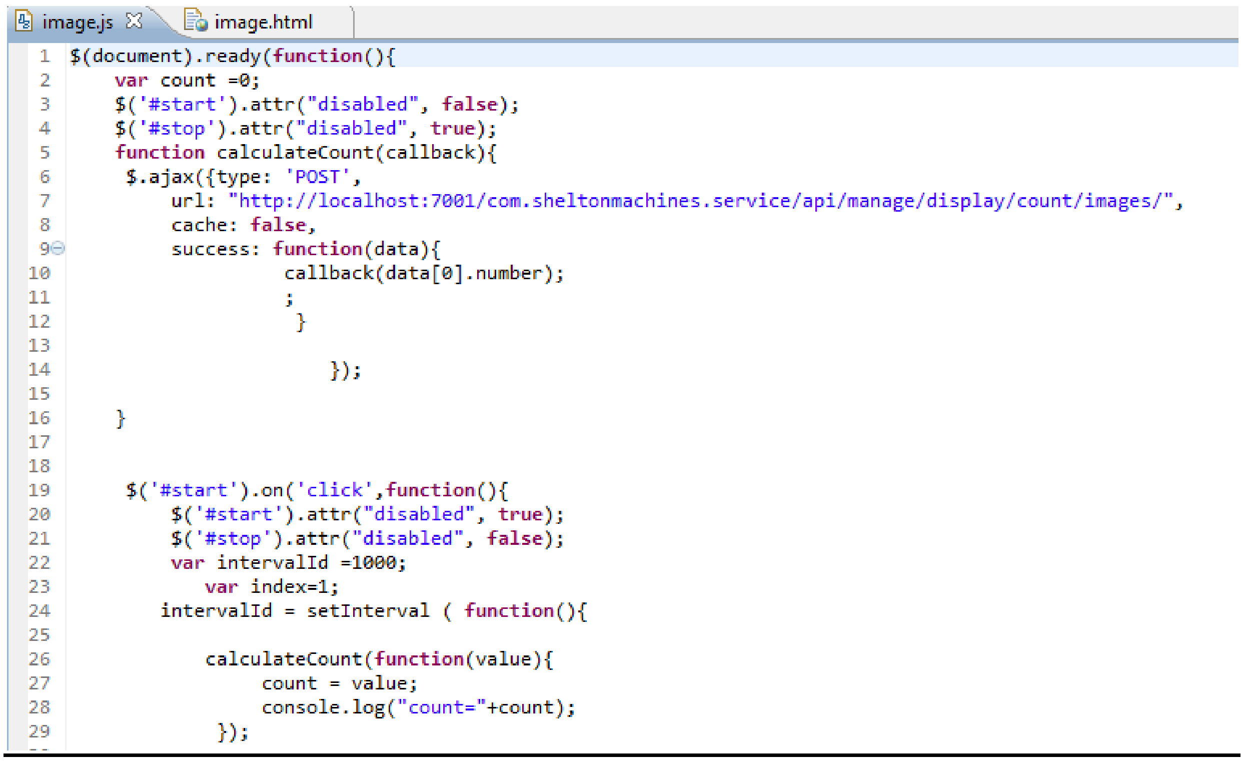


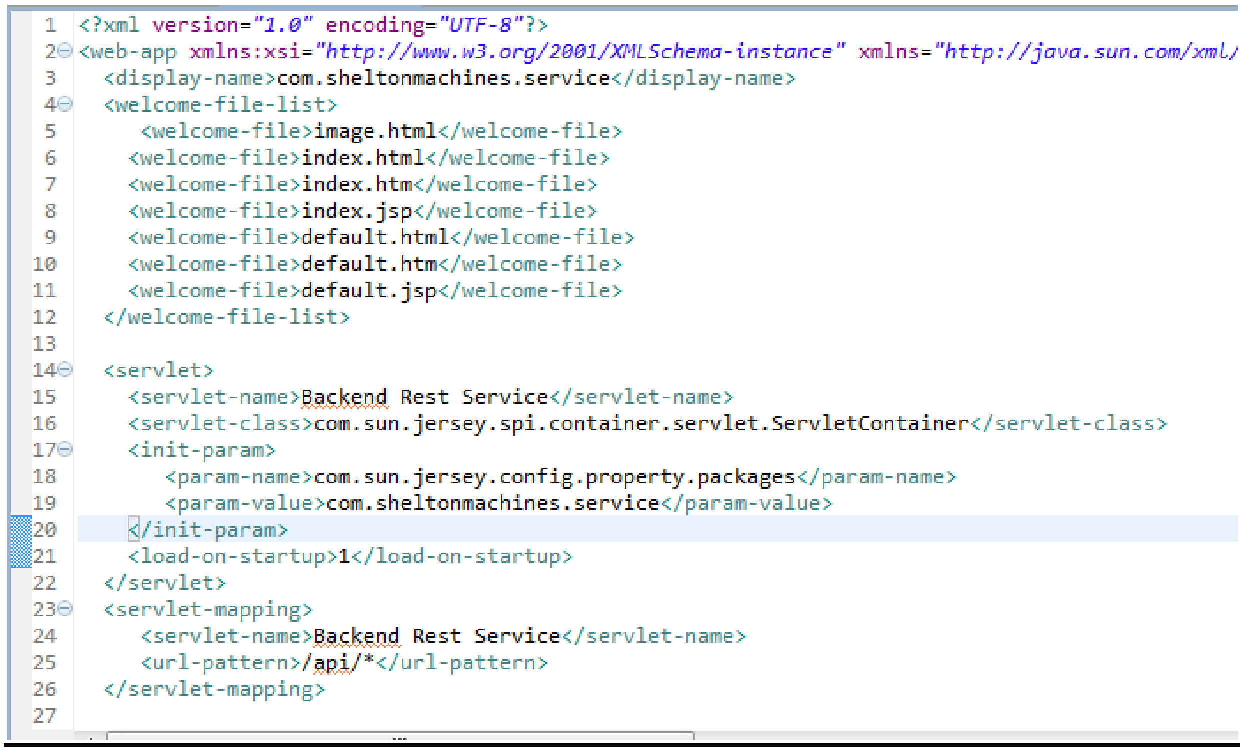
image.js
image.html

```
$(document).ready(function(){
    var count =0;
    $('#start').attr("disabled", false);
    $('#stop').attr("disabled", true);
    function calculateCount(callback){
     $.ajax({type: 'POST',
         url: "http://localhost:7001/com.sheltonmachines.service/api/manage/display/count/images/",
         cache: false,
         success: function(data){
                   callback(data[0].number);
                   ;
                    }

                      });

    }


     $('#start').on('click',function(){
         $('#start').attr("disabled", true);
         $('#stop').attr("disabled", false);
         var intervalId =1000;
            var index=1;
        intervalId = setInterval ( function(){

            calculateCount(function(value){
                 count = value;
                 console.log("count="+count);
             });
```

Figure 12: Code snippet of image.js, the javascript file consuming the RESTful web service

When an empty ‘Dynamic Web Project’ is created in Eclipse IDE for Java EE, a template for web application deployment descriptor (web.xml) gets generated in the WEB-INF directory of the web application project. It provides the configuration and deployment information for the web components of a specific web application (eg: servlets, servlet mappings, URL mapping) .When we hit the client URL , the starting point will be this web.xml. It acts a gateway to the application.

The servlet needs to be configured manually. A servlet can have multiple servlet-mapping. The web.xml for this project is shown in figure 13. When a request comes for a particular URL pattern, the servlet container matches the servlet-name with the url-pattern, finds the corresponding servlet-path and pass the control to the service.



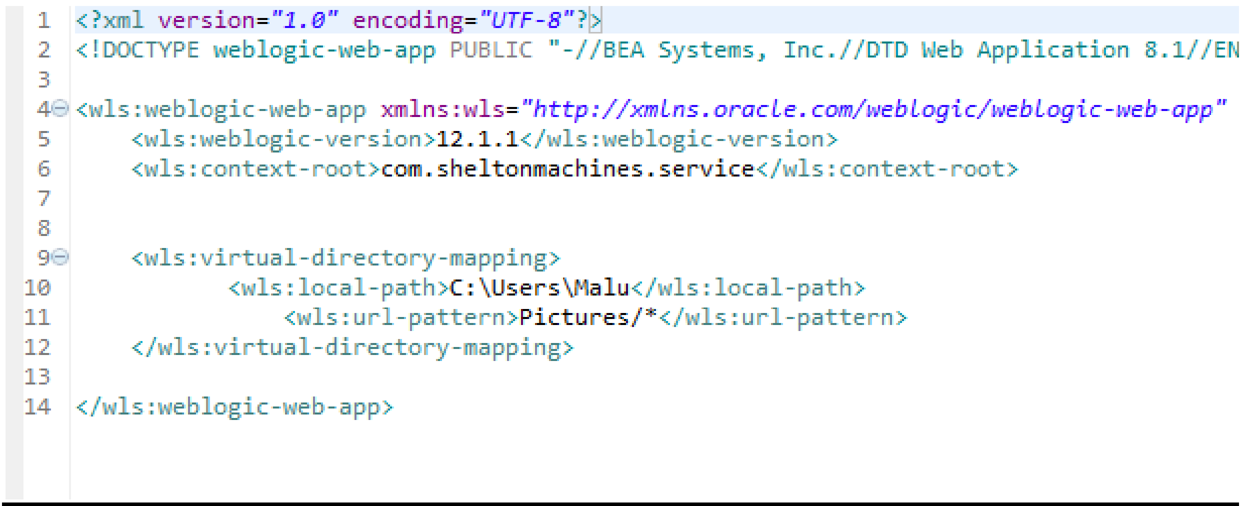
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/
  <display-name>com.sheltonmachines.service</display-name>
  <welcome-file-list>
     <welcome-file>image.html</welcome-file>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>Backend Rest Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
       <param-name>com.sun.jersey.config.property.packages</param-name>
       <param-value>com.sheltonmachines.service</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
     <servlet-name>Backend Rest Service</servlet-name>
     <url-pattern>/api/*</url-pattern>
  </servlet-mapping>

```

Figure 13: Code snippet of Web Application deployment descriptor (web.xml)

Web application deployment descriptor elements for all the application in a user domain are defined in weblogic.xml file which also resides in WEB-INF directory of the web application (figure 14). Lines 9 to 11 in figure 14 shows virtual directory mapping.

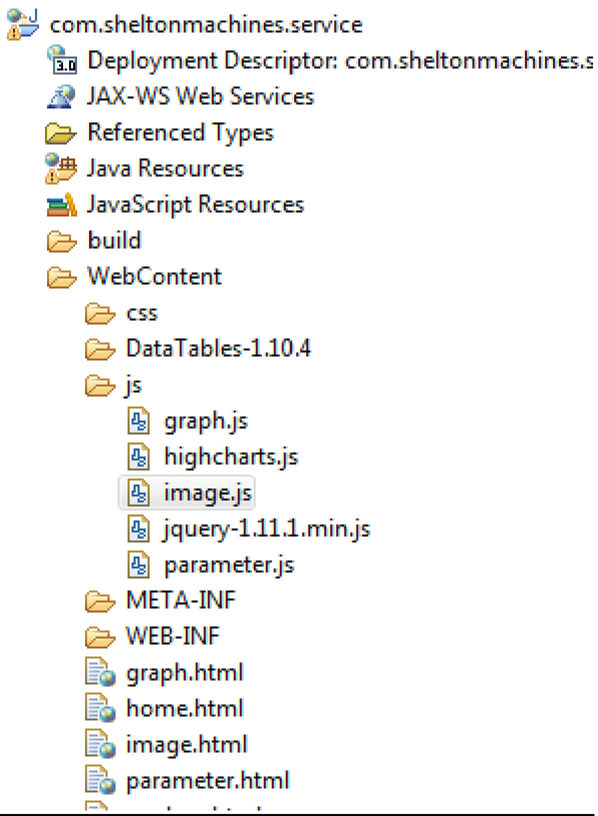
“Use the *virtual-directory-mapping* element to specify document roots other than the default document root of the Web application for certain kinds of requests, such as image requests” [13]. It can be used to include images from another location other than the physical directory that is mapped to application’s root virtual directory. If we specify a directory name, it gets mapped to a physical directory on a local or remote server.

The Client package structure is shown in figure 15. All JavaScript files are in a seperate folder. Table 2 shows the components and how they are organised in the folder structure.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application 8.1//EN

<wls:weblogic-web-app xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app"
    <wls:weblogic-version>12.1.1</wls:weblogic-version>
    <wls:context-root>com.sheltonmachines.service</wls:context-root>


    <wls:virtual-directory-mapping>
           <wls:local-path>C:\Users\Malu</wls:local-path>
               <wls:url-pattern>Pictures/*</wls:url-pattern>
    </wls:virtual-directory-mapping>

</wls:weblogic-web-app>
```

Figure 14: code snippet of Web application deployment descriptor elements (weblogic.xml)

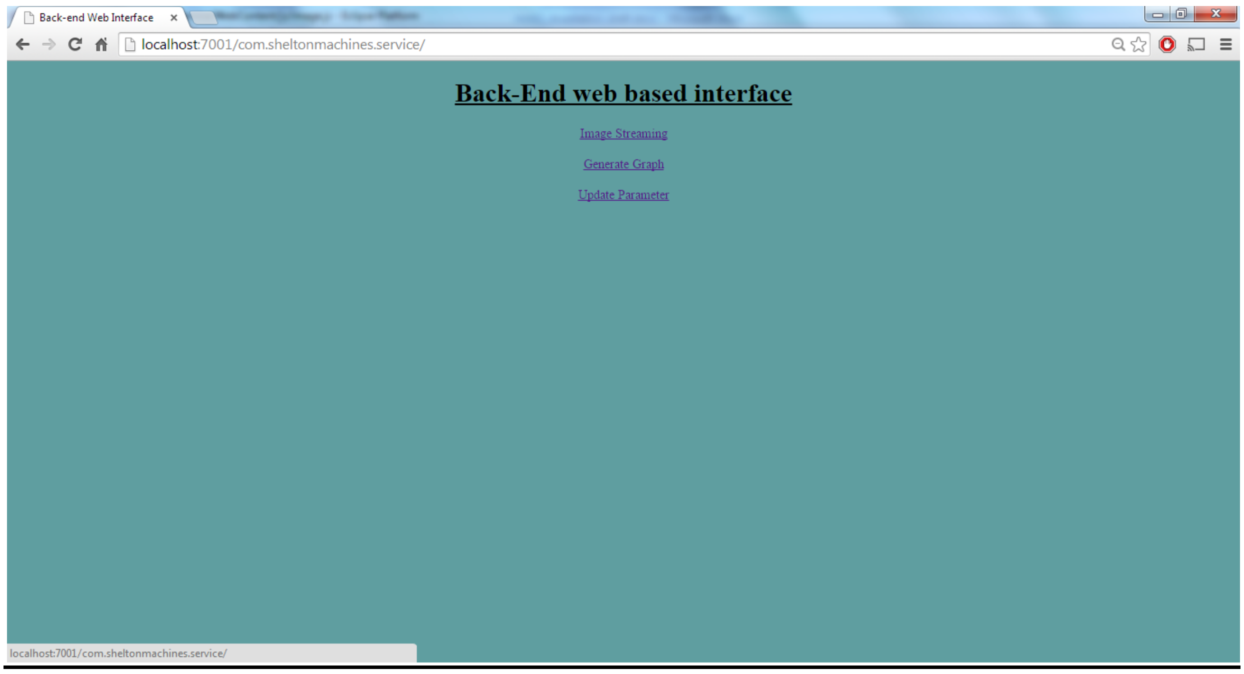


```
com.sheltonmachines.service
  Deployment Descriptor: com.sheltonmachines.s
  JAX-WS Web Services
  Referenced Types
  Java Resources
  JavaScript Resources
  build
  WebContent
    css
    DataTables-1.10.4
    js
      graph.js
      highcharts.js
      image.js
      jquery-1.11.1.min.js
      parameter.js
    META-INF
    WEB-INF
    graph.html
    home.html
    image.html
    parameter.html
```

Figure 15: Client Package

Tier	Layers	Technology used	File Locations
Client Tier	Presentation Layer	HTML Javascript Javascript libraries- jQuery, HighCharts,DataTables	/Webcontent /WebContent/css /WebContent/js
Application Server Tier	Business Layer Application Layer	JAX-RS Java	/src/com.sheltonmachines.service.manage /src/com.sheltonmachines.service.util
Database Tier	Data Access Layer	Java SQL	/src/com.sheltonmachines.service.dao

Table2: Organisation of architecture components in the folder structure


Back-end Web Interface
localhost:7001/com.sheltonmachines.service/

Back-End web based interface

Image Streaming

Generate Graph

Update Parameter

localhost:7001/com.sheltonmachines.service/

Client home page

Please find the client page for image streaming in figure 16. The client-server works as explained below:

1) The user pushes 'Start' button to start the image streaming. On clicking, the 'start' button will get disabled and 'Stop' button will get enabled.

2) The server will then pick up the first image file path from the database and sends it to the client which then displays on its browser.

3) The server will send the next image file path after a delay. This step will continue until the user clicks 'Stop' button or the server reaches the end of the image list. This would result in a 'live' changing of images and would look similar to streaming.

4) Meanwhile, the server keeps checking the database every 1500 milliseconds whether a change has been occurred or not. If any change occurs, it will get reflected at the client side without a refresh.

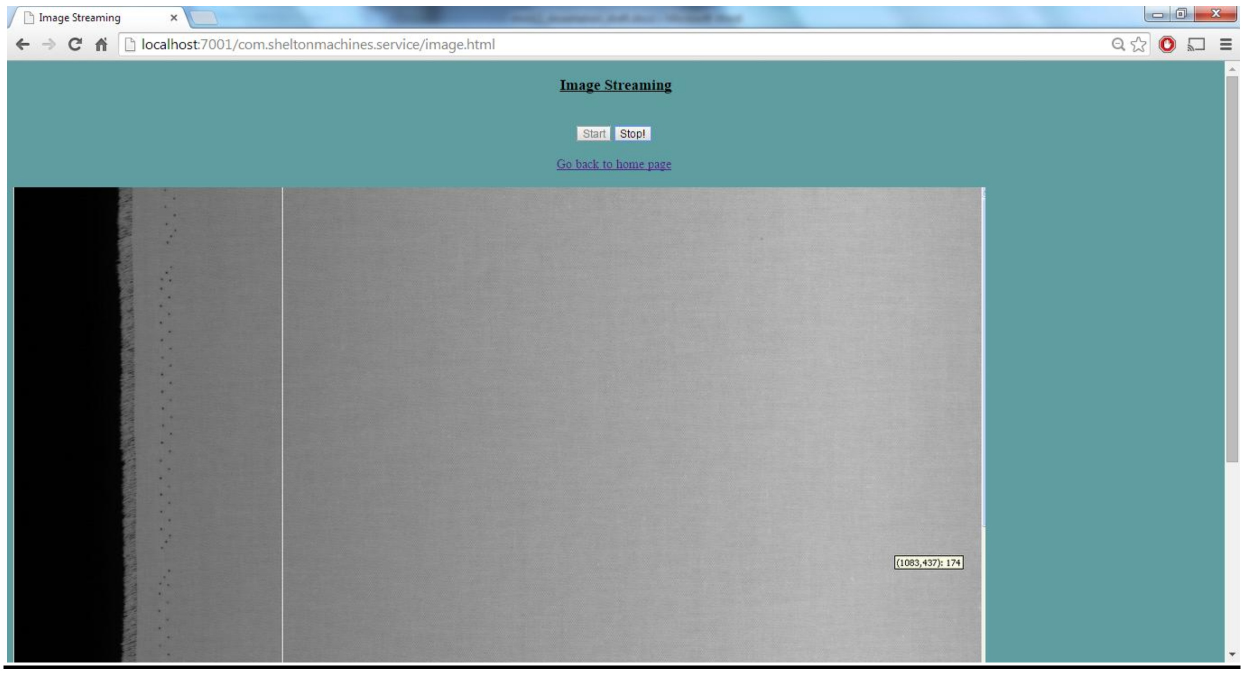


Figure 15: Client page for image streaming

Scenario ii) Update parameters in the database on a button push at client side. On successful update, display a message at client side which is send from the server.

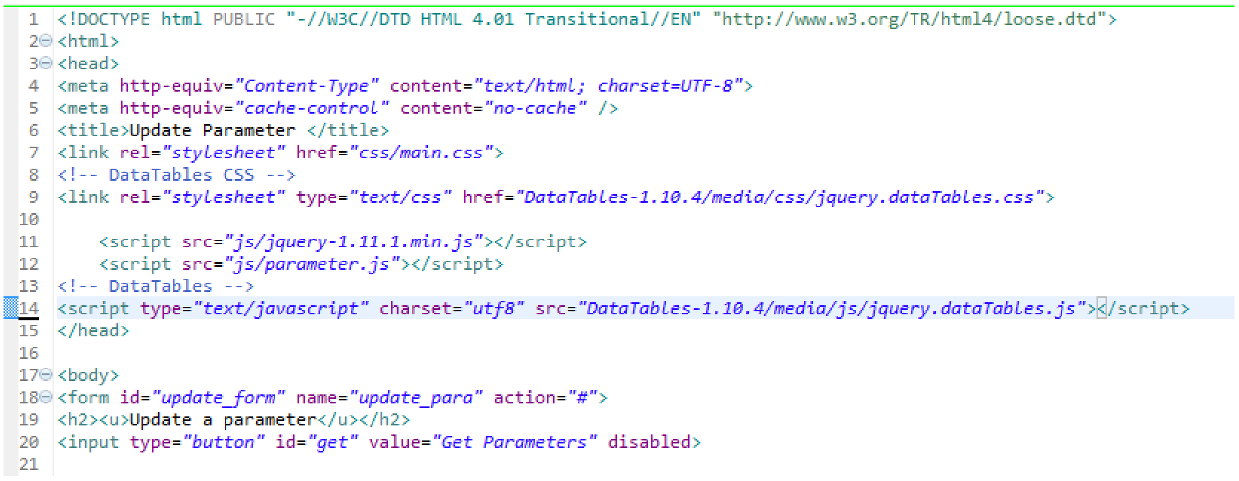
The following table plug-in for JQuery Javascript library is added for the implementation of parameter table.

- Data Tables V1.10.4 - http://www.datatables.net/download/

When the user clicks on 'Get Parameters' button, the client make an Ajax call to the server and then the server return all the parameters in the database which are then displayed on the browser as shown in figure 18.

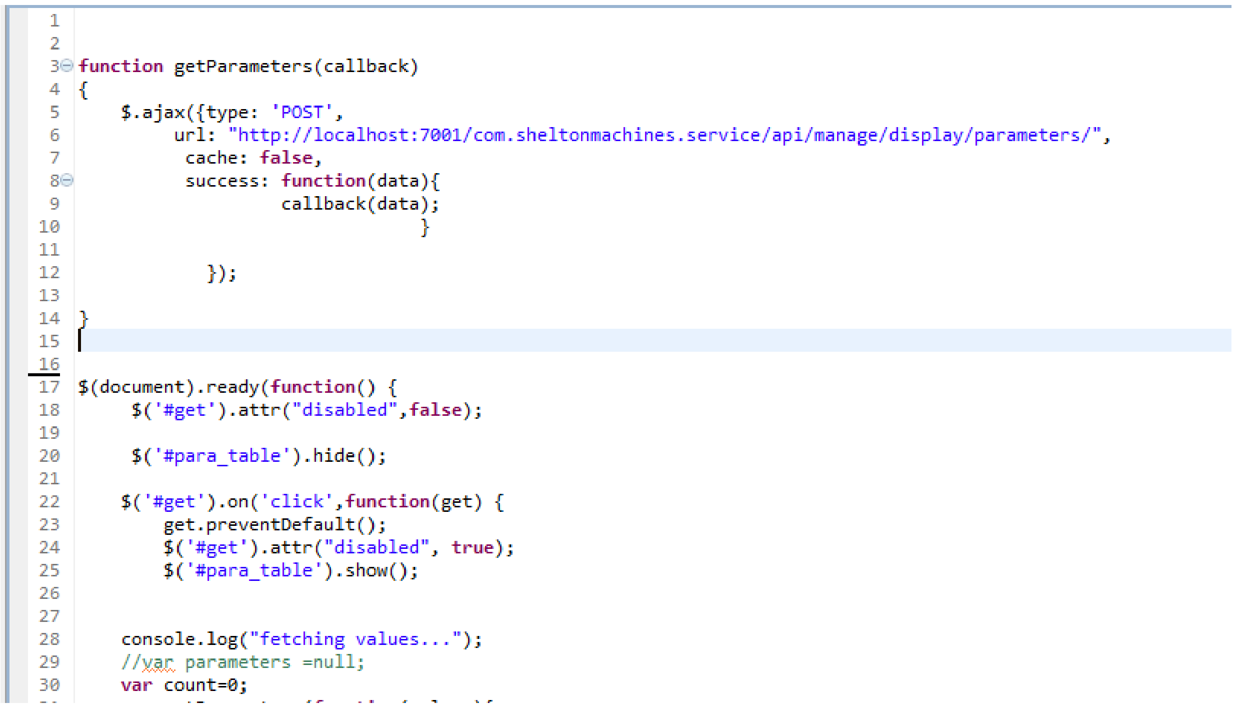
The parameters can be sorted, filtered, edited and can be saved back to the database. When the user clicks on 'Save' button, the details of the edited row is send back to the server and then the server updates the database accordingly.

The html code snippet is shown in figure 16 and the JavaScript code is shown in figure 17.



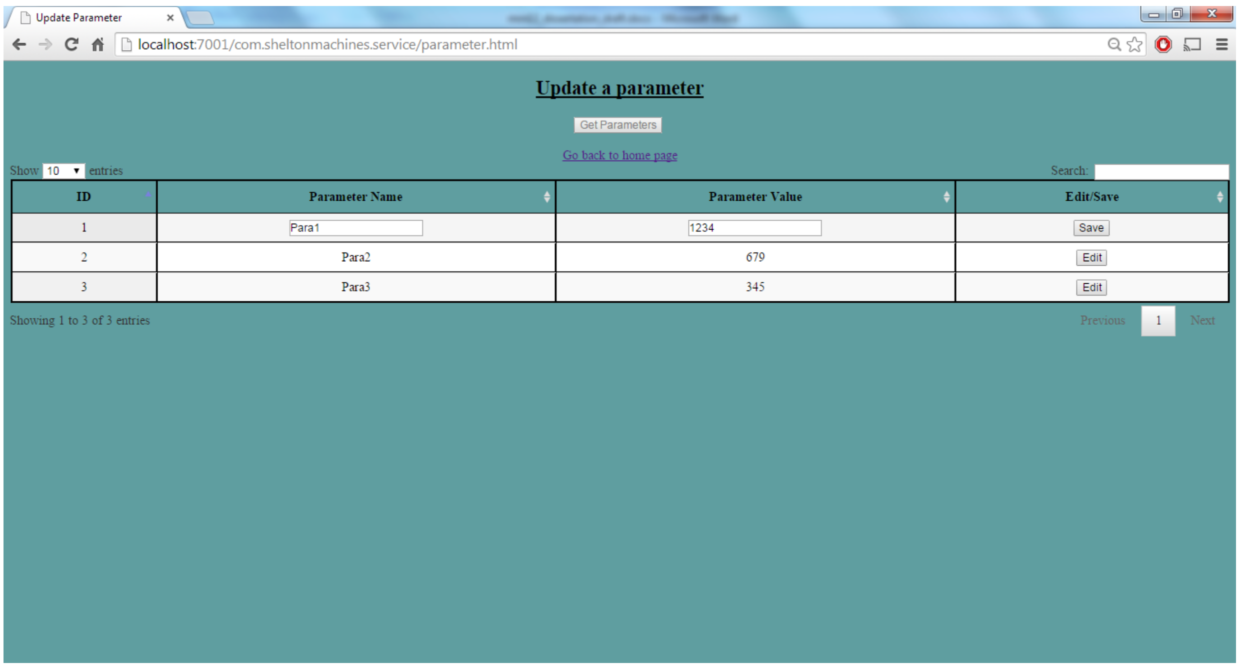
```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="cache-control" content="no-cache" />
<title>Update Parameter </title>
<link rel="stylesheet" href="css/main.css">
<!-- DataTables CSS -->
<link rel="stylesheet" type="text/css" href="DataTables-1.10.4/media/css/jquery.dataTables.css">

    <script src="js/jquery-1.11.1.min.js"></script>
    <script src="js/parameter.js"></script>
<!-- DataTables -->
<script type="text/javascript" charset="utf8" src="DataTables-1.10.4/media/js/jquery.dataTables.js"></script>
</head>

<body>
<form id="update_form" name="update_para" action="#">
<h2><u>Update a parameter</u></h2>
<input type="button" id="get" value="Get Parameters" disabled>

```

Figure 16: Code snippet for parameter.html



```javascript


function getParameters(callback)
{
    $.ajax({type: 'POST',
           url: "http://localhost:7001/com.sheltonmachines.service/api/manage/display/parameters/",
            cache: false,
            success: function(data){
                     callback(data);
                                   }


              });

}


$(document).ready(function() {
     $('#get').attr("disabled",false);

     $('#para_table').hide();

    $('#get').on('click',function(get) {
        get.preventDefault();
        $('#get').attr("disabled", true);
        $('#para_table').show();


    console.log("fetching values...");
    //var parameters =null;
    var count=0;
```

Figure 17: code snippet for parameter.js



localhost:7001/com.sheltonmachines.service/parameter.html

Update a parameter

Get Parameters

Go back to home page

Show 10 entries | Search:

ID	Parameter Name	Parameter Value	Edit/Save
1	Para1	1234	Save
2	Para2	679	Edit
3	Para3	345	Edit

Showing 1 to 3 of 3 entries | Previous 1 Next

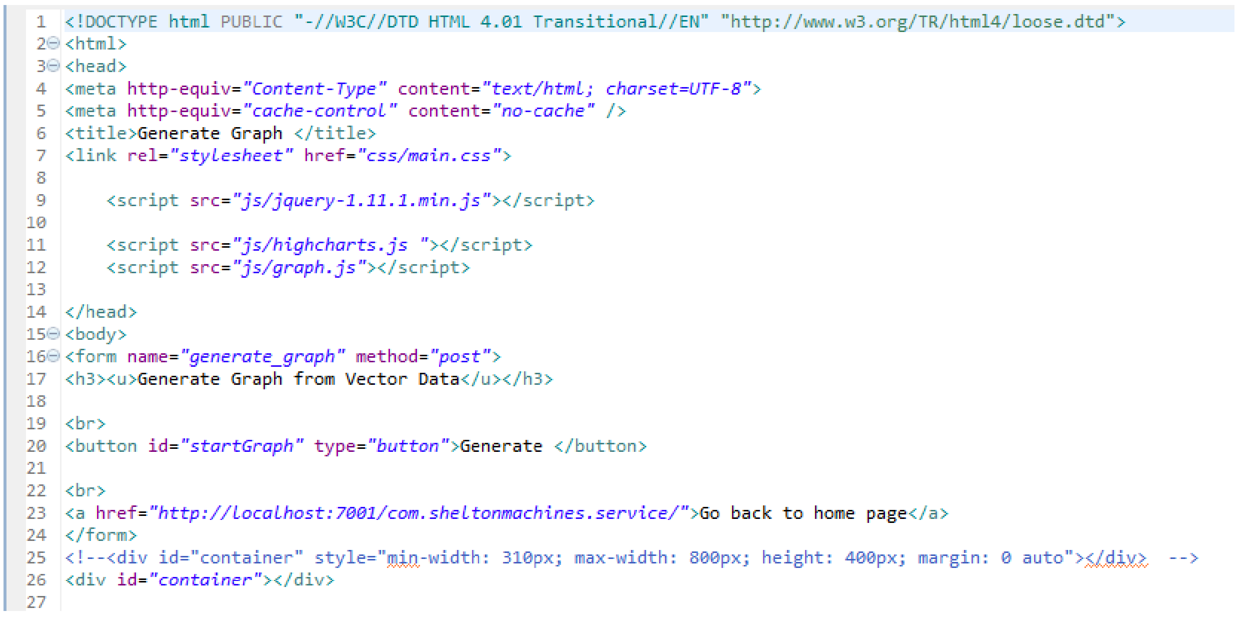
Figure 18: Client Page for Parameter Update

Scenario iii) Display graph at client side from a list of vector data

To achieve this, a JavaScript charting library called Highcharts v4.0.4 downloaded from the below link have been used.

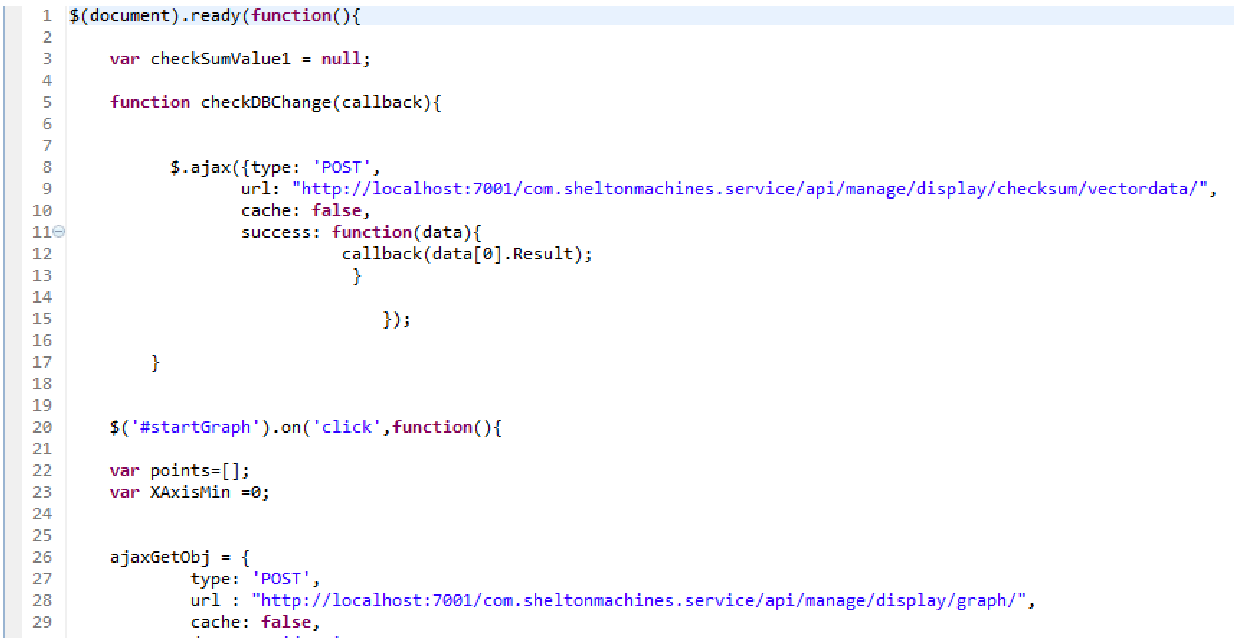
- Highcharts V4.0.4 - http://www.highcharts.com/download

The html code snippet is shown in figure 19 and the JavaScript code is shown in figure 20. The resulting client page is shown in figure 21.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="cache-control" content="no-cache" />
<title>Generate Graph </title>
<link rel="stylesheet" href="css/main.css">

    <script src="js/jquery-1.11.1.min.js"></script>

    <script src="js/highcharts.js "></script>
    <script src="js/graph.js"></script>

</head>
<body>
<form name="generate_graph" method="post">
<h3><u>Generate Graph from Vector Data</u></h3>

<br>
<button id="startGraph" type="button">Generate </button>

<br>
<a href="http://localhost:7001/com.sheltonmachines.service/">Go back to home page</a>
</form>
<!--<div id="container" style="min-width: 310px; max-width: 800px; height: 400px; margin: 0 auto"></div>  -->
<div id="container"></div>
```

Figure 19: Code snippet for graph.html



```
$(document).ready(function(){

    var checkSumValue1 = null;

    function checkDBChange(callback){


          $.ajax({type: 'POST',
                 url: "http://localhost:7001/com.sheltonmachines.service/api/manage/display/checksum/vectordata/",
                 cache: false,
                 success: function(data){
                           callback(data[0].Result);
                           }

                              });

        }


    $('#startGraph').on('click',function(){

    var points=[];
    var XAxisMin =0;


    ajaxGetObj = {
            type: 'POST',
            url : "http://localhost:7001/com.sheltonmachines.service/api/manage/display/graph/",
            cache: false,
```

Figure 20: Code snippet for graph.js

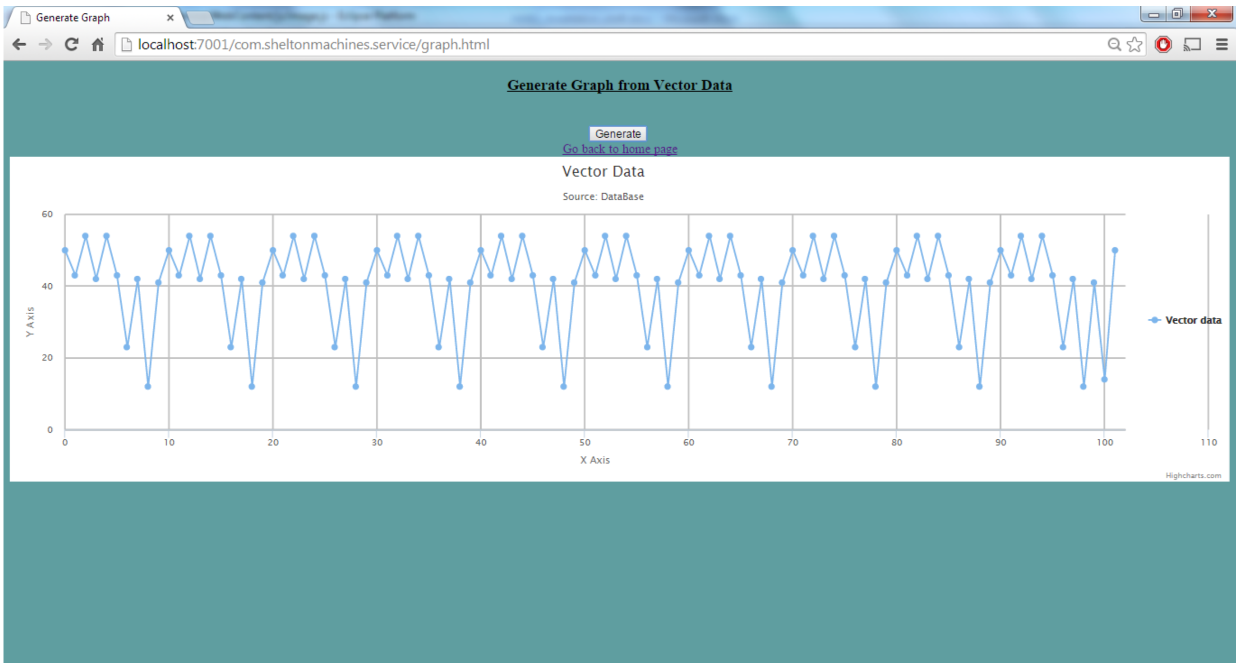


Figure 21: Client page for graph display

Chapter 8

8. Consuming ASP.NET RESTful Web Service by JQuery Client

In progress..

Chapter 9

9. Project Management

In progress....

Chapter 10

10. Evaluation of the solution

There are two subsections for this chapter:

1. Comparative Analysis of the solutions
2. Improvements and future work

Having implemented the web services in Java and C#, this section of the report shall compare the results against each other and draw conclusions on which framework performs better under the given circumstances.

The client is written in the same language. So the comparison lies at examining the performance of the jQuery client while accessing the Java and C# RESTful services.

Given the time scale of the project, only the image streaming scenario is taken into consideration for the comparison purpose.

Comparative analysis of the solutions:

<u>Browser Support:</u>

	IE	Google Chrome	Mozilla FireFox	Opera
Java REST	✓	✓	✓	
.Net REST				

Table 3: Browser compatability

Look and Feel:

	Background	CSS	UI Elements
Java REST			
.Net REST			

Table 4: A comparison on look and feel outcomes

Device Support:

	Windows OS	Mac-iOS	Ubuntu-Linux OS
Java REST	✓		
.Net REST			

Table 5: Devices supported

Response time against amount of data:

A graph with log of time (in milliseconds on Y axis) and the image size in X axis

Response time against database change:

A graph with log of time (in milliseconds on Y axis) and time at which database is changed

Security Records:

In progress....

Other factors:

	Java REST	.Net REST
Effort of cost		
Learning effort		
Maintainability		

Proprietary products		

Table 6:

References:

[1] “JQuery UI 1.8. The User Interface Library for JQuery”,by Dan Wellman

[2]”Apache CXF Web Service Development-Develop and deploy SOAP and RESTful web services”, by Naveen Balani and Rajeev Hathi,PACKT PUBLISHING, BIRMINGHAM-MUMBAI

[3]”Foundations for Efficient Web service Selection”by Qi Yu, Athman Bouguettaya, SPRINGER US, 2010

[4]”Web Services & SOA Principles and Technology”, by Michael.P.Papazoglou, SECOND EDITION,PEARSON

[5] Part 1: Introduction to Jersey—a Standard, Open Source REST Implementation [online], Accessed date [27/09/2014]

Available at: http://www.oracle.com/technetwork/articles/javaee/jersey-part1-159891.html

[6] “RESTful Web Services with Jersey” by Scott Leberknight , Software Developer at Near Infinity Corporation, Jun 07, 2014

[7] RESTful Web Services [online], [Accessed date: [27/09/2014]

Available at: http://www.oracle.com/technetwork/articles/javase/index-137171.html

[8] RESTful Web services: The basics [online], [Accessed date: 14/07/2014]

Abailable at: http://www.ibm.com/developerworks/webservices/library/ws-restful/

[9] Web Services based on SOAP and REST Principles [online], [Accessed date: 25/09/2014]

Available at : http://www.ijsrp.org/research-paper-0513/ijsrp-p17115.pdf

[10] “Introduction to Client Server Computing” by Yadav, Subash Chandra, Singh, Sanjay Kumar, New Age International, 2009

[11] “REST vs SOAP”, By Dion Hinchcliffe ,April 2005[Accessed date: 19/07/2014]

Available at: http://zgia.net/?p=30

[12] **Oracle WebLogic Server** [online], [Accessed date: 27/09/2014]
Available at:
http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

[13] **virtual-directory-mapping** [online], [Accessed date: 29/09/2014]
Available at:
http://docs.oracle.com/cd/E11035_01/wls100/webapp/weblogic_xml.html#wp1039396

[14] “ Programming PHP”, 3rd edition, By Kevin Tatroe, Peter MacIntyre, RasmusLerdorf (Page: 1)

[15] “Learning JQuery”, By Jonathan Chaffer, Karl Swedberg

[16] Category: Ajax [online], [Accessed date: [29/09/2014]

Available at: http://api.jquery.com/category/ajax/

[17]” jQuery:Advantages and Disadvantages” [online], [Accessed date: [29/09/2014]

Available at: http://www.jscripters.com/jquery-disadvantages-and-advantages/

[18] “Get Started with ASP.NET” [online], [Accessed date: [29/09/2014]

Available at: http://www.asp.net/get-started

[19] What is Highcharts? [online], [Accessed date: 27/10/2014]

Available at: http://www.highcharts.com/products/highcharts

[20] https://ffeathers.wordpress.com/2014/02/16/api-types/ [07/12/14]

[21] “RESTful Java Web Services : Master Core REST Concepts and Create RESTful Web Services in Java” *by* Sandoval, Jose; Packt Publishing, Niovember 2009

[22] Java EE development with Eclipse, by Deepak Vohra, Packt Publishing,December 2012

[23] DataTables Table plug-in for jQuery [online], [Accessed Date: 08/09/14]

Available at: http://datatables.net/