Constructing a Searchable Product Ingredient Database With a Website Interface

Avidan Ackerson
avidan@seas.upenn.edu

Faculty Advisor: Dr. Zachary Ives

April 18, 2008
**Section 1: Abstract**

For thousands of visually impaired people, it is impossible to read the ingredient list on articles of food. The text is frequently too small or distorted to be read without additional assistance. With allergies and other health concerns always prevalent, a system must be created to allow an easy way for the visually impaired to know what they’re eating. The ideal solution is a barcode scanner that can be programmed with medical conditions and can reference ingredient lists so users can be alerted to contraindications. However, no official database of barcodes to ingredients exists. (T. Ware, personal communication, September 7, 2007).

This project aims to create a database by using a web crawler to parse various grocery store websites. The crawler will remove the barcode number, product name, and the ingredient list and populate the database. To implement this database, a website will be developed that allows the public to search the database by UPC or product name. Furthermore, the website will contain a link to download the database so anyone can manipulate the data as they see fit.

**Section 2: Related Work**

2.1 – Existing Barcode Databases

Barcodes can be divided into two sets of digits that act as identifiers. The first digits of a barcode are unique to a manufacturer and are currently assigned by GS1, a worldwide organization that sets the standards for barcodes in many different countries. GS1 has a searchable database online called GEPIR (http://www.ucc-gepir.net) that identifies the manufacturer of a product based on the UPC. However, GS1 does not
maintain a database of ingredient lists or even product names associated with the barcodes (T. Ware, personal communications).

There currently exists a number of websites that maintain their own databases of barcodes and the associated product names. The most extensive is the UPC Database (www.upcdatabase.com), which currently contains nearly one million entries. Manufacturers are identified through “a couple of different GS1 sites around the globe that expose this information to the public.” (UPC Database FAQ, www.upcdatabase.com/docs/faq.html, Accessed on April 13, 2008). The product names are added manually by user submissions. The database is searchable by the UPC or a string. Additionally, the database is downloadable in a number of formats. However, this database also does not provide ingredient information associated with the products.

2.2 – Applications of a “Barcode to Ingredient List” Database

While no public database currently exists that maps barcodes to ingredients, many people and companies have developed devices or applications that would make use of such information. At an IEEE conference, Robert Damper and other faculty from the University of Southampton presented a paper on "A Barcode-Scanner Aid For Visually Impaired People." (Damper, Garner, Jordan, Rahman, and Saunders, 1996) Their system scanned barcodes and used speech synthesis to read out the name of the product. A prototype was tested, but the project was eventually abandoned due to complications of database creation. If the store would provide the database, then the scanner would be limited to that store alone. If the database were self-maintained, it would be technically challenging to constantly update. Damper's paper assumed that another organization,
such as the Royal National Institute for the Blind, would be responsible for the database thus removing the burden from the visually impaired. (R. Damper, personal communication, September 24, 2007)

In 1998, Timothy Glyn Hankins filed Patent No. 6,375,077, titled “System for advising a user when selecting a product.” This patent describes a device that scans the barcode of a product and references a database that contains the product’s ingredients. The system would then advise the user of potential issues that may arise from consumption or use of the product. The user would be able to program the system’s alert function by providing his own preferences for the system to refer against. It is not explained explicitly in the patent application how the database of ingredient lists would be populated, but it is implied that it would be the responsibility of the specific grocery stores to maintain their own lists.

**Section 3: Technical Approach**

The database is a MySQL table with three primary columns: upc (VARCHAR(13)), ingredients (TEXT), and productname (TEXT). When the database was constructed, a separate table was used for each website, then all the tables were combined into one overall table using the UNION option when executing the CREATE TABLE…SELECT query. The original tables were kept for further use as well as backups as necessary.

To populate the tables, four websites were selected that were designed by the same developer and therefore all had the same structure. This allowed for the code to be reused for each site with minimal changes. The four sites were NetGrocer
(http://shop.netgrocer.com), ShopRite of Garden State Pavillion
Highlands (shop.mywebgrocer.com/shop.aspx?strid=CA8D846), and Lowes Food
(shop.mywebgrocer.com/shop.aspx?strid=ECF3378). To navigate through the websites,
an open-source library called HtmlUnit (htmlunit.sourceforge.net) was used. This library
simulates a web browser and has the ability to locate and ‘click’ on links on websites.

The beginning of the program opens a connection to the homepage of the website
that will be parsed as well as a connection to the MySQL server using the JDBC. An
ArrayList<String> named onceclicked is initialized, which will maintain the names of
categories that have already been explored. The top level menu links are extracted using
the HtmlUnit function getAnchors() and placed into an ArrayList of type HtmlAnchor, an
object type specific to the HtmlUnit library. Then, the processLinks() method is called.
The two arguments for processLinks() are the new list of anchors to be processed and the
previous set of anchors. In the first set of anchors, the previous list is defined as null.

The processLinks() method is surrounded by a try/catch statement. In the catch
segment of the block, the time the exception was caught is displayed before the exception
is thrown. This is due to the fact that the majority of the exceptions related to dropping
the web connection, and was solely done to determine if there was a consistent time limit
for the connection opened through HtmlUnit.

The processLinks() method is recursive, as it calls itself with each sublevel of
menu options. Since the HtmlUnit library is “clicking” on the links in the menu, the
‘display’ shows not only the new level of categories but also all of the previous set of
categories as well. Therefore, the first step is remove from the new set of anchors
(defined as `ArrayList<HtmlAnchor> newanchors`) all of the old anchors (passed in through the argument `ListArray<HtmlAnchor> oldanchors`). Then, there are a series of if statements to remove top level categories that may have been missed, categories in `oncelicked` that have already been explored, and the link that directs to registration. After going through these statements, the only anchors remaining in `newanchors` are a list of links that have yet to be explored or processed for the database.

If any of the links in the `newanchors` contain the string “,1,” then clicking on the link does not expand to any sublevel of the menu and instead directs to a page where individual products are listed (referred to as a listing page from here on). In this case, the page is cast to the type `HtmlPage`, which is necessary for the HtmlUnit operations. The method `processPage()` is called on the listing page. All other links generate a new set of links which is fed back into a new call of `processLinks()`.

The `processPage()` method takes the listing page and extracts the links on the page and places them as an `ArrayList` of `HtmlAnchors`. It then removes any banner ad links, specialty notifications, and links to the previous set of 10 products. The last requirement prevents an infinite loop of cycling back and forth among the same products. The remaining `ArrayList` of `HtmlAnchors` represent the individual products as well as the “Next” link on the bottom of the page. It also initializes a new Tidy object, which exists in the JTidy library.

To process each individual product, an `HtmlPage` object is generated by clicking on the product’s individual link. A temporary file is opened and the XML code of the page is written to that file. A FileInputStream is created and linked to the file, and is then sent to the JTidy `parseDOM()` method, which returns a DOM Document. The Document
is sent to the `printTree()` method, which recursively goes through all the children and attributes of the DOM tree and looks for the relevant information. The method also is passed a String array. If the UPC is found, it is placed in position 0 of the array, the product name is placed in position 1, and the ingredients are placed in position 2. Upon reaching the end of the Document tree, the array is returned back to the `processPage()` method. The values of the array are placed into a MySQL INSERT command that is contained in a PreparedStatement, which is then executed. Upon conclusion of the insert, the temporary files are deleted and the various connections are closed.

After all the sites are completed, there exists one table for each site. The tables are UNIONed into a new table, which removes the majority of the duplicate entries. For the duplicate UPCs that are not removed through UNION due to trailing characters, the DuplicateRemoval class removes them. An AUTO_INCREMENT column was added to the final table. Then, using the JDBC and PreparedStatements, a list of all UPCs with occurrence greater than 1 was generated. For all of those entries, a PreparedStatement calls up the AUTO_INCREMENT value and deletes the records with an AUTO_INCREMENT value not equal to the first record’s value. This ensures that there exists only one record for each UPC.

As the final stage of this project, a website was developed that allows users to search the database by UPC or product name. One page on the site, search.html, contains two textboxes: one for the UPC and another for the product name. If the UPC is the search term, the value is passed to the products.php page. If the search term is by product name, products2.php is the target. Both PHP pages open a connection to the database, which is currently stored on the fling.seas.upenn.edu server, and executes the appropriate
SELECT query on the passed in term. The result generated is displayed in table format using code based on a skeleton program in the PHP manual. The table also displays which of the four parsed websites contains the selected products. This is also accomplished with a SELECT query, but on the individual tables (products, products2, products3, and products4) as opposed to the unioned values. The website also has a page, download.html, where there is a link to the dump file generated by the table and is available to be downloaded for personal use.

Section 3.1: Challenges Faced

During the course of this project, there were a lot of challenges that needed to be overcome. The most significant challenge was gaining access to the websites in the first place to parse them. The initial plan, which was to use a URLInputStream, returned the equivalent of the 404 page. Subsequent attempts at InputStreams met the same complication, as it appears the websites are configured to return the 404 page if an external search engine tries to gain access. Some format of an InputStream was necessary because the JTidy method `parseDOM()` requires a stream for processing.

Attention was then turned to alternative methods of reaching the website. One idea was to use a Java library that simulated a web browser. The logic behind this was the assumption that if the website registered the incoming connection as a legitimate web browser, the 404 page would not be returned. The primary candidates for such a library were all open-source libraries that could be found on Sourceforge.net.

However, while this was a good idea in theory, finding a usable library was a much harder task. Nearly a month was spent finding a library that would work. Every
candidate when implemented caused the 404 page to be generated. Finally, the most promising candidate, HtmlUnit was tested. This library used a different underlying connection system than a basic URLInputStream. However, it too was problematic. Finally, a solution was found: HtmlUnit worked when the simulated browser was Internet Explorer instead of Firefox. It is not clear at this time why this is the case.

Another challenge was extracting the data from the sites and recursing through the menus. The websites use multiple frames and the menu links were JavaScript functions instead of the standard href HTML anchors. This meant that even if the links were extracted from the page’s code, subsequent connections could not be generated because the JavaScript function could not be executed. Furthermore, even if a direct link could be determined, the function returned the contents of a different frame instead of the updated version of the menu.

Once again, the HtmlUnit library was able to remedy the problem. The library contains support for links that are JavaScript function calls, and the `click()` method of the HtmlAnchor generates all of the changes to the website necessary for the next level of processing. The library also has frame support in objects called FrameWindows. This allowed for easy switching between the product frame and the menu frame. Thanks to the library, data extraction was simplified, although it did increase the amount of memory necessary for the various data structures.

However, at one point, the HtmlUnit library caused a problem that needed a complete workaround. Towards the end of this project, there was a new build released of HtmlUnit that was designed to have better support for JavaScript. However, the build had an updated parser that rendered apostrophes as actual apostrophes, while the previous
parser replaced the apostrophe as &apos;. This meant that when the insert query was executed, the apostrophes in product names were read as the demarcation of a field. This created a lot of exceptions because of incorrect format. In order to remedy this situation, PreparedStatements were used because they overcome issue of quotation marks due to their unique system of setting the variables. Eventually, the previous build of HtmlUnit was reinstalled because the parsing time of the new build was nearly six times more.

Section 4: Conclusions

There are a number of things I would have done differently about this project in retrospect. For one, I would have made the program multithreaded. The average runtime to process an entire website was 7 hours. Unfortunately, this meant that if there were any difficulties in the processing, it would take another 7 hours to rerun the code. Furthermore, an extended process time meant that the connection to the website occasionally times out. In theory, multithreading would have cut the time in half, thus reducing the time lost for reprocessing as well as reducing the odds of a timeout situation.

I also would have liked to implement a better system for avoiding duplicate entries into the tables. Removing duplicates became a hassle towards the end because trailing characters meant the UNIQUE option in the SELECT query returned what would be duplicate entries by visual inspection, but were considered distinct by MySQL. This is why the DuplicateRemoval class was created, although that took a fair amount of time to develop correctly as well.

Overall, I learned a lot through doing this project. I gained a great many skills that I previously lacked. For one, I now have a much better grasp of MySQL, since I never
used it before this project. Now, I know a lot more about the various data types of table entries, and the basic commands that can be executed with regards to the individual entries as well as the tables as a whole.

I also learned a lot about general web design. To successfully extract all of the information necessary, I had to learn a lot more about the DOM format of websites. For the longest time, I couldn’t locate the ingredient lists or the UPC in the DOM tree. It took nearly a month to recognize that the information was not stored in individual nodes, but rather in attributes stored within the nodes, and it required a separate call. This is why I later wrote the `printTree()` method: to get a better visual of the DOM tree.

There were two goals I set for myself with my senior design project: to gain skills I otherwise wouldn’t have the time to take classes for, and to produce something that would be of practical benefit to the public. I can say with certainty that both of these goals have been met.
Annotated Bibliography


The author is a professor at the Austin Community College and holds an MSEE degree. This tutorial introduces the basic format and syntax of the JDBC and how it interacts with the MySQL server. Based on this tutorial, I was able to write the connection syntax that can be found at the beginning of the NetGrocer10 class.


This is the API for the open source library HtmlUnit. There are multiple authors, however the primary author is the representative from Gargoyle Software, who is assisting with much of the programming. This API was essential for developing the code that utilized HtmlUnit’s unique objects and classes.


The authors are professors at the University of Southampton. Their paper proposes a product that scans barcodes and reads out the ingredients. The target audience of the product is the visually impaired. Damper’s paper concludes that the primary logistical issue is the creation of the database, as it is not clear if the responsibility is that of the grocery store or something that individual users would need to download. Such a product would find this database incredibly useful.


This patent was initially issued in Great Britain in 1996, and was subsequently applied for in the United States two years later. The proposed product has a software program that allows users to set what ingredients or conditions should be flagged if present in a product. The check would be made by scanning a barcode and checking the terms against a product ingredient database. This is another example of a product that would find this project’s database useful.


The author holds a Ph.D in computer science from the Graz Technical University. This book is an in depth instructional guide to MySQL. This includes sections on basic PHP functions, the JDBC, and PHP commands. It was an invaluable resource to learn how to set up the MySQL database and tables and to insert data when building the database.

This tutorial was written by a professor at MCC and covers all the basic properties of HTML coding. The tutorial provides examples of different sites and contains lessons that build off each other. I used this tutorial to code the basic structure of the website that implements the database.


This is the official manual for the PHP: Hypertext Processor language and was written by many contributors. This specific section outlined the primary functions necessary to have a PHP page interact with the MySQL database. The manual also provided the skeleton code for my website’s PHP pages, especially the connection syntax and how to display MySQL calls in table format.