

CSE 454

Nicholas Castorina

Mark Jordan

Ken Inoue

**Goals**

Students are typically short on time and money. We purposed to create a website that streamlines the process of buying textbooks for UW classes.

For the frontend, this meant having a slick and intuitive UI with three parts:

1. **Selecting classes:** The first page allows users to select their classes, whether new or used books are desired, and whether or not they wish to purchase the optional texts. They would then submit the results.
2. **Book verification and deselecting:** This second page would be an intermediate page, containing all of the textbooks and their information (title, ISBN, and image) as well as a checkbox indicating whether the user wanted each book to be in placed in his or her cart. This would allow users to verify their books as well as deselect UW-only books or books that they already own. The user would then submit the results.
3. **Choosing the supplier:** This third and last page would contain a table of possible providers (Amazon all merchants, Amazon, Half.com, eBay, Abe Books, etc.) as well as prices with and without standard shipping. The user then could select the vendor and be redirected to a shopping cart with all the needed books. A single shopping cart from one source was the backbone of our site, because other websites already find the cheapest individual books from individual sites.

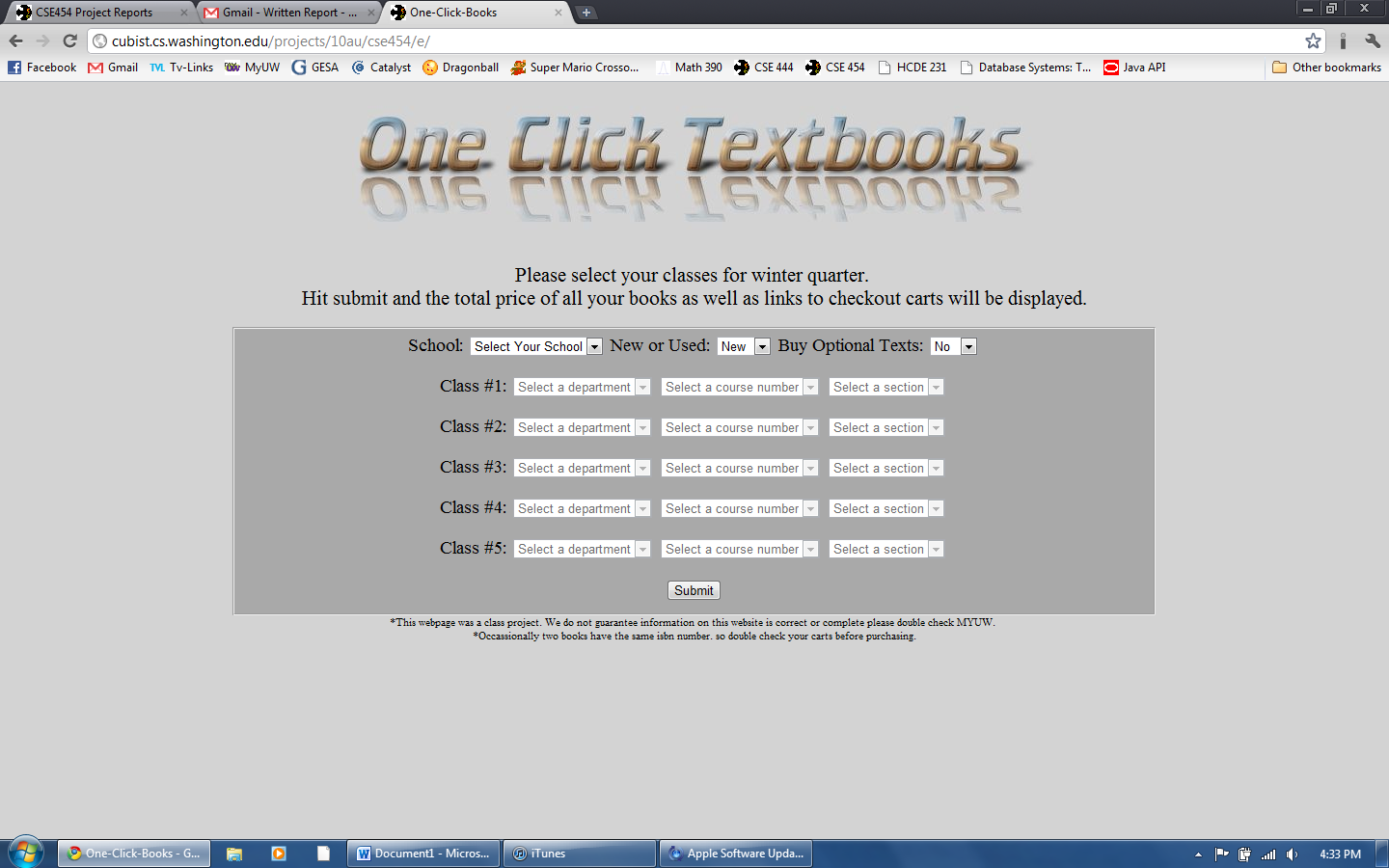
The backend requirements included a fast algorithm for querying both our database of books and the provider websites using their open APIs, and finding the cheapest books to purchase. Also having an up-to-date database of all of the classes and their corresponding books is a must.

**Typical Usage**

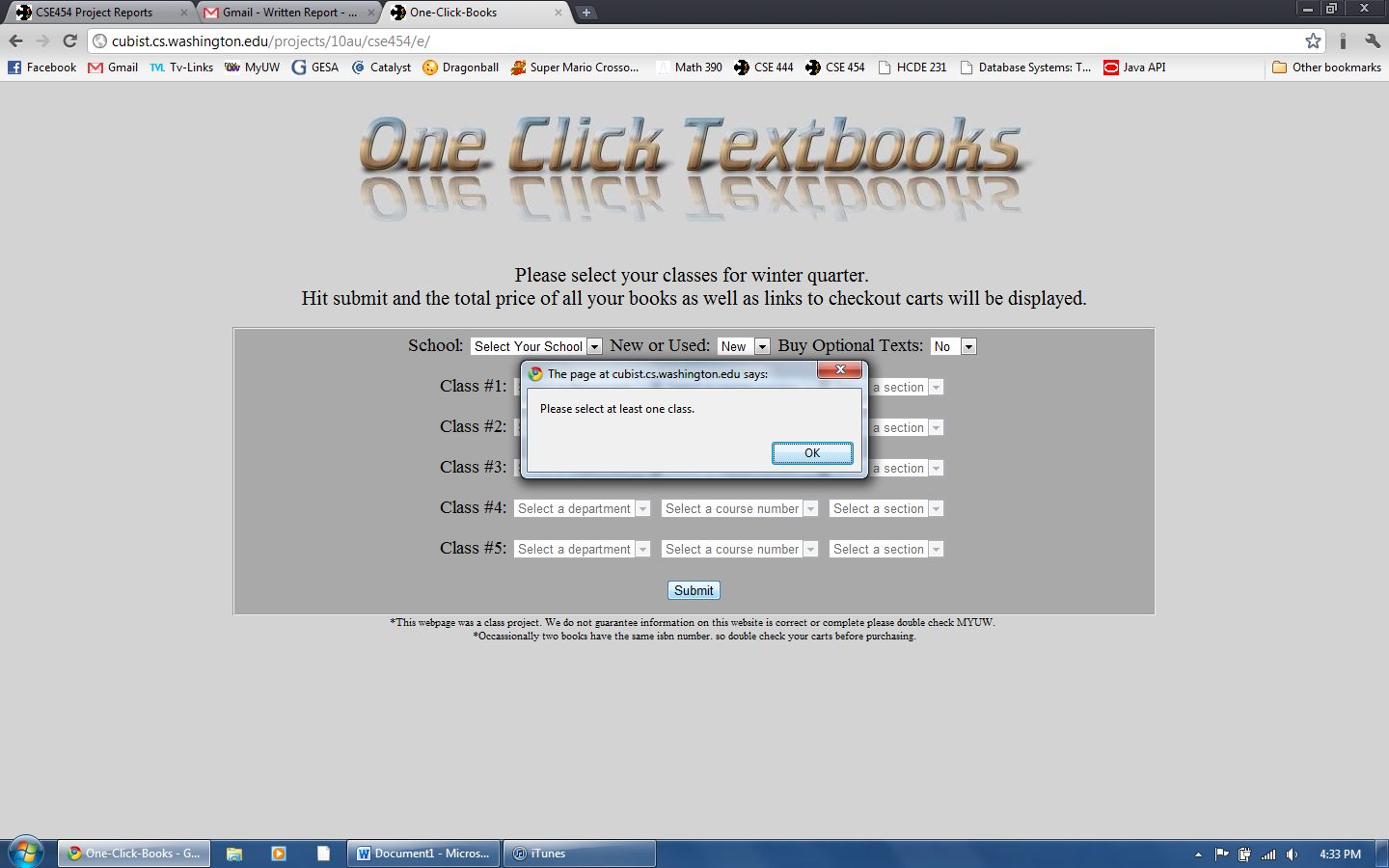
The user begins by going to [www.oneclicktextbooks.com](http://www.oneclicktextbooks.com) or <http://cubist.cs.washington.edu/projects/10au/cse454/e/>. At that point, he/she reaches the homepage (Figure 1) and selects his/her classes. If the user attempts to hit *submit* prematurely (before any classes are selected), an alert will pop up and prevent them from proceeding (Figure 2). Once a school is selected, the “departments” field is populated for all classes. This process continues for the selection of “course number” and “section” fields. The dropdown menus prevent improper input (Figure 3).

After hitting *submit*, the user is redirected to a results page (Figure 4) which contains a table of costs of different providers for all of the books corresponding to their classes. If a source does not contain all of the books, the price appears as a hyphen (-) and the corresponding source in the source column appears as just text as opposed to a link. At this point, the user has the ability to click on any of the provided source links, and can have one of a select few outcomes:

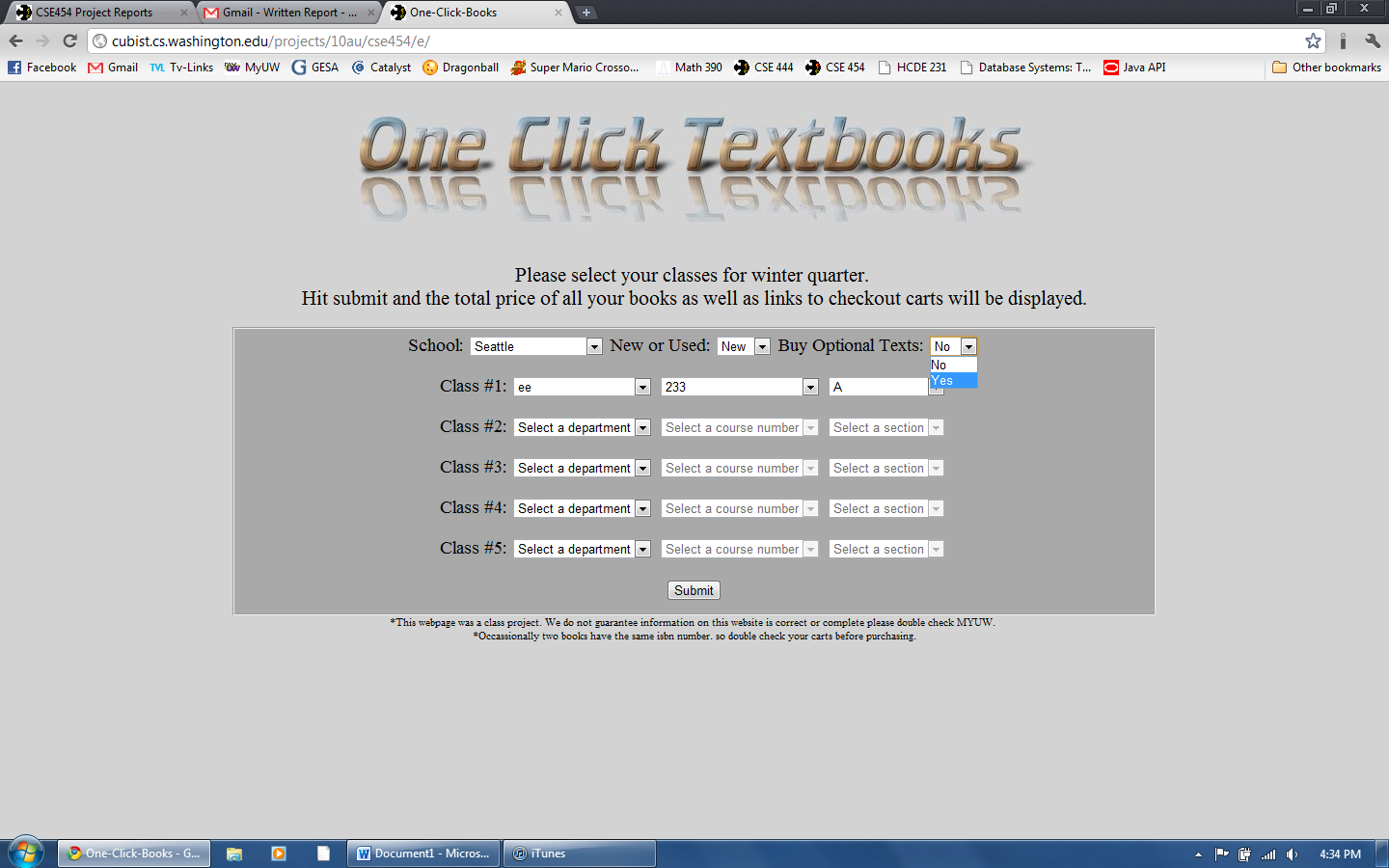
* If the user clicks on either of the Amazon links, he/she is redirected to the cart having all of his/her books.
* If the user selects the Half.com page or the eBay page, the user is redirected to one of two pages, with directions on how to proceed with their purchases (Figures 5 and 7). [Note: Before 12/15/2010, the Half.com link redirected the user to a page that used iframes and JavaScript to build a cart for the user (Figure 6). As of 12/15/2010, the iframes method was disabled by Half.com. Opening Half.com in an iframes became an automatic redirect to the homepage of their website. As a result, we now link to a different page, which allows the user to manually build his/her cart with popup links.]



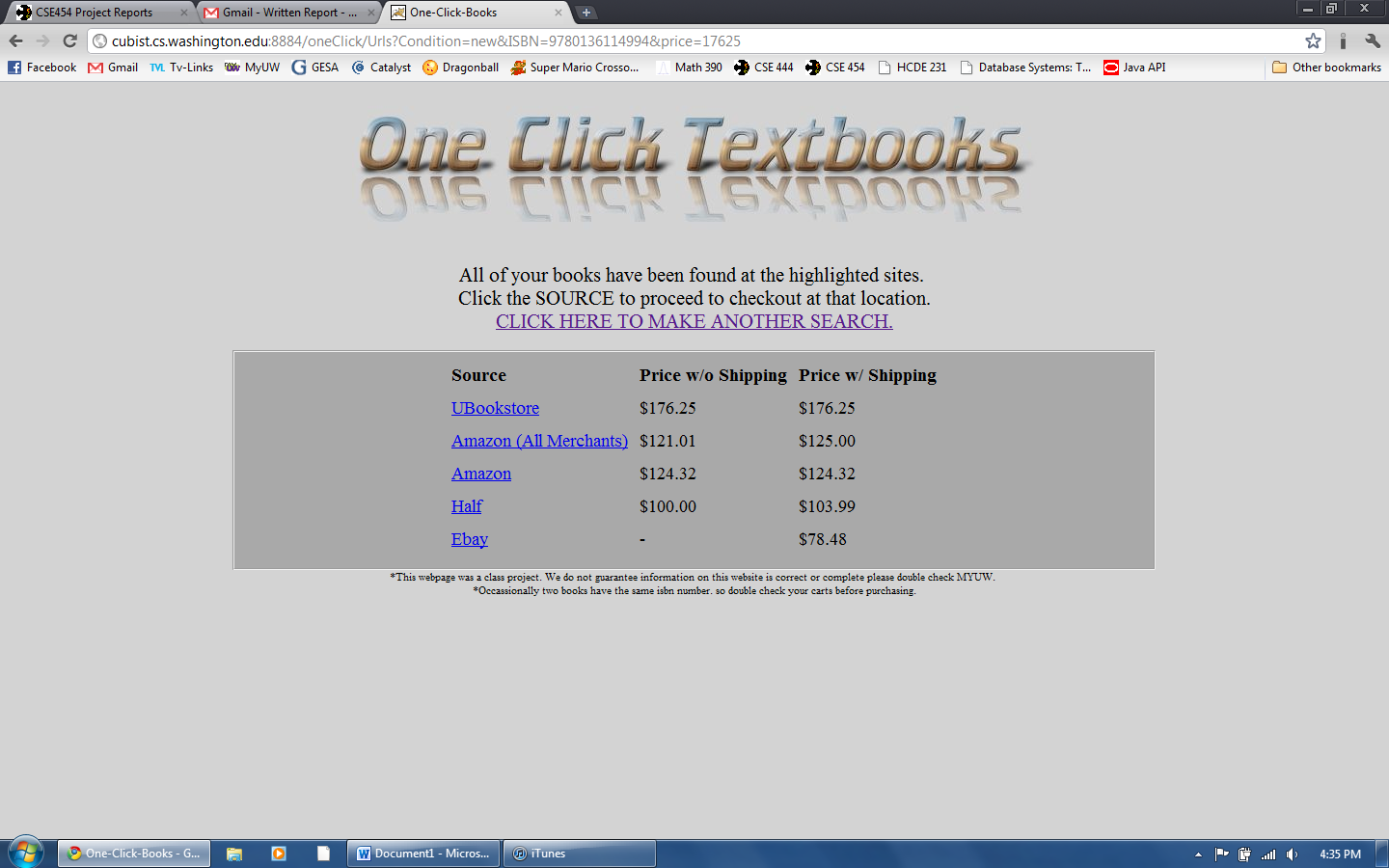
**Figure 1: Homepage**



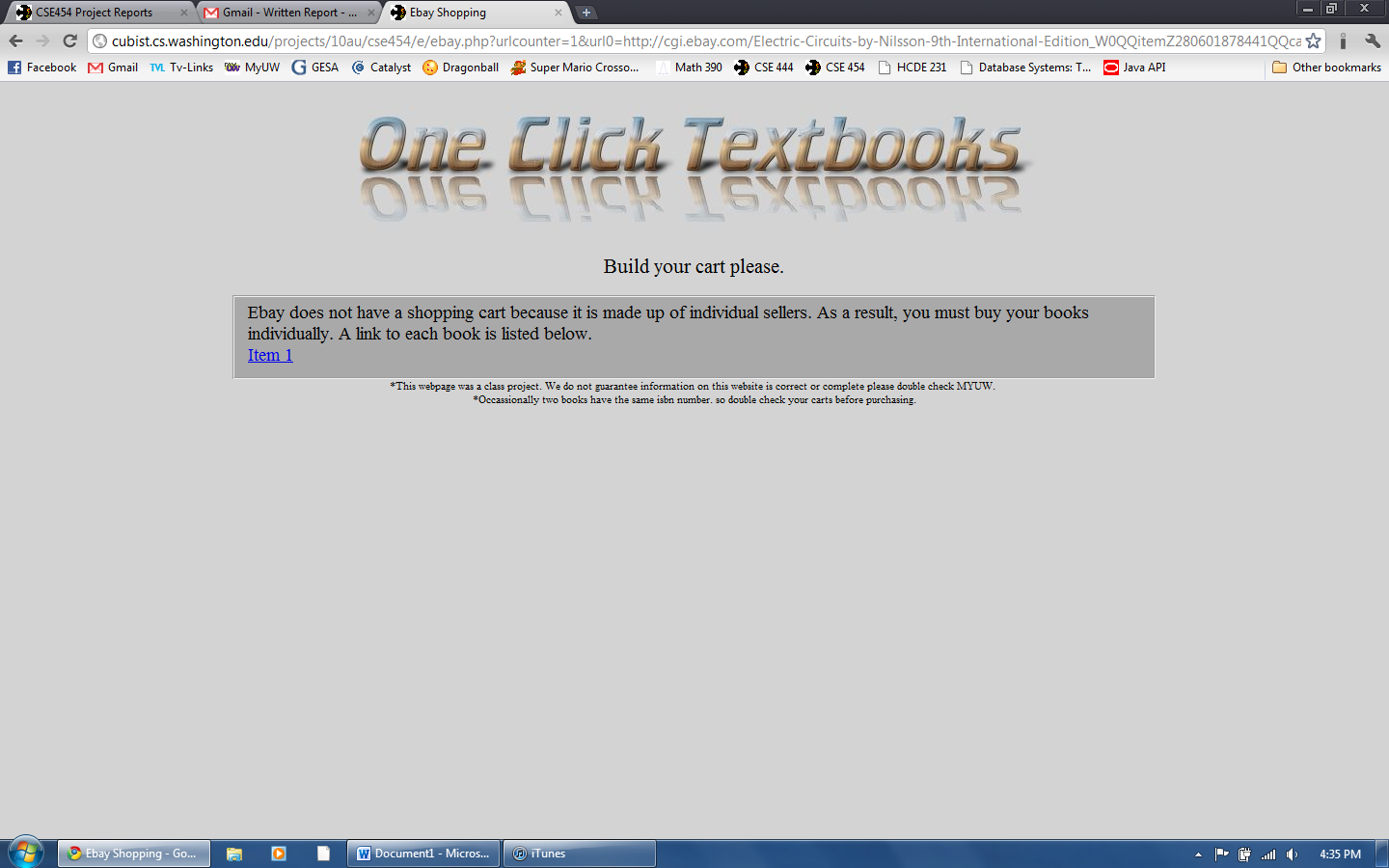
**Figure 2: Homepage alert.**



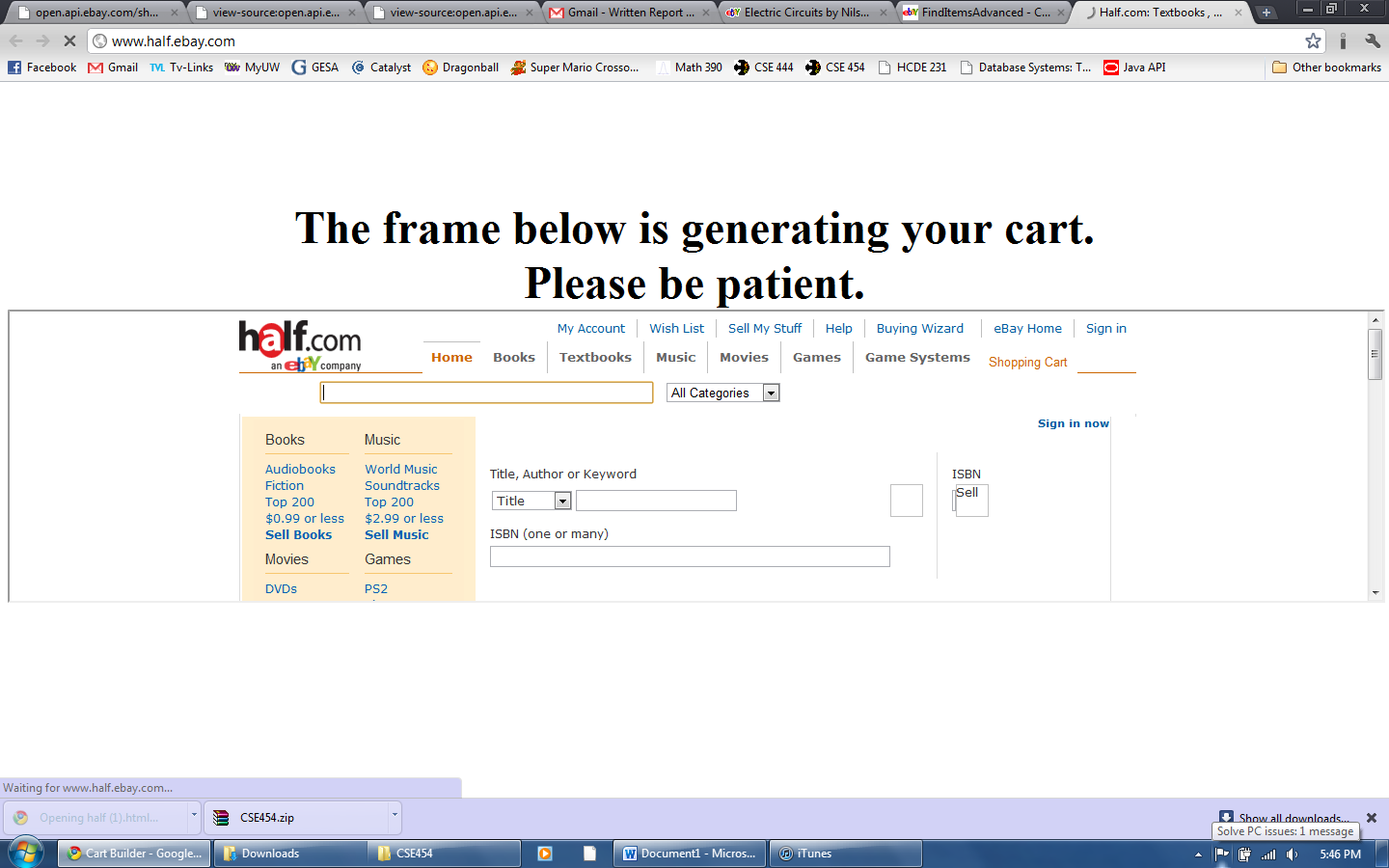
**Figure 3: Choosing classes and book options**



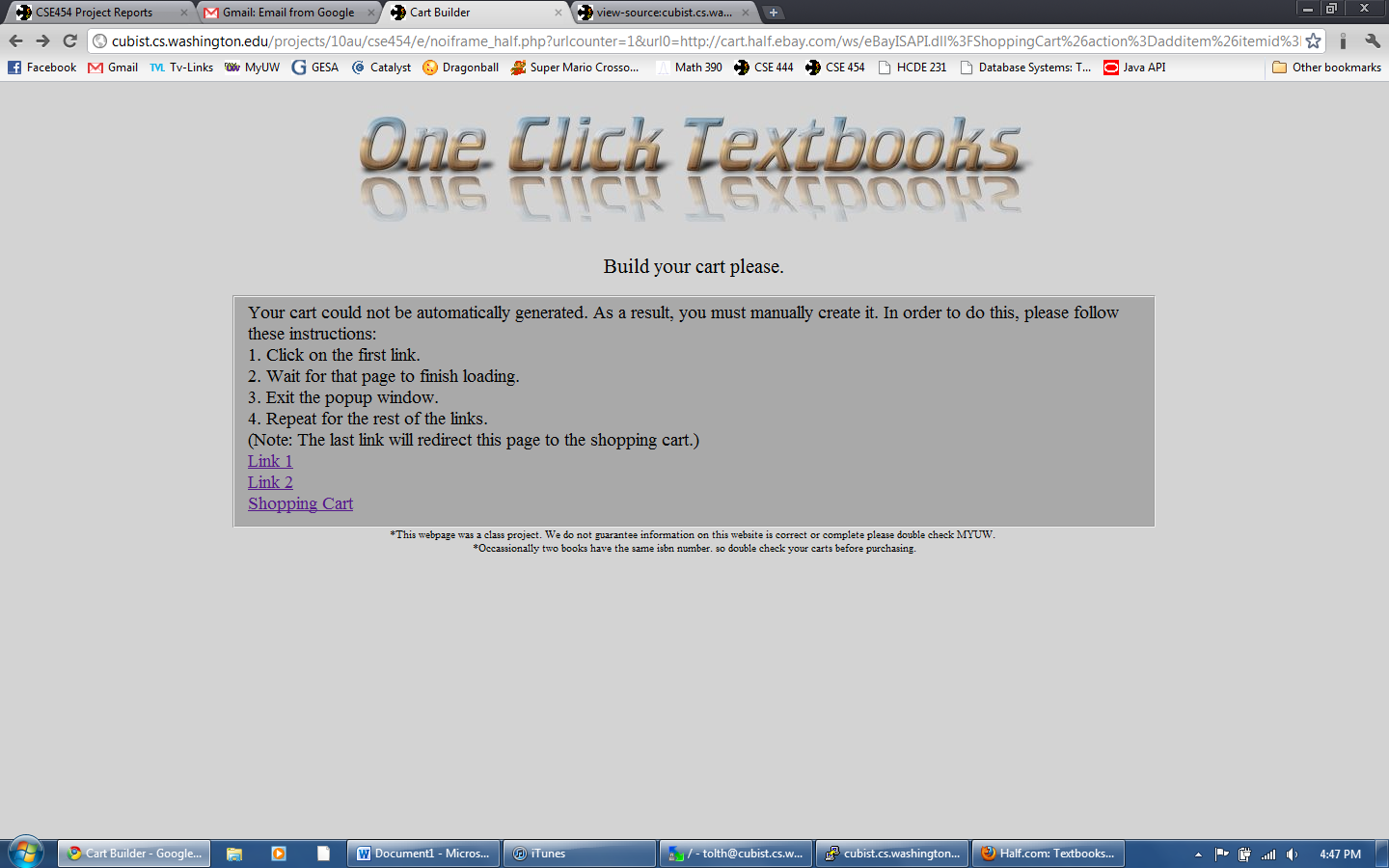
**Figure 4: Results page**



**Figure 5: Purchasing using eBay**



**Figure 6: Half.com redirect**



**Figure 7: Half.com without redirect.**

**Design and Algorithmic Choices**

Our system can be divided into two major sections: the frontend and the backend.

*The Frontend:*

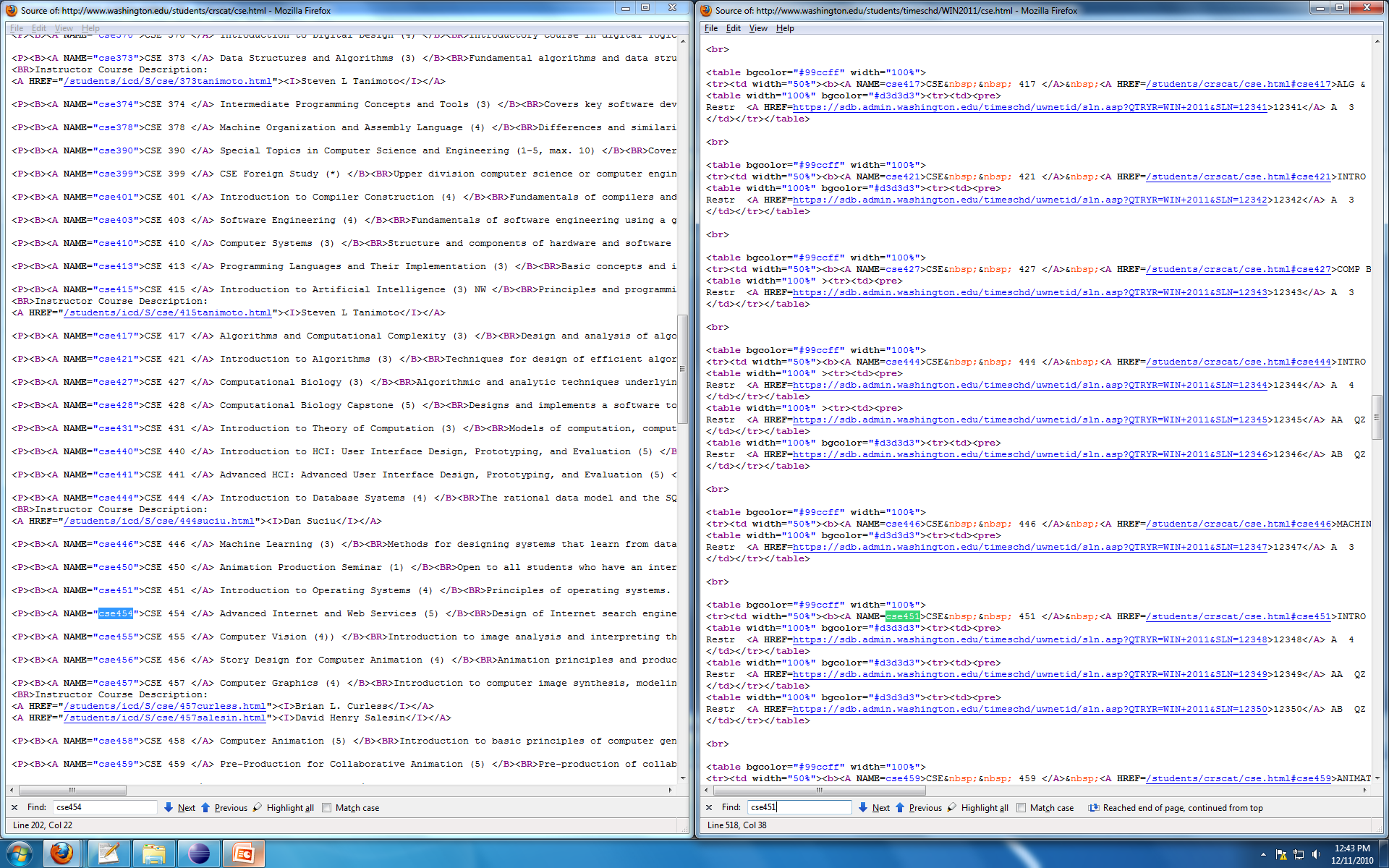
The front end was built using a variety of technologies. HTML and CSS were used to build the pages’ general structure. JavaScript allowed the user to interact with the page; the JavaScript makes an Ajax request to the database in order to fill in the next dropdown menu, as described previously. We chose to use the dropdown menu method to prevent erroneous input from the user. PHP was then used upon submitting the classes in order to retrieve the corresponding book ISBNs and send them to the backend to retrieve the results. [Note: We were unable to add the intermediate book selection and verification page because of time constraints.]

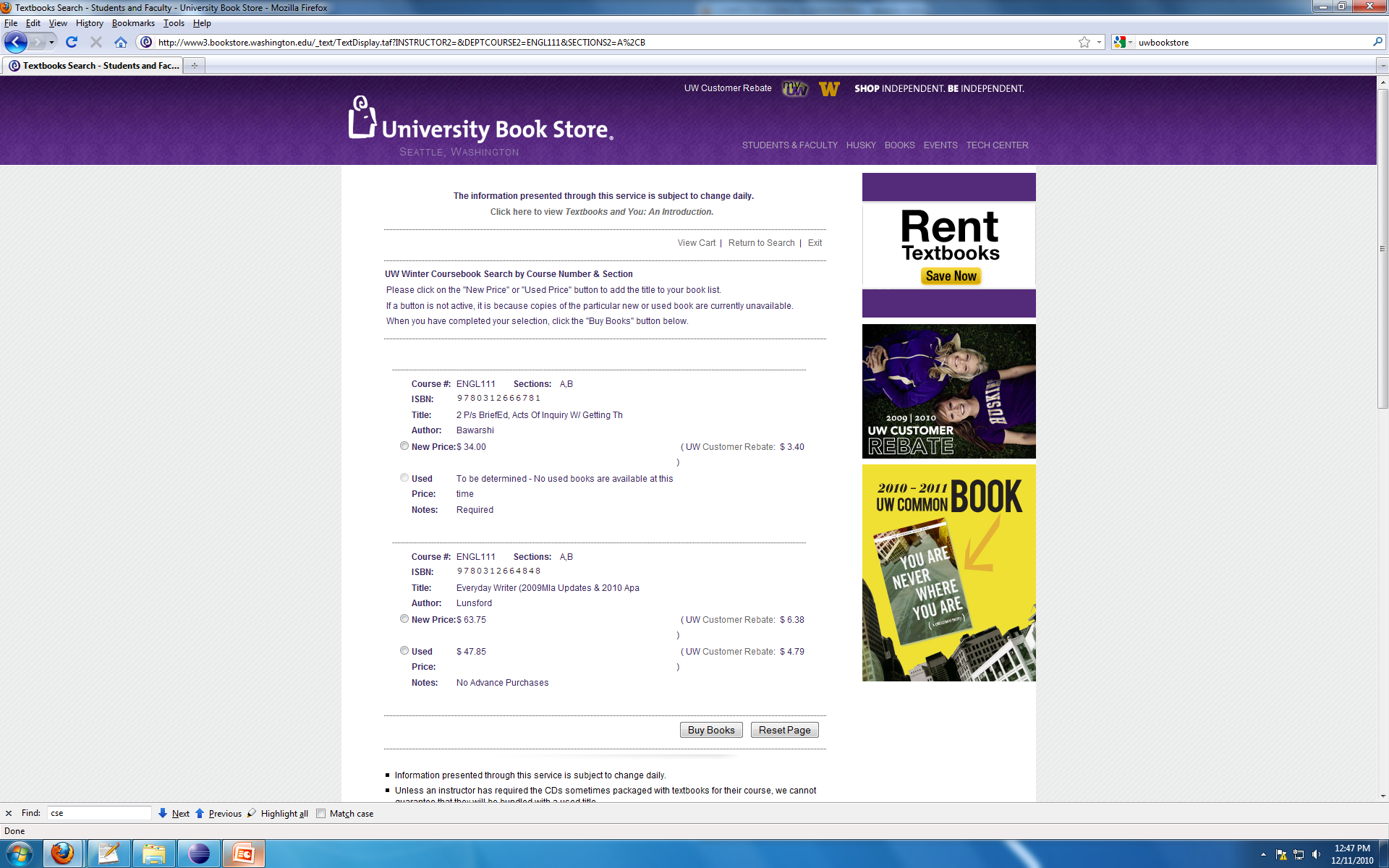
*The Backend:*

One Click Textbooks was built around a database of the books created by crawling the MyUW class schedule and the UW Bookstore. After retrieving information from our database, it queried the other provider websites. We could have queried the UW Bookstore in real time, but we decided it would be preferable to have a table with the values pre-fetched to speed up queries. Also, in the event that we scaled up the project, we would not overload the UW Bookstore's servers.

In addition, the UW Bookstore’s database is very close to static; as is the MyUW class schedule; they change very infrequently, typically at the beginning of the quarter. Thus, crawling their site to populate our database seemed like a feasible option. [Note: We still need to re-crawl their website in order to update our database for the latest changes.] The crawler is currently designed to be able to crawl multiple times each quarter and only update the table rather than just add additional rows.

The crawler starts at either the course catalog or time schedule listing of all the departments, for flexibility in case the UW site format changes. Classes are stripped from each departmental webpage and stored in memory. A post request is then constructed for each class, and sent to the UW Bookstore. All the return links are parsed for the required book ISBN, new price, and used price. This information is stored into three tables: classes, ClassToBookMap, and book. We also added tables for Bothell and Tacoma campuses, since those websites were formatted similarly.

  
**Figure 8: Sample Departmental Course Listing**

  
**Figure 9: Sample Class Section Bookstore Page**

We did not crawl Amazon and other database’s websites for a variety of reason:

* **Their sites are very large.** Thus, the impact of our site making a query is very small compared to the amount of traffic that their websites generally maintain.
* **Their sites provide open APIs.** Thus, if we had crawled the site, we would have had to edit the crawler every time a change is made. Since we utilize their open API, our website is not impacted by this scenario.
* **Changes in availability and price occur often.** If we did not fetch their site in real-time, chances are that our availability and prices would be radically off, resulting from their large amount of typical traffic.

Instead of crawling these large websites, we queried them real-time. We built a Java program with a Tomcat wrapper that queried the provider websites and determined the most cost-effective selection of book offers on each website. This program used recursive backtracking on the offers for each book in order to determine the proper discount for shipping (A user gets a discount if two books come from the same seller).

Another option was to have an approximate shipping price as opposed to a correct shipping price. This would change recursive backtracking to simple addition. It was decided that having an exact price was very important to the user and that the main performance limiter on the program was querying the external websites. Thus, we chose to determine the correct shipping price. The Java program then built a webpage of results, containing a table of providers, price with shipping, and price without shipping. (This information was included for comparison in case the user wanted to expedite shipping.)

**Experiments**

We recruited five people to test our site’s usability by attempting to purchase their books for winter quarter using our site two times: before we had completely finished the UI, and after we had completely finished the UI. The information of our test subjects is listed below.

|  |  |  |
| --- | --- | --- |
| **Name** | **Age** | **Course of Study** |
| Elise Santa Maria | 20 | MS E |
| Kort Reinecke | 20 | ME |
| Erica Ellingson | 20 | CHEM E |
| Katrina Krebs | 20 | AA |
| Paul Teft-Meeker | 19 | ECON |

*Before completed UI feedback and analysis summary:*

* The color scheme looks terrible.
* The “Choosing classes” page is very intuitive.
* The “Proceeding to cart” page needs instructions.
* Several classes were missing.
* The Half.com redirect page sometimes didn’t add all of the books.
* Using the site, subjects found books faster than their traditional methods, and still much cheaper than the UW bookstore.
* There is no way to verify that the books were correct. A picture of each book as long as its title should be displayed at some point during the process.

*Addressing the problems:*

After receiving this preliminary data, we reworked the UI, vastly improving the layout and color scheme. We also added concise instructions at the top of the page. The missing classes were still missing for the second testing phase because the fix required re-crawling UW class schedule and UW Bookstore in order to update the outdated information. The Half.com redirect problem is an error in the open API. If a book becomes out of stock, it takes a bit for the website to recognize that the book is out of stock. This problem was verified by manually attempting to purchase the same book, using a standard search. As for the “way to verify books”, it is a feature that we had hoped to add at the beginning of the project, but did not have time to implement come the end of the quarter. It remained unimplemented.

*After completed UI feedback and analysis summary:*

* The color scheme flows very well. It is simple and elegant.
* The instruction greatly improved understanding of how to use the site.
* Several classes were missing.
* The Half.com redirect continued to have some errors.
* There is no way to verify that the books were correct.
* The site streamlined the book-buying process.

After addressing the original UI failures that could be remedied given the time constraints, the subjects were pleased with the improvements.

**Conclusion and Future Work**

Overall, this project was a success. We completed most of the features that we had originally intended to complete, and strengthened our expertise in frontend and backend technology. Much of the new found knowledge came from responding to unfortunate surprises that arose. Here is a summary of our learning and the reasons behind it:

* **How to write a crawler.** We learned how to write a crawler in order to make our own version of the UW database. It involved having to read through HTML files and write regular expressions in order to grab the desired data.
* **How cookies can and cannot be manipulated.** While building the crawler, we kept running into either timeout errors or incorrect browser errors. As a result, when we queried the UW Bookstore, we had to continually set the cookie to null in order to receive a new cookie, and then modify the cookie we received to trick the website into thinking we were not a Java program, but a browser instead. We were not as successful at cookie manipulation when it came to Half.com. Half.com does not have an open API for shopping carts. Consequently, we had to find an alternative method of creating the cart. Because websites cannot edit cookies from other websites, we were forced to use a combination of JavaScript and iframes in order to simulate the cart building process. This will be explained further in the “JavaScript and Ajax” section.
* **How to emulate a browser with post requests.** In order to build a crawler and to create clickable links for users to generate shopping carts from different providers, we had to learn how to use post requests to emulate a browser.
* **How to use open APIs.** Open APIs are not as intuitive as one might think. On several occasions, it took three times the predicted length of time in order to decipher the open APIs. In the end, though, the general format of open APIs was understood.
* **JavaScript and Ajax.** JavaScript and Ajax were used to fill in the dropdown menus on the homepage by querying the database. JavaScript was originally used to build the Half.com shopping cart because the site did not have an open API for it. This feature has changed since 12/15/2010, for reasons described above and now uses PHP to generate links that allows the user to construct his/her shopping cart manually.
* **PHP.** In order to create the dynamic content of our pages, we had to utilize PHP.
* **Java does not work on web servers, but Tomcat does.** Near the end of the quarter, when we were putting all of the pieces together, we learned that Java does not work on web servers. As a result, we had to build a Tomcat wrapper for the Java program and install Tomcat onto the server at the last minute.
* **Having a well written plan is crucial.** After analyzing our problems and solutions, we realized that most of them could have been avoided if we had just sat down and decided on the technologies that we were going to use beforehand. If we had done this, we would have not been rushed at the end, trying to put all the pieces of the puzzle together. If we had done this, we would have known that Java does not work on web servers and either would have not written the backend in Java, or would have already been ready for converting it to Tomcat. Either way, most of our problems were a result of just diving into project blind.

Resultantly, if we had to restart the project from scratch, having a well-written plan would have been our primary change. This would have allowed our project to flow much smoother and would have probably allotted enough time for us to build the intermediate page feature, described previously.

In terms of future work, we want to add a few additional features, which would add a fair amount of work. The most significant change is adding an intermediate page with the ability to add and remove books on the fly. This way the user could verify and deselect problematic books, as described previously. A smaller secondary change would be adding a “receive by date” field onto the home page. This would be used to determine if the user desired expedited shipping or standard shipping, and the backend calculations would follow.

Whether or not we end up finding time to work on these changes, we all learned valuable lessons in team project management, team communication, and various web technologies.

**Appendices**

*Work Distribution*

**Mark**

Mark performed most of the initial vision and design. He discovered a source for the book data, and implemented a procedure to download the data. He registered a domain, and setup MySQL and phpMyAdmin on Dante to store the data, and then moved it to cubist server when we discovered that would work better. He wrote some sample queries to discover interesting statistics on UW books, such as the expense for the average class, the most expensive class, the cheapest departmental average for books, etc., hoping to display an interesting statistic whenever a user waited for a query.

**Nick**

Nick did a major visual redesign of the website, greatly improving the ease of use, instruction quality, and eye appeal of the website. He wrote the backend that queried Amazon, Half.com, and eBay. He wrote the code that created carts for the websites that have carts, in order to return the user a cart preloaded with books. He also calculated the shipping prices for each bookstore's cart, and developed an algorithm to find the cheapest combination of books plus shipping options, using discounts from buying multiple books.

**Ken**

Ken made an initial website, wrote the code that queried our database to determine the ISBNs required, and converted Nick's Java to a servlet that ran on Tomcat. He re-hosted the services when we moved them from Dante to cubist.

**Group Dynamics**

The work had a somewhat natural flow, where Mark had to create a database of books, then Nick could query website using those books, and then Ken could host the code on the website. However, we discovered that many vendors require a somewhat established website to allow entrance to their affiliate programs. Mark created an under construction webpage that worked for Amazon.com, but it did not work for abebooks.com. Ken got a late start and was not able to bring up a functioning website in time for us to add abebooks.com to our queried vendors. Nick redesigned the website with some help from Mark at the last minute when Ken had not improved the website design in time for testing.

While Mark and Nick both were continually active in the development process, Ken was difficult to contact. Mark and Nick met regularly to discuss how the program was progressing, while Ken procrastinated until much later to begin his portion.

*External Code*

For the crawler, Mark used only a basic Java version of wget as an example to start from. Nick used the Java APIs provided by Amazon. Both of us had to make modifications for the APIs to fit our needs. Nick also used a small tidbit of code from a blog to help with the JavaScript-iframe redirect. (This function was disabled by Half.com, as described previously.) Ken used the open source project Tomcat to host our Java application.

*Read Me*

1. Open Firefox or Google chrome. Make sure pop-ups are allowed if you plan on using the Half.com shopping cart option.
2. Go to [www.oneclicktextbooks.com](http://www.oneclicktextbooks.com) or <http://cubist.cs.washington.edu/projects/10au/cse454/e/>
3. Follow the instructions provided by the website.