# Colorado Technical University

IT252-1404B-01 Advanced Java Programming

Phase 5 — IP

# Stuff–n–Such, Ltd.

Jack Simmonds

Professor: Anthony Lowe

12/2014

app. version 0.0.5

Contents

[Colorado Technical University 1](#_Toc407059743)

[Stuff–n–Such, Ltd. 1](#_Toc407059744)

[Project Overview 5](#_Toc407059745)

[Phase 1 — Client–Server Applications 6](#_Toc407059746)

[Pseudocode: 6](#_Toc407059747)

[Server (Lowe, 2014) 6](#_Toc407059748)

[Customer 6](#_Toc407059749)

[Product 6](#_Toc407059750)

[RequestCustomers 7](#_Toc407059751)

[RequestProducts 7](#_Toc407059752)

[GUI Client 7](#_Toc407059753)

[Class Diagrams 8](#_Toc407059754)

[Screenshots 13](#_Toc407059755)

[Phase 2 — Multi–threaded Server Applications 14](#_Toc407059756)

[Discussion 14](#_Toc407059757)

[Pseudocode 15](#_Toc407059758)

[Incoming 15](#_Toc407059759)

[Changes to class JacksServer 15](#_Toc407059760)

[Screenshot 15](#_Toc407059761)

[Phase 3 — Database–driven Desktop Applications 16](#_Toc407059762)

[Discussion 16](#_Toc407059763)

[Screenshots 18](#_Toc407059764)

[Phase 4 — Database–driven Web Applications 26](#_Toc407059765)

[Discussion 26](#_Toc407059766)

[SQL Design Spec. for Sales tables 26](#_Toc407059767)

[Architecture 28](#_Toc407059768)

[Component Diagram 29](#_Toc407059769)

[Entity Relationship Diagram 30](#_Toc407059770)

[Screenshots 31](#_Toc407059771)

[Main Application Page 31](#_Toc407059772)

[Read All (“Show All Customers”) 32](#_Toc407059773)

[View Customer (Read 1) 33](#_Toc407059774)

[Create Customer 34](#_Toc407059775)

[Update Customer 36](#_Toc407059776)

[Delete Customer (“Destroy”) 37](#_Toc407059777)

[Read All (“Show All Products”) 38](#_Toc407059778)

[Create Product 39](#_Toc407059779)

[Update Product 41](#_Toc407059780)

[View Product (Read 1) 43](#_Toc407059781)

[Delete 44](#_Toc407059782)

[Users Manual 45](#_Toc407059783)

[Back up and Restore 46](#_Toc407059784)

[Phase 5 — Database–driven Web Services 47](#_Toc407059785)

[Discussion 47](#_Toc407059786)

[Future Development Plan 50](#_Toc407059787)

[Explain how web services are used in the real world. Provide at least 3 real world examples. 51](#_Toc407059788)

[Discuss the Java EE technologies you used and the reasons for your choice. Explain how these technologies achieved data persistence.What are the benefits of web services? 53](#_Toc407059789)

[How would you troubleshoot and test a web service? Would you use a debugger? Why or why not? 56](#_Toc407059790)

[Do you think that your client/server application could be enhanced to consume a web service or be used as a web service? Why or why not? 57](#_Toc407059791)

[Architecture, Screenshots, Testing and Users Manual 59](#_Toc407059792)

[Web Services Applications 59](#_Toc407059793)

[REST 59](#_Toc407059794)

[SOAP 60](#_Toc407059795)

[Web Services Screenshots 61](#_Toc407059796)

[REST Application 61](#_Toc407059797)

[SOAP Applications 65](#_Toc407059798)

[SOAP Client Images 67](#_Toc407059799)

[References 73](#_Toc407059800)

# Project Overview

This project will demonstrate the construction of a GUI based application. The application will use a server/client system to enter, store, display, and evaluate the products and customers of Stuff–n–Such, Ltd. Text will entered, by the user, using a GUI. These values will be passed as arguments through customer or product constructors and stored as objects (ArrayLists). Data will be entered using the client, passed through object constructors and strored by the server ( client passes objects to server ). Objects will be passed as serialized data between client and server ( in both directions ). Client enters data, server stores it. Client queries server for data from customer or product tables. Communication works both ways for both client and server. Files will be created on the server to hold customer and product information: as entered with the client (GUI).

In sequence, when the application is loaded: the server will be started, the client will be started and the GUI (contained in the client) will be instantiated. A user will be able to use the GUI to enter or retrieve customer and product information. As this information is entered, it will be converted into objects (either customer or product ) and stored in ArrayLists (one for each object type). The arrays can be used as a data source, to stream serialized objects to the server, when data entry is finished (a SEND button?). The server will receive data streams from the client and read the incoming objects. For this phase of the project input is not a consideration.

For now, the GUI will display lists of customers or products as chosen by pressing either a Customer or Product button. Output from the server’s object output streamer is stored in a list (by the client), and displayed in a textarea in the GUI. In order to create displays of customers and products in the GUI text areas: the objects stored on the server (in text files) are read in, line by line (when the server is started). As each line is read in it is passed through the appropriate constructor (Customer or Product) and the resulting object is saved to an ArrayList. When the GUI client sends a request to the server (for either customers or products) the appropriate list is iterated over and the server outputs the objects, as a serialized stream, to the client. The objects are then received and displayed, by the client (using the appropriate object toString method) in the GUI textarea.

# Phase 1 — Client–Server Applications

## Pseudocode:

### Server (Lowe, 2014)

1. List types Customer & Product area instantiated as ArrayLists

2. instantiate socket server and pass port number (ServerSocket server, 67)

3. readCustomers method is called

4. readCustomers uses a BufferedReader and FileReader to read in a text file line by line

5. as each line is read in it is split and the tokens placed in an array

6. the tokens are passed as arguments to the Customer constructor and an object is created

7. the object is added to the Customer List “customers”

8. steps 3 — 7 are repeated for the readProducts method

9. a while loop is set to contain the following steps

9. a socket is created and set to listen for an incoming client connection (server.accept() )

10. input and output streams are instantiated ( ObjectInputStream, ObjectOutputStream)

11. an Object field is assigned the request value accepted by the server

12. an if/else block is used to determine whether the incoming request object is a RequestCustomer type or a RequestProduct type

13. the ArrayList is output to the client according to the type of object requested

14. the server and socket are left open

### Customer

1. private fields: customer ID, first name, last name, email address

2. class constructor to instantiate customer objects

3. get and set methods for all fields

4. toString method for formatted output of objects

### Product

1. private fields: product ID, product name, description, price

2. class constructor to instantiate product objects

3. get and set methods for all fields

4. toString method for formatted output of objects

### RequestCustomers

1. constructor implements serializable

2. the constructor contains fields which may be used later for querying or sorting

### RequestProducts

1. constructor implements serializable

2. the constructor contains fields which may be used later for querying or sorting

### GUI Client

1. use the NetBeans palette to create a GUI with a textarea and 2 buttons

2. add labeled textfields ( 8 ) to the GUI for user input of customer and product info

// for IP 1 no input from the user is needed. info will be read from file, so the textfields

// will be added at a later date

3. the JButtons (2) are labeled Get Customers and Get Products.

4. a socket is instantiated and a connection made to the server (URL, port #)

5. each button has a handler to retrieve customer and product information from the server and store the incoming objects in a List. Communication of list objects is serialized between the server and client.

6. for display, each List of objects (for the two button handlers) is iterated over and the objects are appended to the textarea (using the object’s formatting toString method)

7. the socket is closed after the list selected (by button press) has been printed to the textarea.

## Class Diagrams

|  |
| --- |
| *JacksServer* |
| -customers: ArrayList<Customer>  -products: ArrayList<Product>  +socket: Socket |
| +«ServerSocket» server: 67  +«ObjectInputStream» input  +«ObjectOutputStream» output  +«constructor» custRead: BufferedReader  +«constructor» prodRead: BufferedReader  +«constructor» anon.: FileReader // 1 each for custRead & prodRead  +main()  -readCustomers(): List<Customer>  -readProducts(): List<Product> |
| the class accesses two text files: customers.txt and products.txt using the private methods readCustomers and readProducts. |

|  |
| --- |
| *GUIClient* |
| -customers: List<Customer>  -products: List<Product>  -customerOutput: JTextArea  -showCustomers: JButton  -showProducts: JButton |
| -«constructor»InetSocketAddress( “localhost”, int portNo ) // URL, port#  -«constructor»socket: Socket()  -«constructor»output: ObjectOutputStream()  -«constructor»input: ObjectInputStream()  -showCustomersActionPerformed() // button handler  -showProductsActionPerformed() // button handler |
| handlers are used to complete processing according to which button on the GUI is pressed (Customers or Products). |

|  |
| --- |
| *Customer* |
| -String idNo  -String firstName  -String lastName  -String email |
| +«interface»Serializable  +«constructor»Customer( idNo, firstName, lastName, email )  +setters for all fields  +getters for all fields  +toString() : ( formatted string version of object ) |
| customer constructor will be used to create objects for transmitting from client to server (and vice versa). These objects may be displayed in the GUI text area, written to file, read from file, read from an array list, or used to populate an array list |

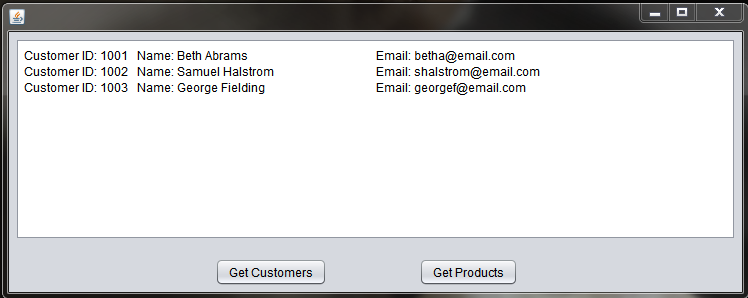
|  |
| --- |
| *Product* |
| -String pID  -String name  -String description  -String strPrice  -double price |
| +«interface»Serializable  +«constructor»Product( pID, name, description, price )  +setters for all fields  +getters for all fields  +toString() : String // formatted string version of object |
| product constructor will be used to create objects for transmitting from client to server (and vice versa). These objects may be displayed in the GUI text area, written to file, read from file, read from an array list, or used to populate an array list |

|  |
| --- |
| RequestCustomers |
| -byID: String  -byFirstName: String  -byLastName: String  -byEmail: String |
| «interface» Serializable  «constructor» RequestCustomers |

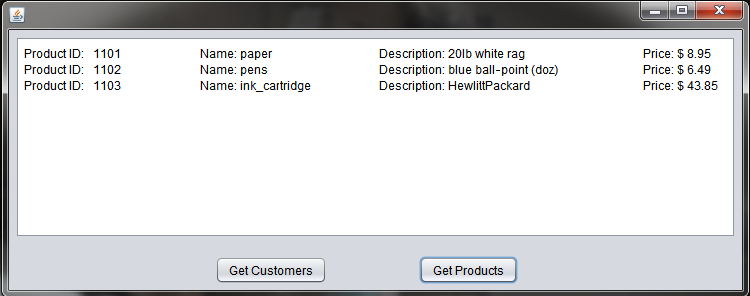
|  |
| --- |
| RequestProducts |
| -byID: String  -byName: String  -byDescription: String  -byPrice: String |
| «interface» Serializable  «constructor» RequestProducts |

|  |
| --- |
| Incoming |
| -objectIn: Object |
| +«interface» Runnable  +«constructor» Incoming  +run() |

## Screenshots



Customers: the data is stored in files, for both customers and products, as comma separated text strings. The output format displayed here is generated using formatted strings in each object type’s toString method.



Products

# Phase 2 — Multi–threaded Server Applications

## Discussion

Class Thread (Oracle, 2014) defines an execution thread (a path currently in use, or available to be used) when an application is running. According to the Java API, an application may have multiple threads running simultaneously. Put another way, threads allow an application to engage in parallel processing. This becomes necessary in client/server applications where a given server may have multiple clients connecting and using an application at the same time.

This phase of the application includes threading to allow the instantiation of a threads for processing client requests for customer or product display in the GUI textarea. When one of the GUI buttons is pressed (Get Customers or Get Products) the server receives a request from the client. That request is saved to a generic Object field. The field is then tested in an if/else block to determine whether it represents an object of type Customer or type Product. Whichever type it is, the object is passed to a runnable class “Incoming” as an argument.

Incoming contains a constructor which assigns the parameter received (as a generic object) to a field (type Object). Again, an if/else block in Incoming’s run method determines whether the object is of type RequestCustomers or type RequestProducts. If the object is type RequestCustomers then the generic object is re-cast as a type RequestCustomers object. The same is done, using RequestProducts, if the generic object is of type RequestProducts.

On the server, and within the if/else block distinguishing object types, an object is constructed of type Incoming. If the if/else test has determined the object is of type RequestCustomers, the Incoming object is assigned to field “in1”. If the request is determined to be of type RequestProducts the object is assigned to field “in2”. In each case, associated thread objects are instantiated passing the appropriate Incoming object as an argument: Thread t1 = new Thread( in1 ) for Customer requests, or Thread t2 = new Thread( in2 ) for Product requests. When the server receives a request from the client, the appropriate thread is started and object type it calls is written back (via serialized output stream) to be displayed in the GUI textarea (after being received as input to the client). The object being referenced in this case, is the ArrayList of either customers or products.

## Pseudocode

This phase includes everything from Phase 1, with the addition of implementing class Incoming (diagram shown in Phase 1, Class Diagrams section).

### Incoming

1. object constructor assigns parameter received to a private class field “objectIn”.

2. method run contains an if/else block which casts objectIn as either a RequestCustomers type request, or a RequestProducts type request.

### Changes to class JacksServer

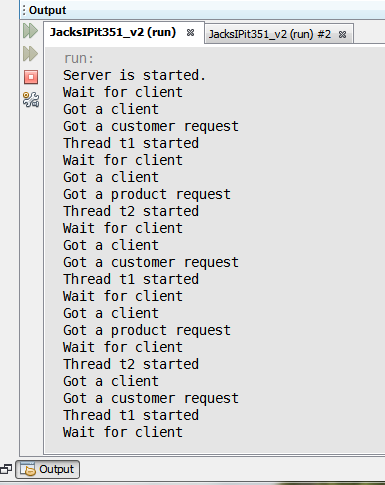
1. within the if/else block (~line 60) thread “t1” is instantiated using the object, created by Incoming’s runnable, as an argument.

2. the priority of the thread is set (this is optional, but may be handy in other situations).

3. the thread is used to call the start method, which runs the run method in Incoming.

4. for testing and demonstation purposes, a line of text is printed to the console showing which thread is in use. This test line is from the Incoming run method and so will only be displayed if the server class properly instantiates the thread.

## Screenshot



# Phase 3 — Database–driven Desktop Applications

## Discussion

select-pg1178, insert pg 1184, update pg 1185, 1186

This application makes use of the previously used Customer and Product classes to create objects for a GUI that allows the user to perform CRUD operations on a MySQL database. The database was given to us by the course professor (as a text file), and contains all the SQL needed to create a database with two tables. The tables are populated with enough data for initial testing of the application.

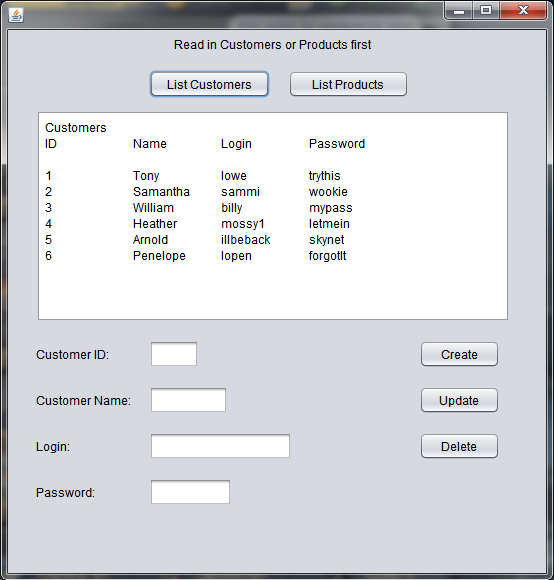
Functionally, the application follows the design pattern set out in Live Chat 6, by professor Lowe (Lowe, 2014). Starting with the JDBC class first used for the week 3 Discussion Board, an interface driven application, with GUI was created. The original JDBC class file is still functional, but is not used when running the GUI and it’s associated class files and interfaces. The MySQL command strings were created from a combination of watching how professor Lowe demonstrated in Live Chat 6, and from the text (Deitel & Deitel, pgs 1178, 1184—1186).

The Customer and Product object constructor files have been somewhat modified to adapt to the fields set in the “sales” MySQL database provided. JDBC files have been created for both Customer and Product objects. These contain the functions set out abstractly in their corresponding Customer and Product data access object interface files. Each interface contains unimplemented methods for each of the CRUD operations. Since the Read function may be applied to a list of objects or a single object, the interfaces contain two versions of the method: read, and readAll. For purposes of this course, only the readAll method is currently implemented by the associated JDBC files. Method readAll is used (by both customer and product classes) to list the contents of the MySQL database tables in the textarea of the GUI.

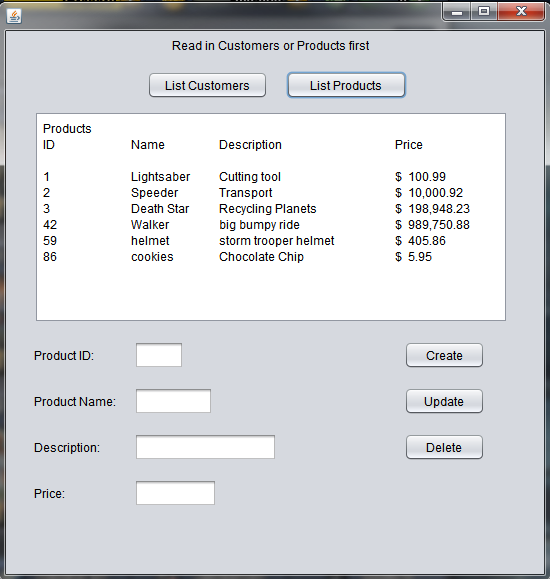
The GUI contains a large textarea, two main buttons for listing either Customers or Products, four labelled texfields for input of user data, and three more buttons for creating new entries to the database tables, updating rows already in the database, or deleting rows from either table. Button handlers in the GUI class are used to call the appropriate methods in the JDBC classes. Those methods override their abstract counterparts in the data access object interface files. The advantage of this architecture is that it adds flexibility to the application. For phase of the course we are using a GUI based application for user interaction, however the DAO/JDBC combination allows that the same functionality could be accessed with a web app, used to read from or print to files, or any other media where CRUD type functionality with a database may be desired (like printing to the console). It would be a matter of replacing the JDBC classes with classes set for the desired functionality, the interfaces ensure that the CRUD operations would be applied with those replacements. The same database functionality can be maintained across different media by using a class that replaces the button handlers of the GUI class, or modifies them as needed.

In order to make use of the Create, Update and Delete buttons, without having to duplicate them in the GUI, switch blocks were used within each of the button handlers for those operations. The Read functionality for Customers and Products is handled by giving each object type its own “List” button. When either list button is pressed a field is set to identify the type of object being accessed by the user. That field is then used to conditionally test for the type of object to be updated, created, or deleted from the database. The main caveate with this approach is that before the user can create, update or delete items in the database tables: they first need to read in either a customer or product table (using the List Customers or List Products buttons). A feature that needs to be added is having the tables automatically reloaded to the textarea after create, update or delete operations are performed. In its current state, the user must re–press the appropriate List button after making changes ( to the database tables) in order to see those changes reflected in ouput.

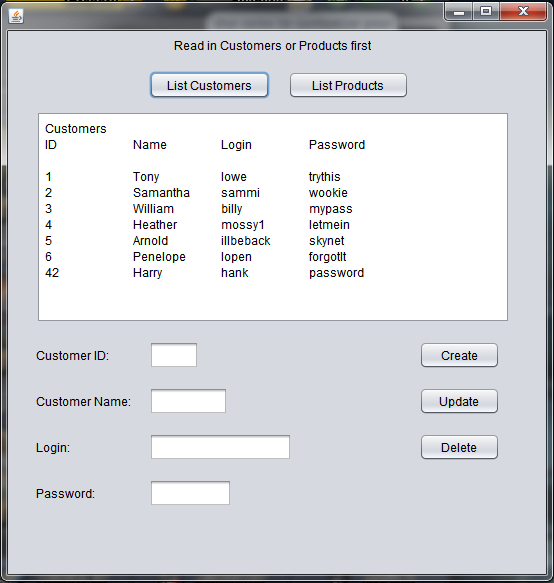
## Screenshots



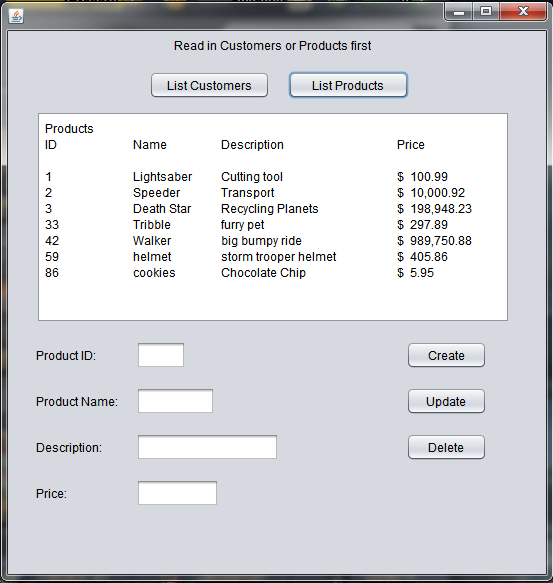
Screenshot of the GUI with texarea displaying the results of pressing “List Customers”. Note that when the List Customers button is pressed, the input field labels are set to match the customer table fields.



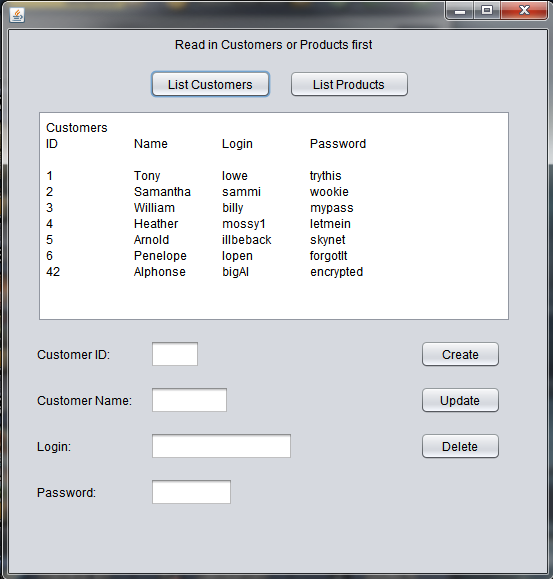
View of the textarea after the List Products button is pressed (shows the product table contents from the MySQL database). Here, the input fields have had their labels changed to reflect the fields in the product table. This is one of the functions of the List button handlers.



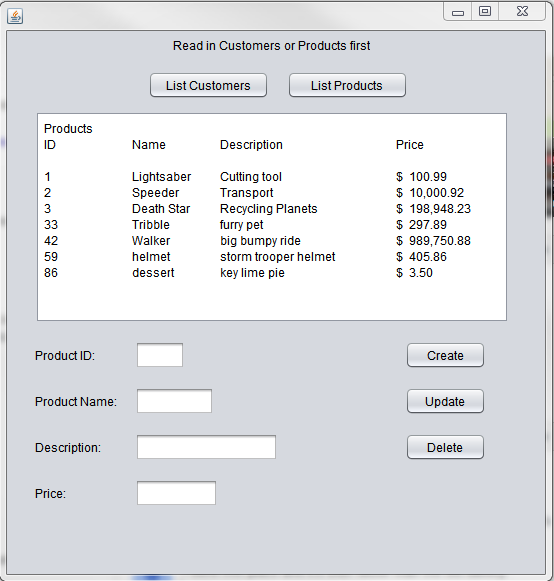
Here a new customer has been added using the text fields and the Create button.



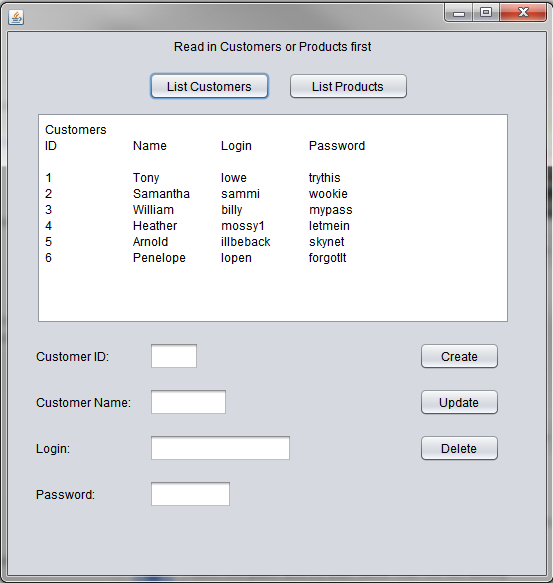
This image demonstrates adding a new product to the database’s product table.



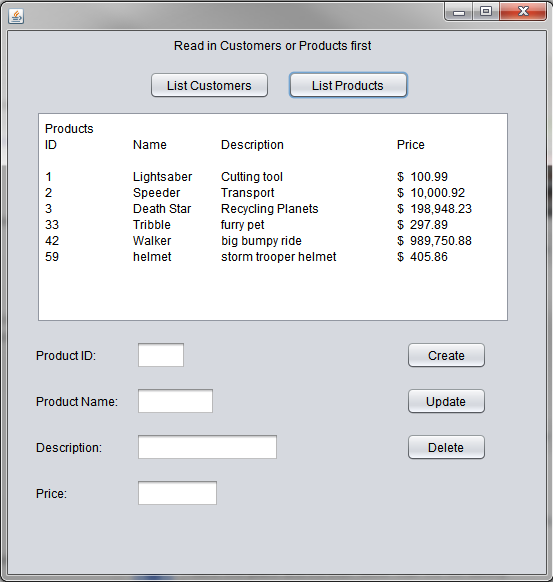
Customer 42 has been updated to reflect a different customer name, login and password. Changes are made to the database tables, and then viewed by re–pressing the appropriate List button.



Product 86 is updated in the database with a new name, description and price.



The customer table, after a row has been deleted (by inserting its ID in the GUI). In this case, customer 42 was deleted.

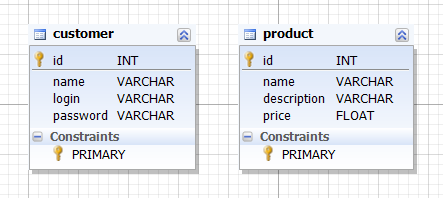


Here, product 86 has been deleted from the database’s product table.

# Phase 4 — Database–driven Web Applications

## Discussion

### SQL Design Spec. for Sales tables



The two tables in the database (SQL shown above) are not relational as of this phase of the project. However, a sales table could be added and the ids for each used as foreign keys. This would allow querying sales by customer and/or product, as well as generating sales reports. (image source: dbForge)

While this web application has many moving parts associated with making it functional, its architecture is not far removed from the application created for Phase 3. In lieu of the GUI for access by the user, the web app. uses a Glassfish server to play the part of web server. The user accesses the CRUD functionality and view of the database through a series of xhtml web pages (9 altogether). The tags in the xml code contain method calls to the JavaServer Faces classes (entity classes), the EJB classes (Enterprise Java Beans), and key/value pairs associated with the Bundle.properties file. The JSF classes contain controller classes which act as intermediaries in accessing the Customer and Product constructor classes (entity constructors) and the objects they construct( EJB classes). The controllers contain the methods for the CRUD operations. The results of these operations are communicated, through the EJB classes, to an entity manager persistence interface which in turn, passes the data values to the database.

The Bundle.properties file contains a list of property values that define the text displayed on each xhtml page (except for MySQL database field values). These property values are accessed through an xml key associated with each value. For exampl “EditProductLabel\_name=Name:”. “Name:” is the label displayed, to the left of the input field, for the name of the product being updated on the Edit Product web page. The value associated with that label is accessed from the xml by the key/value pair “productController.selected.name”. The key/value pair, in this case, is part of an xml input statement where the id is the field “name” within the currently selected Product object in the productController class. Because the xml has it defined as an input type, it is an editable text field.

Rather than having the create, read, update and delete functions located in one class, which implements an interface, the web app uses a hierarchy of files (AbstractFacade, CustomerFacade, ProductFacade). These facade classes act much like the DAO interface in relation to the Customer and Product classes. The CRUD functions for each table type (product and customer) are implemented using controller classes (JSF classes). These controller classes set the values used by the objects handled by the EJB facade and constructor classes. SQL functionality is implemented within the Customer and Product constructor classes: by using sql queries to assign values to the fields of the constructors, and through the implementation of an EntityManager, in the facade classes.

Where the Phase 3 app used a GUI class to create the form seen, and used by the user (through its handler methods), the web app uses Java Server Faces controller files in concert with xml in the xhtml web pages. One each for customers and products. These controller classes act as the bridge between the EJB hierarchy (with its customer and product object constructors & sql scripts) and the xhtml files viewed and interacted with by the web browser’s user. Web pages are created, by the JSF framework, for each object type (customer or product in this case) and for each CRUD function within those two types as well as the main menu page (9 xhtml files the user interacts with). A pagination class provides and manages indexing for each row in the tables and as a bridge between the xhtml pages (with their display,input and output xml) and the EJB classes.

## Architecture

**Presentation** is handled by xhtml files (one for each web page displayed). The xhtml code contans key/value pairs whose keys call for values from a "Bundle.properties class, which contains the text values displayed on the web page.

**Application** functions are initiated through the xml in the web pages (xhtml). As with the presentation, keys in the xhtml are associated with the application, and call methods in the Java classes. Input keys provide user input values to associated application methods, and return values from those methods are either used to update the database, or called (for display) via other xml keys.

The application involves three main components. User access is provided via xhtml web pages running in a browser (the client). Functional programming is contained in a package (which itself contains a number of packages) on a server. The database is operated on and accessed on a MySQL server. Communication between the application server and the database server is effected over a JDBC connection. Messages are passed, using SQL queries, in plain text. Communication between the web client and application server is accomplished via internet connection using the Simple Object Access Protocol (SOAP). These messages, and those between modules within the app., are serialized. Messages are typically objects being passed from one class to another, or values from those objects (which are accessed using set and get methods from the objects’ constructor classes).

**Business Logic** for this application is very straightforward. It involves providing the user with create, read, update and delete functions which are applied to the data in the database. In combination with the presentation functionality: changes to the database are immediately displayed in the client.

**Data access** ishandled through an AbstactFacade class which calls the methods of the EntityManager interface to query the database ("PersistenceContext"). The abstract superclass is associated with two facade subclasses which call the Customer and Product object (entities). The entities (and their properties & values) are used by the persistence methods of the EntityManager in updating the database.

The data path may be mapped as:

client browser↔ xml↔AbstractFacade↔entity facade (Customer or Product) ↔ entity↔entity facade↔AbstractFacade↔EntityManager↔database. The database acts as the “persistence context”. Structurally this would be Client Browser↔App Server↔MySQL Server. Bundle.properties is accessed (via key/value pairs) within the xml in web pages, to provide the text displayed in the client browser.

**Data storage** is handled by a MySQL server which operates a database (“sales”) used by the application. Because data storage is effected through a persistence interface (the EntityManager) other formats could be used (flat files, JSON, XML for example).

### Component Diagram



Application Server

Database Server

Client/Browser







### Entity Relationship Diagram

Entity Manager

Abstract Facade

XML

Client GUI

Tables

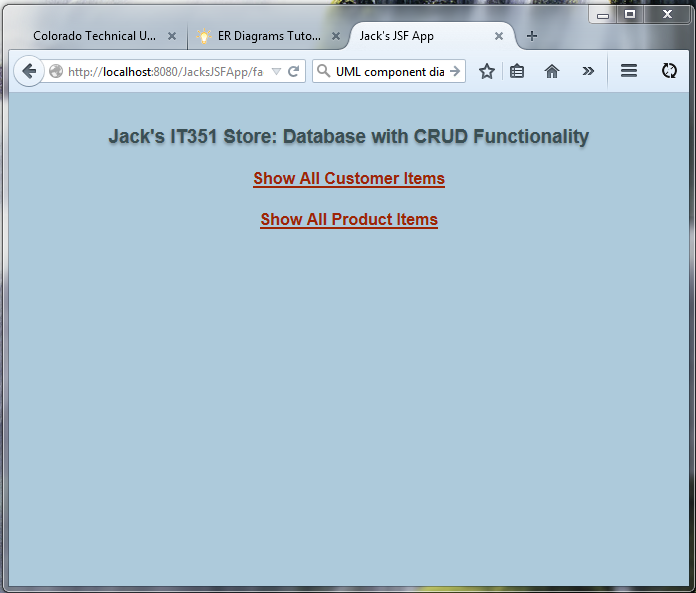
Entity

Entity Facade

Database

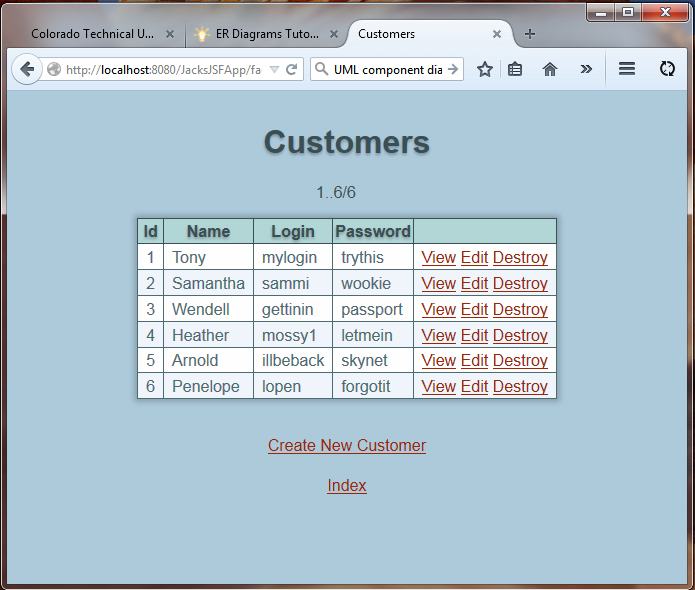
## Screenshots

### Main Application Page



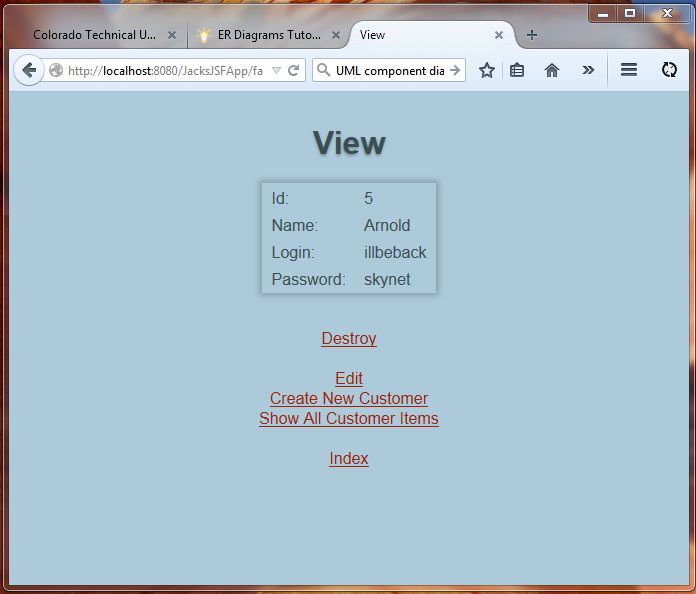
This image shows the opening page of the application (Firefox is being used as a browser). From here, the user can select either Customers or Products and go to a page displaying a table for the type selected

### Read All (“Show All Customers”)

****

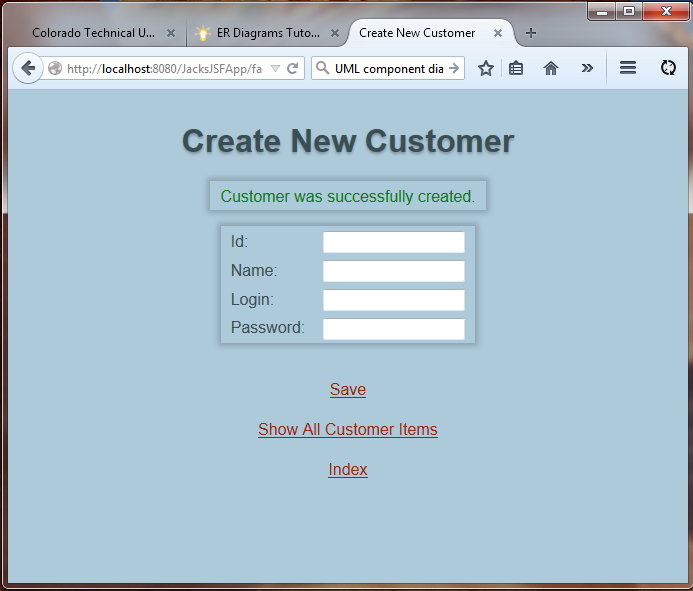
Here, the user has selected “Show All Customers” on the main page, and the table of customers in the database is displayed. Links are provided for entering a new customer, viewing a particular customer, editing the information for any one row, or deleting a customer from the database. All actions above are immediately effected in the database.

### View Customer (Read 1)

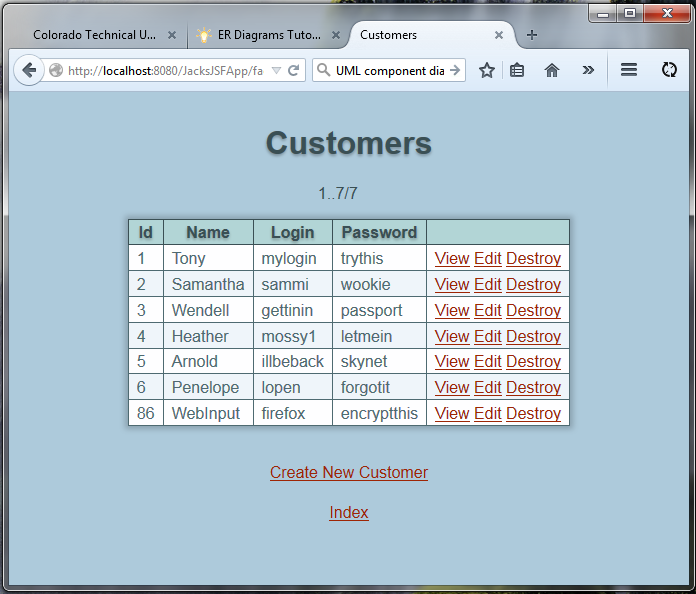
****

If the “View” link is pressed for any customer in the table, this page will open and display the selected user’s information. Links are provided to delete, update, or add a new customer to the database. The Index link returns the user to the main page.

### Create Customer

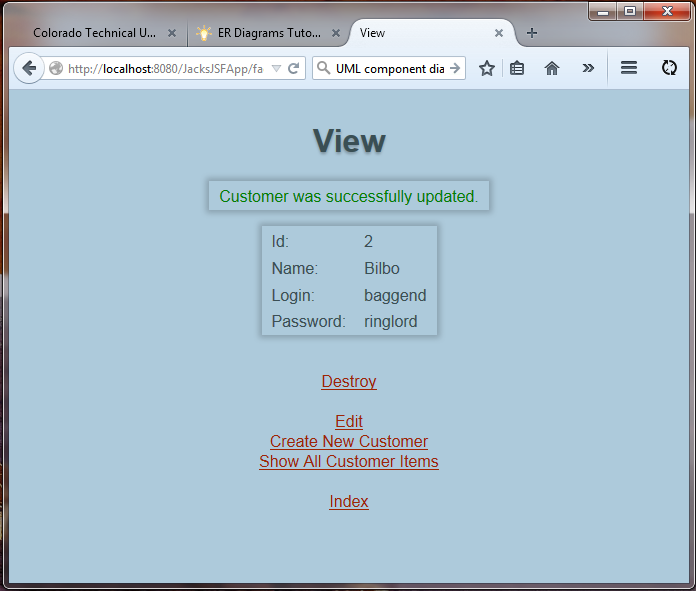
****

From this screen, the user may enter information for a new customer. Once “Save” has been selected, a confirmation message is displayed.

****

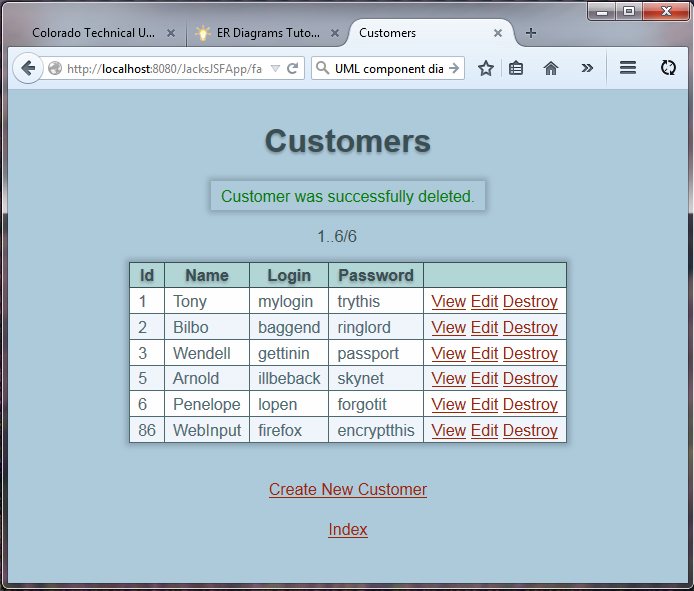
This image demonstrates that the user has successfully added customer number 86 to the database.

### Update Customer



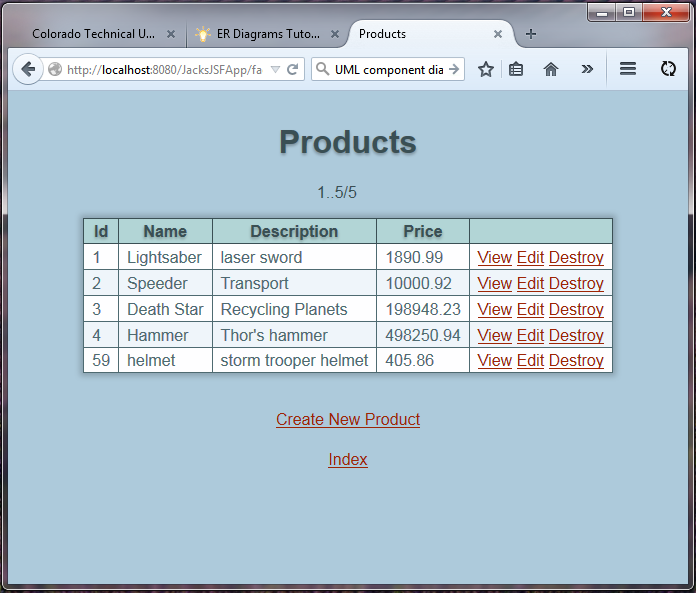
Confirmation that customer 2, previously Sammantha, is now updated to Bilbo. A confirmation message indicates a successful change of information in the database.

### Delete Customer (“Destroy”)

****

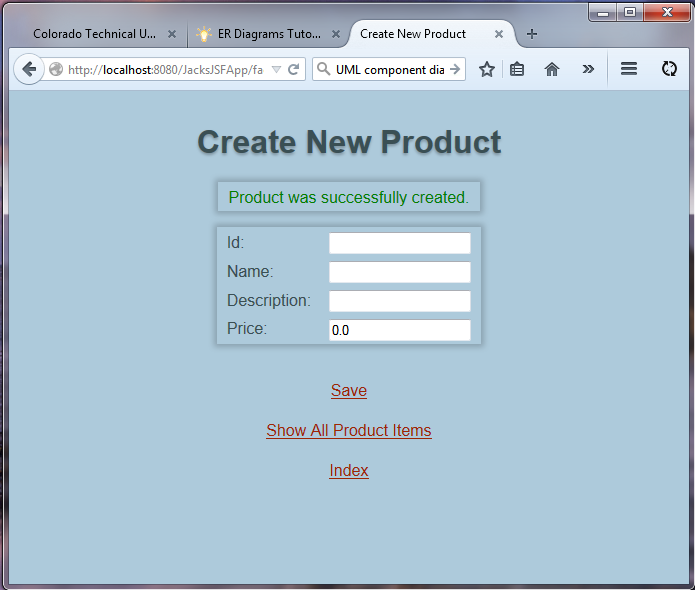
Customer number four has been deleted from the database. The table updates immediately, and a confirmation message is displayed.

### Read All (“Show All Products”)

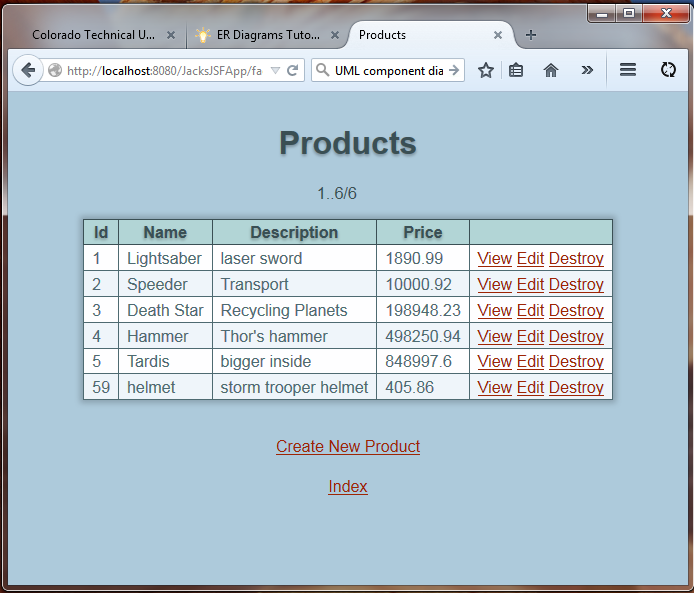
****

Displaying product tables works the same as for the customer tables. Here, the user has selected “Show All Products” on the main page. From here, the user may view individual products, update any or all the information for any product, or delete the product from the database. As with all inner pages, the user always has the option of returning to the main page with the “Index” link.

### Create Product

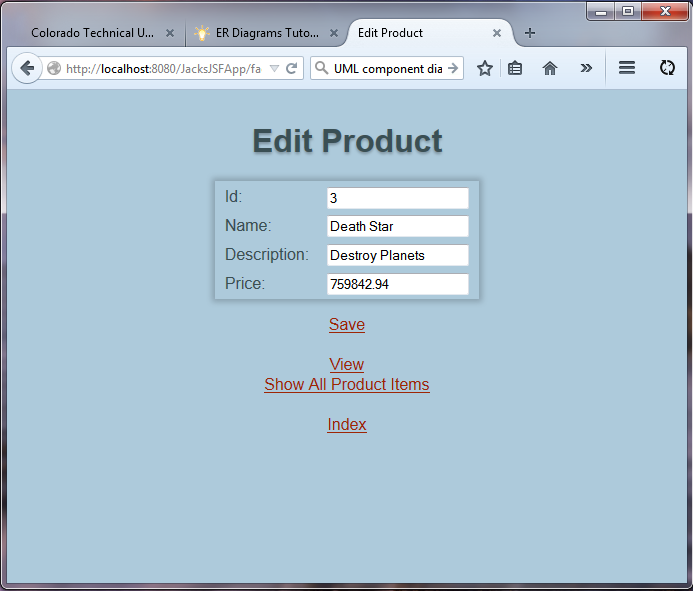
****

Confirmation that a new product has been added to the database.

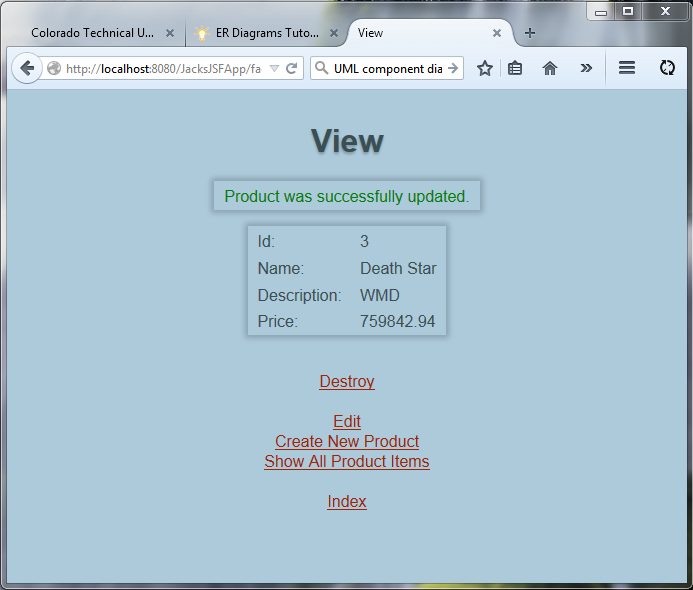
****

The new product is shown in the database (number 5).

### Update Product

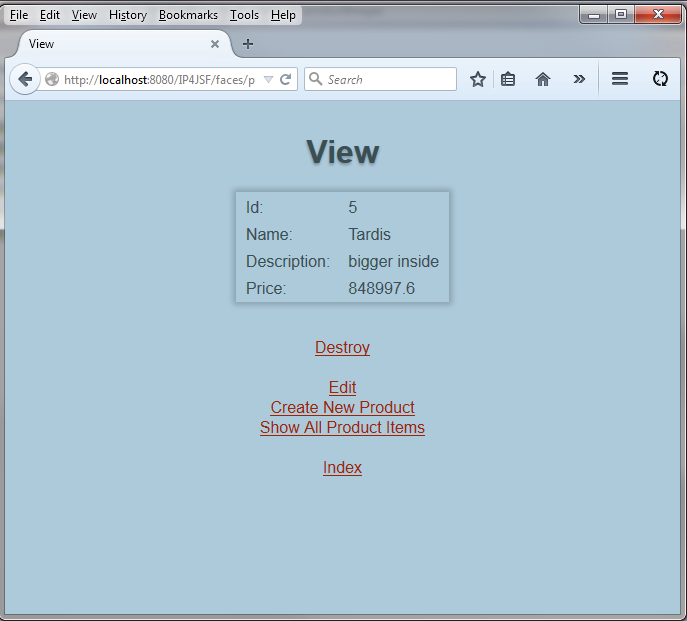


A user is udating the information about product number 3.



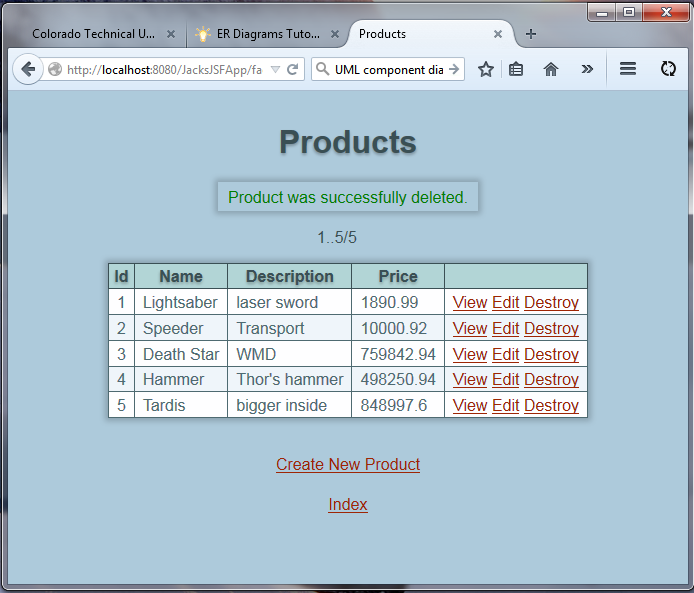
The description has been successfully changed, and a confirmation message is shown. Any, or all of the data for a product (or customer) may be changed when editing a database item.

### View Product (Read 1)



Item five, created earlier, is being viewed. From here the user may choose to perform other operations, or return to the main page.

### Delete



When an item is deleted, “Destroy” link, the database is updated immediately, and the table reflects the change. Item 86 has been deleted. A confirmation message lets the user know what operation was performed. The table reflects the database change immediately.

## Users Manual

This provides a user’s manual that explains how to start and use the application. I MySQL user: ctuonline, password: student, database file: sales.txt-includes SQL script to create database and tables, populated with sample data.

To run the web application:

1. open the project in NetBeans

2. right-click on the IP4JSF App project icon in the projects pane

3. select “run” from the drop-down menu. The application will open in your default browser.

To use the application:

1. click on the link titled "Show All Customer Items"

2. this will display the Customer table in your browser

3. to the right side of the table are links associated with each row of data

4. to view/edit/delete a row of data, click on the View, Edit, or Destroy links

5. View opens a page showing just the data from the selected row.

6. Edit opens a page with the row data showing in textfields which are editable, any one, or all of the textfields may be changed, and then saved (the "Save" link). The table updates immediately.

7. Destroy will delete the selected row of data from the table. The table is updated immediately to reflect the deleted customer (the customer is removed from the database).

8. To create a new Customer, click the Create New Customer link

9. A web page will open showing labeled textfields for entering Id, Name, Login, and Password for the new Customer.

10. enter the desired information for the new customer into the textfields and click the "Save" link. The table updates immediately to reflect the addition of a new customer.

11. The last link on all inner pages is "Index". Clicking the link returns you to the main page. From there you may select either the Customer or Product tables.

12. The Products table works exactly the same as the Customer table. Just follow the steps above to Create, Edit (update), View (read), or Destroy (delete) products.

13. It is assumed that the "user" in this case will have the application running in a browser on their machine.

### Back up and Restore

A backup sql file has been included with the NetBeans project files, if needed. It can be restored to your MySQL server. First: create the database using: mysqladmin create sales

Next, use the command mysql sales < sales.sql to load the database file into the database.

The line below was used to backup the current sales database from the MySQL server:

C:\Program Files\MySQL\MySQL Server 5.6\bin>mysqldump -u jsimmonds –pYourAdminPassword note: the “-“ sign is immediately in front of the “p” switch, and there must be no space between the p and your password

sales > "D:\sales.sql" note: use whatever path points to the sql file

Frod0.mysq1

mysqldump -u jsimmonds –p”AdminPassword” --databases sales > "absolute:\path\to\file\sales.sql" // note: no space between –p and password

the above works. it has to be done with the Windows command prompt in the MySQL bin folder.

To restore, leave out the --databases, and change the command to dump from mysldump. The chevron gets reversed as well. so:

dump -u jsimmonds –p”AdminPassword” sales < "absolute:\path\to\file\sales.sql"

This is a very fussy operation. I had to be executed from the MySQL bin folder, and I had to use my main username and password to effect the backup. This would be because backup/restore permissions were not given to ctuonline / student when the database was setup originally.

However, that login and password will work to perform CRUD operations on the database, once it's loaded into the server. Alternatively, the web app should work with your version of the sales database, without having to "restore" the sales.sql file provided with this assignment’s files.

# Phase 5 — Database–driven Web Services

## Discussion

All of the technologies listed in the requirements for this assignment have been used in the applicactions created. NetBeans is a sort of superframework which provides GUI access to frameworks which in turn use the capabilities of Java EE 7, the JDK, and the MySQL server. JDBC, JSP, EJBs, Servlets, JSF, and XML are all implemented within the JSF application created for week 4. Since the JSF app provides the most user friendly access to the database, it will be used as the primary application for this assignment. In week 5, web services were introduced. They have the advantage of separating the client and server sides of an application structurally which has the advantage of leaving client side development open to using any programming language that can provide interaction with XML. Since XML is a syntax understood by all major operating systems, cross-platform application development is facilitated. A JSF application offers the advantage of having the entire architecture contained in one place. Services are provided through autogenerated web pages, which in turn act as the "client". The web pages can be made available to anyone with an inernet connection and a web browser. If application debugging or upgrades need to be made, they can all be handled by having developers work on the application server system.

The JSF app uses all the above. The Enterprise Beans working with EJB facade classes essentially are servlets for providing interactive database functions (the "Persistence Unit"): between the web pages "clients" the user interacts with, and the database (tutorialspoint). The JSF framework autogenerates Java Server Pages (JSPs) using XML to communicate messages between the users actions, processing provided by the Java application, and the database (using the JDBC API). Most of this is handled by the creation of Enterprise Java Beans (EJBs) which are themselves derived from the primary object constructors of the application (for our purposes the Customers and Products constructor classes). EJBs also provide the persistence management, using the JDBC API to autogenerate the MySQL queries that connect the primary objects to their associated tables in the database.

The web services applications created in week 5 work fine, but user access is currently provided only through test pages generated from within the application. In the case of the SOAP based application, we did build a rudimentary Client application, but have not built web page capabilities into it as yet. Through its use of SOAP messaging, the application provides the most flexibility and security. A series of test methods have been developed for this app. They show that the web services are working fine. However; interaction is currently only available through the console, not web pages. Having developed the methods, a GUI could now be created fairly easily, but there isn't enough time remaining in the course. It is also entirely possible now: to create a JSF style client for the app and provide web based user access.

In weeks 4 and 5 we've built three different types of application. All three are meant for deployment over the world wide web, or via the internet. They all make use of Enterprise Java Beans which introduce the ability to make use of Java application functionality. EJBs provide client services via communication that allows access to java methods and database operations using XML (Java Api for XML Web Services) to pass messages between client/application components. JAX-WS uses the Simple Object Access Protocol (SOAP): which has the advantage of expanding client/server messaging beyond the HTTP limits of POST, GET, PUT and DELETE. SOAP messages are object based and readily serialized. In a JAX-WS architecture this affords the capacity to pass messages which contain any functionality used by an application.

Messages may contain queries (subsequently passed to a database) data objects, or method calls. A Java DataBase Connection driver (JDBC) is used by all three applications for accessing a MySQL database. All three application types (JSF, RESTful, and SOAP based) use EJBs as a bridge between server based Java programming and XML based client interactivity. In the case of the JSF application, all messaging is effected between the application business software and the web pages using xml annotations to tie in with the Java code. The web pages are generated on the server using XMLs key value pairs within semantic tags. The web pages then become the "client" that the user is interacting with. Oracle and NetBeans refer to these web pages as polyfills since they are generated internally by the server based application. The advantage is an all–in–one solution that provides global access (WWW or intranet based) for any user with a browser. Because all messaging is internal to the application, and effected using XML between the Java application methods and XML coded web pages, it is essentially a SOAP style application. In this case, the SOAP messages are all passed within the application itself. Since XML is a language readily understood and used by all operating systems (and browsers), the app. is cross platform compatible by design.

The RESTful app uses EJBs to provide the same internal connection between Java and XML, but the functionality the user sees is through traditional HTTP method calls (POST, GET, PUT, DELETE). The SOAP based application makes use of object based messaging between server and client. It affords REST based functonality only as an option for server based testing of the web services components. A client application could certainly be built around REST style HTTP interactivity, but is certainly not necessary and would diminish the flexibility and security of the client.

IP4's Java Server Faces web app seems best for this week's assignment. Its web client is essentially a polyfill solution contained within a server based application. The application provides the end user with CRUD services for a MySQL database (could be any type of database the app is configured to use). These services are accessed via a set of web pages generated by the application server (Java Server Pages –JSP), but potentially accessible to anyone with an internet connection and web browser. For the course we built the app to run through localhost, but were it deployed from a web facing domain it would be accessible via WWW. The app could also be configured to run company wide over an intranet: in a secure, server/workstation environment.

The SOAP web services app would be another good candidate, but at this stage: lacks the development of a web/browser based client. The test client however; works well and provides a good alternative to HTTP communication over the internet. Serialized and encrypted communication between secured clients and a secured server seem to offer promise in today's high risk environment. As developed within the course, the application has web/browser functionality only in terms of testing from within the application's web services components. This testing functionality generates polyfill web pages (JSPs) which provide CRUD operations for the database associated with the app. Currently, the SOAP tests are working for "findAll" and "count" methods, but generating errors when input is needed from the user (through the web pages). The REST testing also serves up web pages for testing access and provides all CRUD functionality. However, the REST functionality uses HTTP methods, not SOAP. On the plus side, the REST access to the web services affords the ability to use either XML or JSON formats for encoding messages.

## Future Development Plan

The client application designed for use with the SOAP application offers the best potential. It provides a truly separate component from which to access a server based application. As developed here, it has been used for testing the web services through displaying client messages (to the user) in the console. However; the connectivity between client and server/database is established. This means the client could be developed to provide web pages accessible via WWW (JSPs), or use a GUI provided to client users. Both are formats used by many on the internet today. The latter has the advantage of increasing security since only those with the GUI application have access to the web services. As developed, both the application (server side) and client are configured for use over localhost. But, they could be configured for internet use simply by replacing localhost with another domain. Given a public domain, the application would be accessible, by the client, from anywhere the user has intenet connectivity. This architecture could readily be configured for intranet use (within a company or government institution).

I still have much to learn on the topic of threading the application, but creating SOAP message handlers within the web services seems to be the way to go. One of the options, when you right click on a Web Service in the server side application, is to add message handlers. These handlers can be created to asynchronously handle message requests and responses. The SOAP-WS message handler (found on the R click menu of a web service) intercepts messages to provide additional processing (Oracle, JAX-WS). This additional processing could easily be threading. This way all message requests and responses can be asyncronously handled by assigning threads to them before the web services are accessed within the server side application.

I have noticed that the application, as it stands, is already running in a multithreaded environment on Glassfish. I have had as many as 8 instances of the SOAP client running at the same time. I'm not sure what the upper limit is for Glassfish, but additional threading, provided programmatically, may well be redundant.

Hibernate is another topic of immediate interest (tutorialspoint, 2014). Hibernate provides an object–relational mapping (ORM) framework that makes it easier to translate object oriented language style objects into database tables. This helps in creating database queries more efficiently and in a style that creates reusable code. For the course, Enterprise Java Beans have been used, and represent another ORM framework. The queries, in the apps we built in weeks four and five, are generated when the EJBs are created and visible within a number of the application files. They provide the persistence bridge (queries) between the Java objects (Customers and Products) and MySQL database. Spring DAO is another ORM framework which is currently popular (there are many others available).

## Explain how web services are used in the real world. Provide at least 3 real world examples.

Forex and other securities trading have seen large changes over the last 10 to15 years. Forex used to be the province of central banks and the banks they deal with. Now, anyone with a computer, or mobile device can open an account with a dealing desk or bank, and trade currencies (Foreign Exchange - ForEx) 24 hours a day. The same services are made available for any other type of security. Trading is handled, within your account, by using a GUI application connected to a central server system. The centralized servers provide real time securities pricing and chart data, and the user can place real-time trades from the GUI platform. While a lot more complex, the architecture is no different than what we did for week three of the course.(FXCM and TDAmeritrade are two large service providers in this field)

<http://www.dailyfx.com/> & <https://www.tdameritrade.com/home.page>

International Go Server (IGS, AKA PandaNet)

Another use of internet based client/server applications is gaming. The game of Go is a prime example. Players can access other players and share games over the internet using client based GUIs that connect to gaming servers. The International Go Server (IGS aka Pandanet) shows professional tournaments between the world's master players, that you can view and comment on, in real time, from your favorite GUI application. Go is a game that's extremely popular in Asia, and has been around for thousands of years (pre-dates chess). Computer based gaming has allowed that popularity to increase dramatically. It's possible to learn the game from master level players regardless of their (or your) location.

<http://pandanet-igs.com/communities/pandanet>

Formula 1 racing uses a Java client/server application that autoloads a GUI on your device by clicking a link on one of their web pages. The application is not really interactive, but it allows you to monitor data, in real time, coming from all the drivers in a race. If you like high performance racing, it allows for much more detail on what's going on in a race than you can possibly get by watching the same race on TV.

<http://www.formula1.com/>

There are dozens, if not hundreds, of Massively Multi–Player Games: MMPs work using client device based GUIs and controllers (somewhere between a mouse and a keyboard, with specialized buttons) and a game application run on a server system accessible over the internet. Many of these games are designed for use with internet connected devices, like Playstation or Wii, and dedicated to playing games. More recently, these gaming platforms have added the capacity to watch movies or listen to music. A couple to mention would be World of Warcraft, StarTrek Online, Final Fantasy, … (Izawa, E., 2005) .

## Discuss the Java EE technologies you used and the reasons for your choice. Explain how these technologies achieved data persistence.What are the benefits of web services?

A service application component on one machine (typically the application server) is accessible to a client application on another machine (or any number of client machines). In Java we have the JAX-WS and JAX-RS architectures available (Java API for XML Web Services and Java API for RESTful Web Services). The difference lies primarily in the communication protocol used by each.

JAX-WS employs the simple object access protocal, or SOAP. In a SOAP application, XML is used to communicate method and value calls, as well as define the text (and other content) displayed on a web page. SOAP is more complex than HTTP, but also more secure (and much more flexible). JAX-RS employs the typical methods of HTTP, like Post and Get. It is simpler, but less secure and restricted to HTTP messaging methods.

Our use of the JAX-WS API and the SOAP protocol, it may also use other transport protocols. It's defining feature being the XML based messaging system. XML in one system is encoded to invoke functions (or pass values) in another. Because all operating systems are compatible with XML, cross-platform issues are resolved by implementation of the API.

Web services of either type (SOAP or HTTP) offer the advantage of being language independent. The language used to write the the component(s) to be accessed from a server does not have to be the same language as that used to create client applications for making use of the component(s). Both types may use XML as a means of communicating messages (method calls, data transfer, web page text labels, images, video, etc.). How those messages and other elements are displayed on a client side web page (or within a stand alone GUI) may well involve HTML, CSS, JavaScript, Flash, and any number of other languages, which interact with and control the DOM.

For persistence (storing data generated and accessed by the application's users) we used Enterprise Java Beans, and a MySQL database. EJBs provide a framework solution that autogenerates the bridge (JDBC API) between the relevant Java objects and the MySQL database tables. Queries are generated and managed by a persistence manager which itself is generated when the EJBs are created.

**How would you troubleshoot and test a web service? Would you use a debugger? Why or why not?**

Testing and troubleshooting involves several parts with this style of architecture. As we did in the course, build the server side, build the client side. Test the app using localhost to run a web server (like Glassfish, though I also have Apache and FileZilla). The web server handles the server code and web page display (in your chosen browser). In the case of the applications we've made for the course, both the client and server are created in NetBeans, so the code for both can be debugged using NetBean's debugger tool. The MySQL server is also involved and can be accessed through NetBeans "Services" tab in the left pane. The MySQL database can also be accessed directly via command line, or using a MySQL database management GUI tool. Connection problems will show in a NetBeans console pane. With MySQL, the command line window can be used to build and run a database directly and test for CRUD functionality in isolation.

With client/server apps there's always the potential for errors derived from data transfers between components. Those may well not be picked up by the debugger tool, especially if the error is not in the Java code. Communication problems will show up in the server log, and as Java errors in the console pane. Our apps. also have an application server accessing a database on a MySQL database server (JDBC): which will also have a server log and can show errors in the NetBeans console pane. Another source of errors is the servers themselves. NetBeans is pretty good at pointing those out. It lists out server status messages in a pane provided for each server in use.

The types of errors that get displayed (in the console pane) will often point to the source of the problem, although it can be tricky. In the course, on the first GUI app we created, I spent a couple days trying to figure out a Java error where NetBeans/JVM was pointing me at a line in one class file. It turned out to be a code error in another class file altogether. The debugger was no help in that case either. With the SOAP application I spent a week trying to configure around an "accessExternalSchema" error. It turned out to have nothing to do with the NetBeans, JDK, JRE, or GlassFish components recently installed for the course. Actually, the problem stemmed from an old GlassFish installation that was hijacking the application while not being configured to run it. The solution was to simply shut down the server and delete it. I'm still getting a SOAP based messaging error when testing the SOAP app internally (by right clicking the web services and opening test pages), however: the client app works fine, and passes all CRUD tests, so the web service is, in fact working fine.

To answer the question more directly, troubleshooting this type of architecture involves many different possible efforts. Code can be debugged, but a large part of this is configuring components to work together. The interactivity between components may be interrupted for a wide variety of reasons. The functionality of servers and clients may be affected by internal misconfiguration (affecting local functionality) as well as the configuration of connections to other components (affecting messaging).

**Do you think that your client/server application could be enhanced to consume a web service or be used as a web service? Why or why not?**

Actually, the apps we developed in the Live Chats (and for our IP assignments), are web services by design. The only thing keeping them from being available on the internet is the use of localhost for a domain, and a local web server. If reconfigured, the apps can use an internet domain and store the appropriate files in a server using that domain, they would be web services available on the web. It's a matter of how the connections are configured in the IDE when the application is built. In particular, the SOAP application developed during chats 9 and 10 would be a good candidate, although it would require the development of a GUI for client use, or the addition of XML driven web pages. The rest app. already has that functionality, as does the JSF application.

While the JSF application we made for week 4 may not seem like a web services app in the same vein as the SOAP and REST apps constructed in this week's chats, it is actually a self contained SOAP application. The client side is provided (via polyfilled web pages) in a web browser, while the server side could just as well be an internet domain server in lieu of the localhost/Glassfish components we used for the course. By reconfiguring the server (and MySQL server) for my www domain I could make the "client" side available generally to the web. Since the server and client are using XML to communicate using serialized objects, it is essentially a SOAP style web services application. The company that hosts my domain does offer MySQL server service, so that could be handled as well.

## How would you troubleshoot and test a web service? Would you use a debugger? Why or why not?

Testing and troubleshooting involves several parts with this style of architecture. As we did in the course, build the server side, build the client side. Test the app using localhost to run a web server (like Glassfish, though I also have Apache and FileZilla). The web server handles the server code and web page display (in your chosen browser). In the case of the applications we've made for the course, both the client and server are created in NetBeans, so the code for both can be debugged using NetBean's debugger tool. The MySQL server is also involved and can be accessed through NetBeans "Services" tab in the left pane. The MySQL database can also be accessed directly via command line, or using a MySQL database management GUI tool. Connection problems will show in a NetBeans console pane. With MySQL, the command line window can be used to build and run a database directly and test for CRUD functionality in isolation.

With client/server apps there's always the potential for errors derived from data transfers between components. Those may well not be picked up by the debugger tool, especially if the error is not in the Java code. Communication problems will show up in the server log, and as Java errors in the console pane. Our apps. also have an application server accessing a database on a MySQL database server (JDBC): which will also have a server log and can show errors in the NetBeans console pane. Another source of errors is the servers themselves. NetBeans is pretty good at pointing those out. It lists out server status messages in a pane provided for each server in use.

The types of errors that get displayed (in the console pane) will often point to the source of the problem, although it can be tricky. In the course, on the first GUI app we created, I spent a couple days trying to figure out a Java error where NetBeans/JVM was pointing me at a line in one class file. It turned out to be a code error in another class file altogether. The debugger was no help in that case either. With the SOAP application I spent a week trying to configure around an "accessExternalSchema" error. It turned out to have nothing to do with the NetBeans, JDK, JRE, or GlassFish components recently installed for the course. Actually, the problem stemmed from an old GlassFish installation that was hijacking the application while not being configured to run it. The solution was to simply shut down the server and delete it. I'm still getting a SOAP based messaging error when testing the SOAP app internally (by right clicking the web services and opening test pages), however: the client app works fine, and passes all CRUD tests, so the web service is, in fact working fine.

To answer the question more directly, troubleshooting this type of architecture involves many different possible efforts. Code can be debugged, but a large part of this is configuring components to work together. The interactivity between components may be interrupted for a wide variety of reasons. The functionality of servers and clients may be affected by internal misconfiguration (affecting local functionality) as well as the configuration of connections to other components (affecting messaging).

## Do you think that your client/server application could be enhanced to consume a web service or be used as a web service? Why or why not?

Actually, the apps we developed in Live Chats 7—10 (and for our IP assignments), are web services by design. The only thing keeping them from being available on the internet is the use of localhost for a domain, and a local web server. If reconfigured, the apps can use an internet domain and store the appropriate files in a server using that domain, they would be web services available on the web. It's a matter of how the connections are configured in the IDE when the application is built. In particular, the SOAP application developed during chats 9 and 10 would be a good candidate, although it would require the development of a GUI for client use, or the addition of XML driven web pages. The REST app. also needs user friendly functionality, but both offer test pages. Test pages are generated on the server side, and work as a means of testing some of the functions of the Web Services, The JSF application is self contained and autogenerates web pages suitable for any user. The JSF application does not technically contain web services components, but uses SOAP style communication between its web pages and the Java methods. Messages are passed using XML tags with values contained within a JSON syntax. It is using form of messaging that more closely resembles SOAP, than the POST, GET, PUT, and DELETE methods of HTTP.

While the JSF application we made for week 4 may not seem like a web services app in the same vein as the SOAP and REST apps constructed in this week's chats, it is actually a self contained SOAP application. The client side is provided (via polyfilled web pages) in a web browser, while the server side could just as well be an internet domain server in lieu of the localhost/Glassfish components we used for the course. By reconfiguring the server (and MySQL server) for my www domain I could make the "client" side available generally to the web. Since the server and client are using XML to communicate using serialized objects, it is essentially a SOAP style web services application. The company that hosts my domain does offer MySQL server service, so that could be handled as well.

## Architecture, Screenshots, Testing and Users Manual

Architecture, testing with screenshots, and the user’s manual were completed in Phase 4 for the JSF application. They can be found on pages 28—46 of this document. All the CRUD functions have been tested, using the web pages. These web pages effectively function as the client side of the application, although they are an integral part of the server based app, and were autogenerated during its construction.

## Web Services Applications

During week 5 we created two applications containing web services. One uses SOAP style communication, and includes separate Server and Client applications. The other uses REST style messaging according to the HTTP. Given the length of the course, there has not been enough time to develop fully user friendly clients for either application. However both provide the ability to use the web services to display test pages in a browser.

## REST

The REST application allows the user to access the database using POST, GET, and DELETE functions common to HTTP messaging. The user implements POST requests via a text area which allows formatted input. Input is then communicated to the MySQL database and the appropriate functions performed. This application’s web pages provide two forms of message syntax. POSTs, or information displays via GETs, may be formatted as either XML or JSON. Screenshots are provided below to demonstrate the application.

The application is used by first loading it into NetBeans and then running it. A test page will open the default page “TODO write content”, which could be developed into a home page or splash screen for the app. To test the web services, the user may R click on the “RESTful Web Services” folder, in the projects pane, or open the folder and test the services individually (also with a R click on a particular service).

Various choices are available to ensure that the application is functional. Clicking on either entity displays a drop down for that entity (Customers or Products) . Clicking the “Test” button will display a list of all items in that database table. The display may be either JSON or XML, depending on which format of GET is selected in the drop down. The user may also POST by id, create a defined range for display, or DELETE items from the database. Once the test button has been used, a panel is displayed with various buttons. Below the buttons is a text area for display of responses from the database(for GET functions). When POST is selected a textarea is made availabe for typing in requests. The POST may be either XML or JSON. If an XML POST is selected then the user must enter data in XML format. The same is true for JSON. While cumbersome, the test page does offer the ability, within a web browser, to test the functionality of the application.

## SOAP

The SOAP application offers the potential for the greatest flexibility and security. It is made up of two main components, a server side application which contains the web services, and communicates with the database, and a separate client application. The client application contains WSDL files to facilitate communication with the web services provided in the server side application through the use of XML based messages.

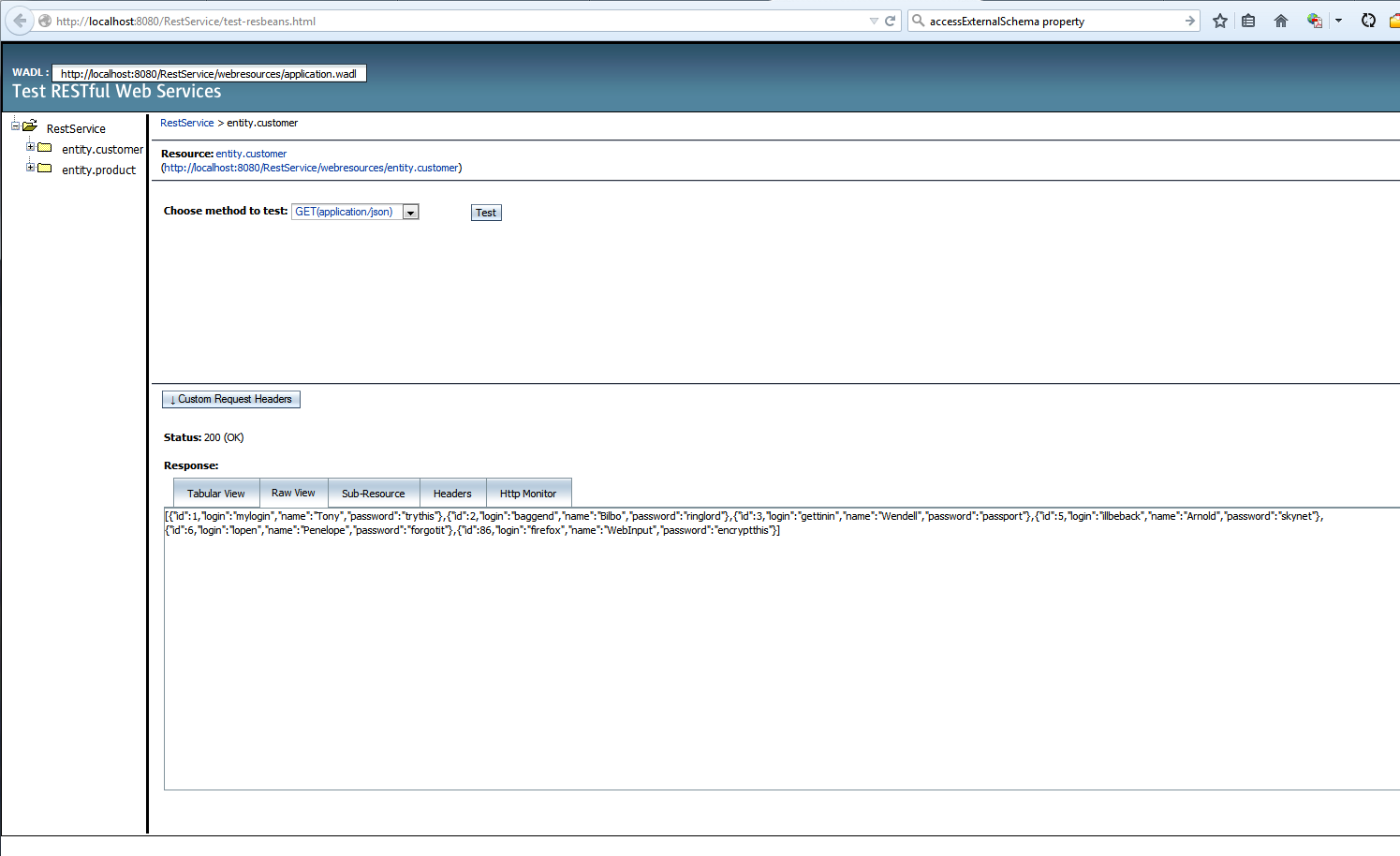
The client could be developed in any language and for either a GUI style user interface, or using web pages (JSPs or other).Given time constraints, we only developed the client to demonstrate the “findAll” functionality, but I have added a menu and almost all the other CRUD functions available through the web service. Find(1), and count methods were not created, but the rest have been. They may be tested in the console using a menu provided.

The server side of the application has the same “Test Web Services” ability as the REST app. By R clicking on the web services folder a drop down menu offers a test choice. This opens a web page with the various CRUD functions displayed, along with input fields (where appropriate).For me, only the “findAll” and “count” functions worked from the test page. Any function requiring user input generated a lengthy server error related to Primary Keys.

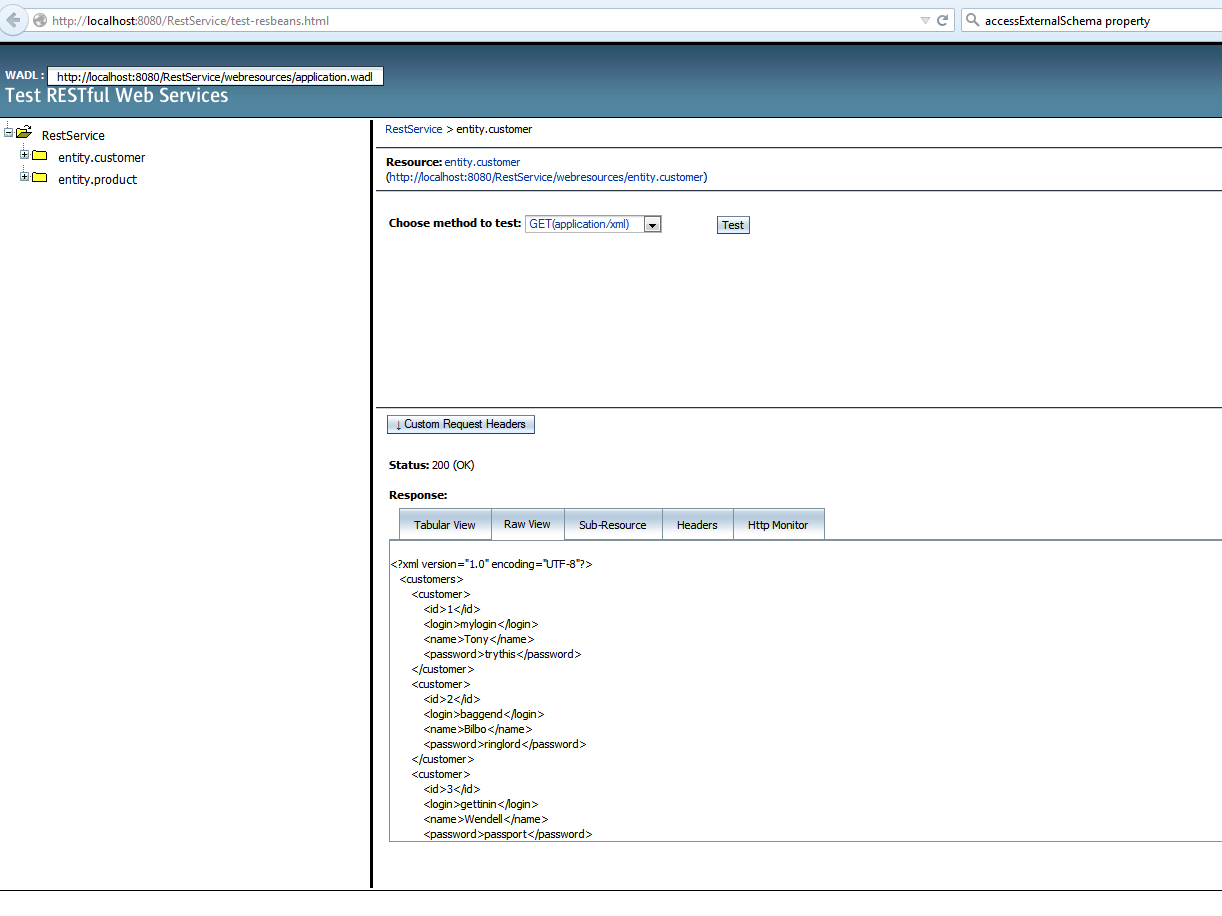
However; having developed a full set of methods within the client, I have determined that the web services are working fine. I’m able to findAll, “edit”, findRange, create, and delete using a Scanner object and the console. A menu was developed to allow easy transitions from one test, to another for either type of object in the database (Customers or Products). Some screenshots will be shown below.With the methods developed, a GUI could be readily constructed to improve user friendliness as well as look and feel.

## Web Services Screenshots

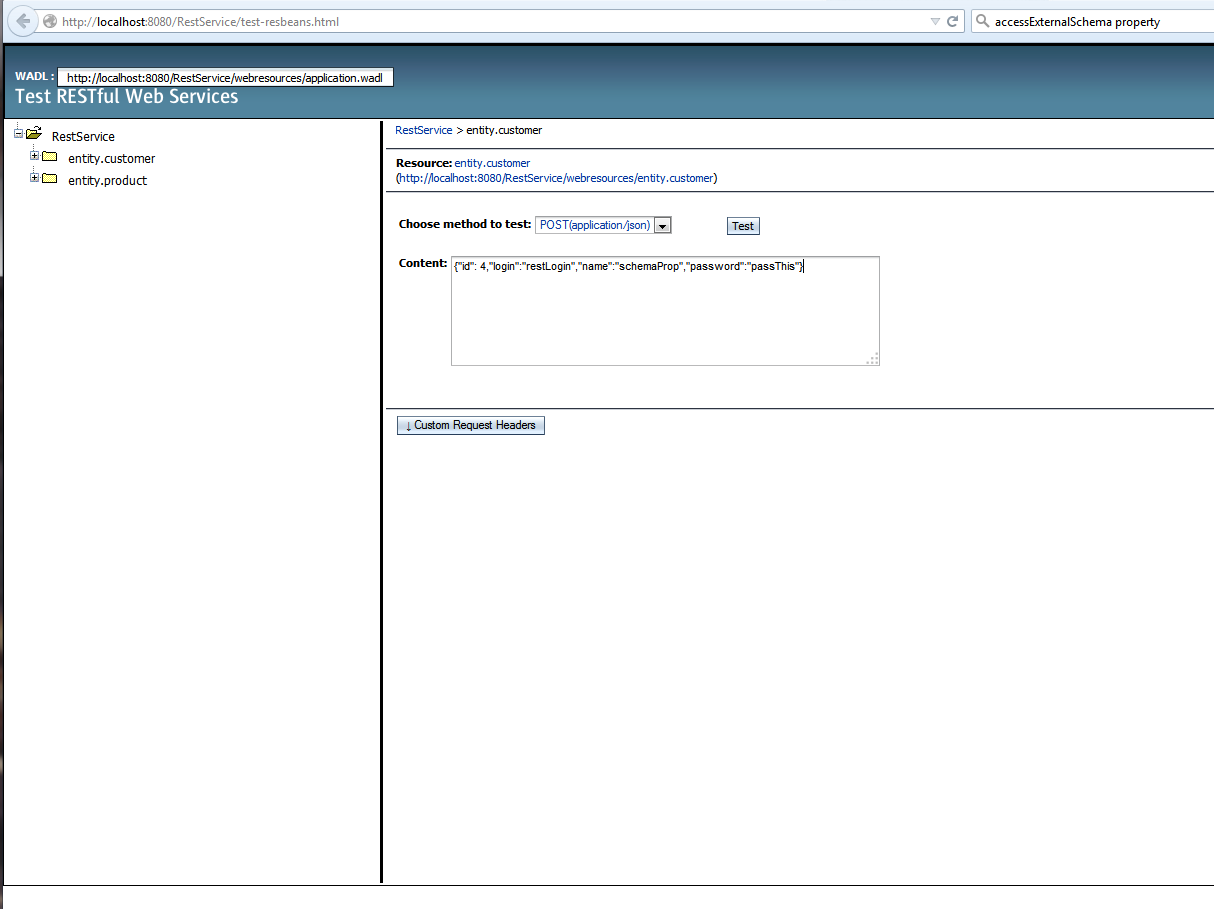
### REST Application



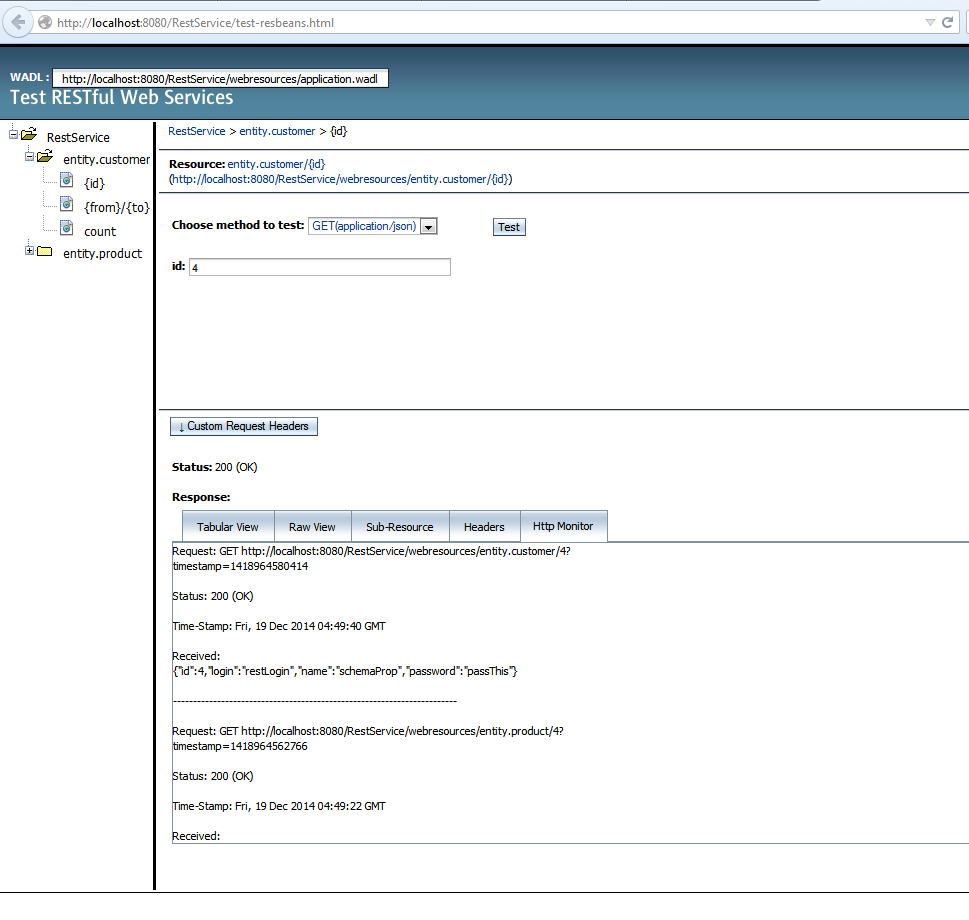
JSON GET to list all customers on the test page ( Web Service findAll method). The user chooses an “entity” on which to perform CRUD functions in the left column. The drop down contains choices for GET and POST using either XML or JSON as a message syntax. Once the choices have been made, the user presses the “Test” button. Results are shown in textarea below the tabs. Immediately above the row of tabs, toward the left, is an HTTP status message. 200 Lets the user know that the communication request was successful.



This is the same request as the one above, but repeated using the XML option.

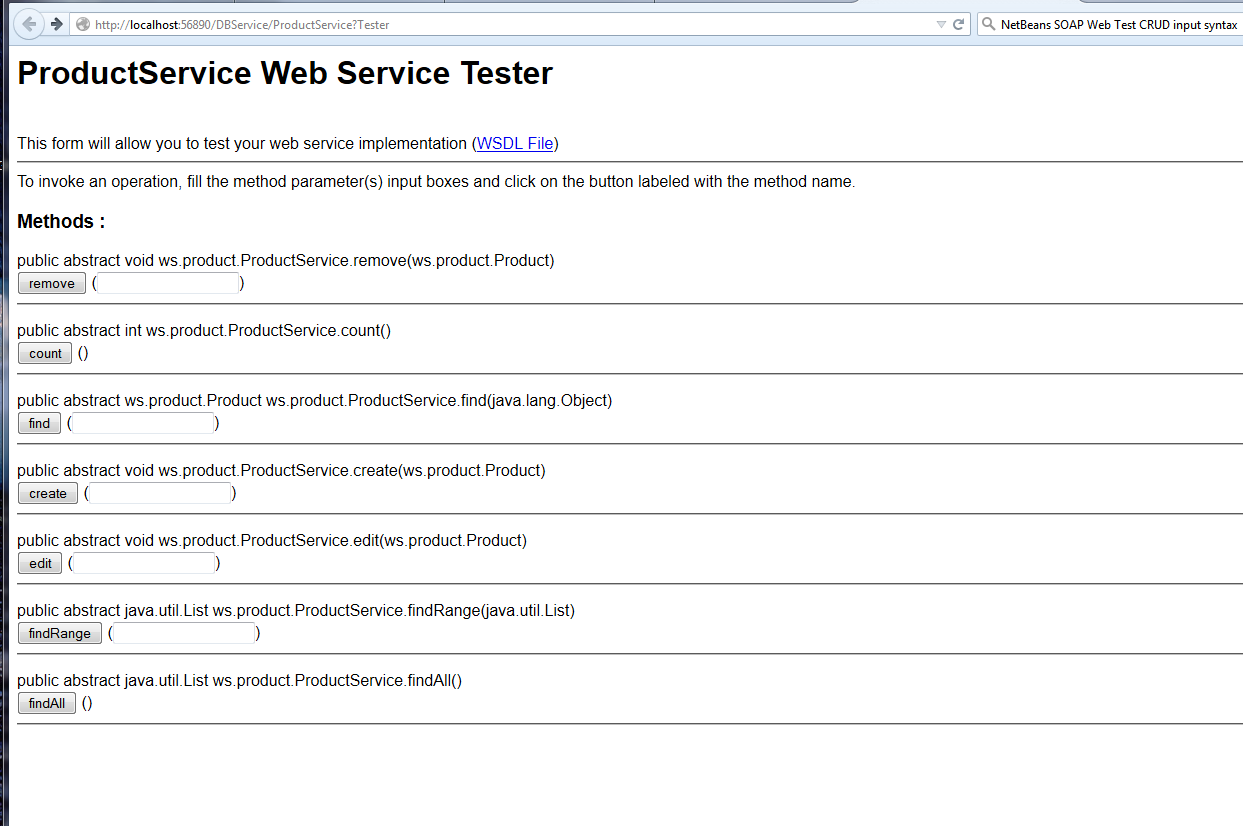


Here, a “create” POST has been typed into the input textarea. Since the JSON option is selected, the POST is written in JSON format.

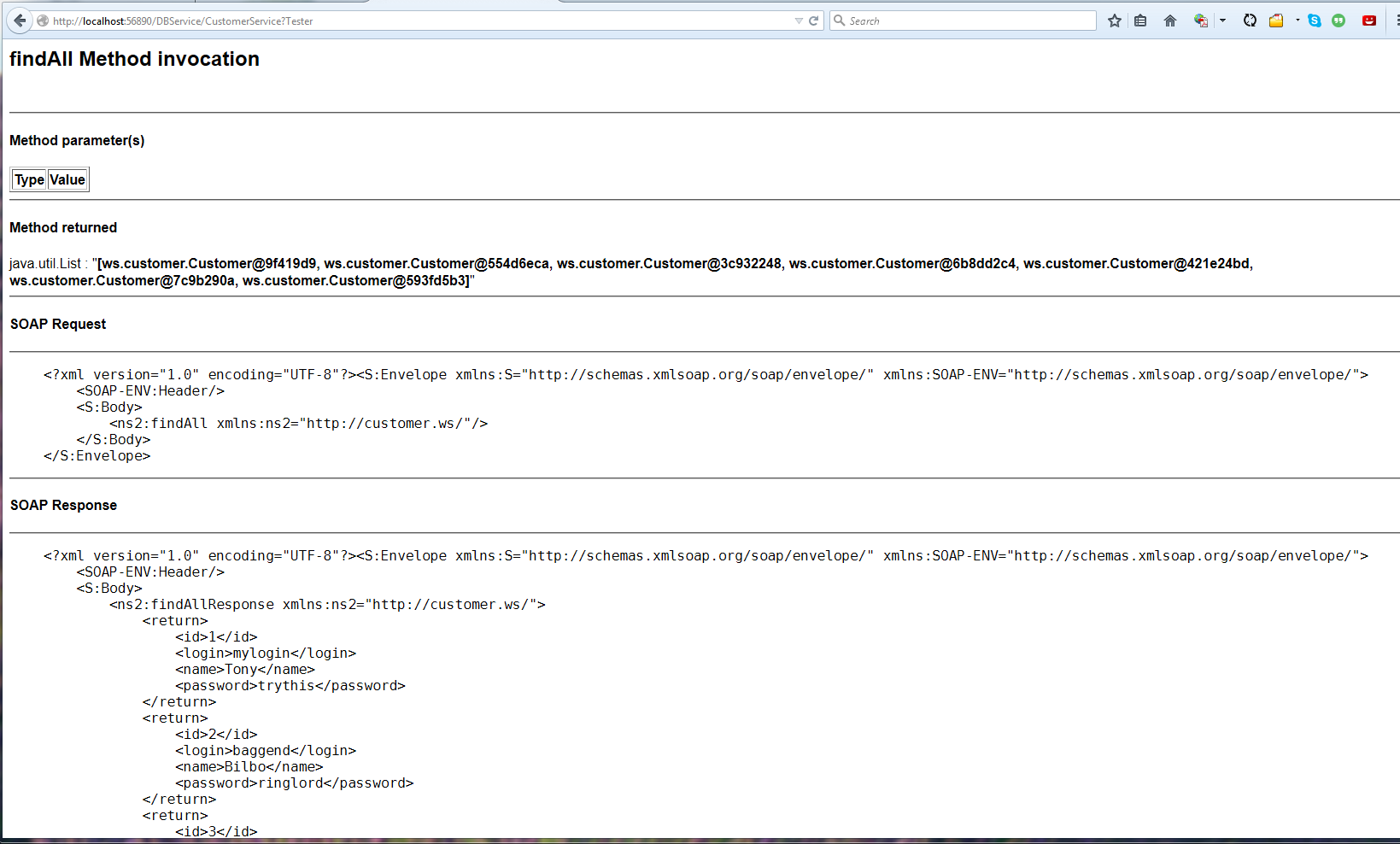


The POST of the previous image was successful (status: 200) and the server returns a copy of the message it recieved. Since the server would have passed the message on to the database, the 200 status also indicates that the database was successfully updated with the new information. The view here uses the Http Monitor pane. Note that messages are time stamped and the message path is displayed.

### SOAP Applications

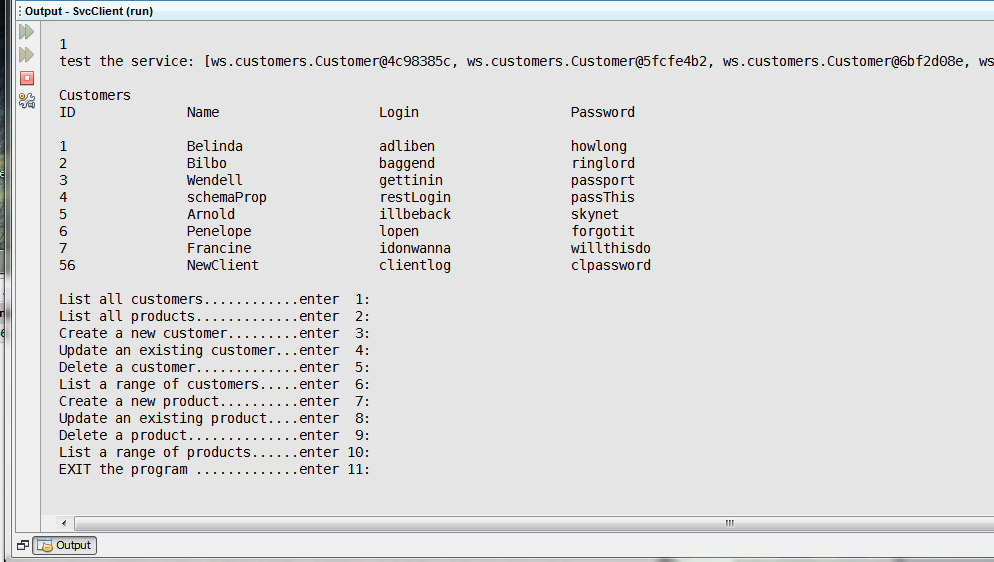


This is an example of a test page generated by R clicking on the Product Web Service in the server side application. It is testing the functionality of the WSDL file (located in the client app). This type of testing establishes that there is a connection between the server application and the client application.

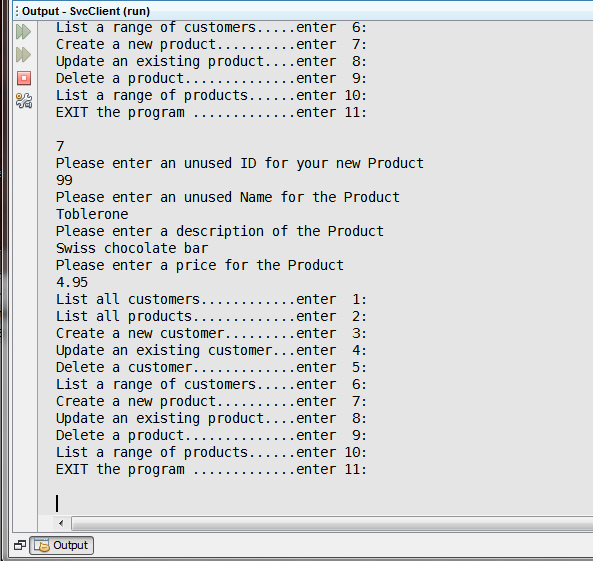


A Customer findAll method has been executed (button at the bottom the form on the previous page shown). The request is displayed in the central pane. The result is shown below in a scrollable pane. Request and response messages are displayed as XML.

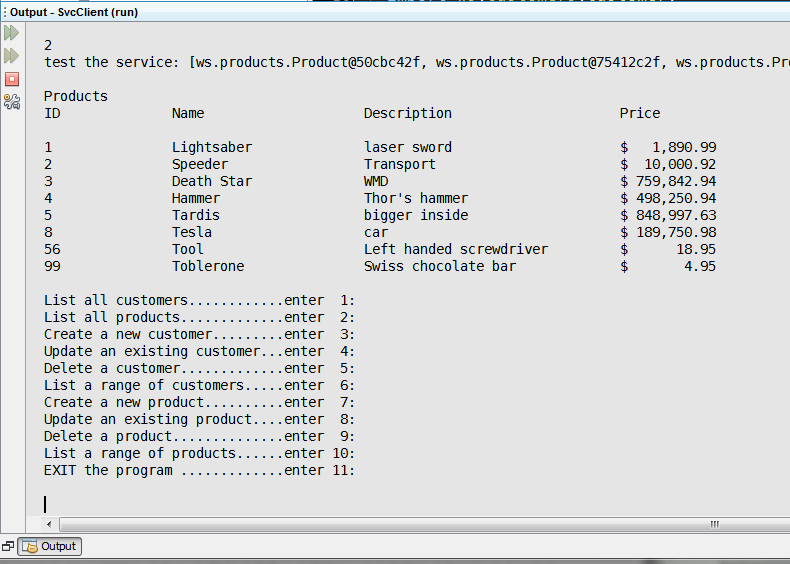
### SOAP Client Images



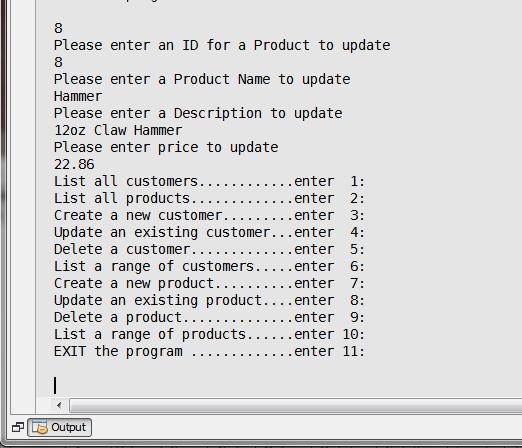
Using the console to display CRUD methods run in the Client app. & using the WSDL methods to effect changes in the database via the server side web service. This shows a findAll on customers.



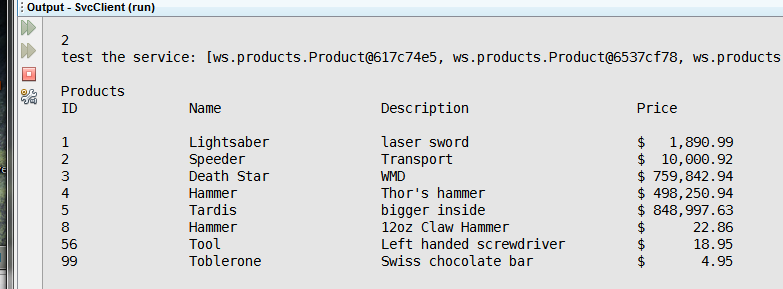
A new product is added to the database.



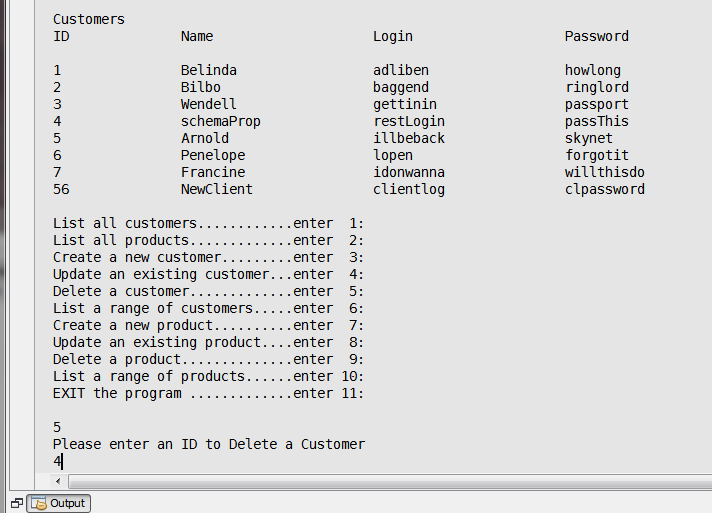
This shows the new product in the product table (id99)



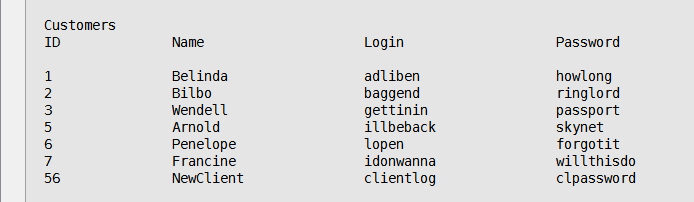
Product with id 8 is being updated.



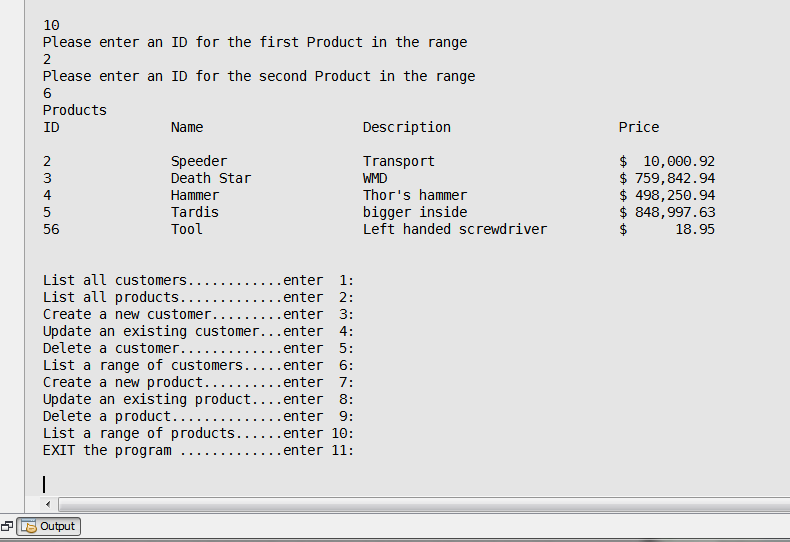
Product 8 has been updated.



Testing the delete function (WSDL “remove”). Here, customer 4 is being removed.



Confirms that customer 4 was deleted from the database.



Using WSDL “findRange” to create a range of products, and displaying them.

# References

dbForge(2014).dbForge Studio for MySQL.Devart: Prague, Czech Republic

<http://www.devart.com/dbforge/mysql/studio/table-designer.html>

Deitel, P., Deitel, H.( 2012 ).Java—How to Program, 9th Edition ( Early Objects Version). Pearson (Prentice Hall): ebook, web

Fowler, M.( 2004 ).UML Distilled, 3rd Edition.Pearson Education, Inc : Boston, Ma.

ebook, web ( class diagram, chapter 3, pg 36 )

Izawa, E.(2005).Rei's Random Guide to MMP Gaming Terms.MIT: Cambridge, MA

<http://www.mit.edu/~rei/game-terms.html>

Jollymore, W.(2013).Debugging Java Programs with NetBeans.Sheridan College: Sheridan, WY <http://www-acad.sheridanc.on.ca/~jollymor/netbeansDebug.html>

Lowe, A.(2014).Advanced Java Programming.Colorado Springs, CO: CTU Online,

Live Chats 1 — 8.

Mallett, A.(2014), Java: Building a GUI with NetBeans.TheUrbanPenguin

web: <http://theurbanpenguin.com/wp/?p=2337>

NetBeans(2012).Color Schemes of the NetBeans IDE, Brandon Dark.NetBeansThemes.com

web: <http://netbeansthemes.com/brandon-dark/> ( to set up themes for your editor )

Oracle(2014). Java Platform, Standard Edition 8 API Specification.Redwood Shores, CA: Oracle, web: <http://docs.oracle.com/javase/8/docs/api/>

Oracle(2014).Java Tutorials, Overloading Methods.Oracle, Redwood Shores, CA:

web: <http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>

Oracle(2012).Oracle Solaris Studio 12.3: Performance Anayzer.Oracle Solaris Studio 12.3 Information Library. <http://docs.oracle.com/cd/E24457_01/html/E21994/afamv.html>

Oracle.(2014).Programming Advanced Features of WebLogic Web Services Using JAX-WS.Oracle: Redwood Shores, CA

<http://docs.oracle.com/cd/E13222_01/wls/docs103/webserv_adv/handlers.html\>

Rouse, M.(2014).JAX-WS (Java API for XML Web Services).TechTarget.Newton, MA

<http://searchsoa.techtarget.com/definition/JAX-WS>

<http://searchsoa.techtarget.com/definition/SOAP>

<http://searchsoa.techtarget.com/definition/REST>

tutorialspoint.(2014).Servlets - Quick Guide.tutorialspoint: Hyderabad Telangana, India

<http://www.tutorialspoint.com/servlets/servlets-quick-guide.htm>

zzynx(2004).Horizontal Line on JPanel.Experts Exchange. web (no physical address)

<http://www.experts-exchange.com/Programming/Editors_IDEs/Q_21055459.html>