Web Search Engine Project Write-Up

Web Search Engine Project Write-Up:

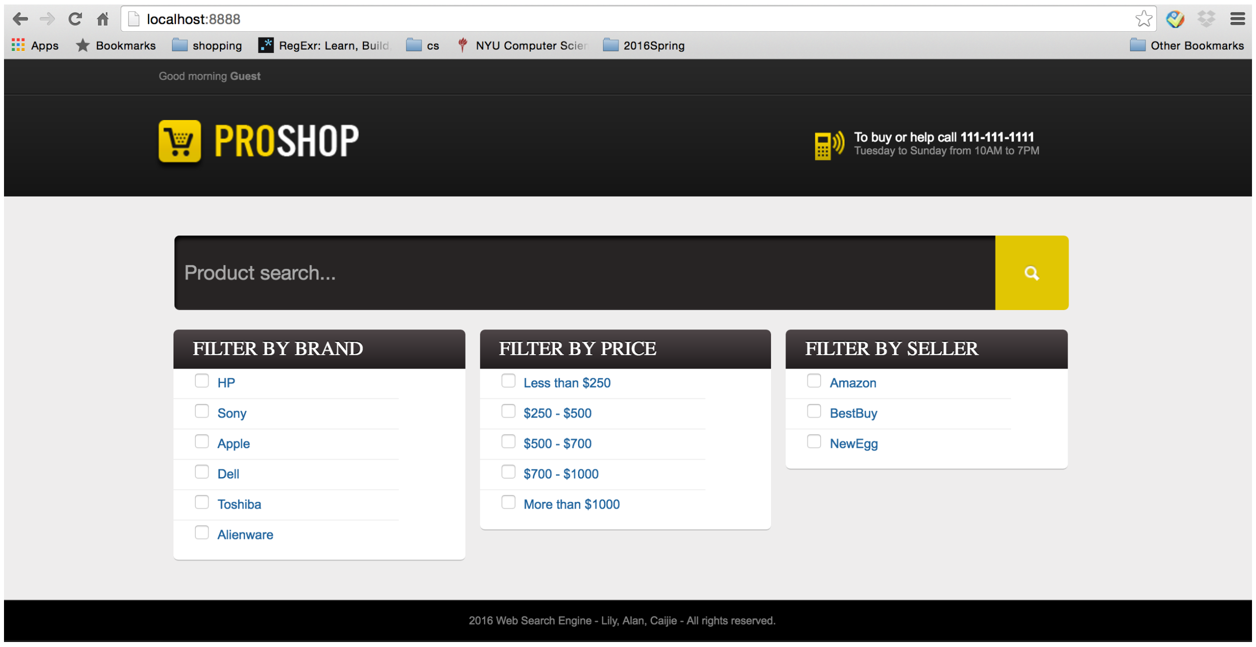
Group members: Alan Yang (asy233), Ying Zhang (zy674), Caijie Zhao (cz1107).

Objective of the project: The objective of our project is to build a search engine in which we can query an electronic hard-good product and be able to filter our search based on certain criteria such as: price range for the product, certain popular manufacturers, and which web source the product is on.

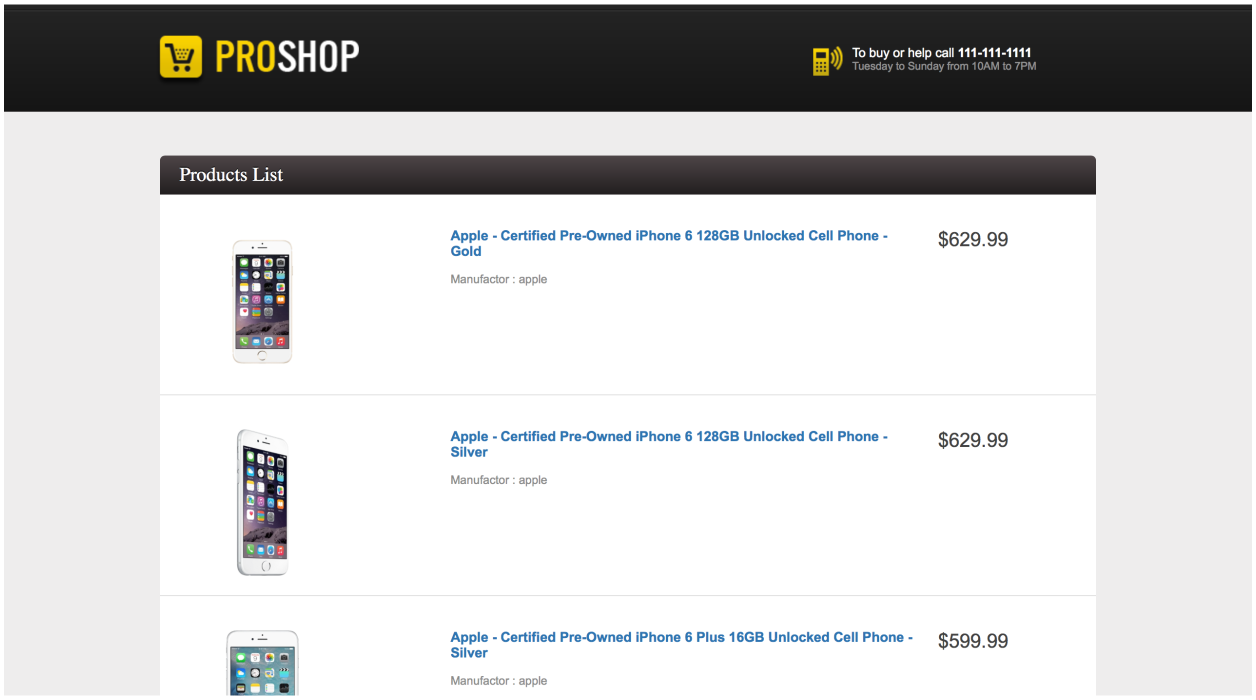
Architecture: For the data, we pre-crawled and retrieved multiple electronic shopping sites such as amazon.com and bestbuy.com and newegg.com as JSON data files first, so that we will not have to crawl the data during the query which can potentially be very slow. For crawling the NewEgg data we used the Scrapy framework. For amazon data we had a script to query the api and get data one at a time. For BestBuy we did a batch download with filters to get only the products that are still being sold and only for hard good products (no software and no movie, music or games). The JSON data files for each web source are very different. The only product information we are using are: name, url, price, image url (indexer indexes the best sized image only), manufacturer/brand, category of the product, features and long and short descriptions.

For the front-end of the project, we will need to build several web pages to show the search engine and results. We used PHP, JavaScript, CSS, HTML, and jQuery to build the front end of our project. To run our project, we just need a local machine to be able to support and run PHP.

The design for the front end will contain a search bar, and 3 filtering sections in which the user can choose which parameters they wish to filter buy. They can check the specific boxes they want to filter by and our retriever will filter the results and return only what you want. For example, if the user were to check the "Apple" box under "Filter by Brand", "$500-$700" under "Filter by Price" and "BestBuy" under "Filter by Seller", your results will only be Apple or Apple related brands in the $500-$700 price range, and only from the BestBuy web source. This filtering of data is done when retrieving the data from the index.



After the search has been made, the user will be redirected to a results page where they will be presented the links with the highest score based on our scoring algorithm which calls our retriever for each query word put into the search bar.


PROSHOP

To buy or help call 111-111-1111
Tuesday to Sunday from 10AM to 7PM

Products List

Apple - Certified Pre-Owned iPhone 6 128GB Unlocked Cell Phone - Gold
Manufactor : apple
$629.99

Apple - Certified Pre-Owned iPhone 6 128GB Unlocked Cell Phone - Silver
Manufactor : apple
$629.99

Apple - Certified Pre-Owned iPhone 6 Plus 16GB Unlocked Cell Phone - Silver
Manufactor : apple
$599.99

For the back-end of the project, we used Lucene libraries, JSON simple libraries, and Java to build the indexer and retriever. We used JSON simple java toolkit to decode and parse our JSON data files into a JSON object array. Each JSON object is the broken down into their specific fields with the fields' types where more data manipulation is made. This data manipulation is necessary because our three data sources are very different from each other. After the very specific data manipulation process, we add the fields and data as TextFields to our Document file for that JSON object. These documents are then written as an index by Lucene IndexWriter.

The Retriever uses Lucene's MultiFieldQueryParser, Analyzer, and TopDocs to query the index for whatever we input into the parameters. We use MultiFieldQueryParser so that we are not only reading the name of the products to retrieve documents. We use name, features (descriptions for BestBuy), url, price, manufacturer, and the web source of the specific document to first query and then filter the results.

For scoring our results, since our data is in JSON file, we could not use the PageRank algorithm which works fine for html file. We thus build an extra layer on top of the Retriever, connecting the Retriever with the queries passed from the front end. This layer gets the query from the front end, processes the query and calls the Retriever. In the end, it passes the results to the front end to display. For each query word, we run the Retriever, combine the results from different word in the query and put them into a set. We calculate the score based on the appearance of each query word and its weight. We assume the first word in the query is the most important one and then the second one, in descending order. This makes sense for most shopping search. Users tend to put the criteria they care most in the first word. Our weight is based on this. The first word gets most weight, and for each word after, -1 for the weight. And the weight for the last one is 1. Then we set an array for each result. The array length equals to the length of the query. For each query word, if it is the result, the corresponding entry get 1, otherwise it is 0. At the end, we use the result array dot the weight array to get the score. And output the result based on the score. Our backend design also supports to score the result based on the price (from low to high), although it is not supported by the front end code.

External software: We will use Lucene, JSON simple and Scrapy frameworks to build the search engine and crawl for the data respectively. jQuery is used in the front end.

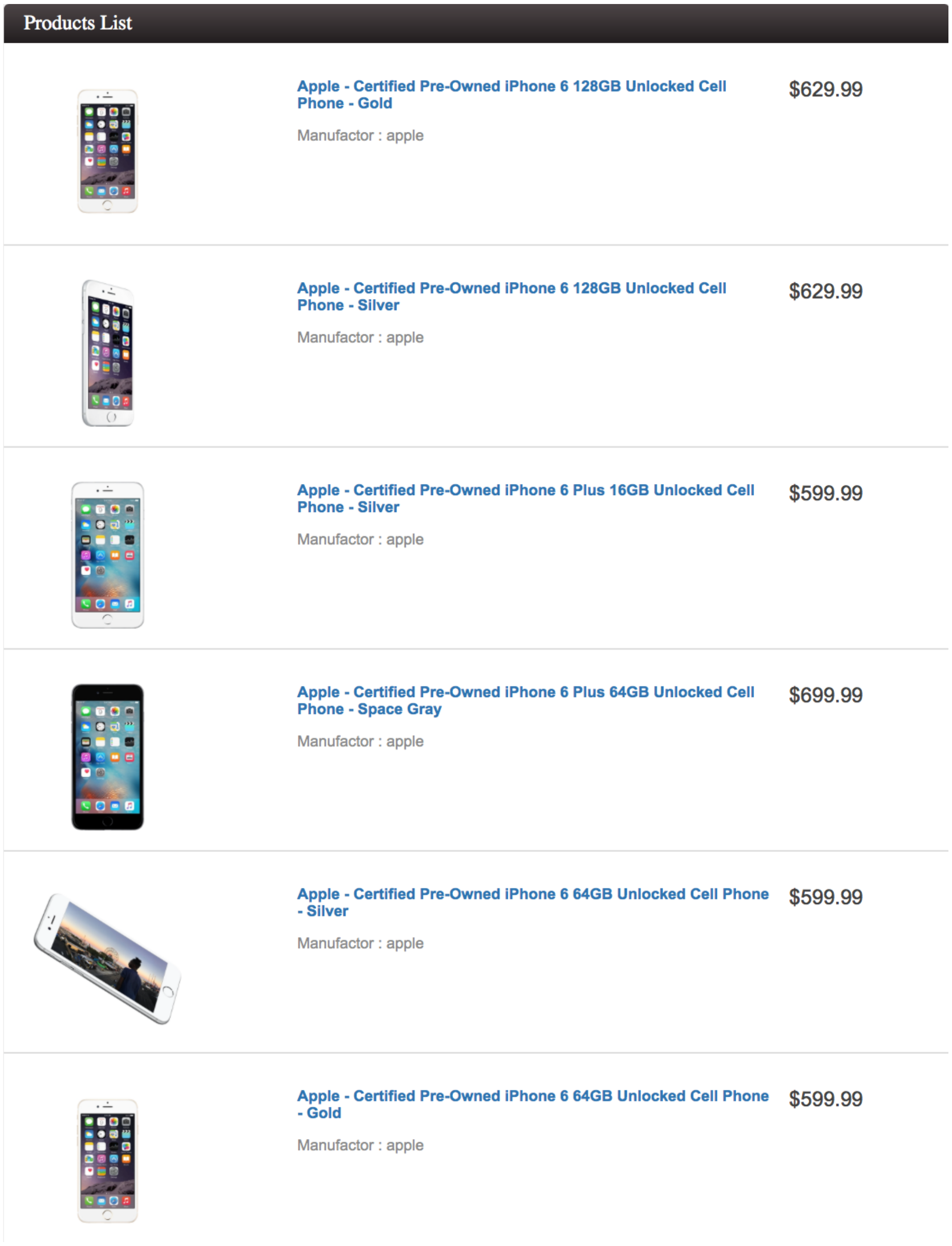
Web resources: **www.amazon.com**, **www.bestbuy.com** and **www.newegg.com** are the three sites we will focus on first in the test phase of the project. We will crawl for the data from **www.newegg.com** using the Scrapy framework and python. While the Amazon and BestBuy data we will get by using their respective APIs.

System feature - programming languages: Java, PHP, Python, JavaScript, CSS, and HTML.

After Project Deliverables:

Examples:

Query: “Phone”
Filters: “Apple” box under “Filter by Brand”, “$500-$700” under “Filter by Price” and “BestBuy” under “Filter by Seller”
Returns:



Products List

Apple - Certified Pre-Owned iPhone 6 128GB Unlocked Cell Phone - Gold
$629.99
Manufactor : apple

Apple - Certified Pre-Owned iPhone 6 128GB Unlocked Cell Phone - Silver
$629.99
Manufactor : apple

Apple - Certified Pre-Owned iPhone 6 Plus 16GB Unlocked Cell Phone - Silver
$599.99
Manufactor : apple

Apple - Certified Pre-Owned iPhone 6 Plus 64GB Unlocked Cell Phone - Space Gray
$699.99
Manufactor : apple

Apple - Certified Pre-Owned iPhone 6 64GB Unlocked Cell Phone - Silver
$599.99
Manufactor : apple

Apple - Certified Pre-Owned iPhone 6 64GB Unlocked Cell Phone - Gold
$599.99
Manufactor : apple

Apple - Certified Pre-Owned iPhone 6 Plus 16GB Unlocked Cell Phone - Space Gray

Manufactor : apple

$599.99

Apple - Certified Pre-Owned iPhone 6 128GB Unlocked Cell Phone - Space Gray

Manufactor : apple

$629.99

Apple - Certified Pre-Owned iPhone 6 16GB Unlocked Cell Phone - Space Gray

Manufactor : apple

$549.99

Apple - Certified Pre-Owned iPhone 6 Plus 16GB Unlocked Cell Phone - Gold

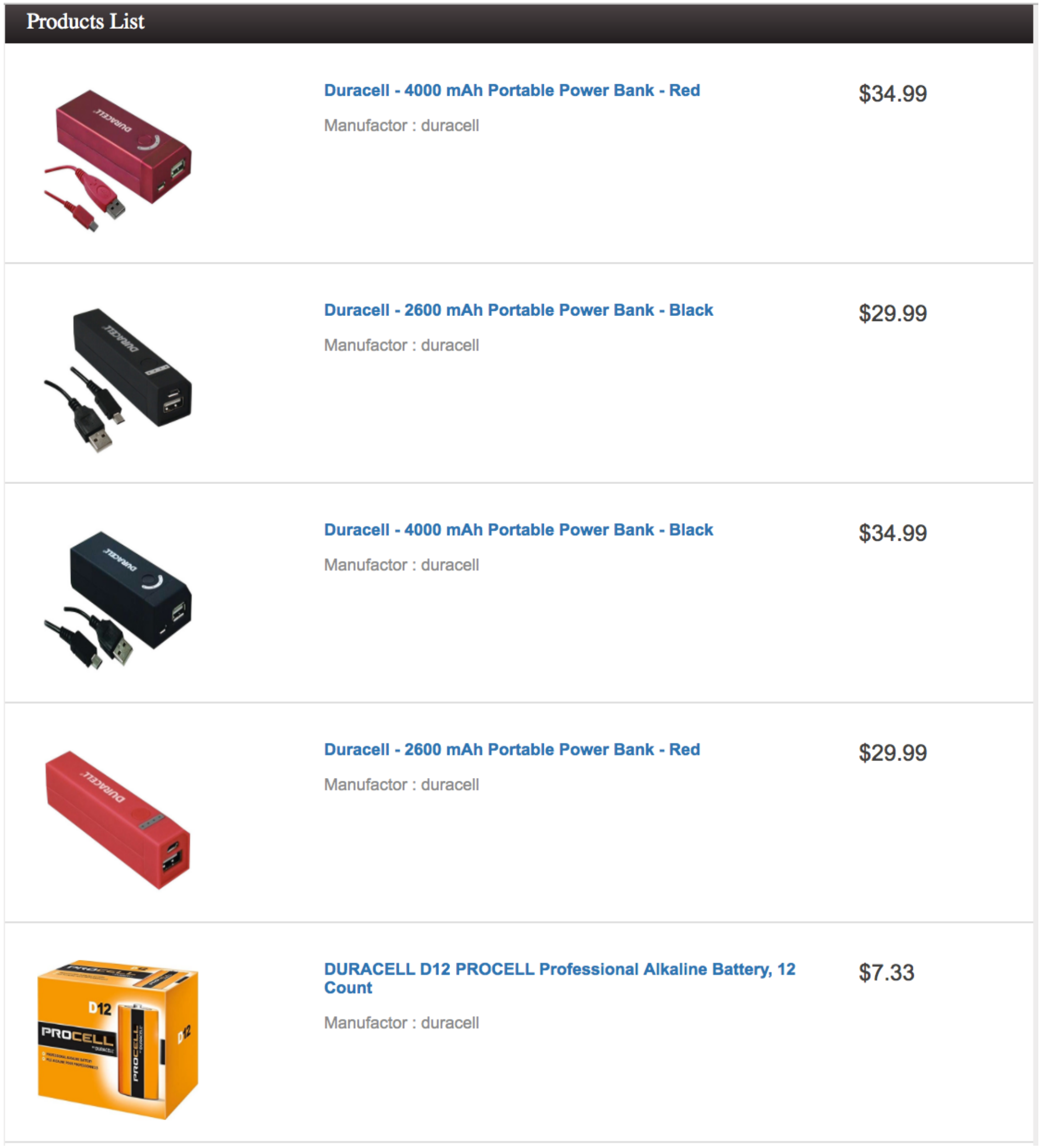
Manufactor : apple

$599.99

Page 1 of 2

1 2

Query: “Duracell Battery”
Filters: none
Returns:



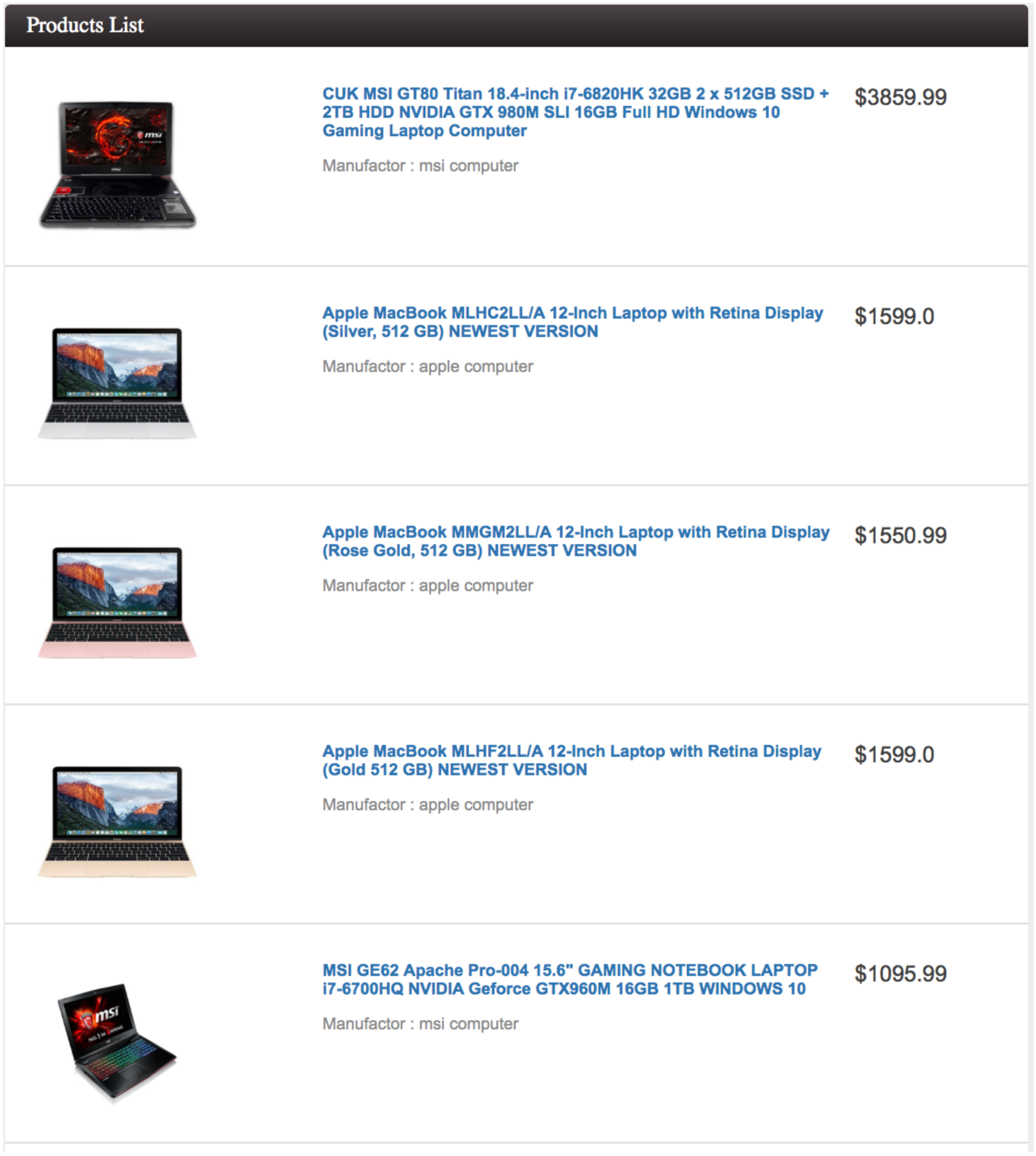
Products List

	Duracell - 4000 mAh Portable Power Bank - Red Manufactor : duracell	$34.99
	Duracell - 2600 mAh Portable Power Bank - Black Manufactor : duracell	$29.99
	Duracell - 4000 mAh Portable Power Bank - Black Manufactor : duracell	$34.99
	Duracell - 2600 mAh Portable Power Bank - Red Manufactor : duracell	$29.99
	DURACELL D12 PROCELL Professional Alkaline Battery, 12 Count Manufactor : duracell	$7.33

	DURACELL D12 PROCELL Professional Alkaline Battery, 12 Count Manufactor : duracell	$7.44
	DURACELL D12 PROCELL Professional Alkaline Battery, 12 Count Manufactor : duracell	$7.33
	Duracell - Quantum AA Batteries (4-Pack) - Red Manufactor : duracell	$6.99
	Duracell - Quantum AA Batteries (8-Pack) - Red Manufactor : duracell	$8.99
	Duracell - NiMH AA/AAA Battery Charger Manufactor : duracell	$23.99

Page 1 of 620

Query: “computer”
Filters: “More than $1000” under “Filter by Price”
Returns:



Products List

CUK MSI GT80 Titan 18.4-inch i7-6820HK 32GB 2 x 512GB SSD + 2TB HDD NVIDIA GTX 980M SLI 16GB Full HD Windows 10 Gaming Laptop Computer $3859.99

Manufactor : msi computer

Apple MacBook MLHC2LL/A 12-Inch Laptop with Retina Display (Silver, 512 GB) NEWEST VERSION $1599.0

Manufactor : apple computer

Apple MacBook MMGM2LL/A 12-Inch Laptop with Retina Display (Rose Gold, 512 GB) NEWEST VERSION $1550.99

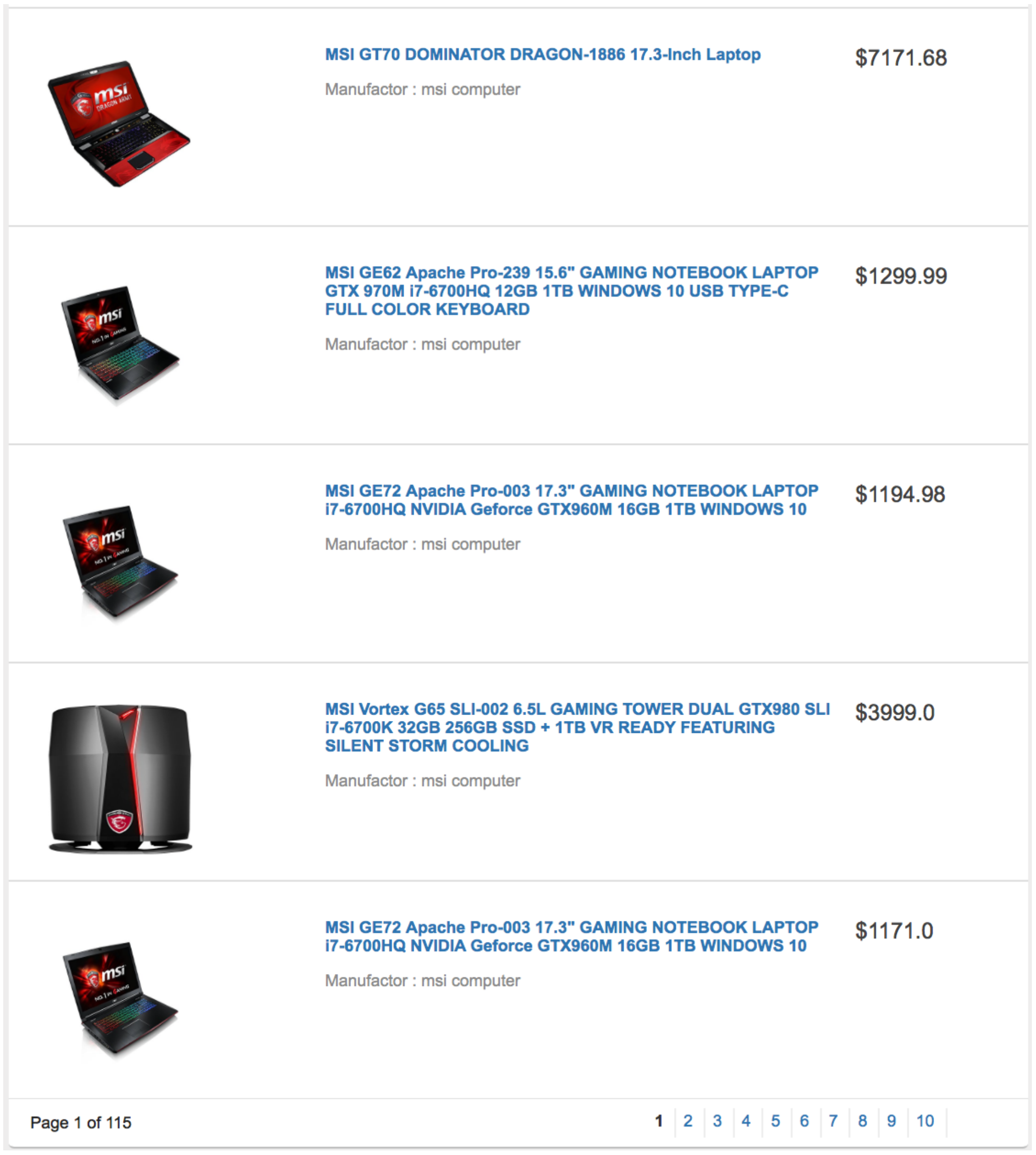
Manufactor : apple computer

Apple MacBook MLHF2LL/A 12-Inch Laptop with Retina Display (Gold 512 GB) NEWEST VERSION $1599.0

Manufactor : apple computer

MSI GE62 Apache Pro-004 15.6" GAMING NOTEBOOK LAPTOP i7-6700HQ NVIDIA Geforce GTX960M 16GB 1TB WINDOWS 10 $1095.99

Manufactor : msi computer



MSI GT70 DOMINATOR DRAGON-1886 17.3-Inch Laptop

Manufactor : msi computer

$7171.68

MSI GE62 Apache Pro-239 15.6" GAMING NOTEBOOK LAPTOP GTX 970M i7-6700HQ 12GB 1TB WINDOWS 10 USB TYPE-C FULL COLOR KEYBOARD

Manufactor : msi computer

$1299.99

MSI GE72 Apache Pro-003 17.3" GAMING NOTEBOOK LAPTOP i7-6700HQ NVIDIA Geforce GTX960M 16GB 1TB WINDOWS 10

Manufactor : msi computer

$1194.98

MSI Vortex G65 SLI-002 6.5L GAMING TOWER DUAL GTX980 SLI i7-6700K 32GB 256GB SSD + 1TB VR READY FEATURING SILENT STORM COOLING

Manufactor : msi computer

$3999.0

MSI GE72 Apache Pro-003 17.3" GAMING NOTEBOOK LAPTOP i7-6700HQ NVIDIA Geforce GTX960M 16GB 1TB WINDOWS 10

Manufactor : msi computer

$1171.0

Page 1 of 115

1 2 3 4 5 6 7 8 9 10

Interesting/Unexpected issues: We originally wanted to include ratings and released date for the products as another part of the search criteria, however, we faced problems with retrieving and crawling rating and released date data for amazon and NewEgg. Lots of the crawled or retrieved data is very disorganized (especially Amazon data) and so there are missing fields or fields put in different places. For example some of the Amazon data has 3 different prices: LowestNewPrice, ListPrice, and LowestUsedPrice. Most of the websites follow LowestNewPrice, but occasionally we get returns as “Too low to display” for this which is not what we want. So in the indexer we check if LowestNewPrice is not there or if it is Too low to display, if so we take the ListPrice (which might be $0.00), and then as a last resort we take the LowestUsed-

Price. Crawling for data also gives some problems; we had to redo the process many times because either the website starts asking if we are robots or we start getting blanks for fields that we used to get values for. We planned to cluster the results into categories but later found that was not necessary. The difference between products was not significant. Most of them have similar features. The difference is brand and price. So we modified our idea to set filter by brand and price.

Systematic evaluation of the project: We think that the outputs for most of our queries are quite reasonable. Our scoring, indexer and retriever were able to give pretty reasonable results based on our data. The more words in the query system (the more specific the query is), the better results our system should give. For a query with one or two words, the weight system does not affect much and the results retain the order from the Retriever mostly.

How to improve our system next time: We can potentially get more data sources such as **thinkgreek.com**. We can also potentially give the user the ability to weight their keywords based on how important they are while having an even split on default. If given the resources, getting better data from these companies/websites will also make our search engine better. If we are given more query history from the user (like what Google did in their system), we can run some machine learning algorithm to design the weight system more wisely based on user preference.

The code can be found in: https://github.com/lily-zhangying/web_search_engine
For the Crawler code, there are 20 different categories and thus 20 crawler projects. They all look similar and we just put 2 of them as examples. If you would like to see all the code, please let us know and we will submit all of them.
For web deployment of the project, we could not find a server that gives us high authorization to run the php code of our project. However, our project can be deployed in the local host with the Linux computer @ Courant Computer Lab. Please refer to README to set up the system.

Citations:

Fang, Yidong. "Json-simple." JSON-Simple. Web. 01 May 2016. <https://github.com/fangyidong/json-simple>.

Fang, Yidong. "Json-simple." Google Archive. Web. 01 May 2016. <https://code.google.com/archive/p/json-simple/>.

"Welcome to Apache Lucene." Apache Lucene. Apache Lucene. Web. 01 May 2016. <https://lucene.apache.org>

"Scrapy | A Fast and Powerful Scraping and Web Crawling Framework." Scrapy | A Fast and Powerful Scraping and Web Crawling Framework. Web. 01 May 2016. <http://scrapy.org/>.

“Bulk Download API.” Bestbuy. <https://developer.bestbuy.com/documentation/bulkDownload-api>

We would also like to thanks Professor Suzanne McIntosh for her advice on getting access to BestBuy data through their API. She recommended we use an ACM email account to acquire the BestBuy API key. We would also like to thank BestBuy, Amazon, and NewEgg for the data we were able to retrieve for the purpose of our project.