

# Mash Up Home Library System

Suresh Jeyaverasingam  
University Of New Brunswick  
Fredericton, NB, Canada  
c244s@unb.ca

Yuhong Yan  
Institute of Information Technology,  
National Research Council  
Fredericton, NB, E3B 9W4, Canada  
Yuhong.Yan@nrc-cnrc.gc.ca

## Abstract

*We present an online system to manage a home library, that consist of multiple collections, such as books and CDs, using Service Oriented Architecture. Many collections can contain hundreds and thousands of books and CDs. Managing a personal collection of such magnitude could be difficult if one were to rely on using databases or spread sheets. Entering detailed information about each item is a huge undertaking and tedious work. We present a solution to solve this complexity by building an online service to manage a home library. The online service maintains a database to record each user's collection. The users will scan the bar code of the books and CDs. The detailed information for the items will be retrieved from Amazon.com via the Amazon Associates Web Service interface and automatically entered into the database. Users are able to edit and search their items. A simple loan management functionality is provided to allow users to borrow out items. Doing so would allow them to share their collections with their friends. Compared to other commercial products for managing home libraries, our solution is a purely online service that absolves users from performing complicated installations and maintenance. The usage of Amazon.com's Web Service to retrieve item information provides a reliable source for information retrieval. This is due to the currency of information available and the fact that Amazon.com has one of the largest catalogues of items, from book and, CDs to home appliances. All of these are made available via the Amazon Associates Web Service. Our system can be extended to manage other items such as prescription medications, and kitchen items. These features make the developed solution an attractive alternative to commercially developed products.*

## 1. Introduction

The idea for developing this system was motivated by one of my friends who is an ardent classical music lover. He collects CDs of string quartets from all over the world. He purchases works not only from famous publishers in

Berlin and Paris, but also from Russia, South Korea and Japan. His personal collection of CDs and DVDs spans seven large book shelves in his library. His huge collection poses a problem for him. Firstly, he is unaware of the exact number of CDs that he has. Secondly, he is unable to remember the amount of duplications he has in his collection. Quite often, after purchasing a music CD, he has to return it back to the store after finding that he has the exact same title in his library. There must be a better way for him to manage his collection. Thus, an idea of using a database to help him manage his collection emerged. Building the database was not a problem but was the input of the information. It is inconceivable to expect anyone with such a large collection to manually input the metadata (such as the composers, players, and tracks) of a product into a database. Such a manual effort is not only intensive but also tedious. The developed database must also be user friendly, as it is geared towards home collectors who are interested in cataloging their collections in the easiest and quickest manner. Thus, the database must not only be simple to use and operate, but also reliable as we don't want users to lose their data after spending hours inputting it in the first place!

Amazon.com [1] has one of the largest databases of various types of music and books around the world. It is also up to date and is continuously growing as new products are added. New collections are also being added regularly, increasingly making it the one stop centre for sourcing metadata about a particular product. Our online system is based on Amazon's Web Service. Service Oriented Architecture (SOA) was utilized here because the developed web application would be consuming an existing Web Service (Amazon Associates Web Service). The following outlines the operation of the developed online web application. The online application maintains a database to record each user's collection. Users will be able to scan the bar code on the back of books and CDs. Detailed information about the items are retrieved from Amazon.com via its Web Service interface and automatically entered into the database. The users are able to edit and search their items. Additionally, a simple loan management functionality is included so that users can share their collections with their own friends. Unlike other

commercial products for managing home libraries, our solution is a purely online service that removes the complicated tasks of installation and maintenance. Additionally, the fact that the database is backed by a large organization makes the integrity of the obtained data reliable. The developed web application is by far the most reliable system compared to commercial products mentioned in this paper. Our system can be extended to manage other items such as prescription medications, and kitchen items.

The rest of this paper is organized as follows. Section 2 details the system requirements to realize the online cataloging system. Section 3 describes existing work on similar cataloging systems. Section 4 describes the Web Service component of the developed online system. Section 5 will describe the implementation of the online system. Features of the new online system are also demonstrated in this section. Section 6 describes about future work that needs to be done, and Section 7 concludes the paper.

## 2. System Requirements

We came up with a list of requirements for our new system. The system:

- Needs to be easy to find a book/CD from the system
- Needs to be easy to add a book/CD to the system
- Needs to handle foreign languages
- Needs no installation other than a bar code scanner because it is an online service
- Can handle multiple users with multiple catalogues
- Can not have a lengthy initial population effort.

## 3. Existing Work

### 3.1. Library Management System

Terms used in this paper are borrowed from library parlance. Explanations of the terms are listed below.

**Catalogue/Collection** refers to a list of collections that a person owns. At the moment, it describes collections of books and music (DVDs, CDs) but can include other items such as kitchen items, home appliances and prescription medications.

**User** refers to clients who are either registered to use the system or guests who are simply browsing the collections.

**Circulation** refers to the management of processes associated when a user either returns or borrows a book. The processes that need to be managed are as follows:

1. **Due Dates and Fines:** Assign due date to users who are borrowing items and implement a fine if a user returns them past the due date.
2. **Hold Call:** Allows another user to place a hold on an existing item if it has already been lent to some one else.
3. **Monetary Transactions:** Allows the payment of Fines
4. **User Management:** Temporarily allows for management of users.
5. **Shelving Cart:** Places all returned items temporarily in the shelving cart before returning those items to their correct locations in the near future.

**Bibliography** refers to items in a collection

**Cataloging** refers to the process of adding new items to the existing catalogue.

### 3.2. Home Library Management

There are several applications currently on the market that appear to fulfill our needs. These applications can be divided into two main categories;

- 3.2.1. Applications that manage single collection
- 3.2.2. Applications that manage multiple collections

#### 3.2.1. Applications that Manage Single Collection

These applications only manage a single collection. There would be individual applications for managing books, music or movies collections. Examples of such applications are listed below.

#### Commercial Applications

**Collectorz.com Book Collector** Collectorz [2] also costs \$40 (actually \$39.95) and works on Windows and Mac. It has the same problem as Delicious Monster (see below) of feeling like it is aimed at smaller databases. It also does not have support for Library of Congress Numbers.

**Helium Music Manager 2008** Helium Music Manager [4] is used to manage CDs, MP3s and vinyl records. It costs \$ 39 and works only on Windows machine (on at least a Windows 2000 operating system). Database is populated by physically reading the music file or by importing data from another application (Windows Media Player, iTunes 4.5, previous version of Helium Music Manager 2006). This is not feasible if a user has thousands of music CDs as he/she is expected to place his/her CD into the CD tray so that the application can read it.

**OrangeCD Catalog** [5] This product seems to be the most promising. It not only can physically scan the media (CDs, DVD, Tapes, LP) but also read the data from a barcode and obtain information about the particular media from online music databases such as FreeDB and Amazon. It costs \$34.95. In addition, it comes with free player software and also allows users to conveniently create web pages (showing his/her collection) on the web. This product seems like a winner, except that its useful if a person is interested in managing only a music collection.

If the costs above are prohibitive, open source and 'free' applications are available to interested users. However, these applications are not as robust as the ones listed above. Examples of such free / open source applications are shown below:

### **Open Source / Freeware Applications**

**Windows Media Player, iTunes, Amarok** These applications need no introduction. They are mostly geared to users who have a small collection of music files. The applications gets track information from online music databases. They are severely limited as they do not allow users to add music information in any other way other than letting the computer physically read the media. They also do not have functionality to keep track of loans and returns.

### **3.2.2. Applications that Manage Multiple Collections**

These applications can manage collections such as books, music, movies and potentially other collections as well. The advantage of these applications is that instead of buying several applications from different developers, you only have to purchase one suite that manages entire collections. This means that there is seamless integration between collections. Additionally, there is only one developer to turn to if a provided application upgrade or patch affects a library system. This is certainly an attractive option from the customer support perspective. A list of applications that manage multiple collections are listed below.

### **Commercial Applications**

**Readerware** Readerware [7] is flexible and pretty fast. It has a decent interface and supports Windows, Linux, Mac, and even Palm. It allows customers to create their own columns. Additionally, it has commendable support for Library of Congress Catalogue information (with the addition of a provided Python script). It will find the LCC number based on the ISBN number. Readerware can also be customized with self written Python scripts. It costs \$40.

**Delicious Monster** Delicious Monster [3] also costs \$40. It runs only on the Mac OS, but that was not the biggest issue for us. Delicious Monster has a slick interface which most Mac users will find familiar. However, it feels like a better solution for organizing hundreds of DVDs than thousands of books. It also does not have good support of Library of Congress Catalogue numbers.

### **Open Source Alternative**

All the preceding listings were commercial applications. There is however an open source application that has all of the desired features. This application is listed below.

**Data Crow 3** [8] It not only manages books and music collections, but also movies, software and photo collections. It even has a simple loan management system. However it is lacking one important feature that this paper has developed, that is the ability to support both multiple users (i.e. catalogue owners) and multiple catalogues. In essence, this application does not allow each member of a household to manage their individual collections. This application is available in any platform as long as Java is installed.

Clearly the existing applications do not fulfill the goals of this paper. A new application that fulfills our goals and is completely web based has to be developed. To reduce development time, we turned to the open source community to find a package that has fulfilled some of our goals, we then modified that package to achieve the remaining goals. The chosen application package is described next.

### **3.2. OpenBiblio**

OpenBiblio [6] is an automated library management system that consists of an Online Public Access Catalog (OPAC), circulation, cataloging, loan management and staff administration functionalities. Figure 1 shows a screenshot of OpenBiblio.

OpenBiblio is written in PHP and utilizes JavaScript to enhance the user experience. It also has a reporting function that queries the database using a custom made .rpt file. The database backend used by OpenBiblio is MySQL. Installation is done by first creating a database and then running a web based installation script. The entire process is seamless and simple.

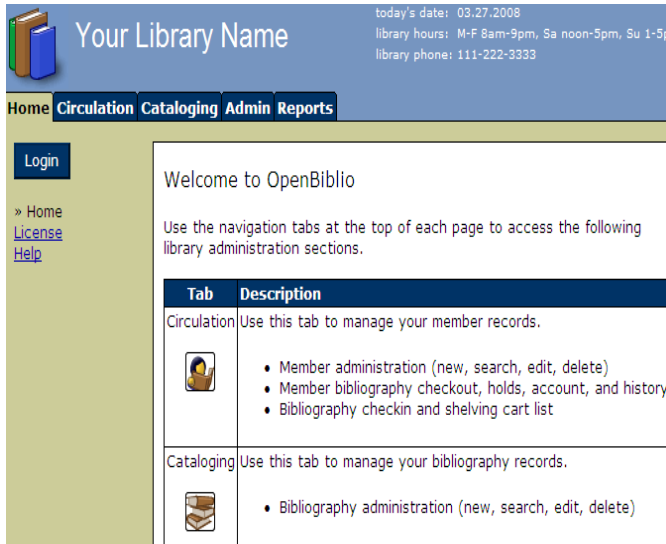


Figure 1: OpenBiblio screenshot

## 4. A Solution based on Web Services

### 4.1. System Operation

The developed online web application will need to operate in the following fashion, as depicted by the Figure 2 below.

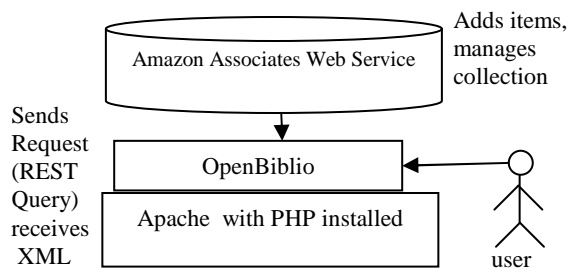


Figure 2: Operation Flow of the Web Application

### 4.2. Amazon Web Services

Amazon Associates Web Service (formerly known as Amazon E-Commerce Service) exposes Amazon's large array of products in a variety of detail to the developers. It allows developers to query the Web Service either via the

REST style query or via the SOAP protocol. Developers can assess the Web Service using a myriad of programming languages. The solution developed in this article uses the REST API to send request to the Amazon Web Service. A REST Request Query similar to the one below was sent

```

http://webservices.amazon.com/onca/xml?Service=AWSE
CommerceService
&SubscriptionId= 000000000AmazonKey
&Operation=ItemLookup
&ItemId= 1933988134
&IdType= ISBN
&SearchIndex=Books
ResponseGroup=SalesRank,ItemAttributes,EditorialR
eview,Reviews,Tracks
  
```

The level of detail of the returned response is governed by the *ResponseGroup* parameter. There are 57 response groups. A similar request query as shown in the preceding example was used to obtain both the Track and book information from Amazon. One could also specify the location to which the query could be addressed to. The developed application queries the main Amazon database that is hosted in United States of America. Compared to other Amazon locations (Canada, France, United Kingdom and Germany), the US location (termed *locale* by Amazon) has the largest listing of products. Naturally, this should be the first database to check to get metadata of a product.

To get complete information about a book or music, these response groups must be returned: *ItemAttributes* and *Tracks*. Additional response groups could be returned to give additional information (such as album art via *Images* or promotional details via *PromotionDetails* response groups).

After sending the request, an XML response would be sent back by the Amazon Web Service. Below is a snapshot of the sample response sent by Amazon.

```

<Item>
  <ASIN>B0002TXT1Q</ASIN>
  <DetailPageURL>..... </DetailPageURL>
  <ItemAttributes>
    <Artist>Joel Miller</Artist>
    <Artist>Kurt Rosenwinkel</Artist>
    <Binding>Audio CD</Binding>
    <EAN>0690579004624</EAN>
    <Format>Import</Format>
  </Item>
  
```

The returned XML would be parsed using Document Object Model (DOM). Other XML parsing APIs could also be used. The parsed data (UPC Code, Artist, Author,

EAN number, Binding, Price, Tracks, Product Dimension and other relevant product details) would be used to populate the relevant tables in the database.

### 4.3. Input Data using Scanner

Most books have a copyright page containing publisher and cataloging information. Books published after 1975 have an ISBN number, while books published after 1985 have a barcode on the back which contains the ISBN number. The ISBN (International Standard Book Number) is a unique number identifying that book. This number contains information about the book, where it is from, and who published it. You can use this number to look up the rest of the information about a book from many online sources.

The problem with using an ISBN number to catalogue a collection is that it is not an intuitive method. Sorting by ISBN numbers would create a list which does not have anything to do with the author or the subject of the books. This would create an effectively random order of books and make it very difficult to find what you are looking for.



Figure 3: Sample UPC Bar code for a book

### 4.4. The system design

The original OpenBiblio package has 24 tables. These tables pertain to the following functions:

- Managing Bibliographies (12 tables)
  - Adding items to the database, including copies of those items and associated status, collection and material codes, US MARC codes,
- Loan Management functionality (2 tables)
  - Checkout privileges, transaction code types. This functionality is closely tied to member management.
- History (1 table)
  - Loan history, Member history
- Member Management (5 tables)
  - Adding/editing/deleting members, loan management.
- Staff Management (2 tables)

- Governs how staffs can make changes to the library. What staff can do depends on privileges given by the administrator.
- Library Administration (2 tables)
  - For managing the look and feel of the library and changing general settings.

Conceptually, the original package works as shown on the next diagram.

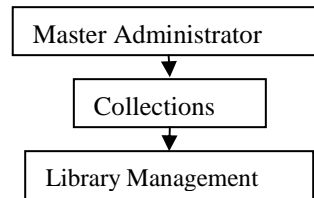


Figure 4: Conceptual Diagram of OpenBiblio Operation

The original OpenBiblio diagram only allows *ONE* person to manage his / her collection. To accomplish the aforementioned goals, the new package has to have the following design.

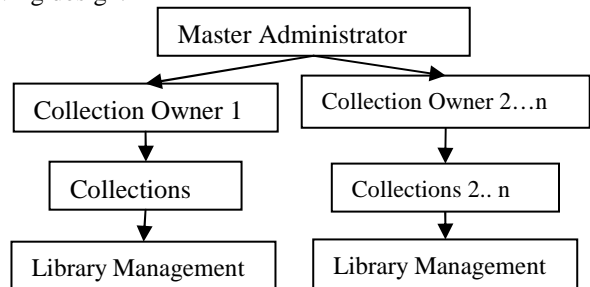


Figure 5: Conceptual Diagram of Home Libraries Operation

## 5. Implementation and Demonstration

To accomplish the design goals of Figure 5, the items below had to be done.

1. Modify the existing tables to accommodate multiple users (who will in turn manage their own collections)
2. Communicate with Amazon Web Service to get the desired product data.
3. Extensive modification of the open source code base to incorporate the design goals.
4. Modify the look and feel of the original product.

### 5.1. Database Tables

To enable multiple users to manage their own

collections (or otherwise known as catalogue) simultaneously, a new table was created. This table was called **newCatalog**. This table consist of two columns, a **cID** column (stores the catalogue ID) and **cName** (stores the catalogue name). Three other existing tables were modified, they were **biblio**, **member** and **staff tables**. In each of these tables, a new column, **cID**, was added to store the catalogue ID. This new column would be used in each query to ascertain whether or not a particular item (labelled *bibliography*) belongs to a particular collection. This way, new items could be added to a particular collection and only users who are restricted to access a particular collection / catalogue could retrieve information belonging to that particular collection.

## 5.2. Amazon Web Service

A third party module, that was developed by *Ritteh*, which can be found at the following url: <http://www.boardingdirectory.com/other/locsr/> was modified. The original module worked by using the *ItemSearch* operation. This operation requires a search Index to be specified (Books, DVDs, Software, etc) and at least one of several defined parameters [Amazon API], which would be the search phrase. After specifying the search phrase (which could be title/author keywords etc), the information would be returned. The downside to this mechanism is that several products related to the search phrase would be returned, most of which would not be relevant to the actual product that you want listed in your collection. For example, if one were to search the database using the search phrase 'PHP', one would get several results pertaining to the subject matter.

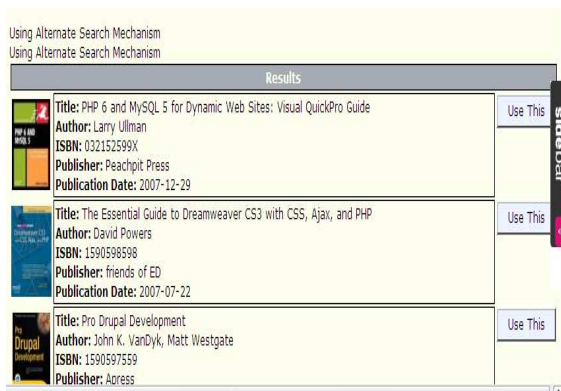


Figure 6: Returned Result if Original Module was used

As evident from the screenshot above, a lot of unnecessary products are displayed. If some 400 products match your search phrase, is it prudent to go through each one of them until you reach the entry that matches your product? To address this, the code was modified to also

use the *ItemLookup* operation. Using this operation, music (various media) and various other types of products (DVD, Electronics, Toys, etc) could be added into an existing collection simply by searching for their identification number. The key benefit to this is by inputting one of three IDTypes (ISBN, UPC or ASIN) of a product that the user already owns, the said product would be easily catalogued into the user's collection. The altered code sends the following parameters as part of its request to Amazon.

- ID Type
  - UPC for music (CDs / DVDs)
  - ISBN for books
  - ASIN (Amazon Standard Identification Number) for other searches
- Locale (connects to Amazon.com database)
- ID Number (UPC/EAN/ISBN/ASIN numbers)
- ResponseGroup

The following screenshots shows the result of specifying an identification number (ISBN) and the result from the query to Amazon.

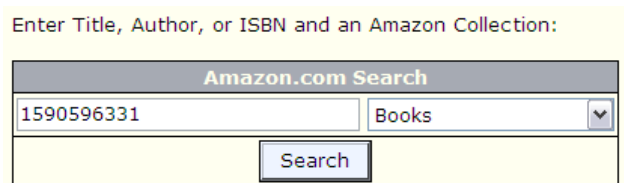


Figure 7: ISBN Query of a product

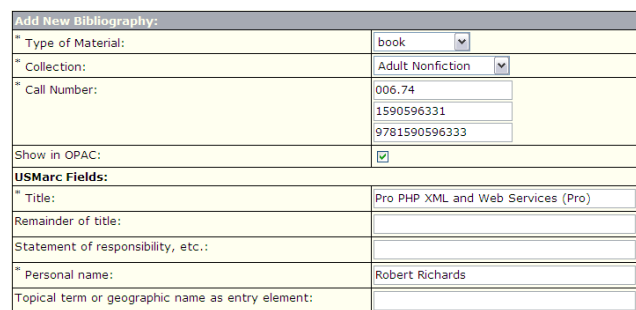


Figure 8: Result from ISBN Query from Amazon

The modified Amazon module provides two benefits.

1. General Look up module if a user isn't sure about the identification number associated with a product he/she already owns
2. A direct way to populate a collection by simply specifying the identification number of the product the user owns.

### 5.3. Modification of Base files

The original OpenBiblio package had hundreds of files structured in packages according to functions it was tasked to manage (i.e. all classes and scripts pertaining to managing the cataloging function were stored in a package appropriately named as *catalog*). The PHP scripts were also written in an object oriented manner, thus developers who were used to working in an object oriented environment would not have any difficulty in understanding the scripts.

The database query classes were modified to accommodate the changes in the database tables. Since a new column was added to **biblio**, **member** and **staff** tables, the appropriate functions in classes that queried the existing tables had to be modified to include the newly added column. The addition of the **newCatalog** entailed the modification of classes and scripts such that only users who owned that collection could have access to that particular catalogue. Likewise, members who wish to borrow a particular item can only do so provided he / she is a member of that collection.

### 5.4. Look and Feel

The original package had the design shown in Figure 1. After making the necessary modifications to achieve the aforementioned design goals, a new application emerged. This application was given the name, "Home Libraries". The look and feel for this application is shown in Figure 8.

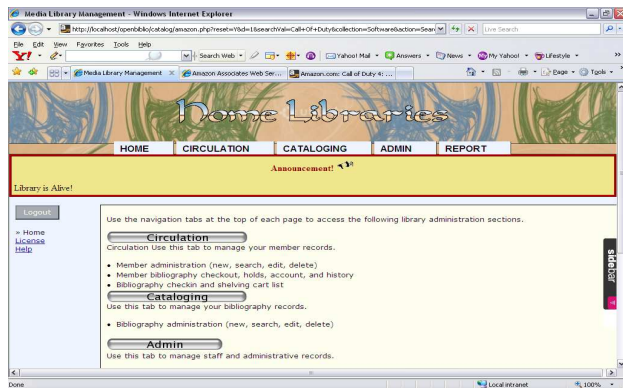


Figure 9: Home Libraries Look and Feel

The original web page was designed with liberal use of tables to structure the displayed content. Although not wrong, this made the code large. CSS table-less design was used as an experiment to try to reduce the front-end code. Initial attempts were successful as we managed to reduce the original tabbed interface (that shows the navigation links at the top of the page) from 144 lines of

nested html tables tags to a mere 8 lines of code using CSS to structure the display of list elements.

Figures 10a and 10b shows the original tabbed interface with associated code.

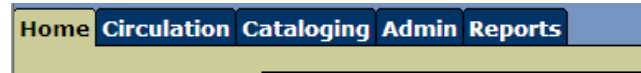


Figure 10a: Original Tabbed Interface

```
<?php if ($tab == "home") { ?>
<td bgcolor="<?php echo H(OBIB_ALT1_BG); ?>
<?php } else { ?>
<td bgcolor="#191970 <?php //echo H(OBIB_
<?php ) ?>

<td bgcolor="<?php echo H(OBIB_BORDER_COLOR)
<td bgcolor="<?php echo H(OBIB_TITLE_BG); ?>"
<td bgcolor="<?php echo H(OBIB_BORDER_COLOR)

<?php if ($tab == "circulation") { ?>
<td bgcolor="<?php echo H(OBIB_ALT1_BG); ?>
<?php } else { ?>
<td bgcolor="<?php echo H(OBIB_ALT2_BG); ?>
<?php ) ?>
```

Figure 10b: Original Tabbed Interface Code

Figures 11a and 11b shows the CSS based implementation of list to display the navigation tabs.

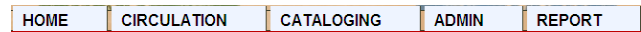


Figure 11a: CSS Based Navigation Tab

```
<center>
<ul id="navigationList">
<li id="navigation-home"><a href=" ../home/index.php">Home</a></li>
<li id="navigation-circ"><a href=" ../circ/index.php">Circulation</a></li>
<li id="navigation-catalog"><a href=" ../catalog/index.php">Cataloging</a></li>
<li id="navigation-admin"><a href=" ../admin/index.php">Admin</a></li>
<li id="navigation-report"><a href=" ../reports/index.php">Report</a></li>
</ul>
</center>

#navigationList{
margin: 0;
padding: 0;
border: 1px solid #42432d;
border-width: 1px 0;
background-image: url("../images/sample2.jpg");
}

#navigationList li{
display: inline;
margin: 0;
}
```

Figure 11b: HTML List and Associated CSS Code to structure the list

The original package has good internationalization support. Although English was the default package, third party developers have developed language packages that could be installed by simply dropping the language package into the appropriate folder location in OpenBiblio. The enhanced web application takes advantage of this internationalization support.

## 5.5. Demonstration Application

A live demonstration of the application could be tried at <http://jsuresh.net/openbiblio> using the **username:** test and **password:** testAdmin. At the demonstration web site, after logging in using the supplied account, the cataloging function could be tried. The user is invited to either enter in a ISBN or UPC number by using one of two methods; i) entering via keyboard or ii) entering through the usage of a bar code scanner. Multiple users could also be created from the test account to test the multi user – multi collection functionality.

## 6. Future Work

Other functionalities worth integrating into the existing developed web application would be:

1. Integration with Facebook so that users can share their collections with friends. Integration with Facebook would also open up the possibility of allowing members to borrow a book from facebook itself.
2. Add more databases to search from, so that there are multiple sources to populate the collection
3. Integration with other Web 2.0 tools such as Google Maps to allow collection owners to track the location of their books or to provide visitor demographics.
4. Integration with other popular social networking sites (Twitter, Bebo, etc) so that members can interface with the community of their following.

## 7. Conclusion

The developed web application, christened as “Home Libraries”, fulfills the following goals.

1. Support multiple user collections
2. Pure web based management of collections
3. Populating an existing collection by querying Amazon’s database.
4. Seamless installation
5. Easy to use interface
6. Internationalization support

To fulfill the stated goals, existing tables were modified and a new table was created (details mentioned in Section 5.3). Numerous PHP classes and scripts were modified to realize the multiple user – multiple collection feature as shown in Figure 4. The Amazon Look up module was enhanced so that users could populate the collection by inputting an identification number directly or if the user wishes, to search for a particular product using a search phrase.

The web based component provides update immunity as users don’t have to worry about their applications crashing after upgrading their software or Operating System. Finally, this application was built out of open source components, thereby significantly reducing development cost and cost savings could be passed to customers.

We believe a “Home Libraries” system would be most useful for families as each member of the family would be able to keep track of their collections independently. Additionally, advantages mentioned in preceding paragraphs would be obtained by collection owners.

## 8. Reference

- [1] Amazon *Amazon Associates Web Service* [On - line] <http://aws.amazon.com> , Seattle, WA, 2008.
- [2] Collectorz.com, *Book Collector* [On-line], <http://www.collectorz.com/book/>, Amsterdam, Netherlands, 2008.
- [3] Delicious Monster. *Delicious Library* [On line], <http://www.delicious-monster.com/>, Seattle, WA, 2008
- [4] Intermedia Software *Helium Music Manager* [On-line] <http://www.helium-music-manager.com/>, Sweden, 2008
- [5] Firetongue Software, *Orange CD* [On-line] <http://www.firetongue.com/>, 2008
- [6] M.Stetson, D.Stevens, *OpenBiblio* [On-line] <http://sourceforge.net/projects/obiblio>, 2008
- [7] Readerware *Readerware* [On-line] <http://www.readerware.com>, Clearlake Oaks, CA, 2008
- [8] R. J. V. D. Waals, *Data Crow 3* [On-line] <http://datacrow.net/>, 2008