**Final Documentation**

**for**

**Price Comparison Application**

Capstone: Fall 2012

Lucy Horpedahl, Nathan Noonan, Daniel O’Doherty, and Nick Richardson

**Table of Contents**

**Table of Contents** 2

**1.** **Introduction** #

**1.1 Client** #

**1.2 Application** #

**2.** **Functional and Nonfunctional Requirements** #

**2.1 Functional Requirements** #

**2.2 Nonfunctional Requirements** #

**3.** **Architecture and Design** #

**3.1 High-Level Architecture** #

**4.** **Implementation Issues** #

**5.** **Testing** #

**6.** **Lessons Learned** #

**Index of Figures** #

**1.** **Introduction**

**1.1** **Client**

The client for the application we chose to design for this project is Justin Richardson, the owner of a small business, Digital Attitude. The business, which is located in Crossroads Mall, sells Japanese animation collectibles, movies, books, video games, and more.

We spent time at Mr. Richardson’s business, learning about how the store runs. Most, if not all, of his products have barcodes which are scanned in order to read the UPC number of the item. The items are entered into a database and organized by this number so that searching for this number will return the proper item. These barcodes are central to both our original application idea and our updated application.

**1.2** **Application**

**1.2.1** **Original Proposal**

Our initial plan was to design a new point-of-sale system for Mr. Richardson’s store. A point-of-sale system handles nearly all of the necessary operations of a retail business. It allows inventory to be searched and updated, purchases to be made, and customer information to be stored.

We believed that Mr. Richardson did not have a proper point-of-sale system, and we planned to create one that would fit his business’s unique needs. For example, Mr. Richardson buys used items from customers and then sells them in his store. He selects the pricing for these transactions by searching Amazon.com, which is one of the few places that sells most of the items that Mr. Richardson sells in his store.

We planned to implement, as part of the point-of-sale system, a “price checker” that would search Amazon.com and display the current price for the items so that Mr. Richardson would not have to do these searches manually. Upon meeting with Mr. Richardson further, we discovered that his point-of-sale system was adequate, but he told us that the “price checker” feature would be very helpful.

**1.2.2 Modified Proposal**

Because the point-of-sale system would have been a very large project, and the system currently used by the client was sufficient for his needs, we decided to narrow the scope of our project to just design an application that would search Amazon.com and retrieve prices for items in Mr. Richardson’s inventory. We planned to include options for filtering and sorting the output as well as for making changes to the prices in the database through the application itself, rather than having to go back to the point-of-sale system to modify the prices.

The application would need to communicate with Mr. Richardson’s existing database, process the data to retrieve the UPC numbers, communicate with Amazon.com to search for the UPC numbers and retrieve the prices, process the prices to display them on an interface, and ultimately communicate with the database again to write new prices back to the database.

Ultimately, we ran across a number of difficulties in designing this application which caused us to modify certain features, but the basic architecture would remain the same. The rest of this document describes in more detail the work that was done to design this application.

**2.** **Functional and Nonfunctional Requirements**

**2.1 Functional Requirements**

**2.1.1 Database Communication**

The application will need to have a connection to the client’s database which contains the store’s inventory.

**2.1.1.1 Read from the database**

The application will need to be able to query the database using SQL to obtain a list of all used products in the client’s database.

**2.1.1.1.1 Use Case**

**Name:** Read from Database

**Actors:** Initiated by user

**Entry Condition:** User launches application

**Flow of Events:**

1. The application will attempt to establish a connection with the database.
2. The application will pass a query to the database requesting a list of items.
3. The database will return the result of the query to the application.

**Exit Conditions:** The application will have a list of items.

**Exceptions:**

* The database is not found or is inaccessible.
* The query returns no results.

**2.1.1.2 Write to the database**

The application will need to be able to write to the database by using SQL queries to modify prices of items.

**2.1.1.2.1 Use Case**

**Name:** Write to Database

**Actors:** Initiated by user

**Entry Condition:** User chooses to modify price(s)

**Flow of Events:**

1. The application will attempt to establish a connection with the database.
2. The application will pass a query to the database containing the item(s) and the new price(s) based on entry in the interface.
3. The database will acknowledge the changes made by the application.

**Exit Conditions:** The database will reflect the price change(s).

**Exceptions:**

* The database is not found or is inaccessible.
* The SQL query returns an error.

**2.1.2 Online Shop Communication**

**2.1.3 Data Processing**

**2.2 Nonfunctional Requirements**

**2.2.1 Hardware and Software Requirements**

The application will need to be run on a PC that has a connection to the client’s database via MS SQL Server and an internet connection. No other specific hardware or software is required.

**2.2.2 Reliability**

In order to execute all functionality, the application will rely upon a connection to the client’s database and a connection to the internet.

**2.2.3 Availability**

Because the application is not safety-critical, its availability needs are less imperative. However, that does not mean that the system should not be available as much as possible. The application should be available particularly during the client’s business hours. Also, the application requires access to the internet, but it should not rely on an internet connection to run, only to update its information. In the event of a loss of internet connection, the application should still display previously obtained information, and it should continue to be connected to the database and able to modify the client’s inventory as desired.

**2.2.4 Security**

Because the application will have access to read from and write to the client’s database, it should be secured from use by unauthorized persons. The pricing information obtained online is publicly available, so there is no security concern for that data. The client’s PC will be the only system running the application, and the client’s own security measures should ensure the security of the application.

**2.2.5 Maintainability**

The application should be relatively autonomous once development is complete. It should not need updates on a regular basis. If updates need to be performed, the application could most likely be updated outside of the client’s normal business hours to prevent impeding the client’s business.

**2.2.6 Portability**

The application will be designed for this particular client and is not being distributed to anyone other than the client. After development and deployment, there will be no need for the application to run on any system other than the client’s.

**3.** **Architecture and Design**

**3.1 High-Level Architecture**

Figure 1: Price Comparison Application Architecture

Point-of-sale system

Inventory Database

Price Comparison Application

Amazon.com/Google shops

*Figure 1* shows how the price comparison application interacts with the existing modules. The application must communicate with the existing database in order to read and modify records in the inventory. The database also communicates with the point-of-sale system, and the changes made by the price comparison application must be reflected in the point-of-sale system.

The price comparison application also must communicate with Amazon.com or Google shops in order to search for specific barcodes and retrieve the price at which the item is being sold by various merchants.

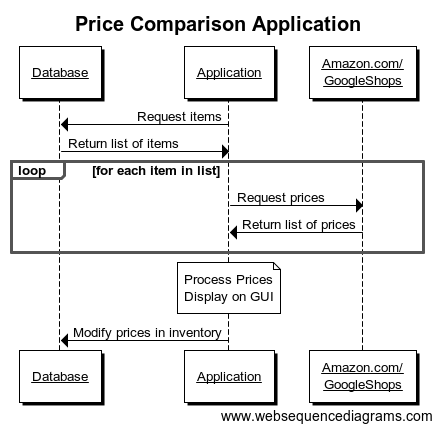


Figure : Price Comparison Application Sequence Diagram

*Figure 2* shows the specific communications necessary for the application. First, the application will retrieve all used items from the client’s database. When it has a list of barcodes, the application will send requests to Amazon.com or Google shops for each item. Each request will return a list of offerings for the item, which will tell how much the item is being sold for at various locations online. After retrieving this list, the application will perform statistical calculations to find an average price for each item. These averages, as well as the current price in the client’s database, will be displayed on an interface that will allow the client’s database to be modified if the user chooses.

**4.** **Implementation Issues**

**5.** **Testing**

**6.** **Lessons Learned**

**X.** **Section**

**X.1** **Heading**

**X.1.1** **Sub-heading**

**X.1.1.1** **Point**

**Index of Figures**

fig. 1: Price Comparison Application Architecture #

fig. 2: Price Comparison Application Sequence Diagram #