University of Leicester

Department Of Computer Science
MSc. Web Applications and Services
CO7501 Individual Project – Dissertation

AUTHOR

Student Name	Email
MALAVIKA REGHUNATHAN NAIR	**mrn12@student.le.ac.uk**

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

Project Supervisor: Prof. Dr. Reiko Heckel

Second Marker: Dr. Ruzanna Chitchyan

Date of Submission: 16 January 2015

ABSTRACT

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. It begins with analyzing the current trends of web technologies and, concludes with the feasibility study of REST architecture to develop the web service API in different modeling languages.

Acknowledgement

I would personally like to thank my project supervisor Dr. Reiko Heckel for his relentless support and feedback throughout this project. In addition I also like to thank my project assessor Dr. Ruzanna Chitchyan for providing me valuable feedbacks during various progress meeting stages. I also take this opportunity to thank the CTO of Shelton Machines Ltd., Dr. Mike Millman for his continuous feedback and encouragement in formulating requirements, setting up the SQL server and for arranging a copy of the Windows 7 OS for the purpose of the project.

Last but not the least I like to thank my family for their continued support and encouragement during my MSc. tenure.

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

SML   Shelton Machines Ltd

CLR   Common Language Runtime

Table of Contents:

1. INTRODUCTION

This dissertation will provide a solution in terms of web technologies for designing a back-end web based user interface for the 'Shelton WebSpector Surface Inspection System', one of the HiPerVison Projects on Shelton Vision's 'Textile Inspection System'. This is achieved by a thorough research on the available web technologies, implementing few scenarios of the system with selected technologies and a comparison of the solutions. This report will also provide a clear description of the scenarios being solved and its usefulness.

1.1   Aim

'Shelton Machines Ltd' (SML) a company that works on textile inspection systems currently has a desktop based application, 'Shelton WebSpector surface inspection system' 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 which can be accessed from all kind of devices to allow its engineers to interrogate the backend remotely to troubleshoot any faults in identifying defects and adjust parameters.
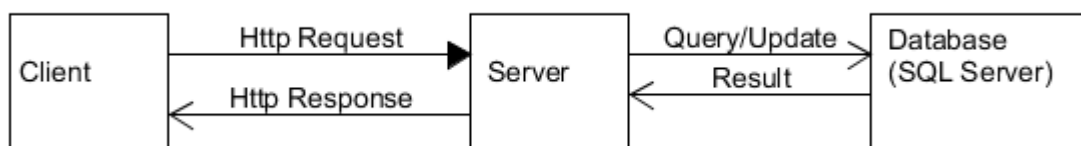
Figure 1a: High-Level Architecture

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 SML, for study purposes only a few core functionalities of the application are taken into consideration.

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 on the Microsoft Windows XP operating system and an 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 SML, who provide 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

As part of a scoping exercise, various discussions were held with SML to arrive at a list of **scenarios** that needs to be implemented which are listed below:

i.   The web application should be capable of live streaming Jpeg images in its original size at the client side. The path for these images should be retrieved asynchronously from the server exploiting a web service API. The image dimensions will be 4096 x 1024 and up to 4MB in size and will be stored at a location where both client and the server have access to.

ii. The application shall display the parameter details from the database when requested, and provide the user the ability to update them. These parameters can be either of the data types – String, Boolean, 32 bit floating or integer.

iii. The web application should be able to generate a graph from a list of vector data. The vector data will contain the axis points required for the graph. The graph should get regenerated upon change in data.

An MS SQL Server should be used to store the data such as image details, parameter details and vector data.

1.3 Project Management

A **Gantt chart** was prepared to manage the allocation time of the project tasks. The schedule started from June 1st 2014 and each column represents end of week. There were regular meetings with the supervisor and SML 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.

Figure 1b shows an excerpt of the Gantt chart.

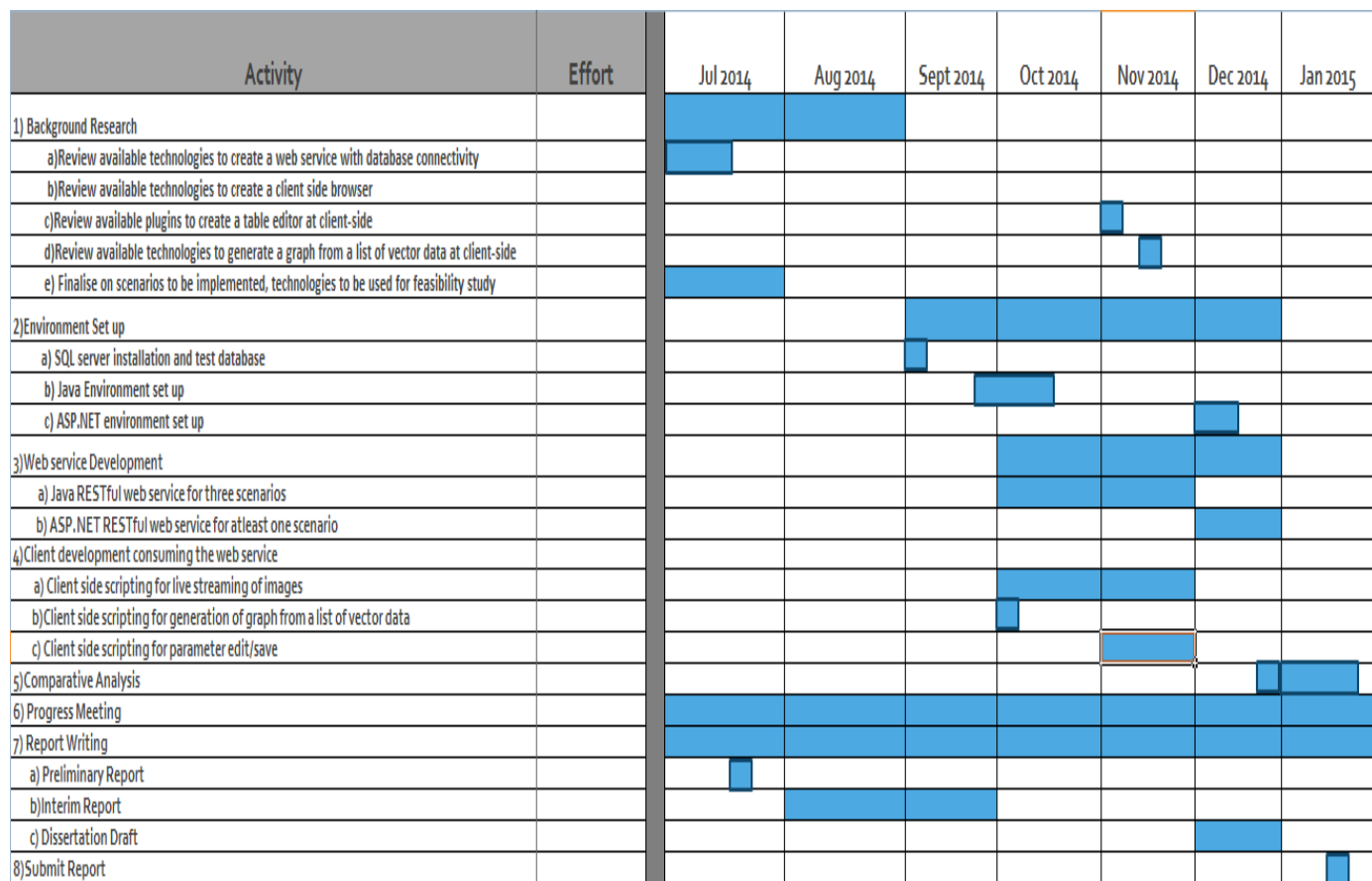| Activity	Effort	Jul 2014	Aug 2014	Sept 2014	Oct 2014	Nov 2014	Dec 2014	Jan 2015
1) Background Research		██	██					
a)Review available technologies to create a web service with database connectivity		██						
b)Review available technologies to create a client side browser								
c)Review available plugins to create a table editor at client-side						██		
d)Review available technologies to generate a graph from a list of vector data at client-side						██		
e) Finalise on scenarios to be implemented, technologies to be used for feasibility study		██						
2)Environment Set up				██	██	██	██	
a) SQL server installation and test database				██				
b) Java Environment set up					██			
c) ASP.NET environment set up							██	
3)Web service Development					██	██	██	
a) Java RESTful web service for three scenarios					██	██		
b) ASP.NET RESTful web service for atleast one scenario							██	
4)Client development consuming the web service								
a) Client side scripting for live streaming of images						██		
b)Client side scripting for generation of graph from a list of vector data					██			
c) Client side scripting for parameter edit/save						██		
5)Comparative Analysis							██	██
6) Progress Meeting		██	██	██	██	██	██	██
7) Report Writing		██	██	██	██	██	██	██
a) Preliminary Report		██						
b)Interim Report			██	██				
c) Dissertation Draft							██	
8)Submit Report								██

Figure 1b: Gantt chart excerpt

 Apache Subversion (SVN), a software versioning and revision control system has been used to maintain current and historical versions of files such as source code and documentation.

1.4 Original Result

The project reviewed available technologies and implement all the scenarios using one solution. However, due to time constraints, it was compared against only one another solution implementing one of the scenarios. At the end, the expectation of the client was achieved successfully by proposing a solution.

2. BACKGROUND

2.1 Client/Server Architecture

The web is a distributed, dynamic and large information repository [QA]. 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 [YS]. *Services* are self-contained modules-deployed over standard middleware platforms- that can be described, published, located, orchestrated and programmed using technologies over a network [MP].

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 [YS]. Fig 2.1a shows how a basic web technology works.
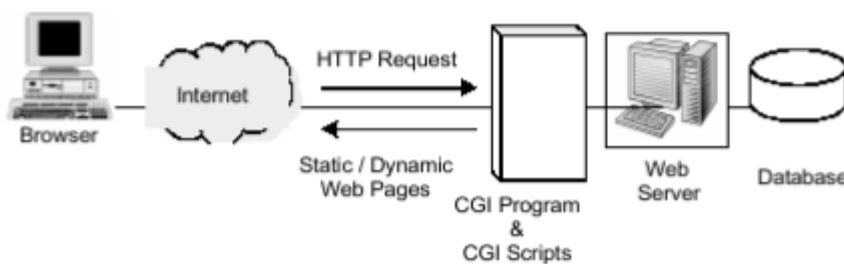

Figure 2a: Web Technology [YS]

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 [YS].

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 [YS].
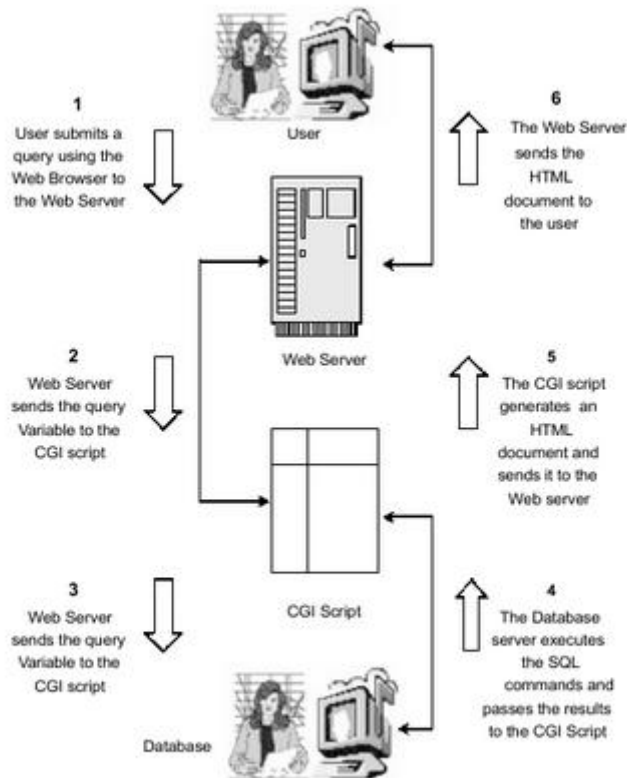
Figure 2b: <u>Communication between a user and web-based database</u> [YS]

2.2 Web Service Type

The main purpose of web service is to create web applications. 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 [SP]. 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 [AR].

Figure 2c : <u>Web-service Architecture</u> [SP]

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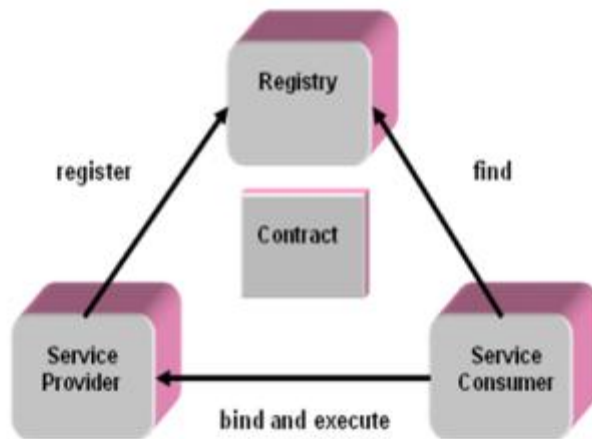
Web service can send the information to the client in any of the transferrable format, the most common being XML and JSON. The method of converting the data to a particular format is called 'data serialisation'.

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.

Most often-used types of web service:

- SOAP

- XML-RPC

- JSON-RPC

- REST

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 [QA]. XML is used to define Simple Object Access Protocol (SOAP).

XML-RPC is an older protocol than SOAP. It uses a specific XML format for data transfer, whereas SOAP allows a proprietary XML format. An XML-RPC call tends to be much simpler, and to use less bandwidth, than a SOAP call [SM].

JSON-RPC is similar to XML-RPC, but uses JSON instead of XML for data transfer [SM].

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 [AR].

RESTful services was first introduced in 2000 by Roy Fielding at the University of California.

Basically, web services are viewed as resources and can be identified by their URLs. Client and server communicate by sending and receiving representation of resources. Resources 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 [JD]. 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.
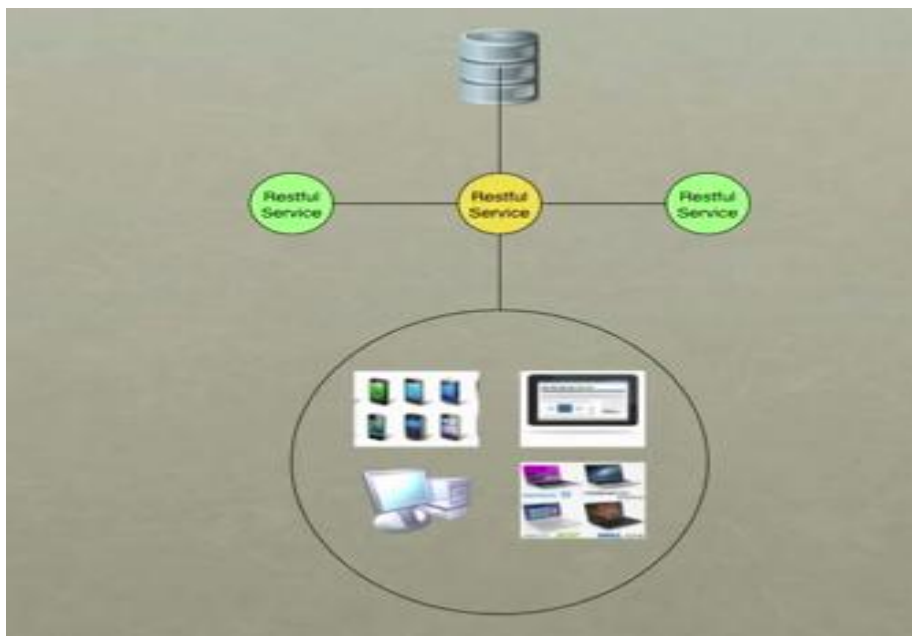

Figure 2d: RESTful Web Service

<u>Advantages of REST over SOAP</u>

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 [SP]. Table 2a illustrates a performance 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 2a: <u>REST Vs SOAP [DH]</u>

In selecting web service type for this project, it should be platform independent, easy to implement, easy to maintain and be capable of operating at high performance.

	Platform Independence	High Performance	Easy to Implement	Easy to maintain
REST	Yes (all Operating Systems & Tool Platforms)	Yes	Yes	Can Use existing tools
SOAP	No	No (compared to REST)	No	Costly

Table 2b. Web Service Type Selection Matrix

A combination of web service specifications could be used in order to obtain a better performance. Based on the above matrix and the advantages of REST, selected REST as the web service type for the solution.

2.3   Web Service API Data Format

Data can be transferred in XML, JSON or both.

Extensible Markup Language (XML) is a text format derived from Standard Generalized Markup Language (SGML).XML provides two enormous advantages as a data representation language [JFFA]:

1. It is text-based.
2. It is position-independent.

Unfortunately, it carries a lot of baggage, and it doesn't match the data model of most programming languages.

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"

               }

}

	MarkupOverhead	Simplicity	Easy to Implement	Baggage	Ease to use
JSON	Less	Yes	Yes	Less	Easy
XML	More	XML is simpler than SGML, but JSON is much simpler[JFFA]	No	More	Difficult

Table 2c. Selection Matrix for web service api data format

2.4 API Modeling Language

Languages that fit the requirements for the REST service include Java, PHP and .Net (C#). Out of these, Java and C# are the native languages.Java through Java EE and .NET through ASP.NET are competing to create dynamic web-based applications. In selecting the modeling language for the solution, the ability to communicate to the C++ application is taken into consideration. The more native the language, the more easy it would be to achieve that. Given the time scale of the project, two native languages are selected for the solutions to be compared – **Java and C#.**

2.5 Client-Side scripting

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.So, jQuery with Ajax has been selected to do the front-end of the application.

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 [JK] 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 [JK].

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 [JAD].

- 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 [CA].

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 [JAD].

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.

AJAX is the ability of a webpage to send and retrieve data asynchronously from a server, without interfering with the display and actions on the webpage. JSON is mostly used in AJAX instead of XML. Ajax apps are browser and platform independent.

3. REQUIREMENTS

The system requirements are presented in this section. The structure of this section is based on a combination of the Volere Requirements Specification template [RR] and the Requirements Management [HJ]

Project: HiPerVison Projects on Shelton Vision's Textile Inspection Systems: Backend web-based interface
Client: Dr Mike Millman, Chief Technology Officer, Shelton Machines Ltd
Contractor: Malavika Reghunathan Nair, University of Leicester

The primary client for this web application is Dr Mike Millman, Chief Technology Officer, Shelton Machines Ltd.
Eventually the client intends to use this product to allow the engineers at the company to interrogate the backend remotely to troubleshoot any faults in identifying defects and adjust parameters.

3.1 Goals of the Project

Purpose of the system

Purpose: To assess the capability of existing web technology (for a RESTful web service API as decided in section 2) to serve as a backend user Interface.
Advantage: To retain the native feel even with cross-platform or web based apps
Measurement: The advantage can be measured through non-functional requirements such as refresh rate and overhead.
Reasonable: The advantage is greater than the assessment of the existing web
technologies.

Shelton Machines Ltd uses the 'Shelton WebSpector Surface Inspection System' to inspect materials for defects. The assessment is done to develop a RESTful web service API to be used as a user interface for the engineers to detect defects and adjust parameters.

Motivation and goals

In order to implement the RESTful interface for the back-end, it needs to perform the following tasks.

- Inspect the incoming URI and figure out which resource it identifies.
- Extract any variables found within the URI and map them to variables.
- Determine the HTTP method used in the request and whether it's allowed for the resource.

- Read the resource representation found in the entity body (if any).
- Use all of this information to perform the underlying service logic.
- Generate an appropriate HTTP response, including the proper status code, description, and outgoing resource representation in the response entity body (if any) [AS].

Then to complete the API, need to consume the REST web service in Javascript. Again, it has to be implemented in another technology to do a critical comparison. These goals should be achieved within the time frame provided.

3.2 Application Context

This section will present the application area of the system and important business processes.

3.2.1 Description of the domain

The engineers at Shelton Machines Ltd use the 'Shelton WebSpector Surface Inspection System' to interrogate the back-end to see why it is not detecting defects and adjust parameters. These parameters are linked to the product database. The information the engineer sees is a mixture of material images, process graphs and other displays. The engineer is able to control the material position during a material playback mode. An overview of the WebSpector system can be found in figure 3a.

Figure 3b shows a snapshot of the 'Webcorder' used to do a material playback mode whose functionalities shall be the main focus of this project. If the system misses a defect, the engineer can detect the defect using this mode. The functionalities associated with playback mode are taken into consideration for this feasibility study. The core functionalities can be simplified into live image streaming, display of a graph, parameter update and displaying a message send from server at client side.
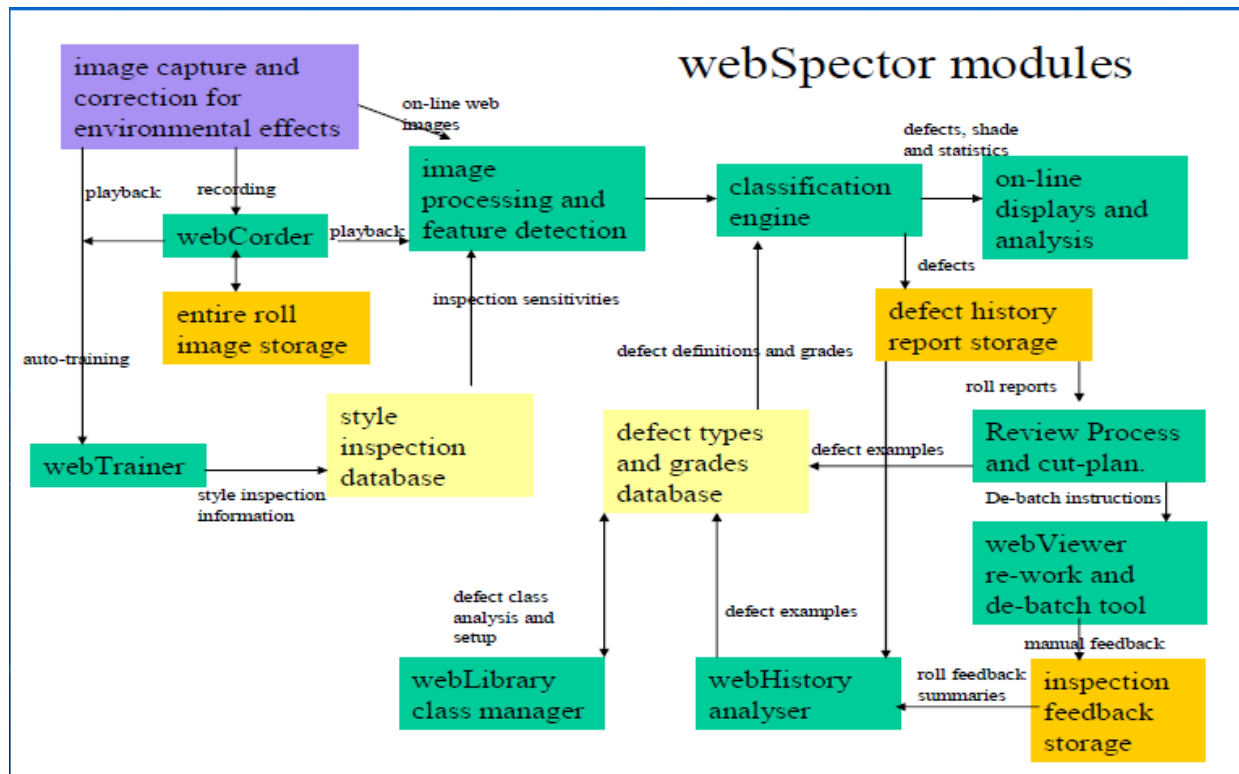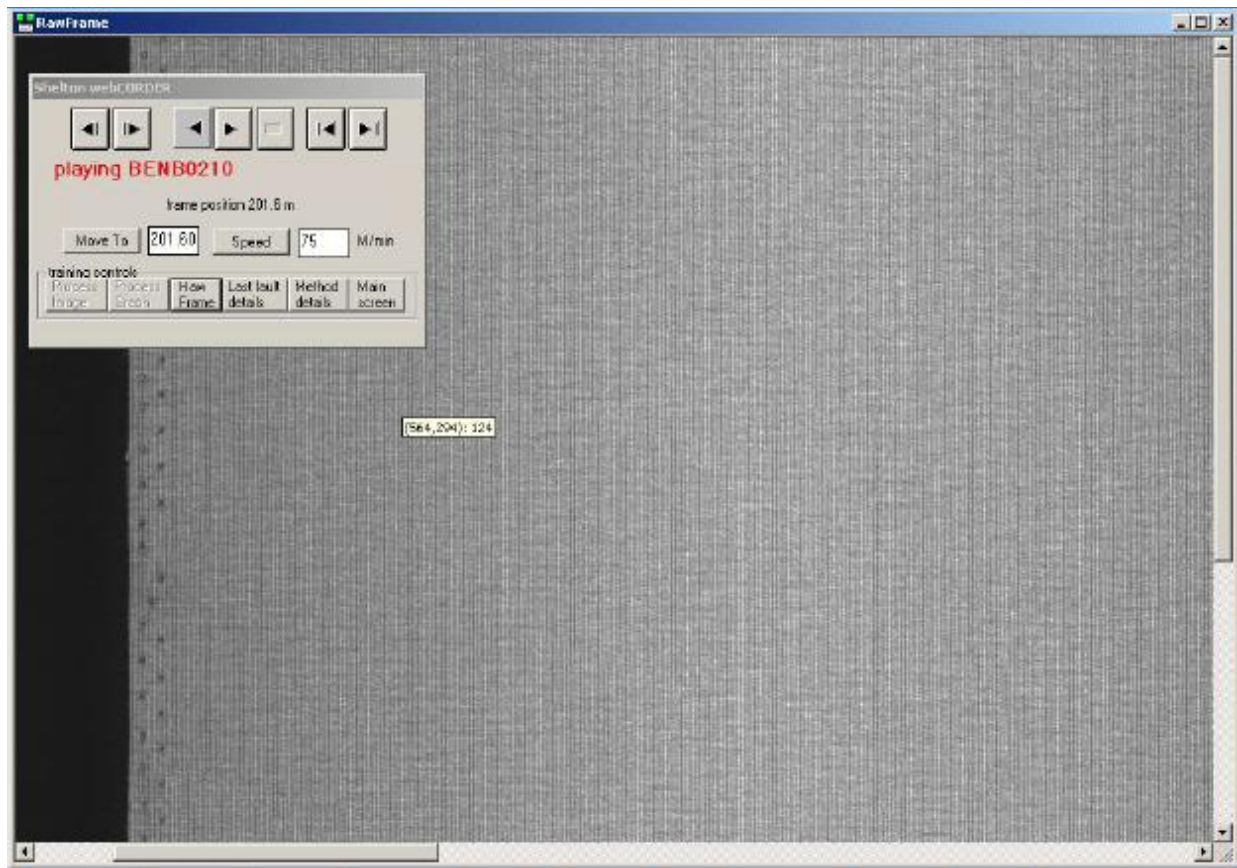
Figure 3a. <u>WebSpector System Overview [SVS]</u>

Figure 3b. <u>WebCorder- full web record and playback [SVS]</u>

3.2.2 *Business Processes*

Significant processes include:
- Inspect material for defects
- Adjust defect parameters

Trigger Event	Engineer needs to detect a defect that has been missed by the system using a playback of the material
Result	A mixture of material images, process graphs, threshold parameters linked to a product database
Participant	Engineer

The description of the main scenario is as follows:

1) The engineer switch to material playback mode
2) Enter the known distance of the fault and view the material image
3) Locate the defect position
4) If the defect cannot be seen by eye then move the material position backwards and forwards to locate it.
5) Select an image process to be checked
6) Run the image containing the defect to display a graph of the process output
7)   Adjusting threshold parameters to view the defect that has been detected
8)   Run some more material to see if the changes are not too sensitive and the system is not detecting false positives
9) If false positives detected, then back to step 4.
10) If it is impossible to avoid false positives and still detect the defect then change the image process configuration or use another one (back to step 3)

Figure 3c : <u>Activity Diagram for the engineering interface</u>

3.2.3 **Web Service Model**

As discussed in section 2, a RESTful web service API has been selected to implement the scenarios. Figure 3d shows the REST architecture. Client send request through URI representation of a resource and server responds with an HTTP status code or a media-type (XML/JSON).More about REST is explained in section 2.2
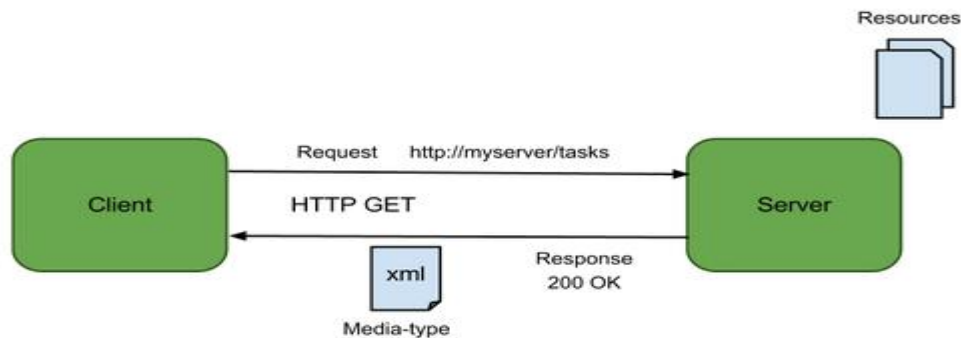

Figure3d :REST Architecture

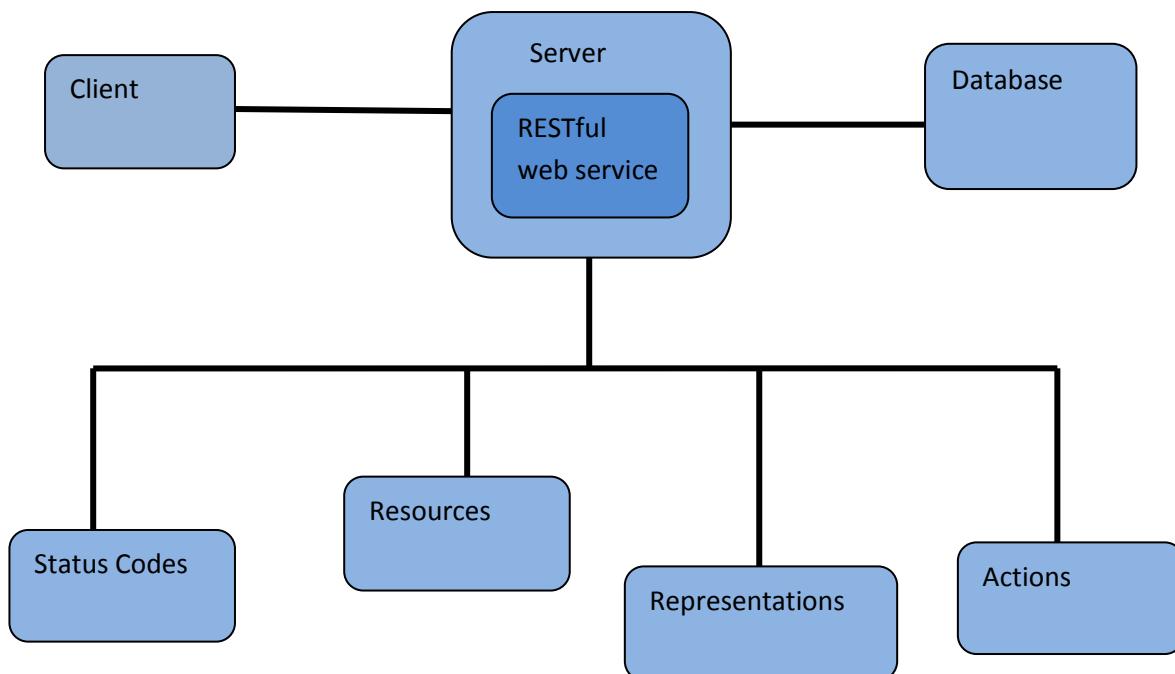The diagram below is an illustration of the domain of a RESTful web service API.

Figure 3e: Domain of a RESTful Web service API

3.2.4 Glossary

Client: The client application

Server: The machine in which the RESTful web service runs.

Database: In this project, MS SQL server 2005 as it is being used by Shelton Machines Ltd

Resources: Web services are viewed as 'Resources' and is given a unique identifier known as a universal resource identifier (URI). The most common type of URI used on the Web today is a uniform resource locator (URL). Since a given URI uniquely identifies a resource, it can be saved for future use and shared with others [AS].

Representations: Representation format of the resource. The format used in this project is JSON.

Actions: CRUD operations (create, retrieve, update and delete) performed on the data by the service using HTTP methods. These CRUD operations can be mapped to a standard set of HTTP methods as shown in Table 3a.

Data Action	Equivalent HTTP Method
CREATE	POST
RETREIVE	GET
UPDATE	PUT
DELETE	DELETE

Table 3a: HTTP Equivalent for CRUD Operations

Status Codes: Standard HTTP status code is used as status code of a response .Table 2 lists the codes used in this project. Only one of them is issued per request server.

Status Code	Description
200	OK
405	Method Not allowed

Table 3b: HTTP Status Codes

3.3 Project Constraints

In this section restrictions on product will be presented.

Time Constraints
This project has a time constraint of only 6 months done on a part-time basis. However, it is not required by Shelton Machines at the moment.

Environmental Constraints
This is a client-server application, such that the image details, parameter details and graph details are held on a database server. As the application is web-based, it will be hosted in a web server.

Hardware Environment
Clients' computers must be able to run graphical user interface. 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)

3.4 Functional Requirements

In this section we will present fundamental subject matters of the product.

3.4.1 The Scope of the Work

The current situation: The 'Shelton WebSpector Surface Inspection System' is used in various industries to inspect materials for defects. It uses a combination of vision hardware (lights, cameras and electronics) plus software. There are two primary software platforms. The first is the front-end where the operator interface, defect analysis and system co-ordination is done. The second is the back-end where the image processing and defect detection is carried out.

The context of the work: A web based back-end user interface for the engineers to interrogate the backend. Three main functionalities of the current system are taken into consideration for the feasibility study which is explained in section 1.2.

3.4.2 The Scope of the interface

The following use case diagram represents the current engineering interface.
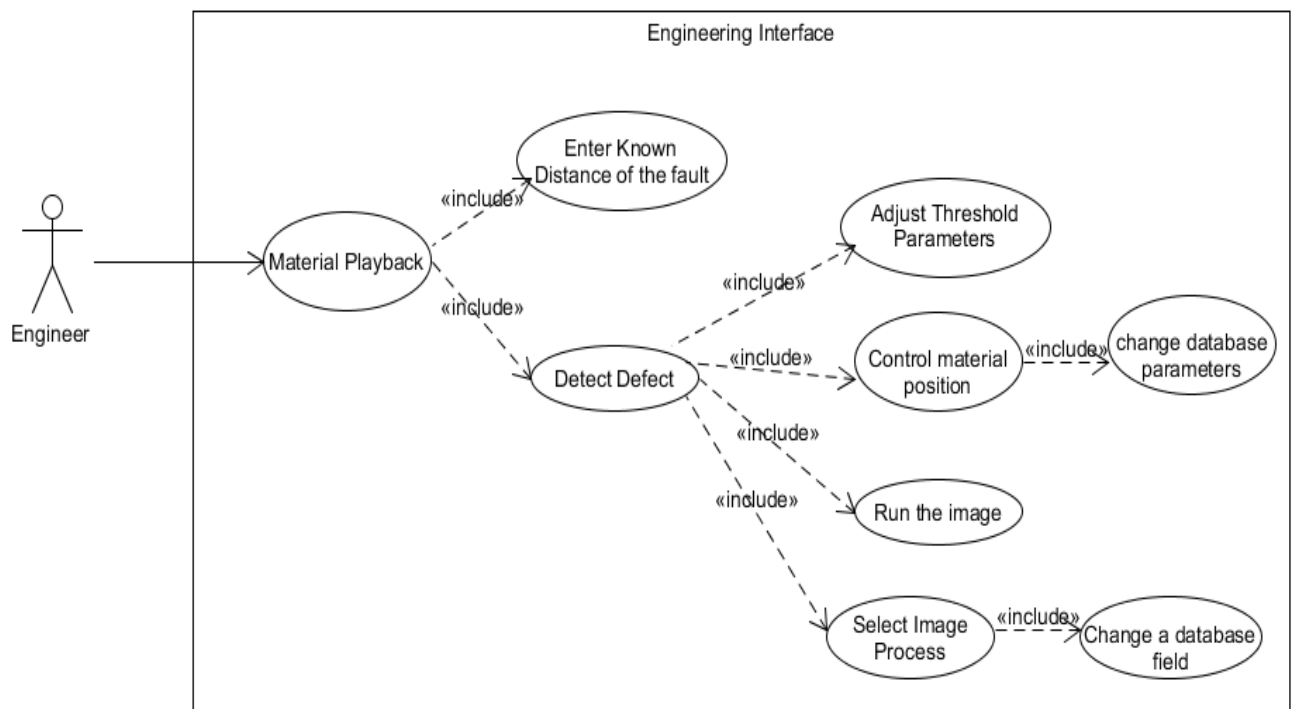
Figure 3f : <u>Use case diagram of the existing Engineering Interface</u>

3.4.3 Functional and Data Requirements

<u>Scenario 1: Image Streaming</u>

Functional Requirement: Functionality to start image streaming
Rationale: Provides the user with the ability to start the process
Fit criterion: The first image displayed shall be the first image in the list of images from the database.

Functional Requirement: Functionality to stop image streaming
Rationale: Provides the user with the ability to stop the image changing before the server comes to the end of the image list.
Fit criterion: The image changing shall stop, retaining the last image displayed

Functional Requirement: The streaming of images should continue until the server reaches the end of the image list. This is unless the user chooses to stop the streaming.
Rationale: Provides the user to view the complete streaming of images.

Fit criterion: The last image displayed shall be the last one from the list of images in the database.

Functional Requirement: Ability to live stream images continuously without getting interrupted by the database change
Rationale: In order to provide the user with live data
Fit criterion: Update/insertion of an image source in database should get reflected at client side without a page refresh (Asynchronously)

Functional Requirement: The images should be displayed in its original size
Rationale: Provides the user to view the images in its actual size for better analysis
Fit criterion: The size of the image displayed at client side should match the image size in its actual location.

Scenario 2: Graph Generation

Functional Requirement: The client should display an interactive chart from a list of vector data which represents the graph details.
Rationale: Provides the user with the ability to view points of the graph
Fit criterion: The graph shall have an X-Axis, Y-Axis and a tool tip to view the points of the graph.

Functional Requirement: Ability to regenerate the graph when there is an update.
Rationale: In order to provide the user with an up to date data.
Fit criterion: Change in database shall get reflected at the client side without a refresh

Scenario3: Parameter Update

Functional Requirement: Ability to edit and save the defect parameters.
Rationale: In order to adjust the threshold parameters
Fit criterion: A message from server shall be displayed at the client side on successfully saving a parameter

Data Requirement: The database server currently used by Shelton Machines is Microsoft SQL server 2005. A test database containing image paths, vector data and parameters shall be created using the SQL server for the purpose of this feasibility study.

3.5   Service Functions

3.5.1 Use case diagram

The following Use Case Diagram represents the most obvious functions of the web service API.

Figure 3g : Subsection of use case

3.5.2 Description of Use Case Image Streaming

Name:	Image Streaming
Actor:	User
Goal(of the user)	To view the material image during a playback mode
Precondition:	1.Image paths should be present in the database 2.Both client and server should

	have access to the image storage
Postcondition:	Live streaming of images(in actual size) in a web browser
Flow:	1. The user selects 'Image Streaming' on the active web page 2.The user then select 'Start'
Special Requirements:	1.The images are of size 3 MB 2. The image streaming should be fast 3. Any update in the database should get reflected at client side.

3.5.3 Description of Use Case Graph Display

Name:	Graph Display
Actor:	User
Goal(of the user)	To view the graph of the process output
Precondition:	Vector data should be present in the database in the below format first item = number of points 2nd item = y axis minimum 3rd item = y axis maximum 4th item to n item = points
Postcondition:	An interactive graphical representation of the vector data
Flow:	The user clicks on 'Generate graph' on the active web page
Special Requirements:	If there is a database change, the graph should get regenerated without a refresh.

3.5.4 Description of Use Case Parameter Update

Name:	Parameter Update
Actor:	User
Goal(of the user)	To edit and save the parameters
Precondition:	Parameter details should be present in the database
Postcondition:	An interactive graphical representation of the vector data

Flow:	1.The user clicks on 'Parameter update' on the active web page 2.The user then clicks 'Edit' to edit the parameter 3.Then user clicks 'Save' to save the parameter
Special Requirements:	A success message send from server should get displayed at client side on successful saving of the parameter.

3.6 Non-Functional Requirements

In this section we will discuss properties of the functions discussed in section 3.4

3.6.1 Look and Feel

The project aims at a feasibility of using current web technologies for a web based user interface. The backend system is still to be developed with a C++ backend. The client requested to implement the main functionalities that will be used by the system. So the overall look and feel is not really a constraint within the scope of this project. In future, this requirement will have to be modified.

3.6.2 Usability

Learning Effort: The less the learning effort required, the better the solution will be.
Fit criteria: This is tested by the number of weeks taken to complete the solution

3.6.3 Performance

Speed requirement: The response time in image streaming shall be fast enough so that it looks like a video.
Fit criteria: No response shall take longer than 5 seconds.

Refresh rate requirement: The refresh rate of the images or graph when there is a database change shall be fast enough to avoid interrupting the user
Fit criteria: This is tested by calculating the frames per second

Overhead requirement: The CPU load on the process shall be low
Fit criteria: This is tested by checking the CPU load on the processors

3.6.4 Operational

Requirement for Expected Physical Environment: The web service shall be ported to a non-microsoft platform/ non-PC platform.

Requirement for interfacing with adjacent systems: The product shall work on the last four releases of the five most popular browsers.

3.6.5 Maintainability and Support

Maintenance Requirement: Database data should easily be updated by the administrator without affecting the user interface

Maintenance Requirement: The solution shall be well structured and self documenting .

Fit criteria: This can be tested by checking the solution is object oriented and the number of lines of code required.

3.6.6 Proprietary Rights

Proprietary requirement: The software or tools or plugins used for the solution shall be under free licence to use.

4. Solution using Java Web Service

In this section the design, implementation and testing of the solution using JAX-RS (Java API for RESTful Web service) will be analysed.

4.1 Design

4.1.1 Architecture

Multi-Tier Architecture

The solution is based on a multi-tier architecture, which comprises of the following tiers as shown in figure 4a.
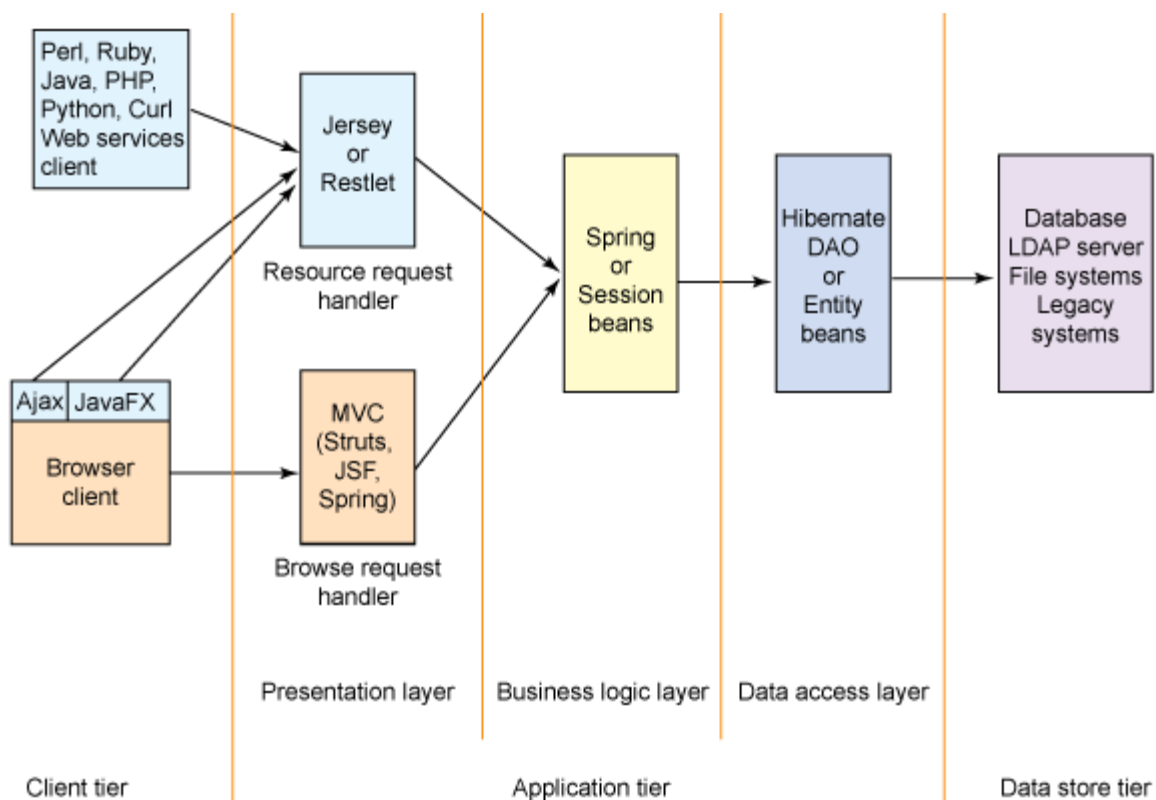

Figure 4a : Diagram of a multi-tiered Web application environment [BS]

Business rules are centralized into the business logic layer that serves as an intermediary for data exchange between the presentation layer and the data access layer. Data is provided to the presentation layer in the form of domain objects or value objects. Decoupling the Browser Request Handler and

Resource Request Handler from the business logic layer helps facilitate code reuse, and leads to a flexible and extensible architecture [BS].

As an alternative, the components in the business layer and data access layer can be implemented as EJB components with support from an EJB container that facilitates the component life cycle and manages the persistence, transactions and resource allocations. However, this does require a Java EE-compliant application server such as JBoss, WebLogic and would not work with Tomcat [BS].

Figure 4b shows the architectural diagram of the proposed solution. The architecture diagram outlines the fact that there would be 3 Services in the solution:
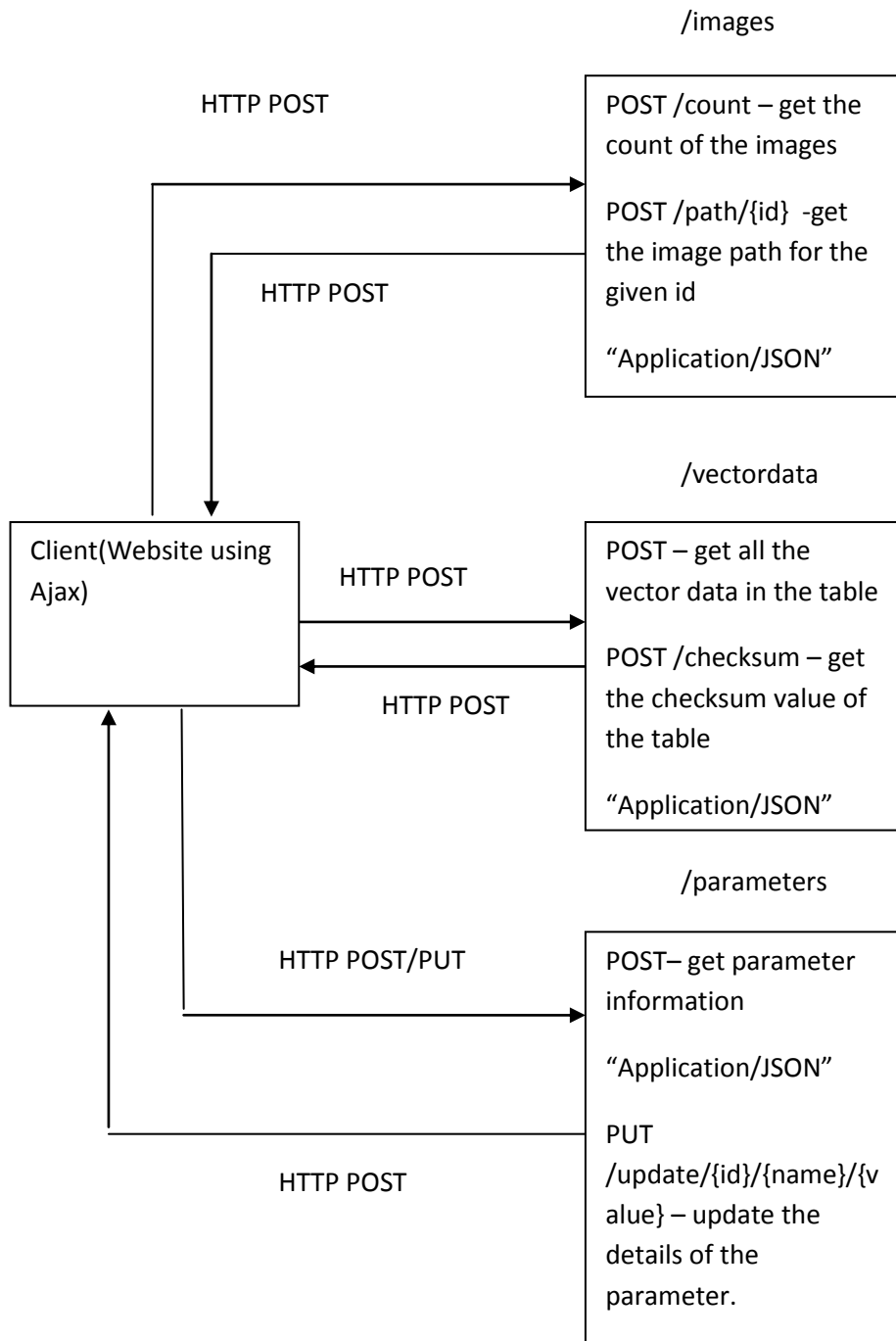
- Image
- VectorData
- Parameters

/images

HTTP POST

| POST /count – get the count of the images |
| POST /path/{id} -get the image path for the given id |
| "Application/JSON" |

HTTP POST

Client(Website using Ajax)

/vectordata

HTTP POST

| POST – get all the vector data in the table |
| POST /checksum – get the checksum value of the table |
| "Application/JSON" |

HTTP POST

/parameters

HTTP POST/PUT

| POST– get parameter information |
| "Application/JSON" |
| PUT /update/{id}/{name}/{value} – update the details of the parameter. |

HTTP POST

Figure 4b: Architecture Diagram

Here the HTTP method POST has been used instead of using GET because POST is a little safer than GET because the parameters are not stored in browser history or in web server logs.

Technologies and Toolkits used

The web technologies that were used in this solution are:
- jQuery (with Ajax) – send request and receive response
- Microsoft SQL Server – to store the test data
- JAX-RS(JSR 311) – Java API for RESTful Web Services
- Jersey – JAX-RS reference implementation for building RESTful Web Services
- SVN – project management
- Oracle Web Logic –Web application container

The following deployment diagram depicts the run-time static view of the Client-Server web architecture for the solution
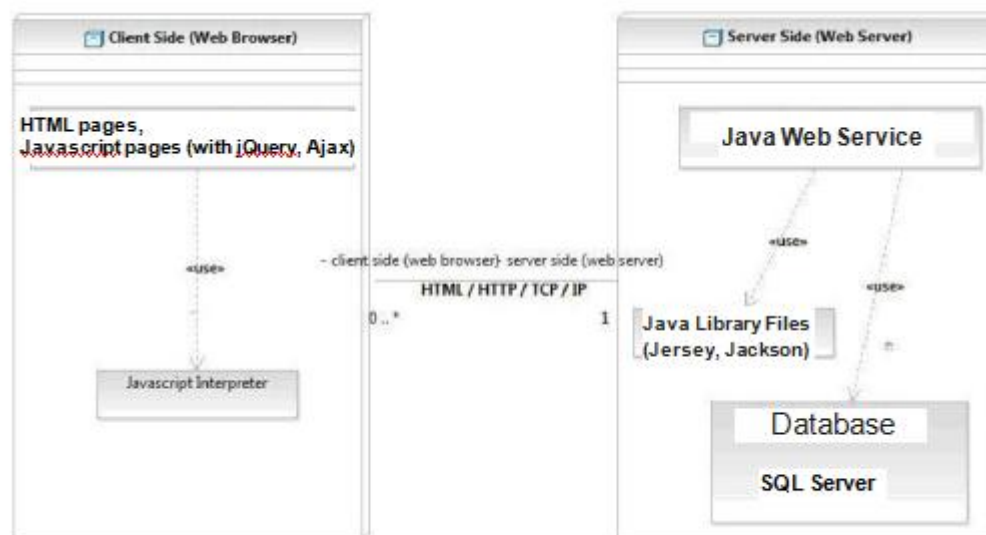

Figure 4c: Deployment diagram of the web service

Figure 4d below shows the model designed for the solution which conforms to the architecture diagram shown in figure 4b.
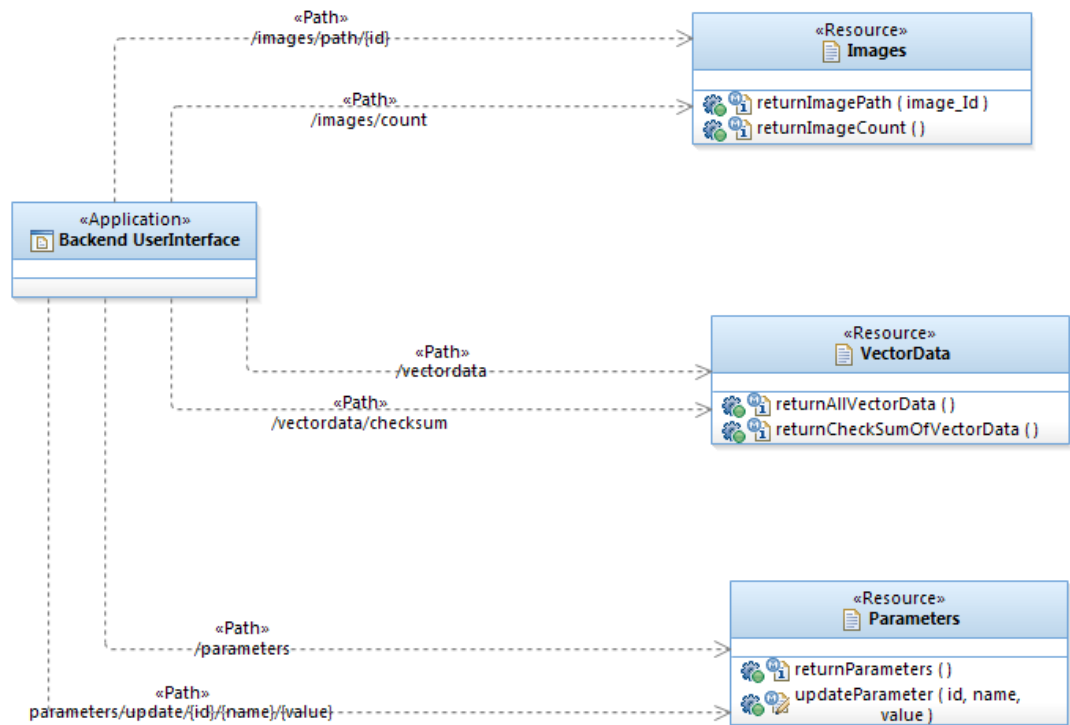
Figure 4d: RESTful Backend User Interface Model

4.1.3 Analysis Sequence Diagram

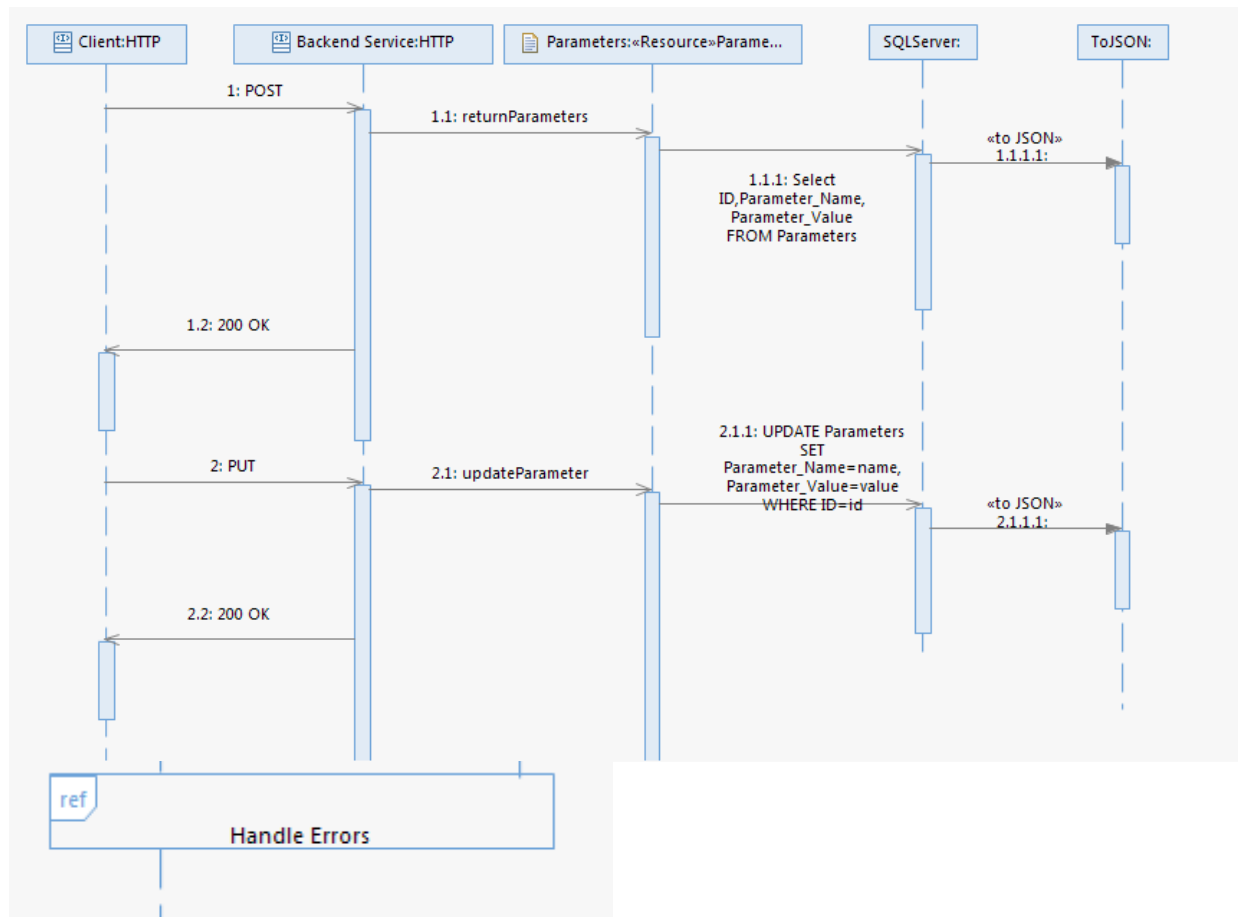The below sequence diagram, figure 4e shows the interaction between components of the solution.

Figure 4e. Sequence Diagram of component interaction

The following sequence diagram, figure 4f is used to emphasize the interaction of a POST request for the method returning the image path. The use case scenario displays the sequence of the POST request required for getting the image path for image streaming.

Figure 4f : Sequence Diagram of POST Request

The sequence diagram depicts the below steps:
1. A jQuery client makes an HTTP request with method type POST and "1" as the identifier of the image
2. The client sets the representation type it can handle through the Accept request header field
3. The web server receives and interprets the POST request to be a retrieve action. At this point, the web server passes control to the RESTful framework to handle the request. The job of the REST framework is to ease the implementation of the REST constraints.In this case, jersey framework has been used. Business logic and storage implementation is the role of the domain-specific Java code [BS].

4.1.5 Solution components

In this solution, the web service shall be implemented using **Java** language.

Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation and hence used in this solution. The Jersey implementation provides a library to implement Restful web services in a Java servlet container [LV]. The three important java libraries selected for this solution are **Jersey, Jackson and Jettison**. Jersey framework provides Jackson and Jettison library files for JSON related functions.

Java EE provides an API called JAX-RS, which is a standard that makes it easy to create a RESTful web service. Jersey Framework is a reference implementation of JAX-RS. JAX-RS is implemented as EJB components and this require a Java EE-compliant application server such as JBoss or WebLogic. Moreover, a commercial java app server offers better scalability. **Oracle WebLogic** is a popular commercial java app server with better support and a beautiful GUI, hence selected for this solution.

Database selected for this solution is **Microsoft's SQL server 2005** because that is the server being used by the client, Shelton Machines Ltd.

REST service can be consumed by any client side programming languages. **jQuery** has been selected for this solution because it makes it much easier to use Javascript on the website, light-weight, has plugins for almost any tasks and most of the big companies use it. The jQuery plugin selected for displaying the graph is 'Highcharts' and another plugin used for the parameter update is 'Datatables'.

The following is the summary of the components that have been opted for use in this solution:

- Service Language – Java
- JAVA Library Files – Jersey ,Jackson, Jettison
- Web Server – WebLogic
- Database- MS SQL Server 2005
- Client Language – jQuery(along with highcharts and datatables)

The below table shows the tools and libraries used for this solution and the corresponding links from which it can be downloaded:

Software/Tool	Web location
Java JDK	http://java.sun.com/
Oracle WebLogic	http://www.oracle.com/technetwork/middleware/weblogic/downloads/wls-main-097127.html
OEPE	http://www.oracle.com/technetwork/middleware/fusion-middleware/downloads/index.html
SQL Server Management Studio Express	http://www.microsoft.com/en-gb/download/details.aspx?id=8961
Microsoft JDBC driver 4.0 for SQL server (sqljdbc4.jar)	http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774
Java Library Files (Jersey, Jackson,Jettison)	https://jersey.java.net/download.html
jQuery	http://jquery.com/download/

Table 4a: Web location for the software /tool used

4.2 Implementation

As concluded in the previous section, the REST service is written in Java using java libraries from Jersey framework and runs on WebLogic server. All the server side coding and client side jQuery coding was written within Eclipse IDE that comes as a part of OEPE package. The test database is created using Microsoft SQL Server Management Studio Express. Google chrome has been used as the primary browser to test the service/api.

4.2.1 Server

4.2.1.1 SQL server

For the purpose of this project, a test database should be set up using MS SQL server. That has been done using Microsoft's SQL server Management studio express as per the instructions provided by SML. The steps provided by SV for the installation of SQL server can be found in appendix A.

A test database 'TestDB' was created using SQL server 2005 with three test tables 'Images' , 'VectorData' and 'Parameters' whose specifications are shown in figures 4g,4h and 4i.

For testing purpose of the project, the images shall be stored in the C drive of the local machine.

Figure 4g: <u>Specification for table 'Images'</u>

Figure 4h: <u>Specification for table 'Parameters'</u>

Figure 4i: <u>Specification for table 'VectorData'</u>

4.2.1.2 Web Server

Before developing the service, the web server(a Java EE container) should be set up and configured. The latest stable version of **Oracle WebLogic**, version 12c (12.1.1) has been selected to use as explained in section 4.1.5. The server can be controlled from the Eclipse IDE. As the server needs to connect to the SQL server, it would require a JDBC driver.

The below mentioned Type 4 JDBC driver (Microsoft JDBC driver 4.0 for SQL server- sqljdbc4.jar)that provides database connectivity through the standard JDBC application program interfaces (APIs) available in Java Platform have been used.
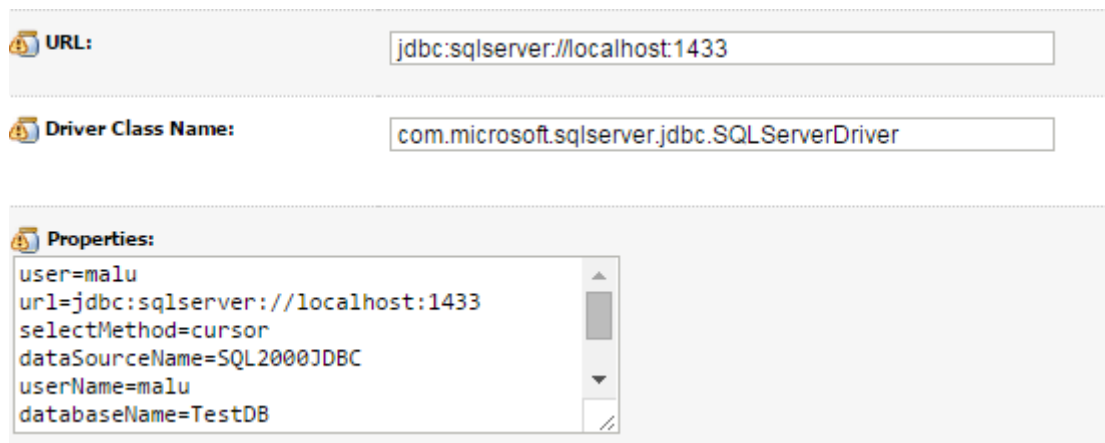
This jar file has been placed in the 'lib' folder of WebLogic and the respective CLASSPATH has been added to the common environment batch file of WebLogic server.

4.2.2 Service

4.2.2.1   Database Connection

Oracle WebLogic has a beautiful GUI. A datasource for a database connection can be created from the admin console of WebLogic and it can be accessed from the java code using its reference id (JNDI name) set during the creation of the datasource.
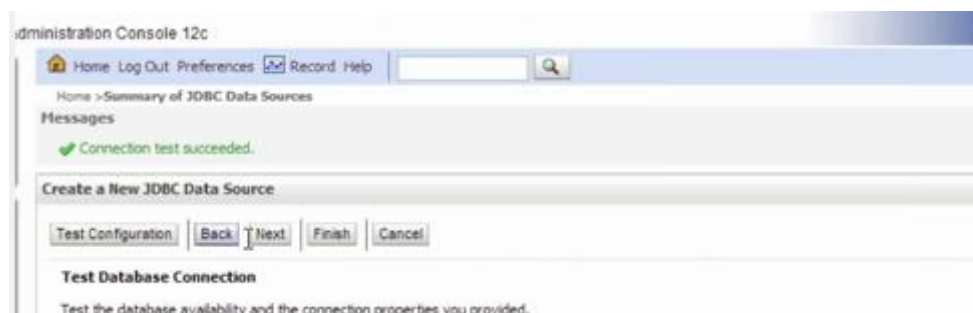
MS SQL data source with the JNDI name 'SqlDataSource' is created from the oracle WebLogic Administration console. Figure 4j shows the properties of the created JDBC datasource.

Figure 4j: <u>JDBC DataSource properties</u>

The connection can be tested from the WebLogic administration console and success message will be displayed as shown in figure 4k if a successful connection is established.

Figure 4k: <u>Database connection test</u>

Accessing the datasource from code:

As mentioned above, the datasource can be accessed using the JNDI name. In this case, it is 'Sqldatasource'. The below code shows how it is accessed from the java class.(location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/src/com/sheltonmachines/service/dao/SqlDB.java)

```java
import java.sql.*;

import javax.naming.*;
import javax.sql.*;

/**
 * This class will return the MS SQL server connect object
 * @author Malavika.
 */
public class SqlDB {
        //SqlDataSource holds the database object
        private static DataSource SqlDataSource = null;
        //state is used to look up the database connection in weblogic
          private static Context state = null;

          public static DataSource SqlDataSourceConn() throws Exception {

                // if the database object is already defined,
                // then return the connection

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

                 // state is used to lookup the database object in weblogic
                try{
                        if(state == null){
                                state = new InitialContext();
                        }

                // SqlDataSource will hold the database object
                SqlDataSource = (DataSource) state.lookup("DataSourceSql");
                }
                catch (Exception e){
                        e.printStackTrace();
                }


                return SqlDataSource;
            }

            /**
             * This method will return the connection to the SchemaSqlDB schema
             * only java class in the dao package can use this method as the scope
             * is protected
             * @return Connection to MS SQL Server database.
             */

            protected static Connection databaseConnector() {
            Connection conn= null;
```

```
        try{
                conn = SqlDataSourceConn().getConnection();
                return conn;
        }
        catch(Exception e){
                e.printStackTrace();
        }
        return conn;
    }
}
```

This above code returns a connection to the database and it can be used to query the database. An example is shown below:

```
conn = databaseConnector();
query = conn.prepareStatement("SELECT Count(IMAGE_PATH) AS number FROM Images");
ResultSet rs = query.executeQuery();
```

4.2.2.2 Creation of JSON

The database result set retrieved in section 4.2.2.1 is converted to JSON format before returning it to browser. It is a lightweight format easily readable by both human and software. JSON format is explained in detail in section 2.3 . To retain consistency, all the results are converted to JSON format even though it was not really necessary.

There might be external java library files available to do the conversion but having our own java class for the conversion gives lot of flexibility in the long run. It allows doing something specific to the application. When using a 3rd party tool, it would be usually hard to modify it.

In order to format the database rows to JSON, a separate java class has been written. In this solution, for study purposes, simple database tables are being used which returns a single row many of the times. But when it comes to the real project, the database would be returning more number of rows and it would easy to decipher at client side if it is in JSON format.

Two important classes used in this for the conversion of database result to JSON format are:

- ***org.codehaus.jettison.json.JSONArray***
- ***org.codehaus.jettison.json.JSONObject***

JSON array is a collection of JSON objects. A JSON Object represents each row of the record set. The below code explains the creation of the JSON array:

(location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/
com.sheltonmachines.service/src/com/sheltonmachines/service/util/ToJSON.java)

```java
//JSON array to be created and returned
    JSONArray json = new JSONArray();

    try {

       // retrieving all the column names from the database ResultSet
       java.sql.ResultSetMetaData rsmd = rs.getMetaData();


       while( rs.next() ) {

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

          //Initializing a JSONObject for each of the rows
          JSONObject obj = new JSONObject();

          //each column is places into the JSON Object
          for (int i=1; i<columnCount+1; i++) {

             //getting the column name
            String col_name = rsmd.getColumnName(i);

             /*mapping the sql data type to the json object
                     * this handles the different datatypes
             */

            if(rsmd.getColumnType(i)==java.sql.Types.ARRAY){
              obj.put(col_name, rs.getArray(col_name));

          }
```

It loops through each row and place each column into a JSON object.This JSON object is then put into the JSON array. Once the loop completes through all the rows, the JSON Array would be completed and then returned to the requestor.

Each column in a table can have different datatypes. We need to account for each of these datatype (like varhcar, Boolean, integer, float) so that the code knows how to handle that data and place it into the JSON object.

4.2.2.3 Backend User Interface API

Basic Structure

This section explains the server side implementation of functionalities explained in section 3.4.3.

A web service API is invoked using a URL by the client/browser. API will receive the request and start deciphering the URL. A string in the URL shall route to the web service. At this point, the remaining URL string will be deciphered to see if there is a java class to handle this particular path. Java

Jersey terminology is used to route the path to a java class which is described below. After executing, it shall return data back to the browser.

The Jersey framework uses annotations with HTTP Verb to annotate Java Objects to create RESTful web services. The key annotations used are:

- @PATH - Identifies the URI path that a resource class or method serve requests for. In other words, this will route the path to the java class.
- @PATHPARAM- allows the use of some of the URL string inside the method.
- @GET - Indicates that the annotated method responds to HTTP GET requests
- @POST- Indicates that the annotated method responds to HTTP POST requests
- @PUT- Indicates that the annotated method responds to HTTP PUT requests
- @PRODUCES- indicate what type of output a method will produce. It can specify more than one output type
- @CONSUMES- indicates what type of data sent up in the body of HTTP message. It is a way to limit what type of HTTP message can access specific methods. Can use our own or predefined MediaType [JUG].

In order to route the URL to the respective web service, the string used for identifying the web service shall be mentioned in the web.xml file which is the starting point of the API. The servlet is defined by the 'servlet' element in the web.xml file. The element 'servlet-mapping' defines the URL pattern for the servlet defined. The below code shows that anything with url path '/api/' would route it to the servlet code. The servlet code is nothing but java classes which returns the response of the api. If '/api/' is not in the URL path, route will try to find the HTML file. The code also outlines the fact that the servlet creation is done using the following packages of jersey framework:

- **com.sun.jersey.spi.container.servlet.ServletContainer**
- **com.sun.jersey.config.property.packages**

```xml
<!-- defining the RESTful service -->
<servlet>
  <servlet-name>Backend Rest Service</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
  <!-- param-value should be the same as the 'display-name'mentioned above -->
  <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.sheltonmachines.service</param-value>
  </init-param>
  <!-- this is the servlet to be loaded first -->
  <load-on-startup>1</load-on-startup>
</servlet>
<!-- mapping the URL in the pattern /api/* to the servlet code.servlet-name should match the one defined above --
<servlet-mapping>
    <servlet-name>Backend Rest Service</servlet-name>
    <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

The above code defines the base URL for all the methods serving requests as 'com.sheltonmachines.service/api/' (location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/ WebContent/WEB-INF/Web.xml)

Scenario 1– Image Streaming

For live image streaming functionality to work, we require to get the image path of the images, one at a time and the number of image paths present in the database at the time of accessing an image path, that way monitoring for a database change.

A java class with two methods has been defined to achieve this, one for getting the image path given an image ID and the second one to get the count of the image paths.

The following code defines the Java class to serve the requests for image streaming.[Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/src/com/sheltonmachines/service/manage/V1_Images.java]

```java
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.codehaus.jettison.json.JSONArray;
import com.sheltonmachines.service.dao.SchemaSqlDB;

/**
 * Backend UserInterface
 * @author Malavika
 * @version 1.0
 */
```

```java
/**
 *
 * This is the root path to the resource - Images.
 * In the web.xml file, it is specified that /api/*
 * needs to be in the URL to get to this class.
 *
 * V1 is used in the URL path for versioning.This is the first version, v1.
 *
 *Example URI to get to the root of this api resource:
 * http://localhost:7001/com.sheltonmachines.service/api/v1/images/
       */
@Path("/v1/images/")
public class V1_Images {

            /**
             * This method is nested one down from the root
             * and it returns the number of image paths
             * @return MediaType.APPLICATION_JSON
             * @throws Exception
             */
     @Path("/path/{id}/")
     // indicates HTTP verb to get to this method
     @POST
     // indicates the HTTP message body
     @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
      // indicates the output type
     @Produces(MediaType.APPLICATION_JSON)

       public Response returnImagePath(@PathParam("id") int image_id)
                              throws Exception
    {

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

The annotations used in the code have been explained in the beginning of this section. The above code depicts that, the URL string that needs to be added to the base URI in order to route the api to the java method 'returnImagePath' would be '/v1/images/<id>' where <id> is the id of the image path to be retrieved.

The Java method does the following operations:

- Access database and query the database table
- Format the result to JSON
- Return the JSON object to the client

As explained in subsection 4.2.2.1, a datasource object is used to establish a connection to the database. The following code shows how the datasource object is used to open a connection and query the database. [Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/src/com/sheltonmachines/service/dao/SchemaSqlDB.java

```java
**
 * This class holds all sql queries performed on the database.
 * This class extends SqlDB class to inherit all the methods of that class
 * @author Malavika
 *
 */

public class SchemaSqlDB extends SqlDB {

    /**
      * This method allows to get the count of the image paths
      * @return - count of the images in json format
      * @throws Exception
      */
    public JSONArray queryReturnImageCount() throws Exception {

     //use of 'PreparedStatement' avoids data injection
    PreparedStatement query = null;
    Connection conn = null;

    // an instance of ToJSON class is used to convert the query result to JSON
    ToJSON converter = new ToJSON();
    JSONArray json = new JSONArray();

            try{
                    // getting the database connection
                    conn = databaseConnector();
                    //performing SQL query on the image table
                    query = conn.prepareStatement("SELECT Count(IMAGE_PATH) AS
                                                        number FROM Images");

                    ResultSet rs = query.executeQuery();

                    //converting the result to JSON array
                    json = converter.toJSONArray(rs);
                    query.close();// close connection
                        }
```

The above code also outlines the fact that the query result is converted to JSON array format before returning it. JSON format conversion is explained in subsection 4.2.2.2.

This JSON array shall be returned back to the client. How the client handles the JSON array shall be discussed in greater detail in section 4.2.3.

The API can be tested by just entering the URL in a browser if it is using the HTTP GET method.

The other function required for live image streaming is to get the image count which is implemented in a similar fashion. The count is calculated by using the below SQL query:

query = conn.prepareStatement("SELECT Count(IMAGE_PATH) AS number FROM Images");

More details of this method can be found in the api specification, section 4.2.3

Scenario 2– Graph Display

The implementation of this scenario is similar to the above function except that different java methods have been used. One java method will retrieve the vector data from the database while the other one checks for any database change.

The below SQL query is used to retrieve the vector data from the database:

query = conn.prepareStatement("Select VectorData1 FROM VectorData")

In order to check for a database change, the below SQL query has been used. This query will return the same number as long as there is no change in the table contents.

query = conn.prepareStatement("SELECT CHECKSUM_AGG(BINARY_CHECKSUM(*)) AS Result FROM VectorData WITH (NOLOCK)");")

More details of these methods can be found in the api specification, section 4.2.3

Scenario 1– Parameter Update

The implementation of this scenario is also similar to the above two scenarios except that difference in java methods. Two java methods are used for this; one java method will retrieve the parameter details from the database while the other one writes the edited data back to the table.

The below SQL query is used to retrieve the parameter details from the database:

query = conn.prepareStatement("Select ID,Parameter_Name,Parameter_Value FROM Parameters")
query = conn.prepareStatement("UPDATE Parameters SET Parameter_Name=?, Parameter_Value=? WHERE ID=?");

The web service method used to update the parameter is as follows.
 [Location in svn:https://campus.cs.le.ac.uk/svn/mrn12/code/tags/
com.sheltonmachines.service/src/com/sheltonmachines/service/manage/V1_Parameters.java]

```java
/**
      * This method returns a message after updating the parameter table.
      * @return MediaType.APPLICATION_JSON
      * @param id
      * @param name
      * @param value
      */
    @Path("/update/{id}/{name}/{value}")
    // indicates the HTTP verb to get to this method
    @PUT
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    //indicates the type of output
    @Produces(MediaType.APPLICATION_JSON)

    public Response updateParameter(@PathParam("id") int id,
                                    @PathParam("name") String name,
                                     @PathParam("value") String value)

     throws Exception {


       String returnString = null;
       int http_code;
            // instances to create the json array and database schema
            JSONArray jsonArray = new JSONArray();
            JSONObject jsonObject = new JSONObject();
            SchemaSqlDB dao = new SchemaSqlDB();

            try{

          //call the SchemaSqlDB method to update the parameter table and pass the
          // arguments
                  http_code = dao.queryUpdateParameter(id,name,value);

       //if the method returns a http code of 200, put a success message to the json
       // object to be returned
                  if(http_code == 200) {
                  //put method is used to add data to json object
                  jsonObject.put("HTTP_CODE", "200");
                  jsonObject.put("MSG", "Item has been updated successfully");
                  }
                  //handling error
                  else {
                          return Response.status(500).entity("Server was not able
                          to process your request").build();

                  }
```

More details of these methods can be found in the api specification, section 4.2.3

4.2.2.4   API Specification

In this section, the request and response details of the methods in the api is explained. The methods used are:

- return image path
- return image count
- return parameters
- update parameter
- return all vector data
- return checksum of vector data

All the possible responses are listed under 'Responses' for each method. Only one of them is issued per request server. All responses are in JSON format.

i)   return image path

This method is used to get the image path from the database given an image id.

Request

Method	URL
POST	com.sheltonmachines.services/api/v1/images/path/< id>/

Type	Params	Value
URL_PARAM	< id>	Number

Table 4b: Request details of return image path

<id>: Id of the image to be retrieved

Response

Status	Response
200	Image path of the image with the given id in JSON format [{"IMAGE_PATH":"Pictures\\Frame_000010.jpg"}]
500	"Server was not able to process your request"

Table 4c : <u>Response details of return image path</u>

ii)   return image count

This method is used to get the number of images from the database. A separate method is used to get the image count because it is called in set intervals, so that the image streaming will continue till the end of the list, even if a new record is added while streaming.

Request

Method	URL
POST	com.sheltonmachines.services/api/v1/images/count/

Table 4d: <u>Request details of return image count</u>

Response

Status	Response
200	The count of the image in JSON format [{"number":12}]
400	"Server was not able to process your request"

Table 4e: <u>Response details of return image count</u>

iii)   return all vector data

This method returns all the vector data from the database.

Request

Method	URL
POST	com.sheltonmachines.service/api/v1/vectordata/

Table 4f: <u>Request details of return all vector data</u>

Response

Status	Response
200	An array of vector data [{"VectorData1":"105"},{"VectorData1":"0"},{"VectorData1":"60"},{"VectorData1":"50"},{"VectorData1":"43"}, …………………]
500	"Server was not able to process your request"

Table 4g: Response details of return all vector data

iv)   **return checksum of vector data**

Request

Method	URL
POST	com.sheltonmachines.service/api/v1/vectordata/checksum/

Table 4h: Request details of return checksum of vector data

Response

Status	Response
200	[{"Result":12913}]
500	"Server was not able to process your request"

Table 4i : Response details of get checksum of vector data

i)   **return parameters**

Request

Method	URL
POST	com.sheltonmachines.service/api/v1/parameters/

Table 4j: Request details of return parameters

Response

Status	Response
200	[{"ID":1,"Parameter_Name":"Para1","Parameter_Value":"1205"},{"ID":2,"Parameter_Name":"Para2","Parameter_Value":"6712"},{"ID":3,"Parameter_Name":"Para3","Parameter_Value":"349"}]
500	"Server was not able to process your request"

Table 4k : Response details of return parameters

ii)   **update parameters**

Request

Method	URL
PUT	com.sheltonmachines.service/api/v1/parameters/update/<id>/<name>/<value>

Type	Params	Value
URL_PARAM	Id	Number
URL_PARAM	Name	String
URL_PARAM	Value	String

Table 4l : Request details of update parameters

<id>: The unique id of the parameter to be updated

<name>: The name of the edited parameter

<value>: The value of the edited parameter

Response

Status	Response
200	[{"MSG": "Item has been updated successfully" }]
500	"Server was not able to process your request"

Table 4m: Response details of update parameters

Summary of resources:

Table 4n gives a summary of the resources that are defined in the web service api.

Resource	URI	HTTP methods supported	
/path	v1/images/path/{id}/	POST	returnImagePath
/count	v1/images/count/	POST	returnImageCount
/vectordata	v1/vectordata/	POST	returnVectorData
/checksum	v1/vectordata/checksum/	POST	returnCheckSumOfVectorData
/parameter	v1/parameters/	POST	returnParameters
/update	/parameters/update/{id}/{name}/{value}	PUT	updateParameter

Table 4n: Summary of resources

4.2.3 Client

The client written in JQuery shall consume the Jersey-based RESTful web service explained in section 4.2.3. The following external Javascript files have been used to implement the client; the available links are mentioned in section 4.1.5

- jquery-1.11.1.min.js
- highcharts.js
- jquery.dataTables.js

The GUI for this application is intended only to support the usability of the web service, HTML coding along with CSS has been used to create the pages. To make the page responsive, Ajax calls has been used with jQuery libraries.

The landing page of the application should be mentioned in the beginning of the welcome file list in the web.xml file.

```
<welcome-file-list>
 <!-- the file user should be seeing on accessing the application -->
  <welcome-file>home.html</welcome-file>
  <welcome-file>readme.html</welcome-file>
</welcome-file-list>
```

A simple HTML code is used for the home page and is as shown in the below screenshot, figure 4l .The three links represents the three scenarios.
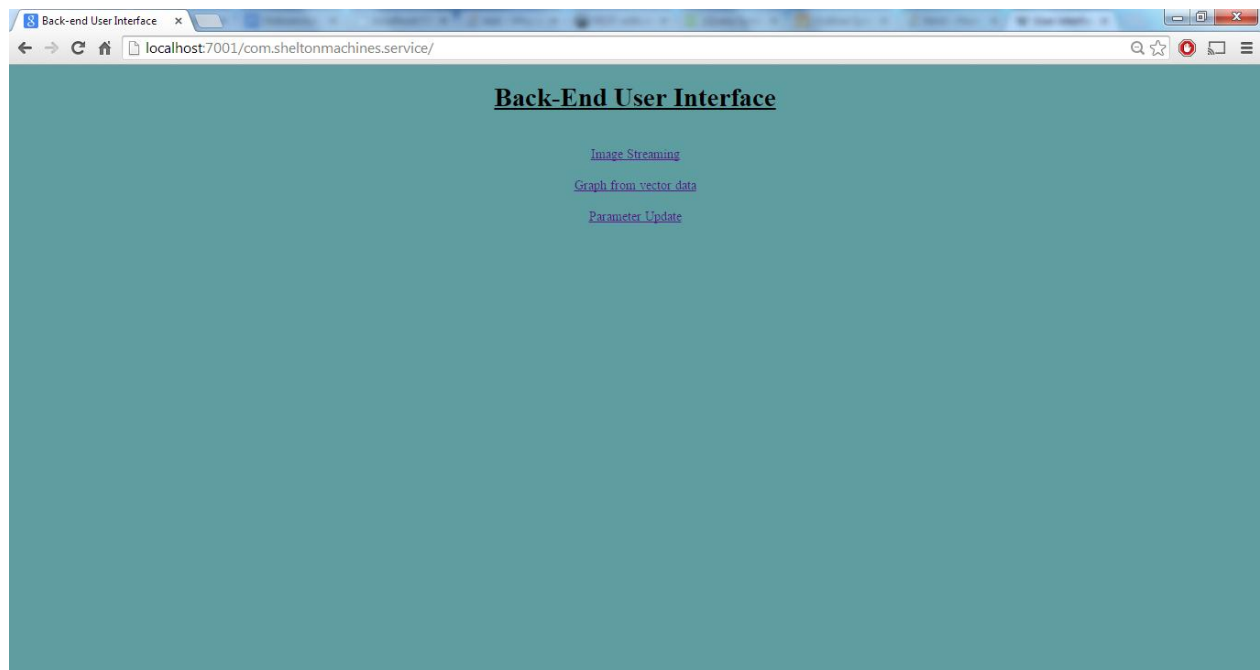

Figure 4l: Home page of the web application

'Image Streaming' function

When the user clicks on 'Image streaming' link, it takes to the html page for image streaming as shown in figure 4m.
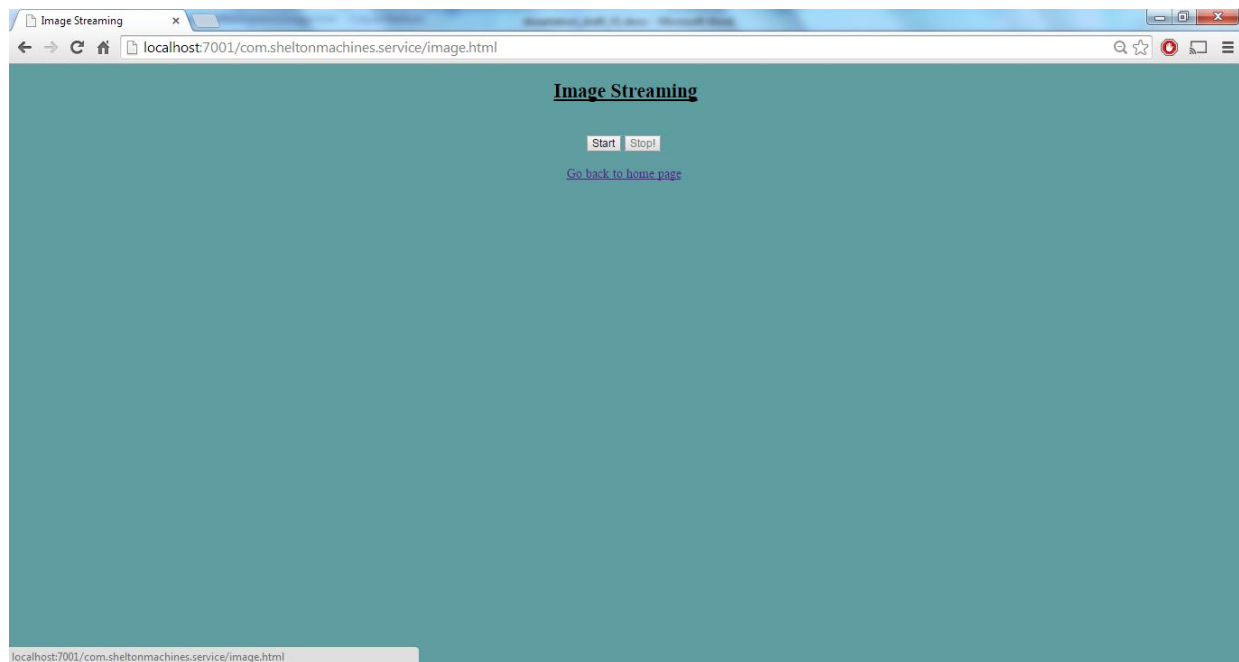
Figure 4m: <u>GUI for image streaming</u>

When the user clicks the 'Start' button, the client makes an Ajax call to the respective REST service to get the count of the images at that time. The code for the ajax call is as shown below:

[Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/WebContent/js/image.js]

```
//an Ajax request send to the URI of the resource to get the count of the image
paths

$.ajax
({type: 'POST',
  url:"http://localhost:7001/com.sheltonmachines.service/api/v1/images/count/",
  cache: false,
  success: function(data){
      callback(data[0].number);

      },
  // handling the error if request fails
  error: function(jqXHR, textStatus, errorThrown) {
                          console.log("Error!!"+ jqXHR.responseText);
              }
          });
      }
```

On success of that request, the client makes another Ajax call to the respective service URL to get the image path. The request is then send to the server. The server then responds with the image path in JSON format. The below code snippet shows how the jQuery client deciphers the image path from the JSON array and puts it into HTML code. It loops through the JSON array and identifies the JSON Object.

```
// Ajax request is send to the URI of the resource to get the image path
$.ajax({type: 'POST',
        url:
"http://localhost:7001/com.sheltonmachines.service/api/v1/images/path/"+index,
cache: false,
success: function(images){
$.each(images,function(i,image)
        {
      //changing the img element in the html code
      $("img").attr("src", image.IMAGE_PATH);
      }) ;
```

[Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/
com.sheltonmachines.service/WebContent/js/image.js]

In the URL mentioned in the above code, index is the image id to be send to the server using JQuery library.The index is then incremented by 1 till it reaches the image count unless the user clicks 'Stop' button.
The image storage can be accessed by both the server and client. For the purpose of the project, the images are stored in the C: drive of the machine.

All images for a set of Web applications can be stored in a single location, and need not be copied to the document root of each Web application that uses them. For an incoming request, if a virtual directory has been specified servlet container will search for the requested resource first in the virtual directory and then in the Web application's original document root. This defines the precedence if the same document exists in both places [DWA]. This can be achieved by using the virtual-directory-mapping element as shown below:
[Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/
com.sheltonmachines.service/WebContent/WEB-INF/weblogic.xml]

```
<!-- Virtual directory mapping to map the image path to the physical memory -->
    <wls:virtual-directory-mapping>
        <wls:local-path>C:\Users\Malu</wls:local-path>
        <wls:url-pattern>Pictures/*</wls:url-pattern>
    </wls:virtual-directory-mapping>
```

 On success of the Ajax call, the images are displayed in the browser. The two Ajax calls mentioned above are made every 200 milliseconds using a 'SetInterval' function, resulting in a live streaming of images.

In between, if the user clicks the 'Stop' button, the code would clear the interval Id as shown in the code below, thereby stopping the image from changing.

```
//On clicking the stop button intervalid is cleared
$('#stop').on('click',function(){
                        $('#start').attr("disabled", false);
                        $('#stop').attr("disabled", true);
                         clearInterval(intervalId);
                         index=1;
                        count=0;
```

```
                    });
```
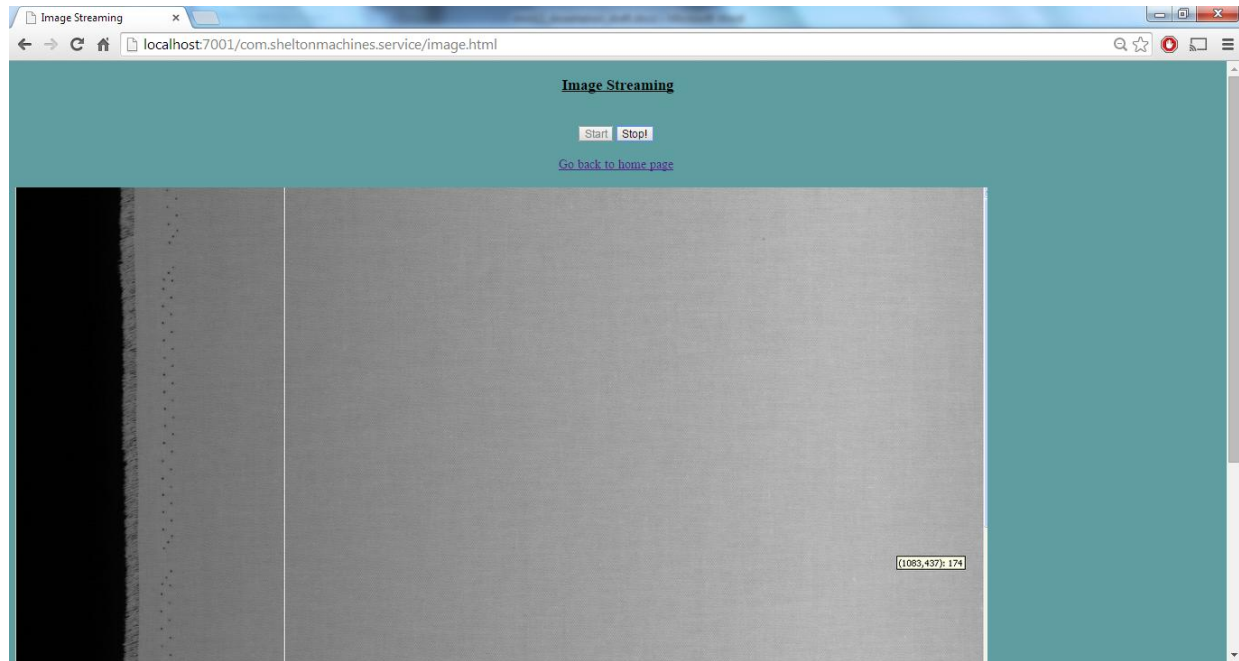
The resulting client page is as shown below in figure 4n:

Figure 4n: <u>Response page for Image Streaming</u>

'Graph Display' function

When the user clicks on 'Graph from Vector Data' link, the user shall be redirected to another page on which the process graph is generated from the list of vector data received from the server.

At first, the client makes an Ajax call to the service URL to find the checksum average of the vector data table and store this value to a variable. The details of the method is specified in section 4.2.3.The client makes this call every 1500 milliseconds and compare it with the previous data to check for a database change.

Initially and thereafter for every database change, client makes an Ajax call to the respective service URL using JQuery library to get all the vector data. The server then responds with the vector data in JSON format. On success of the Ajax call, the graph is displayed using the jQuery plugin 'highcharts.js'. The plugin needs to be modified to set the Y Axis, X Axis and the points. Highcharts has a tool tip feature to read the points of the graph.

Except X Axis Minimum, rest all points are present in the record set which is specified in the use case, section 3.5.3.

The X axis miminum is set to the minimum value of all the vector data. The below code shows the calculation of Xaxis minimum and parsing of points. [Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/WebContent/js/graph.js]

```javascript
// the points of the graph are pushed into  a points array from the json object
                success: function(obj){
                        points=[];
                        console.log(obj.length);//debug
                        for(var i=3; i<obj.length;i++)
                          {

                                points.push(parseFloat(obj[i].VectorData1));

                          }
                //calculating XAxis minimum
                for(var i=3; i<obj.length;i++)
                          {

                          if(obj[i].VectorData1 < XAxisMin)
                             {
                                     XAxisMin= obj[i].VectorData1;

                             }
                          }
```

The graph gets displayed in the Google chrome browser as below for the test data:

Figure 4o: Response page for graph display

To plot the graph the javascript file, 'highcharts.js' has been used which is downloaded from the below location
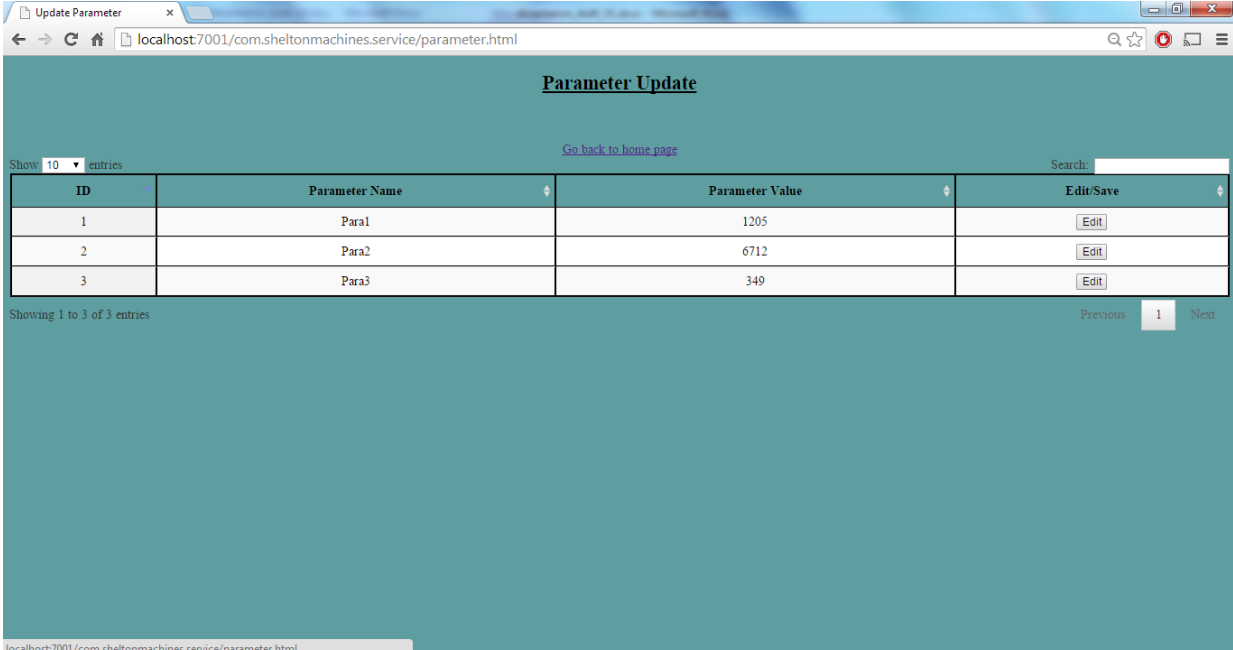
- HighCharts V4.0.4 -http://www.highcharts.com/download

The code to plot the points in the graph is referenced from http://www.highcharts.com/demo/line-basic

'Parameter' Update function

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 'Parameter Update' link,the user would be redirected to a page the client make an Ajax call to the respective service URL to get the parameters and then the server return all the parameters in the database. Then using the DataTables library , the parameters are displayed on the browser as shown in figure 4p:

Figure 4p: Response page for Parameter Update

The parameters can be sorted, filtered, edited and can be saved back to the database.

When the user clicks on 'Edit' button, the parameter name and value fields become editable and 'Edit' button changes to 'Save' button as shown below in figure 4q:
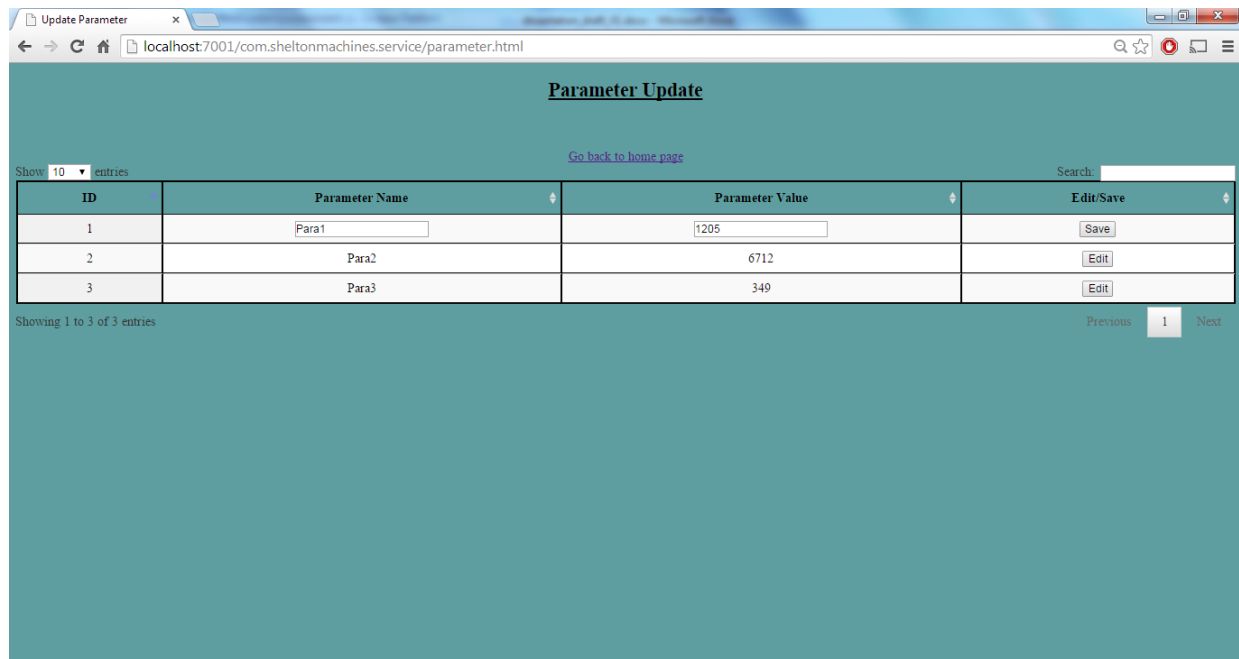

Figure 4q: <u>'Edit' button changing to 'Save' button</u>

When the user clicks on 'Save' button, the client makes an Ajax call to the respective service URL to update the database and send the parameters. The below code explains this [IE]. [Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ com.sheltonmachines.service/WebContent/js/parameter.js]

```
// this function sends the edited column values to the the server through the api URI
function saveRow ( pTable, nRow )
        {
                var data=pTable.fnGetData(nRow);
                var id= data[0];
                var jqInputs = $('input', nRow);
                pTable.fnUpdate( jqInputs[0].value, nRow, 1, false );
                pTable.fnUpdate( jqInputs[1].value, nRow, 2, false );

                 pTable.fnUpdate( '<input type="button" class="edit" value="Edit">', nRow, 3,
                false );
                 pTable.fnDraw();
                ajaxObj = {
                        type: "PUT",
                                url:
                "http://localhost:7001/com.sheltonmachines.service/api/v1/parameters/upd
                ate/" + id + "/" + jqInputs[0].value+  "/" + jqInputs[1].value,
                        cache: false,
                        success: function(data) {
                                //console.log(data);
                                alert( data[0].MSG );
```

```
        },
        error: function(jqXHR, textStatus, errorThrown) {
            console.log("Error!!"+ jqXHR.responseText);
        }

    };
```

Thus the detail of the edited row is send back to the server and then the server updates the database accordingly. If the database is updated successfully, server sends back a success message as shown in figure 4r.
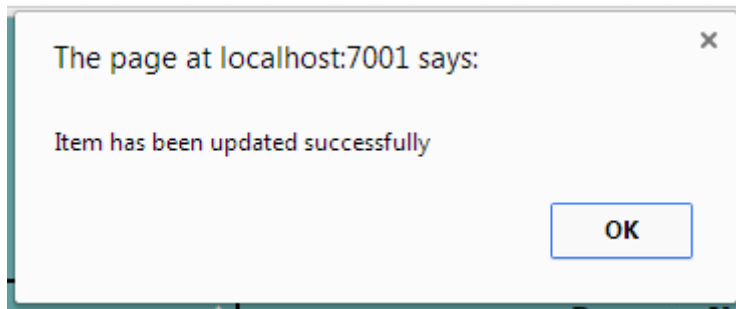

Figure 4r: Success message on parameter update

4.3   Testing

In Section 4.2, the client is exploiting the web service api, thereby showing that the api works. This section shows the test report on the functional and non-functional requirements of the web application exploiting the API.

Test Data

The below test data was stored in the database table conforming to the database set up in section 4.2.1.1.

Images – 12 images of size ranging from 250KB to 3MB. (Provided by SML)

Vector Data – 100 Points with YAxis minimum=0 and YAxis maximum=60 (Provided by SML)

Parameters – 3 integers (random values)

 Testing environment:

Operating System: Windows 7 Home Premium

Hardware: 4.00GB RAM, 32-bit (x86)

Default browser: Google Chrome, Version 39.0.2171.99 m

The application URL of the app when implemented using the above code-
http://localhost:7001/com.sheltonmachines.service/

Functional Testing

This has been done manually and the test report is shown in below table.

Test Case	Test Steps	Expected Result	Pass/Fail
Validate whether clicking on a start button starts the image loading from the first image till the last image.	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button	Image Streaming should start displaying the first image whose path is the first in the list of paths stored in the database and continue streaming until it reaches the end of the list	**Pass**
Validate whether clicking on 'stop' button stops the image streaming	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4.User clicks 'Stop' button	Image streaming should stop at the last displayed image.	**Pass**
Validate whether an update of an image source in database gets reflected at client side without a refresh	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4. Insert an image path in the database.	Image streaming should continue without interruption and should display the last image whose path is inserted in the database.	**Pass**

Validate whether the images are showing in its original size	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4. Check the size of the image stored.	The size of the image displayed in the browser should be the same.	**Pass**
Validate whether the user is able to start the image streaming after stopping it	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4.User clicks 'Stop' button 3.User clicks 'Start' button	Image streaming should start again.	**Pass**
Validate whether a graph is generated according to the vector data from the database	1.User hits the application URL 2.User clicks on 'Generate Process Graph'	The process graph should be displayed with the points from the database	**Pass**
Validate whether a tool tip is available in the graph to view the points.	1.User hits the application URL 2.User clicks on 'Generate Process Graph' 3.Hover over the graph	User should be able to view the points in the graph	**Pass**
Validate whether a change in database gets reflected at the client side without a refresh	1.User hits the application URL 2.User clicks on 'Generate Process Graph' 3.Update the database with new data	The graph should get regenerated without doing a refresh	**Pass**
Validate	1.User hits the	All parameters in	**Pass**

whether all parameters from the database are displayed on the client side	application URL 2.User clicks on 'Parameter Update' link	the database should get displayed.	
Validate whether editing and saving a parameter at client side saves it in the database	1.User hits the application URL 2.User clicks on 'Parameter Update' link 3.User clicks on 'Edit' button and edit the parameter value 4.User clicks on 'Save' button 5.Check the database table for the parameter	The user should be able to edit and save the parameter. The database should get updated succesfully	**Pass**
Validate a message from server is displayed on the client side on successfully saving a parameter	1.User hits the application URL 2.User clicks on 'Parameter Update' link 3.User clicks on 'Edit' button and edit the parameter value 4.User clicks on 'Save' button 5. Configure the message in the server to 'Item saved successfully' to send to client on successful updating of the database.	At client side the message 'Item saved successfully' should appear	**Pass**
Validate whether the user is able to repeatedly edit and save the	1.User hits the application URL 2.User clicks on 'Parameter Update' link 3.User clicks on	User should be able to edit the parameter again	**Fail**

parameter	'Edit' button and edit the parameter value 4.User clicks on 'Save' button 5.Click on 'Edit' button again		

Non-Functional Testing

The table below shows the report on testing the non-functional aspects of the web service API.

Tools used for the testing are:

- Chrome developer tool (debugging tool built into Google chrome)
- Fiddler4 (Available at http://www.telerik.com/download/fiddler)
- Windows Task Manager – CPU performance

Learning Effort

It had taken around 6 weeks of learning for the web service language and client side scripting.

Performance:

Table 4o shows the response for the resources in terms of time and bytes. Figure 4s shows a sample of the output in Fiddler

```
Request Count:    1
Bytes Sent:      516        (headers:516; body:0)
Bytes Received:  212        (headers:151; body:61)

ACTUAL PERFORMANCE
--------------
ClientConnected:      19:36:57.006
ClientBeginRequest:   19:37:11.043
GotRequestHeaders:    19:37:11.043
ClientDoneRequest:    19:37:11.044
Determine Gateway:    0ms
DNS Lookup:           0ms
TCP/IP Connect:       0ms
HTTPS Handshake:      0ms
ServerConnected:      19:36:57.582
FiddlerBeginRequest:  19:37:11.044
ServerGotRequest:     19:37:11.044
ServerBeginResponse:  19:37:11.055
GotResponseHeaders:   19:37:11.055
ServerDoneResponse:   19:37:11.056
ClientBeginResponse:  19:37:11.056
ClientDoneResponse:   19:37:11.056

      Overall Elapsed:      0:00:00.013

RESPONSE BYTES (by Content-Type)
--------------
      ~headers~: 151
application/json: 61
```

Figure 4s: Sample fiddler output for solution1

Resource	Average Response Time(milliseconds)	Response Bytes (for Headers)	Average Response Bytes (for application/json)
/images/count	36	151	31
/images/path	20	151	61
/vectordata/checksum	81	151	34
/vectordata	166	151	2,285
/parameters	32	151	194
/parameters/update	108	151	80

Table 4o: Performance table for solution1

Other performance parameters measured are listed below:

- Response time by the client to display the image was between 130- 500 ms
- Time between image changes =30ms.The below figure shows the transfer timeline of a 3MB image.
- Refresh rate of the image on a database change = 4sec approx.
- Refresh rate of graph =2 sec approx.

- Transfer time for an image= 31ms
- CPU load on Image Streaming : 50%
- CPU load on the display of graph: 27%

Operational

The functional test cases mentioned above has been successfully performed in all major browsers listed below:

- Google chrome 39.0
- Internet Explorer ,Version 11.0
- FireFox Version 31.0
- Safari ,Version 5.1.7
- Opera , Version 26.0

Maintainability:

The **project metrics** for this solution in implementing the three scenarios are shown below(this includes all the java files, javascript files, xml files and html files for all the three scenarios). This is calculated using the software - Understand SciTools (available at https://scitools.com/)

Project Metrics

Files:	21
Program Units:	1806
Lines:	16933
Blank Lines:	2261
Code Lines:	6974
Comment Lines:	7177
Statements:	7206

The code was **organised** in such a way that it can be easily maintained and audit (as shown in figure 4t). A **version number** is introduced in the 'api' so that the server side code can be changed without affecting the client side application.
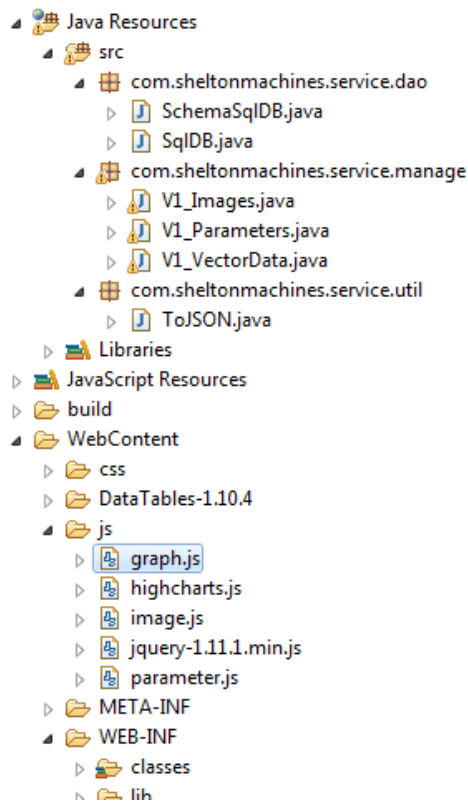
Figure 4t : _Structure of the package for solution1_

Proprietary rights:

All the software and tools for this solution are under free license to use.

5. Solution using ASP.NET Web API (C#)

This section describes the approach for the solution using ASP.NET Web API. Due to time constraints, only scenario 1 of section 3.4.3 has been taken into consideration.

5.1   Design

In this solution, the web service has been implemented using REST architecture as explained in section 3.2.3.

ASP.NET Web API has been selected to use in this solution.ASP.NET Web API is a framework for building web APIs on top of the .NET Framework. The processing model of a Web API is shown in figure and it explains the three layers: hosting, message handler pipeline, and controller handling [GPPHD].
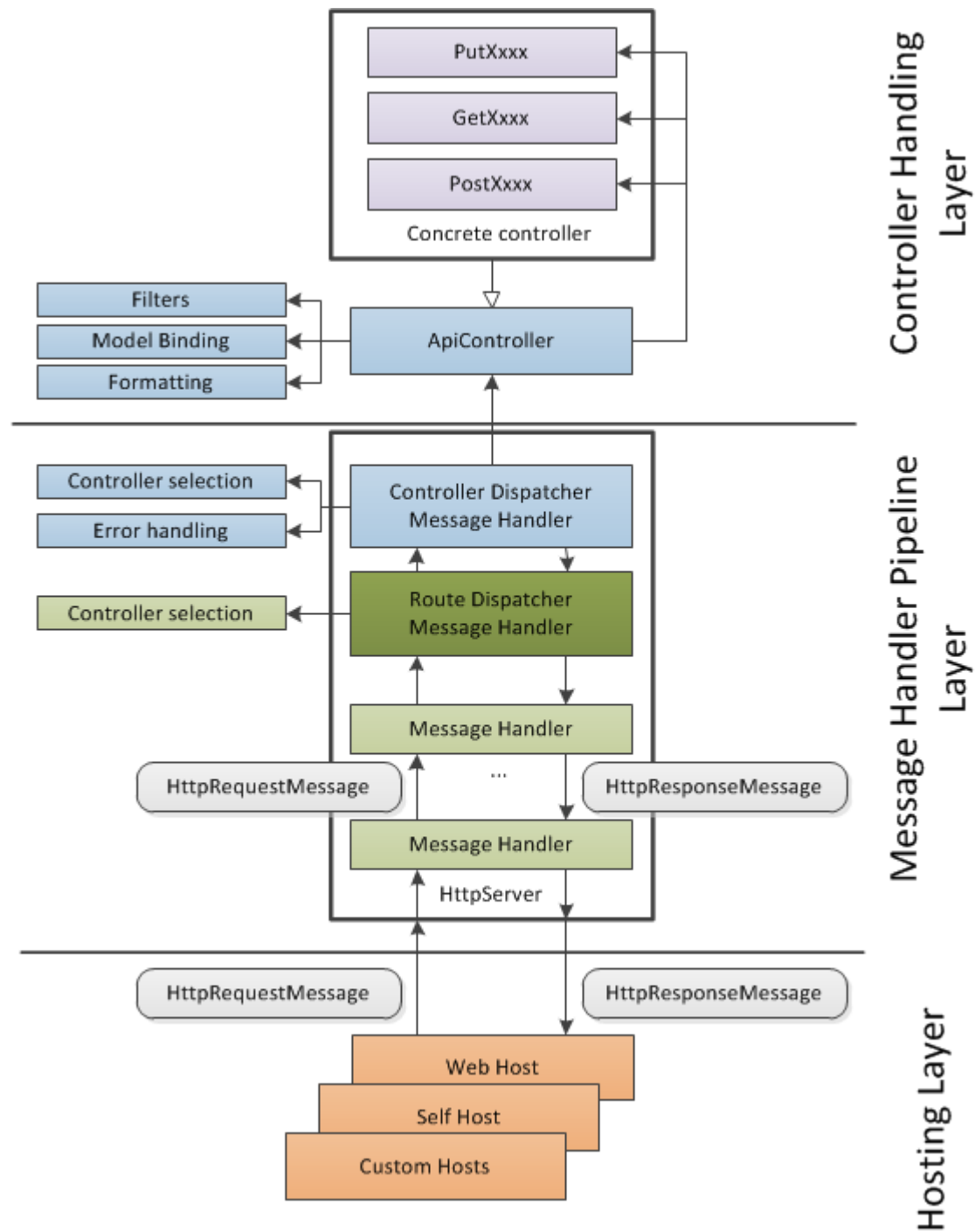
Figure 5a: <u>Simplified ASP.NET Web API Processing Model</u> [GPPHD]

ASP.NET Web API was previously called as WCF Web API and recently merged into ASP.NET MVC 4 beta. The ASP.NET Web API comes with its own controller called ApiController which should be used to create the REST service.

For the purpose of the project, only scenario1 of section 3.4.3 (Image Streaming) is taken into consideration for this solution. To create the web service, the resources and the actions performed over them to the HTTP methods and address are identified as below:

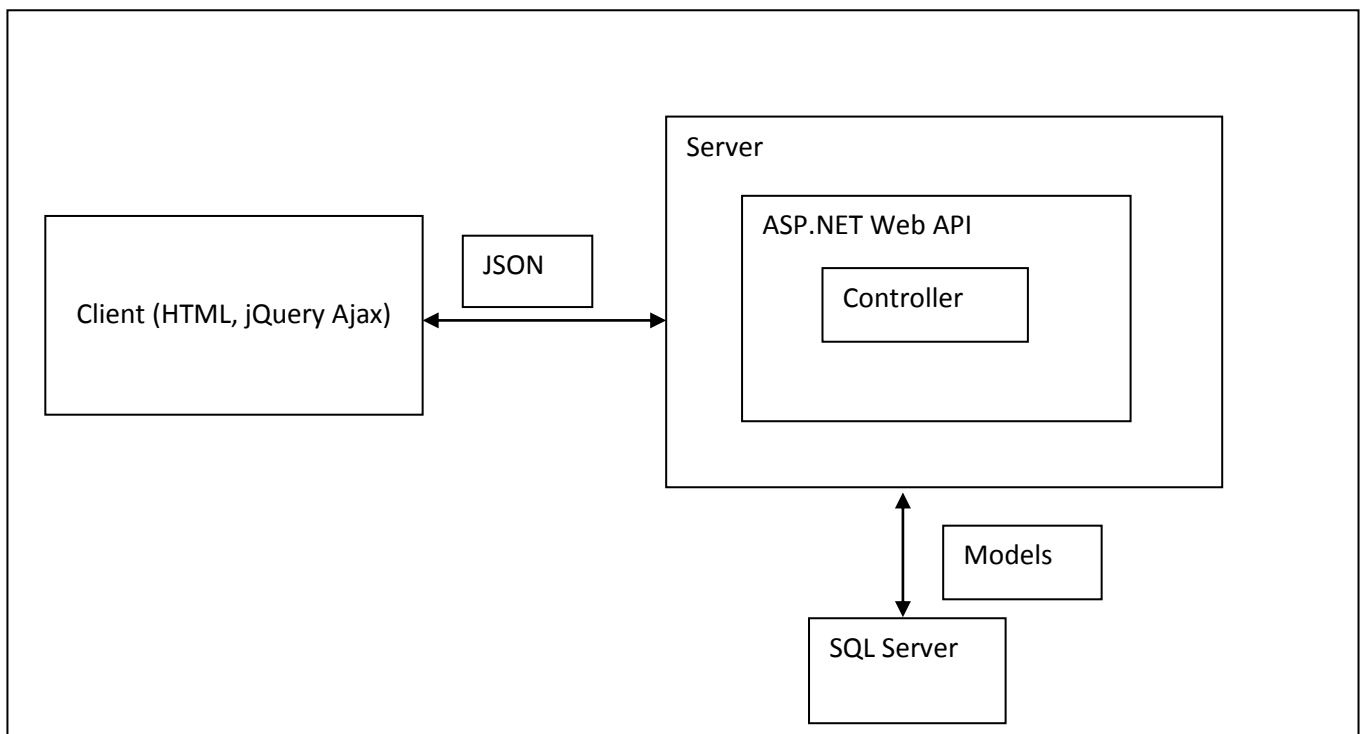| Action	Method	URI
Get the count of the images	GET	/count
Get a single task	GET	/id

Figure 5b: Design for solution 2

The front-end web pages(client) uses jQuery with Ajax to display the results. A *model* is an object that represents the data in your application. ASP.NET Web API can automatically serialize your model to JSON, XML, or some other format, and then write the serialized data into the body of the HTTP response message. As long as a client can read the serialization format, it can deserialize the object. Most clients can parse either XML or JSON [MW].

In Web API, a *controller* is an object that handles HTTP requests which inherits the '**ApiController'** class [MW].

The Web API 2 web services may be hosted in Internet Information Services (IIS) or self-hosted as a service or console application, typically under the Windows Server operating system.

Version 2.0 of ASP.NET Web API introduces the OWIN host adapter, available via the Microsoft.AspNet.WebApi.Owin package. This new alternative allows the usage of any OWIN-compliant host [GPPHD].

Software versions used in this solution are:

- Visual Studio 2013

- Web API 2

- .Net 4.5

5.2 Implementation

5.2.1 SQL Server

The same database has been used in this solution as explained in section 4.2.1.1.

5.2.2 Web server

IIS Express is the default web server for the web applications developed in Visual Studio 2013.

IIS Express is a lightweight, self-contained version of IIS optimized for developers. IIS Express makes it easy to use the most current version of IIS to develop and test websites. It has all the core capabilities of IIS 7 and above as well as additional features designed to ease website development including:

- It doesn't run as a service or require administrator user rights to perform most tasks.

- IIS Express works well with ASP.NET and PHP applications.

- Multiple users of IIS Express can work independently on the same computer.

  http://www.iis.net/learn/extensions/introduction-to-iis-express/iisexpress-overview

5.2.3 Service

Adding a model

As explained in section 5.1, a model is an object to represent the data in the application. The data is accessed from the database and is mapped to the model object to use in the application.

ASP.NET web API automatically serialize the data into json format and write it to the body of the HTTP response.

The below code shows the models created for the resources mentioned in section 5.1. [Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ BackendApp/Models/Image.cs]

```csharp
namespace BackendApp.Models
{
  // The properties of the resources are added to the data class
  public class Image
  {
     public string Id { get; set; }
     public string Name { get; set; }
     public string Path { get; set; }
  }

  public class ImageCount
  {
     public int Number { get; set; }
  }
              }
```

Adding a Controller

The controller defines two methods that return the image detail

- The GetImages method that return the count of the images
- The GetImagePath method that return the path of the image given an id.

Controller Methods	URI
GetImages	/api/images/count
GetImagePath	/api/images/*id*

The Web API route can be configured in the WebApiConfig.cs class as shown below. [Location in svn: https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ BackendApp/ /App_Start/WebApiConfig.cs]

```
            // Web API routes
             config.MapHttpAttributeRoutes();

             config.Routes.MapHttpRoute(
                 name: "DefaultApi",
                 routeTemplate: "api/{controller}/{id}",
                 defaults: new { id = RouteParameter.Optional }
             );
```

The below shows a code snippet of the controller class: [Location in svn:
https://campus.cs.le.ac.uk/svn/mrn12/code/tags/ BackendApp/
Controllers/ImagesController.cs]

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web;
using System.Web.Http;
using System.Data;
using System.Data.SqlClient;
using BackendApp.Models;

namespace BackendApp.Controllers

{
    /**
      * This class is the controller class for the image model
      * This class returns the SQL query result in json format
      *
      * **/
    public class ImagesController : ApiController
    {
        // Get api/images/count
        public IEnumerable<ImageCount> GetImages()
        {
            ImageCount[] count = null;
            //Getting the database connection
             SqlConnection con = new SqlConnection(@"Data Source=MALU-PC\SQLEXPRESS;Initial
            Catalog=TestDB;Integrated Security=True;");
            con.Open();
            //executing query to calculate the count of the image paths
             SqlCommand cmd1 = new SqlCommand("SELECT Count(IMAGE_PATH) AS number FROM
            Images", con);
            SqlDataReader dr1 = cmd1.ExecuteReader();
             //mapping the query result to model objects to get json format.
              while (dr1.Read())
            {
                count = new ImageCount[]
                {
                new ImageCount { Number = dr1.GetInt32(0) }

                };
            }
```

The above code also outlines the fact that the controller opens a connection to the database and query the database to get the result.

5.2.4 Client

Being used by any programming language is one of the greatest advantages of a RESTful Web Service.

Therefore, the same client implementation explained in 'Image Streaming function' of section 4.2.3 has been used here except the virtual directory mapping element. Here, the virtual directory mapping element has been configured in the 'applicationhost.config' file of IISServer as shown below for the scenario.

```
<application path="/Pictures" applicationPool="Clr4IntegratedAppPool">
    <virtualDirectory path="/" physicalPath="c:\users\malu\Pictures" />
</application>
```
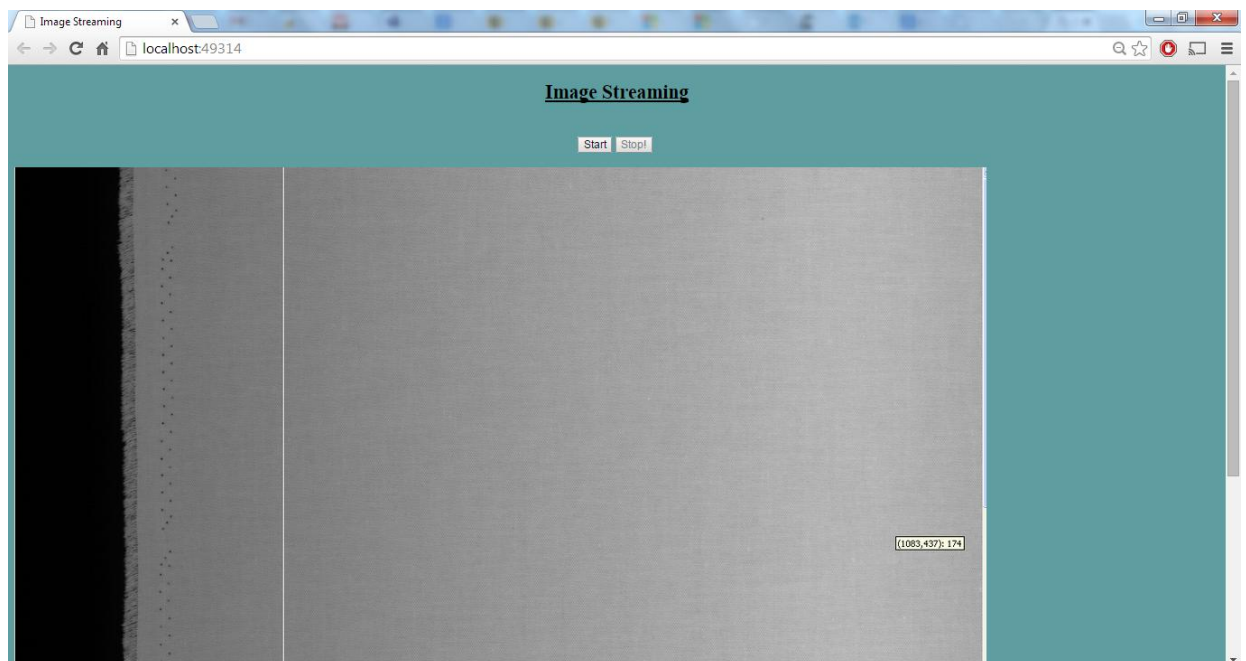
Running the application on a Google chrome browser produces the following output , figure 5a:

Figure 5c. Response page for Image Streaming using ASP.NET

5.3 Testing

In Section 4.2, the client is exploiting the web service api, thereby showing that the api works. This section shows the test report on the functional and non-functional requirements of the web application exploiting the API.

Test Data

The below test data was stored in the database table conforming to the database set up in section 4.2.1.1.

Images – 12 images of size ranging from 250KB to 3MB. (Provided by SML)

Vector Data – 100 Points with YAxis minimum=0 and YAxis maximum=60 (Provided by SML)

Parameters – 3 integers (random values)

The base URL of the app when implemented using the above code- http://localhost:49314/index.html

Testing environment:

Operating System: Windows 7 Home Premium

Hardware: 4.00GB RAM, 32-bit (x86)

Default browser: Google Chrome, Version 39.0.2171.99 m

Functional Testing

This has been done manually and the test report is shown in below table.

Test Case	Test Steps	Expected Result	Pass/Fail
Validate whether clicking on a start button starts the image loading from the first image till the last image.	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button	Image Streaming should start displaying the first image whose path is the first in the list of paths stored in the database and continue streaming until it reaches the end of the list	**Pass**
Validate	1.User hits the	Image streaming	**Pass**

whether clicking on 'stop' button stops the image streaming	application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4.User clicks 'Stop' button	should stop at the last displayed image.	
Validate whether an update of an image source in database gets reflected at client side without a refresh	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4. Insert an image path in the database.	Image streaming should continue without interruption and should display the last image whose path is inserted in the database.	**Pass**
Validate whether the images are showing in its original size	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4. Check the size of the image stored.	The size of the image displayed in the browser should be the same.	**Pass**
Validate whether the user is able to start the image streaming after stopping it	1.User hits the application URL 2.User clicks on 'Image Streaming' link 3.User clicks 'Start' button 4.User clicks 'Stop' button 3.User clicks 'Start' button	Image streaming should start again.	**Pass**

Non-Functional Testing

The table below shows the results of testing the non-functional aspects of the web service API.

Tools used for the testing are:

- Chrome developer tool (debugging tool built into Google chrome)
- Fiddler4 (Available at http://www.telerik.com/download/fiddler)
- Windows Task Manager

Learning Effort

It had taken around 2 weeks to implement the image streaming alone.

Performance:

The table 5a shows the response for the resources in terms of time and bytes.

Resource	Average Response Time(milliseconds)	Response Bytes (for Headers)	Average Response Bytes (for application/json)
/api/images	177	399	15
/api/images/1	148	403	64

Table 5a. Performance table for solution 2

Other non-functional parameters measured are listed below:

- Response time by the client to display the image was between 200-700 ms
- Time between changing from 1st image to 2nd image=487ms.The below figure shows the transfer timeline of a 3MB image.
- Refresh rate of the image on a database change = 7sec approx.
- CPU load on Image Streaming : 90%

Operational

The functional test cases mentioned above has been successfully performed in all major browsers listed below:

- Google chrome 39.0
- Internet Explorer ,Version 11.0
- FireFox Version 31.0
- Safari ,Version 5.1.7
- Opera , Version 26.0

Maintainability:

The **project metrics** for this solution in implementing the one scenario is shown below(this includes all the C# files, javascript files and html files).This is calculated using the software - Understand SciTools (available at https://scitools.com/)

Project Metrics
Files:	8
Program Units:	582
Lines:	9475
Blank Lines:	1618
Code Lines:	6318
Comment Lines:	1506
Statements:	2896

The code was organised in such a way that it can be easily maintained (as shown in figure 5d.
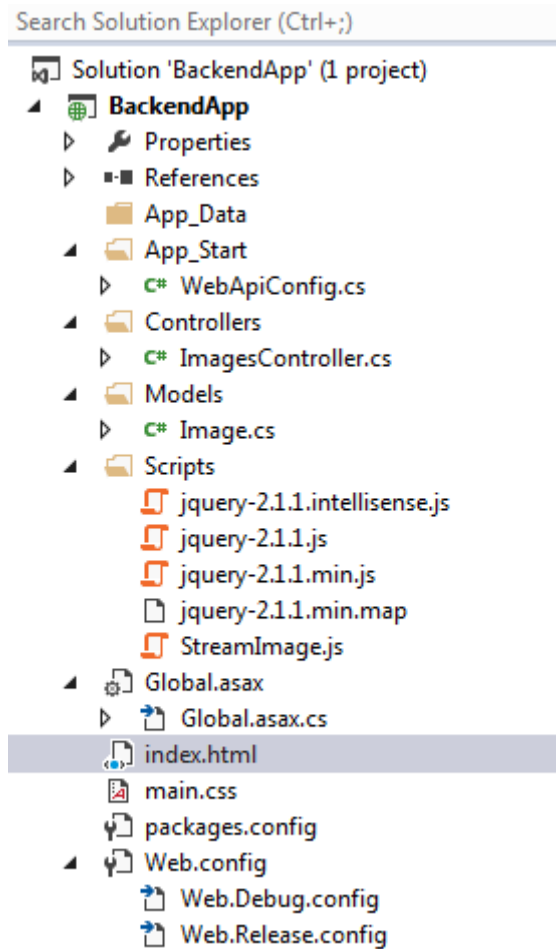
Figure 5d : Structure of the package for solution 2

Proprietary rights

All the software and tools for this solution are under free license to use.

6. Comparison and Evaluation

This section does a critical comparison of the above two solutions.

Figure 6a and 6b shows the difference in transfer timeline for the same image (of size 3MB) using java and .NET respectively.

TRANSFER TIMELINE

| | 1s | 2s | 3s | 4s | 5s | 6s | 7s |

Klopman_raw_image.jpg

Figure 6a. Transfer Timeline for an image of size 3MB in solution 1(Java)

TRANSFER TIMELINE

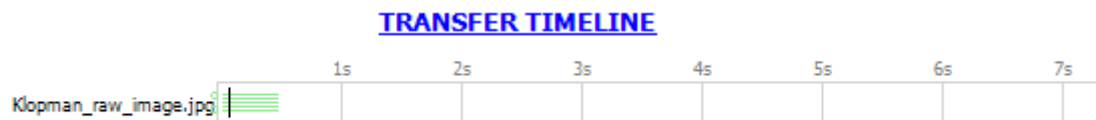| | 1s | 2s | 3s | 4s | 5s | 6s | 7s |

Klopman_raw_image.jpg

Figure 6b. Transfer Timeline for an image of size 3MB in solution 2 (.NET)

Table 6a shows the comparison of most relevant factors of 'Image Streaming function' based on the test results of both the solution:

	Solution1	Solution 2
Learning Effort	4 weeks	2 weeks
No:of code lines for the solution	6974(for three scenarios)	6318 (for one scenario)
Met functional requirements	Yes	Yes
Response Bytes (content: Headers)	151 (Figure 6c)	399 (Figure 6d)
CPU load on the process	50%	90%
Refresh rate of the image on a database change	4sec	7 sec
Time elapsed to display an image at client side	130-500 ms	200-700 ms

Table 6a. Comparison of Test Results
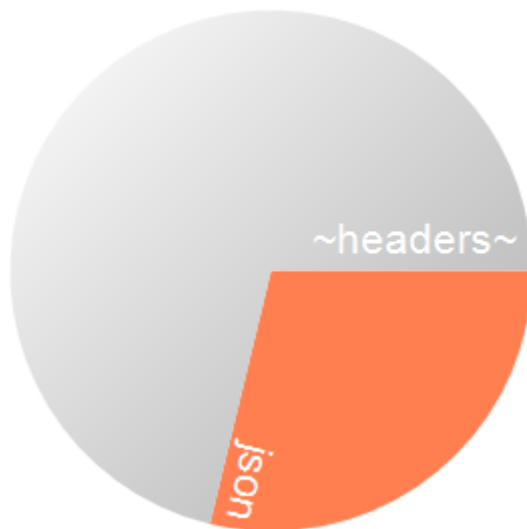
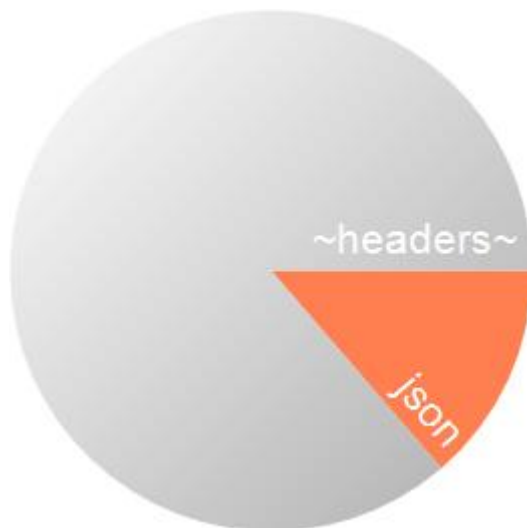Figure 6c. <u>Response Headers and JSON using solution1(Java)</u>

Figure 6d. <u>Response Headers and JSON using solution2(.NET)</u>

Other facts that can be compared against the two approaches are listed in Table 6b.

	Solution1	Solution 2
Platform Dependency of client	Platform Independent	Platform Independent
Platform Dependency of server side code	Any Operating System	Any operating system that has an installed CLR
Ability to communicate to C++ code.	Possible (Eg: using Java Native Interface(JNI))	Possible (Eg: Using Platform Invocation Services (P/Invoke))

Table 6b. Comparison in general

This evaluation shows that, to design a RESTful web service for the solution, using java framework is significantly faster than using one written using ASP.NET.

7. Conclusion

7.1 Summary

The web application developed was able to successfully implement the scenarios described in section 3.4.3 and was able to conclude that a RESTful web service API using Java would be a solution to the problem. However second solution with which it was compared could implement only one scenario due to time constraints. Due to the scope of the project, various user interfaces in the current system were not implemented, however major functionalities were implemented with a future proof design which would be a stepping stone to a complete web based user interface that can be delivered in many different OS platforms.

7.2 Improvements and future works

This section outlines the areas that a future similar project could improve on this one.

Below are a few improvements that could be made to the methodology and implementation of a future similar project:

-could be developed into a single page web application

-The client is doing long-polling using a 'setInterval' method of jQuery library with an interval of 200 milliseconds for the Ajax calls. If the server takes longer than that time to return the client call, it would end up with a bunch of Ajax request queues that would not return in the same order. So the best bet would be to use web sockets instead of long-polling.

-could expand the number of languages that are tested by including modelling languages such as PHP and JavaScript framework such as angular.js

-could consider security as this was not addressed in this project.

-another area that could be broadened is in testing the server and the client on different operating systems.

There are a few works that could be done to the implementation of the web application in the future:

-In future, the backend of the current surface inspection system will be re-written in C++and the web service should be able to communicate with the C++ code which can then be achieved with the help of a wrapper functions available in the market.

-The C++ code would update the product database and the web service could be re-written with the real database tables. The SQL queries could be updated accordingly.

-A more sophisticated GUI could be developed at the client side which is as good as the current desktop application.

References:

[AR] Alex Rodriguez.” RESTful Web services: The basics” .Last modified 06 November 2008.*Accessed on 14/07/2014* <http://www.ibm.com/developerworks/webservices/library/ws-restful/>

[AS] Aaron Skonnard, Pluralsight . "The Web Platform", Last modified October 2008. A Guide to Designing and Building RESTful Web Services with WCF 3.5. *Accessed on 26 /08/2014.* <http://msdn.microsoft.com/en-us/library/dd203052.aspx>

[BS] Bruce Sun, Java Architect, National Center for Atmospheric Research, A multi-tier architecture for building RESTful Web services. *Accessed on 06/11/2014* <http://www.ibm.com/developerworks/library/wa-aj-multitier/ >

[CA] Category: Ajax .*Accessed on 29/09/2014* <http://api.jquery.com/category/ajax/>

[CRA] Creating a REST service using ASP.NET Web API. *Accessed on01/01/2015* <http://www.codeproject.com/Articles/426769/Creating-a-REST-service-using-ASP-NET-Web-API>

[DH] Dion Hinchcliffe. "REST vs SOAP". April 2005. *Accessed on 19/07/2014* < http://zgia.net/?p=30 >

[DWA] Developing Web Applications for WebLogic Server. *Accessed on 26 /08/ 2014.* <http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/weblogic_xml.html#1039396 >

[GPPHD]Glenn Block, Pablo Cibraro, Pedro Felix, Howard Dierking, and Darrel Miller Designing Evolvable Web APIs with ASP.NET. e-book. *Accessed on 10/11/2014*<http://chimera.labs.oreilly.com/books/1234000001708/index.html >

[HJ] H. Janicke. "Requirements Management". CO7308 Software Measurement and Quality Assurance

[IE] Inline Editing .Last modified 31st May 2012.*Accessed on 10/11/2014*<http://datatables.net/blog/2012-05-31>

[JAD] "jQuery:Advantages and Disadvantages" *Accessed on 29/09/2014*<http://www.jscripters.com/jquery-disadvantages-and-advantages/>

[JD] Julien Dubois. "Part 1: Introduction to Jersey—a Standard, Open Source REST Implementation". Last modified June 2010 . *Accessed on 27/09/2014* <http://www.oracle.com/technetwork/articles/javaee/jersey-part1-159891.html >

[JFFA] "JSON: The Fat-Free Alternative to XML".*Accessed on 07/12/14* <http://www.json.org/xml.html>

[JK] Jonathan Chaffer; Karl Swedberg ."Learning JQuery: better interaction, design, and web development with simple JavaScript techniques", Packt Pub. 4th Edition, 2013

[JQS] jQuery Syntax. *Accessed on 26 /08/2014.* < http://www.w3schools.com/jquery/jquery_syntax.asp >

[JUG] Jersey 1.8 User Guide. <https://jersey.java.net/nonav/documentation/1.8/user-guide.html#jax-rs>

[KJ] Kanjilal, Joydip. ASP.NET Web API: Build RESTful Web Applications and Services on the .NET Framework Packt Publishing Ltd, December 2013

[LV] Lars Vogel . Version 2.4, Last modified 20 August 2014 RESTful web services with Java (Jersey / JAX-RS). Accessed on 22/11/2014 <http://www.vogella.com/tutorials/REST/article.html>

[MP] Michael.P.Papazoglou "Web Services & SOA Principles and Technology". SECOND EDITION, PEARSON

[MW]Mike Wasson. Getting Started with ASP.NET Web API 2 (C#) *Accessed on 29 /11/2014* <http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api >

[QA] Qi Yu, Athman Bouguettaya. "Foundations for Efficient Web service Selection". SPRINGER US, 2010

[RR] Robertson, James and Robertson, Suzanne. "Volere Requirements Specification Template." 2014. Edition 17.
Principles of the Atlantic Systems Guild Limited. *Accessed on 26/08/ 2014.* <http://www.volere.co.uk/template.htm>.

[SJ] Sandoval, Jose , RESTful Java Web Services : Master Core REST Concepts and Create RESTful Web Services in Java . Packt Publishing, November 2009

[SM] Sarah Maddox."API Types" .*Accessed on 07/12/14*<https://ffeathers.wordpress.com/2014/02/16/api-types/ >

[SP] Snehal Mumbaikar, Puja Padiya .”Web Services based on SOAP and REST Principles”, *Research paper, Department of Computer Engineering, R. A. I. T.* International Journal of Scientific and Research Publications, Volume 3, Issue 5, May 2013. *Accessed on 25/09/2014*. <http://www.ijsrp.org/research-paper-0513/ijsrp-p17115.pdf>

[SVS] Shelton Vision Systems. ” Taking surface inspection to a new level” *.Power point slides provided by Shelton Machines Ltd.*

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

<u>Appendix A</u>

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

SQL Management Studio Installation Instructions

<u>Steps:</u>

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: **********