SWE400–1501A–01:Software Construction

# Re–Enigneering a CRUD Web Service Client, IP5

Jack Simmonds

Professor:Carol Tilden

2/9/2015

[Re–Enigneering a CRUD Web Service Client, IP5 1](#_Toc411343641)

[1. Project Outline 3](#_Toc411343642)

[1.1 Component Diagram 5](#_Toc411343643)

[1.2 Entity Relationship Diagram 6](#_Toc411343644)

[2. API Evaluation 7](#_Toc411343645)

[XML 7](#_Toc411343646)

[JDBC 10](#_Toc411343647)

[Interface List<T> 12](#_Toc411343648)

[javax.swing.Jrame 14](#_Toc411343649)

[javax.swing.UIManager 16](#_Toc411343650)

[3. API Efficiency, Reliability, and Maintainability 18](#_Toc411343651)

[List<T> 18](#_Toc411343652)

[Efficiency: 18](#_Toc411343653)

[Reliability: 19](#_Toc411343654)

[Maintainability: 19](#_Toc411343655)

[JFrame 21](#_Toc411343656)

[Efficiency: 21](#_Toc411343657)

[Reliability: 22](#_Toc411343658)

[Maintainability 23](#_Toc411343659)

[4. Configuration and Change Management 25](#_Toc411343660)

[Overview: the Change Management Process 25](#_Toc411343661)

[Project Change Management Diagram 26](#_Toc411343662)

[Traceability Matrix 28](#_Toc411343663)

[Sample Test Script 29](#_Toc411343664)

[Sample Test Case 30](#_Toc411343665)

[5. Software Construction Tools and Techniques 31](#_Toc411343666)

[Screenshots: 32](#_Toc411343667)

[*Display all customers* 33](#_Toc411343668)

[*Display all Products* 34](#_Toc411343669)

[*Find by ID Range* 35](#_Toc411343670)

[*Find by ID* 36](#_Toc411343671)

[*Edit Database Entry* 37](#_Toc411343672)

[*Delete* 38](#_Toc411343673)

[*Layout Update* 39](#_Toc411343674)

[Version Control and Change Management 40](#_Toc411343675)

[References 41](#_Toc411343676)

# 1. Project Outline

In IT351 we completed a number of applications. During week 5 we created a web services SOAP (Simple Object Access Protocol) based application which is designed to provide CRUD (Create, Read, Update, Delete) functionality to its users. The application works fine, but due to the time constraints of the course (we developed two different types of applications in week 5 alone) the client side of the application was minimally developed. The standalone client, as created for the course, had just enough functionality to test that it was successfully connecting to the application’s web service (via client/server connection). We developed one further test to ensure that the client was also able to retrieve data from the application’s database, and print out object names: for the objects stored in each of two database tables. I have since gone ahead and developed the client to be fully functional, but it uses the console for input and ouptut.

Earlier in IT351 (week 3) we developed a GUI based application, using the NetBeans palette to create a user facing, JFrame Form based interactive tool: for implementing CRUD funcitonality with a database. I would like to combine lessons learned from that phase and the SOAP app to provide a standalone client for the web service application. In brief, add a user friendly, stand–alone interactive user interface that provides database/CRUD services through use of the SOAP application’s web service. The client already has fully developed WSDL files, and I have created methods which use them to implement CRUD functionality (outside the course).

The architecture of the system is complex enough to satisfy the requirements for SWE400. There are numerous APIs in use within the application. The client alone is using the Java library Generic List API, and will be accessing many classes from the Java awt and Swing libraries as the GUI is built. The Application implements classes from Java io, Javax persistence, the Maven framework, JAX-WS (the core API for the application on both the server and client sides) as well as the XML API through the javax xml bind/annotation API (Enterprise Java Beans). The system also employs a MySQL server to hold and run the database. This brings the JDBC API and JNDI into play within the application. Enterprise Java Beans are used to generate entity beans (the persistence layer). The entity beans are then used to generate the web service Java files, and they in turn form the basis of the generated Enterprise Beans. On the client side, WSDL files (XML) are generated from the web services along with sets of Java files generated based on the methods detailed in the Enterprise Java Beans and XML in the WSDL files. The Java files are annotated to provide hooks for the XML that is generated and used for messaging (as well as the test web pages).

I will continue to use NetBeans to complete the project, and as the tool of choice while developing the reengineering plan document for SWE400. Architecturally, the system is client/server with the addition of a database and its server. The client, application server and database server would normally be deployed on different machines. Communication is effected between them under the SOAP protocol which passes XML syntaxed messages in serialized form. Communication between the application side and the database is handled by text messages (also serialized) generated according to the JDBC API. JDBC uses java to encapsulate the MySQL commands used (and returned) by the database, and JNDI to create and store the connection between the app server and the database server.

## 1.1 Component Diagram



GUI Client

Client Application Device

Database Server

Application Server







During the development and testing phase; the Client, application server and database server: are all configured to run on the same machine. Once all tests are passed and the product is deliverable, it can be configured to meet the buyer’s needs.

## **1.2 Entity Relationship Diagram**

Entity Manager

Abstract Facade

XML

Client GUI

This diagram represents the flow of data between elements within the application. The client app contains WSDL (web service definition language) files which communicate with the server side application. The web service in the application communicates with the database. The database provides the persistence layer. Customers and products are treated as entities which are handled and passed as serialized objects in messages between components.

Tables

Entity

Entity Facade

Database

# 2. API Evaluation

This section provides for an overview of APIs used by the project application. Their design considerations, suitability, completeness and entry points will be discussed and diagrammed. The application contains many APIs of which the following represent the major ones involved. In particular, the main purpose of the re–engineering project is to provide a user facing, client GUI. While all APIs covered are crucial to the functionality of the client app, the GUI is the main focus of the project. This makes the JFrame API of primary importance here.

## XML

Extensible Markup Language represents a syntax API. The syntax involves creating tags with an inherent hierarchy. Tags are typically semantically named according to their functionality. The innermost XML tags are used to contain data, either as individual datum, or as key/value pairs. As an example, an object may be written as:

<customers>

<customer>

<name>Some Name</name>

<email>name@mail.com</email>

<phone>123-456-7890</phone>

</customer>

</customers>

The structure is reminiscent of Matroska dolls. Each outer tag represents a higher level container for data contained within a final, innermost set of tags. This syntax may be used to pass messages, store data, or create data presentations on HTML pages. Because XML is a syntax recognized by all major operating systems and web browsers, it creates a cross platform bridge when creating applications. The syntax also provides an inherent hierarchical structure. Because of this, data contained within xml tags can reflect the same hierarchy used to define objects in Java, or other languages.

The xml example above may describe the layout of an object and its properties on a web page, an object within a list of objects in a collection sent as a SOAP or REST message, the properties and values of an object within a Java collection, or data to be stored in a database (using the tag names as database, table and table field names). The specific data shown above may easily be replaced with variable names. A syntax commonly used in XML is to place property/value pairs within tags: <tag>property : value</tag>. This format aligns well with another syntax API, JSON. JSON uses different punctuation characters, and doesn't have the built in hierarchical property, but is commonly used for storing collections of objects.

XML is used in the project web service and client through the implementation of JAX-WS (java.javax.xml.ws.Service where class Service extends java.lang.Object, Oracle 2011). Java web service classes for customers and products are annotated to effect binding of data to XML from Java. These annotated classes contain references to the methods contained in the Java object classes (customers and products) along with their object facade superclass (both classes involve the same CRUD methods, but with differing object properties and values).

The annotated web service classes are used to construct the WSDL files (XML) in the client application. The WSDL files are used to generate Java files for each functional method of the web service, which in turn are accessed through Java method calls within the client application. While there are a number of steps involved, the way to view this is: Java to annotated Java to XML to annotated Java to Java, going from server to client applications, and the reverse. The data sources for this project application are a database on the server side, and user input and output on the client side. Viewed this way, the entry point for XML in the application is the annotated .java files which are used to build their XML counterparts.

Specifically, this project is concerned with re-engineering the client side application, but the services provided to the user (by the client) are actually defined and executed (in part) from the server side application. Accordingly, some of the APIs discussed here are used by either the server app, the client app, or both. In the case of XML, it represents a part of the message syntax used by both when sending and receiving messages (serialized customer & product objects). The flexibility of XML to provide a link between server based applications and client applications (written in a language of choice) makes the API; suitable to the task at hand, complete in it's ability to provide a messaging format need, and it fits seamlessly into the overall design of a client/server web service application.

Server side:

SOAP messages and WSDL files (client)

Client side:

## JDBC

Java DataBase Connectivity— provides an interface for connecting Java functional code and a MySQL database. The API describes a syntax for embedding MySQL queries within Java programming statements. JDBC goes hand in hand with the JNDI interface (Java Naming Directory Interface), which provides an API for setting and tracking named connections (URLs, COS, LDAP, NIS) within a distributed application (Sundsted). The project application makes use of DNS to bind a Java name reference to a URL, username and password. Between the two interfaces: Java can be written that connects an application to a MySQL database (JNDI) and contains the syntax for creating SQL queries which are embedded (as text strings) in Java methods (JDBC).

The JNDI syntax involves assigning (binding) a Java field name which references the connection within subsequent code where the connection is needed. The connection is specified by naming the database server, the URL and port for the server, the database name, the username and password of an authorized user. This information is stored using the SQL DriverManager and accessed through the SQL Connection interface. Once the connection is defined the database tables can be accessed using Java String fields assigned with text that matches SQL query statements (JDBC). The JDBC API contains interfaces and methods for defining database table access (createStatement(), executeUpdate(), executeQuery(), ResultSet, Statement). The interfaces are used to type fields which contain sql queries, and the values returned by those queries from the database.

JDBC is suitable to the task of providing a bridge between Java coded methods and a database (in this case MySQL). The task of allowing Java methods to manipulate data within an SQL database is complete in providing all the necessary functionality for executing the needed SQL queries. Architecturally set, as a bridge between Java and SQL JDBC provides the necessary design element to effect CRUD web services between an application server and a distributed client base. The entry point within the application is the entity beans which contain the definition of the persistence context.

Server side application web service:

## Interface List<T>

The List interface is part of the java.util package (Oracle, 1993 &2014) and an extension of the Collection framework. List defines the use of ordered datastructures that handle arrays, lists, or trees of programmer defined objects (sequential collections). The interface contains all the needed methods for manipulating objects within various types of collections. Among these methods are those needed to perform CRUD tasks with the customer and product objects contained in the project application (iterator, add, remove, set, size, contains, equals, sublist, and clear).

The client application accesses the database tables (the connection) and stores their data as objects (ResultsSet). The ResultsSet is assigned to a List. The List can then be manipulated according to a set of methods (the List methods above) which provide the necessary CRUD functionality (create, update, read by id, read all, range, delete). Each of these methods has it's counterpart in SQL queries contained in the server-side application's web service (the JDBC interface and methods).

The List interface provides a suitable and complete solution to handling database tables as containers of objects which can be iterated over to perform needed mutations to data contained within those tables. By architectural design, the List objects are placed between the incoming and outgoing client messages, and the methods that make use of the message contents (customer and product objects). These objects make up the targets and resulting message contents before and after mutation by the client application's CRUD methods. The entry point for the List interface is in the client application's CRUD methods. Each method calls for a connection to the application web service (and accordingly the database). The returns from these calls, or the data sent (depending on the type of method: update, read, create, …) accesses a list or list object provided by using the List interface.

Client application:

List API implemented

## javax.swing.Jrame

The swing package, and its extension: the JFrame API, allow for ready creation of interactive user interfaces for the collection and display of data. Structurally, JFrame (and the swing package) are an extension of the java.awt (abstract window toolkit) Frame class API. JFrame provides a root level window with methods to control its display visibility, adding, removing, and layout of elements within the container (Oracle, 1993 & 2014).

This project is primarily concerned with providing a user GUI for a distributed client of a server based application. The JFrame API provides the container and elements needed to create a form for user input and output display. The form will use JTextfield inputs, a JTextarea display field, and JButtons to activate handlers which interact with; the application's functionality, the web service application, the database and the user.

The JFrame provides a complete set of form elements needed to provide the intended database services to the user. Combined with the UIManager discussed below, it provides a suitable cross-platform solution to giving the end user a familiar interactive interface for viewing and modifying the data tables to which the web service connects. By design, the JFrame is the final structural component of the database/server/client system provided by the web service. It also makes up the primary goal of the re-engineering project defined by this document: to provide an intuitive user facing GUI allowing complete access to a customer and product database. The entry point for the JFrame API is the creation of a JFrame Form application. It is the first step in creating the client app. Once the JFrame is created, JFrame form elements are added to it. Handlers for those form elements are bound to them. The handlers provide the application's CRUD functionality.

Client application GUI:



## javax.swing.UIManager

The UIManager API is used to control the look and feel of GUIs to make them more attractive and familiar to the user (Oracle, 1996, 2014). The UI manager helps to create interfaces that the user will accept as typical of the operating system they are using. It also allows developers to modify system elements to fit particular design needs of a project. For instance buttons in an application: may have the shape typical of a Windows, Linux, or OS X based system. However, the developer may also access the properties of those buttons to effect changes to their default size, colors and shape.

For the project at hand, the GUI look and feel will be handled using a set standard of values for each of the form elements involved (Nimbus laf). Although, using the UIManager's methods, changes could be made to the values of properties stored in the Nimbus list of properties and values associated with JFrames and all of its content elements (combobox, textfield, textarea, buttons, borders, background colors, margins, padding, etc).

The UIManager API provides a complete set of methods for controlling the appearance and interactive functionality of form elements contained in a JFrame based application (connecting GUI elements to JFrame handlers). The project aim is to provide a distributed GUI client which will act as the user interface to a server based web service. It provides for the suitable implementation of a set of interactive features which make the client intuitive for users who have no knowledge about the underlying programming and processes involved in the application. Architecturally the GUI provides the final design element in the implementation of a client/server CRUD application. The API provides the code needed to make the application cross-platform compatible with a familiar look and feel for its users. This saves the developer time and effort which would otherwise add significantly to the cost and schedule needed to complete the project. The entry point for the UI manager is the instantiation of a JFrame object. When the GUI is instantiated for use by the user, the UIManager is called to define the look and feel of the JFrame and all the form elements it contains.

Client GUI:

GUI instantiated at runtime

# 3. API Efficiency, Reliability, and Maintainability

## List<T>

### Efficiency:

#### Strengths:

1. List allows for direct access to the collections of table objects, imported via the web service, for use within the various CRUD methods (the tables can be imported directly into a List object).

2. Because the data set is small, processing time is saved by not converting the List into a binary tree, or other data structure.

3. List<T> contains all the methods needed for manipulating the database objects as required by the web service (find, findAll, remove, sublist, iterator, add).

4. List (and accordingly all of its methods) are easily imported into the project.

#### Weaknesses:

1. If the amount of data stored is scaled up (over 1000 customers or products) a list is not an efficient structure to sort or search.

2. Converting to a binary tree structure would involve adding significant code to the project. Recommendation: a binary tree structure could be implemented and would allow the application to scale up well (to handle large amounts of data within each database table).

### Reliability:

#### Strengths:

1.The generic list collection (List<T>) is a Java library.

2.It has been thoroughly tested and found to work, error free by Oracle and Java engineers over a period of several years.

3.Java is very reliable at handling its core library dependencies (when they’re imported).

#### Weaknesses:

1.The database tables are being accessed via the internet by the client’s methods. Network service interuptions could create problems in processing data.

2.With large amounts of data, the speed of the network could become a factor.

The above could be mitigated by making it clear to the customer that the application will function most reliably and efficiently when deployed either on a high-speed intranet or using a broadband connection (where the client app is distributed outside an intranet).

### Maintainability:

#### Strengths:

1.The code in the application is modular and well commented. Each CRUD function is defined in its own method. Using List<T> has helped in keeping methods small, and clear to read and understand.

2.As the code has evolved versions have been assigned and older versions have been stored.

3.Change control can be implemented along with a traceability matrix and documented test results. (As re-engineering proceeds and the GUI is developed)

4.The developer already has the code needed to convert from object lists to binary trees.

5.Using List as a source of data for the CRUD methods has made the code very concise and easily understood.

6. Methods and field names are given names which match the database table fields and describe the functions being executed semantically (like find, findAll, edit, update, remove…).

#### Weaknesses:

1.Formal change control is not yet implemented.

2.A traceability matrix has not yet been implemented.

3.There has been no user based testing completed or documented as yet.

Change control, formal version control and a traceability matrix need to be completed and maintained.

Testing needs to be documented, scripted, and conducted with actual intended users (once the app is re-engineered and a GUI has been developed). This is important as the amount of data stored for customers and products becomes greater, and tests the efficiency of List<T> objects and methods.

## JFrame

### Efficiency:

#### Strengths:

1.Well suited to developing form style GUIs in NetBeans quickly (using the palette).

2.Has all the form elements needed for the re-engineering project here (no CSS to mess with).

3.Provides a look and feel that mimics the user’s operting system element features. Familiarity helps in making an application intuitively easy to use, and quickly adopted for intended tasks.

4.Efficient use of RAM and disk space for the amount of graphics produced.

5.Form elements are instantiated at runtime and so can be semantically named to fit the application.

6.Relatively easy to set up validation for user input fields within form element handlers.

#### Weaknesses:

1.Changing the look and feel is quite involved, if you choose to do so.

2.Animations and other interactive effects are difficult to create (much easier in CSS).

3.Layout for this project will be easy, but layouts can be very time consuming and involve multiple containers (and the use of multiple layout types).

Correcting the above could be simplified by making it easier to incorporate CSS styling within Java graphic elements. I do believe it’s possible, but have not yet learned how to do so. Layouts are difficult in any language, so I’m not sure there is any better solution than what the JFrame API currently offers. Being able to nest different layout types within different nested containers offers a lot of the same advantages currently being achieved through the use of CSS’ Flexbox API.

### Reliability:

#### Strengths:

1.Being from the Java Swing package, JFrame and its associated elements are native to Java. As such they have undergone extensive use and testing, both by Oracle and by Java’s many developers: over a period of many years.

2.It is relatively easy to incorporate input and output elements into event handlers. This makes coding efficient and usability/functionality reliable.

3.JFrame GUIs can easily be made with flexible dimensions. This is a big plus when developing apps that may be used across many display sizes.

4.Like Java generally, apps made using JFrame GUIs are cross platform complatible. Users who may use the same app on Windows, OS X or Linux can rely on the same functionality across all those platforms.

#### Weaknesses:

1.I haven’t really experienced any reliability issues with JFrames and GUIs. Once coded, they instantiate with very reliable consistency.

2.The issues I have had relate more to the application of look and feel properties as handled by the UIManager, but that’s a separate API.

### Maintainability

#### Strengths:

1.If GUI elements are named appropriately at the outset, changing variable names, updating to add or remove elements, or changing parameters (like gaps and field lengths) is relatively easy.

2.JFrames have several layout types available and by nesting them it’s possible to arrive at fairly intricate layout designs, when needed.

3.Because every element has its own sizing and positioning metrics, a GUI can be updated or changed as requirements change and design changes are called for.

4.A JFrame is the root container for a GUI. Accordingly, it is possible to nest many different containers within the root window. This makes it possible to return to a project and add tabbed pages, and other features as an application evolves or expands over time. This allows for an application that can grow in functionality while still retaining familiarity for its original users.

#### Weaknesses:

1.It can be difficult or time consuming to make general changes to look and feel. For instance, if a decision is made to change the size of input fields, each input field in a form needs to be changed individually.

2.While there is a mechanism for overriding the UIManagers applied look and field (like Nimbus for instance) I have found implementing these types of changes to be somewhat inconsistent. By that I mean some changes work, and others are ignored.

3.There are not enough look and feel packages available to provide the kind of variety available through the use of something like CSS and JavaScript.

4.Packages like Nimbus and Metal contain elements and styles that match up with all the elements available within a JFrame GUI (which is good) but, they use element names that don’t match the terminology used in the JFrame API which makes it very difficult to match UIManager changes to specific JFrame elements. In general, I’ve found the laf APIs are poorly documented which makes them difficult to apply. Things like padding and margins have different names depending on whether you look in the JFrame API or the Nimbus or SeaGlass laf APIs. Elements themselves have completely different names, so unless you have an image to go by, you can’t be sure what color or dimension is being applied to what JFrame feature by the UIManage.

In CSS code can be written such that changes can be made with one rule that affects all input fields on a form, or all of any particular element type, or all elements within a particular container. It would be nice if the same kind of generalizations could be made with JFrames.

The problems between look and feel available packages and the nomenclature applied to JFrame elements in its Java API could be remedied with better naming standards, and more complete documentation of the look and feel packages. However; that is admittedly more the fault of the look and feel package developers than a problem with the JFrame API.

The API for implementing the UIManager (which is a subset of the JFrame API) needs to be clearer as to how elements and their parameters are stored within look and feel packages. It is not well documented as to what type of collection is involved and how to access it. It’s also not clear whether changes applied using the UIManager are changing the contents of a look and feel bundle or just overriding default values at runtime.

# 4. Configuration and Change Management

It was interesting to see that during NASA’s SMAP launch Saturday morning they were mentioning the addition of notes to a change log during the countdown.

## Overview: the Change Management Process

The Information Technology Infrastructure Library, now known as the ITIL Foundation, org., provides a framework for IT processes and management. The foundation also serves as the administrator of a series of IT certification courses and exams. The framework provides a structure for the management of system software and change control. This framework is described as a configuration management database ( CMDB ). ITIL works hand in hand with CMMI (the Capability Maturity Model Integration) which measures how far up a 5 step rating scale a particular company has progressed in terms of managing its software production processes (Carnegie Mellon, 2014).

How changes to software systems (requests for change: RFCs) are handled is one of the key elements of both the ITIL framework and the CMMI ratings review. ITIL recommends a process where changes, as requested, are first reviewed by a management board or committee (Change Control Board). The purpose of the board is to review the change request and determine the severity of the need for change. The board decides what aspects of the overall system will be impacted, whether the change is urgently needed (such as repairing/correcting a system malfunction) or something wanted, but not urgent (like the addition of new features). The board also needs to determine who will be impacted and notifies any involved parties as necessary.

## Project Change Management Diagram

This diagram shows the flow of steps involved in implementing changes to the current application. “GH” indicates GitHub.

For this project the proposed changes will be considered “normal” (Tilden, 2015-slide10) and an appropriate process will be put in place. The changes are planned and will be implemented by the original system author. The changes relate to the look and feel of input and output, not the data itself. Changes will result in a new client application, which will require unit and Q&A testing to confirm functionality according to specific requirements. The specific requirements, which stem from the requests for change, are listed in a traceability matrix below this overview. Test cases and test scripts will be written and used to run tests, and record results for the new client. Test results will be add to GitHub documentation. The project manager should notify the CCB as each phase is completed. GitHub will act as the CMDB: it can provide version control and a diff log. Sample test cases, scripts and traceability matrix are shown below.

Given the amount of time remaining in the course, it is unlikely that GitHub functionality will be implemented. However, it should be possible to implement the RFCs listed in the traceability matrix and test the functionality of the GUI client application.

## Traceability Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Description | Test Script ID | Test Case ID |
| RFC 1 | The client application shall provide a user GUI | TS-GUI-01 | TC-GUI-01 |
| RFC 1.1 | The GUI shall allow the user to view all customers | TS-ViewAllC | TC- ViewAllC |
| RFC 1.2 | The GUI shall allow the user to view all products | TS-ViewAllP | TC- ViewAllP |
| RFC 1.3 | The GUI shall allow the user to view by customer ID | TS-Find1C | TC- Find1C |
| RFC 1.4 | The GUI shall allow the user to view by product ID | TS-Find1P | TC- Find1P |
| RFC 1.5 | The GUI shall allow the user to view a range of customers | TS-ViewRangeC | TC- ViewRangeC |
| RFC 1.6 | The GUI shall allow the user to view a range of products | TS-ViewRangeP | TC- ViewRangeP |
| RFC 1.7 | The GUI shall allow the user to update customer data (for individual customers) | TS-UpdateC | TC- UpdateC |
| RFC 1.8 | The GUI shall allow the user to update product data (for individual products) | TS-UpdateP | TC- UpdateP |
| RFC 1.9 | The GUI shall allow the user to add new customers to the database (create) | TS-CreateC | TC- CreateC |
| RFC 1.10 | The GUI shall allow the user to add new products to the database (create) | TS-CreateP | TC- CreateP |
| RFC 1.11 | The GUI shall allow the user to delete a customer from the database | TS-DeleteC | TC-DeleteC |
| RFC 1.12 | The GUI shall allow the user to delete a product from the database | TS-DeleteP | TC-DeleteP |
| RFC 2.1 | The GUI shall provide a button for selecting customer type objects | TS-SelectC | TC-SelectC |
| RFC 2.2 | The GUI shall provide a button for selecting product type objects | TS-SelectP | TC-SelectP |
| RFC 3 | The GUI shall provide an exit button to close the app and terminate the connection to the database. | TS-ExitApp | TC-ExitApp |
| RFC 4 | Layout modified to increase title font size, even up layout of textfields, labels and buttons. | TS-EditLAF | TC-EditLAF |
|  |  |  |  |

## Sample Test Script

|  |  |  |  |
| --- | --- | --- | --- |
| **JAX-WS CRUD Client GUI** | | | |
| **Test Script Information** | **Test Script Details** | | |
| **Script ID** | TS-GUI-01 | | |
| **Description** | Instantiate a JFrame Form GUI | | |
| **System or subsystem** | interactive user form | Version | v0.0.2 |
| **Test Type** | Functional | Date Of Test | 02/05/2015 |
| **Purpose Of Test** | To ensure that the GUI is instantiated with all needed fields and buttons | | |
| **Test Environment** | Glassfish Server hosting the JAX-WS CRUD application & SQL Server hosting a MySQL database. GUI access to web service via localhost | | |
|  |  | | |
| **Automated Test Steps** | Test script should perform the following:  start Glassfish server  start SQL database | | |
|  |  | run web service application | |
|  |  | run GUI client application | |
|  |  |  | |
| **Expected Result(s)** | The JFrame Form GUI should appear with labeled data entry fields and necessary buttons. | | |
| **Actual Result** | The GUI opens as expected | | |
| **Pass/Fail** | Pass | | |

## Sample Test Case

|  |  |  |  |
| --- | --- | --- | --- |
| **JAX-WS CRUD Client GUI** | | | |
| **Test Script Information** | **Test Script Details** | | |
| **Script ID** | TC-GUI-01 | | |
| **Description** | Test server startup and GUI instantiation | | |
| **System or subsystem** | Client loading | Version | v0.0.2 |
| **Test Type** | Functional | Date Of Test | 02/05/2015 |
| **Purpose Of Test** | To ensure that the user is able to initiate the web service and GUI | | |
| **Test Environment** | Glassfish Server hosting an online database application | | |
|  | MySQL Server with MySQL database | | |
| **Manual Test Steps** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| In NetBeans the tester should: |  |  |  |
| 1) Start the Glassfish server under the services tab | The server should start and display its startup in the NetBeans console | Server starts and startup steps are displayed in the console | Pass |
| 2) Start the MySQL server | Database server should start and display its running state with a green arrow on its icon | Database started and green arrow is displayed on server icon | Pass |
| 3) Run the WS app & Client application in NetBeans | The GUI should open in the user’s monitor | The GUI opened in the user’s monitor | Pass |

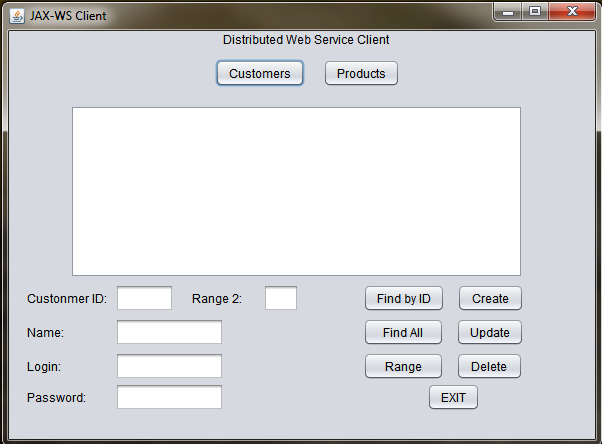
# 5. Software Construction Tools and Techniques

For this project a number of tools were used within the NetBeans IDE. A JAX-WS web service application was configured to access a MySQL database and provide SOAP based messaging for performing CRUD services using a remote client application. Connection between the web service and database is effected through a JNDI socket connection and via a websocket connection (to the client): provided by the service application. The connection is instantiated by the client application, as needed to perform CRUD functions with elements in the database.

The application is generated through the configuration of a Maven framework accessed through NetBeans. The Client functionality is provided by the WSDL files generated by the framework working in concert with CRUD methods hand coded as POJOs. For this project, the hand coded methods are used within the field and button handlers of a JFrame Form GUI. They provide a distributable interface for client users to access the database of customers and products. The intention is that the user should have a highly intuitive access to a database for creating, reviewing, editing, and deleting customer and product data, as needed. This relieves the user of needing to have any knowledge of the inner workings of a MySQL database, the functional programming needed to perform CRUD actions (and JDBC), or the internet connectivity programming needed to create a remote client.

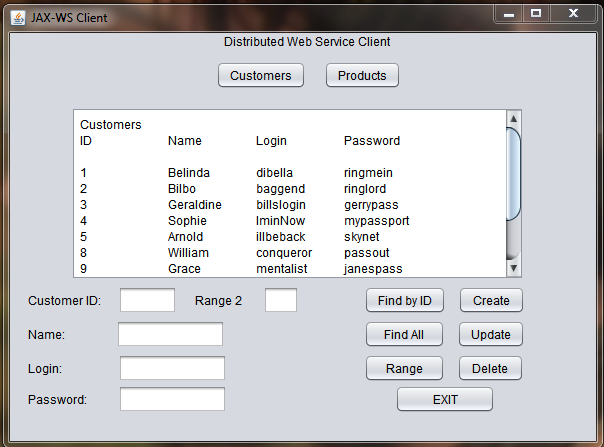
The GUI contains two buttons immediately under the title. The user presses one to choose whether they will be acting on a customer or product table in the database. Five labeled text fields are provided for entering customer or product data. The labels change according to which top button is initially pressed. Six buttons are provided for executing the various CRUD functions. Reading tables may be done by choosing one of three buttons. “Find by ID” will search and display a table object by ID number. “Find All” displays all the customer or product objects in their respective table. “Range” allows the user to input two IDs and display all table items within the entered range of IDs. Items may be added to the database using the four input fields and the “Create” button. Any item in either database table may be edited by entering its ID and any other field to be edited, then pressing the “Update” button. Items may be deleted by entering their ID number and pressing “Delete”. Screenshots of these actions are shown below.

## Screenshots:

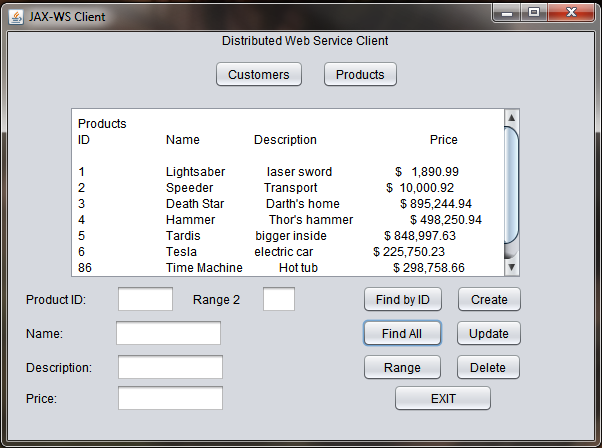


Prototype

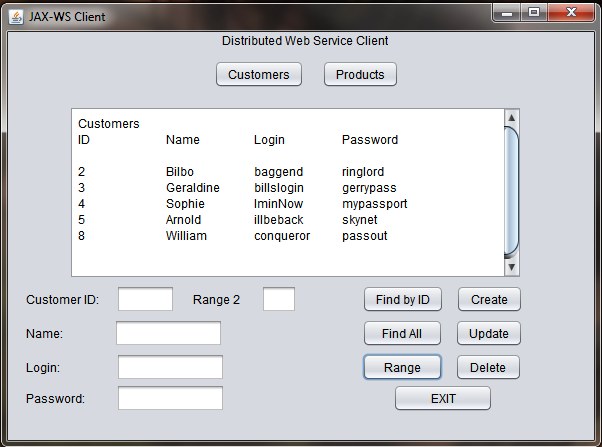
### *Display all customers*



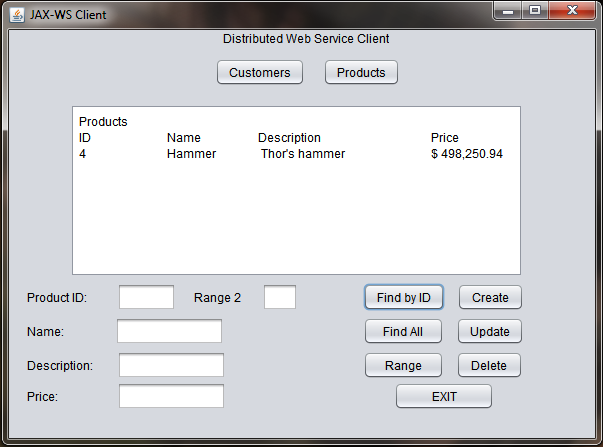
### *Display all Products*



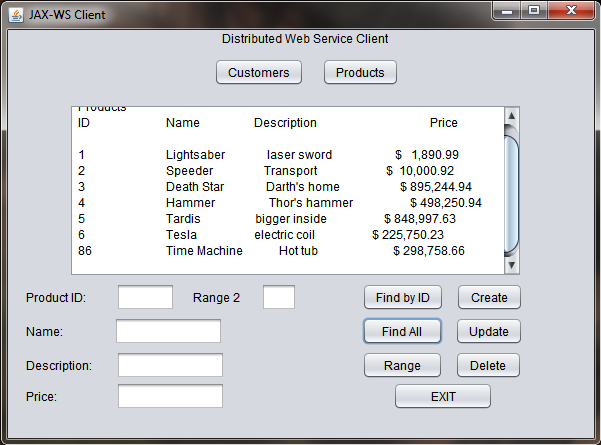
### *Find by ID Range*



### *Find by ID*

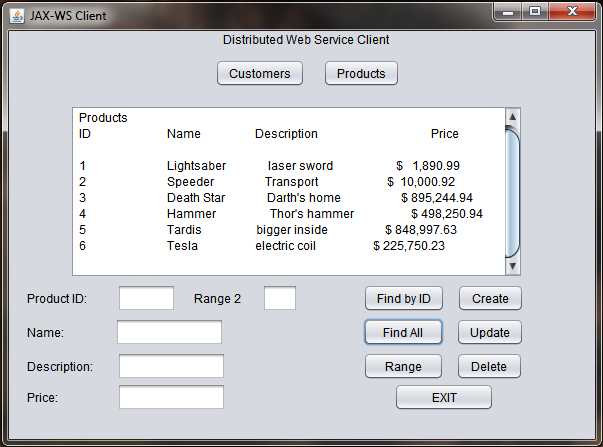


### *Edit Database Entry*



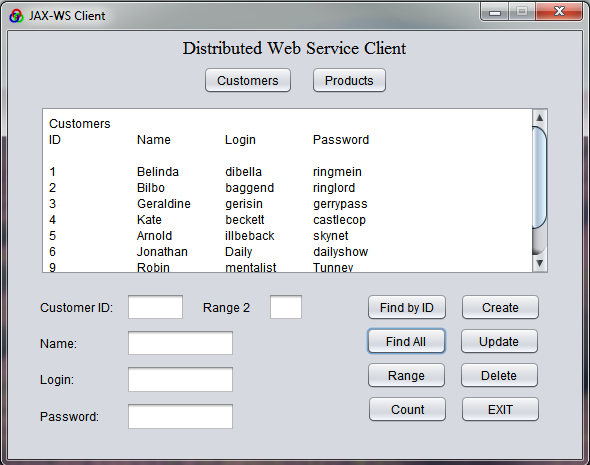
Here, item 6 was edited from “electric car” to “electric coil”.

### *Delete*



Item 86 was removed from the product table.

### *Layout Update*

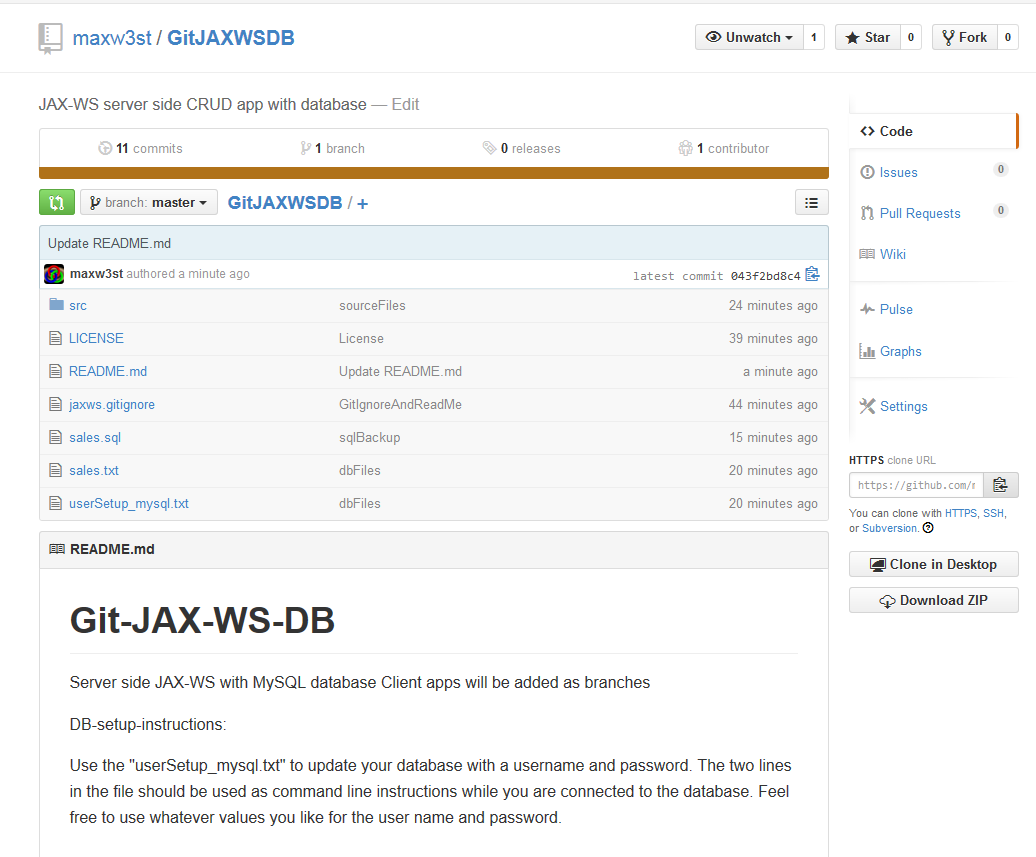


Update to add custom icon to frame, change title font size and even up layout of textfield labels, fields, and buttons.

All RFCs from the traceability matrix have been completed and the table has been updated to reflect this LaF update.

## Version Control and Change Management

A repository has been created at GitHub to store a backup copy of the application. It also provides change control as every change to the program is documented as changes are committed and the local version of the app is synced with it. This also provides a form of version control as changes can be rolled back, and major changes can be made using new branches off the original. The first version (v0.0.1) contained a completed server side application with MySQL database and client app. for testing the connection to the web service and database. The next version (v0.0.2) contained a client with the twelve methods necessary for implementing full CRUD functionality. Interaction was console based and included a menu for selecting user functions. This version extends the console app and provides the same functionality through the creation of the distributable GUI shown in the screenshots above (v0.0.3).



# References

ChillingEffects.(2014).Reverse Engineering-FAQs.Berkman Center for Internet & Society, Harvard University: Cambridge, MA <https://www.chillingeffects.org/topics/15>

Carnegie Mellon Inst.(2014).CMMI Institute:Pittsburgh, PA

[http://cmmiinstitute.com/#home](http://cmmiinstitute.com/%23home)

ITIL.(2008).Information Technology Infrastructure Library.ITIL Foundation, org: web

<http://itilfoundation.org/index.html>

Oracle.(1993, 2014).Java API: Class JFrame.Oracle: Redwood City, CA

<http://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>

Oracle.(2014, 1993).Java API: Interface List<E>.Oracle: Redwood City, CA

<http://docs.oracle.com/javase/7/docs/api/java/util/List.html>

Oracle.(2011).Java API: Service.html.Oracle: Redwood City, CA

<https://jax-ws.java.net/nonav/jaxws-api/2.2/javax/xml/ws/Service.html>

Oracle.(2014).The Java Tutorials, Lesson: JDBC Basics.Oracle: Redwood City, CA

<http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

Oracle.(2014, 1996).The Java Tutorials, Lesson: Modifying the Look and Feel.Oracle: Redwood City, CA <http://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/index.html>

Tildon, C.(2015).Change Management – change categories (slide 10).CTU:Colorado Springs, Co

Change Mangagement pdf

Poz, F.(2013).Intro to Reverse Engineering Software Day 1 P1: What is reverse engineering and what can it tell me.OpenSecurityTraining.info (web) <https://www.youtube.com/watch?v=cNbrU10-WB4>

Rouse, M.(2014).Reverse Engineering.TechTarget:Newton, MA

<http://searchsoftwarequality.techtarget.com/definition/reverse-engineering>

Sundsted, T.(2000).JNDI overview, Part 1: An introduction to naming services.JavaWorld.com:

<http://www.javaworld.com/article/2076888/core-java/jndi-overview--part-1--an-introduction-to-naming-services.html>

Note: This document contains charts taken from week 5 of IT351, and an outline taken from the project proposal letter. The document also contains templates for a traceability matrix, test case and test script taken from CS455.